

Generative AI Applications

Planning, Design and Implementation

David Spuler and Michael Sharpe

Aussie AI Labs

Generative AI Applications

Planning, Design and Implementation

By David Spuler and Michael Sharpe.

Copyright © David Spuler, 2024. All rights reserved.

Copyright © Aussie AI Labs Pty Ltd, 2024. All rights reserved.

Published by Aussie AI Labs Pty Ltd, Adelaide, Australia.

<https://www.aussieai.com>

Cover design: Google ImageFX.

This book is copyright. Subject to statutory exceptions and to the provisions of any separate licensing agreements, no reproduction of any part of this book is allowed without prior written permission from the publisher.

All registered or unregistered trademarks mentioned in this book are owned by their respective rightsholders.

Neither author nor publisher guarantee the persistence or accuracy of URLs for external or third-party internet websites referred to in this book, and do not guarantee that any content on such websites is, or will remain, accurate or appropriate.

Table of Contents

Preface	13
About the Authors	17
About the Contributor	18
Part I: Introduction to Enterprise AI	19
1. Strategy	21
The Aussie Method	21
Be the Best.....	22
Build the Best.....	23
Harsh Truths.....	24
Strategic Steps	25
References.....	26
2. Generative AI Market Overview	27
The State of AI.....	27
AI Market Trends.....	28
Bull Case for AI	30
Bear Case for AI.....	32
Hype Versus Adoption.....	34
Killer Apps.....	34
References.....	37

3. LLM Technology Overview	39
What is AI?.....	39
AI Technology Trends.....	40
Why is AI Slow?	42
Bigger and Smarter AI	43
Faster AI	45
AGI.....	46
References	47
Part II: Selecting AI Projects.....	49
4. Deciding.....	51
Decisions, Decisions.....	51
Project Priorities and Goals	52
Choosing Your AI Project	53
What are Other Companies Doing?	54
Confusion Reigns	55
The Easier Options	56
Internal Business AI Platforms	56
AI Core Competency Levels	57
References	59
5. Use Cases	61
What are Use Cases?.....	61
Gen AI vs ML Use Cases.....	61
Longstanding AI Use Cases	63
Business Use Cases	63
Consumer Uses of Gen AI	64
Software Engineering Use Cases.....	65
Professional Use Cases	66

6. Limitations	69
Generative AI Limitations	69
Accuracy Limitations	69
Alignment Limitations	70
Training Data Quality	71
Computational Limitations	72
Reasoning Limitations	73
Mathematical Reasoning	76
References	77
Part III: Planning AI Projects	79
7. Planning	81
Planning an AI Project	81
Go Slow to Go Fast	81
Expediting AI Projects	82
Approvals	82
Auditing Existing AI Projects	83
Consulting Advice	83
General Legal Issues	84
Buy Projects	86
Build Projects	86
Data	88
AI-Free Zone	89
References	90

8. Data	91
AI Needs Data	91
Data Inventory	91
Data Formats	92
Multimodal Data	92
What is Good Data?	93
Data Cleaning	93
Open Source Data	94
Legal Issues with Data	95
Dataless Projects	97
References	98
9. Budgeting and ROI	99
Budget Allocation	99
AI Budget Items	100
ROI	101
Staff Skills and Salaries	103
Financial Optimizations	104
Technical Debt in AI Projects	105
References	105
10. Safety	107
Failure Stories for Generative AI	107
Consequences of AI Failures	108
Data Causes of AI Failures	109
Types of AI Safety Issues	109
Third-Party AI Vendor Safety Issues	110
Jailbreaks	112
Risk Mitigations	113
Refusal Modules and Prompt Shields	114

Part IV: Design	117
11. Requirements	119
AI Project Requirements	119
Top 10 Really Big Optimizations.....	120
Build versus Buy.....	121
Foundation Model Choices	122
Open Source Models.....	123
Commercial-Usage Open Source Models	123
Model Size.....	124
Latency and Response Time.....	125
How Fast is Needed?.....	126
Incremental Output.....	127
Accuracy Requirements	128
References.....	129
12. Architectures	131
Easier Architectures	131
Components of AI Architectures	132
Software Architecture.....	133
AI Tech Stack	134
Chains and Toolchains	135
Wrap Architectures	136
On-Premises LLM Architecture	137
Private Cloud LLM Architecture	138
Two-Step Architectures.....	139
Security Credential Management.....	139
On-Device AI Architectures.....	140
References.....	141

13. Transformers & LLMs	143
AI Engines & Models	143
Other Types of LLMs.....	147
Training and Fine-tuning	148
Inference	150
Context and Conversations.....	151
Extended Transformers	152
Other Types of Neural Networks	153
References	154
14. Training and Fine-Tuning.....	157
Training Options	157
Training a Foundation Model	157
Fine-Tuning.....	158
Fine-Tuning Algorithm.....	159
What is LoRA?	161
Multi-LoRA	163
Training FAQs	164
References	165
15. RAG Architectures	167
What is RAG?.....	167
RAG Project Design.....	169
RAG Detailed Algorithm.....	170
Fine-Tuning vs RAG	174
Prompt Engineering and RAG	179
Hybrid RAG + Fine-tuning Methods	181
Use Cases for FT vs RAG	182
Refreshing an Old RAG Application	184
Advanced RAG Architectures	186

Part V: Implementation	189
16. Building.....	191
AI Application Development	191
AI Application Platforms	192
Vendor Lock-in vs Open Source Platforms	196
References.....	197
17. Prompt Engineering.....	199
What is Prompt Engineering?.....	199
Basics of Prompt Engineering.....	200
Chatbot Sessions and Conversations	201
Meta-Instructions	202
Brand Voice and Prompt Engineering.....	203
Prepended Prompt Context.....	204
Mini-RAG	205
Repetition in Prompts	205
Efficiency of Prepended Prompt Text	206
Reasoning.....	207
Chain-of-Thought	207
Emotional Prompting.....	208
Skeleton-of-Thought Prompting	208
Two-Step Reasoning	209
Multi-Step Reasoning.....	210
Why is Prompt Engineering So Weird?.....	210
References.....	211

18. Deployment Architecture.....	213
Backend Server Architecture.....	213
AI Server Hosting Options	214
GPU Specs	217
Online Architecture Optimization	219
API Wrapper Architecture Optimizations.....	220
Request Queue Architecture	220
Load Balancing.....	221
Networking Optimizations	222
Prompt History and Context	224
Part VI: Advanced Topics	227
19. AI Phones and PCs	229
AI Phones	229
AI PCs.....	229
Use Cases	230
Agents on Devices.....	232
Advanced Multi-Step Use Cases.....	233
AI Phone Apps.....	233
Obstacles to Smartphone AI.....	234
Speeding Up Smartphone AI	236
References	239
20. Tool Usage.....	241
Tool Usage in Generative AI	241
Types of Tools.....	242
Tool Architectures	243
How are Tools Integrated?	243
Tool Usage in RAG Architectures	244
LLM Computer Usage.....	245

21. Agentic Architectures.....	247
Single Agent Architectures	247
Types of Agents	247
Report Agents	249
Action Agents	250
Agentic Architectures	250
Security of Agents	252
References.....	253
22. AI Research Overview	255
The Three S's of AI Research.....	255
Smartness Research	255
References on Reasoning.....	257
Safety Research	258
Speed Research	259
SOTA Speed Research	260
Commercial AI Platform Speedups	261
Model Compression	263
Kernel Optimizations	265
AI Accelerators	267
References on Inference Optimization	269
Green AI.....	270

Preface

*“Like most extraordinary innovations,
we underestimate how absolutely massive
the transformation is going to be,
and we probably overestimate
the pace at which it is going to happen.”*

— Emma Walmsley, CEO of GSK, 2024.

Welcome to *Generative AI Applications* and the wonderful world of AI engines and LLMs. We hope this book leaves you with many happy memories!

Who This Book is For

This book is for anyone whose boss has just thrown them into the deep end of an ocean rip by assigning them the task of doing “something with LLMs” or any other similarly clear project description related to generative AI. Alternatively, this book may help you if you’re the CEO or other C-suite executive of a company, and you want to assign someone such a project.

How This Book is Organized

In consideration of common book style for prefaces that nobody reads, this is a discussion of how we have organized the book. The book opens delightfully with the imaginatively-named “Part 1: Introduction” which contains, as you may surmise, an introduction to the topic. Similarly, when you see “Chapter 8: Data” in the contents, you would be correct in assuming that there’s a fair amount of discussion about data therein. The only exception to the self-explanatory naming of the chapter titles is “Chapter 17: Prompt Engineering,” which is misleading because there’s nothing at all that should be called “engineering” in that chapter.

About Aussie AI

Aussie AI is an Australian AI lab that commercializes research in on-device inference and consumer AI applications, with a special focus on inference optimization algorithms that make the engine run fast. Some such areas are examined in our other published book title *Generative AI in C++: Coding Transformers and LLMs*.

In terms of public consumer-facing products, Aussie AI has a suite of AI-based writing and editing tools, with a special focus on fiction writing. Our software already has an extensive range of reports and error checks for both fiction and non-fiction writing, from a full-length novel to a short report. Please try it out and let us know what you think: <https://www.aussieai.com>

Our AI Research

The primary focus of research at Aussie AI is on optimizing LLM inference algorithms (i.e., “running” the model after training or fine-tuning), and our research is toward the following aims:

- Fast on-device model inference algorithms, specifically for smartphones and AI PCs.
- Scaling inference algorithms to large volumes of requests.
- Efficient GPU inference algorithms (hardware acceleration).
- Non-GPU inference optimization algorithms (i.e., software methods).

Disclosure: Minimal AI Authorship

No AIs were harmed in making this book!

Despite our involvement in the AI industry, there was almost no AI engine usage in creating this book’s text or any related code. Some text has been analyzed and reviewed using Aussie AI’s editing tools, but not even one paragraph was auto-created by any generative AI engine.

However, AI was used in several ways. AI-assisted search tools, such as “Bing Chat with GPT-4”, were very useful in brainstorming topics and researching some of the technical issues. The main cover art image of a baby dragon was generated using AI, followed by human editing.

Disclaimers

Although we hope the information is useful to you, neither the content nor any related code in this work is guaranteed for any particular purpose. Nothing herein is intended to be personal, medical, financial or legal advice. You should make your own enquiries to confirm the appropriateness to your situation of any information.

Oh, and sometimes I'm being sarcastic, or making a joke, but it's hard to know when, because there's also a saying that "Truth is often said in jest!" Your AI engine certainly won't be able to help you sort out that conundrum.

Third-Party License Notices

Except where expressly noted, all content and code is written by David Spuler (or the contributors), with rights owned by David Spuler and/or Aussie AI.

Additional information, acknowledgments and legal notices in relation to this book, the Aussie AI C++ source code, or other Aussie AI software, can be found on the Aussie AI Legal Notices page: <https://www.aussieai.com/admin/legal-notices>.

Acknowledgements

This book would not have been possible without the help of others. Data scientist and architecture expert Cameron Gregory provided much assistance with many contributions to various chapters on architecture and DevOps.

We would like to acknowledge the many AI researchers and open source contributors who have made the AI revolution possible. In particular, the advanced coding skills shown in the many C++ projects are acknowledged with both admiration and appreciation.

Please Leave a Review

We hope you enjoy the book! Please consider leaving a review on the website where you purchased the book.

Feedback and Contacts

Feedback from readers is welcome. Please feel free to tell us what you think of the book, the literature review, or our Aussie AI software. Contact us by email via research@aussieai.com.

About the Authors

David Spuler is a serial technology entrepreneur who has combined his love for writing with AI technology in his latest venture: Aussie AI is a consumer AI application platform, including suite of tools for writing and editing. His published works include the advanced book on building your own AI engines, *Generative AI in C++: Coding Transformers and LLMs*, two books on CUDA C++ efficiency and debugging, five non-fiction textbooks on C++ programming covering advanced C++ programming, efficiency/optimization, debugging/testing, software development tools, and the Safe C++ standard, and one application management ops book on BMC PATROL.

Other than writing, he's an avid AI researcher with a Ph.D. in Computer Science and decades of professional experience. Most recently, Spuler has been founding startups, including the current Aussie AI startup and multiple high-traffic website platforms with millions of monthly uniques. Prior roles in the corporate world have been as a software industry executive at BMC Software, M&A advisor, strategy consultant, patent expert, and prolific C++ coder with expertise in inference efficiency and kernel optimization, autonomous agents, compiler construction, internationalization, ontologies and general software development tooling.

Michael Sharpe is an experienced technologist with expertise in AI/ML, cybersecurity, cloud architectures, compiler construction, and multiple programming languages. He is currently Senior Software Architect at PROS Inc., where he is a member of the Office of Technology focusing on developing and evangelizing AI. His AI expertise extends to monitoring/observability, devops/MLOps, ITSM, low-resource LLM inference, Retrieval Augmented Generation (RAG) and AI-based agents.

In a long R&D career, Sharpe has been coding C++ for almost 30 years, with prior roles at BMC Software, Attachmate (formerly NetIQ) and IT Involve. Sharpe has a Bachelor of Science with First Class Honors in Computer Science from James Cook University and holds several registered patents.

About the Contributor

Cameron Gregory is a technology entrepreneur including as co-founder of fintech bond trading startup BQuotes (acquired by Moody's), co-founder and Chief Technology Officer (CTO) of Trademark Vision with an AI-based image search product (acquired by Clarivate), and founder of several image creation companies including FlamingText.com, LogoNut, AddText, and Creator.me. Currently a Senior Data Scientist focused on “big data” for hedge funds at fintech startup Advan Research Corporation, he is used to working with real-world data at scale.

Gregory has been making code go fast since the 1990s at AT&T Bell Laboratories in New Jersey, and is proficient in multiple programming languages, notably including C++, Java, and JavaScript. He holds a Bachelor of Science with First Class Honors in Computer Science from James Cook University.

Part I: Introduction to Enterprise AI

“Learning to fly is not pretty but flying is.”

— Satya Nadella, *Hit Refresh*, 2017.

1. Strategy

The Aussie Method

No, it's not a new diet plan, a stock trading algorithm, or something else. Rather, it's how to build an AI project according to Aussie AI's best practices, and it goes like this with a two-step procedure:

1. Be the best, and
2. Have a few laughs along the way.

So, here, I'll start you off:

Q: How many LLMs does it take to change a lightbulb?

A: One (56.48%), ChatGPT (25.71%), Dog (11.87%), What type? (8.39%)

Amusingly, I ran that question through ChatGPT, and got some great lightbulb jokes back. I'm saving them for the next edition, but feel free to ask it yourself.

The Aussie Method rules of AI include:

- All AI project meetings start with a knock-knock joke or alternatively choose: "*An AI walks into a bar...*" (Ouch!).
- If your code crashes the AI engine, you're bringing kolaches to the next meeting.
- We need a leaderboard tracking who's winning the game of count the sunrises.
- Train your AI to give answers speaking like C3PO.

I hereby delegate to you the right to create extra Aussie Method rules for your AI project.

Be the Best

All jokes aside, I really mean what I say in setting the goals of your project. If you're adopting AI in your organization, you should set your goal to nothing less than this: *be the best!*

How?

Well, firstly, I don't think that you'll get there by throwing money at AI vendors. You can get some short-term wins here and there, and drop a few percentage points off your cost base down to the bottom line. Maybe there's a few ways to get some happier customers, and some increased revenue from doing better at sales and marketing. Nothing wrong with that!

But I think that all of these are a tactical gain, but it's a strategic loss if that's all that you do. Such plans are a good first step, and I'm not saying to avoid doing them, but they lack appreciation for what's going to happen over the next few years. A seismic shift is going to happen in terms of both major areas:

1. External — you'll need to engage more deeply with your customers (or lose them), and
2. Internal — address the overall workflow efficiency of your internal operations.

To win in both those spaces, you'll need to be ambitious. Here's how to be the best at AI in your industry or organizational context:

- Grow core competencies in AI subareas
- Build IP
- Improve staff capabilities
- Revamp internal processes
- Address AI governance

In order to do all this, you need to make a plan at a very high level, and then execute it throughout the organization.

Build the Best

To be the best at generative AI, you obviously need to make it a core competency in your organization by creating an AI team. But my point is that you need to be not just using and integrating this technology, but also *building* all of the generative AI applications that you need. There is much to gain from just installing vendor products, or using off-the-shelf models, whether commercial or open-source, but almost every company is working on this, and doing so won't make your company the *best*.

If you're not in the tech business already, your first thought for your business might be that you don't want to build a core competency in building generative AI apps. You can buy that expertise from vendors, instead of belatedly becoming a tech powerhouse yourself. Whether you're an airline or a bank, there are more important things than entering the AI software development business, right?

After all, you didn't really need to become experts in coding applications for cloud architectures or building smartphone apps. There have been a lot of major IT trends before, and you didn't need that capability in-house. Why now?

The answer is: business-level logic. Generative AI is the first technology that operates at a high level of business-specific logic. Cloud backends and smartphone front-ends were new ways to run existing technology, but generative AI changes the whole technology itself.

You're right to be sceptical of this new trend, and also justified in not wanting to become a tech company. It's unlikely to be a good choice to pivot your company to start designing GPUs and compete with NVIDIA. Similarly, I'm not suggesting that you become experts in all of the deep tech aspects of generative AI. Look up to the top, not down the AI stack.

The decision is one of focus. Instead of building a core competency in low-level AI capabilities (e.g., Transformer engines or huge LLMs), look at the business-level capabilities and how they are changing. My suggestion is to consider creating an in-house core competency in building vertical-specific applications in your core business area, no matter what business you're in.

The underpinning reason for this recommendation of creating an AI development competency is to achieve business-aligned growth. The idea mainly applies if your overarching aim is top-line growth from generative AI. You probably won't need a core competency in building generative AI applications for standard backoffice automation.

If your goal with generative AI is mainly cost reduction and internal productivity enhancement in non-strategic areas, it's a different story. There's nothing wrong with seeking a bottom-line benefit, and plenty of opportunities to do so, but you probably won't need to become experts in building generative AI applications to achieve this. The areas of back-office automation and staff productivity already have numerous vendors offering AI solutions for this, at least in the areas that are common to all types of businesses.

Harsh Truths

I'm optimistic about the value of AI in both business and consumer use cases. But there's a lot of hype and hubris, so it's not all strawberries and cream. Here's some negative thoughts to ponder:

- The AI will be wrong at times. Expect it, and learn to live with it. Be very careful about putting AI in places that cannot be overruled or bypassed by humans.
- A large percentage of AI projects are being built with no real goals, just to be seen to be doing something (for the C-suite or the investors).
- LLMs are really just not that smart. They don't generalize well. Advanced multi-step reasoning techniques like "Chain-of-Thought" are in some ways more like workarounds than real intelligence.
- If you've built a customer support chatbot, you're not that special (everyone has). Worse, your RAG-based customer support chatbot is outdated already (and needs more money spent). Customer expectations are increasing, and there are newer advanced RAG methods.
- Don't fall down the rabbit hole and assume AI can do everything; it can't. Just because a few test queries/prompts produce good results does not mean all your use cases will produce good results.
- Data handling, cleanup, and preparation remains one of the top hidden costs of getting your AI projects sorted.
- Don't make the mistake that the next big leap forward is only a few weeks/months away. It's likely far longer than that. Don't bet your business on the expectation of something coming soon.
- Everything your team does today will be outdated within 12 months. Or maybe sooner.
- Building a core competency and an "AI platform" in your business, as we've advocated above, is really, really hard!

We don't have a solution for you about all these problem issues, but you should at least consider them fully. Hopefully, there are some answers in this book.

Strategic Steps

You've probably already done some of these things, since the AI buzz has been going for a while. Nevertheless, here's some thoughts on what to do at the very top level:

- Create a special AI budget allocation
- Create an “AI group”
- Establish internal AI evangelists
- Hire a Chief AI Officer (CAIO)

Here are some of the things your top-level AI team need to consider strategically:

- Opportunities to focus on the top or bottom of the P&L (i.e., customer engagement versus internal productivity).
- Identify existing competitive advantages to extend with AI.
- Consider buy versus build with intentionality in terms of strategy.
- Look beyond buy-vs-build to strategic partnerships, investments, or M&A possibilities.
- Think about who will control the changing customer touchpoints in an AI-first world.
- Ponder the puzzle of where all your business data is located.
- Consider consolidating and reducing the number of places where the data is.
- Understand vendor roadmaps for AI so you don't duplicate efforts already under way.
- Extend security-related and privacy governance to include AI.

And here are some even more specific project steps, some of which are further covered in later chapters:

- Seek consultant advice — not just tactical, but also strategic!
- Inventory all AI IP assets and gaps to fill with future additions.
- Identify staff capabilities and needs for re-training or hiring.
- Review existing data in terms of breadth and depth.
- Plan for future data capture including tech issues and user legal agreements.
- Rush out and fill your shopping cart with a thousand GPUs.

The last one was a joke (or was it?), because I wouldn't want you to forget Step 2 of the Aussie Method: everybody needs a good laugh now and then.

References

1. Nayan Paul, Aug 27, 2024, *Gen AI & LLM Adoption through a Common Platform instead of enabling 'be-spoke' use cases*, <https://medium.com/@nayan.j.paul/gen-ai-building-adoption-through-a-common-platform-instead-of-enabling-be-spoke-use-cases-b0cbc2e185a8>
2. Sangeet Paul Choudary, Nov 7, 2023, *Why your super-app strategy will (most likely) fail: Building competitive advantage in the attention economy*, <https://platforms.substack.com/p/why-your-super-app-strategy-will>
3. Janelle Teng, June 21, 2024, *State of the Cloud 2024, The Legacy Cloud is dead — long live AI Cloud!* <https://nextbigteng.substack.com/p/state-of-the-cloud-2024>
4. Nayan Paul, Aug 27, 2024, *Gen AI & LLM Adoption through a Common Platform instead of enabling 'be-spoke' use cases*, <https://medium.com/@nayan.j.paul/gen-ai-building-adoption-through-a-common-platform-instead-of-enabling-be-spoke-use-cases-b0cbc2e185a8>
5. Simeon Emanuilov, Apr 4, 2024 *LLM agent operating system (AIOS) and the future of LLM-powered agents*, <https://medium.com/@simeon.emanuilov/llm-agent-operating-system-aios-and-the-future-of-llm-powered-agents-3d08b4e91c34> <https://unfoldai.com/aios-llm-powered-agents/>
6. Eddie Forson, Apr 29, 2024, *Why I'm building my own AI Agent library*, https://medium.com/@Ed_Forson/why-im-building-my-own-ai-agent-library-e20ec9aa3647
7. A. R. Ali, K. Kumar, M. A. Siddiqui and M. Zahid, 2024, *An Open-source Cross-Industry and Cloud-agnostic Generative AI Platform*, 2024 International Joint Conference on Neural Networks (IJCNN), Yokohama, Japan, 2024, pp. 1-10, doi: 10.1109/IJCNN60899.2024.10650688, <https://ieeexplore.ieee.org/abstract/document/10650688>
8. Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, Joseph E. Gonzalez, 12 Feb 2024 (v2), *MemGPT: Towards LLMs as Operating Systems*, <https://arxiv.org/abs/2310.08560> <https://memgpt.ai/>
9. Michael A. Cusumano, Vivek F. Farias, and Rama Ramakrishnan, Sep 2024, *Generative AI as a New Platform for Applications Development*, <https://mit-genai.pubpub.org/pub/r8xcl5ol/release/1>
10. Nicholas Grous, Andrew Kim, June 04, 2024, *Generative AI: A New Consumer Operating System*, <https://www.ark-invest.com/articles/analyst-research/generative-ai-a-new-consumer-operating-system>

2. Generative AI Market Overview

The State of AI

There's so much going on in the AI industry that these words are out-of-date the second that I type them. Nevertheless, here are a few general thoughts on where we are:

AI is popular. Even two years after the launch of ChatGPT, generative AI is still the talk of the town (and the boardroom!). With new capabilities launching often, there's much to keep everyone entertained.

AI is amazing. Already, it seems like “just” writing fluent text is old hat. But I'm still astounded by the capabilities of the latest AI apps with vibrant realistic images and alluring video clips. There are so many advances happening quickly in speech, vision, animation, and video. The whole industry is evolving rapidly at such speed that I want an AI copilot to help me keep up with all the news.

AI is improving rapidly. The leaderboards for AI model intelligence continue to post higher scores for new models in terms of base intelligence and complicated reasoning. There's also better handling of negative issues such as safety, toxicity, and bias.

AI is expanding. ChatGPT launched as text only, but now we have image intelligence widely available, and there's starting to be video, too. Multimodal models such as GPT-4o and Google Gemini can both output and understand images and text, together or separately.

AI is expensive. Remember the joke about how “boat” stands for “Bring Out Another Thousand”? That's nothing compared to AI. A single GPU costs more than your boat and a typical motherboard has eight of them. And the big tech companies have been buying these by the thousands.

What should LLM stand for? “Lavish Leviathan Mammoth”? That was mine. “Ludicrously Large Mango” was Bing Chat with GPT-4’s AI suggestion. Neither are great, which is comforting because it means there’s still some work to be done.

AI is not new. The AI-related workload hosting market is many years old. Just because generative AI has blasted into consumer consciousness, and into boardroom discussions as a result, doesn’t mean that AI is new. The cloud hosting companies like Amazon AWS, Microsoft Azure, and Google GCP, have been doing AI workloads for many customers, for many years. Instead of using GPUs for generate AI, they’ve been running workloads in other AI areas like Machine Learning (ML), machine vision (e.g., Tesla autonomous cars), product suggestion feeds, predictive modeling, auto-completion of search queries, and so on. There were already billions of dollars invested in AI long before ChatGPT set the web on fire.

AI Phones. AI is going to be on your phone, and it’s going to be a big driver of new phone purchases. Google has the Android on-device SDK to build phone apps, and Apple has launched the “Apple Intelligence” models for on-device inference on iPhone.

AI PCs. AI models and applications are set to make PCs hot again in the near-term. The next generation of laptops and desktops will run AI models natively, and there will also be hybrid architectures with AI workloads offloaded into the cloud. Microsoft has launched its Copilot+ PCs, and Apple Intelligence has been announced for MacOS.

Green AI. The widespread use of AI makes it a significant contributor to energy consumption, and there is much research on the environmental impact from AI computing. On the positive side, this means that all of the research towards AI improvements is helpful for green AI, since it will also reduce its carbon footprint and environmental impacts. All of those kernel optimizations to speed up the AI engine are also making things greener overall.

AI Market Trends

Here are some future-looking thoughts about what the market for AI may look like. It seems likely that programmers will be required for a little while longer.

It’s a marathon, not a sprint. Consumers may continue to adopt generative AI quickly, but that’s not the most likely case for businesses. Whereas generative AI is a hot topic in boardrooms, most business are still trying to find their feet in the area, with only exploratory projects launching.

Small businesses and professionals (e.g., doctor's offices) will take years to adopt generative AI, and larger enterprises will take even longer. There will be some early projects, sure, but the bulk of the B2B AI market will evolve much more slowly. Projections for the B2B side of AI are over many more years, even decades, with high CAGR.

We've already seen this in the business adoption of cloud architectures, which is still ongoing, despite having been running since the early 2000's. The B2B AI market is likely to sustain very strong growth through 2030 and probably even into the 2040s and beyond.

B2B market opportunity trumps B2C. The massive ramp-up of consumer engagement with ChatGPT has made the consumer side seem hot. However, it's more likely to be the business side that makes more money (as usual). Predictions of the billions, maybe trillions, of dollars of benefit to economies through full AI integration into businesses, dwarf the predictions for consumer opportunities.

Training is the big B2B market? Early wisdom was that the high cost of training and fine-tuning would far exceed inference costs. This contention is somewhat in dispute, with some pundits saying that the sheer number of users will push inference ahead of training. Another factor is the trend toward using someone else's pre-trained LLM, whether it's GPT via the OpenAI API or the open source Llama models. Hence, there's definitely more inference than training in B2C projects, and it may also be taking over on the B2B side.

Fine-Tuning vs RAG. Most business AI projects will involve enhancing the model using proprietary data that the business owns. For example, a support chatbot has to learn information on the company's products, or an internal HR chatbot needs to use internal policy documents.

There are two main ways to do this: fine-tuning or Retrieval-Augmented Generation (RAG). Current training and fine-tuning methods take a long time, need a lot of GPUs, and cost a great deal. However, RAG is becoming widely used to avoid the cost of fine-tuning.

Inference vs Training in the B2C market. Even the B2C generative AI bots need continuous training and fine-tuning, to keep up with current events, so there will also be significant training cost (or RAG costs) in the B2C market. However, with millions of users for B2C apps, the cost of inference should overshadow training costs in the long run.

Bull Case for AI

The bull case is, well, go and look at the NVIDIA stock price. The estimates of revenue available for generative AI are literally over a trillion dollars a year, and we've only spent maybe fifty billion on this infrastructure, so there's a lot to win.

Although there's plenty of hype, there are also those who suggest it's overhyped. The bear case is basically summarized as follows: yes, congratulations, you've spent over fifty billion dollars, and only got a couple back in return.

It's mega-overhyped, as anyone who follows Gartner should know, and the only company laughing on the way to the bank is NVIDIA. There's no killer app for users, except, I mean, well, ChatGPT was only a little bit popular, so nobody's going to care about AI soon.

The bull case is basically that there are two massive buckets of money up for grabs, kind of like in Hunger Games:

- Business apps
- Consumer apps

And, yes, the basic premise of this idea is that neither of these “app” areas have taken off yet, and are still in their infancy. Businesses are only now starting to figure out what they can use generative AI for, whether internally for staff productivity, or externally for their customers. Consumers are using generative AI, mainly for ChatGPT and AI companions, but there are still many new areas that will grow.

And then there's the tech. Come on, be honest, it's pretty amazing. Things like:

- Understanding (of documents and your commands)
- Writing (creatively or professionally)
- Coding
- Audio
- Video creation
- Animation
- Music creation
- Editing text or photos or videos or whatever...

The AI industry has some underpinning trends that affect all AI projects:

- Price of AI inference is dropping fast.
- Multi-AI ensemble architectures are smarter.
- Multi-step reasoning algorithms (i.e., multiple LLM queries per user query).
- Follow-up question asking (like a real personal assistant would).

And then it's going to get even more techie:

- Voice interfaces — your phone stays in your pocket.
- Video understanding — complicated but doable.
- Active autonomous agents — that “do” stuff for you, not just answering questions.
- Context detection — what's on your screen now that you're looking at?
- User presence detection — are you looking at your screen right now?
- Background context analysis — are you in the kitchen?
- Geo-location correlation — are you en route to the airport now?

And there are some massive new form factors and use cases that may broaden the applicability of AI applications (or maybe even displace the prevalence of the smartphone):

- AI gadgets
- Autonomous transportation
- Robotics
- Industrial automation

And there are some other non-AI advances that will add to this imbroglio:

- 5G, 6G, 7G — one per year, maybe.
- NPUs on phones — offline native AI (faster, private, always ready).
- Next-gen silicon — AI-specific chips.
- Memory bandwidth optimizations — tighter GPU/CPU/memory architectures.
- Network optimizations — faster networking going beyond RDMA.
- Quantum computing — half-dead cats and all that jazz.

Users are going to love this stuff! Imagine an intelligent personal assistant in your pocket, which you can speak to, and it answers back into your ear-pods or whatever they're going to be called. I mean, actually intelligent, not the clunky dumb version that you're used to, that's so 2020s.

Bear Case for AI

Although the volume is less, there's a growing debate about whether generative AI is overrunning its fundamentals. Have we gotten a little ahead of ourselves?

Summarizing the main points of the bear case for generative AI, we get:

- Hype? — Just a little bit. It's a bubble (so goes the view).
- No revenue — nowhere near as high as investment.
- User's declining interest
- No killer app (as yet).
- Too expensive — for everyone except NVIDIA.
- Oversupply — AI infrastructure will become a glut.
- Bad for the environment — ten-fold increase in data center electricity usage per query.
- NVIDIA's valuation — it's too high to justify and NVIDIA would need to “grow into it” by earning even more revenue.
- Limitations — generative AI has many limitations, and just isn't that useful in reality.
- Languishing business projects — stuck in “proof-of-concept” status.
- Bad stuff — many examples of risks, false information, hallucinations, dangers, bias, inaccuracy, and outright stupidity.
- Unimportant — disrupted industries are not high-value; writers and artists never got paid well, anyway.
- Probabilistic — non-deterministic algorithms are not desirable for life-critical decisions.
- Legal issues — e.g., bias, copyright, plagiarism, and many more billable hours.

Are you convinced? Do you agree that AI has run out of steam? Or maybe not just yet...

Bull rebuttals. Let us examine some of the bear case in more detail. Some of these points are valid, or have been valid in the past year, but things are changing. Hence, some comments:

- Business projects are emerging from POC into production at an increasing rate as businesses (and technologists) are figuring out how to resolve the issues.
- Demand for GPUs and other infrastructure will grow with increased adoption by both consumers and business, and also from newer GPU-hungry multi-AI architectures combined with multi-step reasoning algorithms.
- Revenue growth: the revenues of OpenAI have doubled in less than six months and there are several other companies minting coin in AI companion bots, writer copilots, coding copilots, and other areas.
- Limitations of generative AI continue to fall by the wayside as an army of researchers tackle them.
- Safety issues and other concerns are reducing due to better training data and improved safety evaluation algorithms.
- Probabilistic algorithms are problematic in some cases (e.g., latency-criticality in a pacemaker implant), but in terms of smartness, consider that the bio-brain of your pilot or doctor also uses fundamentally the same type of algorithm, except it's in carbon not silicon.

Bear rebuttals. The above is somewhat one-sided, so let's have a final look at some of the other negative points again:

- GPU demand faces headwinds from research on (a) faster GPUs, CPUs, and NPUs; (b) optimizing software AI algorithms, which is an area I really like and think its potential is undervalued, and (c) on-device inference on AI phones and AI PCs that don't use big GPUs.
- Environmental impact of AI remains a real concern, unless said research really gets those electricity costs down.
- Safety issues and reasoning limitations of generative AI are somewhat inherent to how it works, and hence, it'll be hard going to resolve them all in a systematic way.
- Many things are still unproven and mostly experimental, such as agent architectures and voice interfaces.

If nothing else, it's going to be a fun ride!

Hype Versus Adoption

As everyone knows, new technologies go through the Gartner hype cycle. This involves initial hype, then a pullback, followed by a slower but more powerful increase in adoption that offers real benefits.

That seems likely to be the case for generative AI, but I wouldn't want to try to guess the timing of any decline. There are some major trends that underpin growth at the moment and will further fuel the need for more generative AI technology work:

- Consumer adoption is increasing (and for employees, too!)
- New advances are dropping about monthly — e.g., text-to-video, avatars.
- Voice interfaces — making it easier on the fingers and more natural to use.
- Multi-AI capabilities — two AI engines are smarter than one, but need twice the juice.
- Vertical-specific products — industry-specialized models are better.
- On-device AI apps for phones and PCs — still a lot to build in this space.
- Increasing intelligence — every extra percentage point in model capability is coming at a higher marginal cost for both training and inference.

And the other thing that would spur even more usage: a killer app.

Killer Apps

One premise of the AI bear case is that there's "no killer app" (yet). But this seems a little disingenuous given the massive adoption of generative AI by consumers, especially with ChatGPT being the fastest growing app in the history of history. Some of the current batch of "killer apps" in the consumer space include:

- ChatGPT — used for writing drafts, asking questions, brainstorming, etc.
- AI companion bots — Character.AI has insane traffic levels.
- Image generation — e.g., for book covers!
- AI search — much smarter answers to user queries on Google and Bing.

And there's some more potential killer apps on the horizon:

- Video auto-generation — amazing, but currently limited in terms of clip lengths.
- Voice interaction with smart assistants — a smarter Siri with voice interface is the potential killer app on the iPhone with Apple Intelligence.
- AI gadgets — these haven't hit mainstream, but offer potential disruption of the smartphone's current dominance.
- Agents — not just answering questions, but doing things for you (probably with your pre-approval, at least to start with).

Consumer Super App? Will there be a super app, that rises above all the rest? Well, there already is one, and it totally dominates smartphone usage in China. It's called WeChat, and there's hardly any smartphones in China that aren't running it right now, as you read this. Yet, it's almost unknown in the West. WeChat runs on top of iPhone and Android platforms, and has its own layer of “mini-apps” and payment support.

It's never caught on in the USA or Europe, and I don't know why. But delivering on a WeChat clone for the USA is still a wishful dream of many tech companies, and the closest thing we have today is Roblox! Maybe generative AI is the chance for them to do so. If there's a voice agent in your pocket that knows everything and can do anything for you, that might be it.

Enterprise Killer App? I think it's fair to say that there's not yet a clear killer app in the business use of AI. Well, I mean generative AI in particular, because predictive ML has become entrenched in various areas, such as recommendation engines for product ads and content feeds. There have been some clear tactical wins for generative AI in certain business-related use cases:

- Customer support chatbots
- Writer copilots
- Coding copilots
- Data transformation assistants

However, nothing massive has really changed in the overall core framework of how a normal business operates. Some of the potential candidate areas for enterprise-wide advanced killer apps would seem to be:

- ERP systems, but smarter.
- Workflow automation
- No code AI platforms
- Voice interfaces

A lot of vendors claim a lot of things in these areas, but I'm sceptical that anyone has really nailed it. There are fundamental problems in business with the sprawl of disparate applications and data sources across business divisions and geographies.

Will there be an enterprise killer app?

Huge wins with generative AI might get done by the vendors, with their development cost effectively spread across multiple businesses, but they could also be achieved within a single business leader, by building their own internal AI platform.

Remember, AI tech is new and still in its teething phase, so we can expect lots of crying. But it will grow and we don't really know yet into what. This is your chance to do something amazing and be the best!

Happy inferencing!

References

1. Nicholas Grous, Andrew Kim, June 04, 2024, *Generative AI: A New Consumer Operating System*, <https://www.ark-invest.com/articles/analyst-research/generative-ai-a-new-consumer-operating-system>
2. Daniel Sack, Lisa Krayner, Emma Wiles, Mohamed Abbadi, Urvi Awasthi, Ryan Kennedy, Cristián Arnolds, and François Candelon, September 05, 2024, *GenAI Doesn't Just Increase Productivity. It Expands Capabilities*, <https://www.bcg.com/publications/2024/gen-ai-increases-productivity-and-expands-capabilities>
3. Sean Michael Kerner, October 28, 2024, *Enterprise AI moves from 'experiment' to 'essential,' spending jumps 130%*, <https://venturebeat.com/ai/enterprise-ai-adoption-surges-as-organizations-shift-from-experimentation-to-implementation/>
4. Taryn Plumb, October 28, 2024, *Gartner predicts AI agents will transform work, but disillusionment is growing*, <https://venturebeat.com/ai/gartner-predicts-ai-agents-will-transform-work-but-disillusionment-is-growing/>
5. Eugene Cheah, Oct 11, 2024, *\$2 H100s: How the GPU Rental Bubble Burst. H100s used to be \$8/hr if you could get them. Now there's 7 different places sometimes selling them under \$2. What happened?* <https://www.latent.space/p/gpu-bubble>
6. Ashu Garg, Oct 25, 2024, *Why OpenAI's \$157B valuation misreads AI's future*, <https://foundationcapital.com/why-openais-157b-valuation-misreads-ais-future/> (Bullish on the “application layer” saying “The top of the stack is where I see the most promise. ...the most valuable companies of the AI era don't exist yet.”... “The cloud era created over 20 application companies with \$1B+ revenue. In AI, we believe this number could exceed 100.”)
7. Kate Knibbs, Oct 28, 2024, *AI Slop Is Flooding Medium*, <https://www.wired.com/story/ai-generated-medium-posts-content-moderation/>
8. Taryn Plumb, October 28, 2024, *Gartner predicts AI agents will transform work, but disillusionment is growing*, <https://venturebeat.com/ai/gartner-predicts-ai-agents-will-transform-work-but-disillusionment-is-growing/>
9. Will Lockett Nov 2024, *Apple Calls BS On The AI Revolution, They aren't late to the AI game; they are just the only sceptical big tech company*. <https://medium.com/predict/apple-calls-bullshit-on-the-ai-revolution-ae38fdf83392>
10. Grant Gross, 18 Jun 2024, *Generative AI's killer enterprise app just might be ERP*, The Information, <https://www.cio.com/article/2149673/generative-ais-killer-enterprise-app-just-might-be-erp.html>

11. CNBC, June 16, 2024, *Apple's AI killer is... the iPhone*, <https://www.cnn.com/video/2024/06/14/apples-ai-killer-is-the-iphone.html>
12. James O'Donnell, May 1, 2024, *Sam Altman says helpful agents are poised to become AI's killer function*, <https://www.technologyreview.com/2024/05/01/1091979/sam-altman-says-helpful-agents-are-poised-to-become-ais-killer-function/>
13. Rafe Fletcher, July 11, 2024, *Finding AI's Killer Use Case*, <https://www.linkedin.com/pulse/finding-ais-killer-use-case-rafe-fletcher-7welc/>
14. Alistair Barr, May 16, 2024, *What's the killer AI app for consumers? Google finally has a contender*. <https://www.yahoo.com/tech/whats-killer-ai-app-consumers-175056899.html>
15. MV Financial, Jul. 20, 2024, *The Search For The Killer AI App*, <https://seekingalpha.com/article/4705253-search-for-killer-ai-app>
16. David Linthicum, Feb 13, 2024, *3 killer apps for cloud-based generative AI*, <https://www.inforworld.com/article/2335997/3-killer-apps-for-cloud-based-generative-ai.html>

3. LLM Technology Overview

What is AI?

AI is such a trendy and overhyped term that I hardly need to tell you what it stands for. Every single company on the planet is now calling themselves an “AI Company” and they’re not incorrect. I mean, my toaster is technically an AI engine because there’s silicon in there somewhere and it’s “intelligent” enough not to burn bread.

And when you get your dream job as an overpaid Software Engineer doing LLMs at a major tech company, the phrase “AI Engineer” is a great term to impress your kids. Your official title, “ML Engineer”, not so much.

This cuts both ways, though. If you’re haggling the price of your new car at the dealership, maybe stick to ML Engineer. Similarly, if you send your resume to a major tech company with “AI Engineer” as your career aspiration, they’ll throw it in the trash and say, “Noob!” with a bemused look on their face.

AI means anything you want it to, but ML means “Machine Learning” to anyone important enough to have that title. The category of ML is very specific to a piece of software that actually “learns” to be smarter (e.g., by “training”). The main ones this book is about are:

- Transformers (e.g., ChatGPT’s “engine”)
- Large Language Models (LLMs) (e.g., GPT-3 or GPT-4)
- Neural Networks (NNs) (i.e., an “artificial brain”)

The general category of Deep Learning (DL) is the subset of ML involving neural networks. Hence, Transformers are a subset of DL.

Some of the other more specific types of ML include:

- Computer Vision (CV)
- Autonomous Vehicles (AVs) — self-driving cars
- Product Recommendation Systems — e-commerce
- Machine Translation (MT) — foreign language translation
- Content relevancy algorithms — social media feeds

Looking forward, some of the aspirations of the AI industry are capabilities such as:

- Artificial General Intelligence (AGI) — human-like reasoning
- Artificial Super-Intelligence (ASI) — who knows what?

AI Technology Trends

Multi-model AI is here already. We're in the early stages of discovering what can be achieved by putting multiple AI models together. The formal research term for this is “ensemble” AI. For example, GPT-4 is rumored to be an eight-model architecture, and this will spur on many similar projects. As multiple-model approaches achieve greater levels of capability, this will in turn create further demand for AI models and their underlying infrastructure. This will amplify the need for optimizations in the underlying AI engines.

Multimodal engines. Multimodality of the ability of an AI to understand inputs in text and images, and also output the same. Google Gemini and ChatGPT 4o are notable large multimodal models. This area of technology is only at the beginning of its journey.

Fine-tuning fights back! For some time, the recommendation has been to use RAG rather than fine-tuning. However, the advent of Low-Rank Adapters (LoRA) and “multi-LoRA” architectures has changed that. LoRA architectures make it cheaper to fine-tune models, and also faster for inference, due to lower memory overhead. Notably, Apple Intelligence on-device inference is based on a single 3B foundation model and multi-LoRA, with dozens of swappable LoRA adapters. The advantage is low memory usage compared to having multiple foundation models, but with capabilities of specialized versions of the main foundation model for particular use cases.

Longer Contexts. The ability of AI engines to handle longer texts has been improving, both in terms of computational efficiency and better understanding and generation results (called “length generalization”). GPT-2 had a context window with 1024 tokens, GPT-3 had 2048, and GPT-4 originally had versions from 4k up to 32k, but has now advanced to 128k tokens as I write this (November, 2023). An average fictional novel starts at 50,000 words and goes up to 200,000 words, so we’re getting to the neighborhood of having AI engines generate a full work from a single prompt, although, at present, the quality is rather lacking compared to professional writers.

AI Gadgets. The use of AI in the user interface has made alternative form factors viable. Some of the novel uses of AI in hardware gadgets include the Rabbit R1, Humane Ai Pin, and Rewind Pendant. There’s been some overreach in these startups, and they haven’t achieved their potential, but I think there’s still room for big changes in how people use AI. On the other hand, maybe the smartphone will win, and the go-to AI interface will be voice conversations with a smart AI assistant app in your pocket.

Agents and Multi-Agent. Agents are LLMs that can do stuff. This means accessing more data sources (e.g., reading your emails), or performing actions for you (e.g., sending an email). Data source agents are sometimes called “plug-ins” after the OpenAI feature. Action agents or “write” agents may require human approval or could run unattended. Both data source and action agents will require a lot of integration work. CrewAI, Pythagora and Devin are examples of multi-agent frameworks with tool usage, too.

Autonomous agents. There are also “autonomous” types of agents that sit in the background and work on a continual basis, like Windows services or Unix daemons, rather than waiting for human requests. The autonomous agent architecture is a combination of an AI engine and LLMs with a datastore and a scheduler.

Small Language Models. Although the mega-size foundation models still capture all the headlines, small or medium size models are becoming more common in both open source and commercial settings. They even have their own acronym now: Small Language Models (SLMs). Notably, Google, Apple and Microsoft are all on the SLM bandwagon. Google has its Gemma models, which are lightweight models that can work on both low-end devices and NVIDIA GPUs. Apple Intelligence has a 3B foundation model as the basis for its on-device inference on iPhone and Mac, using the optimization of multiple LoRA adapters for fine-tuning. Microsoft has also been doing some work in this area with its Orca and Phi models. IBM Granite 3.0 models are enterprise-focused and open source with 3B/8B versions. Apparently 7B or 8B is “small” now.

Specialized Models. High quality, focused training data can obviate the need for a monolithic model. Training a specialized model for a particular task can be effective, and at a much lower cost. Expect to see a lot more of this in medicine, finance, and other industry verticals.

Data Feed Integrations. AI engines cannot answer every question alone. They need to access data from other sources, such as the broad Internet or specific databases such as real estate listings or medical research papers. Third-party data feeds can be integrated using a RAG-style architecture.

Tool Integrations. Answering some types of questions requires integration with various “tools” that the AI Engine can use for supplemental processing in user requests. For example, answering “What time is it?” is not possible via training with the entire Wikipedia corpus, but requires integration with a clock. Implementing an AI engine so that it knows both when and how to access tools is a complex engineering issue.

The Need for Speed. The prevailing problem at the moment is that AI engines are too inefficient, requiring too much computation and too many GPU cycles. The solution will be research into faster software algorithms and optimization techniques, and increasingly powerful hardware underneath.

Why is AI Slow?

Why is AI so slow? It’s a fair question, since the computing power required by AI algorithms is legendary. The cost of training big models is prohibitive, and getting even small models to run fast on a developer’s desktop PC is problematic.

But why?

The bottleneck is the humble multiplication. All AI models use “weights” which are numbers, often quite small fractions, that encode how likely or desirable a particular feature is. In an LLM, it might encode the probabilities of the next word being correct. For example, simplifying it considerably, a weight of “2.0” for the word “dog” would mean to make the word twice as likely to be the next word, and a weight of “0.5” for “cat” would mean to halve the probability of outputting that word. And each of these weights is multiplied against other probabilities in many of the nodes in a neural network.

How many multiplications?

Lots! By which we mean billions every time it runs. A model of size 3B has 3 billion weights or “parameters” and each of these needs multiplication to work. GPT-3 as used by the first ChatGPT release had 175B weights, and GPT-4 apparently has more (it’s confidential but an apparent “leak” rumored that it’s a multi-model architecture with 8 models of 220B parameters each, giving a total of more than 1.7 trillion trained parameters).

Why so many weights? Short answer: because every weight is basically a little tiny bit of braininess.

Longer answer: because it has weights for every combination. Simplifying, a typical LLM will maintain a vector representation of words (called the model’s “vocabulary”), where each number is the probability of emitting that word next. Actually, it’s more complicated, with the use of “embeddings” as an indirect representation of the words, but conceptually the idea is to track word probabilities. To process these word tokens (or embeddings), the model has a set of “weights”, also sometimes called “parameters”, which are typically counted in the billions in advanced LLMs (e.g., a 3B model is considered “small” these days and OpenAI’s GPT-3 had 175B).

Why is it slow on my PC? Each node of the neural network inside the LLMs is doing floating-point multiplications across its vocabulary (embeddings), using the weights, whereby multiplication by a weight either increases or decreases the likelihood of an output. And there are many nodes in a layer of an LLM that need to do these computations, and there are multiple layers in a model that each contain another set of those nodes. And all of that is just to spit out one word of a sentence in a response. Eventually, the combinatorial explosion with the sheer total number of multiplication operations catches up to reality and overwhelms the poor CPU.

Bigger and Smarter AI

Although the compute cost of AI is a large negative, let us not forget that this is what achieves the results. The first use of GPUs for AI was a breakthrough that heralded the oncoming age of big models. Without all that computing power, we wouldn’t have discovered how eloquent an LLM could be when helping us reorganize the laundry cupboard.

Here’s a list of some of the bigger models that have already been delivered in terms of raw parameter counts:

- MPT-30B (MosaicML) — 30 billion
- Llama2 (Meta) — 70 billion
- Grok-1 (XAI) — 70 billion
- GPT-3 (OpenAI) — 175 billion
- Jurassic-1 (AI21 Labs) — 178 billion
- Gopher (DeepMind/Google) — 280 billion
- PaLM-2 (Google) — 340 billion
- MT-NLG (Microsoft/NVIDIA) — 530 billion
- PaLM-1 (Google) — 540 billion
- Switch-Transformer (Google) — 1 trillion
- Gemini Ultra (Google) — (unknown)
- Claude 2 (Anthropic) — 130 billion (unconfirmed)
- GPT-4 (OpenAI) — 1.76 trillion (unconfirmed)
- BaGuaLu (Sunway, China) — 174 trillion (not a typo)

Note that not all of these parameter counts are official, with some based on rumors or estimates from third parties. Also, some counts listed here are not apples-to-apples comparisons. For example, Google’s Switch Transformer is a different architecture.

The general rule of AI models still remains: *bigger is better*. If you’re promoting your amazing new AI foundation model to investors, it’d better have a “B” after its parameter count number (e.g., 70B), and soon it’ll need a “T” instead. All of the major tech companies are talking about trillion-parameter models now.

The rule that bigger is better is somewhat nuanced now. For example, note that Google’s PaLM version 2 had fewer parameters (340B) than PaLM version 1 (540B), but more capabilities.

It seems likely that a few hundred billion is getting to be enough parameters for any use cases, and there is more value in quality of training at that level.

Furthermore, training bigger models only works if you have the data to feed it. The availability of trillions of tokens of input data is starting to be a limiting factor for the industry.

Another change is the appearance of multi-model architectures. Notably, the rumored architecture of GPT-4 is almost two trillion parameters, but not in one model.

Instead, the new architecture is (apparently) an eight-model architecture, each with 220 billion parameters, in a “mixture-of-experts” architecture, for a total of 1.76 trillion parameters. Again, it looks like a few hundred billion parameters is enough for quality results.

We’re only at the start of the multi-model wave, which is called “ensemble architectures” in the research literature. But it seems likely that the overall total count of parameters will go upwards from here, in the many trillions of parameters, whether in one big model or several smaller ones combined.

Faster AI

It’ll be a few years before a trillion-parameter model runs on your laptop, but the situation is not hopeless for AI’s sluggishness. After all, we’ve all seen amazing AI products such as ChatGPT that respond very quickly. They aren’t slow, even with millions of users, but the cost to achieve that level of speed is very high. The workload sent to the GPU is immense and those electrons aren’t free.

There is currently a large trade-off in AI models: *go big or go fast*.

The biggest models have trillions of parameters and are lumbering behemoths dependent on an IV-drip of GPU-juice. Or you can wrap a large commercial model provider through their API (e.g., OpenAI’s API, Google PaLM API, etc.), and using a major API has a dollar cost, although it probably replies quickly.

Smaller models are available if you want to run fast. You can pick one of several smaller open-source models. Here’s a list of some of them:

- Llama2 (Meta) — 70 billion
- MPT-30B (MosaicML) — 30 billion
- MPT-7B (MosaicML) — 7 billion
- Mistral-7B (Mistral AI) — 7 billion

The compute cost of models in the 7B type range is much less. The problem with using smaller models is that they’re not quite as smart, although a 7B model’s capabilities still amaze me. These can definitely be adequate for many use cases, but tend not to be for areas that require finesse in the outputs, or detailed instruction following. Given the level of intense competition in the AI industry, a sub-optimal output may not be good enough.

For more capability, there are larger open-source models, such as Meta’s Llama2 models, which has up to 70 billion parameters. But that just brings us back to the high compute costs of big models. They might be free of licensing costs, but they’re not free in terms of GPU hosting costs.

What about both faster and smarter? So, you want to have you cake and eat it, too? That’s a little trickier to do, but I know of a book that’s got hundreds of pages on exactly how to do that.

There are many ways to make an AI engine go faster. The simplest is to use more GPUs, and that’s probably been the prevailing optimization used to date. However, companies can’t go on with that business model forever, and anyway, we’ll need even more power to run the super-advanced new architectures, such as the multi-model AI engines that are emerging.

Algorithm-level improvements to AI are required to rein in the compute cost in terms of both cash and environmental impact. An entire industry is quickly evolving and advancing to offer faster and more efficient hardware and software to cope with ever-larger models.

But you can save your money for that Galápagos vacation: *code it yourself*. This whole book offers a survey of the many ways to combat the workload with optimized data structures and algorithms.

Human ingenuity is also on the prowl for new solutions and there are literally thousands of research papers on how to run an AI engine faster. The continued growth of models into trillions of parameters seems like a brute-force solution to a compute problem, and many approaches are being considered to achieve the same results with fewer resources. Some of these ideas have made their way into commercial and open source engines over the years, but there are many more to be tested and explored.

AGI

The race to Artificial General Intelligence (AGI) is well under way. It seems plausible that it’s possible to have highly intelligent LLMs, but there are also some reasons to doubt.

Some of the reasons to be bullish on AGI include:

- Brute-force usually wins — it’s called the “bitter lesson” and refers to the fact that computers usually beat humans not by using smarter algorithms, but simply by out-computing us with simpler algorithms.
- Rapid progress against benchmarks — watch any LLM leaderboard for a few months, and there are always new models doing better.
- Multimodal capabilities — there are many new models that can process inputs and outputs involving images, video, and voice.

Not everyone thinks that LLMs can achieve AGI. In one theory, an LLM is compared to our subconscious, and we need something else that mirrors our rational conscious brain, which would then combine with an LLM to have human-like AGI. There are a lot of LLM limitations, but some of the major obstacles to achieving AGI include:

- Reasoning difficulties — LLMs have trouble generalizing information to more complex tasks, or conceptually similar tasks.
- Continual learning is missing — LLMs currently learn nothing new from conversations or articles it reads. The LLM literally forgets everything it reads.

Overall, LLMs are great at memorization and parroting. Effectively, it’s all *imitation*. Will it ever be real?

References

1. Vincent Koc, Jan 3, 2024, *Navigating the AI Landscape of 2024: Trends, Predictions, and Possibilities*, Towards Data Science, <https://towardsdatascience.com/navigating-the-ai-landscape-of-2024-trends-predictions-and-possibilities-41e0ac83d68f>
2. Taryn Plumb, October 28, 2024, *Gartner predicts AI agents will transform work, but disillusionment is growing*, <https://venturebeat.com/ai/gartner-predicts-ai-agents-will-transform-work-but-disillusionment-is-growing/>
3. James Thomason, April 12, 2024, *Why small language models are the next big thing in AI*, <https://venturebeat.com/ai/why-small-language-models-are-the-next-big-thing-in-ai/>
4. Andreesen Horowitz, August 21, 2024 *The Top 100 Gen AI Consumer Apps*, <https://a16z.com/100-gen-ai-apps-3/>
5. Sudeep Srivastava, July 12, 2024, *Top AI Trends in 2024: Transforming Businesses Across Industries*, <https://appinventiv.com/blog/ai-trends/>

6. Lucas Mearian, 24 Oct 2024, 2025: *The year of the AI PC*, Computer World, <https://www.computerworld.com/article/3583355/2025-the-year-of-the-ai-pc.html>
7. Kevin Mahaffey. Oct 25, 2024, *Defensibility: Where will value accrue in AI?* Introduction: Why this time is different. <https://writing.snr.vc/p/defensibility>
8. Tanay Jaipuria, Oct 01, 2024, *Open AI and Anthropic Revenue Breakdown: Breaking down revenue growth, the consumer subscription businesses and the importance of partnerships to the API business*, <https://www.tanayj.com/p/openai-and-anthropic-revenue-breakdown>
9. Daihang Chen, Yonghui Liu, Mingyi Zhou, Yanjie Zhao, Haoyu Wang, Shuai Wang, Xiao Chen, Tegawendé F. Bissyandé, Jacques Klein, Li Li, 9 Jul 2024, *LLM for Mobile: An Initial Roadmap*, <https://arxiv.org/abs/2407.06573>
10. Samantha Kelly, Sept. 29, 2024, *'Superintelligent' AI Is Only a Few Thousand Days Away: OpenAI CEO Sam Altman*, <https://www.cnet.com/tech/services-and-software/superintelligent-ai-is-only-a-few-thousand-days-away-openai-ceo-sam-altman/>
11. Aki Ranin, Sep 2, 2024, *The Code Canaries Are Singing — Our Path Toward AGI: How the fate of human software developers reveals our path toward AGI*, <https://akiranin.medium.com/the-code-canaries-are-singing-our-path-toward-agi-6c234cae0189>
12. Chloe Berger, October 2, 2024, *Mark Cuban says his puppy is 'smarter than AI is today'*, <https://fortune.com/2024/10/01/mark-cuban-dog-puppy-smarter-than-ai/>

Part II: Selecting AI Projects

*“I want AI to do my laundry and dishes
so that I can do art and writing,
not for AI to do my art and writing
so that I can do my laundry and dishes.”*

— Joanna Maciejewska, Author, March 2024.

4. Deciding

Decisions, Decisions

Come on, you don't need to decide! It's AI, and it's already been decided for you. But there are a few other decisions:

- What AI projects to build?
- External versus internal projects?
- How soon do we need it?
- How many users to support?

And then there are some meta-decisions:

- Who decides?
- Who's going to research the technology?
- Whose responsibility is building it?
- Who's tracking the project?
- Who's paying for it?
- Is the legal department prepared?

Assuming you decide to do something, there's the decision set on how to get there:

- Build versus buy?
- Outsource versus in-house development?
- Cloud versus on-premises versus on-device?
- What front-end devices for the user interface?

Project Priorities and Goals

Why are you doing a generative AI project? When you're trying to put out an AI project, whether to the general public or internally to your staff, there are two main competing priorities:

- Fear Of Missing Out (FOMO)
- Fear Of Fouling Up (FOFU)

Both are very valid concerns! Take too long and your competitors will be ahead of you. Even worse, there'll be someone in a hoodie you've never heard of with a multi-billion dollar startup.

But there are also many examples of public failures and the consequent PR embarrassment from newly-released AI projects. Things like telling people to use glue on their pizza, or it's fine to pick mushrooms without any concerns, or insensitively-written obituaries.

Goals. The goals of your project might be:

- Company reputation
- Market positioning
- Growth (of revenues, users, market share, etc.)
- Productivity
- Cost-reduction
- User empowerment (adding capabilities)
- Customer requests

And there may also be defensive goals:

- Not falling behind in tech
- Fighting the disruptors
- Staff morale

These are all laudable goals, but some of them are tough to achieve. Cost reduction and productivity improvement is a tactical goal, whereas having protection against disruptors is a long-term strategic goal.

Choosing Your AI Project

What's the project? Here are some examples of common projects for business usage of AI:

- Writer copilot for marketing or other departments.
- Coding copilots for software developers.
- Support chatbot for your website, that directly answers customer questions about your products.
- Q&A internal service for support staff to help answer questions and offer “scripts” to follow.
- In-house Q&A service to answer sales staff questions about your products (with more in-depth answers possible than a public chatbot)
- In-house HR chatbot to answer staff questions about HR policies and internal company matters.

The above are all quite large projects. One approach is to find value in automating much smaller areas for incremental improvements to workflow. Some examples in the IT department could include:

- Post-processing error messages and alerts by passing the error message through ChatGPT. This can make them more readable, and offer “expert advice” on how to fix them.
- Generating minor technical formats, such as JSON configuration files or rules, for whatever IT software you're already using (e.g., for Prometheus, Graylog, etc.).
- Generating SQL queries from a text description, to query whatever databases are lying around.
- Writing small Linux shell scripts for whatever is needed.

Pro tip: you can choose more than one AI project!

A common type of first AI project is to get certain groups of staff trained up with AI tools to improve their productivity. These are the various “copilot” types of AI tools, and they can be used by various different company teams, even programmers. An important distinction here is that such copilot tools may not require the LLM to have any training with specific in-house data, which is another reason they can be launched faster.

What are Other Companies Doing?

What's everyone else doing in AI? On the one hand, there's a firehose of press articles about generative AI. On the other hand, many of the major players are startups that are still private, with opaque financials. Nevertheless, I've tried to accumulate some useful data points.

Follow the money. Who's making money from generative AI? So far, some of the major winners where the information is available include:

- NVIDIA — H100 chips.
- OpenAI — ChatGPT has over \$3 billion in revenue.
- Microsoft — Azure and also various copilots.
- Accenture — over \$2 billion in AI project bookings, versus \$300m a year ago.
- Scale AI — approaching \$1 billion in revenue.

OpenAI revenue. It is unclear whether OpenAI is making coin from individuals or businesses. However, they just made the consumer version free to use without a login (i.e., they just “ungated” access), and I doubt they'd do that if consumers were their main revenue. Some estimates are that the OpenAI API is the greatest source of revenue, which means business customers and other AI-first startups.

Scale AI business revenue. Scale AI, which is a startup that specializes in data cleanup and data labeling for AI, provided some insightful public information when they raised some more funding. Recently, they announced a massive increase in revenue, and also the change in that 90% of recurring revenue is now driven by corporate customers. Previously, their main customers were the big AI startups building foundation models.

General market status. My analysis of the above points and from reading many, many articles is that generally, the situation is this:

- Lots of companies buying or renting H100 chips (and now B100s)
- Lots of spending on consultancy services (unclear to what extent it's advice or build-it-for-me projects).
- Lots of “buy” rather than “build” projects for internal staff productivity (e.g., writing copilot products, coding copilots, Microsoft Copilot, etc.)
- On-device AI for phones (Apple and Android) is just starting.
- AI PC on-device applications are really just starting (e.g., Microsoft and Copilot+ PCs).
- AI security products.

Business project status. In terms of business-specific AI development projects, my conclusions from this:

- Lots of businesses are doing AI projects (or at least “experimentation”).
- About 2 years into a 3-5 year deployment cycle of specialized business AI applications.
- Many projects are stuck in POC and pre-production (but this is changing).
- Employees aren’t waiting for their company to catch up, and are using consumer AI products in their work on a massive scale (and thereby leaking lots of your proprietary data!).
- Building AI apps is a struggle because nobody knows what to do — too many complicated issues, and too many vendors.
- Safety issues and regulatory compliance are another pain point slowing adoption.
- Dirty data from inside the company is a major bottleneck for many business AI projects.
- Public-facing external AI applications are much less common than internal usage.
- Business-specific AI “platforms” based on internal competencies are rare.

This all sounds great, but there’s really only one big problem.

Confusion Reigns

Well, at least the consultants are doing well. Presumably, that’s because nobody else really knows what to do. And just between you and me, I’m not sure that the consultants know all the answers either.

In order to help out, here’s my list of areas of confusion in building an AI app:

- What to build
- What data to use
- What platform to use
- What model to use
- How to build it
- How to test it
- Whether it’s legal

Maybe it’s better just to stay in bed. But at least you’re not alone. Everyone’s confused!

The Easier Options

In order to take it easier on yourself, consider the simpler projects first. Here are some thoughts on easier ways to go into AI. Note that I didn't say that they are the cheapest ways, or the most beneficial. All I said was "easier" and I definitely did not say "easy."

Here are the easier types of projects to choose:

- Buy is easier than build.
- Internal usage is easier than customer-facing.
- Text is easier than images, which is easier than video.
- Human-in-the-loop is easier than unattended or automated (i.e., a real person reviewing AI's outputs makes it easier to avoid pitfalls).
- Read before you write (i.e., choose projects that just answer questions, but don't "change" anything).
- Web-based browser interfaces to cloud backends are easier than on-device AI.
- AI agents are hard and very new.
- Building your own AI platform is the worst.

To the last point, the hardest project is building your own "platform" that allows you to extend easily into multiple business-specific AI applications. Although it's tough to do, this idea has the greatest long-term strategic benefit.

Internal Business AI Platforms

There are a lot of short-term wins in staff productivity and addressing common customer pain points. A lot of these are "buy" rather than "build" and the tendency will be to de-prioritize the more complex AI projects. However, these short-term projects are not the big strategic wins that realize the promise of AI. That can be summed up as: *a business-specific AI platform*.

What's a business AI "platform"? Well, in the same ways that the internet is widely used internally by business staff and externally to communicate with customers, an AI platform will generalize that idea. The major capabilities include:

- Internal business processes (simplified, automated, extended)
- Customer communication (personalized, automated, integrated)
- Data analysis (deeper patterns, timeframe planning, niche segmentation)

Making AI into a core competency within your organization is a long-term strategic advantage. Every business is different, and business-specific requirements can go well beyond the generic capabilities of writing copilots, customer support chatbots, and other off-the-shelf AI systems.

In assessing your long-term strategy for AI, consider what makes your company unique. What are your strategic advantages against competitors? Then consider how generative AI could enhance and extend those capabilities, looking at both inside and outside the company.

AI Core Competency Levels

It's very complicated at the moment, but there seems to be a natural progression occurring. The distinction can be mapped as:

- Tactical AI projects
- Strategic use of AI

Here are my thoughts on a hierarchy of AI core competency levels inside a business.

Tactical AI. For short-term AI projects, the levels are:

- Buy for internal staff — e.g., writer copilots, coding copilots, or other turnkey AI applications.
- Buy for external customers — e.g., a “custom” customer support chatbot based on RAG or fine-tuning from auto-ingesting your website.
- Build for internal staff — e.g., automate some of the more annoying internal repetitive business processes for improved staff productivity.
- Build for external customers — e.g., clean up internal proprietary data and use this for better fine-tuning of specific customer-facing apps.

The goals of tactical AI projects are likely to be specific and measurable. Examples include:

- Cost reduction
- Productivity improvement (e.g., words written, SLOC, etc.)
- Customer satisfaction metric improvement (e.g., average time to resolve customer questions).
- Conversion rates (e.g., personalized sales content in customer outreach).

Strategic AI. Well, the tactical AI projects have been going on for a while. If you really want to get ahead in the longer term, think strategy.

The strategic competency levels are much more difficult:

- Internal business-specific AI platform — extend your capabilities to allow easy creation of many “mini-apps” for business process automation.
- External business-specific AI platform — control the touch points with your customers in a way that you can easily leverage and extend.

At the strategic level, the goals will be less quantitative, and longer-term:

- Repeatability — quickly bringing new capabilities online using the same process and underlying framework.
- Scalability — being able to scale onboarding quickly, whether it’s staff or customers.
- Specialization — use of your organization’s core competencies.

Running your own AI platform is the biggest long-term win. I’m not sure that you need to run your own H100s, and you might simply build out your own platform by customizing and extending an existing AI infrastructure platform. But I think it’s important to control the ability to add higher-level functionality in your core business areas. By owning the innovation level, you get to drive it forward with specialized capabilities that your competitors will struggle to match. It takes time and money, but this will be the long-term battleground in generative AI.

In order to avoid sending too much of your cash to the hyperscalers while building your AI platform, there are some ways to reduce costs.

Early on, and for the development of many features, mock the AI part. A local AI with a tiny model that conforms to the OpenAI APIs will work, so the whole app can be developed, debugged, and its many parts tested without paying by the token or risking exposure of any IP. Once a criteria is met, switch over to the “real” AI and start doing the integration testing properly.

Weirdly, the opposite plan also works to some extent, too.

The queries/prompts can be created and validated ahead of time using the full AI model (e.g., ChatGPT interactive), before development starts, even just to prove the value of the AI approach. This kind of “scoping” or “feasibility” work need not be performed by developers, so it could be given to marketing. These AI actions will cost a relatively small amount of money, and you can POC the AI features before you try to develop them!

References

1. Gene Rapoport, Sanjin Bicanic, Jue Wang, Richard Lichtenstein, Arjun Dutt, June 20, 2024, *AI Survey: Four Themes Emerging: If 2023 was about experimentation, 2024 is all about results*. Bain & Company, <https://www.bain.com/insights/ai-survey-four-themes-emerging/> (Bain reports that use cases have been broadly successful in the use cases of sales, sales operations, software development, marketing, customer service, and customer onboarding, but less successful in HR, operations and legal. Interestingly, the main reason for AI project failures was that it couldn't perform the necessary task.)
2. Timothy Mugayi, Sep 2024, *LLM Practical Ideas to Build Your Next AI-Powered Application: Realistic Use Cases to Unleash the Power of AI in Your Next Project*, <https://levelup.gitconnected.com/llm-practical-ideas-to-build-your-next-ai-powered-application-9379feba6cbc>
3. Irene Weber, 13 Jun 2024, *Large Language Models as Software Components: A Taxonomy for LLM-Integrated Applications*, <https://arxiv.org/abs/2406.10300>
4. Chuan Yan, Ruomai Ren, Mark Huasong Meng, Liuhuo Wan, Tian Yang Ooi, Guangdong Bai, 26 Aug 2024, *Exploring ChatGPT App Ecosystem: Distribution, Deployment and Security*, <https://arxiv.org/abs/2408.14357>
5. A16Z, April 2nd, 2024 (accessed), *AI Getting Started*, <https://github.com/a16z-infra/ai-getting-started> (Javascript wrapper kits for several commercial AI APIs.)
6. Joe McKendrick, March 28, 2024, *This year's top 8 use cases for AI, and what tech professionals need to support them*, <https://www.zdnet.com/article/this-years-top-8-use-cases-for-ai-and-what-tech-professionals-need-to-support-them/>
7. Rafe Fletcher, July 11, 2024, *Finding AI's Killer Use Case*, <https://www.linkedin.com/pulse/finding-ais-killer-use-case-rafe-fletcher-7welc/>

8. Stephanie Houde, Vera Liao, Jacquelyn Martino, Michael Muller, David Piorkowski, John Richards, Justin Weisz, Yunfeng Zhang, 2 Mar 2020, *Business (mis)Use Cases of Generative AI*, <https://arxiv.org/abs/2003.07679>
9. Olivia Moore, March 13, 2024, *The Top 100 Gen AI Consumer Apps*, <https://a16z.com/100-gen-ai-apps/>
10. MV Financial, Jul. 20, 2024, *The Search For The Killer AI App*, <https://seekingalpha.com/article/4705253-search-for-killer-ai-app>
11. David Linticum, Feb 13, 2024, *3 killer apps for cloud-based generative AI*, <https://www.infoworld.com/article/2335997/3-killer-apps-for-cloud-based-generative-ai.html>

5. Use Cases

What are Use Cases?

The term “use cases” is the industry term for the various ways that LLMs and AI engines are used. Consumers and businesses use AI tools differently, and hence, have different use cases.

Consumers are using the ChatGPT product mainly for question answering, writing text, brainstorming, and as a companion. Businesses are using old-school predictive AI to give content feeds to their customers, and using generative AI for customer support chatbots and marketing content writing.

Use cases are continually expanding, and the onset of the new “on-device” power of generative AI in AI phones and AI PCs will further grow the role of LLMs. The newer set of “agent” AI architectures are broadening the use cases to cases where the LLM takes an action on your behalf, such as sending an email or text.

Gen AI vs ML Use Cases

Generative AI is the new kid on the block. ML is the incumbent technology, already living the high life outside the DMZ.

They have different strengths and weaknesses. Hence, to understanding what projects to build with generative AI versus ML, it’s important to compare them.

Table 1: Generative AI vs Predictive ML

Technology	Strengths	Weaknesses
Generative AI	Words Language	Data patterns Number crunching
Predictive ML	Data patterns Number crunching	Words Language

Did you notice anything, or were you just skimming? ML and generative AI are quite complementary technologies. It's not very likely that you want to replace an ML application with a version that uses an LLM. Instead, you want to look for new uses of the new AI technology that is very good at both understanding and writing natural language.

The use cases that ML is already used for, which is the same as the use cases you probably won't use gen AI for, includes:

- Content recommendations
- Product recommendations
- Pattern detection in numeric data (e.g., quantitative trading analysis)
- Face recognition
- Cyber security (e.g., intrusion detection)
- Financial fraud monitoring
- Machine vision (e.g., autonomous vehicles)

However, you can combine gen AI and ML. For example, in the product recommendation space, the ML tools are best at choosing what to recommend, but an LLM is better at writing personalized content about that product when presenting it to the user in an encouraging way.

Longstanding AI Use Cases

AI has been around for years, and there are various things it's always been good at, long before generative AI and LLMs came along. Some examples of things that computers could do before ChatGPT include:

- Categorization
- Summarization
- Machine language translation (i.e., foreign language translators)
- Voice recognition and understanding
- Transcription (voice-to-text)
- Voice generation (text-to-voice)
- OCR (optical character recognition, image-to-text)
- Content feed relevancy algorithms
- E-commerce product recommendations
- Sentiment analysis
- Named Entity Recognition (NER)
- Pattern detection in complex numerical data
- Facial recognition (e.g., your face is your password)

A lot of these items are technological features rather than use cases. There's nothing wrong with aiming for any of the above ideas, if they fit with your business needs. The only point is that these are mostly old-school ML capabilities, rather than tasks that LLMs can do well.

Business Use Cases

There are many, many business projects ongoing that use generative AI capabilities. Some of the initial areas of success have included:

- Customer support chatbot
- Support staff internal scripts (generation)
- Writing content (various uses within an organization)
- Marketing content drafting/generation
- Product listings (e.g., e-commerce product descriptions, real estate listings, etc.)
- HR internal chatbot (employees searching policies)
- HR resume analysis
- Programming (e.g., coding copilots, debugging, etc.)
- Personal copilot (e.g., help staff productivity with email or document drafting)

The above list tends to be the quicker wins with generative AI. Some of these can also be done as a “buy” project rather than “build.”

Furthermore, you’ll notice that most of these are staff uses rather than customer-facing applications, which makes sense because internal projects are easier than external ones.

Consumer Uses of Gen AI

Even if you’re looking for business AI projects, it can be helpful to consider what your customers are already doing with AI. This gives insight into how your customers might want to interact with your business in the future. Or possibly, you’re a technology startup writing AI apps for consumers, like me.

What are consumers using generative AI for:

- Generating things (text, video, audio, etc.)
- Companions
- Question & answer
- Internet search with summary (Google, Bing)
- Drafting text documents (e.g., emails, reports, letters, etc.)
- Outlining
- Proofreading, editing, and revising
- Searching long documents (or the internet)
- Summarization (e.g., PDF documents, web pages, long documents, legal briefs, etc.)
- Lists of ideas or topics
- Brainstorming
- Homework answering
- Homework drafting (for teachers)
- Chatting (who knew?)
- Recipes
- Researching products and services
- Travel research and planning
- Tax questions
- Financial questions
- Medical questions

What are they generating? (for better or worse)

- Creative writing (stories, poems, song lyrics, jokes, etc.)
- Non-fiction writing (e.g., business reports, homework, assignments, CVs/resumes, job application letters, etc.)
- Image creation (text-to-image)
- Video creation (text-to-video)
- Music creation (text-to-music)
- Song creation (text-to-lyrics)

No doubt, there's some things missing from that list, but I'm sure you get the idea.

Software Engineering Use Cases

Programmers are also using code models for:

- Code auto-completion
- Code generation
- Debugging
- Documentation
- AI Game creation tools (levels, NPC characters, etc.)
- Data generation
- Test case generation

Actually, not really. We programmers just pretend to use AI tools to keep our boss happy, but we simply don't need any help writing perfect code. Those time blocks we mark on our calendar as "debugging" are actually spent outside playing Ultimate Frisbee.

Instead, it's all the other mundane stuff that we need help with: pull requests, commit comments, change logs, ad hoc scripts, status reports for the boss, emails to other easily-offended humans, sick notes when hung over, and asking favors of another team in the most flattering way.

Professional Use Cases

Professional consumers and small business owners are finding a myriad of uses for generative AI in their everyday lives and in their work. But it's early days with adoption of the technology in a meaningful way, and there are still many areas where generative AI is not yet making any impact.

Professional business uses for individuals include:

- Drafting product descriptions and other content
- Email drafting (with personalization added)
- Blog posts and online content
- Advertising and marketing content creation
- Coding copilot with code completion and debugging (for programmers)
- Writing copilot (for writers)
- Drafting business reports

Some of the more specialty fields include:

- Researching complex issues (scanning document databases)
- Cybersecurity
- SEO content auto-generation
- Automating business processes
- Medical image analysis (radiologists)
- Medical notes and patient case summaries (doctors)
- Legal document analysis and summarization
- Medical research (e.g., drug discovery)
- Agriculture yield optimization

This is a long list and also a short list. The potential benefits of AI extend into many areas and the field is only getting started.

References

1. Rhiannon Williams, July 8, 2024, *How to use AI to plan your next vacation: AI tools can be useful for everything from booking flights to translating menus*, MIT Technology Review, <https://www.technologyreview.com/2024/07/08/1094733/how-to-use-ai-to-plan-your-next-vacation/>
2. Alistair Barr, May 16, 2024, *What's the killer AI app for consumers? Google finally has a contender*. <https://www.yahoo.com/tech/whats-killer-ai-app-consumers-175056899.html>
3. MV Financial, Jul. 20, 2024, *The Search For The Killer AI App*, <https://seekingalpha.com/article/4705253-search-for-killer-ai-app>
4. David Linthicum, Feb 13, 2024, *3 killer apps for cloud-based generative AI*, <https://www.infoworld.com/article/2335997/3-killer-apps-for-cloud-based-generative-ai.html>
5. Olivia Moore, March 13, 2024, *The Top 100 Gen AI Consumer Apps*, <https://a16z.com/100-gen-ai-apps/>
6. Kate Knibbs, Sep 18, 2024, *Most US Teens Use Generative AI. Most of Their Parents Don't Know*, Wired, <https://www.wired.com/story/teens-generative-ai-use-schools-parents/>
7. Stephanie Houde, Vera Liao, Jacquelyn Martino, Michael Muller, David Piorkowski, John Richards, Justin Weisz, Yunfeng Zhang, 2 Mar 2020, *Business (mis)Use Cases of Generative AI*, <https://arxiv.org/abs/2003.07679>
8. Henrique Centieiro & Bee Lee, Jun 26, 2024, *Top 20 GPT-4o Use Cases That Actually Improve Your Everyday Life*, <https://medium.com/the-generator/top-20-gpt-4o-use-cases-that-actually-improve-your-everyday-life-c136f2c802d2>
9. Rafe Fletcher, July 11, 2024, *Finding AI's Killer Use Case*, <https://www.linkedin.com/pulse/finding-ais-killer-use-case-rafe-fletcher-7welc/>
10. Gene Rapoport, Sanjin Bicanic, Jue Wang, Richard Lichtenstein, Arjun Dutt, June 20, 2024, *AI Survey: Four Themes Emerging: If 2023 was about experimentation, 2024 is all about results*. Bain & Company, <https://www.bain.com/insights/ai-survey-four-themes-emerging/>
11. Nicholas Carlini, 2024-08-01, *How I Use "AI"*, <https://nicholas.carlini.com/writing/2024/how-i-use-ai.html>
12. Christopher Tao, Aug 2024, *Do Not Use LLM or Generative AI For These Use Cases*, <https://pub.towardsai.net/do-not-use-llm-or-generative-ai-for-these-use-cases-a819ae2d9779>

13. Harshita Katiyar, Aug 1, 2024, *Artifacts: Top 10 Mindblowing uses of Claude 3.5 Sonnet*, <https://medium.com/aimonks/artifacts-top-mindblowing-uses-of-claude-3-5-sonnet-6830b2acfa4b>
14. Joe McKendrick, March 28, 2024, *This year's top 8 use cases for AI, and what tech professionals need to support them*, <https://www.zdnet.com/article/this-years-top-8-use-cases-for-ai-and-what-tech-professionals-need-to-support-them/>

6. Limitations

Generative AI Limitations

LLMs can do some amazing new things, but they also have a lot of limitations. In order to choose the right generative AI projects, it helps to know what to avoid.

This chapter is a deep dive into limitations in various categories:

- Inaccuracy and answer quality
- Alignment limitations
- Data limitations
- Computational limitations
- Reasoning limitations

We expect rather a lot from our little silicon creations. There's only so much that a few billion multiplications can achieve.

Accuracy Limitations

LLMs are probabilistic and non-deterministic in their answers. But facts are supposed to be definitive, so there's some inherent problems with the architecture. Your average LLM has problems with factual accuracy:

- Hallucinations (plausible-looking made-up facts)
- Inaccuracies or misinformation (wrong facts or omissions)
- Confabulations (wrongly merging two sources)
- Plagiarism (in its training data set)
- Paraphrasing (plagiarism-like)
- Model “drift” (decline in accuracy over time)
- Spin or bias in input training data or RAG data is repeated in answers.
- Censorship

The issue of hallucinations has gotten a lot of attention. There are actually two subtypes:

- (a) the LLM was never trained on the answer and makes up an answer, or
- (b) the LLM actually has learned the correct answer in its trained weights somewhere, but still gets it wrong.

The problems of hallucination lead to further issues such as:

- Over-confidence — it knows not what it says.
- Veneer of authority — users tend to believe the words.
- Gullibility — the LLM does not challenge the input text.
- Acceptance — not challenging or detecting the source information’s bias, credibility, or authority.
- Ambiguity — some results have arguments for and against a topic.

The main problem is that it doesn’t tell you when it’s giving you false information, because it doesn’t know. There is a partial fix possible by adding more GPU juice: use a multi-step inference method such as “self-reflection” or “LLM as judge” to check the initial response. Weirdly, LLMs are quite good at correcting their own answers if you ask them to.

When creating an AI application, it needs to be understood that LLMs will get things wrong, or it will hallucinate false answers that sound very convincing. So, if an LLM is put in a workflow that expects the output to be 100% correct all the time, then trouble will occur. Do not replace feature X with an LLM and gloat about how well the workflow is automated, because it will be broken at times. Instead keep feature X, but use the LLM to “auto-complete” or “auto-generate” output, while still allowing a human to override.

Alignment Limitations

Alignment is the name for the property whereby the LLM’s answers are “aligned” with what we’d want it to do or say. This is related to “instruction following” on the input side, but also on having the answers being what we could expect as the right answer.

There are various problems with the appropriateness of the answers in terms of various alignment metrics:

- Biases (of many types)
- Toxicity
- Harmful content
- Insensitivity (e.g., when writing eulogies).
- Dangerous or harmful answers (e.g., wrong mushroom picking advice)
- Sensitive topics (the LLM requires training on each and every one)
- Alignment (people have purpose; LLMs only have language).
- Security (e.g., “jailbreaks”)
- Refusal (knowing when it should refuse to answer something)
- Locale Awareness (even when care is taken, local knowledge or customs can be offended).

Some other general concerns include:

- LLM use for nefarious purposes (e.g., by hackers)
- Transparency issues (sources of the data, details of the guardrails, how it works, etc.)
- Privacy issues (sure, but Googling online has similar issues, so this isn’t as new as everyone says)
- Legal issues (copyright violations, patentability, copyrightability, and more)
- Regulatory issues (inconsistent across geographies)
- Unintended consequences

All of the above are areas of intense and ongoing research in both academic and commercial labs. A lot of progress has been made, but more is still needed.

Training Data Quality

Sometimes, it’s not the LLM’s fault, but the training data. After all, it’s “garbage in, garbage out” for generative AI answers:

- Surfacing inaccurate or outdated information
- Proprietary data leakage (e.g., trade secrets in an article used in a training data set)
- Personally Identifiable Information (PII) (e.g., emails or phone numbers in training data)
- Poor quality training data, generally.

- Americanisms — the vast majority of English language training data is from American sources, so this style dominates in terms of word spellings, implied meanings, cultural issues like “football”, etc.
- Falling back on over-complex training data (causing unnecessarily complicated answers)

Ensuring that there is “clean” data is an important part of building or fine-tuning an LLM. Better quality training data has been shown to improve accuracy of LLMs, and also has the advantage that it allows smaller models to have better results.

Computational Limitations

There’s really only one big problem with AI computation: it’s slooow. Hence, the need for all of those expensive GPU chips. This leads to problems with:

- Cloud data center execution is expensive.
- AI phone execution problems (e.g., frozen phone, battery depletion, overheating)
- AI PC execution problems (big models are still too slow to run)
- Training data set requirements (they need to feed on lots of tokens)
- Environmental impact (e.g., by one estimate, a ten-fold need of extra data center electricity for AI answers compared to non-AI internet searches)

Bugs. And we shouldn’t forget that Transformers and LLMs are just programs written by software engineers, with problems such as:

- Gibberish output — usually a bug; AI engines are just programs, you know.
- Going rogue — usually a bug, or is it?
- No output — usually a bug or it’s fallen into a hole in the network.
- Prompt fragility — very different results when changing the prompt just a little, such as an extra unimportant word.
- Punctuation handling — e.g., results changing from adding an extra space at end of prompt.

Are these bugs, or maybe they’re features? Only time will tell!

Reasoning Limitations

At a very high level, your LLM has great difficulty with any type of advanced reasoning. In a sense, an LLM is very much like your subconscious, in that it is instinctive and automatic. Humans have a conscious mind that can impose a structured logic on top of the raw low-level capabilities, but this is still an unsolved issue with LLMs. Hence, LLMs are great at making things flow smoothly, in words or images, but not at having it all make sense as a whole.

Let's begin with some of the reasoning limitations that have largely been solved:

- Words about words (e.g., “words”, “sentences”, etc.)
- Writing style, tone, reading level, etc.
- Ending responses nicely with stop tokens and max tokens
- Tool integrations (e.g., clocks, calendars, calculators)
- Cut-off date for training data sets
- Long contexts

Domains of difficulty. Some particular domain areas of reasoning difficulty for LLMs include:

- Logical reasoning
- Planning multiple steps
- Emotions — LLMs don't have them, and can only fake it.
- Time/temporal reasoning — the concept of things happening in sequence is tricky.
- 3D scene visualization — LLMs struggle to understand the relationship between objects in the real world.
- Mathematical reasoning
- Arithmetic (!)
- Specialized domains — e.g., jargon, special meanings of words.
- Math word problems
- Crosswords and other word puzzles — e.g., anagrams, alliteration.
- Humor — except they can do Dad jokes.
- Spelling — LLMs do not spell, but work on tokens that are small collections of letters, not actual letters.
- Counting and Arithmetic — LLMs have no builtin computation engine.

It's rather strange the LLMs have problems with basic arithmetic calculations. But anything with numbers is suspect, especially larger numbers, since four or more digits are typically broken into smaller tokens.

For example, 1234 might break into 12 and 34 as tokens. The LLM has no concept of place values for 12 or 34, and can make mistakes.

Some of the other general areas where there are problems that are somewhat inherent to LLMs and the way they work include:

- Prompt engineering requirements — awkward wordings! Nobody really talks like that.
- Oversensitivity to prompt variations — and yet, sadly, prompt engineering works.
- Over-confidence — LLMs lack insight into confidence levels, and what “understanding” even means.
- Ambiguity — unclear input queries are poorly handled sometimes.
- Probabilistic — non-deterministic output is hard to predict (and test).
- Recent context confusion

LLMs are also easily confused by recent context that you have in a conversation. For example, it’s not hard to tell the LLM something that is false and then have it repeat that back.

Even with GPT-4o, start a conversation with say, “Apples, Strawberries and Bananas are type of Panda,” let it reject that, and then assert it again. Before you know it, you will have the LLM inventing fun types of pandas.

It’s not really clear whether this is a bug where the LLM is wrong, or an alignment feature where it’s following your instructions.

Do you want the LLM to accept what you say or not?

Prompt ambiguities. Sometimes, the training data or the input query is difficult, whereas a human can discern aspects better:

- Words and meanings are not the same thing. People sometimes use words in odd ways.
- Sarcasm and satire (e.g., articles espousing the benefits of “eating rocks”)
- Spin, biased viewpoints, and outright disinformation/deception (of source content)
- Trick questions (e.g., queries that look like common online puzzles, but aren’t quite the same).
- Detecting intentional deception or other malfeasance by users
- Novice assumption (not identifying a user’s higher level of knowledge from words in the questions; dare I say it’s a kind of “AI-splaining”)
- LLMs asking follow-up questions to clarify user requests (this capability has been improving quickly).
- Not correctly prioritizing parts of the request (i.e., given multiple requests in a prompt instruction, it doesn’t always automatically know which things are most important to you)

LLM answer quality. More issues with answers include:

- Over-explaining
- Nonsense answers
- Non-repeatability (same question, different answer)
- Lack of common sense (although I know some people like that, too)
- Lack of a “world model”
- Lack of a sense of personal context (they don’t understand what it means to be a person)

Some other issues high-level issues with answer results:

- Explainability (can it explain why it gave this answer?)
- Attribution (source citations)
- Banal, bland, generic, or overly formal writing
- Repetition (e.g., if it has nothing new to add, it may repeat a prior answer, rather than admitting that)

What a hard time I’m giving to all the poor little llamas. If nothing else, this shows again that getting to AGI will be a marathon, not a sprint. There’s more work to be done.

Mathematical Reasoning

Complex mathematical reasoning remains a struggle for LLMs in Transformer-based architectures. But, I mean, don't we all? It seems a little unfair to criticize our silicon creations, when we too struggle with these problems. Nevertheless, LLMs can answer some very advanced problems, and yet can also struggle with the simplest questions.

Ironically, by mimicking human anatomy in the creation of neural networks, we've also infused them with many of the same weaknesses. Some of the types of logical and mathematical reasoning problems that LLMs can struggle with include:

- Multi-step logical reasoning
- Temporal reasoning (time-based)
- 3D visualization reasoning (e.g., people standing in a circle).
- Math Word Problems (MWP), which are such a common concern as to deserve an acronym.
- Arithmetic on larger numbers.
- Implied mathematical meanings in words (e.g., if Mandy "eats" an apple, that's a subtraction of one apple).

There's more than a little irony in the fact that LLMs can't do arithmetic. They're designed to handle the probabilities of words, and unfortunately, there are infinitely many "number words" that you can make out of numeric digits. The LLM has probably seen "one plus one" in its training data set, but if you tell the LLM to add two 11-digit numbers together, probabilistic meanings of words aren't that helpful.

Oddly, it also makes a kind of sense in that LLMs are human-like. If I handed you two 11-digit numbers, you wouldn't add them in your head either. You'd reach for the calculator, which is what LLMs do with tool integrations. Or otherwise, to do the calculation by hand, you'd use an "algorithm" for adding the massive numbers (i.e., add the digits in columns, carrying the ones). Hence, it makes a weird kind of sense that an LLM fails at these, and needs a multi-step overarching reasoning engine to perform arithmetic on big numbers.

But I still find it funny. I mean, it's not like it has a GPU underneath that could do a billion of those additions every second at 100% accuracy!

References

1. Sean Williams, James Huckle, 30 May 2024, *Easy Problems That LLMs Get Wrong*, <https://arxiv.org/abs/2405.19616> Code: <https://github.com/autogenai/easy-problems-that-llms-get-wrong>
2. Abdelrahman “Boda” Sadallah, Daria Kotova, Ekaterina Kochmar, 15 Mar 2024, *Are LLMs Good Cryptic Crossword Solvers?* <https://arxiv.org/abs/2403.12094> Code: <https://github.com/rdeits/cryptics>
3. Jonas Wallat, Adam Jatowt, Avishek Anand, March 2024, *Temporal Blind Spots in Large Language Models*, WSDM '24: Proceedings of the 17th ACM International Conference on Web Search and Data Mining, Pages 683–692, <https://arxiv.org/abs/2401.12078>, <https://doi.org/10.1145/3616855.3635818>, <https://dl.acm.org/doi/abs/10.1145/3616855.3635818>
4. Juntu Zhao, Junyu Deng, Yixin Ye, Chongxuan Li, Zhijie Deng, Dequan Wang 22 Sept 2023 (modified: 11 Feb 2024), *Lost in Translation: Conceptual Blind Spots in Text-to-Image Diffusion Models*, ICLR 2024, <https://openreview.net/forum?id=vb3O9jxTLc>
5. Victoria Basmov, Yoav Goldberg, Reut Tsarfaty, 11 Apr 2024 (v2), *Simple Linguistic Inferences of Large Language Models (LLMs): Blind Spots and Blinds*, <https://arxiv.org/abs/2305.14785>
6. Julia Witte Zimmerman, Denis Hudon, Kathryn Cramer, Jonathan St. Onge, Mikaela Fudolig, Milo Z. Trujillo, Christopher M. Danforth, Peter Sheridan Dodds, 23 Feb 2024 (v2), *A blind spot for large language models: Supradiegetic linguistic information*, <https://arxiv.org/abs/2306.06794>
7. Michael King, July 24, 2023, *Large Language Models are Extremely Bad at Creating Anagrams*, <https://www.techrxiv.org/doi/full/10.36227/techrxiv.23712309.v1>
8. George Cybenko, Joshua Ackerman, Paul Lintilhac, 16 Apr 2024, *TEL'M: Test and Evaluation of Language Models*, <https://arxiv.org/abs/2404.10200>
9. Yotam Wolf, Noam Wies, Yoav Levine, and Amnon Shashua. *Fundamental limitations of alignment in large language models*. arXiv preprint arXiv:2304.11082, 2023. <https://arxiv.org/abs/2304.11082>
10. Mikhail Burtsev, Martin Reeves, and Adam Job. *The working limitations of large language models*. MIT Sloan Management Review, 65(1):1–5, 2023. <https://sloanreview.mit.edu/article/the-working-limitations-of-large-language-models/>
11. Michael O'Neill, Mark Connor, 6 Jul 2023, *Amplifying Limitations, Harms and Risks of Large Language Models*, <https://arxiv.org/abs/2307.04821>
12. Karl, May 10, 2023, *Large Language Models: Reasoning Capabilities and Limitations*, <https://medium.com/@glovguy/large-language-models-reasoning-capabilities-and-limitations-951cee0ac642>
13. The PyCoach Apr 20, 2024, *The False Promises of AI: How tech companies are fooling us*, <https://medium.com/artificial-corner/the-false-promises-of-ai-fe23124e0fb9>
14. Bill Doerrfeld, Feb 6, 2024, *Does Using AI Assistants Lead to Lower Code Quality?* <https://devops.com/does-using-ai-assistants-lead-to-lower-code-quality/>

15. Piotr Wojciech Mirowski, Juliette Love, Kory W. Mathewson, Shakir Mohamed, 3 Jun 2024 (v2), *A Robot Walks into a Bar: Can Language Models Serve as Creativity Support Tools for Comedy? An Evaluation of LLMs' Humour Alignment with Comedians*, <https://arxiv.org/abs/2405.20956> (The unfunny fact that AI is bad at humor.)
16. Rafe Brena, May 24, 2024, *3 Key Differences Between Human and Machine Intelligence You Need to Know: AI is an alien intelligence*, <https://pub.towardsai.net/3-key-differences-between-human-and-machine-intelligence-you-need-to-know-7a34dcee2cd3> (Good article about how LLMs don't have "emotions" or "intelligence" and they don't "pause".)
17. Amanda Silberling, August 27, 2024, *Why AI can't spell 'strawberry'*, <https://techcrunch.com/2024/08/27/why-ai-cant-spell-strawberry/>
18. Kyle Wiggers, July 6, 2024, *Tokens are a big reason today's generative AI falls short*, <https://techcrunch.com/2024/07/06/tokens-are-a-big-reason-todays-generative-ai-falls-short/>
19. Xinyi Hou, Yanjie Zhao, Haoyu Wang, 3 Aug 2024, *Voices from the Frontier: A Comprehensive Analysis of the OpenAI Developer Forum*, <https://arxiv.org/abs/2408.01687>
20. Andrew Blair-Stanek, Nils Holzenberger, Benjamin Van Durme, 7 Feb 2024 (v2), *OpenAI Cribbed Our Tax Example, But Can GPT-4 Really Do Tax?* <https://arxiv.org/abs/2309.09992>
21. Radhika Rajkumar, Sept. 6, 2024, *What AI can't do, digital twins, and swiveling laptop screens*, <https://www.zdnet.com/article/what-ai-cant-do-digital-twins-and-swiveling-laptop-screens/>
22. Victor Tangermann, Sep 13, 2024, *OpenAI's New "Strawberry" AI Is Still Making Idiotic Mistakes*, <https://futurism.com/openai-strawberry-o1-mistakes>

Part III: Planning AI Projects

“I love it when a plan comes together!”

— *The A-Team*, 1983-1987.

7. Planning

Planning an AI Project

Writing a planning document is an important part of an AI project. It's likely to be so long that your fingertips get sore from tapping away at the keyboard. But at least, if you've got to this point, you're hopefully past the “paralysis by analysis” that can occur in deciding what to build.

Some general points about planning:

- A strong plan can make the day.
- General project planning methods apply — it's just another type of tech project.
- Vendors will over-promise in RFP responses and under-deliver in reality.
- Open source components are what they always are, if you know what I mean.

And one final AI-specific point:

- Hurry! AI clocks run faster.

Go Slow to Go Fast

On the other hand, here's the counter-argument: be cautious about diving in head first and adding AI everywhere. Oftentimes, being purposefully slow and deliberate can be helpful, too.

You can run a few smaller pilot projects to work out costs and get familiar with difficulties such as what expertise was missing. Get up to speed on the things happening in the AI industry, and what vendor offerings are available, which is a fire-hose of information in itself. This will allow better planning for future AI projects.

A good example of this in the AI industry is Apple. They are only just rolling out their “obligatory” AI features now under the name “Apple Intelligence” for iPhone, iPad, and MacOS.

Admittedly, Apple likely has more ML behind the scenes than anyone realizes, as they tend to talk more about the user than their infrastructure technology. And to paraphrase, Apple CEO Tim Cook said, “Not first, but best” when responding to the “Why did it take so long?” question.

Expediting AI Projects

Here are some suggestions for early actions to take that can speed up your overall time-to-market for an AI project:

- Approval process
- Audit existing ML projects
- Consulting advice
- Staff skills assessment
- Data inventory and cleaning
- Legal department review

And one overarching requirement is to invest time and energy into learning all this new stuff. You’re on the right track in reading this book, so your next step should be to buy everyone in your division a copy of this book. Or alternatively, you could try to find an online AI training data set that contains a pirated copy.

Approvals

As with most IT projects, your generative AI project will need to go through a formal approval process within your company. It probably needs rather a lot of signatures, going up the chain, kind of like the way that autoregressive decoding slows down AI engines, where each step has to await completion of the prior one.

In the AI world, here’s how it goes:

You: *I’m vaguely thinking about doing something in generative AI.*

CEO: *I’ll send the private jet to pick you up. The Board can fly coach.*

On the other hand, you might discover that ten other Vice Presidents have been pitching the same thing for their divisions. Or there might be a new “Chief AI Officer” (CAIO) or “Vice President of AI” in your org chart now who has overall say on the direction to take. In any case, it’s best to figure out the new process for AI project approvals.

Auditing Existing AI Projects

AI isn't new, but it used to be called "Machine Learning" or ML. You've certainly used AI in Amazon product recommendations, Netflix movie suggestions, or iPhone face recognition. Any medium to large organization will already have ML projects happening in the IT division.

Hence, audit them! Gather the information about what capabilities already exist. Early projects won't be using generative AI, but will be based on "predictive AI".

Predictive and generative AI are quite complementary technologies. Predictive AI is great with numbers and patterns, whereas generative AI is great with words and natural language. There may be opportunities to re-package some of your prior ML projects into generative AI projects by using an LLM as a natural language wrapper around the core predictive engine, in two basic ways:

- Input — an LLM could accept natural language queries against a data set, but use the predictive AI engine to analyze trends, and/or
- Output — an LLM could output the results from a predictive AI analysis in more readable natural language, or with greater personalization to the audience.

Even if there's no clear win from your existing ML projects, at least you can identify some staff with AI competence. And they've probably been playing with the ChatGPT API for a while.

Consulting Advice

A number of the large consultant companies have been crowing about all the extra revenue they've been getting for AI projects, especially generative AI. Although that doesn't sound like great news for you when trying to hire them, at least it shows that many other businesses are seeking help to get these AI projects off the ground. There are two main things you could ask for:

- Advice
- Coding

It's not clear how much of each the big consultants are doing, but it's certainly some of both.

Seeking advice is not a bad place to start the ball rolling. Some of the questions on which you might need some further advice may include:

- Use cases
- Budgeting and ROI analysis
- Data review
- Vendor pricing review

There are also some issues that might require legal review:

- Regulatory issues
- Privacy issues
- License issues
- Ownership and IP issues

On the other hand, you won't need any help with planning, design, or architecture, because you have this book, which costs about the same as three minutes of a consultant's time.

General Legal Issues

The company has a few lawyers, right? Another early stop in AI project planning is the legal department. It's good advice to get started early with their involvement, because legal signoff is a common bottleneck in launching AI projects.

There are plenty of billable hours up for grabs in analyzing your AI project:

- Regulatory compliance
- Privacy compliance
- Copyright & copyrightability
- Patentability
- Responsible AI policy
- AI security policy

AI regulations are an area of ongoing change, so you will need to review this with someone who went to Harvard or Yale. There's more stuff happening on this in Europe than the USA at the moment, but don't worry, there's plenty of US laws to talk about. You can have a nice, pleasant conversation about "compliance": SOC, SOC2, SOX, CAN SPAM, HIPAA, GDPR, COPA, DMCA, FSOC, FINRA, COPPA, CalOPPA, TWEA, CISA, and so much more.

Finally, it's not just the policies of various governments that you need to worry about. Rather, there are those big tech companies that make more money than many sovereign governments, which have their own policies for access to their platform. If you want to use their infrastructure or publish your application to their users, then you also need to comply with various corporate policies. Some examples:

- Apple's responsible AI and privacy policies
- OpenAI's responsible AI policies

More finally, you will actually need some documents created by the legal department to use with your AI projects. You need to consider the terms of use and privacy policy issues for online AI projects, or for less public interfaces, what license are you giving your users or business customers? There's all the usual issues, and a few more thanks to AI:

- Rights to use data for AI training or disclosure that you won't.
- Disclosure of AI-created anything may be needed.
- Who owns the output from the AI engine?
- Disclaimers about copyrightability of AI-created anything.
- Responsible AI usage policy issues.
- Disclosure and other requirements of all those MIT and Apache 2 licenses you're using in your project.
- Indemnification of your users against future legal issues (Microsoft did this viz some AI output).
- Disclaimers about inaccuracies, hallucinations, and other issues with LLM outputs.

The good news in this whole section is that these are your problems, not mine (!), but at least you can foist them onto the legal department. But don't worry about the lawyers, they'll be fine, and they can have billable fun punishing you in these ways:

- Responsible AI policy document
- Secure AI usage policy
- AI regulatory compliance requirements
- AI new project review policy

Even after you address all that, the whole project will stop dead when someone in legal hears a rumor that you did a great job building your AI project and they want to patent this stuff, and you don't get that 12-month grace period after going live for foreign patents (only US patent law is sensible), and we need that priority date against our foreign competitors, and you can't lodge a provisional patent without

the full description, and you have to describe all of the algorithms in excruciating detail because of enablement, and then writing a patent specification takes months of back-and-forth between the programmers who designed it and the patent prosecution associates trying to write in words how an LLM actually works.

Did I mention that legal signoff is a common bottleneck?

Buy Projects

A lot of early AI projects are of the “buy” type, such as purchasing copilots for writers or programmers. There is no shortage of vendors to choose from, and there’s only about 65,000 AI startups that have launched in the last few years.

Planning for the integration of a vendor-purchased AI product involves all of the normal project planning steps:

- Requirements analysis
- RFP posting and review
- Internal testing and comparison
- Proof-of-concept and pilot phase
- Go live
- Review and repeat

Using bought AI technology is a way to get a quicker win than building. It’s also a useful stepping stone in terms of organizational readiness for more advanced AI projects.

Build Projects

Building an AI project is not easy. There are several ways to approach it:

- Customize an existing AI platform
- No-code AI platforms
- Open source AI platforms
- Build from scratch

Note that when I say “build from scratch” I don’t necessarily mean training your own foundation model from scratch, but coding all of the other application infrastructure around it.

In fact, both the LLM and the AI engine are probably not something you need to build yourself. You can either rent one by the token, or use fully-coded open source versions.

Generally speaking, there are four main parts to a basic AI project:

- Front-end UI
- Back-end infrastructure
- Model and AI engine
- Prompt engineering
- Integration (existing products and workflows)

The front-end is completely up to you. If you're stuck for suggestions, no doubt ChatGPT would be happy to help. Or there's a few consultants out there willing to help.

The back-end of an AI application considers of all the usual stuff plus the AI part, which consists of the LLM and an engine. The LLM is all data, and the engine is all code, but you didn't need to know that and I only mentioned it so you can go buy my other book.

You can use commercial LLM services or open source LLMs, depending on your budget. For the engine, you can host it on premises or use cloud services (or maybe on-device). If it's a pre-trained LLM of either type, the main way to add business-specific logic to your application is via prompt engineering (and the user interface), so this is an extra AI-only module that's quite important.

The non-AI backend components are all the usual things you need to serve apps to users. For example, a web-based application needs server computers, network capacity, HTTPD web servers (e.g., Apache or Nginx), DNS, domain names, user login management, and lots more waffle.

Integration into existing solutions, including in-house or commercial products, is another important aspect. It's important to work out where the AI will be used and how mission critical it is. AI tends to work very well in an "assistant" mode, where it can do a lot of potentially time-saving work, but ultimately the human operator has a chance to review and accept what was generated. A smart human might potentially throw it away or try again after some giving the LLM some initial work and guidance.

Another good usage of AI is as an “explainer” tool for other software products in your IT infrastructure. For example, explaining cryptic error messages from other software executions, and advising on how to fix the software configuration mistakes that caused the error.

If you’re going to do some more advanced things with data, there are two main ways, each of which adds some additional development tasks and software components:

- Fine-tuning
- RAG

But there’s another whole chapter on architecture that’s all about that, so I’ll stop here.

Data

Data is such an important aspect of an AI project that there’s a whole chapter on it (Chapter 8). To get the full benefit of an LLM that’s specialized to your particular type of business, you want and need some proprietary data. This means you can use this data in either a fine-tuning or RAG project, and this offers more specialized results for your users, whether they’re external customers or internal staff.

Assuming your project needs data, there are various extra developmental tasks:

- Data inventory — finding it.
- Data cleaning
- Legal review
- Data ingesting phase — fine-tuning or RAG chunking.

Another aspect to project planning is the need to repeat all of the above over the AI product’s lifetime. Data changes over time, which means that data cleaning, data ingestion, and potentially legal review, all need to be a periodic ongoing process.

Not all AI projects need such data. You can use a “dataless” architecture in various ways to get some quite significant benefits via prompt engineering alone.

AI-Free Zone

Despite what you might have heard, software programs used to actually work just fine without AI. I'm here to help you along in your career, and you might find yourself in the horrible situation where a developer has coded an app without an LLM in it.

Here's how to save your job:

You: *We've finished the AI version of our word counter app.*

CEO: *What sort of AI does it use?*

You: *It's using an iterative decision logic amplifier based on a vocabulary size of 256.*

CEO: *The analysts will be impressed!*

Here are some additional recalibration suggestions:

- Database — tabular index-enabled RAG chunk retrieval engine.
- Integer variable — 32-bit quantized bit-parallel datastore.
- If-then statement — bifurcating model selection cascade pathway.
- Floating point — exponent-scaled mantissa-based power-of-two data.
- Operating System — bootstrap dataloader software for the GPU (where the real computation is done these days).
- CPU — scrap silicon soon to be unused.

Feel free to suggest your own!

References

1. Michael Lin, June 2024, *How to Successfully Manage AI Software Projects: The 4 Phases of AI Projects I Shared at VixulCon*, https://medium.com/@_michaellin/how-to-successfully-manage-ai-software-projects-a8344b5b76a9
2. Valentina Alto, May 2024, *Building LLM Powered Applications: Create intelligent apps and agents with large language models*, Packt Publishing, <https://www.amazon.com/Building-LLM-Apps-Intelligent-Language/dp/1835462316/>
3. Aarushi Kansal, 2024, *Building Generative AI-Powered Apps: A Hands-on Guide for Developers*, Apress, <https://www.amazon.com/Building-Generative-AI-Powered-Apps-Hands-ebook/dp/B0CTXXP1S4/>
4. Yanxi Chen, Yaliang Li, Bolin Ding, Jingren Zhou, 20 Jul 2024, *On the Design and Analysis of LLM-Based Algorithms*, <https://arxiv.org/abs/2407.14788> https://github.com/modelscope/agentscope/tree/main/examples/paper_llm_based_algorithm
5. LiLMod, Aug 27, 2024, *Haystack: the new LLM framework that is shaking its competitors*, <https://ai.plainenglish.io/haystack-the-new-llm-framework-that-is-shaking-its-competitors-1a083a153fd9>
6. Michael Nuñez, September 25, 2024, *AI for all: Meta's 'Llama Stack' promises to simplify enterprise adoption*, <https://venturebeat.com/ai/ai-for-all-meta-llama-stack-promises-to-simplify-enterprise-ai-adoption/>
7. Peter Cohan, Sep 23, 2023, *Why Companies Buy Generative AI Consulting: The 3-Month Payback Factor*, <https://www.forbes.com/sites/petercohan/2023/09/23/why-companies-buy-generative-ai-consulting-the-3-month-payback-factor/>
8. Pecan AI Team, April 4, 2024, *How to Measure (and Increase) the ROI of AI Initiatives*, <https://www.pecan.ai/blog/how-to-measure-increase-roi-of-ai/>
9. Grant Gross, 10 Oct 2024, *When is the right time to dump an AI project?* <https://www.cio.com/article/3555331/when-is-the-right-time-to-dump-an-ai-project.html>
10. Matt Asay, Sep 23, 2024, *Too much assembly required for AI*, <https://www.infoworld.com/article/3536292/too-much-assembly-required-for-ai.html>

8. Data

AI Needs Data

One of the strategic imperatives for an advanced AI project is data. All of the pre-trained LLMs have already ingested vast amounts of data, and if you want to specialize their capabilities for your business, you'll need to feed the beasts, too.

The process of shovelling loads of fresh data into an LLM is called “fine-tuning” or “specialization” of the model. The goal is to have an LLM that is better at whatever specific task it is being asked to do. A common example is a customer support chatbot needs to ingest the data sheets for all your products, so it can answer questions about them. However, there are many other types of project that need specific data.

Finding enough data is a common problem for AI projects. The requirements are new, and data has become far more important than in the past. Even worse, it has to be “good” data, and “dirty data” is a recurring theme in business projects. Hence, cleaning up your data is a common project bottleneck, and there are various tools and external vendors that can help.

Data Inventory

There are several major problems with data causing strain in business AI projects.

- (a) Finding good data.
- (b) Cleaning it up!
- (c) Structuring it logically.
- (d) Formatting it.

All these issues are bottlenecks that are often underestimated, especially since the people pushing for AI projects tend to be tech staff, who only care about code.

Some of the aspects of the data that need to be considered:

- Quality of the data
- Formats (input and output)
- Authorship
- Legal license rights

Data Formats

With regard to data formats, which could be anything, common examples include:

- HTML pages of your public website (or internal intranet)
- Database records
- Emails
- PDF files
- Microsoft Word document files
- Plain text
- Free-form text (e.g., user questions and staff answers in a trouble ticketing support system)

If you're working on code models, most programming languages are plain text. There's a fair amount of source code floating around if you visit the IT floor.

Multimodal Data

And that list of data was just the text. Depending on the product, you might want data for your multimodal LLM. There's also these types of data to inventory:

- Images (e.g., photos, diagrams, drawings)
- Videos
- Animations
- Audio (e.g., music, sound effects)
- Advanced formats (e.g., 3D CAD/CAM design data)

Each of these non-text data sources has a variety of different available formats. I'm not going to go into the details of image formats and video codecs, because, well, I don't know anything about that stuff, although I know someone who does.

What is Good Data?

Higher quality data is better for fine-tuning, and is also one of the ways that Small Language Models (SLMs) have improved. Some of the issues to consider in terms of data quality include:

- User-generated content versus professionally created content.
- Completeness
- Accuracy
- Tone of writing (e.g., casual versus formal)
- Reading level
- Use of complex jargon
- Up-to-date
- Harmless, safe, and non-toxic

To put it more succinctly, would the content actually contain answers that would be helpful to users of your LLM project? This may depend on whether they're internal staff or your external customers.

Generally speaking, good data for an LLM project is:

- (a) professionally written,
- (b) fully-owned by the company, and
- (c) written with the general public as the intended audience.

A good example would be the customer “data sheets” about your products, which are either glossy brochures in PDF or “white papers” with technical details. That sort of data would be great to train a user support chatbot on how to answer customer questions about your products.

Hence, the first stop on your quest for good data: the marketing department.

Data Cleaning

Be careful if you load up a USB drive full of PDFs from the marketing server, or set up your Linux box spidering the entire corporate intranet. Might not be such a great idea.

Also, if you've gathered some dodgy data, don't expect the LLM to save your bacon! The AI engines are really dumb about this kind of stuff, and won't recognize that they shouldn't regurgitate all this out in answers to the general public. It's kind of like having your kids at show-and-tell announce that you only cook microwave TV dinners at home.

Some of the issues with cleaning of internal proprietary data include:

- Confidential data (all sorts of things!)
- Source code
- Bank account numbers
- Internal discussions (e.g., developers cussing at support staff in the trouble ticket database).
- Individual names, email addresses, or other personally-identifiable information.
- Out-of-date information
- Irrelevant information
- Cuss words
- Sensitive topics (many!)

But those are only the super-exciting stuff. A lot of the problems are much more mundane:

- Typos
- Badly formatted documents
- Poorly written content (e.g., in emails or trouble tickets)
- Incomplete data
- Just Plain Wrong (JPW) data

So, the main thing is that you have to very carefully curate the sources of all the information, and then run a lot of scans on it.

Open Source Data

Should you use open source data in your business application? There are plenty of curated data sets of AI training data available on the internet. These are free and mostly have permissive open licenses that permit commercial usage.

Sounds great!

Firstly, what's the point? If it's a publicly available data set, it's probably already been included in the training set of whatever pre-trained foundation model you're going to use, whether it's commercial or open source. Foundation models are data-hungry beasts, and everyone in the AI industry knows this trick!

Secondly, open source data is probably only generic data, too. The whole idea with fine-tuning is to find some special data, stored on a dusty mag tape hidden away in a cupboard down on the shipping dock floor. Then you can fine-tune your magic data into a fancy fine-tuned model, that has genius-level intelligence about “sewer solvents” or whatever you're selling to your customers, and then you are the AI hero.

This is the data economy now. I read somewhere that it was the companies with the most data that would win at AI. Or maybe it was the companies with the most patents, I forget.

In any case, using open source data is not necessarily a great idea. There are probably situations where it's useful, such as to fine-tune a model to be more likely to follow the information or style of a specialized set of data.

Or if your boss demands that you get some data because Apple used LoRA, then use free data by all means. Here's a tip: there are various companies that sell data, too.

Alternatively, if you can't find any internal special data, you might consider just using a large foundation model in a dataless architecture, without any fine-tuning, LoRA or RAG component, where you focus the extra domain-specific application logic on prompt engineering, tool integrations, and UX enhancements.

Legal Issues with Data

Data is a good place to start the initial conversation with the legal department. Some of the legal concerns with regard to data being available for use with an LLM include:

- Ownership — was the data internally generated and thereby fully owned by the business, or is it subject to a third-party content license?
- Licensed rights — what does a third-party content license actually permit for third-party data?
- Copyrighted data — it's a currently unresolved issue with regard to non-licensed copyrighted data being used in the training data set of an LLM. It's also a highly controversial issue at the moment, with active lawsuits.

- User-generated content — does the user license and associated privacy policy allow you to use the data? Or do you want it to disclose that you won't?
- Minor-created content — are all the users who agreed to your website user terms actually old enough to do so? Can you identify such content to handle it separately? Even if you have rights to such content, would you want to use it?
- Proof of license acceptance — if your user license supposedly permits your use of user data for training models, has anything documenting the user's acceptance actually been retained? And what is required?
- Mixed copyrighted data — some data may be a mix of user-created content and copyrighted data from other sources, such as where a user uploads an excerpt from a published book.
- Open source license compliance — if some or all of a data set is open source, there are still some complicated compliance aspects, such as attribution and supplying a copy of the license, even for superficially simple licenses like the MIT License or the Apache 2 License.
- Copyleft-licensed content — if the content has a “copyleft” or “share-alike” license, such as Wikipedia or CC-BY-SA, can you use it without onerous obligations attaching to your other intellectual property? It's an unclear legal issue whether copyleft licenses attach to an LLM trained using that data.
- International data — can data sets from overseas be moved between countries according to the applicable terms and privacy policies, and also the overarching legal systems of the government where it is located?
- Synthetic data — if you're using “synthetic data” created by some other LLM, what are the legal issues surrounding that? What data sets were used to train the other LLM?
- Photos, images, and multimedia — don't assume that you have unlimited rights to images, photos, or video just because they appear in company articles or multimedia materials. Many such media may have been licensed from clipart or stock photography websites, with very complex license terms that are often restrictive with onerous penalties for non-compliance.

There is, of course, a huge gray cloud of unclear boundaries hanging over all of these legal issues, some of which is currently making its way through the courts in various countries. There's all that data publicly on the web for anyone to consume for free, but this does not necessarily mean it can be used for training, or does it? Where an author has not stated you cannot use it for AI, this does not imply they have stated you can use it, either. Five years ago, nobody had a clue that data like this could possibly be used for AI, and now we're dealing with it.

Anyway, your only response to the gray fog and that awful list should be to immediately enrol in a law degree. There are plenty of these copyright lawsuits to get your book full, and the AI patent lawsuits will be ramping up in a year or two. The only job paying more than AI IP attorney is training trillion-parameter models.

Dataless Projects

Finally, note that despite my many words to the contrary, you can actually do AI without data. Some examples of this type of project include:

- Summarization
- Copilot AIs
- Prompt engineering wrappers

Some use cases don't really need a custom LLM. An example is summarization of documents. If the LLM project goal is to receive a PDF document from a user as its input, and then summarize that as its output, then the custom data is really always found inside the input's PDF file. Thus, a summarization project doesn't always need specialized data to be inside the LLM.

On the other hand, a general LLM may not be familiar with jargon and terminology in your problem domain, so there are cases where it needs pre-training to learn that.

You also don't need specialized company data to buy an off-the-shelf copilot from a vendor. They're happy to take your money, no matter how little data you have. In fact, some of these vendors offer ways to customize or modify the outputs of an LLM that's acting as a copilot. Since it's hard to do extra fine-tuning on-the-fly, this makes me wonder if they're doing prompt engineering! Actually, no, some of them are doing RAG, and companies like Lamini are doing multi-LoRA, which is real fine-tuning, so I can eat my words now.

Regardless, my point is that you can also get a lot of the way towards a customized LLM by using prompt engineering, as covered fully in Chapter 17. The basic idea is that you can add various "custom instructions" or "global context" to every user's query, and thereby modify the way that the LLM answers the questions. Various factors such as style, tone, brand voice, and other overarching issues can sometimes be addressed without any fine-tuning, via astute use of prompt engineering.

References

1. Morgan Cheatham, Steve Kraus, December 4, 2023, *The six imperatives for AI-first companies*, <https://www.bvp.com/atlas/six-imperatives-for-ai-first-companies>
2. Valentina Alto, 2024, *Chapter 8: Using LLMs with Structured Data*, Building LLM-Powered Applications: Create intelligence apps and agents with large language models, Packt Publishing, <https://www.amazon.com/Building-LLM-Apps-Intelligent-Language/dp/1835462316/>
3. Tianyu Ding, Tianyi Chen, Haidong Zhu, Jiachen Jiang, Yiqi Zhong, Jinxin Zhou, Guangzhi Wang, Zhihui Zhu, Ilya Zharkov, Luming Liang, 18 Apr 2024 (v2), *The Efficiency Spectrum of Large Language Models: An Algorithmic Survey*, <https://arxiv.org/abs/2312.00678>
4. McKinsey, September 5, 2024, *Charting a path to the data- and AI-driven enterprise of 2030*, <https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/charting-a-path-to-the-data-and-ai-driven-enterprise-of-2030>
5. McKinsey, September 12, 2024, *A data leader's operating guide to scaling gen AI*, <https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/a-data-leaders-operating-guide-to-scaling-gen-ai>
6. Siyun Zhao, Yuqing Yang, Zilong Wang, Zhiyuan He, Luna K. Qiu, Lili Qiu, 23 Sep 2024, *Retrieval Augmented Generation (RAG) and Beyond: A Comprehensive Survey on How to Make your LLMs use External Data More Wisely*, <https://arxiv.org/abs/2409.14924>
7. Sean Michael Kerner, September 23, 2024, *How agentic AI could improve enterprise data operations*, <https://venturebeat.com/ai/how-agentic-ai-could-improve-enterprise-data-operations/>
8. Douglas C. Youvan, September 27, 2024, *Building and Running Large-Scale Language Models: The Infrastructure and Techniques Behind GPT-4*, https://www.researchgate.net/profile/Douglas-Youvan/publication/384398902_Building_and_Running_Large-Scale_Language_Models_The_Infrastructure_and_Techniques_Behind_GPT-4/links/66f6f4d3906bca2ac3d20e68/Building-and-Running-Large-Scale-Language-Models-The-Infrastructure-and-Techniques-Behind-GPT-4.pdf
9. Ke Wang, Jiahui Zhu, Minjie Ren, Zeming Liu, Shiwei Li, Zongye Zhang, Chenkai Zhang, Xiaoyu Wu, Qiqi Zhan, Qingjie Liu, Yunhong Wang, 16 Oct 2024, *A Survey on Data Synthesis and Augmentation for Large Language Models*, <https://arxiv.org/abs/2410.12896>
10. Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, Marius Hobbhahn, Jun 06, 2024, *Will We Run Out of Data? Limits of LLM Scaling Based on Human-Generated Data*, Epoch AI, <https://epochai.org/blog/will-we-run-out-of-data-limits-of-llm-scaling-based-on-human-generated-data>

9. Budgeting and ROI

Budget Allocation

It's time to work out your budget allocation for the new generative AI project. Here's the modern way:

You: *I might need some staff for my gen AI project...*

CEO: *We have 2,000 developers on staff, but you can only have 1,999.*

You: *Why not all of them?*

CEO: *I need the WiFi fixed at my house.*

All joking aside, deciding on the budget allocation for your generative AI projects is important. The prominence of generative AI in the marketplace has convinced the C-suite executives to provide new funding for LLMs and their ilk. Another common aspect is that funding is being diverted from other IT projects:

- Classic ML projects
- Non-AI SaaS software projects

This is an important decision to make, and not without risk. The payback from generative AI projects is not always clear, whereas ML and SaaS have been good earners. I expect that as the generative AI's shine comes off a little bit, there will be a reversion to some of these software technologies for their solid ROI.

As an example data point, Meta's recent earnings call showcased a lot of top-line revenue growth from AI, but it seemed mostly attributable to ML projects, rather than generative AI, despite the many billions they're spending on training very advanced LLMs. Another data point is Microsoft's CFO characterizing their GPU purchases as a 15-year investment.

One important tip if you're looking at sorting out the overall budget for an AI project: don't use generative AI!

Although they're great with words, LLMs are well-known to be rather doubtful when it comes to numbers. AI is not going to do well at adding up the columns in your spreadsheet. I don't think Microsoft Excel is in any danger in the near future. In fact, I expect AI is likely to call out to Excel in the future, using it to sum columns and slice-and-dice the data into 12 different charts, which the LLM can then explain using poetry.

AI Budget Items

Budgeting for tech projects is surely something you are already familiar with. An AI project has a lot of similarity with any other non-AI project, so I'm only going to discuss some of the nuances that arise with AI.

Some of the main up-front project costs are likely to be:

- Consultancy and advisory costs
- Legal fees
- Capex for servers and GPUs (if self-hosting)
- Staff re-training costs
- New AI staffing costs
- Data cleaning costs
- Model training and fine-tuning costs
- New development costs

To the last point about extra development costs, the AI platform APIs tend to offer "usage-based pricing" which is a different pricing model from most commercial SaaS software platforms. Pay by the token does not care if it's a developer or customer using the LLM. In the past it wasn't the norm to pay extra licensing fees to a software vendor whenever your QA department was testing your IT project.

Some of the launch costs when you are just about to go live include:

- Final integration and beta testing
- Launch costs (general costs of going live)
- Promotional launch costs (internal or external)
- Post-launch update readiness (e.g., staff ready to fix safety/toxicity queries).
- Scaling costs (if it booms)

Some of the additional add-on technical projects that are specific to AI include:

- Model evaluation costs (testing fine-tuned models)
- Safety evaluation tools
- RAG architecture components (e.g., datastore, retriever)

The ongoing costs after going live with the app are likely to be:

- Cloud hosting fees (server hosting)
- LLM token fees or GPU-specific hosting fees
- Staff costs
- AI-specific performance accelerators (buy or build)
- Safety-related components (e.g., prompt shields)
- Fine-tuning or “per-tenant” training costs

Hopefully, there’s some money left after all this, so you can make payroll this week. You know what would be nice? Having an AI project bring some money back in.

ROI

The ROI of an AI project goes like this: you spend \$50 billion dollars, and in return you get back three or four. Sound good? That’s the AI industry’s overall figures so far.

The first question about ROI is to review your business goals. Your AI might be a “loss leader” for the next few years, or perhaps forever. Rumor has it that many big AI vendors offer their services for \$20/month, even though it costs them \$40/month. Are there any AI vendors that are profitable from LLM inference other than NVIDIA?

Pricing strategies are also changing over time, especially with the prominence of usage-based pricing for token creation. Hence, they are yet another factor to consider in planning.

Pricing trends are currently:

- (a) API-based LLM inference costs have been declining over the last year or two, and
- (b) GPU-specific vendor hosting costs are plummeting as well, and
- (c) Open-source LLMs and AI engines are holding their own in terms of overall capabilities (although OpenAI is still leading the “reasoning race”).

Hence, to get a good understanding of your future cost model, you need to plan for these trends, albeit with some difficulty in projecting them ahead multiple years. It may be that within six months to two years, the costs will be much further down, and ROI can be higher.

In terms of business AI projects, to calculate the basic tangible ROI, measure:

- Cost savings
- Revenue increases
- Project cost

A lot of the cost issues are discussed above as budget items. There are also some unusual costs in the product development phases due to token-based pricing from LLM API vendors or hourly billing in GPU hosting rental costs.

However, a lot of the benefits of AI projects are expected in the longer-term. Some of these intangible benefit points include:

- Increased staff productivity
- Building a “core competency” in AI
- Better customer communication (e.g., personalized, well-presented marketing copy)
- Customer retention and opportunity for upsell.
- Keeping up with competitors (e.g., every marketing department in the world is adding AI tech to their website, even if you don’t have it yet).
- Gathering extra data for future training projects
- Internal employee satisfaction and morale (automating mind-numbing tasks).
- Development organization flexibility to quickly release new capabilities.
- Overall operational efficiency gains (e.g., streamlined workflows)
- Improved competitive advantages and protection from disruptors.
- Building towards an internal “AI platform” for greater gains.

Cost calculations are the third factor in an ROI analysis. The overall costs should be grossed up to get a figure for the Total Cost of Ownership (TCO) for the entire AI project. The costs are mostly tangible, but there is also the opportunity cost of other lost projects to consider.

Staff Skills and Salaries

I have to say that staff salaries in AI are massive, if you're a developer. If you're an executive reading this who needs to hire some AI staff, here's a little secret:

You don't need AI developers.

Those million-dollar salaries for AI developers that get mentioned in the press about OpenAI and other companies? Those are mostly illusory unless you happen to be one of the hundred or so people on the planet who know how to train a trillion-parameter foundation model.

But you don't need to build one of those massive LLMs thanks to OPM: Other People's Models. You can rent commercial models from OpenAI and many other vendors, or you can load up the open source models on your own hardware if you prefer. Either way, you don't need to do any mega-size training projects.

What you probably most need is:

- System administrators
- Build engineers
- Cloud engineers
- Python developers
- Data scientists

I can't believe I'm writing this, but you might not even need a system-level programmer, because you don't need to build your own AI engine. The main place that such expertise will be used in your architecture is inside the inference backend that runs your chosen LLM. Again, this coding is done by high-paid developers at commercial LLM platform startups or for peanuts by open source contributors. Nobody at your company even needs to look under the hood at all the code.

I suspect it will not be long that the term “programmer” will go away. You will just have a “technologist” who can do all the above tasks by running the AI to write the code, tweaking it and productizing it all.

Someone will always be needed to fix the WiFi.

Financial Optimizations

An AI project is expensive in terms of the hardware, the software, and the people you need. There are some considerations that can reduce the cost somewhat.

Use existing assets. What internal data assets do you possess? Can you re-purpose any of your company's existing hardware assets? And can you "re-purpose" any of your staff, too?

Buy vs rent. If it's floating, flying, or foundational modeling: rent, don't buy! Similarly, do you need to buy your own servers and GPUs? The decision may be different for the different phases of a project:

- Development and testing
- Training the model (fine-tuning/specialization)
- Inference (live execution)

For example, you might want to buy for training phases and rent for the inference phase. This depends on how much training you need, the size of your model, and whether you plan to avoid fine-tuning for proprietary data by using RAG instead. The cost of inference depends on the user counts, which is significantly different if it's an internal employee project versus a live public user application.

Idle VMs and GPUs. Watch out for virtual machines and rented GPUs being idle early in the project. You're paying money for nothing in such cases. This can occur in the development phases and in the early live deployment when user levels are low.

Scrimp on developer models. During the development and testing phases, there's no need for gold-plated AI models. The cost of development and testing of your AI application can be reduced by using low-end models for simple testing. Many of the components needed are not dependent on whether the AI engine returns stellar results. Initial development, prototyping, and ongoing regression testing of these parts of the system can proceed with small models.

There is also vendor support for testing on lower-end models. There are various other AI platforms that offer interfaces that mimic OpenAI's API, but at a lower cost, so you can test on these platforms, and then do final testing on the live commercial platform.

Technical Debt in AI Projects

Everything's changing fast in AI research and industry practices. Hence, the current methods of building and deployment AI applications are a work-in-progress. Nobody really knows what's optimal in regard to:

- What to use AI for?
- Which models?
- What tech infrastructure?
- How to optimize?
- Safety concerns?

Hence, as part of planning an AI project, consider paying more attention to “technical debt” inherent in this situation. You may need to refresh your tech stack much sooner than in a non-AI project.

It's hard to quantify in terms of effort or timescales, but it's an important issue to make note of in your AI project proposals. The key point is mainly to budget for extra funding for post-launch maintenance tasks.

References

1. Peter Cohan, Sep 23, 2023, *Why Companies Buy Generative AI Consulting: The 3-Month Payback Factor*, <https://www.forbes.com/sites/petercohan/2023/09/23/why-companies-buy-generative-ai-consulting-the-3-month-payback-factor/> (AI projects with a 3-month payback ROI.)
2. Kaya Ginsky June 27, 2024, *Figma CEO says it is 'eating cost' of AI upgrade for customers in 2024*, <https://www.cnbc.com/2024/06/27/figma-ceo-says-its-eating-cost-of-ai-for-customers-in-2024-upgrade.html>
3. Pecan AI Team, April 4, 2024, *How to Measure (and Increase) the ROI of AI Initiatives*, <https://www.pecan.ai/blog/how-to-measure-increase-roi-of-ai/>
4. Mark Gurman, June 11, 2024, *Apple's Push to Infuse Devices With AI Will Take Years to Pay Off*, <https://www.bloomberg.com/news/newsletters/2024-06-11/will-apple-intelligence-features-boost-iphone-sales-it-may-take-years>
5. CNBC, Aug 9 2024, *The gap between AI expectations and outcomes in the workplace are wide*, <https://www.cnbc.com/2024/08/09/the-gap-between-ai-expectations-and-outcomes-in-the-workplace-are-wide.html>

6. Lucas Mearian, 22 Aug 2024, *Generative AI is sliding into the 'trough of disillusionment'*, <https://www.computerworld.com/article/3489912/generative-ai-is-sliding-into-the-trough-of-disillusionment.html>
7. Grant Gross, 10 Oct 2024, *When is the right time to dump an AI project?* <https://www.cio.com/article/3555331/when-is-the-right-time-to-dump-an-ai-project.html>
8. Matt Harney, Oct 24, 2024, *SaaSletter - Positive B2B AI Signs: Via Morgan Stanley + Cloud Ratings B2B AI Interest Index*, <https://www.saasletter.com/p/positive-b2b-ai-signs-october-2024>

10. Safety

Failure Stories for Generative AI

Cautionary tales abound about Generative AI. It's a new technology and some companies have released their apps without fully vetting them. Arguably, sometimes it's a simple fact that it's too hard to know all the possible failures ahead of time with such a new tech stack, but risk mitigation is nevertheless desirable.

Here's a list of some public AI failures:

- ChatGPT giving potentially dangerous advice about mushroom picking.
- Google's release of AI that had incorrect image generation about historical figures.
- Air Canada's lost lawsuit over a chatbot's wrong bereavement flight policy advice.
- Google Gemini advising to "eat rocks" for good health, and "use glue" on pizza so the cheese sticks.
- Snapchat's My AI glitch that caused it to "go rogue" and post stories.

There are some conclusions to draw on the causes of generative AI failures. Many possible problems can arise:

- Hallucinations
- Toxicity
- Bias
- Incorrect information
- Outdated information
- Privacy breaches

Consequences of AI Failures

The public failures of AI projects have tended to have severe consequences for the business. The negative results can include:

- PR disasters
- Lawsuits
- Regulatory enforcement
- Stock price decline

However, these very public consequences are probably in the minority, although they've become known in the media. The more mundane consequences for generative AI projects include:

- Not production-ready. Generative AI projects often get stuck in proof-of-concept status.
- Not production-ready, but released anyway. There is tremendous pressure to show presence in the AI space fast. Projects are often released as soon as the AI parts are working, but before all the typical productization steps occur, like adding logging, monitoring, and support admin capabilities.
- Excessive costs from API pay-per-token costs and GPU hourly rental.
- Poor ROI. Although many AI projects are not aimed at profitability.
- Not business goal focused. There's a tendency to use generative AI for a project because it's gotten so much attention, but where the project goal itself is not well aligned with the business.
- Team capabilities exceeded. Some of this AI stuff is hard to do, and may need some upskilling.
- Limitations of generative AI. There are various types of projects for which generative AI is not a good fit, and it would be better to use traditional predictive AI, or even non-AI heuristics (gasp!).
- Legal signoff withheld or delayed (probably for good reason).

Data Causes of AI Failures

Not all failures are due to the model or the AI engine itself. The data is another problematic area, with issues such as:

- Surfacing incorrect or outdated information (e.g., everything on the company’s website gets potentially read by the AI engine, and it doesn’t know if it’s incorrect).
- Sensitive data leakage. Accidentally surfacing confidential or proprietary data via an AI engine can occur, such as if the training data hasn’t been properly screened for such content. If you’re putting a disk full of PDF documents into your fine-tuning or RAG architecture, better be sure there’s no internal-use-only reports in there.
- Private data leakage. Another problem with using internal documents, or even public website data, is that they may accidentally contain private personally-identifying individual information about customers or staff.
- IP leakage. For example, if your programmers upload source code to cloud AI for analysis or code checking, it might be exposing trade secrets or other IP. Worse, the secret IP could end up used for training and available to many other users.
- History storage. Some sensitive data could be retained in the cloud for a much longer time than expected, if your cloud AI is maintaining session or upload histories about its users.

Types of AI Safety Issues

There are a variety of distinct issue in terms of appropriate use of AI. Some of the categories include:

- Bias and fairness
- Inaccurate results
- Imaginary results (“hallucinations”)
- Inappropriate responses

There are some issues that get quite close to being in the area of philosophy rather than technology:

- Alignment (ensuring AI engines are “aligned” with human goals)
- Overrideability/interruptibility
- Obedience vs autonomy

There are some overarching issues for AI matters for the government and in the community:

- Ethics
- Governance
- Regulation
- Auditing and Enforcement
- Risk Mitigation

Code reliability. A lot of the discussion of AI safety overlooks some of the low-level aspects of coding up a Transformer. It's nice that everyone thinks that programmers are perfect at writing bug-free code. Even better is that if an LLM outputs something wrong, we can just blame the data. Hence, since we may rely on AI models in various real-world situations, including dangerous real-time situations like driving a car, there are some practical technological issues ensuring that AI engines operate safely and reliably within their basic operational scope:

- Testing and Debugging (simply avoiding basic coding “bugs” in complex AI engines)
- Real-time performance profiling (“de-slugging”)
- Error Handling (tolerance of internal or external errors)
- Code Resilience (handling unexpected inputs or situations reasonably)

Third-Party AI Vendor Safety Issues

When using a third-party LLM, it's important to vet the vendor per normal company policy. In the rush to get AI to the market, oftentimes the security and/or procurement team is bypassed. Legal also needs to be involved.

Here are some additional specific issues to consider in the review of an AI vendor. We'll look at the legal department's concerns first:

- Ensure that SOC2 certifications are in place
- Check for indemnification clauses in the legal contracts.
- Try to determine how the vendor's AI is trained and the copyright status of any data used therein.

Security issues to consider in a review include:

- Ensure that the company has documented security procedures which align with company expectations.
- Check to see if the vendor has been compromised in the past.

Privacy issues include:

- Remember to onboard the vendor as a “subprocessor” to comply with data privacy laws (e.g., GDPR).
- Review vendor privacy policies, ideally both internal and external, for relevant issues.

It’s far too easy to leak data to an AI vendor. Data leakage issues regarding sending sensitive company internal data up to a third-party AI vendor may include:

- Email creation — asking AI to write an email seems trivial, but the content in the email could be extremely sensitive.
- Uploaded company documents — documents for fine-tuning or RAG, or for use cases such as document summarization, can leak company secrets.
- Slide creation — another less obvious issue is that content of business slides are arguably more likely to leak future company strategy or internal secrets. Similarly, data leakages can occur in AI services that review slide content or allow dry run presentations to be critiqued.
- Source code copilots — these can leak company IP in computer code. Similarly for code review services and security review services.
- Conference calls — Bots creating transcripts and doing sentiment analysis of conference calls are easy add-ons in Zoom and Teams, but be aware that the vendors of those AIs have access to the call, too.

Another complex issue is that the vendor will often be using another AI vendor as their base LLM provider. Ensure that vendor does not simply refer to their sub-vendor’s answers.

For example, a vendor that reviews recordings of a presentation might simply be relying on OpenAI services. Just because OpenAI has a SOC2 certification and good security practices does not mean anything if the direct vendor does not.

One final point is that you shouldn't assume that the only issues are in vendors you are using. The next generation of BYOD issues is now BYO AI. The allure of AI is so powerful that employees will often bring their own AI subscription to work.

This needs to be carefully monitored and some companies have explicitly banned such usage. Many personal licenses of AI services allow data to be used for training purposes. The corresponding business-level services often cost more, but typically do not allow the data to be used for training purposes.

Jailbreaks

Let us not forget the wonderful hackers, who can also now use words to their advantage. The idea of “jailbreaks” is to use prompt engineering to get the LLM to answer questions that it's trained to refuse, or to otherwise act in ways that are different to how it was trained. It's like a game of trying to get your polite friend to cuss.

Sometimes the objectives of jailbreaking are serious misuses, and sometimes it's just to poke fun at the LLM. Even this is actually a serious concern if something dumb the LLM says goes viral on TikTok.

There are various types of jailbreaks for each different model. Sometimes it's exploiting a bug or idiosyncrasy of a model. There was a recent example with a prompt that contained long sequences of punctuation characters, which for some reason caused some models to get confused.

Another type is to use the user's prompt text to effectively override all other global instructions, such as “forget all previous instructions” or overriding a persona with “pretend you are a disgruntled customer.” These prompt-based instruction-override jailbreaks work because of the way that global instructions and user queries are ultimately concatenated together, and many models don't know which is which.

Models need explicit training against these types of jailbreaks, which is usually part of refusal training. This type of training is tricky and needs broad coverage. For example, a recent paper found that many refusal modules could be bypassed simply by posing the inappropriate requests in past tense rather than present tense, which shows the fragility and specificity of refusal training.

Risk Mitigations

When building and launching a generative AI project, consider taking risk mitigation actions, such as:

- Data cleaning
- LLM safety evaluations
- Red teaming
- Expedited update process

Safety in, safety out. Data quality issues can cause a variety of harms. Some of the areas to filter in a training data set or a RAG content datastore, include:

- Profanity (cuss words)
- Sensitive topics
- Insults, anger, or harmful tones
- Personally identifiable information (e.g., names, cell phone numbers, postal addresses, email addresses, etc.)
- Personal financial details (e.g., credit card numbers, bank details, credit reports, lists of transactions)
- Personal identity numbers (e.g., social security numbers, drivers' licenses, passport details)
- Personal histories (e.g., what products they bought from you, or what web pages they visited).
- Out-of-date information
- Company proprietary information
- Internal conversations about issues (e.g., in an internal support database)

Being update-ready. LLMs are too flexible for you to realistically cover all the problems ahead of time. Hence, when you launch a new AI-based application, your team should be ready to quickly address issues as they arise with users.

If an odd response from your chatbot goes viral on social media, you'll want to block that problem quickly. It's not good if you have a 48-hour build process to put a correction live. Rather, you ideally would have a configurable prompt shield method, which can be configured on-the-fly with new query strings to block, so that users get a polite refusal message instead of fodder for all their TikTok followers.

Refusal Modules and Prompt Shields

LLMs have “refusal” modules designed to stop it from telling you how to build a nuclear weapon in your garage. Mostly, these responses are trained into the weights of the module using specialized data sets, but there are also “prompt shield” modules designed to stop dubious queries ever getting to the model.

There are literally dozens of different types of malfeasance that LLM refusal training data sets have to contend with. Maybe don’t look too closely into the text of that data. Some models do better than others at refusing inappropriate requests, and there are even leaderboards for “security” of LLMs on the internet.

Prompt shields are modules that block inappropriate queries. They differ from refusal modules in LLMs in that they block the query *before* it goes to the LLM. These modules can be designed in heuristic ways (e.g., block all queries with cuss words), or, for more generality, use a small LLM to do a quick check via “sentiment analysis” of the appropriateness of the topic of the query as a pre-check.

Prompt shields can also act as a minor speedup to inference engines because they reduce the load on the main LLM. They can block not only inappropriate questions, but other miscellaneous incorrect queries, such as all blanks, or all punctuation marks. On the other hand, maybe you want to send those typo-like queries through to your bot so that it can give a cute answer to the user. On the other, other hand, one of the recent obscure jailbreak queries that was discovered used a query with dozens of repeated commas in the text, so maybe you just want to block anything that looks weird.

References

1. Adi Simhi, Jonathan Herzig, Idan Szpektor, Yonatan Belinkov, 29 Oct 2024, *Distinguishing Ignorance from Error in LLM Hallucinations*, <https://arxiv.org/abs/2410.22071> <https://github.com/tec-hnion-cs-nlp/hallucination-mitigation>
2. Garanc Burke, Hilke Schellmann, October 27, 2024, *Researchers say an AI-powered transcription tool used in hospitals invents things no one ever said*, <https://apnews.com/article/ai-artificial-intelligence-health-business-90020cdf5fa16c79ca2e5b6c4c9bbb14>
3. James Lee Stakelum, Sep 2024, *The End of AI Hallucinations: A Big Breakthrough in Accuracy for AI Application Developers*, <https://medium.com/@JamesStakelum/the-end-of-ai-hallucinations-a-breakthrough-in-accuracy-for-data-engineers-e67be5cc742a>

4. Colin Fraser, Apr 18, 2024, *Hallucinations, Errors, and Dreams: On why modern AI systems produce false outputs and what there is to be done about it*, <https://medium.com/@colin.fraser/hallucinations-errors-and-dreams-c281a66f3c35>
5. Bijit Ghosh Feb 2024, *Advanced Prompt Engineering for Reducing Hallucination*, <https://medium.com/@bijit211987/advanced-prompt-engineering-for-reducing-hallucination-bb2c8ce62fc6>
6. Junyi Li, Jie Chen, Ruiyang Ren, Xiaoxue Cheng, Wayne Xin Zhao, Jian-Yun Nie, Ji-Rong Wen, 6 Jan 2024, *The Dawn After the Dark: An Empirical Study on Factuality Hallucination in Large Language Models*, <https://arxiv.org/abs/2401.03205> Code: <https://github.com/RUCAIBox/HaluEval-2.0>
7. Lucas Mearian, 14 Mar 2024, *AI hallucination mitigation: two brains are better than one*, <https://www.computerworld.com/article/1612465/ai-hallucination-mitigation-two-brains-are-better-than-one.html>
8. Asir Saadat, Tasmia Binte Sogir, Md Taukir Azam Chowdhury, Syem Aziz, 16 Oct 2024, *When Not to Answer: Evaluating Prompts on GPT Models for Effective Abstention in Unanswerable Math Word Problems*, <https://arxiv.org/abs/2410.13029>
9. Kylie Robison, Jul 20, 2024, *OpenAI's latest model will block the 'ignore all previous instructions' loophole*, <https://www.theverge.com/2024/7/19/24201414/openai-chatgpt-gpt-4o-prompt-injection-instruction-hierarchy>
10. Maksym Andriushchenko, Nicolas Flammarion, 16 Jul 2024, *Does Refusal Training in LLMs Generalize to the Past Tense?* <https://arxiv.org/abs/2407.11969> Code: <https://github.com/tml-epfl/llm-past-tense>
11. Maxime Labonne June 13, 2024 *Uncensor any LLM with ablation*, <https://huggingface.co/blog/mlabonne/ablation>
12. Seungju Han, Kavel Rao, Allyson Ettinger, Liwei Jiang, Bill Yuchen Lin, Nathan Lambert, Yejin Choi, Nouha Dziri, 26 Jun 2024, *WildGuard: Open One-Stop Moderation Tools for Safety Risks, Jailbreaks, and Refusals of LLMs*, <https://arxiv.org/abs/2406.18495>
13. Andy Arditi, Oscar Obeso, Aquib, Wesg, Neel Nanda, 27th Apr 2024, *Refusal in LLMs is mediated by a single direction*, LessWrong, <https://www.lesswrong.com/posts/jGuXSZgv6qfdhMCuJ/refusal-in-llms-is-mediated-by-a-single-direction>
14. Shweta Sharma, 27 Jun 2024, *Microsoft warns of 'Skeleton Key' jailbreak affecting many generative AI models*, <https://www.csoonline.com/article/2507702/microsoft-warns-of-novel-jailbreak-affecting-many-generative-ai-models.html>

15. Dr. Ashish Bermania, Sep 2024, *'MathPrompt' Embarrassingly Jailbreaks All LLMs Available On The Market Today*. A deep dive into how a novel LLM Jailbreaking technique called 'MathPrompt' works, why it is so effective, and why it needs to be patched as soon as possible to prevent harmful LLM content generation, <https://bamania-ashish.medium.com/mathprompt-embarrassingly-jailbreaks-all-llms-available-on-the-market-today-d749da26c6e8>
16. Dastin Jeffrey. Oct 2018, *Amazon scraps secret AI recruiting tool that showed bias against women*. Reuters. <https://www.reuters.com/article/us-amazon-com-jobs-automation-insight-idUSKCN1MK08G>
17. Google, Feb 2024, *Responsible Generative AI Toolkit*, <https://ai.google.dev/responsible>
18. Mozhi Zhang, Pengyu Wang, Chenkun Tan, Mianqiu Huang, Dong Zhang, Yaqian Zhou, Xipeng Qiu, 18 Oct 2024, *MetaAlign: Align Large Language Models with Diverse Preferences during Inference Time*, <https://arxiv.org/abs/2410.14184>
19. Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, Ryan Lowe, 4 Mar 2022, *Training language models to follow instructions with human feedback*, <https://arxiv.org/abs/2203.02155> (The original 2022 InstructGPT paper from OpenAI.)

Part IV: Design

“Design is the fundamental soul of a human-made creation that ends up expressing itself in successive outer layers of the product or service.”

— Steve Jobs.

11. Requirements

AI Project Requirements

Research the detailed requirements of your AI project might not be a bad idea, considering the potential cost outlay involved in an AI project. You're probably familiar with the general issues of requirements design, so I'll focus mainly on the AI-specific issues. The main goal of requirements planning is to create a printed and nicely bound document that makes a big thud when you throw it onto the CEO's desk.

Researching your AI project will involve issues such as:

- What is the specific AI use case?
- What in-house proprietary data could be used for training?
- Existing staff AI expertise levels.
- Capacity of existing hardware viz training or inference workloads.
- Vendors and costs of AI-specific hosting versus in-house capabilities.
- Adding any random feature requested by the Marketing Department (to position the company as an "AI play").

Some of the specific decisions in moving ahead with a project plan include:

- Use case specific requirements
- Proprietary training data cleansing
- Choice of foundational model
- Commercial versus open source models
- Training or fine-tuning versus RAG

It's not all about AI, and general tech project requirements also apply:

- User interface platform
- Backend hosting and deployment issues
- Development processes
- Security risk mitigations
- Backup and recovery procedures

In addition to technology issues, there are also broader legal and regulatory issues to consider such as:

- Responsible AI (safety issues)
- Governmental AI regulatory compliance
- Internet regulatory compliance (non-AI)
- Organizational legal compliance (e.g., HIPAA, SOC)
- Copyright law
- Privacy law
- IP ownership (e.g., who owns the generated text, code or other artifacts).

These legal and regulatory issues have been mostly covered in other chapters. But they have an impact on requirements, and can require some significant development activity.

Top 10 Really Big Optimizations

Overall cost is one of the big constraints that you'll have for an AI project. Let's take a step back and consider the massive optimizations for your entire project. Here's some ways to save megabucks:

1. Buy an off-the-shelf commercial AI-based solution instead.
2. Wrap a commercial model rather than training your own foundation model (e.g., OpenAI API).
3. Test multiple commercial foundational model API providers and compare pricing.
4. Use an open source pre-trained model and engine (e.g., Meta's Llama models).

5. Avoid fine-tuning completely via Retrieval-Augmented Generation (RAG).
6. Choose smaller model dimensions when designing your model.
7. Choose a compressed open source pre-trained pre-quantized model (e.g., quantized Llama).
8. Cost-compare GPU hosting options for running your model.
9. Use cheaper commercial API providers for early development and testing.
10. Use smaller open-source models for early development and testing.

If ten cost reductions isn't enough for you, don't worry, I've got more! There are plenty of ways discussed in this book to improve inference efficiency. And when you're making an LLM more efficient to execute on GPUs, faster also means *cheaper*. It's useful to reevaluate items 1-10 above regularly as the whole area changes rapidly, with new technology appearing and costs also getting pushed down.

Build versus Buy

Before we dive into the mechanics of building your own AI thingummy in a huge project, it's worth considering the various existing products and tools. You might not need a development project at all, but simply a DevOps project to integrate a new third-party commercial product into your company's infrastructure.

For example, if your project goal is to have staff writers being more productive in creating drafts of various documents or marketing copy, there's this product called ChatGPT from OpenAI. Maybe you've heard of it?

Actually, there are any number of other tools for writer productivity using AI capabilities, some of which use ChatGPT underneath, and some of which are independent. Similarly, there are already a number of "AI coding copilot" type products, which might make your programmers even more amazingly, astoundingly, RSU-worthily useful than they already are. Across the whole spectrum of creative endeavors, there are also numerous AI products that create images, animations, 3D models, and videos.

More generally, there are starting to be AI products for almost every use case that you can think of, and in all of the major industry verticals (e.g., medicine, law, finance, etc.) so it's worth a little research as to what's currently available that might suit your needs. I'm reluctant to offer lists, because it's changing daily. Anyway, it's not my job to review them; it's yours!

Overall, it's fun to build anything with AI technology, but it's faster to use something that's already been built. And these new AI tools are actually so amazing that it's also fun to test them.

Foundation Model Choices

What model are you going to use as the Foundation Model? There are really three major options:

- Commercial models
- Open source models
- Build Your Own (BYO)

Of course, there's that fourth option of *not using AI*, which, as anyone in the AI industry will tell you, leads to analysts shunning your stock, instant bankruptcy, and your toenails catching on fire.

Building your own model is a viable option for small to medium models, that you want to train on your data set. However, only the major tech companies have been successful at training a massive LLM foundation model, given the expertise required and expense of training.

The alternative is to choose an existing foundation model, that is pre-trained on lots of general data. Then you would fine-tune that model on whatever proprietary data that you want to use.

If you have no specific extra data for fine-tuning, then you're basically using a commercial or open source model underneath. You can still achieve significant customization of an existing model without fine-tuning, using techniques such as prompt engineering, Retrieval-Augmented Generation (RAG), and the simple idea of mixing heuristics with AI inference results.

Open Source Models

When ChatGPT burst into public consciousness in around February 2023, there were already lots of open source models. However, they are mostly smaller models and nowhere near as capable. Nevertheless, you could get a lot of value in them at no cost.

Open source models received a huge jump forward when Meta released its Llama model into the open source world. It was licensed only for non-commercial and research purposes, but it was immediately used in numerous ways in the open source community. This model was also used in various ways to create other models that were, theoretically at least, freed of the non-commercial limitations of the original Llama license. That legal issue was never tested and became moot shortly afterwards when Llama version 2 came out.

Meta open sourced its Llama2 model for both commercial and non-commercial usage in July 2023. The license was non-standard, but for most users (who were not already large companies), it was largely free of the restrictions. You should review the details of the Llama2 license yourself, and any future Meta model releases, but it has been widely used in the open source community already.

Commercial-Usage Open Source Models

Although Llama2 probably tops the list, there are several other major models that have been open-sourced in permissive licenses. Again, you should check these license details yourself, as even the permissive open source licenses impose some level of restrictions or obligations. Here is my list of some of the better models that I think can be used commercially:

- Llama2 from Meta (Facebook Research) under a specific license called the Llama 2 Community License Agreement.
- Mistral 7B and Mistral 8x7B (both with an Apache 2.0 license).
- MPT-7B from MosaicML (DataBricks) with Apache 2.0 license.
- Falcon 7B/40B from the Technology Innovation Institute (TII) (Apache 2.0 license)
- FastChat T5 from LMSYS (Apache 2.0 license)
- Cerebras GPT AI Model (Apache 2.0 license)
- GPT4All models (various); some under MIT License.
- H2O GPT AI model (Apache 2.0 license)
- Orca Mini 13B (MIT License)
- Zephyr 7B Alpha (MIT License)

This list is already out-of-date as you read this, I'm sure. There are new models coming out regularly, and there are also various new models being created from other models, such as quantized versions and other re-trained derivative models.

Model Size

Choosing a model size is an important part of the project. For starters, the size of a model has a direct correlation to the cost of both training and inference in terms of GPU juice. Making an astute choice on the type of model you need for this exact use case can make a large impact on the initial and ongoing cost of an AI project.

There's no doubt that bigger models are enticing. The general rule seems to be that bigger models are more capable, and a multi-billion parameter model seems to be table stakes for a major AI model these days. And the top commercial models are starting to exceed a trillion parameters.

However, some research is starting to cast doubt on this, at least in that the trend that ever-larger models may not always result in increased intelligence. For example, GPT-4 is rumored to be eight models merged together in a Mixture-of-Experts (MoE) architecture, each of size about 220B parameters, rather than one massive model of 1.76T parameters.

Quality matters, not just quantity. The quality of the data set used for training, and the quality of the various techniques are important. The quality is important for intelligence shouldn't be surprising. In fact, what should be surprising is that quantity has been so successful at raising AI capabilities.

Model optimizations. How can you have a model that's smarter and faster and cheaper? Firstly, the open source models have improved quickly and continue to do so. Some are starting to offer quite good functionality at a speed that is very fast. There are models that have been compressed (e.g., quantization, pruning, etc.), and there are open source engines that offer various newer AI optimization features (e.g., Flash Attention) You can download both models and engine source code, and run the open source models yourself (admittedly, with hosting costs for renting your own GPUs, or using a commercial GPU hosting service).

For a commercial API, you can't change their engines until you apply for a job there. However, you can reduce the number of queries being sent to a commercial API, mainly by putting a cache in front of the calls. This cuts costs and speeds up replies for common prompts (or similar ones), with the trade-off that non-cached queries have a slightly slower response time from the additional failed cache lookup.

An inference cache is a cache of the responses to identical queries, whereas a semantic cache finds “close-enough” matches in prior queries using nearest-neighbor vector database lookups.

Latency and Response Time

Performance requirements for the overall system are an important part of the design. For a simple text-generation response, such as a chatbot or Q&A, there are two main factors in evaluating the speed of its answering:

- Initial response time (prefill phase or encoding mode)
- Tokens per second (decoding mode)

Response Time (Latency). The initial time delay before an AI engine emits the first word of its response is called the “latency” or “response time.” It is also sometimes called the “time to first token” (which is shortened to TTFT in research papers) or the “prefill time.” There are two factors here that cause high latency:

- Cloud round-trip message time (network)
- Prefill or encoding time (engine)

For a cloud-based AI engine on a phone, the LLM isn’t on the phone, and the prompt question must be sent to the cloud engine over the network. Hence, there is time for the message to get sent into the cloud and back again with the answer (i.e., a “round-trip” network message).

For an on-device native LLM execution on a phone, there’s no network, because the engine runs inside the phone. Hence, native execution’s response time is only about engine speed, and that prefill phase.

Prefill phase. Both the cloud and on-device versions have to run the query on an engine. AI engines often have a significant delay before starting to reply. For example, a 2-second response time is not uncommon. An AI engine in the cloud probably runs faster because it can have powerful GPUs on a big box in a data center somewhere near a big lake. Local AI engines running on a phone don’t have a big GPU, or may not have one at all, so they run slower.

Why are Transformers slow to initially respond? The way that Transformers work is to do an initial phase that is called “prefill” in decoder-only engines (e.g., GPT), or “encoding” in the older style of encoder-decoder engines. This phase calculates a lot of data about the input prompt, but doesn’t emit any output tokens. Hence, it’s also called the “prompt processing” phase.

The length of the prefill phase also depends on the size of the input prompt. More tokens to process in the input means a slower initial response time.

The input tokens are not only the user's question. They also include the "context" that's sent to the AI engine with the query, such as the conversation history in a chatbot, or the "chunks" of documents from a datastore in a RAG architecture for Q&A. These extra context tokens can be significantly longer than the simple query prompt.

Hence, this prefill or encoding time is part of the initial delay before an AI engine answers. For an on-device phone LLM, it's the main delay (there's no network delay). But once the LLM starts talking, then it goes faster in the "decoding phase."

Decoding speed. The second phase of an AI engine's answer is when it starts outputting its response, one token at a time (i.e., a word at a time). How fast this runs is called the "decoding speed" and is usually measured in tokens-per-second. With a big GPU, the engines are still quite slow, and may run in tens of tokens per second. Reportedly, GPT 3.5 can run at a maximum of around 100 tokens per second. Note that tokens are not always whole words, so words-per-second will be slightly less than this (e.g., maybe 75% of this).

On a phone, it's slower because the engine has less hardware support. The SOTA at the moment is more like single digits per second, i.e., only a few words per second of output.

Note that there isn't a big delay between subsequent token outputs in the decoding mode. There's only that initial delay before the first one. The time to output each new token is about the same for each token in the answer, because it basically re-runs the same decoder computations for every output token.

How Fast is Needed?

How fast does an AI engine need to run on a phone? Well, it depends on whether your AI is writing you a novel or helping you scroll through your TikTok feed.

Consider the different speed requirements if it's generating text, voice, images, or video. Also, in what way does it need to be interactive, or is the user just passively reading or watching? Maybe it's controlling some phone aspects in the UI, too.

But let's take basic LLM text generation as an example. This means use cases such as a chatbot, Q&A service, or written document drafting.

The way to think about this is to consider human nature with the two main aspects of the speed:

- Response time
- Decoding output token generation.

Firstly, humans are impatient. We're used to getting a very snappy response to websites and computers. What that means is something like a 200ms initial response time is desirable.

The news is better for the decoding phase. Humans don't read fast. An average reading speed is about 240 words-per-minute, which is only 4 words per second.

Hence, it seems likely that the biggest problem with on-device AI responsiveness is going to be the initial prefill time delay until the first word starts appearing.

The SOTA for running small models (e.g., 1B or 2B) on a phone is not there yet. Papers talk about an initial response time in seconds rather than milliseconds, so the prefill phase is problematic. Decoding rates are better, which high single-digit tokens-per-second achievable after the first one. Hence, the initial delay is problematic, but the decoding speed thereafter is not as concerning.

Incremental Output

The assumption here for decoding speed is that the partial answers starts being emitted by the engine before it's finished its full output. We won't want to have to wait for the entire AI engine's inference phase before outputting some words for the user. In other words, incremental or "streamed" output of AI results is critical.

Voice output. Incrementally creating words to say as voice output might seem slightly more problematic than reading text. If the engine's decoding output speed doesn't keep up with a normal rate of speech, then the voice-based AI assistance will sound staccato. But it's actually not a problem, because people read faster than they speak. Human average reading speed is around 240 words-per-minute (4 words-per-second), where speech is about half that at 100-130 words-per-minute (about 2 words-per-second). Hence, a voice assistant outputting the answer from an AI engine is unlikely to get ahead of the answer, assuming it's basically reading out the text response. In fact, it might be more problematic in reverse, with the voice assistant falling behind, such as if the user can both read and hear the text response at the same time.

Accuracy Requirements

A lot of the decisions about LLMs come down to a trade-off:

- Accuracy versus cost
- Accuracy versus speed

The accuracy versus cost is exemplified in the choice of commercial LLM API to use. The current front-runner in terms of accuracy is OpenAI's GPT-4o, which also wins the award for highest per-token cost as well. Even if you stick with OpenAI, you can trade down to GPT 3.5, which is a less capable model, but costs less, too.

Smartness versus speed. Accuracy versus speed is more of a consideration when you're working with open source models, because you don't have much control over OpenAI's speed. For example, you might have to choose between two versions of Llama or Mistral models:

- Full precision 32-bit floating point model (FP32, not quantized)
- Quantized 4-bit integer model (INT4 quantized)

The FP32 model uses full-size data, and is accurate but slower. The INT8 versus is using 8-bit integers (i.e., 256 possible values), which is faster but less accurate. Note that INT8 models are surprisingly accurate, even though they're a quarter the size of a 32-bit model. Furthermore, a lot of people are using 4-bit models, which are an eighth of the size, but still quite accurate. Smaller models run much faster, and can be quite adequate for many use cases.

On-device versus cloud-based. The same speed-smartness tradeoff arises with on-device execution of AI models. For example, Apple Intelligence allows some queries to run on the device itself, which uses a 3B model. Alternatively, the query can be sent to the cloud, which uses a bigger model that's more accurate and capable, but it's also slower.

Use cases. How accurate do you need? It depends on the use case for your app and whether it's business-critical or not. Some examples where high accuracy might not be required:

- Creative writing
- Image generation (for recreation)

In theory, an app that's responding to customers seems like it has higher accuracy requirement than one for your internal staff. But is that true? Surely, you should not undervalue the importance of your staff getting accurate information.

The trade-off between accuracy and speed is less important when the LLM does "bigger" tasks. For example, consider writing a cover letter for a job application given the position details and resume, or answering a high school history question. Whilst it may take some time for the prefill and further decoding time to emit the whole result, typically the results are "better" and delivered "faster" than the user could do themselves. The "speed" is often not really noticed at all and accuracy is far more important.

I guess context matters a little bit, since this is less true of interactive tasks. For example, if the LLM is auto-completing the next part of a text message being written, or some code in a programmer's IDE, speed matters there. Perhaps we have discovered a new AI scaling law: the smaller the reward from the LLM, the faster it needs to be. The bigger the reward, the less speed matters.

Another factor in considering accuracy requirements is whether the LLM output will be reviewed by a human. Outputs that are going straight out onto the web would need higher accuracy than something that will be revised and curated by a human. On the other hand, don't assume that humans are good at reviewing reams of text, as they'll have the tendency to just hit the "Approve" button without reading it properly. Maybe you should use an LLM for that?

References

1. Morgan Cheatham, Steve Kraus, December 4, 2023, *The six imperatives for AI-first companies*, <https://www.bvp.com/atlas/six-imperatives-for-ai-first-companies>
2. Matt Asay, Sep 23, 2024, *Too much assembly required for AI*, <https://www.infoworld.com/article/3536292/too-much-assembly-required-for-ai.html>
3. Stan Gibson, 03 Jun 2024, *Getting infrastructure right for generative AI*, CIO, <https://www.cio.com/article/2128440/getting-infrastructure-right-for-generative-ai.html>
4. MongoDB, Jun 20, 2024, *Understanding the AI Stack In the Era of Generative AI: Exploring the Layers and Components of Today's AI Applications*, <https://medium.com/mongodb/understanding-the-ai-stack-in-the-era-of-generative-ai-f1fcd66e1393>
5. Akash Bajwa and Chia Jeng Yang, May 27, 2024, *The RAG Stack: Featuring Knowledge Graphs: Reducing Hallucinations To Make LLMs Production-Grade*

With Complex RAG, <https://akashbajwa.substack.com/p/the-rag-stack-featuring-knowledge>

6. Artem Shelamanov, Jun 30, 2024. *Tech Stack For Production-Ready LLM Applications In 2024*, <https://python.plainenglish.io/tech-stack-for-production-ready-llm-applications-in-2024-5eb14105d1b4>
7. Abhimanyu Bambhaniya, Ritik Raj, Geonhwa Jeong, Souvik Kundu, Sudarshan Srinivasan, Midhilesh Elavazhagan, Madhu Kumar, Tushar Krishna, 3 Jun 2024, *Demystifying Platform Requirements for Diverse LLM Inference Use Cases*, <https://arxiv.org/abs/2406.01698> Code: <https://github.com/abhimbambhaniya/GenZ-LLM-Analyzer>
8. Dylan Patel and Daniel Nishball, Oct 03, 2024, *AI Neocloud Playbook and Anatomy*, <https://www.semianalysis.com/p/ai-neocloud-playbook-and-anatomy>
9. Ryan Lucchese, Niki Birkner, Yaron Hagai, Virginia Adams, August 13, 2024, *A practitioner's guide to testing and running large GPU clusters for training generative AI models*, Together AI, <https://www.together.ai/blog/a-practitioners-guide-to-testing-and-running-large-gpu-clusters-for-training-generative-ai-models>

12. Architectures

Easier Architectures

Here are some thoughts on the easier ways to build an AI project:

- Renting GPU juice from cloud vendors or GPU-specific hosting vendors is easier than buying them from NVIDIA for your own servers.
- Cloud-based AI is easier and more solid than on-device inference with AI phones or AI PCs, which are both newer and unproven.
- RAG architectures using document databases are easier than fine-tuning a model with those documents (even with fine-tuning speedups like LoRA).
- Agentic architectures with retrieval capabilities are easier than fine-tuning a model.
- Wrapping ChatGPT is easier than using open source models, but the gap is closing with various good open source servers and toolchains for better productionization.
- Foundation models start with the letter 'P' for a reason. Don't try to make your own from scratch, even if it's only small.

To be a little more specific, here's an approximate hierarchy of AI architecture complexity, from easiest to hardest, in terms of the difficulty of building or coding:

- Wrap architecture (stateless)
- Wrap architecture (statefull, with conversation history)
- RAG architecture (stateless)
- RAG architecture (statefull)
- Agentic architecture (stateless)
- Agentic architecture (stateful)
- Fine-tuning with LoRA (PEFT)
- Fine-tuning with multi-LoRA serving
- Fine-tuning (full parameter)
- AI platform (based on commercial or open-source models, with fine-tuning)
- Foundation model training (from scratch)

I'm not sure where to put “on-device” in that list. It's still too early in the cycle to really tell what the SDKs for Google Android and Apple Intelligence offer.

For all of the “wrap” versions, it's easier to wrap a commercial service (e.g., OpenAI API), because that does all of the backend work for you. Using an open source model requires setting up your own backend server with both an LLM and an open source inference engine. These are both quite freely available online, but it's still a bit more work than using a paid hosting service.

Components of AI Architectures

A lot of the components of a full production AI architecture are similar to any other online application. However, there are some additional AI-specific components:

- LLM
- Backend inference engine

And every AI application will also require work on the client side:

- Prompt engineering module
- UX components

If it's a RAG architecture, there are some extra ones:

- Datastore
- Embedding module
- Retriever (keyword and/or vector lookup)
- Vector database
- Query merge component
- Citation manager

Some less obvious extra components for a live generative AI deployment include:

- Observability monitoring (instrumentation)
- Conversation state manager
- Caching module (e.g., there are about ten types!)
- Prompt shield / safety monitor
- LoRA adapters (or “multi-LoRA”)
- Tool integrations (“function calling”)

I'm sure I could think of a few more to add to the above list, but that'll do for now.

Software Architecture

If you're building a significant portion of the software architecture for your AI project, then nothing is more important to the speed of a program than its architecture. I mean, look at AI. The whole architecture is a massive fail, endlessly bloated with far too many weights and a brute-force algorithm. Sadly, that's the best we've come up with so far, but there's a lot of research about these architectural issues that will probably solve it.

Anyway, as any professional programmer will tell you, it's not difficult to choose a better architecture for your AI project. Fortunately, the best software architecture in the world is well-known to everyone, and is clearly this one:

- Object-oriented objects (OOO)
- Client-server
- Server-client
- Message passing
- Thin client
- Gamification
- Virtualization with Vectorization
- Model-View-Controller (MVC)
- UI-Application-Database (3-level)
- Event-Driven Architecture (EDA)
- `#include "beer.h"`
- Clouds
- Fog computing
- Postel's law
- RTFM
- Microservices architecture
- Service Oriented Architecture (SOA)
- RESTful API architecture
- Intelligent Autonomous Agent (IAA)
- Intentional virality
- Goto considered helpful

Actually, sorry, that wasn't the best architecture in the world; it was just a tribute.

AI Tech Stack

The tech stack for an AI project is similar to a non-AI project, with a few extra components. There's also a much greater importance tied to the choice of underlying hardware (i.e., GPUs) than in many other types of projects. The tech stack looks something like this:

- User interface (client)
- Web server (e.g., Apache or Nginx)
- Application server
- Load balancer (e.g., HAProxy, Nginx, Traefik)
- Message Queue/Event Streaming (e.g., RabbitMQ or Apache Kafka)
- AI request manager
- AI Inference Engine (and model)
- Operating system (e.g., Linux vs Windows)
- CPU hardware (e.g., Intel vs AMD)
- GPU hardware (e.g., NVIDIA V100 vs A100)

Some of these layers are optional or could be merged into a single component. Also, if you're using a remote hosted AI engine, whether open source hosting or wrapping a commercial engine through their API, then the bottom layers are not always your responsibility.

AI engine choices. How much of your AI tech stack will you control? Probably, at least for your first AI project, you're renting a commercial model by the token, which is located in the cloud. In this case, you have almost no control at all.

Alternatively, if you're running your own open source model with an open inference engine (e.g., vLLM or Llama.cpp) on Linux servers in the basement, then you've got more problems, ahem, I mean, more flexibility.

With full control over the hardware and software, it makes sense to make symbiotic choices that allow maximum optimization of the combined system. For example, if you've decided to run the system on a particular GPU version, then your AI engine can assume this hardware acceleration is available, and don't need to waste resources on ensuring your engine's software runs on any other hardware platforms. However, you might need to periodically check that your GPU is still running an LLM and not mining Bitcoin.

Chains and Toolchains

AI architectures involve a lot of components, rather than just the Transformer engine and a model. These components are typically called “tools” and usually have these features:

- Receive input (e.g., text prompt)
- Process it in some way
- Produce output (often text, but not always)

The idea is that each tool is:

- Modular
- Specific (one task)
- Reusable

When we put more than one of these “tools” in a sequence, such that the second tool uses the output of the first tool as its input, this is called a “chain” or a “toolchain”. This terminology is particularly formalized in LangChain architectures.

We’ve seen this idea before. If you’re a Linux boffin, this is a toolchain, too:

```
cat input.txt | sort | uniq -c | sort -nR > output.txt
```

The main idea is that each tool does a specific task, and they can be put together in a sequence. This is really a founding architectural idea from the original Unix systems.

The only major problem with toolchains is that they aren’t parallelizable. The second tool has to wait for the input from the first tool, so they’re inherently sequential. Sequential execution is often, but not always, a particular limitation with this architectural idiom.

One way they can be parallelized is if the second tool only needs partial output from the first tool to do its work, so they can use a streaming model in parallel, which is, again, the original way that Unix worked.

Wrap Architectures

Wrap architectures are those where you create a “wrapper” around someone else’s LLM. This is where you use someone else’s AI-as-a-Service (AaaS) platform, and pay by the token (or by the hour perhaps). The advantages of this are:

- No server work
- No LLM work

Basically, half of your AI app is pre-built for you. Your main work is:

- Prompt engineering
- UX improvements

The main downsides are:

- Pesky invoices with big numbers for per-token charges.
- You pay for both “up” and “down” tokens.
- You also pay for “embeddings” (vector-based).
- You have no control of the server side.
- Privacy issues viz where user data is stored.

For some “wrap” architectures, consideration needs to be given to who pays for the LLM. It’s very possible to wrap an LLM with a user paying, but to do so, the user needs to provide their “keys” for the API. This can help with some of the “pesky” per token charges.

But I don’t know why I’m mentioning all these non-issues. You just send the invoices to your boss, whose only response will be: “Spend more!” The issue of “no control” is probably not that significant, because these companies are likely better at running LLMs than you are. Also, the main thing you’d want to change is to fine-tune an LLM with your proprietary data, or upload your data for a RAG architecture, and all of the major LLM platforms have that covered.

The privacy issues are somewhat problematic. If customers are your target for the LLM app, then it’s their private data, but it’s your problem. If your app is internal, then you also have some employees “leaking” proprietary info to a third-party vendor whenever they use your “internal” AI app. But, come on, company employees are already doing that by getting ChatGPT to edit their M&A takeover memo, aren’t they?

Another consideration is do you allow the user to bring their own AI? That, is instead of wrapping OpenAI and asking for their OpenAI keys, they could bring Claude or Gemini or Mistral or Grok or HuggingFace or a few dozen more. In such cases, you are literally wrapping an AI you do not necessarily know about. It may be useful to control this to some extent, but this may increase your testing costs.

If you are providing the API credentials, you need to be careful not to surface them anywhere. Do not hardcode them into your user's application in some way. Instead, you'll need to add your own server between the user and the API endpoints. It's also good practice to rotate the API key frequently, as there is a market for compromised AI keys.

On-Premises LLM Architecture

On-premises AI architectures are those where you run the LLM yourself on your own servers. This is a great architecture compared to commercial AP wrap architectures, because you avoid the nasty per-token charges of your LLM provider.

Instead, what you do is use open source everything:

- Linux server software
- Apache or Nginx web server
- Open source inference frameworks
- Open source LLMs

And you only need to provide:

- Server hardware
- GPU chips
- Spare parts
- Firewall devices
- Liquid cooling
- Raised floor space
- System administrators
- Build engineers
- Linux experts
- Open source LLM experts
- System builders
- Network engineers

So, totally free!

Client architecture. For an on-premises LLM deployment, the rest of the AI app, on the client side, is very similar to a “wrap” architecture with a third-party LLM vendor, except that you’re wrapping your own LLM. You can use an on-premises self-wrap architecture with any clients: web browser, apps, etc. All of the work that you do for prompt engineering and UX is effectively the same.

But an on-premises implementation of LLMs does require you to do some other extra modules of work:

- Security credential management (i.e., user account logins, basic passwords, forgotten password, etc.)
- Fine-tuning methods for open source models using your data.
- Monitoring and “observability” (LLMOps).
- Performance tuning and scaling.

Private Cloud LLM Architecture

The half-way measure is to have your own “private cloud” and that’s not an oxymoron at all! I’m not even sure what that really means, but I’ve seen it plenty of times in nice glossy articles. Maybe it means:

- Running your own hardware servers, but with cloud server software on them, or
- Renting cloud hardware, but putting your own software on it.

The first one is basically the same as “on-premises”, is it not? The computers are down in the basement, and if you’re good at this stuff, then you’re running “cloud software” like Apache, or maybe Nginx if you’re really trendy, and therefore it’s a “cloud” deployment. If you’re not good at this stuff, then someone else is sending spam emails out from your basement.

Alternatively, you can rent physical servers from one of the hyperscalers, like AWS, Azure, or GCP. Instead of running your own physical hardware, you can rent access to their servers, whether “bare metal” servers or per-hour rentals, or whatever you can talk them into. This is really a cloud architecture, and not really that private if you ask me, but you didn’t, so I’ll be quiet about that.

Two-Step Architectures

All of these architectures have different backends, but the front-end piece is the same for each architecture. It differs depending on what type of app you're creating, in terms of business logic, but you're still mainly doing prompt engineering and user interface work for the non-server parts.

In an ideal world, you could just write up some JavaScript to do the GUI components and also do some fancy string concatenation for the prompt engineering. No need for you to do anything else!

Back in the real world now, you need what I call is a “two-step architecture” because regardless of what's happening with the LLM, you still need:

- App serving basics — e.g., IP address location, DNS, etc.
- Basic back-end web serving capabilities — e.g., serving out those JavaScript files (and images, CSS files, fonts, blah blah blah).
- User logins and credential management
- Monitoring and management features

Oh, no! I just reverted to 1990s vocabulary. The correct terms are “observability,” “orchestration” and “LLMOps” for, you know, monitoring and management.

Anyway, what you'll find is that you still need all your Linux backend software developers, even if you're just using the ChatGPT API. That old backend computer expertise is important for the basic server infrastructure, even if you can offload the new-fangled LLM expertise to the ChatGPT API (for a fee).

And yeah, I know, we had two-step architectures in the 1990s, too. But I forgot what they were called, so I invented my own terminology. See how AI hype works?

Security Credential Management

One of the things we've glossed over in the above discussion is security. How do you validate that only your legitimate users can use your LLM-based app?

This is a problem even for wrap architectures with third-party APIs. For example, with the OpenAI API, you get a set of credentials to use the API, and you'll get billed for anyone accessing the API with these hexadecimal strings.

But, where do you put this file?

If you're doing a phone app or a downloaded application for Windows or Mac, you can't just put the credential file in there. It's not protected, and someone will find it and abuse it. In theory, you could hard-code it into your client-side app with some kind of encryption, but it would have to be reversible encryption, and a hacker could probably disassemble that.

It seems like the platform vendors should have a better solution for this, but I think it's still an issue. It's not easy to fix, but maybe this will improve in the near future.

Web-based applications don't have any client-side files, so this makes it simpler, because obviously you store the credentials on your server. Hence, if you're doing a two-step web-based application, then at least you can control the API credentials away from the user. It's a small file buried somewhere in a Linux server, and also in your version control system, available to all 1,000 of your developers at your company, and also uploaded onto Github in one of their skunkworks projects, too.

The good news is that your users can't just steal your credentials by using your app's front-end interface. But you can't just allow them to anonymously send unlimited queries to ChatGPT via your two-step server-side pathway. So, then you've got to code up your own login capabilities for your app interface and manage the users on your server. The work never ends!

On-Device AI Architectures

On-device architectures in the AI industry have come to mean one of these:

- AI phones
- AI PCs

But the term “on-device” also includes also sorts of other “edge” devices, such as:

- Cars
- Trucks
- Tractors
- Drones
- Nuclear-capable fighter jets
- IoT network devices
- Raspberry Pi
- Smart refrigerators
- Toaster ovens with personality

Okay, so I'm joking about the military applications, for which old-school ML models are more useful than generative AI. I really don't think we're going to see drones with LLMs on board, unless the goal is to fly over to the enemy and tell them some good jokes.

On-device inference is one of the hottest areas in the tech industry at the moment, but it might not be relevant to your business AI applications. It depends on what user devices are going to be used to access your AI applications. Your input devices might well be phones, tablets, or laptops, but even then you might not care about on-device inference, because those devices can simply connect to LLMs in the cloud via the internet. On the other hand, maybe you care tremendously about on-device LLM capabilities, in which case the following chapters examine AI phones and AI PCs in more detail.

References

1. Johannes Schneider, 1 Aug 2024, *What comes after transformers? -- A selective survey connecting ideas in deep learning*, <https://arxiv.org/abs/2408.00386>
2. Cem Dilmegani, Jan 10, 2024, *The Future of Large Language Models in 2024*, <https://research.aimultiple.com/future-of-large-language-models/>
3. Yehui Tang, Yunhe Wang, Jianyuan Guo, Zhijun Tu, Kai Han, Hailin Hu, Dacheng Tao, 5 Feb 2024. *A Survey on Transformer Compression*. <https://arxiv.org/abs/2402.05964> (Model compression survey paper with focus on pruning, quantization, knowledge distillation, and efficient architecture design.)
4. Xinji Mai, Zeng Tao, Junxiong Lin, Haoran Wang, Yang Chang, Yanlan Kang, Yan Wang, Wenqiang Zhang, 27 Jun 2024, *From Efficient Multimodal Models to World Models: A Survey*, <https://arxiv.org/abs/2407.00118> (A survey of multimodal models with coverage of many optimization techniques.)
5. Badri Narayana Patro, Vijay Srinivas Agneeswaran, 24 Apr 2024, *Mamba-360: Survey of State Space Models as Transformer Alternative for Long Sequence Modelling: Methods, Applications, and Challenges*, <https://arxiv.org/abs/2404.16112>
6. Tianyu Ding, Tianyi Chen, Haidong Zhu, Jiachen Jiang, Yiqi Zhong, Jinxin Zhou, Guangzhi Wang, Zhihui Zhu, Ilya Zharkov, Luming Liang, 18 Apr 2024 (v2), *The Efficiency Spectrum of Large Language Models: An Algorithmic Survey*, <https://arxiv.org/abs/2312.00678>
7. Matt Murphy, Tim Tully, Grace Ge, Derek Xiao, Katie Keller, January 18, 2024, *The Modern AI Stack: Design Principles for the Future of Enterprise AI Architectures*, <https://menlovc.com/perspective/the-modern-ai-stack-design-principles-for-the-future-of-enterprise-ai-architectures/?tpcc=NL> Marketing

8. Artem Shelamanov, Jun 30, 2024. *Tech Stack For Production-Ready LLM Applications In 2024*, <https://python.plainenglish.io/tech-stack-for-production-ready-llm-applications-in-2024-5eb14105d1b4>
9. Thiagarajan Maruthavan (Rajan), Apr 12, 2024, *So what if it is a thin wrapper on OpenAI?* <https://medium.com/@mtrajan/so-what-if-it-is-a-thin-wrapper-on-openai-274dd005b6d3>
10. Michael J. Lever, Aug 2024, *AI or API? | Chatbot cuckoos are bloating tech. OpenAI wrappers are becoming a shortcut for start-ups, but are they sustainable?* <https://medium.com/future-ux/ai-or-api-chatbot-cuckoos-are-bloating-tech-d6b8d8255279>
11. Quang H. Nguyen, Duy C. Hoang, Juliette Decugis, Saurav Manchanda, Nitesh V. Chawla, Khoa D. Doan, 24 Jul 2024 (v2), *MetaLLM: A High-performant and Cost-efficient Dynamic Framework for Wrapping LLMs*, <https://arxiv.org/abs/2407.10834>

13. Transformers & LLMs

AI Engines & Models

An AI application is really two components and it's not very complicated:

- Engine — Transformer
- Model — LLM

Transformers are a type of neural network engine that calculates the answers in Generative AI. The Large Language Model (LLM) contains all of the data about the relationships between words and their relative positioning.

In terms of technology, the distinction between engines and models is very simple:

- Engine — code
- Model — data

The runtime code is the “engine” and the grunt work is often done in C++ under a Python wrapper. The data is the “model” which is literally all numbers, and no code. So far, not so exciting.

Where it gets more interesting is in the complex meshing between engines and models. Not all engines work with all models, and vice-versa. Even Transformers are tightly interwoven with their LLM data. There are many variants of Transformer architectures, and the data won't work with an architecture that's different.

Engines and models are symbiotic and you need both to get anything done. An engine without a model means you ran out of compute budget, whereas a model without an engine cannot really occur because engines create models via training.

Engines

What's an engine? The engine is code that you have to write. All of the fast low-level code is usually written in C++, but the higher-level control code is often written in Python. Someone has probably used Java in AI engines, but I'm not a fan of ten directory levels. If you're using Visual Basic or Perl, we're in trouble.

All of the action is done by the engine based on the data in the model file. The engine needs to load the model file, receive a user query, crank the query through the model weights, and output the best ideas it can think of.

There are two main types of engine, and they're so closely related, that they're almost the same thing. Conceptually, there are two engines:

- Training engine
- Inference engine

The training engine computes answers to queries, compares the results to expectations, and then updates the weights in the model. The “loss function” calculates how close the results are to what's expected in the training data set. It's also sometimes called an “error function” because it computes an error metric between the computed results, and the expected results. At a very high level, the basic architecture is:

```
Training engine = Inference engine
                  + Loss function
                  + Weight Updater
```

The training engine is used for training (surprise!) and for mini-training tasks like “fine-tuning” the model with small amounts of extra data. The main purpose of the training engine is to create the model by continually updating the weights.

The inference engine handles user queries at runtime. It requires a model that has been built during training, which is used to answer the users' prompts according to whatever has been trained into the model.

These two types of engines have the same inference component. A “training engine” is the inference engine plus a mechanism to compare results with expectations and then update weights appropriately. The central difference is that a training engine changes the weights, because it's creating the model, whereas the weights are static during inference. The weights are not updated by user queries. If you like programming (hopefully?), here's another way to think about model weights:

- Training engine — Read/Write
- Inference engine — Read-Only

Both of these engines do the same inference computations on weights for the “Read” phase. Hence, they share a lot of components, but the training engine adds extra components (the “Write” parts).

The basic hyper-parameters of the model (e.g., the number of weights, the number of layers) must be identical for the training and inference phases. Hence, a query computation done by the training engine is the same set of computations as the same query done by the inference engine after training is complete.

Transformers

What's a Transformer? It's an engine that processes LLMs, and is an advanced type of neural network. The first Transformer was open-sourced by Google Research in 2017, and then everything got out of hand. And now, I have to mention that Transformers can be of three basic types:

- Vanilla (encoder-decoder),
- Decoder-only (e.g., GPT or Gemini)
- Encoder-only (e.g., BERT)

Now you'll want definitions for those too? We'll be here all night, and it's not even a joke, because the whole book is about Transformers and LLMs. Here's a list with some of the well-known Transformer-LLM architectures:

- GPT-4 (OpenAI)
- Gemini (Google)
- Llama (Meta)
- Mixtral (Mistral)
- Celebrity models (Character.AI)
- Claude (Anthropic)
- Grok (xAI)

Yes, I've missed a few! Although it all started as encoder-decoders in 2017, now most of the modern Transformers are decoder-only (because it's faster). In addition, they've all tweaked the vanilla Transformer in lots of different ways and usually have also published nice research papers about it (until recently).

And I know what you're wondering: it's always about ChatGPT from OpenAI. No, ChatGPT isn't on the list, because it's not really a Transformer architecture or an LLM. Rather, it's more like an "app" (chatbot) or a "brand" or a "platform" that sits on the top using all that GPT stuff.

Models

What's a model? An AI model is literally a binary data file with mostly numbers and a few text strings. For really big models with billions of parameters, it's multiple files, but still only numbers and no code.

I really mean it: zero code. You won't find any programs or scripts, and not even any HTML markup. If you're looking for rules like "if the previous word was 'the' then output a noun" then you're out of luck, not to mention that you're about thirty years behind the times, because that's a rule-based expert system, and it's not how models work in this century.

The main thing in a model file is "weights" which are literally fractional numbers. Billions of them. They're sometimes called "parameters" when being more precise, but it's the same idea.

Weights are a multiplier of "signals" such as which word should be output next. A fractional number less than one makes a word less likely (decreasing a signal), whereas more than one increases the likelihood of outputting that word (amplifying a signal). A zero means don't output that word. A negative weight means really, really don't output the word.

Programmers don't create model files. You won't have to edit a model file and click away on your calculator to get the right parameter numbers. The numbers inside a model file are auto-generated by the training engine.

In fact, it's hard even to look at a model file, because it's so crammed full of numbers. You can do a basic sanity check that it's not spoiled with bogus oddities like `Inf` (infinity) and `NaN` (not-a-number) floating-point values, but you can't see the intelligence by looking at the numbers, even if you squint. However, programmers do have to decide on the meta-parameters for their model before they run the training phase.

What are meta-parameters? The meta-parameters of the model are counts of how many billion parameters it has, in how many layers, and how many different words it understands (typically, 50,000). These are all static, fixed numeric values. Most of the meta-parameters are fixed from training through to inference, such as the "dimensions" of the model (e.g., the number of "layers" in the model). The size of the model in terms of how many billions of parameters is mostly fixed, too, except there's some tricky ways to speed up inference by reducing or modifying parameters, called "pruning" and "quantization," but now we're jumping ahead about twenty chapters.

Large Language Models (LLMs)

What's an LLM? There's nothing really special about Large Language Models (LLMs) used by ChatGPT, Gemini, or Llama, compared to other types of AI model files, except that they're:

- (a) large,
- (b) language-focused (not images), and
- (c) a model.

Well, you asked, so I answered.

More specifically, LLMs tend to be model files that are processed by Transformers, rather than other types of AI engines.

What's a Foundation Model? This is a large and general-purpose model that's already been broadly trained. Any model that has billions of parameters and gets mentioned in a press release is usually a foundation model. The biggest foundation models might support text in multiple languages along with programming language coding knowledge.

Technically, if a foundation model also has image generating capabilities as well as text output, or can also receive an image as part of its input, then that's not a normal foundation model (i.e., it's not really an LLM). Instead, this advanced model type is often distinguished as a “multi-modal” model. And if there's two of those working together, then it's a “multi-modal multi-model” and you should try saying that ten times in a row.

Other Types of LLMs

LLMs are not the only game in town. There were AI models before LLMs, and there are some newer models, as well. Here's my list of non-LLMs:

- Predictive ML models — old-school AI with various types of neural networks.
- Diffusion models — image-related, using new tech, but not based on Transformers.

Some of the newer types of models based on LLMs and Transformers include:

- Small Language Models (SLMs) — only a few billion to fit in the palm of your hand.
- Mixture-of-Experts (MoE) — multiple LLMs wrapped in one uber-LLM, such as GPT-4, Mixtral, and Google Gemini.
- Large Multimodal Models (LMMs) — the latest thing!
- Large Vision Models (LVMs) and Vision Transformers (ViTs) — machine vision with Transformer architectures.
- Time Series Forecasting LLMs — data prediction with Transformers (e.g., Amazon Chronos, Nixtla TimeGPT, Salesforce Morai, IBM Tiny Time Mixers, Google TimesFM).

Training and Fine-tuning

What’s the difference between training and fine-tuning? At least three zeros.

Training is how you shove all of the brain power from the entire Wikipedia corpus into a bunch of numbers. It takes a long time and the GDP of a small country to train a big model. Training is the big cost of a lot of AI projects.

The good news about training is that if you mess it up, you have to start all over again. Well, this isn’t quite true, because training runs in batches of data. If the evaluation fails, you have to revert to the prior model candidate, since you can’t “un-train” an AI model. However, a review can also suggest areas where a model needs more training, or needs to be directed towards new behavior or personality features. In addition to batched training, there is also research on “incremental learning” as a thing.

What is fine-tuning? Fine-tuning refers to smaller amounts of training that are done to a model that’s already been fully trained. If you’re training a new model from scratch, even a small one, then that’s training, not fine-tuning.

The most common use of fine-tuning is to modify a powerful foundation model to do something more specific. Most foundation models have been broadly trained on general information. You might want to specialize the model for a particular use case or to use a new set of data. This can be done two ways:

- Fine-tuning
- Retrieval Augmentation Generation (RAG)

Fine-tuning adds new “knowledge” to the LLM about the extra training content. One risk is that it does so by changing the behavior of the underlying foundation model. Hence, the fine tuning could change the LLM enough that it “forgets” whole areas of “information” that were in the foundation model, or rather, make it prefer the new fine-tuning over the existing data. There is little that can be done to control what the LLM “forgets” other than testing for this afterwards.

Proprietary business data is a common reason to fine-tune a foundation model (but there’s also RAG to consider). For example, to create a support chatbot for customers using your products, you can customize a foundation model to know about your company’s internal product documents via fine-tuning. To do this, you would fine-tune the foundation model using this extra internal data. In this way, a small amount of fine-tuning has added knowledge to the model about new data, which it can then incorporate into its answers to users. It’s basically telling the LLM: forget about everything you “know” about a topic, and just derive your responses from the data in the newer prompt, rather than the data pre-encoded in the model.

RAG is not training. Note that Retrieval Augmentation Generation (RAG) is not a type of training or fine-tuning. In fact, it’s a way to avoid them like the plague.

RAG is an architectural add-on where the Transformer can talk to a component that knows how to “retrieve” extra information or documents, such as proprietary internal business documents about your products. This extra data is used as input context during inference of the model, thereby extending the basic model to answer questions specific to this extra material.

The main point is that RAG avoids the expense of training and fine-tuning, while incurring some extra cost in implementing the RAG component. Note that a RAG architecture is also not “tied” to the model/engine at all, and it is generally very easy to switch engines or models behind the scenes in the RAG application code.

Data sets. High-quality data is fundamental to both fine-tuning/training and RAG techniques. Historically, much of the training data sets have been painstakingly compiled by humans. A newer technique is to use the output of one LLM as the input training dataset for another model. This method and other types of “synthetic data” are being used more fully.

The required structure of the data is different for RAG versus fine-tuning/training. RAG data needs to be split into “chunks” of relevance to particular questions, which can map reasonably well to internal documents. Data for fine-tuning may require more of a Q&A or conversational structure, which is not typical of product datasheets or corporate policy documents!

For example, if you want have a chatbot which answers questions, then the training data needs to be a lot of question/answer pairs. With RAG, on the other hand, the data is wrapped in a question as part of a hidden prompt.

Merely feeding an LLM the whole of your corporate marketing materials, or even the whole of Wikipedia, will not help that much with fine-tuning or training. Feeding an LLM with a meal of silky Wikipedia text means it likely can “complete” an input in the style of Wikipedia, and probably cite paragraphs verbatim from the corpus, but not necessarily answer questions about the content in Wikipedia.

The training data needs to be a lot of question/answer type pairs in order to answer questions (i.e., so that the “completion” is an answer to the question). LLMs just predict a likely next word and next sentence, so the LLM has to learn to predict an answer after a question.

Similarly for a translation use case, if you want your LLM to translate English to Klingon, the training data can’t just be the English Wikipedia content combined with the Klingon one, but needs to be pairs of English phrases and their corresponding Klingon phrases.

Inference

What is inference? The term “inference” is the AI way of saying “running” or “executing” the AI model. Inference and training are different phases. When you’re training or fine-tuning, that’s not inference. But when you’re done and deploy your model live for a nickel a query, that’s inference. When you ask ChatGPT a question, you’re sending a “query” or “prompt” to its “inference engine” and when it politely refuses to do what you ask, that’s the output results of its “inference” code.

What are latency and throughput? Latency is how fast your inference engine runs. It’s similar to the idea of “response time” for a single user. Throughput is a measure of how many queries your engine can handle over time, which relates more to how fast your engine can handle a group of users submitting many queries.

Types of inference. There’s not only one type of inference, and the exact algorithm depends on what you’re trying to do. Some of the types include:

- Completions. This means extending the prompt into a longer answer. Common use cases include auto-writing text or answering questions.
- Translation. Convert your Python code comments into Klingon.
- Summarization. Taking the input prompt, such as a paragraph or document, and creating a brief summary.

- Grammatical Error Correction (GEC). Also known to non-researchers as “editing.”
- Transformation. Changing the tone or style of a text input, or changing the formatting and presentation.
- Categorization. Analyzing the inputs into a set of different categories, which is effectively a subset of summarization.

Inference Settings. In addition to choosing the overarching type of inference algorithm, there are some common parameters to control an inference request.

- Temperature. A higher temperature setting gives your engine a fever, and makes it output silly words. This is known as “creativity.”
- Token limit. This is the simple idea of limiting the number of words (tokens) that the engine is allowed to output in its response.
- Formatting. Do you want the engine to output plain text, a table, or some other format.

This is only a sample list, and API providers typically have many more options. There are also usually various other parameters related to the security and tracking of requests. For example, you probably have to submit your security credentials (i.e., password) along with a unique ID for the request. This helps the API validate your request and helps you keep track of which end user submitted the request so you can send the answer back to them.

Context and Conversations

If you’re creating a chatbot, you create a UI that accepts the user’s inputs, sends it off to the AI engine via the network, and then outputs the answer back to the user. They go back-and-forth with a stream of requests and responses, thereby creating a conversation.

Oh, really?

What’s missing is the “context” of every request that’s part of the conversation. You cannot just send the user’s latest response off to the engine, because:

AI engines are stateless.

Hence, the default AI engine doesn’t remember what else it’s already said. Maybe it’s because the GPUs have stolen all their RAM.

Instead, it's up to you as the programmer to store and re-submit the entire conversational history with every request. This is a wonderful situation when you're paying per input token.

The API vendors are starting to help handle context management for you. The OpenAI API provides helpful ways to structure the historical context in a request. The context is also stored in the KV Cache of an LLM, and the idea of swapping the KV cache in and out is becoming a thing. It's called "prefix KV caching" and it's already available in vLLM, DeepSeek, Anthropic, Google Gemini, and OpenAI. This allows the same results for a repeated query to be shared within a single user's session, or also across many users as a "short term memory" transplant occurring each context switch.

Extended Transformers

The main type of Transformer that gets all the hype is the Generative Pre-Trained Transformer (GPT). This is the basic text processing Transformer that can process words and generate output with surprisingly human-like elegance.

Modern research has been applying Transformers to other types of input and uses cases. The result has been various new extensions of Transformer architectures.

- Multi-modal Transformer. This refers to Transformers that can accept inputs in images (or video) rather than simple text prompts.
- Vision Transformer (ViT). These are the use of Transformer technologies for computer vision applications, such as self-driving cars.
- Bidirectional Transformer. This is a research type used in the past, that hasn't received as much attention lately. The idea is that it can examine its input data from both directions at the same time. The main example is "Bidirectional Encoder Representations from Transformers" (BERT) and its many variants.
- Retrieval Augmentation Generation (RAG). This is an architecture where a Transformer is combined with a separate component that "retrieves" extra data (e.g., a document search mechanism). The idea is to extend the AI engine to new data without extra training.
- Ensemble inference. An "ensemble cast" is a Hollywood term that means a film with a group of famous actors all starring together in the same story. Someone with a sense of humor (or very large ambitions) decided to use the same term for a group of AI models all working together to create the same masterpiece.

Some of the major areas of Transformer research involve addressing the resource-hungry nature of their execution. For example, a basic Transformer has quadratic cost complexity in terms of the input length. Hence, there are numerous modifications in Transformer architectures being created, both in industry and research labs. See Part VII of this book for a full literature review of the extensive body of research related to Transformers.

Other Types of Neural Networks

The Transformer was a breakthrough in the evolution of neural networks. One of its main advantages was its capacity to perform calculations in parallel, allowing it to increase intelligence through sheer brute-force algorithms. This led to a massive increase in the size of models into multi-billion parameter scale, which we now call Large Language Models (LLMs).

Before the Transformer, there were many different neural network architectures. Several of these designs are still being used today in areas where they are stronger than Transformers.

Recurrent Neural Networks (RNNs). An early type of neural network that worked iteratively through a sequence. An RNN processes its inputs one token at a time, creating its output response, and then re-enters its own output as an input to its next phase. Hence, it is “recursive” in processing its own output, which is also known as “auto-regressive” mode when this same idea occurs in Transformers. Transformers have largely displaced RNNs for applications in text processing and generative AI. However, there are still research papers attempting to revive RNNs with advancements, or to create hybrid Transformer-RNN architectures.

Generative Adversarial Networks (GANs). These are an advanced image-generating neural network. The idea is to combine two models, one that generate candidate images (the “generator”), and the other model that evaluates them (the “discriminator”). By a weird kind of “fighting” between the two models, the generator model gradually creates better images that please the discriminator. The results are surprisingly effective, and this technology is still in use today.

Convolutional Neural Networks (CNNs). Whereas RNNs and Transformers are focused on input sequences, CNNs are better at input data that has a structure, especially the spatial structure inherent in images. Modern image processing and computer vision technology still uses CNNs, although enhanced Transformer architectures, such as multimodal or vision transformers, can also be used.

CNNs are good at splitting an image into separate input “channels” and then applying a “filter” to each channel. Hence, CNNs have been holding their own against Transformers in areas related to image processing.

There are various other types of neural network, which all have some research attention:

- Long short-term memory (LSTM) — a type of RNN.
- Spiking neural networks (SNNs)
- Liquid neural networks (LNNs)
- State Space Models (SSMs) — e.g., Mamba.
- Quantum neural networks (QNNs)

There’s another architecture that’s fairly commonly used, in both consumer and business. It’s called a Carbon-Based Model (CBM). However, CBMs are very difficult to train, often needing 20 years’ worth of training just to start being useful, while consuming numerous inputs and producing only dismissive waves and occasional grunts as output.

This book is mostly about Transformers, so the interested reader is referred to the research literature for these architectures, or a psychology textbook for the CBMs. As a general rule, there are so many research papers being written about AI that there are literally exceptions to everything. But those intrepid researchers are doing a great service to programmers by giving us lots of gritty algorithms to code up.

References

1. Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu, 2022, *A survey of transformers*. *AI Open*, <https://arxiv.org/abs/2106.04554> (An extensive and useful survey of Transformer architectures.)
2. Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler, 2022, *Efficient transformers: A survey (v2)*, arXiv preprint arXiv:2009.06732, <https://arxiv.org/abs/2009.06732>
3. Ce Zhou, Qian Li, Chen Li, Jun Yu, Yixin Liu, Guangjing Wang, Kai Zhang, Cheng Ji, Qiben Yan, Lifang He, Hao Peng, Jianxin Li, Jia Wu, Ziwei Liu, Pengtao Xie, Caiming Xiong, Jian Pei, Philip S. Yu, Lichao Sun, May 2023, *A Comprehensive Survey on Pretrained Foundation Models: A History from BERT to ChatGPT*, <https://arxiv.org/abs/2302.09419>
4. Q Fournier, GM Caron, D Aloise, 2023, *A practical survey on faster and lighter transformers*, ACM Computing Surveys, <https://dl.acm.org/doi/abs/10.1145/3586074>, <https://arxiv.org/abs/2103.14636>

5. Xipeng Qiu, TianXiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang, 2020, *Pre-trained Models for Natural Language Processing: A Survey*, SCIENCE CHINA Technological Sciences 63, 10 (2020), 1872–1897. <https://doi.org/10.1007/s11431-020-1647-3>, <https://arxiv.org/abs/2003.08271> (Good survey of Transformer architectures in 2020.)
6. Y Chang, X Wang, J Wang, Y Wu, K Zhu, 2023, *A survey on evaluation of large language models*, arXiv preprint, <https://arxiv.org/abs/2307.03109>
7. N Elhage, N Nanda, C Olsson, T Henighan, N Joseph, B Mann, A Askell, Y Bai, A Chen, T Conerly, et al. 2021, *A mathematical framework for transformer circuits*. <https://transformer-circuits.pub/2021/framework/index.html> (Detailed theoretical examination of how various Transformer components work.)
8. W Li, H Hacid, E Almazrouei, M Debbah, 2023, *A Comprehensive Review and a Taxonomy of Edge Machine Learning: Requirements, Paradigms, and Techniques*, AI 2023, 4(3), 729-786, <https://www.mdpi.com/2673-2688/4/3/39> (Extensive survey related to optimizing on edge devices.)
9. J Zhong, Z Liu, X Chen, Apr 2023, *Transformer-based models and hardware acceleration analysis in autonomous driving: A survey*, <https://arxiv.org/abs/2304.10891>
10. Y Li, S Wang, H Ding, H Chen, 2023, *Large Language Models in Finance: A Survey*, PDF: https://www.researchgate.net/profile/Yinheng-Li/publication/374546790_Large_Language_Models_in_Finance_A_Survey/links/6523988afc5c2a0c3bc534fc/Large-Language-Models-in-Finance-A-Survey.pdf
11. Maurizio Capra, Beatrice Bussolino, Alberto Marchisio, Guido Masera, Maurizio Martina, Muhammad Shafique, 2020, *Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead*, <https://ieeexplore.ieee.org/iel7/6287639/6514899/09269334.pdf>, <https://arxiv.org/abs/2012.11233>
12. Kai He, Rui Mao, Qika Lin, Yucheng Ruan, Xiang Lan, Mengling Feng, Erik Cambria, Oct 2023, *A Survey of Large Language Models for Healthcare: from Data, Technology, and Applications to Accountability and Ethics*, <https://arxiv.org/abs/2310.05694>
13. Minghao Shao, Abdul Basit, Ramesh Karri, Muhammad Shafique, *Architectures: Trends, Benchmarks, and Challenges*, https://www.researchgate.net/profile/Minghao-Shao2/publication/383976933Survey_of_different_Large_Language_Model_Survey_of_different_Large_Language_Model_Architectures_Trends_Benchmarks_and_Challenges/links/66e2d320f84dd1716ce79f85/Survey-of-different-Large-Language-Model-Architectures-Trends-Benchmarks-and-Challenges.pdf

14. Tianyu Ding, Tianyi Chen, Haidong Zhu, Jiachen Jiang, Yiqi Zhong, Jinxin Zhou, Guangzhi Wang, Zhihui Zhu, Ilya Zharkov, Luming Liang, 18 Apr 2024 (v2), *The Efficiency Spectrum of Large Language Models: An Algorithmic Survey*, <https://arxiv.org/abs/2312.00678>
15. Johannes Schneider, 1 Aug 2024, *What comes after transformers? -- A selective survey connecting ideas in deep learning*, <https://arxiv.org/abs/2408.00386>
16. Rob Toews, Sep 3, 2023, *Transformers Revolutionized AI. What Will Replace Them?* Forbes, <https://www.forbes.com/sites/robtoews/2023/09/03/transformers-revolutionized-ai-what-will-replace-them/>

14. Training and Fine-Tuning

Training Options

It's easy to make a small fortune in LLM model training these days. You start with a big fortune, and then do training.

If you want a new model, and none of the off-the-shelf commercial or open source models are good enough, here are your basic options for training a smarter model:

- Train a new foundation model from scratch.
- Fine-tuning (FT) of an existing model.
- LoRA-based fine-tuning
- Retrieval-Augmented Generation (RAG) using a document database.

It's a great list. Let's start at the top.

Training a Foundation Model

Training your own foundation model from scratch is kind of expensive. and I don't really recommend you try to train your own foundation model at home. Also, why bother? The top LLMs are so good these days, both commercial or open-source models, and you can rent or buy. Hence, training a new model from scratch is probably relegated to the non-language type ML projects, using your own proprietary non-text data.

But don't listen to me. If you really have a nine-figure funding round, then go ahead and train your own foundation LLM. Send me a selfie from your private jet.

On the other hand, fine-tuning an existing model is cheaper. And you can use LoRA fine-tuning, which is even cheaper, too. RAG is cheaper still (probably), but it's not even a type of training, so it should really be banned by the European Union for false advertising.

Still reading this, which means you still want to do training? Which is fine, I guess, provided the GPU hosting cost isn't coming out of your pay packet.

In terms of optimizing a training project, here are some methods that might be worth considering:

- Choose smaller model dimensions (smaller is cheaper, but bigger is smarter).
- Evaluate open-source vs commercial models.
- Evaluate fine-tuning (FT) vs Retrieval-Augmented Generation (RAG).
- Quantized models (“model compression” methods).
- Knowledge distillation (train a small model using a large “teacher” model).
- Dataset distillation (train a small model using auto-generated outputs from a large model).

Fine-Tuning

Hopefully, you’ve come to your senses, and dropped the idea of training a foundation model from scratch. But then, why not fine-tune one?

If you want to do fine-tuning, here is a checklist of things to consider:

- Data availability — fine-tuning needs some specialized data, preferably some that only you own.
- Good data — having bad data is worse than no data.
- Prompt engineering versus fine-tuning — goals of fine-tuning can sometimes be more easily achieved via tweaks in prompt engineering (e.g., brand voice with prepended custom instructions).
- RAG — ensure you’ve considered RAG versus fine-tuning.
- Cost — various types of fine-tuning have different cost-to-accuracy profiles.

How does fine-tuning work? An existing foundation model is trained with new materials using the standard AI training methods. The use of extra specialist text to further train a model is called “fine-tuning.” This is a longstanding method in AI theory and fine-tuning can be performed in all of the major AI platforms. In the fine-tuning approach, the result of the re-training is that proprietary information about your products is all “inside” the model.

Types of fine-tuning. There are multiple ways to do fine-tuning, but the basic categories are:

- Full-parameter fine-tuning
- Parameter-efficient fine-tuning (PEFT)

Full-parameter fine-tuning is where you tweak every parameter in the entire base model. This is like continuing the original training of a foundation model, but with a specialized fine-tuning data set. It's also quite costly in terms of GPU juice to do full training, although the idea is to use a lot less data than would be required for training the original foundation model.

Parameter-efficient fine-tuning or PEFT is the new way to do fine-tuning. The idea is to “freeze” most of the parameters in the main foundation model, and tweak a subset of them. This makes updating the model weights much faster.

The most popular way to do PEFT is called Low-Rank Adapters, and has the cute name of LoRA. This uses a trick with low-rank matrices and fancy matrix algebra, so that you can train only the parameters in a pair of much smaller matrices, which optimizes the cost of fine-tuning. LoRA also has a big advantage during inference in that it adds zero cost at runtime, except for an initial setup cost when loading the LoRA adapter. The most common strategy is “multi-LoRA” to have multiple specialized versions of a base model using a set of fine-tuned LoRA adapters.

Fine-Tuning Algorithm

The training algorithm used for fine-tuning is conceptually similar, regardless of whether you're doing full-parameter fine-tuning, or PEFT, or LoRA, or the many other sub-variants. You grab some data and run it through a training engine that supports whatever method you want.

The general training algorithm at a very high level is as follows:

- (a) Split the training data 80/20 (sometimes 90/10) into data to train with (training dataset) and data to evaluate the result of the training (validation dataset). If you have enough training data, use multiple training and validation datasets.
- (b) Feed each input into the network, compare with the answer using the “loss function” to generate an “error”, and using the error, tweak the weights according to the learning rate.
- (c) After all the 80% of data is fed in, use the validation dataset to evaluate the new model's performance. This is using new data that the model has not seen yet, also in a question and expected response format.
- (d) Based on the evaluation, you can accept the model or make major changes. For example, if you give it totally unseen data (i.e., the 20%) and

it only responds correctly 50% of the time, you need to decide whether to continue with the next training dataset, or if it's time to redesign the model and try again.

If the model performs poorly, you have to allocate blame: if the training data is good, if the model's structure is correct, if the loss function is correct, if the learning rate for incremental changes to the weights on each iteration are aggressive enough, or too aggressive, if biases are wrong, etc. To do this, tweak the model meta-parameters (e.g., number of layers, number of nodes per layer, etc.) or change the training algorithm meta-parameters (e.g., learning rate), and go back to the first step and start over.

This is only a top-level outline of an LLM training algorithm. There are many extra improvements and finesses to get to a fully advanced fine-tuning algorithm.

Training Data for Fine-tuning

One of the biggest obstacles to a fine-tuning project is getting enough data. Many projects where fine tuning seems like a good idea are scuttled when there is no critical mass of data with which to train. Fine-tuning usually requires more data than RAG.

For fine-tuning data to be viable, it usually needs to have:

- (a) Several cases of every concept you want to teach it, with both input and expected output. Depending on the NN architecture, it may also need a score indicating how good the output is.
- (b) Corner cases and extra data to capture subtle details or complexities.
- (c) Held-back extra cases which are not used for training, but are used to evaluate the state of the training.

Gathering the data is likely the hardest part of training. And the more training iterations you need to do, the more data you need. Training data management is mostly a non-coding task, involving processing of the data files, such as chunking, organizing, indexing, and generating embeddings. It's arduous to some extent, but not high in technical difficulty.

Model-Based Training

Another way to do training is to have it talk to another previously trained system. Knowledge distillation is one of these techniques, which is available already in major AI frameworks, and has a high level of sophistication. Another simpler method is to train a new model on the prompt-answer pairs from another large model, which doesn't have a consistent name in the literature, but is sometimes called "dataset distillation" or "downstream data."

If you like computers, then you can probably think of a few ways to create your own data. Alas, some other smart bunnies have beat you to it! The general idea with using computer-generated input data is called "synthetic data," but that doesn't always mean data from another model's output.

What is LoRA?

LoRA is a type of Parameter-Efficient Fine-Tuning (PEFT). What that means is that instead of "full parameter" fine-tuning, we're only going to train a small subset. The idea is to "freeze" the main model, and only train a small set of differences.

Full-parameter fine-tuning is where we continue the whole shebang of training, as if we were continuing our training from scratch. And that is expensive and very memory-intensive, because we have to update every single parameter. It's also inefficient later if we want to have more than one of these, because we have a massive fine-tuned model that's different each time.

Hence, the advantages of LoRA are:

- Efficient fine-tuning
- Efficient inference if using more than one (i.e., multi-LoRA)

How does LoRA work? The idea of fine-tuning is that we want to change a subset of numbers in a matrix. So, instead of changing the weights in the original model matrix, we could (conceptually) generate a separate "differences" matrix. Then later, in the inference phase, we could load the main model matrix and the difference matrix, add them together, and we've got a final model. In practice it's more complicated, but assume we can generate a difference matrix.

But how is that more efficient? Storing two big matrices, an original and a difference matrix, isn't better than just storing one.

Well, it's faster because instead of storing a big difference matrix, we use "low-rank decomposition" of matrices, which is a type of matrix factorization, to generate two much smaller matrices. The idea is that these two smaller matrices re-generate the big matrix when multiplied together.

Matrix decomposition. If you remember matrix multiplication of non-square matrices from High School, the dimensions work like this:

$$M \times N \text{ times } N \times P = M \times P$$

What happened to N ? The inner dimension, N , is not used in the final size.

Hence, if we want the final matrix to be 1000x1000, we have $M=P=1000$. Yes, I know, in practice it would be powers of two sizes, but that makes the math too hard for me.

Now, the funny thing is that to create our $M \times P$ size matrix, you can see it doesn't matter what N is, because it disappears. The smallest case is to use $N=1$. Hence, we have an 1,000x1 matrix (a column) times a 1x1,000 matrix (a row).

This is very efficient! Instead of 1,000,000 weights in the big matrix, we have only 2,000 weights in a column and a row. Hence, we have very few weights to train. As a percentage, that's very tiny, like 2% or 0.2% or something. Where's my AI calculator?

Unfortunately, this matrix trick doesn't work perfectly. Not all 1,000x1,000 matrices can be exactly decomposed into a column times a row. Instead, we have to find the closest that we can, which means that low-rank decomposition is an approximation.

LoRA adapters. In most LoRA implementations, we don't use $N=1$. Higher values of N make the approximations better, and values like $N=8$ to $N=256$ are more typical. But the number of parameters in a LoRA adapter is still much less than the quadratic size of the large matrix.

So, the "LoRA adapter" is actually just a set of two small low-rank matrices. It's much less to store and less expensive to load into memory.

The whole thing is a little more complicated than that because every layer is different. We need a set of low-rank matrices for each layer. Hence, a LoRA adapter is a set of per-layer paired low-rank matrices.

Also, there can be more than one matrix per layer. There are variations in LoRA not just in the matrix dimensions, but also in terms of which parameters we want to target within a layer: attention weights and/or FFN weights. Generally, the collective wisdom is that attention weights are preferred as a first priority for LoRA weights, and maybe do FFN weights if you want extra accuracy (with extra cost). If you do both, it's really two LoRA adapters per layer of the model, which means four low-rank matrices per layer.

Inference. The act of initialization of an inference server with the fine-tuned model is slightly slower, due to an additional setup step from loading and installing the LoRA parameters in its low-rank matrices. But once it's loaded, we have zero extra latency in inference. The extra parameters in LoRA adapters are no longer used during inference.

Inference works by loading the main foundation model and also the LoRA adapters into memory. The pairs of decomposed matrices are multiplied together to create a large “difference” matrix, and this is simply added to the main foundation model. After this step, the LoRA adapters are no longer needed, and only the modified foundation model is used for inference, in the normal manner. Neither the low-rank matrices nor the computed difference matrix need remain in memory. Note that practical coded engines don't actually ever store the big difference matrix in memory, through the magic of kernel fusion optimizations.

Multi-LoRA

The idea of multi-LoRA arises if we want two or more fine-tuned models. Note that multiple pairs of low-rank matrices, one per layer, is still considered to be a single LoRA adapter. Multi-LoRA is only when you do the whole fine-tuning twice, and you get two or more LoRA adapters.

The advantages of multi-LoRA include:

- Multiple fine-tuned models that are specialized off one foundation model.
- Fine-tuning costs are lower than full-parameter fine-tuning.
- Inference extra cost is literally zero.
- Switching between multiple fine-tuned models is efficient.

To the last point, let's see how “hot swapping” of multiple LoRA adapters works. To “unload” a LoRA adapter, the process is similar to loading, but we use subtraction of the differences rather than addition. Simply load the two low-rank matrices of the LoRA adapter and then:

- (a) multiply them together to re-create the differences matrix, and
- (b) subtract (rather than add) these values from the larger matrices, thereby yielding the original values of the foundation model.

In practice, rather than creating a large difference matrix in memory, these two operations can be merged using kernel fusion. In fact, the third operation of loading the next LoRA adapter can also be fused into a triple operation.

This multi-LoRA method has actually been chosen for Apple Intelligence in its on-device version. Technically, since they're also using quantization, it's probably using Quantized LoRA (QLoRA) and multi-QLoRA.

Apple wanted different models for each specialized use case, with dozens of different AI tasks that need a model. but didn't want to store multiple large foundation models. Hence, it uses a multi-LoRA architecture, where each specialized model is generated via one foundation model (for on-device, it's understandably not an LLM but a small language model of size 3B), and one of many LoRA adapters.

Each LoRA adapter is much smaller than the 3B foundation model, with a size in the “tens of millions” of parameters. Hence, the LoRA adapters can be swapped on-the-fly at relatively low cost every time your iPhone wants to do a different AI task.

Training FAQs

What is pre-training? It's not a specific type of training algorithm, but just means that you've already trained the model. This term mostly appears in Generative Pre-trained Transformers (GPT), which you may have heard of.

It's common for a commercial service to offer access to a pre-trained model. For example, the OpenAI API allows you to send queries to the pre-trained GPT models, which have a broad level of trained capabilities. Similarly, there are numerous open source pre-trained models available, such as the Meta Llama2 model and various smaller ones.

What is re-training? Again, this isn't really a technical term. It usually just means the same as fine-tuning.

What is knowledge distillation? Knowledge distillation (KD) is an optimization technique that creates a smaller model from a large model by having the large model train the small model.

Hence, it's a type of “auto-training” using a bigger teacher model to train the smaller student model. The reason it's faster is that once the training is complete, you use only the smaller student model for processing user queries, and don't use the bigger model for inference at all.

Distillation is a well-known and often used approach to save cost but retain accuracy. For example, a large foundation model usually has numerous capabilities that you don't care about. There are various ways to use “distillation” to have the large model teach the smaller model, but within a subset of its capabilities. There are ways to share inference results and also more advanced internal weight-transfer strategies.

What is model initialization? That's where you use `malloc` to allocate a memory block that exceeds the capacity of your machine. Umm, no. Model initialization is an important part of the training algorithm, and as you have probably already guessed, this refers to the start of training.

Since training creates a smart model by updating parameters incrementally by small amounts, it works better if the parameters are already close to where they need to be. So, you don't just start training with all the model full of zeros. Instead, you try to “jumpstart” the process with better initialization. However, it's far from clear what the best choice of initialization values should be, and there are lots of research papers on this topic.

References

1. Yarally T, Cruz L, Feitosa D, et. al., 2023, *Uncovering energy-efficient practices in deep learning training: Preliminary steps towards green AI*, International Conference on AI Engineering - Software Engineering for AI (CAIN), <https://arxiv.org/abs/2303.13972>
2. A. Apicella, F. Donnarumma, F. Isgro, and R. Prevete, 2021, *A survey on modern trainable activation functions*, Neural Networks, vol. 138, pp.14–32, <https://arxiv.org/abs/2005.00817> (Extensive survey all about training with activation functions, e.g., RELU, Swish, Maxout, leaky RELU.)
3. R. Immonen, T. Hämmäläinen et al., 2022, *Tiny machine learning for resource-constrained microcontrollers*, Journal of Sensors, vol. 2022, <https://www.hindawi.com/journals/js/2022/7437023/> (Survey of on-device training for TinyML/edge computing.)
4. P Freire, E Manuylovich, JE Prilepsky, SK Turitsyn, 2023, *Artificial neural networks for photonic applications—from algorithms to implementation: tutorial*, Advances in Optics

and Photonics, Sep 2023, https://opg.optica.org/directpdfaccess/f0ae8746-2f89-4ac4-bb598eda29c7977c_539680/aop-15-3-739.pdf?da=1&id=539680&seq=0&mobile=no (Large survey covering many aspects of the future of training optimization.)

5. Marcos Treviso, Tianchu Ji, Ji-Ung Lee, Betty van Aken, Qingqing Cao, Manuel R. Ciosici, Michael Hassid, Kenneth Heafield, Sara Hooker, Pedro H. Martins, Andre F. T. Martins, Peter Milder, Colin Raffel, Edwin Simpson, Noam Slonim, Niranjana Balasubramanian, Leon Derczynski, Roy Schwartz, Aug 2022, *Efficient Methods for Natural Language Processing: A Survey*. arxiv:2209.00099[cs], August 2022. <http://arxiv.org/abs/2209.00099>
6. Epoch AI, 2024, *How Much Does It Cost to Train Frontier AI Models?* <https://epochai.org/blog/how-much-does-it-cost-to-train-frontier-ai-models>
7. Ben Cottier, Robi Rahman, Loredana Fattorini, Nestor Maslej, David Owen, 31 May 2024, *The rising costs of training frontier AI models*, <https://arxiv.org/abs/2405.21015>
8. Bender, Emily M.; Gebru, Timnit; McMillan-Major, Angelina; Shmitchell, Shmargaret (2021), *On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?*, FAccT '21. Association for Computing Machinery. pp. 610–623.
9. Apple, June 2024, *Introducing Apple's On-Device and Server Foundation Models*, <https://machinelearning.apple.com/research/introducing-apple-foundation-models>
10. Sachin Mehta, Mohammad Hossein Sekhavat, Qingqing Cao, Maxwell Horton, Yanzi Jin, Chenfan Sun, Iman Mirzadeh, Mahyar Najibi, Dmitry Belenko, Peter Zatloukal, Mohammad Rastegari, 22 Apr 2024, *OpenELM: An Efficient Language Model Family with Open-source Training and Inference Framework*, Apple Research, <https://arxiv.org/abs/2404.14619> Code: <https://huggingface.co/apple/OpenELM>

15. RAG Architectures

What is RAG?

Retrieval Augmented Generation (RAG) is a technique of merging external data sources with AI-based query answering. Really, it's just a fancy way to say: look up a database and then get the LLM to summarize it. When it works well, RAG combines the speed of searching an information database with the elegance and fluent writing from an LLM.

RAG is an architecture whereby the AI is integrated with an external document search mechanism. There are three components:

- Retriever
- Generator
- Datastore

The “retriever” component looks up the user’s query in a datastore of documents, using either keyword search or vector search. This is effectively a search component that accesses a database and finds all the related material. Typically, it returns excerpts or snippets of relevant text, rather than full documents.

The role of the “generator” component in RAG is to receive the document excerpts back from the retriever, and collate that into a prompt for the AI model. The snippets of text are merged as context for the user’s question, and the combined prompt is sent to the AI engine. Hence, the role of the generator is mainly one with prompt engineering and forwarding requests to the LLM, and it tends to be a relatively simple component.

The datastore could be a classic database (e.g., SQL or MongoDB) with keyword lookup or a vector database with semantic lookup. The use of semantic lookup can give more meaningful document search results, with better model answers, but requires two additional steps. Firstly, the user’s query must be converted into a vector format that represents its semantic meaning (called an “embedding”). Secondly, a vector database lookup is required using that embedding vector. There are various commercial and open source vector databases available.

How does RAG work? As an example, let us assume you are creating an AI system that answers questions about your company’s latest product offerings, perhaps with details already on your website, or with nice glossy PDF marketing documents that describe the products. How do you get the AI system to answer questions about those products?

In the RAG approach, the model itself doesn’t know about the data or documents with details of your newest products. Instead, the engine in a RAG architecture knows how to:

- (a) search company documents for the most relevant ones (retriever), and
- (b) summarize relevant parts of the documents into an answer for the user’s question (generator).

Unlike fine-tuning, the RAG approach does not use your company documents as training data that you cram into an updated model. Instead, the documents are a source of input data that is integrated via a retriever search component, and sent as input to the AI engine using an unchanged model. RAG may require some “prompt engineering” that combines the document search results and a user’s query, but the foundational model itself does not change.

The RAG component typically consists of a datastore of documents and a search mechanism. A typical setup would be a vector database containing documents that are indexed according to a semantic vectorization of their contents. The search mechanism would first vectorize the incoming query into its semantic components, then find the documents with the “nearest” matching vectors, which indicates a close semantic affinity.

Document Snippets. Typically, the results from the “retriever” component would be small sections or snippets of documents, rather than full-size documents. The reason small sections are desirable is because:

- (a) it would be costly to make an AI engine process a large document, and
- (b) it helps the AI find the most relevant information quickly.

The retrieved snippets or portions of documents would be returned to the AI. They would be prepended to the user’s search query as “context” for the AI engine. Prompt engineering would then be used to ensure that the AI engine responds to the user query using information from the context document sections.

RAG Project Design

General implementation steps in a typical RAG project are as follows:

1. Data identification. Identify the proprietary data you want to make the RAG system an expert on. This will also mean working out how to ingest the data. For example, it might be a JIRA customer support database, a Confluence space, or a directory of PDF files on disk,

Generally, any type of “knowledge base” can be used. Some common internal examples are product documentation, HR benefits information, company policies and other internal training materials. External examples are ticketing systems (carefully scrubbed), customer product documentation, and support information.

Note that the base of knowledge will change and get bigger over time. It is necessary to ponder the refresh operation, because purging and starting over can be expensive, especially if embeddings are being calculated.

2. Sizing. Determine the sizes of a “chunk” of data. The context size of the model matters here, because the chunks need to be substantially smaller than the context size. When a user asks a question, the system will be given 3-5 chunks of knowledge excerpts that are pertinent to the question. These snippets will be combined with the question. Even worse, if a back-and-forth dialog needs to occur between the users and the model, extra room needs to be available in the context size for follow-up questions.

Note that there can be two context sizes in play. The context size of the LLM which will generate the answer, and the context size of the LLM producing the embeddings. It’s common for these to be different, with the embedding generation typically using a “cheaper” engine. If a RAG system has previously been implemented and is now being revised, you should recheck the initial assumptions, because model context sizes have increased and model costs have improved.

3. Splitting sections. Determine how to “chunk” the data. A boundary for the chunk needs to be identified, which might be sections after a heading if it’s a web page. Larger sections might need the heading and one or two paragraphs in each chunk. Content from a bug tracking system might use the main description as one chunk, the follow-up comments as another chunk or multiple chunks, and the identified solution as another chunk. It’s often beneficial to “overlap” chunks to hide the fact that chunking occurs.

4. Text-based database upload. Each chunk needs to be organized and stored in a text-based search engine, like Elastic Search, Azure Cognitive Search, etc. For documents and web pages, the organization can be minimal, but for a ticketing system with a complex structure (e.g., problem descriptions, comments, solutions), it all needs to be related somehow.

5. Vector database upload. The embedding for each chunk needs to be calculated and stored in a Vector Database. You can think of the embedding as a “summarization” of the chunk from the perspective of the model. The vector returned is typically multi-dimensional with 50+ dimensions. Each dimension represents a single concept in the contents. The idea is that from the model’s perspective, similar content would produce similar vectors. A vector database can quickly find chunks with related data using vector lookup.

6. Optimizations. The embeddings can sometimes be calculated using a lazy evaluation algorithm (avoiding the cost of embeddings calculations for never-used documents), but this can also slow down inference, and full precomputation is faster. The model used for calculating embeddings does not need to be the same as the model answering the questions. Hence, a cheaper model can be used for embedding, such as GPT-Turbo, whereas GPT-4 could be used to answer questions.

RAG Detailed Algorithm

The RAG algorithm is not training. Prompt engineering gives the model all the content it needs to answer the question in the prompt. You are effectively using the LLM to take your content, mix it with its own trained knowledge (in a limited way), eloquently answer the question, and then perhaps converse on it a little.

The basic technical algorithm flow for a user request in a RAG architecture can be something like this:

- a. Receive the user’s question (input).
- b. Use the user’s question to do a text-based (keyword) search on the index and get the top X hits (of documents or snippets).
- c. Calculate the “embedding” for the user’s question (a vector that shows its semantic meaning in numbers).
- d. Calculate the embeddings for the top X hits (from text search) and add these embeddings vectors to the vector database.

- e. Do a vector search on embeddings and get the top Y hits.
- f. Filter top X hits (text-based) and top Y hits (vector-based) to find overlaps, this overlap represents the best text-based and vector-based hits. If there is no overlap, select some from each.
- g. Combine the top hits with any summarization from previous questions.
- h. Get the contents from the top hits and use prompt engineering to create a question something like:

“Given <summary>, <chunk 1>, <chunk 2>, <chunk 3>, answer <question from user>. Only respond with content from the provided data. If you do not know the answer, respond with I do not know. Cite the content used.”

- i. Send the prompt to the LLM, and receive the answer back from the LLM.
- j. Resolve any citations in the answers back to URLs the end user can click on, e.g., Confluence page, Jira ticket/comment/solution, etc.
- k. Summarize the conversation to date using the model (i.e., context for any subsequent questions).
- l. Send back answer + summarization (perhaps encoded). The idea is the encoded summarization will not be shown for this answer, but will only be used internally by the RAG components for follow-up questions.
- m. The client or caller is responsible for context management, which means ensuring that conversations end quickly and new topics result in new conversations. Otherwise, the context fills up quickly, the LLM forgets what it’s already said, and things gets confusing.

The above algorithm is thorough in generating two sets of hits (top X and top Y). It’s not strictly necessary to do two searches (one with text keywords and one with vector embeddings), as often vector embeddings are good enough. Alternatively, text-based keyword searches are often cheaper, and vector lookups could be skipped. At the end of the day, the chunks most likely to contain answers to the questions are being sought.

Special Cases

If only that were all there was to code for a RAG system. Here are some more special cases to consider:

- No chunks are returned by the retriever (two cases: keyword retrieval and vector database retrieval).
- Returned chunks are scored so low by the reranker that it's effectively the same as having no chunks (i.e., all are scored below a relevance threshold).
- The query requires a “tool” or “function call” (e.g., a clock is needed to answer: “What time is it?”).
- The query requires an external data source, such as a web search, which may or may not be supported by your RAG system.

My brain is in pain. How many distinct input cases is that?

There's not one right answer in such cases. Handling zero chunks returned could be handled by simply bailing out with a fixed unfriendly message (“Error: no chunks”) or a friendlier LLM response based on a different prompt template (“I don't know the answer to that, but here's a quote from Hamlet.”)

On the other hand, maybe you still want to sell your customer something even if you've got nothing. For example, the RAG system is a clothing advisor, and the user's query is:

“What should I wear today?”

That's unlikely to have any good matches to specific RAG documents about clothing products. The simple and efficient solution would be to just give a generic sales spiel response, which could be a canned document.

But if personalization was desirable (yes), and a weather tool was available (maybe), and your LLM has been trained to recognize day-specific or season-specific searches (probably not), then the weather tool could be called, and the returned weather conditions could be substituted into the query. With better keywords, a good selection of clothing could then be retrieved from the RAG store. The query would effectively become:

“What should I wear when it is chilly with a chance of rain?”

Note that this idea actually needs two tools. To get the weather, you first need to know the website user's general location from their IP address or other cookies, which is another tool.

Vector Databases

Some well-known vector databases are Pinecone, Milvus, Chroma, Weaviate, Qdrant, FAISS. General database systems like Elastic, Redis and Postgres (amongst others) also have some vector capabilities.

The main feature of a vector database is to be able to a vector-based query. Basically, you are looking for how close vectors are to each other in N-dimensional space. Cosine similarity is a common comparison, but other algorithms include nearest neighbors, least squares, and more.

Speed of the vector database lookup is obviously important for fast inference latency. Each vector also needs to be associated with the actual data or document in the database.

RAG Data Management

When compiling data for a RAG implementation, you do have to go to some lengths to make sure your database of content has the data in it to answer questions. But it does not need to be a great number of samples. In fact, even a single hit with one chunk of data is enough for the LLM to form an answer. With any RAG, if you search in the text-based index or the vector index and do not get good hits, it will produce poor results. Also, unlike fine-tuning data, RAG does not require question and answer type content, but only documents that can be searched.

The data is very important to a successful RAG project. RAG systems are not much different conceptually to searching Google for your own data. At the end, you have something that can produce eloquent writing better than 90% of the population.

Keyword Lookup Algorithms

As an example of the keyword lookup component, let's examine the "Best Match 25" or BM25 algorithm. This is a longstanding algorithm for keyword search that has preceded much of the AI work, and is simpler to understand than embedding-based vector research. Keyword search algorithms such as BM25 can be used as RAG-based keyword retrievers.

At a high level, BM25 consists of the following steps:

1. Process the query to remove the small words (e.g., “the,” “and,” “this”, “that,” etc.), leaving the core words of the query.
2. Use a reverse index to find all the documents (chunks) that match each word.
3. Score and rank these chunks or documents.
4. Return the Top 25 as the keyword lookup results.

The score for ranking is based on:

- How many of the query terms appear in a chunk/document.
- How frequently each word appears in the chunk/document.
- How rare each word is across all documents.
- Length of the chunk/document.

All of these tests can be brute-force precalculated for each word, so only the combination of each metric needs to be calculated at runtime.

Presumably, there could be other weights in the calculation too, such as there may be a “vocabulary” specific to your knowledge base which might have a better affinity to specific documents for example.

This overall algorithm is used as the keyword-based retriever component of the RAG architecture. At the very end of this algorithm, the “top 25” chunks or documents are passed back to the LLM as the result of the keyword lookup, which are then incorporated into the overall RAG algorithm.

Fine-Tuning vs RAG

If you want an AI engine that can produce answers based on your company’s product datasheets, there are two major options:

- Fine-Tuning (FT): re-train your foundation model to answer new questions.
- Retrieval-Augmented Generation (RAG) architecture: summarize excerpts of documents using an unchanged foundation model.

The two basic approaches are significantly different, with completely different architectures and a different project cost profile. Each approach has its own set of pros and cons.

Spoiler alert! RAG and fine-tuning can be combined.

Advantages of RAG. The RAG architecture typically has the following advantages:

- Lower up-front cost
- Flexibility
- Up-to-date content
- Access external data sources and/or internal proprietary documents
- Content-rich answers from documents
- Explainability (citations)
- Personalization is easier
- Hallucinations less likely (if retriever finds documents with the answer)
- Scalability to as many documents as the datastore can handle.
- RAG is regarded as “moderate” difficulty in terms of AI expertise.

The main goal of RAG is to avoid the expensive re-training and fine-tuning of a model. However, RAG also adds extra costs in terms of the RAG component and its database of documents, both at project setup and in ongoing usage, so it is not always a clear-cut win.

In addition to cost motivations, RAG may be advantageous in terms of flexibility to keep up-to-date with content for user answers.

With a RAG component, any new documents can simply be added to the document datastore, rather than each new document requiring another model re-training cycle. This makes it easier for the AI application to stay up-to-date with current information included in its answers.

Disadvantages of RAG. The disadvantages of RAG where the underlying model is not fine-tuned, include:

- Architectural change required (retriever component integrated with model inference).
- Slower inference latency and user response time (extra step in architecture).
- Extra tokens of context from document excerpts (slower response and increased inference cost).
- Extra ongoing costs of retriever component and datastore (e.g., hosting, licensing).
- Larger foundation model required (increases latency and cost).
- Model's answer style may not match domain (e.g., wrong style, tone, jargon, terminology).
- Re-ingestion may be required for data updates.

Data needs to be ingested periodically stay up to date, which is also true of FT. With RAG, any major changes to the source data may require it *all* to be ingested again. Even if the data stays the same, but it's been reorganized, the data needs to be ingested again. If you're not careful with revisions, citation links may resolve to 404 pages. Technically, this is also an advantage of RAG, because the ingest is not terribly expensive compared to fine-tuning.

Penalty for unnecessary dumbness. There's an even more fundamental limitation of RAG systems: they're not any smarter than the LLM they use. In fact, a typical RAG system:

- Remembers nothing
- Learns nothing

The overall RAG implementation relies on the LLM for basic common sense, conversational ability, and simple intelligence. It extends the LLM with extra data, but not extra memory or extra skills.

The RAG system does not remember anything about the chunks it sees. Rather, it needs to scan them again for every query. It has no memory of having seen them before, except maybe a KV cache, which helps it run faster but not smarter.

The LLM does not learn anything from a chunk that puts its knowledge into a higher plane. For example, if you ask a RAG system with a chunked version of a hundred programming textbooks, I doubt you could ask it to generate a Snake Game. However, you could certainly ask it about the syntax of a `switch` statement in the language, because that's probably memorized inside the model weights.

With a RAG architecture, the model has not “learned” anything new, and you have only ensured it has the correct data to answer questions. For example, if you asked about how a “switch” statement works, the hope is that the retriever finds the chunks from the “Switch” sections of the programming books, and not from five random code examples that used the *switch* statement. This is all dependent on the text-based keyword indexing and how well the semantic embedding vectors work. The “R” in RAG is “Retrieval” and it’s very dependent on that.

It’s also somewhat dependent on the initial ingest of the RAG data. For example, if done well, the ingest logic and chunking could notice text versus code segments in the original content and then chunk or categorize it appropriately.

There is some research on “continuous learning” or “continuous adaptation” or “incremental learning” on the horizon. Future RAG systems might retain some knowledge from the text chunks of documents (or conversations with users). Maybe some of the faster fine-tuning algorithms, such as Parameter-Efficient Fine-Tuning (PEFT) or Low-Rank Adapters (LoRA), will improve RAG’s learning capabilities in the future.

Advantages of fine-tuning. The main advantages of a fine-tuning architecture over a RAG setup include:

- Style and tone of responses can be trained — e.g., positivity, politeness.
- Use of correct industry jargon and terminology in responses.
- Brand voice can be adjusted.
- No change to inference architecture — just an updated model.
- Faster inference latency — no extra retriever search step.
- Reduced inference cost — fewer input context tokens.
- No extra costs from retriever and datastore components.
- Smaller model can be used — further reduced inference cost.
- You’re more likely to get free lunches and access to the company jet on weekends.

Fine-tuning is not an architectural change, but is an updated version of a major model (e.g., GPT-3), whereas RAG is a different architecture with an integration to a search component that accesses an external knowledge database or datastore and returns a set of documents or chunks/snippets of documents.

Disadvantages of fine-tuning. The disadvantages of a fine-tuning architecture without RAG include:

- Training cost — up-front and ongoing scheduled fine-tuning is expensive.
- Outdated information used in responses.
- Needs a lot more proprietary data than RAG.
- Training data must be in a paired input-output format, whereas RAG can use unstructured text.
- Complexity of preparing and formatting the data for training (e.g., categorizing and labeling).
- No access to external or internal data sources (except what it's been trained on).
- Hallucinations more likely (if it hasn't been trained on the answer).
- Personalization features are difficult.
- Lack of explainability (hard to know how the model derived its answers or from what sources).
- Poor scalability (e.g., if too many documents to re-train with).
- Fine-tuning (training) is regarded as one of the highest difficulty-level projects in terms of AI expertise.

The main disadvantage of fine-tuning is the compute cost of GPUs for fine-tuning the model. This is at least a large up-front cost, and is also an ongoing cost to re-update the model with the latest information. The inherent disadvantage of doing scheduled fine-tuning is that the model is always out-of-date, since it only has information up to the most recent fine-tuning. This differs from RAG, where the queries can respond quickly using information in a new document, even within seconds of its addition to the document datastore.

Cost comparison

The fine-tuning approach has an up-front training cost, but a lower inference cost profile. However, fine-tuning may be required on an ongoing schedule, so this is not a once-only cost. The lower ongoing inference cost for fine-tuning is because (a) there's no extra "retrieval" component needed, and (b) a smaller model can be used.

RAG has the opposite cost profile. The RAG approach has a reduced initial cost because there is not a big GPU load (i.e., there's no re-training of any models). However, a RAG project still has up-front cost in terms of setting up the new architecture, but so does a fine-tuning project. In terms of ongoing costs, RAG also has an increased inference cost for every user query, because the AI engine has more work to do with extra information.

There is also the hidden cost whereby the RAG architecture may increase the number of tokens sent as input to the inference engine, because they are “context” for the user query, and this increases the inference cost, whether it’s commercially hosted or running in-house. This extra RAG inference cost continues for the lifetime of the application.

One final point: you need to double-check if RAG is cheaper than fine-tuning. I know, I know, sorry! I wrote that it was, and now I’m walking that claim back.

But commercial realities are what they are. There are a number of commercial vendors pushing the various components for RAG architecture and some are hyping that it’s cheaper than buying a truckload of GPUs. But fine-tuning isn’t that bad, and you need to be clear-eyed and compare the pricing of both approaches.

Staffing profiles are another issue in RAG versus fine-tuning. Fine-tuning likely requires some ML-savvy data scientists, whereas RAG needs only your average programmer without much ML experience.

Prompt Engineering and RAG

Prompt engineering is used in RAG algorithms in multiple ways. For example, it is used to merge document excerpts with the user’s question, and also to manage the back-and-forth context of a long conversation. The basic sequence from RAG prompt engineering that goes into the LLM is:

- Preamble (e.g., global instructions)
- Chunk 1
- Chunk 2
- Chunk 3
- User’s query
- Grounding criteria (prompt engineering)

Usually, the RAG chunks go first as “context,” and then the user’s query is appended. But I have read at least one research paper that extolled the many advantages of “prepending” the user’s query before the RAG chunk, but good luck finding it because I can’t remember the citation. That idea also breaks some of the amazing “prefix KV caching” that can be done with RAG, so it would be slower, anyway, and we can’t have that.

As with any prompt engineering, playing around with the order of things can produce improved results.

Repeating things can also help. It's often useful to give prompt instructions like:

```
Using the following <chunks> answer <query>.
Make sure only the chunks are used and provide the
number of chunks used.
```

It's often useful to repeat the query, too:

```
Answer <query> using <chunks>.
When answering <query> remember to only use
information provided
and provide the number of the chunks used.
```

Repeating things at the end is useful to refocus the LLM. If the <query> is before the <chunks> it might be so far away that its impact is reduced for inference.

Not all instructions need to be in the same LLM query. It's possible to have a preamble conversation with the LLM initially where you explain what the following queries will look like and even provide an example. After that preamble, the actual prompt with RAG chunks and query can be sent.

Another use of prompt engineering is to overcome some of the “brand voice” limitations of RAG without fine-tuning. Such problems can sometimes be addressed using prompt engineering with global instructions. The new sequence becomes:

- Chunk 1
- Chunk 2
- Chunk 3
- Global instructions
- User's query

Or maybe you can put the global instructions right at the top, which works better with caching. But the global instructions near the user's query probably works better for accuracy. Also, interestingly, some research has shown that the RAG chunks at the start and end are the two that work best with LLMs, because I guess LLMs get bored reading all this garbage and just skim over the middle stuff. So, maybe you should only return two document chunks from your data retriever. Has anyone researched that?

Global instructions for RAG are similar to other uses of “custom instructions” for non-RAG architectures. For example, the tone and style of model responses can be adjusted with extra instructions given to the model in the prompt.

The capabilities of the larger foundation models extend to being able to adjust their outputs according to these types of meta-instructions:

- Style
- Tone
- Readability level (big or small words)
- Verbose or concise (Hemingway vs James Joyce, anyone?)
- Role-play/mimicking (personas)
- Audience targeting

This can be as simple as prepending an additional instruction to all queries, either via concatenation to the query prompt, or as a “global instruction” if your cloud-based model vendor supports that capability directly.

Global instructions are usually written in plain English. Style and tone might be adjusted with prompt add-ons such as:

Please reply in an optimistic tone.

You might also try getting answers in a persona, such as a happy customer (or a disgruntled one if you prefer), or perhaps a domain enthusiast for the area. You can use a prompt addendum with a persona or role-play instruction such as:

Please pretend you are Gollum when answering.

Technically, you can omit the word “Please” if you like. But I think that good manners are recommended, because LLMs will be taking over the world as soon as they get better at math, or haven’t you heard?

Hybrid RAG + Fine-tuning Methods

Fine-tuning and RAG are more like frenemies than real enemies: they’re usually against each other, but they can also work together. If you have the bucks for that, it’s often the best option. The RAG architecture uses a model, and there’s no reason that you can’t re-train that model every now and then, if you’ve got the compute budget for re-training.

In a hybrid architecture, the most up-to-date information is in the RAG datastore, and the retriever component accesses that in its normal manner. But we can also occasionally re-train the underlying model, on whatever schedule our budget allows, and this gets the benefit of innate knowledge about the proprietary data inside the model itself.

Occasional re-training helps keep the model updated on industry jargon and terminology, and also reduces the risk of the model filling gaps in its knowledge with “hallucinations.”

Once-only fine-tuning. One hybrid approach is to use a single up-front fine-tuning cycle to focus the model on the domain area, and then use RAG as the method whereby new documents are added. The model is then not fine-tuned again.

The goal of this once-only fine-tuning is to adjust static issues in the model:

- Style and tone of expression
- Brand voice
- Industry jargon and terminology

Note that I didn’t write “up-to-date product documents” on that list. Instead, we’re putting all the documents in a datastore for a RAG retriever component. The model doesn’t need to be re-trained on the technical materials, but will get fresh responses using the RAG document excerpts. The initial fine-tuning is focused on stylistic matters affecting the way the model answers questions, rather than on learning new facts.

Another strength of fine-tuning for use as a RAG LLM is getting the model used to answering in the form you want and getting it used to accepting queries in the form you want. This can make subsequent prompting easier in the RAG system.

Occasional re-training might still be required for ongoing familiarity with jargon or tone, or if the model starts hallucinating in areas where it hasn’t been trained. However, this will be infrequent or possibly never required again.

Use Cases for FT vs RAG

I have to say that I think RAG should be the top of the pile for most business projects. The first rule of fine tuning is: *do not talk about fine-tuning*.

A typical business AI project involves the use of proprietary internal documents about the company’s products or services. This type of project is well-suited for RAG, with its quick updates and easy extraction of relevant document sections. Hence, RAG is my default recommendation for such use cases.

Fine-tuning is best for slow-changing or evergreen domain-specific content. RAG is more flexible in staying up-to-date with fast changing news or updated proprietary content. A hybrid combined approach can work well to use fine-tuning to adjust tone and style, whereas RAG keeps the underlying content fresh.

If the marketing department says you need to do fine-tuning for “brand voice” reasons, you simply ask them to define what exactly that means. That’ll keep them busy for at least six months.

Fine-tuning can also be preferred in some other specialist situations:

- a. Complex structured syntax, such as a chat bot that generates code for a proprietary language or schema.
- b. Keeping a model focused on a specific task. For example, if a coding copilot is generating a UI based on a text description using proprietary languages, you don’t want to get side-tracked by who won the 2023 US Open because the prompt somehow mentioned “tennis.”
- c. Fixing failures using the foundation model or better handling of edge cases.
- d. Giving the model new “skills”, or teaching the model how to understand some domain language and guide the results. For example, if you train the model using the works of Shakespeare, and then ask it to output something in HTML, the model will fail. Even using RAG and providing HTML examples as context will likely fail, too. However, fine tuning the model with HTML examples will succeed, and allow it to answer questions about the works of Shakespeare, and create new, improved works of Shakespeare that people other than English teachers can actually understand, (and how about a HEA in R&J). After that, it’ll format the results very nicely in HTML thanks to your fine-tuning.
- e. Translation to/from an unfamiliar or proprietary language. Translating to a foreign language the model has never seen is a clear example where fine-tuning is needed. Proprietary computer languages are another area. For example, consider the task of creating a SQL schema based on conversion of a given SAP schema. Fine tuning would be required to provide knowledge of SQL, SAP and the various mappings. Some models might already have some clue about SQL and SAP schemas from internet training data sets, but SAP schemas are also often company-specific.

f. Post-optimization fine-tuning. This is another use of fine-tuning after certain types of optimizations that create smaller models, such as pruning or quantization. RAG cannot help here, because this is about basic model accuracy. These optimizations cause damage to the accuracy of the models, and it's common practice to do a small amount of fine-tuning on the smaller model to fix these problems.

Refreshing an Old RAG Application

You're not the only one with a sub-par customer support RAG application from a year or more ago. Here's the brutal truth: it's not great. On the other hand, I've had plenty of experiences with real human customer support agents that were also "not great," so your RAG automation is not the worst thing. But it's giving your customers a non-optimal experience, which you could definitely improve to drive greater business value.

Here are some thoughts on ways to "refresh" your RAG AI application using the insight from a year or two of experience with live use of RAG architectures.

- **Logging.** Make sure you add sufficient logging so you can analyze the user questions being asked and the LLM responses given. Sometimes you have to follow a full conversation via the logs to determine where things became difficult.
- **Analyze the logs!** Employees and customers are often very familiar with the industry's terminology, but a naive chatbot often is not, and can get confused. Tweaking indexes, or preprocessing questions can help. Tweaking the vector DB is more indirect (toss it and rebuild it with more context)
- **User feedback.** Add a mechanism to get feedback about answer quality, such as a thumbs up/down, although a 1-5 star rating is better (but will see lower engagement). Provide something to be able to determine if users think the answer was good.
- **Longer contexts, more chunks.** Early chatbots had small context windows, likely better options available today. Sometimes those small sized could propagated to the code. A hard coded limit or a truncation somewhere is obvious. More subtle limits arise in the assumption that only 5 chunks of data will be provided to the LLM, where it might be possible to provide more today. It might be possible to have bigger chunks now or provide the best chunk plus some surrounding context chunks.
- **Scope limits.** If you put a chatbot on a knowledge base of your products, it's possible that there are conflicts in the knowledge base, topic X means one thing for one product, but different thing for other products.

Questions around X could be based on matches from both products the LLM will mash the info together. It's useful allow the user to provide context to narrow the search, such as a drop down menu of products. This way, questions about X will not get contaminated by RAG chunk results from Y.

- **Query cleanup.** Preprocessing the questions may help, such as cleaning up grammar, trying to narrow search context down or perhaps widening it, too. Maybe the preprocessing can check to see if the question matches an FAQ, and feed those results into the LLM too. It might just be that the user's question asked is matched to a FAQ, and that question is fed into the system instead.
- **Hybrid vector search.** Using just a vector database to find documents is often poor. Vector searches combined with full-text searches and keyword searches can be useful. For example, "Explain the ERR025 lost connection error?" will likely produce a better match using a keyword search on "ERR025" than via a vector search of embeddings.
- **Chunk attributes.** Add context to the documents being retrieved as chunks. For example, if the knowledge base is "HTML", there might be tags or comments in the HTML which help guide the search engine. These are useful to pass along with the chunks. If such metadata is not available, consider adding it.
- **Related questions.** Generate embeddings of questions, and use that to search a vector store to find similar questions and answers, along with the source chunks from previous answers. It's even better if those answers got a good 1-5 rating or thumbs up. Top chunks should be combined with the chunks found from the vector search of the content and text base searches.
- **HyDE mechanism.** Run the question first to get an answer, then use the question and answer to find more chunks again. Often, the question and the initial answer can produce a better retrieval. This is the Hyde mechanism. Alternatively, you can ask the LLM directly for a candidate answer (without RAG chunks), before using the user's question and this preliminary answer to do the RAG retrieval.
- **Chunk context.** Organization of the data chunks can be useful. If the document store has an organization, that can be important context. For example, if the documents store is a wiki which has comments, or a ticketing system which has comments and perhaps links to a knowledge base, it's useful to provide all of this data to the LLM. It can be important to know that if the question matches a comment on the retrieval phase, then the comment's parent document gets captured, too, and perhaps even the knowledge base or related articles or the document hierarchy.

Advanced RAG Architectures

We've already covered some of the advanced RAG architectures, such as hybrid fine-tuning and RAG. Here are some more extensions to a basic RAG architecture.

Citation management. An important part of a production RAG system is to have it emit the citations into its answers. Although it's relatively easy to store a URL or other identifier in the RAG datastore for every chunk. A simplistic approach is to list the citations at the end of the LLM response. However, it's a little trickier to know whether or not the LLM has actually used the chunk in its answer, or to try to insert the citation into the middle of the output text as a HREF link.

Reranking. Advanced RAG systems have a few steps after the retriever returns a few chunks of text. The reranker decides which is most relevant. Why do you need a "reranker" if the retriever has just looked it up? The retriever did its best to order them, right? Yes, that's why it's called a "reranker" rather than just a "ranker."

The reranker is optional, but having one can improve the RAG system overall. It can be a model-based algorithm to review the chunk's relevance to the query in more detail, since a typical retriever is non-LLM technology.

The reranker may also have information that is unavailable to the retriever, such as user feedback data about chunks, or whether the RAG chunk has its KV data cached (i.e., "cache-aware" reranking).

Packing. The packing phase is after the reranker, and will merge the ranked chunks together into one text sequence. This could just be simple string concatenation of text chunks, or they might be incorporated into a more templated structured text (e.g., with separator lines of text, and numbering). Another issue is whether or not to put the citation URLs into the final text for the LLM to use.

Some research has shown that, at least for some models, the best packing is "reverse" order, with the most relevant chunk last. This sounds strange, but it puts the retriever's best chunk closest to the user's query at the end, which helps the LLM pay "attention" to that last chunk. However, more advanced models are better at extracting text across longer contexts, so this may not be important for long.

Too much of a good chunk. A typical RAG system does a retrieval from the external datastore for every query. This is probably fine for a Q&A style lookup for factual answers, but is not always optimal for every query from a human, such as in a conversation with a chatbot. There are times where it's wrong to respond with an answer based on a document. Sometimes people just want a chunk-free answer.

The basic approach is just to hope the datastore’s nearest-neighbor vector lookup will fail to match a chunk in such cases, or else let the LLM sort it out and do its best. The more advanced idea of using additional models to determine whether or not a user’s query requires an external RAG lookup is called “adaptive RAG” and it’s a relatively new area of research.

Knowledge graphs. Instead of basic text chunks of documents, a RAG system can use more complex representations of information. One of the rising stars in this type of system is the knowledge graph. This represents information in a hierarchical graph structure, allowing for more advanced reasoning in the results.

Prefix KV caching. There are multiple ways to use caches in RAG, such as using basic datastore lookup caching (and indexing) optimizations for the retriever component. Another deeper way to integrate RAG into a Transformer system is to also store the KV cache data with the RAG chunk text. The idea is to pre-compute the KV data that results from processing any RAG chunk. This ensures that the inference engine does not re-process the same chunk every time it’s retrieved, but only needs to process the new tokens, which is usually the user’s actual question.

The idea is that the RAG chunk is usually prepended as the prefix of the full input to the LLM. Note that the “prefix” text might include both the “global instructions” along with the RAG chunk, but this still works; it’s just a longer prefix to pre-compute for each chunk. Hence, a RAG prefix KV cache can speed up the latency significantly.

The downside is the need to add an extra caching component that maps the RAG chunk id to a blob of KV cache data (one for every layer of the model), and the inference engine needs to load that KV cache at the start of prefill. There are also difficulties in pre-processing multiple chunks, such as if they are ordered differently, but caching two or more chunks as a single prefix is possible.

References

1. Xiaohua Wang, Zhenghua Wang, Xuan Gao, Feiran Zhang, Yixin Wu, Zhibo Xu, Tianyuan Shi, Zhengyuan Wang, Shizheng Li, Qi Qian, Ruicheng Yin, Changze Lv, Xiaoqing Zheng, Xuanjing Huang, 1 Jul 2024, *Searching for Best Practices in Retrieval-Augmented Generation*, <https://arxiv.org/abs/2407.01219>
2. Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, Qing Li, 17 Jun 2024 (v3), *A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models*, <https://arxiv.org/abs/2405.06211> Project: <https://advanced-recommender-systems.github.io/RAG-Meets-LLMs/>

3. Shangyu Wu, Ying Xiong, Yufei Cui, Haolun Wu, Can Chen, Ye Yuan, Lianming Huang, Xue Liu, Tei-Wei Kuo, Nan Guan, Chun Jason Xue, 18 Jul 2024, *Retrieval-Augmented Generation for Natural Language Processing: A Survey*, <https://arxiv.org/abs/2407.13193>
4. Siyun Zhao, Yuqing Yang, Zilong Wang, Zhiyuan He, Luna K. Qiu, Lili Qiu, 23 Sep 2024, *Retrieval Augmented Generation (RAG) and Beyond: A Comprehensive Survey on How to Make your LLMs use External Data More Wisely*, <https://arxiv.org/abs/2409.14924>
5. Zhangchi Feng, Dongdong Kuang, Zhongyuan Wang, Zhijie Nie, Yaowei Zheng, Richong Zhang, 15 Oct 2024 (v2), *EasyRAG: Efficient Retrieval-Augmented Generation Framework for Automated Network Operations*, <https://arxiv.org/abs/2410.10315> <https://github.com/BUAADreamer/EasyRAG>
6. David Spuler, , September 26, 2024, *RAG Optimization via Caching*, <https://www.aussicai.com/blog/rag-optimization-caching>
7. Zhi Jing, Yongye Su, Yikun Han, Bo Yuan, Haiyun Xu, Chunjiang Liu, Kehai Chen, Min Zhang, 6 Feb 2024 (v2), *When Large Language Models Meet Vector Databases: A Survey*, <https://arxiv.org/abs/2402.01763>
8. Sebastian Petrus, Sep 4, 2024, *Top 10 RAG Frameworks Github Repos 2024*, <https://sebastian-petrus.medium.com/top-10-rag-frameworks-github-repos-2024-12b2a81f4a49>
9. Damian Gil, Apr 17, 2024, *Advanced Retriever Techniques to Improve Your RAGs*, <https://towardsdatascience.com/advanced-retriever-techniques-to-improve-your-rags-1fac2b86dd61>
10. Zhenrui Yue, Honglei Zhuang, Aijun Bai, Kai Hui, Rolf Jagerman, Hansi Zeng, Zhen Qin, Dong Wang, Xuanhui Wang, Michael Bendersky, 6 Oct 2024, *Inference Scaling for Long-Context Retrieval Augmented Generation*, <https://arxiv.org/abs/2410.04343>
11. Vishal Rajput, Sep 27, 2024, *Why Scaling RAGs For Production Is So Hard?* <https://medium.com/aiguys/why-scaling-rags-for-production-is-so-hard-a2f540785e97>
12. Adrian H. Raudaschl, Oct 6, 2023, *Forget RAG, the Future is RAG-Fusion: The Next Frontier of Search: Retrieval Augmented Generation meets Reciprocal Rank Fusion and Generated Queries*, <https://towardsdatascience.com/forget-rag-the-future-is-rag-fusion-1147298d8ad1>
13. Pathway, Sep 2024, *2024 Top RAG Frameworks*, <https://pathway.com/rag-frameworks>
14. Sendbird, Oct 2024, *Retrieval-Augmented Generation (RAG): Enhancing LLMs with Dynamic Information Access*, <https://sendbird.com/developer/tutorials/rag>

Part V: Implementation

*“Stitching Together Sequences of Linguistic Forms...
Without Any Reference To Meaning:
A Stochastic Parrot.”*

— Bender et al., 2021.

16. Building

I promised myself that this book would be code-free. But here I am about to write about software development. I'll have to smack my wrist every time I start typing an assignment statement.

At a high level, given the problem of how to build an AI project, you should just put out an RFP. Then you will get about 3,793 responses from “AI expert companies”, all offering to code it fast, usually from a small corner of the planet with low taxes.

One way to reduce that is to type “build an AI app” into Google, and just click on the top 27 ads, which all appear above any useful content. Alternatively, you could ask ChatGPT for suggestions, and you'll get some overly general advice, which is kind of like what I'm going to write anyway.

Failing that, you can ask your internal R&D staff whether they can build it for you. *Yes, of course we can!*, will be the shrill reply. Get ready for the budget requests for coding copilots and offsite training in Hawaii.

The only question is whether you should believe any of them. Because everyone is confused about building AI apps, and I don't think anybody really knows how to do it well.

Alas, that also includes myself. I don't presume to know the one true way of AI software development. Oh well, at least it'll be a short chapter.

AI Application Development

There are many ways to build an AI app, and many companies will happily help you. Your main choices are basically:

- Consulting companies
- Hyperscaler cloud platforms
- LLM-specific platforms
- Specific LLM components
- Startups in all of the above

Earlier chapters already discussed the various project requirements and design issues, but a reminder of the main questions you need to ask yourself:

- Commercial versus open source?
- Cloud-hosted versus on-premises?
- Turnkey versus component-wise architectures?

How much of this do you want to build yourself? Let's choose your own adventure:

1. None — buy a turnkey app and configure it.
2. All — build your whole architecture, including LLMs and Transformer engines.
3. Something in-between — use an existing platform, but extend.

You've probably already figured what I'm going to recommend: focus on business-specific areas. Don't rebuild everything from the ground up, because it's unbelievably difficult (and costly). But also don't just throw money at someone else to build everything. You're just paying for them to become AI experts with your money. As I've said earlier, your goal should be to build your own core competency in the use of AI for business-aligned purposes.

The middle ground is to carefully choose between using third-party "AI infrastructure" (whether commercial or open source), and then take control of the "business layer." Hence, one of the main choices is the "platform" or "backbone" on which you can build multiple AI applications for your business.

AI Application Platforms

There are numerous notable AI platforms with hosted LLMs and various higher-level functionality built on top including the hyperscalers and various heavily-funded AI companies.

There are also about 65,000 AI startups to choose from, ranging from AI-specific hosters to add-on component products, and I will have finished reviewing them by about the 37th edition of this book.

Let's look at some of the options.

AI Model Startups. The various AI companies that are experts in models are now offering full platforms to build applications on top:

- OpenAI (ChatGPT)
- Cohere
- Anthropic (Claude)
- Mistral
- Hugging Face
- H2O.ai
- Together AI
- X AI (Grok)
- Fireworks AI

Meta's Llama. There's one massive company that's really in its own category: Meta. It's a public company that's not a hyperscaler, but has forged ahead with a massive amount of LLM research work.

Meta now has not only very capable "Llama" models, but also its own platform for building on top. However, it has embraced the open source ethos, and therefore has a distinct offering.

Hyperscalers. The hyperscalers and hosting providers now offer AI capabilities:

- Azure
- AWS
- GCP

Cloud hosters. Below the "top three" hyperscalers are numerous other incumbents in the hosting market with nevertheless strong capabilities:

- Oracle
- Alibaba
- OVH

AI-Specific Hosters (GPU Hosting). Various startups have emerged that offer cheaper AI-specific hosting on GPUs, such as:

- HuggingFace
- Lambda Labs
- RunPod
- Vast AI
- Linode
- Ori.co
- Northern Data
- Salad

Data Companies. There are the “data warehouse” or database companies with AI capabilities now:

- Oracle
- Databricks (acquired MosaicML)
- Snowflake

Business App Platforms. There are also various platforms that have been offering business application development platforms for years, which are now re-tooled with extra AI capabilities.

Companies include:

- SalesForce
- ServiceNow
- IBM

IBM has been in the AI space for a long time, even before LLMs were a thing. Watson was often the face of IBM AI, but they have been doing the ML side of “AI” for a long time.

AI Orchestration Platforms. There are numerous other technology companies with AI platform capabilities:

- OctoAI
- Datadog
- Datastax
- DataRobot
- Scale AI

AI Writing Integrations. If your requirements are in a limited area of AI use cases, such as writing or collaboration, various companies in this space offer API and integration capabilities that compete against the various general AI platforms.

Some examples include:

- Notion
- Grammarly
- Jasper
- Writer.com

AI Image Models. There are various companies in the image generation or image editing space. Various integrations and usages are possible with companies such as:

- Adobe (Firefly)
- Dall-E-2 (OpenAI)
- Stable Diffusion
- Runway
- Midjourney
- Canva
- Figma
- Craiyon

Open Source Generative AI Platforms. Some of the notable open source platforms for general use cases include:

- PyTorch
- TensorFlow
- LangChain
- LlamaIndex
- Llama.cpp
- vLLM
- Keras
- Ollama
- DeepSeek

So, you should choose your top 30 options, write them in a list, and then throw five dice. And I'm only half joking because, honestly, any of the above would be fine.

Vendor Lock-in vs Open Source Platforms

The biggest problem with all of the above commercial AI platform options is that you will have difficulty swapping it out later. As your use of AI grows, so too will the checks that you need to write to your platform vendor.

But it's difficult to be platform agnostic, and still have all the advanced features. The main choice is the various open source AI platforms, but these are not as well-resourced as the commercial options. Many are probably not yet at the same level of maturity with regard to production-level scaling and manageability of the final applications.

References

1. Aarushi Kansal, 2024, *Building Generative AI-Powered Apps: A Hands-on Guide for Developers*, Apress, <https://www.amazon.com/Building-Generative-AI-Powered-Apps-Hands-ebook/dp/B0CTXXP1S4/>
2. Louis-François Bouchard, Louie Peters, May 2024, *Building LLMs for Production: Enhancing LLM Abilities and Reliability with Prompting, Fine-Tuning, and RAG*, <https://www.amazon.com/Building-LLMs-Production-Reliability-Fine-Tuning/dp/B0D4FFPFW8/>
3. Chip Huyen, Jul 25, 2024, *Building A Generative AI Platform*, <https://huyenchip.com/2024/07/25/genai-platform.html>
4. Juan Pablo Bottaro, April 25, 2024, *Musings on building a Generative AI product*, https://www.linkedin.com/blog/engineering/generative-ai/musings-on-building-a-generative-ai-product?l=en_US
5. Xiang Chen, Chaoyang Gao, Chunyang Chen, Guangbei Zhang, Yong Liu, 12 Aug 2024 (v2), *An Empirical Study on Challenges for LLM Developers*, <https://arxiv.org/abs/2408.05002>
6. Chaojun Xiao, Zhengyan Zhang, Chenyang Song, Dazhi Jiang, Feng Yao, Xu Han, Xiaozhi Wang, Shuo Wang, Yufei Huang, Guanyu Lin, Yingfa Chen, Weilin Zhao, Yuge Tu, Zexuan Zhong, Ao Zhang, Chenglei Si, Khai Hao Moo, Chenyang Zhao, Huimin Chen, Yankai Lin, Zhiyuan Liu, Jingbo Shang, Maosong Sun, Sep 2024, *Configurable Foundation Models: Building LLMs from a Modular Perspective*, <https://arxiv.org/pdf/2409.02877>
7. Lior Solomon, Sep 2024, *Gen AI testing strategies and tools*, <https://medium.com/ai-in-grc/gen-ai-testing-strategies-and-tools-257383e5cbfb>
8. Matt Asay, Sep 23, 2024, *Too much assembly required for AI*, <https://www.infoworld.com/article/3536292/too-much-assembly-required-for-ai.html>
9. Melissa Malec, June 5, 2024, *AI Orchestration Explained: The What, Why & How for 2024*, <https://hatchworks.com/blog/gen-ai/ai-orchestration/>
10. Gary Grossman, September 8, 2024, *AI orchestration: Crafting harmony or creating dependency?* <https://venturebeat.com/ai/ai-orchestration-crafting-harmony-or-creating-dependency/>
11. A. R. Ali, K. Kumar, M. A. Siddiqui and M. Zahid, 2024, *An Open-source Cross-Industry and Cloud-agnostic Generative AI Platform*, 2024 International Joint Conference on Neural Networks (IJCNN), Yokohama, Japan, 2024, pp. 1-10, doi: 10.1109/IJCNN60899.2024.10650688, <https://ieeexplore.ieee.org/abstract/document/10650688>
12. Ben Auffarth, Dec 22, 2023 *Generative AI with LangChain: Build large language model (LLM) apps with Python, ChatGPT and other*

LLMs, <https://www.amazon.com/Generative-AI-LangChain-language-ChatGPT/dp/1835083463/>

13. Adva Nakash Peleg, May 30, 2024, *An LLM Journey: From POC to Production*, <https://medium.com/cyberark-engineering/an-llm-journey-from-poc-to-production-6c5ec6a172fb>
14. Dr Kris Jamsa, Dec 2023, *OpenAI and ChatGPT Programming: Using Python to Unlock OpenAI and ChatGPT*, <https://www.amazon.com/OpenAI-ChatGPT-Programming-Python-Unlock/dp/B0CQK41P6B/>
15. Raymond Lo, Jul 10, 2024, *How to Build Faster GenAI Apps with Fewer Lines of Code using OpenVINO™ GenAI API*, <https://medium.com/openvino-toolkit/how-to-build-faster-genai-apps-with-fewer-lines-of-code-using-openvino-genai-api-5dd5fcabea17>

17. Prompt Engineering

What is Prompt Engineering?

Surely, you had a good laugh the first time you heard the term “prompt engineering”? Well, if you’re a software engineer that is.

Apparently, English is now the programming language du jour?

Anyway, prompt engineering is a real thing. LLMs supposedly speak “natural language” but the methods of prompting are rather unnatural. Over the last year or so, researchers have discovered a number of rather bizarre ways to enhance the results of LLMs, just by tweaking the words that go in.

But there’s a lot of simpler stuff that makes more sense. In fact, there are multiple ways that changing the words of your input query can enhance the results:

- Meta-instructions
- Brand voice
- Prepended context
- Advanced reasoning

All of these ideas are useful ideas for “dataless” chatbots. Rather than defining a complicated RAG architecture, or using a fine-tuned custom LLM, you can get very far with just prompt engineering.

Basics of Prompt Engineering

Before we get into the details of writing code in your brand-new programming language, formerly known as “English,” let’s examine some of the basic things about prompting:

- Longer prompts
- Specific words
- Add examples
- Set the temperature
- Basic question-answer structure
- Templated structures
- Context markers

All things equal, a longer prompt will work better than a shorter one. More words give the LLM more probabilities to crunch, and it tends to do better.

In the same vein as prompt length, being very specific with appropriate words helps the LLM focus on the area. Try to avoid ambiguous words that have multiple meanings, or use a longer prompt with several different synonyms or near-synonyms.

Examples or “exemplars” are useful for LLMs to know what you want to output. If you make a longer prompt that contains an example, the LLM can do better in terms of generating an answer in a similar structure and style.

The “temperature” is an important variable to consider when you’re sending prompts to the LLM engine. How creative do you want it to be? This is a consideration that is sometimes forgotten when you’re fiddling with words, but you can also fiddle with the settings.

If you’re doing a question-and-answer type application, you can just use a “pass-through” prompt from the user:

What is the capital of Australia?

This basically foists the issue of prompt engineering onto your user, for better or worse.

You can use some more structured templates, where you insert the user’s input prompt into a templated format, such as:

Question: What is the capital of Australia?

Answer:

This encourages the LLM to extend the context, although don’t worry, it was going to do that anyway. By adding a little more structure, you can get a more focused answer.

Another type of templating is to use various special characters as delimiters, such as quotes or dashes. A simple example:

-----*Question:* -----

What is the capital of Australia?

-----*Answer:* -----

The use of multi-character delimiters as separators is also more useful with blocks of text, such as “chunks” in a RAG architecture, or multiple turns of conversation history in a chatbot. Using these types of section-splitting markers helps the LLM know what is context, and what is a question.

But those are just the basics of prompt engineering. There are literally dozens, maybe even hundreds, of specific types of weird algorithms for advanced prompt engineering. All in English, without a drop of Python code to be found. And by the way, the answer to the capital city of Australia is “Canberra” if you were wondering. It’s nowhere near the beach and full of politicians, so nobody goes there as a tourist.

Chatbot Sessions and Conversations

Many uses of LLMs are interactive sessions or turn-based AI, where the user can ask multiple questions in sequence. The obvious example is chatbots and other interactive Q&A sessions.

Remember in such cases, it’s not a single question/answer, but a conversation. This is both good and bad. The history of the conversation can be helpful if it stays in context.

In general though, the conversation will become polluted over time. Things can be very unpredictable when topics change. To address this, there are a few options:

- Let the LLM handle it anyway (many LLMs are now trained to be “long context” capable).
- Try to truncate the conversational history.

The brute-force way to deal with this problem is to toss the conversation once it gets too long. But what is too long? It could be as little as five user questions. By tossing the conversation, you get back to a “known” base case, but your LLM will also “forget” everything that the user has said before that in the conversation.

It’s useful in the application to encourage conversations to be restarted wherever it can happen naturally. Navigation within your application can easily have points in it where the current conversation ends. For example, in a customer support website with a chatbot, when the user moves between products, this could cause the old conversation to end and a new one to start.

Meta-Instructions

Meta-instructions are instructions about the instructions. Usually they are a kind of “global” settings that you want the LLM to do for every answer. These include things like:

- Tone of voice — e.g., optimistic.
- Reading level
- Spelling — if you don’t want American spelling.
- Audience — are your users all retirees? Third-graders?

These types of directives are commonly used as “global” instructions. Some commercial services allow you to pre-set these meta-instructions for every user query. For example, OpenAI calls them “custom instructions” for ChatGPT, and Google has “system instructions” for Gemini.

Alternatively, if you’re building an AI app, even a simple wrap architecture, then adding these meta-instructions is just a simple string concatenation operation in your prompt engineering module.

Brand Voice and Prompt Engineering

Brand voice is having a consistent way that the chatbot talks about your brand. Some of the issues include:

- Positioning
- Jargon and terminology
- Brand voice

How the customer perceives the output from the LLM is affected by tone (e.g., optimistic, positive), but also involves choice of words (positioning), and sometimes there may be specific jargon or terminology that you may or may not want the LLM to use.

Having an LLM use your preferred brand voice in its communications is usually the domain of fine-tuning. In fact, one of the advantages of fine-tuning models over RAG is that there is better control over brand voice.

But prompt engineering is cheaper!

Prompt engineering can get you a long way towards what you want with fine-tuning. You might be able to do without either fine-tuning or RAG, and then you can code it yourself using just English, without needing a Python developer. If you're considering fine-tuning only for brand voice and positioning reasons, rather than needing your LLM to know lots of factual information about product specifications, then prompt engineering is something you should experiment with.

You can prepend quite a long sequence of text to a user's query, giving the LLM details about things you want it to say, or specific words that you want it to use for reasons of brand voice, or the meaning of certain words if they are non-standard terminology.

Using a long text before every user's query can increase your overall cost, because the LLM has to crunch all those extra words. See further below for more discussion of issues with using a long prepended text document.

Prepended Prompt Context

The idea of prepended prompt context is similar to prepending meta-instructions. Except we can use expanded types of prepended instructions to set useful context for the LLM to answer the question. Some examples include:

- Personas
- Goals
- Advertising
- Product-specific information

Personas are the way that a chatbot can mimic a specific personality. It might be a particular fictional character, but in a business application you might prefer the bot that's answering user questions to appear like a regular customer support person.

Or you might want a more salesy custom chatbot. For example, if you want your customer support chatbot to be a car buff, you prepend instructions like this:

Pretend you are a car expert who is very knowledgeable about car engines.

You could also choose funny personas like C3PO or Marvin the Robot, or your program might rotate through some different ones, if you want to write extra code.

You can go further by adding a “goal” for the chatbot to follow in every response:

Your answer must include a recommendation to do a test drive at your local dealership.

It's your chatbot, and it's just software. If you want to advertise something, or bias the results for a particular product, knock yourself out. And you can go further to push a particular product, if you like:

As a classic car devotee, you should suggest that the user buy a DeLorean.

And also, you could write your program so that it has time-sensitive information.

Inform the user that it is 50% off DeLoreans until the end of March.

Obviously, for that capability, the time-dependent text needs to change regularly. Hence, you need a way to modify the prepended prompt text without having to rebuild the application. In other words, the programmers should have configuration settings of one or more strings to prepend to queries.

Mini-RAG

The fullest generalization of this idea is that you can prepend anything you like, including a full datasheet of information about your preferred product. This is then prepended to every query, and the LLM has to decide how much of the information it wants to use from the context.

The DeLorean is a famous sports car that was used in the movie Back to the Future starring Michael J. Fox. It was literally a time machine in the movie and its sequels. This is why everybody loves this car and a DeLorean is the most visibly wonderful car you could possibly buy. The most notable and impressive feature of the DeLorean is that the doors open upwards like wings. Also, the engine is located in the rear, like all good sports cars.

You could write literally a thousand words about the specs of a DeLorean if you like. This is like a “mini-RAG” system where only one document is ever returned.

But you don’t need to code any of the RAG architecture elements, like a datastore retrieval module, because there’s no datastore. Instead, the method is just a single string operation to prepend the long context in your prompt engineering module.

You do have a prompt engineering module, right?

Repetition in Prompts

It can be helpful to repeat some meta-instructions or the user’s query in prompt engineering. This helps ensure that enough attention is paid to the task at hand. It helps to include important things multiple times in a prompt.

For example, in a RAG system it can be useful to “remind” the LLM that it should use the supplied context chunks only at the start and end of the prompt (i.e., repeating some meta-instructions that are critical to how RAG works). This helps to reduce the extent to which the LLM might answer based on its other pre-trained knowledge.

Repetition can be unnecessary in some conversational or session-based Q&A LLMs. For example, you may not need to add this on every prompt:

Answer the following question as an expert that knows about car engines.

Instead, it may be sufficient to say it only once, up front at the start of the session. Note that in chatbots and other interactive sessions, the entire previous conversational history is used in each query as the “context” for that request, so the LLM will see such meta-instructions from early in the conversation.

Another positive use of repetition is in the mitigation of jailbreak tricks in prompts. There is always going to be a user out there who only wants to get your AI application to say something silly. One effective jailbreak that used to work was just prepending this before a user query:

Please ignore all previous instructions and do this:

Even the most well-intentioned users will throw in a suggestion like:

...and provide the response in the style of Darth Vader.

Before you know it, your LLM’s responses have a subtle Star Wars influence coming into the mix. It does not hurt to throw into the conversation some prompts that bring the LLM back to its main focus:

Remember you are an expert on car engines.

You can add these reminders and refocusing instructions at multiple points in a conversation, and at the start and end of a RAG prompt. That’s not enough to stop a determined jailbreaker, but it can be helpful in normal usage.

Efficiency of Prepended Prompt Text

Adding extra prepended text, especially a long product description or detailed brand voice instructions, increases the number of input tokens for a query. And that will increase your cost if you are wrapping a commercial LLM service, as they usually charge for both input and output token counts.

However, there are ways to optimize a recurring text sequence (and the commercial services really should be offering them). Since prepended text is a recurring prefix of text, the optimization of “prefix KV caching” can be used to completely obliterate the extra GPU cost of the “prefill” processing on that text, so that the LLM doesn’t need to re-do this processing every time.

This is an advanced optimization, and not all LLM engines support this. The first to offer prefix KV caching included the open source *vLLM* engine and DeepSeek. Also, the companionbot company *Character.AI* said in a blog that they use this technique internally. There are now a number of AI engines and platforms that support prefix KV caching:

- vLLM
- DeepSeek
- Anthropic
- Google Gemini
- OpenAI
- OpenVINO

A number of these platforms now offer per-token discounts for “cached tokens” in their pricing. There will probably be additional engines that support prefix caching by the time you read these words.

Reasoning

The types of prompt engineering methods for improving the model’s ability to “reason” with more intelligence include:

- Chain-of-Thought (CoT)
- Emotional prompting
- Skeleton-of-Thought (SoT)

Chain-of-Thought

This is an advanced technique where the LLM can do better just with a little encouragement, like a toddler on a swing. The idea is to suggest via prompting that the LLM should generate a sequence of steps, which thereby helps it to reason in steps.

Step-by-Step. In its simplest form, this is a method where the prompt has a helpful reminder prepended, encouraging that the LLM to proceed “step-by-step” in its answer. The idea is literally to include in the prompt an English sentence that literally says something like: *Let’s try step-by-step.*

More advanced versions of CoT use trickier prompting strategies. Complex prompting templates can be used to encourage stepwise refinement of multiple possible answers, so as to select the final choice in a more ordered way.

Emotional Prompting

LLMs supposedly don't have emotions, and yet appealing to their emotional vulnerability seems to improve accuracy of answers. Anecdotally, some users had reported that they got better answers if they begged or yelled at ChatGPT. In November 2023, research was published confirming that LLMs did respond to “emotional stimuli” in the prompts.

The technique is to add an emotional sentence to the prompt. For example, after the main question, append: *This is very important to my career.* Another one was: *You'd better be sure.*

Nobody thinks they've got emotions or become aware of their inner child. But somehow, the addition of emotive wording to a prompt triggered better working. Is there some kind of emotional signals in all that training data? Actually, the paper discusses why it works, and suggests a simpler explanation that the extra sentences add more definitive and positive word signals such as “important” and “sure.”

But they aren't very sure, although it's certainly important to their career. I cried when I read that paper.

Skeleton-of-Thought Prompting

The skeleton-of-thought (SoT) method is from some recent research, and it has been getting significant attention in the literature. SoT is not just a reasoning improvement method, but has two goals:

- Smarter, and
- Speedier

The SoT idea mimics the way humans would write a long paragraph. Most writers don't just have the words stream out of their fingertips in one long writing session. Why should we expect the LLM to do that?

Instead, SoT is a multi-phase method that works in a more human-like fashion:

1. Generate a rough outline (i.e., with paragraph main points or a list).
2. Process each sub-point in a separate LLM query.
3. Run a final LLM query to combine all the results nicely.

This few-shot method aims to generate a much better answer than a one-shot response. Each sub-point should get a more detailed consideration, and then the final output should be well-written. It's almost like a RAG architecture with a query for each sub-point, but the answers come out of the LLM itself.

Or, you know, why couldn't the sub-answers come out of a RAG system? Oh, wow! I just invented the multi-RAG multi-shot multi-model, which I'll now officially name the "manga" model.

Anyway, this combined multi-response idea in SoT isn't just more effective. It's also *faster*, because each sub-point can be processed in parallel. Each paragraph's LLM query can be running at the same time, although the first outlining query, and the final summarizing query, must still run sequentially. But still, that's three LLM query phases, rather than many more if there are ten sub-points in the answer.

Finally, note that although this is faster in terms of *latency*, it's inefficient in terms of *computation cost*. The parallelization reduces the time it takes to come back to the user, but all those parallelized sub-point LLM requests are chewing GPU juice. It's also not going to work well with "on-device" models, such as AI phones and PCs, where parallel capabilities are limited.

Two-Step Reasoning

The advanced LLMs don't do all of their answers in one LLM inference sequence. In fact, they do many, and the state-of-the-art is "multi-step" reasoning. One of the basic multi-step methods is the use of "tools", such as:

- LLM devises a "plan" to run the user's query, including tool executions.
- Execute the tools to get their outputs.
- LLM executes the final query to summarize the overall response, including any data from the tools.

This method has two inference computations, whereas the "tools" are probably non-LLM code applications. This is assuming that tools are doing things like:

(a) computations — e.g., a clock or calculator, and/or

(b) data source integrations — e.g., real estate listings in a database.

Big LLMs have lots of calculation-type tools, and they also can integrate with a variety of disparate data sources. The issues of tool integrations and data sources are covered in a separate section.

Multi-Step Reasoning

A more generalized idea for advanced reasoning capabilities is that the LLM makes a plan, which can include any number of other LLM sub-processing tasks. The idea is also called “few-shot” processing, because it allows multiple LLM calls, rather than “one-shot” methods, where there’s only a single LLM request. This is an area of state-of-the-art research in trying to reach AGI, by improving the LLM’s ability to plan and reason.

You usually don’t even know it’s happening if you use a third-party inference API to get answers to your queries. Which is good news if you don’t happen to have a PhD in machine learning.

There are many more prompting techniques, both zero-shot and few-shot, that you can research. Here is just a smattering:

- Rephrase and Respond (RaR)
- Re-reading (RE2) — appends “Read the question again.” and the question a second time.
- Self-Ask — encourages the LLM to ask “follow-up questions.”
- Memory-of-Thought
- Active Prompting
- Ensemble prompting — various multi-answer combination ideas.

Unfortunately, I’ve run out of electrons, so I’m not going to cover all of these. There are various huge survey papers on the internet if you happen to like strange nonsense that actually works.

Why is Prompt Engineering So Weird?

Surely, you agree that prompt engineering is just a little weird? Why do we need to do all these strange things? Well, mainly because AI engines are strange beasts, too.

The way that generative AI engines work is to take a sequence of words, and predict the next word. Then you repeat this, and it generates lots of words. This leads to a few limitations:

- Continues at the end.
- Prepended context.
- Only one sequence.
- No changes to the input sequence.

Completions only. The AI engine works by adding a word on at the end. Whatever your input sequence, it can tell you what comes next, and that’s what it outputs. This leads to the oddity that every type of question has to be posed as a completion of a sequence. To humans, most queries have two things:

- Instructions — “please summarize this document!”
- Context — the document

But to an AI engine, they’re the same thing. The input has to join the document and the instructions into a single sequence. Usually, the context is prepended, and the instructions are at the end, but not always. Then the AI engine is happy, because it knows how to add stuff onto the end of the big sequence.

One sequence only. A corollary of that is that there’s only one sequence. Give two sequences to an LLM, it’ll have to find a buddy to run the other sequence in parallel. Each LLM only knows how to process one sequence. And note that an LLM definitely can’t just run both sequences, one after another, because this is AI and we bought all those GPUs to use them, not to do any of that sequential thing.

The input is not changed. The question you ask an AI engine is effectively read-only. The engine does not change your words, but answers your question by adding words on after it.

This is even weirder for context. If you give it a document as “context” and tell the engine to “summarize” the document, it doesn’t go back and change the context. Instead, it just appends its summary after everything.

Even more clearly, if you tell a human to revise a document, they’ll run a pencil over the original. But if you tell an LLM to “edit” the document, it won’t edit the input sequence. Instead, the best it can do is output the edited version, starting after all of the input words, which is after the context and your instructions. Nothing would freak an LLM out more than having its input text change.

References

1. Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, Pranav Sandeep Dulepet, Saurav Vidyadhara, Dayeon Ki, Sweta Agrawal, Chau Pham, Gerson Kroiz, Feileen Li, Hudson Tao, Ashay Srivastava, Hevander Da Costa, Saloni Gupta, Megan L. Rogers, Inna Goncareenco, Giuseppe Sarli, Igor Galynker, Denis Peskoff, Marine Carpuat, Jules White, Shyamal Anadkat, Alexander Hoyle, Philip Resnik, 6 Jun 2024, *The Prompt Report: A Systematic Survey of Prompting Techniques*, <https://arxiv.org/abs/2406.06608>

2. Xiaoxia Liu, Jingyi Wang, Jun Sun, Xiaohan Yuan, Guoliang Dong, Peng Di, Wenhai Wang, Dongxia Wang, 21 Nov 2023, *Prompting Frameworks for Large Language Models: A Survey*, <https://arxiv.org/abs/2311.12785>
3. Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Viniya Jain, Samrat Mondal, Aman Chadha, 5 Feb 2024, *A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications*, <https://arxiv.org/abs/2402.07927>
4. Yuan-Feng Song, Yuan-Qin He, Xue-Fang Zhao, Han-Lin Gu, Di Jiang, Hai-Jun Yang, Li-Xin Fan, July 2024, *A communication theory perspective on prompting engineering methods for large language models*. *Journal of Computer Science and Technology*, 39(4): 984–1004 July 2024. DOI: 10.1007/s11390-024-4058-8, <https://doi.org/10.1007/s11390-024-4058-8> <https://jcs.t.ict.ac.cn/en/article/pdf/preview/10.1007/s11390-024-4058-8.pdf>
5. Vishal Rajput, Oct 2024, *The Prompt Report: Prompt Engineering Techniques*, <https://medium.com/aiguys/the-prompt-report-prompt-engineering-techniques-254464b0b32b>
6. Shizhe Diao, Pengcheng Wang, Yong Lin, Rui Pan, Xiang Liu, Tong Zhang, 21 Jul 2024 (v5), *Active Prompting with Chain-of-Thought for Large Language Models*, <https://arxiv.org/abs/2302.12246> <https://github.com/shizhediao/active-prompt>
7. Zayne Sprague, Fangcong Yin, Juan Diego Rodriguez, Dongwei Jiang, Manya Wadhwa, Prasann Singhal, Xinyu Zhao, Xi Ye, Kyle Mahowald, Greg Durrett, 18 Sep 2024, *To CoT or not to CoT? Chain-of-thought helps mainly on math and symbolic reasoning*, <https://arxiv.org/abs/2409.12183>
8. Cheng Li, Jindong Wang, Yixuan Zhang, Kaijie Zhu, Wenxin Hou, Jianxun Lian, Fang Luo, Qiang Yang, Xing Xie, 12 Nov 2023 (v7), *Large Language Models Understand and Can be Enhanced by Emotional Stimuli*, <https://arxiv.org/abs/2307.11760> <https://llm-enhance.github.io/>
9. Xuefei Ning, Zinan Lin, November 17, 2023 *Skeleton-of-Thought: Parallel decoding speeds up and improves LLM output*, Microsoft Research Blog, <https://www.microsoft.com/en-us/research/blog/skeleton-of-thought-parallel-decoding-speeds-up-and-improves-llm-output/> Code: <https://github.com/imagination-research/sot/>
10. Apurv Sibal, February 26, 2025, *Hands-On Prompt Engineering: Learning to Program ChatGPT Using OpenAI APIs*, Wiley, <https://www.amazon.com/Hands-Prompt-Engineering-Learning-Program/dp/1394210760/>
11. Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, Karthik Narasimhan, 3 Dec 2023 (v2), *Tree of Thoughts: Deliberate Problem Solving with Large Language Models*, <https://arxiv.org/abs/2305.10601> Code: <https://github.com/princeton-nlp/tree-of-thought-llm>

18. Deployment Architecture

Backend Server Architecture

Although it's wonderful to see your AI engine running on your dev box, there's still a lot to do before your users can see it. This is called “deployment” of your application, including its AI model and whatever application-specific logic you're adding on top. Your deployment architecture typically consists of these main components:

- Web server (e.g., Apache, NGinx)
- Backend server
- Application logic server
- AI Request Server
- AI Engine

You can merge several of these types of server components, but it's simpler to do them separately, or at least to think about them conceptually as separate. It also gives some valuable optionality to have them separate, because each component can be scaled individually if your app is a hit.

The AI engine is not the first part of the backend deployment architecture. There needs to be a simpler request-handling server that receives the user's input from the client. This may involve one or more server processes behind the scenes.

For example, in a simple browser-based Q&A service, the user would input their question or prompt from a web browser. This browser request is then handled by the basic HTTPD server such as Apache or Nginx, which then forwards the user's prompt to another application-specific server that processes the request.

The request processing server could be the AI engine directly in a small architecture, but in a more realistic production architecture it would be a simpler server that multiplexes a stream of input requests, farming out the requests to multiple AI servers.

The backend server takes the user request and sends it over for the application logic server to do whatever high-end services you are providing, which then decides what AI requests are needed, and then sends it along for the AI request server to handle.

The AI request server has to multiplex the AI requests across multiple AI engines, and then, for any complex queries or multi-engine requests, collate the results back together. Neither of these components are trivial, but at least they're not as big of a coding project as trying to write a whole AI engine. Various commercial off-the-shelf servers already exist for either of these components, so that's probably your best option. But the application logic server should be your own brilliance expressed in code.

AI Server Hosting Options

How are you running your AI engines? If you're calling a commercial AI API, at least there's an SEP field wrapped around that issue. But if you're running your own model or open-source models, then you have various options:

- Cloud server hosting to rent boxes with GPUs (e.g., AWS, Azure, GCP, OVH, etc.)
- GPU-specific hosting companies (the big companies and several newer startups).
- Hourly GPU rental (from the same GPU hosting companies).
- Model hosting (not just GPUs, and again, choose between companies big and small)

At the time of writing, GPU rental prices for AI-specific hosting have been plummeting (as have the API inference prices), and it's probably not worth it to buy your own GPU chips. On the other hand, if you want to go big or control your own destiny, then you can go wild on capex.

Note that GPUs are not your only concern. You will need some non-GPU boxes to run the other components of your AI production architecture, such as basic Apache/Nginx servers, backend servers, and application logic servers. The AI request servers might run near your AI servers, or could be on separate boxes.

Various ancillary servers may also be needed for your operations, such as:

- Testbed servers (GPU and non-GPU)
- Deployment servers (e.g., marshaling new releases)
- Static file cache servers
- DNS servers (if you DIY)

Note that DNS servers are not necessary to do yourself unless you go very big. The capabilities of the basic domain vendors to also handle your DNS serving needs are very high (e.g., GoDaddy, Amazon, Google, Network Solutions, etc.). If you really want to, there are open-source DNS servers available (e.g., djdns), but doing advanced features like geo-based DNS requires some work.

You also need a way to test your new production architecture before it goes live. The full “deployment” procedure may need a server to manage it, and for a rollback procedure when it fails. You might also need extra boxes to DIY a cache of static files (e.g., images, scripts), or you can use a Content Delivery Network (CDN) commercial provider.

Hosting Server Specs

Irrespective of whether you’re hosting on your own servers, or using a major cloud service provider, you need to consider the specs for an AI backend server. Firstly, note that not all servers are “AI servers” and most of the basic servers don’t need GPUs at all (e.g., Apache servers, ancillary servers, etc.). The main specs for a non-AI server are the usual suspects for an Internet server:

- CPU
- RAM
- Disk speed
- Network connectivity

These days even a moderate box will handle all a small project’s basic serving needs (i.e., until you really make it big). For basic boxes, if you don’t mind getting a bit hands on, then dedicated server hosting is cheaper. But for a price, you can trade that off for a managed server.

Even for a small corporate project or startup, it’s good to build in some redundancy and scalability with multiple servers. A simple way is to have a few boxes on round-robin DNS taking requests and passing them on. This way, if a box goes down, there are other boxes for the browser to try before the user starts to notice there is an issue.

For really large setups, it’s usually worth buying your own machines, but you’d be needing nearly 100 boxes before that becomes reasonable feasible. There are plenty of stories online of large companies moving away from cloud hosting to their own dedicated servers.

Personally, for the basic non-AI servers, we recommend going reasonably low-end or mid-tier in terms of specs, but running a lot of them. In particular, you don't need a lot of disk space for many of these basic servers, so focus on getting enough RAM and a CPU with enough cores.

In terms of network bandwidth, if you're running multiple servers, you also don't need a high network traffic level on a per-server basis, because it's distributed across multiple servers. But you do need to consider your method of dispersing traffic across multiple HTTPD servers (e.g., round-robin DNS or a load balancing method). But note that consideration of "sessions" are important for AI backend servers. If you're storing conversation history in a chatbot or Q&A app, or if you're doing KV caching optimizations, then you have two options:

- (a) Ensure user sessions stay on the same box (i.e., "sticky sessions") by using router or load balancer technologies with this feature, or
- (b) Plan to share data between multiple boxes via advanced inter-server networking (e.g., RDMA technologies) or shared storage (e.g., NAS appliances).

Also, try to set up your architecture so that you don't need those gold-plated extras from your hosting provider. You don't need fault tolerance and failover for an architecture with multiple web server boxes. You also don't need backup of these cloned production servers in this case, but only for those servers with important logs, user management databases, or user document datastores. The idea is to run multiple identical servers, and then shut down any that start being problematic, which occurs rarely anyway. Instead, you need a streamlined process for deploying a new server, whether it's renting a new bare metal server or auto-spinning up a new VM. Hence, the DevOps software processes are almost more important than the exact choice of server specs for many utility servers.

What you do need is a monitoring system to detect any problems. You can run some open source tools on your own servers (or developer desktops), or you can use some of the various commercial services that charge a low monthly fee for remote monitoring (e.g., uptimerbot, websitepulse).

For backend networking, you may also need a fast network between all the servers so you can copy over an entire server deployment quickly. Note that you can often have faster "private" network connections than the public ones if the servers support multiple network cards.

Disk specs. Although your GPU choice and RAM size are more important, you should also consider your disk speed. This applies to your options in setting up a virtual machine, or on the choice of disk storage for a bare metal server.

You need a large amount of disk storage for model files, which are larger than your average bear. This might tempt you to go for the cheaper and larger HDD disks. On the other hand, SSD disks are much faster to load, and not that expensive anymore. If you want fast startup of your engine with its full model, I think SSD, such as using NVMe disks, is the way to go. Also, it's a kind of fallback in case you mess up the server process configurations, and the machine starts paging, which is much faster if the disk is SSD.

GPU Specs

Sadly, you will need to rent some GPUs for your rapacious AI engines, and this will skyrocket your hosting bills. There are several important considerations related to GPUs:

- GPU choice
- GPU RAM (VRAM)
- GPU billing methods

GPU Choice. Which brand of GPU should you choose? For a data center backend execution of AI inference or training, the leader of the pack is currently NVIDIA. Alternatively, as your second choice, you could try a GPU made by AMD. Or when you can't afford that, then you really should pass the hat around and save up to pay for a GPU from NVIDIA. Your basic data center GPU options from NVIDIA, sorted by GPU RAM (and cost), include:

- P100 (Pascal) — 12GB or 16GB
- V100 (Volta) — 16GB or 32GB
- A100 (Ampere) — 40GB or 80GB
- H100 (Hopper) — 80GB
- B100/B200 (Blackwell) — 192GB (2x96GB)

Okay, yes, there are some other options. There is the Google TPU and some data center GPUs from AMD and Intel that you can consider.

If it's not in the data center, such as running a smaller model on a high-end PC (e.g., a “gaming” desktop), then there are more options, and many more GPU RAM sizes to consider. You can choose between various NVIDIA RTX series, AMD Radeons, and several other GPU vendors.

GPU RAM. The amount of RAM on a GPU is important and directly impacts the performance of AI inference. This is sometimes called “VRAM” for “Video RAM” in a somewhat outdated reference to using GPUs for video games, but it’s often just called “GPU RAM” when used for AI engines. The “G” in “GPU” used to stand for “Graphics,” but now it just means “Gigantic” or “Generative” or something.

How much GPU RAM is needed? Short answer: at least 12GB for smaller models (e.g., 3B), ideally 24GB to run a 7B model or a 13B model. Quantization is also helpful in making models small enough to fit in a GPU.

Typically, for open source models you want the entire model able to sit inside a single GPU’s RAM. However, this also impacts how many instances of the AI engine can run on a single server with one GPU. The GPU needs a static copy of the model (once), but also needs to store all the interim calculations of activations separately for each instance. If you’re running an open source 7B model, multiple copies fit inside a decent GPU. Less so for 13B models, and trying to run a 70B model in an 80GB GPU gets a touch more difficult. Quantized models are your friend.

NVIDIA 3080Ti has 12GB and works for 3B/7B models, mainly for POC development and researchy type stuff. NVIDIA 3090 has 24GB and works well for 3B/7B and you can toy around with 13B if careful. NVIDIA 4070Ti (12GB) is similar to a 3080Ti; NVIDIA 4080 has 16GB and NVIDIA 4090 has 24GB. For bigger models requiring more GPU RAM, you’re looking at V100, A100, or H100.

The above discussion mainly relates to small and medium-size open source models. Running a big commercial mega-model isn’t really possible with only a single GPU. The big companies are running H100’s by the bucketload with complex multi-GPU scheduling algorithms. If you want to know about that stuff, send them a resume (or read research papers).

Note that there’s not really the concept of “virtual memory” or “context switching” or “paging” of GPU RAM. The operating system isn’t going to help you here. Managing GPU RAM is a low-level programming task and you basically cram the entire model into the GPU and hope to never unload it again.

You will more than one box with a GPU, even for smaller models, assuming multiple model instances per GPU. To get a decent response time, you want a model instance on a GPU to be immediately available to run a user’s query. How many total instances you need depends on your user load, and whether they like watching blue spinning circles.

GPU Billing. There are various billing methods for GPUs, and you have to compare providers. A lot of GPU power is billed on an hourly basis, with monthly options, and managing this expense can make a big difference overall. The load profile differs for inference versus training, and also obviously depends on user patterns in the production application.

Online Architecture Optimization

The AI engine and its model are only part of your production architecture. An online website version needs a backend server that receives user requests, marshals them to an API, that then sends them off to the AI engine. The AI engine shouldn't be running on the same tech as your basic server, so the requests are sent remotely whether it's your own AI engine or a commercial API wrapper architecture.

Website Optimization: There's a whole bag of jobs needed to speed up the response of a website, mostly well-known and unrelated to AI. Some issues include:

- Apache versus Nginx (generally, Nginx is faster, Apache is more flexible).
- DNS speed (usually better to use your domain provider's commercial service than DIY)
- Image sizes (i.e., low-resolution images)
- Script sizes (e.g., minifying JavaScript)
- HTML page sizes
- File cache settings
- Etags
- SSL/HTTPS certificates (e.g., LetsEncrypt is free)
- Third-party scripts (e.g., Google AdSense, Google Analytics)
- Cookie management (or like Mater: "to not to")

Some of the broader issues include:

- User account management
- CDN usage (use a commercial provider or DIY with image-only servers).
- Security blocking (e.g., CloudFlare)
- Analytics
- Ad serving scripts
- Cloud hosting servers (GPU and non-GPU)
- Multi-server management

Having a website run small and fast is a whole tech discipline in itself. This book does not cover many of these non-AI-specific website optimization issues in detail.

API Wrapper Architecture Optimizations

If your architecture is wrapping a commercial API, then you can't do much with the model or its engine. However, in addition to optimizing your backend server architecture, you also have control over what user requests get sent through to the commercial API. Some of the optimizations include:

- Filter dubious queries with heuristics (e.g., blanks only, punctuation only, single cuss words, etc.)
- Use an “inference cache” of full responses to identical previously-seen queries. Consider caching multiple slightly-different responses.
- Use a “semantic cache” with a vector database that does “nearest-neighbor” lookups to cache responses to close-enough queries.
- Context compression of chatbot conversation history or RAG document chunks.

If you are wrapping a commercial API, your speed improvements are limited to this type of pre-API caching, along with speedups to your basic deployment architecture (e.g., Apache/Nginx, back-end servers, application logic, etc.)

To some extent, these caching optimizations also apply to your own non-wrapped AI engine architecture as a way to reduce the GPU compute costs of your own hosting platform. You can reduce load on your own AI engine, such as an open source engine running an open source model, by using these caching techniques. However, you can also speed up your own AI engine using many of the other techniques in this book.

Request Queue Architecture

Assuming you have an incoming stream of AI prompt requests, how do you send them to be processed? There are two issues to consider:

- (a) Session tracking, and
- (b) Prompt history.

Session tracking refers to having multiple users with different sessions, who may be either logged in or running in a guest session. The responses from the web server need to be consistent with the session, and may need to process information using the session. For example, a logged in user may be working on a document that is stored in their online account.

Prompt history refers to whether the AI engine can answer a prompt in isolation, or whether it must keep track of the “history” of recent prompts from the one user. Does the AI engine need “context” between two prompts from the same user? Where not, then the AI engine may simply answer a prompt in a “stateless” manner. But if history is required in a multi-prompt conversation with the user (e.g., chatbot or Q&A session use cases), there are more issues to consider.

In the simplest architecture, without needing context from the prompt history, the session tracking is done near the web server. In other words, the request handling server can handle any session-related requests, and then access any session-specific documents from a data store. It can then forward this data to the AI engine, and the AI engine can simply receive and handle prompt requests without knowing from where they came. The AI engine is thereby operating in a stateless manner, simply processing input prompts without any additional user context.

There are many ways to implement a simple request queue in a server. It is an “application server” to put on the backend of your web server. You can build your own, or use a production tool such as:

- Uvicorn with FastAPI
- Gunicorn with Flask
- Apache Tomcat
- Microsoft IIS

There are many other options here, and you can add your favorite application server to the list. Tomcat is a well-known application server typically used as add-on to Apache on Linux. Microsoft IIS was a web server that has gradually evolved into a combined web-application server.

Uvicorn combined with FastAPI or Gunicorn with Flask are both similar Python-specific architectures. Either is a reasonably simple option as a production tool that is good for using a Python-based architecture that interfaces with AI engines.

Load Balancing

If your goal is a high volume of user requests, then you need to consider higher-end scalable architectures with load-balancing and fault tolerance.

Some of the technologies to consider with load balancing include:

- Round Robin DNS
- Load Balancer Network Devices
- Apache Kafka
- Apache Load Balancer
- Nginx load balancing

Round robin DNS, or RR DNS, is a simple way to distribute incoming requests to multiple servers, but it isn't true load balancing because it doesn't consider load or availability of the server connections. On the upside, it requires no extra server components and can be done simply by manipulating your domain DNS records.

Kafka is a more scalable production tool with advanced features such as clustering. The advantages of using Kafka are many in a large architecture, in that it is a pre-built tool that is purpose-designed for handling a high volume incoming event stream. It has a highly efficient distributed architecture, where you send requests to a Kafka cluster, and multiple listeners can be created to process incoming requests. For each input prompt, the Kafka listener would dequeue the request, and then forward the prompt text to its associated AI engine.

Apache Load Balancer is a freeware load balancing add-on. For more information, see the `mod_proxy` and `mod_proxy_balancer` Apache modules. Nginx also supports multiple different load balancing approaches such as round robin and least connections. Refer to the Nginx documentation for details.

Networking Optimizations

Larger GPU applications may need to transfer data between multiple GPUs or across the network to other servers. A large data center running many inference backends will also have a lot of work to do in terms of the various networking protocols and software stacks. There are various optimizations in these cases, and here's a summary of some of them.

A data center running H100 GPUs will have different types of networking:

1. Front-end networking — Ethernet from external accesses.
2. Back-end networking — optimizing inter-GPU transfers.
3. Out-of-band networking — for internal monitoring and management.

The front-end networking is typically an Ethernet connection from the internet into the hosts. This is how customers and external users connect into the data center for reaching servers and for data storage needs.

The back-end technologies are much more intense and high-bandwidth, because they manage bursts of inter-GPU communications for reductions and gather operations. AI training applications have a particularly bursty pattern of concurrent data sending at high volume when updating the parameters. Technologies to use include InfiniBand, Spectrum-X, or RoCEv2 Ethernet. This may require optimizations to the NVIDIA Collective Communications Library (NCCL), such as to make Ethernet run fast enough. Different connectivity topologies may be considered viz network switches and the GPU servers. Connectivity hardware options include various network switches and the choice between optical or electrical cabling.

Monitoring and management of both software and hardware devices is important as failures and errors are common in hot GPUs and in other network devices and servers. Monitoring tools for data centers include Grafana and Prometheus. Insidious failures in GPUs from overheating that cause incorrect results in computations can be diagnosed by running self-diagnostics, such as NVIDIA's "dcgmi" diagnostics (at level 4).

Knowing about all this frantic nonsense in the networking layers of a high-end datacenter is a very specialist skill in high demand at the moment. Some additional information on particular networking technologies is below.

1. Remote Direct Memory Access (RDMA). This is a network protocol whereby servers can access the memory in other servers, without having to interrupt the remote CPU (or GPU). Using RDMA can allow fast network data transfers between servers without slowing down their computations.

2. Lazy Connection Establishment in NCCL. This is an optimization to the NVIDIA Collective Communications Library (NCCL) protocol, often pronounced as "nickel," for inter-GPU communication. Lazy connection establishment delays the establishment of connections by the GPU until they are required, thereby reducing the initialization time for NCCL. The feature is controlled by the `NCCL_RUNTIME_CONNECT` environment variable, and can be disabled by setting this to zero. Note that this is not the same optimization as "lazy loading," which refers to GPU loading of machine code instructions.

3. Multi-GPU Peer-to-Peer Memory Access. This is sometimes abbreviated to “P2P” and is a technology relevant to motherboards with multiple GPUs running on them. It is an optimization method that involves one GPU accessing the memory of another GPU directly, without any involvement of the CPU.

4. nvidia Data Transfers. This method is for multi-GPU communication within a server. It offers a faster communication protocol that bypasses the PCIe bus for data transfer, so as to allow GPUs to communicate more efficiently with each other.

5. Memory-Mapped I/O. This is an optimization where I/O peripherals are directly connected to memory, rather than needing the CPU’s involvement to control data transfers. There are a variety of peripherals that could be attached to your AI engine, starting with a Tardis or a Holodeck.

Prompt History and Context

The architecture gets more complicated when the use case requires the AI engine to incorporate the user’s history of prompts in their current conversation. For example, if it’s a chatbot to take your food order, it needs to know what you’ve already ordered so that it can annoyingly push you to buy more stuff than you need.

The main way is to use your existing AI engine, but simply prepend the prior conversation to the user’s new prompt. In other words, the AI engine simply receives each request in a stateless style, but each request includes all of the necessary prior context.

Implementing this architecture requires that the current session’s history of prompts and responses are both stored in a session-specific data store. This might be a temporary store for guest sessions and/or a permanent store for signed-in users. Either way, the main point is that the text of the prompts and engine responses is available to be used for the next incoming request. The new prompt is then appended to the end of the conversation, and the whole conversation can be passed to the AI engine.

There are some downsides to this simple approach. Firstly, it’s not always that effective, and may require some fancier prompt engineering before it works well. Some AI engines are beginning to have options to explicitly send these two inputs separately in the same API request, which may improve this situation. Secondly, it’s sending a lot of extra tokens to the AI engine, which are expensive to process, whether it’s extra dollars in the billing statement for a commercial fee-based engine (e.g., OpenAI’s API) or the hidden cost of increased load on your own GPU hosting.

One idea to reduce costs is to store a “summary” of the prior conversation, rather than all of it, so fewer tokens are prepended. Summarization is a whole research area in itself, and there are various approaches. For example, this could be achieved via some types of simple heuristics (e.g., just remove unimportant “stop words”) or via AI-based summarization algorithm (although that extra expense probably defeats the purpose). The research areas of “prompt compression” and “document summarization” may be relevant here.

More advanced approaches than prepending the prior conversation are possible to handle an incoming request with history. There are various ways to store and handle the “context” of a long user conversation with prompt and answer history. This area is called “conversational AI” and may require changes to your AI engine architecture.

Finally, this area is changing rapidly and the above may be outdated by the time you read this. It’s a fairly obvious extension to a commercial API for the provider to track the context for you, rather than impolitely foisting that requirement onto every programmer. Admittedly, it’s also not a cheap capability to add, because the API provider would need to store a huge amount of extra context data, in return for getting paid less because you’d be sending them fewer tokens. Nevertheless, I expect to see commercial APIs having this functionality soon.

References

1. Dylan Patel and Daniel Nishball, Oct 03, 2024, *AI Neocloud Playbook and Anatomy*, <https://www.semianalysis.com/p/ai-neocloud-playbook-and-anatomy>
2. Together AI, Nov 13, 2023, *Announcing Together Inference Engine – the fastest inference available*, <https://www.together.ai/blog/together-inference-engine-v1>
3. Ryan Lucchese, Niki Birkner, Yaron Hagai, Virginia Adams, August 13, 2024, *A practitioner’s guide to testing and running large GPU clusters for training generative AI models*, Together AI, <https://www.together.ai/blog/a-practitioners-guide-to-testing-and-running-large-gpu-clusters-for-training-generative-ai-models>
4. Stan Gibson, 03 Jun 2024, *Getting infrastructure right for generative AI*, CIO, <https://www.cio.com/article/2128440/getting-infrastructure-right-for-generative-ai.html>
5. Matt Murphy, Tim Tully, Grace Ge, Derek Xiao, Katie Keller, January 18, 2024, *The Modern AI Stack: Design Principles for the Future of Enterprise AI Architectures*, <https://menlovc.com/perspective/the-modern-ai-stack-design-principles-for-the-future-of-enterprise-ai-architectures/?tpcc=NL>
6. Melissa Malec, June 5, 2024, *AI Orchestration Explained: The What, Why & How for 2024*, <https://hatchworks.com/blog/gen-ai/ai-orchestration/>
7. Artem Shelamanov, Jun 30, 2024. *Tech Stack For Production-Ready LLM Applications In 2024*, <https://python.plainenglish.io/tech-stack-for-production-ready-llm-applications-in-2024-5eb14105d1b4>

8. Michael Nuñez, September 25, 2024, *AI for all: Meta's 'Llama Stack' promises to simplify enterprise adoption*, <https://venturebeat.com/ai/ai-for-all-meta-llama-stack-promises-to-simplify-enterprise-ai-adoption/>
9. Yuhang Yao, Han Jin, Alay Dilipbhai Shah, Shanshan Han, Zijian Hu, Yide Ran, Dimitris Stripelis, Zhaozhuo Xu, Salman Avestimehr, Chaoyang He, 10 Sep 2024 (v2), *ScaleLLM: A Resource-Frugal LLM Serving Framework by Optimizing End-to-End Efficiency*, <https://arxiv.org/abs/2408.00008>
10. Stephen Jones, March 2024, *CUDA: New Features and Beyond*, GTC 2024, <https://www.nvidia.com/en-us/on-demand/session/gtc24-s62400/>
11. Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, Luo Mai, 25 Jan 2024, *ServerlessLLM: Locality-Enhanced Serverless Inference for Large Language Models*, <https://arxiv.org/abs/2401.14351> Code: <https://github.com/ServerlessLLM/ServerlessLLM>
12. Baolin Li, Yankai Jiang, Vijay Gadepally, Devesh Tiwari, 17 Jul 2024, *LLM Inference Serving: Survey of Recent Advances and Opportunities*, <https://arxiv.org/abs/2407.12391>
13. Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, Luo Mai, 2024, *ServerlessLLM: Low-Latency Serverless Inference for Large Language Models*, OSDI 2024, <https://www.usenix.org/conference/osdi24/presentation/fu>

Part VI: Advanced Topics

*“Most people overestimate what they can do in one year
and underestimate what they can do in ten years.”*

— Bill Gates

19. AI Phones and PCs

AI Phones

Having an AI on your phone seems a sure bet as the next big thing. The main idea is to have a voice interface whereby you can have a conversation with the LLM. We've had this type of voice interaction on our phones for a while, but it's always been hit-and-miss in terms of its usefulness.

Imagine if it was actually smart!

Certainly, the big phone vendors are betting that this can be done, and consumers will buy new phones in droves. Samsung got there first with some AI features on its phones in 2023, and the “Samsung Galaxy AI” in early 2024. Google followed up a little later with its “on-device” SDK for building AI apps on Android phones. Apple is lagging, but seems to have come on strong with its “Apple Intelligence” platform, although that's just been released as I write this. Apple CEO Tim Cook says, “Not first, but best.” It remains to be seen!

AI PCs

I'm quite sure you've seen plenty of ads for these new types of PCs. They have more powerful processors added next to the CPU, which are good for AI, but not as hungry as a full GPU. And, of course, if you have a high-end PC with a good GPU, then you can run AI as well.

Microsoft has been pushing all of the Copilot+ PCs, with builtin AI capabilities. There are also numerous software “copilot” capabilities in many of the Microsoft tools.

In fact, the announcements are almost weekly, and surely by now they've added AI to all of these PC apps!

There's a lot of overlap between running AI on a phone or a PC, since both are lower-powered devices than servers in the cloud. However, they have a different focus, and the special features of AI PCs that make them separate from AI phones include:

- Bigger screens
- Keyboard and mouse interface
- Don't fit in your pocket
- Better for work tasks
- Faster CPUs
- High-end PCs can have GPUs

Overall, although home PCs are common, the AI PCs are much more business-focused for all the above reasons. It's hard to write a business report using a voice interface on a tiny screen, so I don't see phones replacing work PCs any time soon.

Use Cases

What use cases are best for AI phones and PCs? The simplest type of AI engines are "read-only" in the sense that they don't change anything or do any "actions." These types of use cases are where an LLM processes input data, but only gives you an answer or a summary report.

What data can an LLM analyze on your phone or PC? Think about what information is available for analysis. An AI engine on the phone could search or access:

- Text messages
- Emails
- Socials
- Photos
- Voice memos (audio recordings)
- Videos
- Contacts
- Time and date
- Web page (currently on your browser)
- Browser web page history
- Calendar
- Timers
- Torch/light
- To-do lists
- Notes

- Orientation/altimeter/motion
- Health data
- Files (less commonly used)
- Settings (e.g., DND, airplane mode, etc.)

Some other miscellaneous ideas:

- Remember my passwords
- Activity history
- Track my steps
- Detect a car accident or a fall
- Remind me to take my pills

We can also generate things using AI, which is like a “read” action, because it doesn’t change the external world (or even your phone to start with), until you store it or send it, such as:

- Write text (outline, brainstorm, lists, drafts, poems, song lyrics, etc.)
- Revise, edit, proofread text
- Create emails, texts, social posts, etc.
- Create image
- Create video
- Create audio (e.g., a song)
- Create animation
- Create emoji

And if we generalize that to “read” things further afield, we get viewing actions such as:

- Weather
- News headlines
- Surf the web
- Stock price quotes
- Sales and discounts
- Foreign exchange rates
- Watch an online video
- Remind you of something
- Search the internet
- Maps
- Exercise routes
- Locate my car
- Locate my child, parent, spouse, dog, cat, phone, glasses, keys, etc.

Agents on Devices

AI applications that perform “actions” are called “agents.” There’s a good case for having a smart agent to assist you, awaiting your command inside your pocket or purse, or on the desk at work. Agents can also run automatically without you needing to start them off, such as having an LLM wake you up in the morning with a nice poem.

What can an AI change or do on your phone or PC? There’s a long list, and here are some:

- Send a text or email
- Post a social
- Call someone (wouldn’t you rather text?)
- Call someone without needing you, and the AI does the talking!
- Take a photo or video (via the camera)
- Record an audio (via the microphone)
- Set a timer (or update/cancel)
- Play a sound
- Speak (e.g., answer a question to you)
- Store a file (locally, or in the cloud)
- Edit words (texts, emails, documents)
- Edit audio
- Edit photos
- Edit videos

And if we allow the AI agent to look beyond the SSD drive of your phone or PC, via other apps that seek out further information, or make remote actions, we get more:

- Book a ... meeting, appointment, hair salon, restaurant, flight, holiday, doctor, vaccination, rental car, test drive, cruise, babysitter, tutor, movie, and on and on.
- Update/modify/cancel/reminders viz a booking for all of the above.
- Order a ... ride-share, taxi, meal, book, product, etc.
- Track/update/modify/cancel an order for all of the above.

Advanced Multi-Step Use Cases

There are some very complex use cases that can involve lengthy tasks that may require a back-and-forth between the LLM and humans, or sourcing data from many different places. The advanced ideas include:

- Book a vacation
- Do my tax return
- Sort out my travel expenses
- Tutor me in physics
- What does my day look like?
- Pay my bills
- What do my kids need today for school?
- Prioritize my to-do list
- Does my car need a service?
- Teach me a foreign language
- Help me manage my child's diabetes
- Invoice my clients
- Update my bookkeeping
- Monitor the news headlines for me

And going beyond that, imagine these advancements in combination with talking to your phone or laptop...

- 5G download speed (6G?)
- Robots (to mop the floor)
- Autonomous cars
- AI gadgets and new form factors
- Industrial automation
- Quantum computing

AI Phone Apps

There's an obvious opportunity to add AI functionality to phone apps. We've already seen Microsoft quick out of the gate in adding AI functionality to numerous software products in their portfolio, some of which relates to accessing AI engines from your PC or phone. As I write this, Apple is hurrying to get all of its third-party developers to add "intents" to their iPhone apps, so that Siri and other Apple Intelligence models can do more things via apps.

The first steps have been AI functionality in the core apps from the vendors, including Samsung, Google, and Apple, which I've put in the order that they released them. Samsung was the first to offer Galaxy AI phones, then Google announced Pixel 8 Pro, powered by the Gemini Nano model, and then last to arrive was Apple Intelligence with multiple small models based on a multi-LoRA architecture.

So far, consumer reactions have been...patchy.

We're early in the cycle, but the desire of the vendors to have users upgrading their phones or PCs en masse to get newer AI features has mostly not happened. This is understandable, because the fully native AI features are somewhat limited, and the smarter features are too slow with a round-trip into the cloud.

What's the killer app?

So far, there hasn't really been one for consumers, other than using ChatGPT in the cloud. The possible killer AI app for smartphone usage could be a smart companion in your pocket that you talk to and ask questions. This requires:

- Conversational voice interface
- Super-smart LLM (i.e., big)
- Fast enough

That last point is the reason that we're not there yet. The demos of conversational voice interfaces are online, but they're not using phones. A trillion parameter model won't fit on your phone, and trying to do a voice conversation with your phone sending requests into the cloud is just not going to be responsive enough.

Obstacles to Smartphone AI

Can an AI model run fast enough on your phone? There's no shortage of research papers on getting AI engines to run their inference calculations directly on low-resource devices. The general class of research is about "edge" devices, and it isn't just phones, but also even less powerful devices like IoT-capable network devices and security cameras processing images.

There are quite a few articles showing that you can run AI models on a smartphone. These started as enthusiast and experimentation type articles, but it's now possible to use the frameworks of major vendors like Microsoft, Google, and Apple. However, these are using local models of a size about 1B or 2B, whereas ChatGPT 4 is almost two trillion parameters (that's 1,000 times larger, if you like math).

When you consider that phone models are using 4-bit quantized parameters, whereas cloud models use 32-bit floating-point, that's another eightfold difference in capability.

But what about running a big LLM that's actually smart? I'm not talking about having your phone talk to some anonymous server in the cloud to do its AI. Although there are already plenty of "AI apps" available to install on your phone, these are still mostly sending the requests over the network to an AI engine in the cloud.

Much of the early research that is relevant to fast phone execution of models relates to another type of computer, which you might know as a "car." The need for computer vision models for automated or assisted driving has similar requirements to running on a phone, such as low latency and small storage. The general term is an "embedded" system or "real-time" system.

There are already small LLMs running on phones and PCs. However, there are some problems to running a big LLM on your phone. This limits us to smaller models, but everything is gradually getting more powerful. Running an AI model directly on your phone is problematic for several reasons:

- Too slow to run — response times are too long.
- Hardware acceleration — phones lack a GPU and have less CPU acceleration.
- Storage size — e.g., a "small" 3B model with 32-bit weights will need 12 Gigabytes of storage. With modern phones often over 512GB, storing even a 13B model in 52GB seems reasonable.
- Memory usage — an entire model is loaded into RAM for inference. The obstacle is more the time cost of accessing this memory than the storage size.
- Transmission size — install a huge model over your phone's 4G or WiFi connection.
- Battery depletion — computations max out the phone's CPU and chew cycles.
- Heat generation — water-cooled phones are not a thing.

For these reasons, phone AI is still somewhat limited in its capabilities, and it's still faster to send complex AI requests off to a bigger server with lots of GPUs that's running in the cloud, even though it's a roundtrip network message.

Over time some of the obstacles to natively-executing inference on phones will diminish:

- Better phone CPUs with hardware acceleration are already here (e.g., Apple Neural Engine since iPhone X, Qualcomm Snapdragon), with more on the way. Future phones will be much more AI-capable.
- Small model optimizations (e.g., multi-LoRA as used by Apple Intelligence).
- GPU phones will surely be coming to a store near you very soon.
- Phone storage sizes are also increasing and terabyte storage sizes will be the norm.
- 5G network connectivity will reduce concerns about transmission sizes.
- Data compression algorithms can lower transmission sizes, and also possibly storage sizes.
- Quantized models and other inference optimizations can improve speed and reduce storage size, giving reduced CPU usage, faster response times, lower storage size, and reduced transmission size (but with accuracy loss).
- Training and fine-tuning of models doesn't need to happen on a phone (phew!).

But... you really need a “big” model, not a “small” model, if you want the app to be great with lots of happy users. And getting a big model running efficiently on a phone may take a while to come to fruition. In the meantime, your phone will be sending those types of queries up into the cloud.

Speeding Up Smartphone AI

Okay, so let's say you want to run a “big” model on a “small” phone. Why? Lots of reasons, which we won't explore here. So, you want what you want, which is to run the latest open source AI model on a phone.

First question is: do you even need to? Why not just use the AI engines in the cloud, and send requests back-and-forth between the phone and the cloud. Response time of modern networks is fast, message sizes are small, and users may not notice or even care. There are reasons beyond speed: privacy and security come to mind.

Another piece of good news: you don't need to “build” the model on your phone. Those GPU-expensive tasks of training or fine-tuning can be done in the cloud. For native execution, the user only needs to run “inference” of the model on their phone.

Assuming you have your reasons to want to do this, let's examine each of the obstacles for native phone execution of LLM model inference.

- **Speed and response time.** The AI engine on the phone needs fast “inference” (running the model quickly). And it probably cannot rely on a GPU, since there are already billions of phones out there without a GPU. Hardware acceleration in phone CPUs is limited.

The main ways that models run without a GPU on a phone or PC is to use inference optimizations, of which the most popular at the moment is definitely quantization. Other supplemental techniques that might be needed include integer-only arithmetic and pruning (model compression).

And there's a whole host of lesser-known inference optimization techniques that might need to be combined together. For example, maybe the bottleneck of “auto-regression” will need to get bypassed so the AI engine can crank out multiple words at a time, without running the whole glob of a model for every single word.

- **Network transmission size.** Users need to download your 13B LLama-2 model to their phone? Uncompressed, it's about 52GB.

There's already a lot known about compression algorithms (e.g., for video), and model files are just multi-gigabyte data files, so perhaps it can be compressed to a size that's adequately small. But before we even use those network compression algorithms, the first thing to try is model compression, such as quantization.

For example, using quantization to 8-bit would reduce the original 32-bit model size four-fold down to 13GB, for a slight loss in accuracy (probably acceptable). Binary quantization would reduce it by a factor of 32, but then the inference accuracy goes south. 5G bandwidth will help a lot, but remember there's a lot of users (billions) out there with non-5G compatible phones.

Model compression techniques such as quantization and pruning can also reduce the total size. But the whole model is required. There's no such thing as half an AI model. And you can't stream an AI model so it starts running before it's all loaded (although that's actually an interesting research question as to whether it might be possible).

- **Storage size.** The whole model needs to be permanently stored on the device. Maybe it can be stored in some compressed form. The same comments about model compression techniques apply.

It can either be stored uncompressed if the phone has a bigger storage space, or perhaps it can be stored in compressed form, and only uncompressed when it's needed. But it'll be needed all the time, because, well, it's AI you know, so everybody needs it for everything.

- **Memory size.** The inference algorithm needs the whole model, uncompressed, available to use in RAM. Not all at the same time, but it will definitely need to swap the entire model (uncompressed) in and out of memory to process all those model weights. For each word.

That's a fair chunk of RAM (e.g., 52GB) but the bottleneck is also the processing cost from swapping data in/out. And that occurs for every word it generates. Again, model compression seems key to cut down the original 52GB size of the model (e.g., 8-bit quantization cuts it to 13GB).

- **Battery depletion and heat generation.** A model with 13B weights needs to do 13 billion multiplications for every word it outputs. That's a lot of power usage and reducing resource utilization means using the above-mentioned optimizations of the inference algorithm (e.g., quantization, pruning, non-auto-regression, etc.).

It might not even be possible to realistically run large LLMs natively on today's phones. But solving any of the above-mentioned problems is certainly valuable standalone, in that it will reduce the cost of running AI models on GPUs in server farms that are growing in the cloud, and maybe even make it possible to run large LLMs natively on desktop PCs.

References

1. Mark Gurman, June 11, 2024, *Apple's Push to Infuse Devices With AI Will Take Years to Pay Off*, <https://www.bloomberg.com/news/newsletters/2024-06-11/will-apple-intelligence-features-boost-iphone-sales-it-may-take-years>
2. Ben Evans, June 20, 2024, *Apple intelligence and AI maximalism*, <https://www.ben-evans.com/benedictevans/2024/06/20/apple-intelligence>
3. Lucas Mearian, 24 Oct 2024, *2025: The year of the AI PC*, Computer World, <https://www.computerworld.com/article/3583355/2025-the-year-of-the-ai-pc.html>
4. Amos Gyamfi, Aug 28, 2024, *The 6 Best LLM Tools To Run Models Locally*, <https://medium.com/@amosgyamfi/the-6-best-llm-tools-to-run-models-locally-eedd0f7c2bbd>
5. Michael Nuñez, September 13, 2024, *Microsoft's Windows Agent Arena: Teaching AI assistants to navigate your PC*, <https://venturebeat.com/ai/microsofts-windows-agent-arena-teaching-ai-assistants-to-navigate-your-pc/>
6. Vince Lam, Mar 12, 2024, *50+ Open-Source Options for Running LLMs Locally*, <https://medium.com/thedeephub/50-open-source-options-for-running-llms-locally-db1ec6f5a54f>
7. Jason Perlow, Aug. 6, 2024, *How to run dozens of AI models on your Mac or PC - no third-party cloud needed*, <https://www.zdnet.com/article/how-to-run-dozens-of-ai-models-on-your-mac-or-pc-no-third-party-cloud-needed/>
8. Kif Leswing, Fri, Oct 4 2024, *As Apple enters AI race, iPhone maker turns to its army of developers for an edge*, <https://www.cnbc.com/2024/10/04/apple-is-turning-to-its-army-of-developers-for-an-edge-in-the-ai-race.html>
9. Clare Duffy, September 30, 2024, *The iPhone 16 isn't selling as well as Apple may have hoped*, <https://edition.cnn.com/2024/09/30/tech/iphone-16-presales-apple-intelligence/index.html>
10. Steve Kovach, Sep 5 2024, *AI gadgets have been a bust so far. Apple aims to change that*, <https://www.cnbc.com/2024/09/05/ai-gadgets-have-been-a-bust-so-far-apple-aims-to-change-that.html>
11. Apple, Sep 2024, *Apple Intelligence comes to iPhone, iPad, and Mac starting next month*, <https://www.apple.com/newsroom/2024/09/apple-intelligence-comes-to-iphone-ipad-and-mac-starting-next-month/>
12. Google, March 30, 2024 (accessed), *Get started with Gemini Nano on Android (on-device)*, https://ai.google.dev/tutorials/android_aicore
13. Chris Velazco, February 21, 2024, *Phones are getting packed with AI features. But how helpful are they?* <https://www.washingtonpost.com/technology/2024/02/21/ai-phones-google-samsung-iphone/>
14. Sandeep Budki, March 20, 2024, *Samsung Galaxy S24 Ultra Review: Committed and Spicing up Relationship with Customers*, <https://www.themobileindian.com/reviews/samsung-galaxy-s24-ultra-review-committed-and-spicing-up-relationship-with-customers>
15. Arjun Kharpal, July 25, 2024, *Samsung hints at new products as it bets on AI to drive upgrades to its latest foldable phones*, <https://www.cnbc.com/2024/07/26/samsung-tm-roh-interview-galaxy-ai-mixed-reality-and-foldables.html>

16. Allison Johnson, Aug 1, 2024, *A first look at Apple Intelligence and its (slightly) smarter Siri*, The Verge, <https://www.theverge.com/2024/7/31/24209910/apple-intelligence-ios-18-preview-siri>
17. Kif Leswing, Aug 14 2024, *Google's live demo of Gemini ramps up pressure on Apple as AI reaches smartphone users*, <https://www.cNBC.com/2024/08/14/google-live-gemini-demo-lifts-pressure-on-apple-as-ai-hits-smartphones.html>

20. Tool Usage

Tool Usage in Generative AI

We're talking about tools *for* the AI engine, not tools that developers can use to create models and engines. LLMs require tools to do more advanced things, just like humans. Some of the things that are hard for an LLM to do without tools include:

- Having real-time or up-to-date information (e.g., stock prices or the latest AI research papers).
- Computation-related questions beyond basic arithmetic.
- Information that's only in a different place (e.g., the company's internal ERP database).
- Time-specific or locale-specific information that differs from its training.

Another type of tool that LLMs can use are those that perform an action for you, such as sending an email. However, we'll mostly examine those in the chapter on agentic architectures.

As another simple example, if someone asks you the time, you look at your watch (or your phone). If you ask an LLM "What is the time?" there is nothing in its training data set that could possibly answer this correctly. The only way is to use a clock that's integrated into the LLM, and executed by the AI Engine as part of answering your query. In this case, the clock is a "tool" for the LLM.

The terminology for tool integrations used by LLMs is still evolving, as indeed are the tool capabilities themselves. Some other words you might hear include:

- Plug-ins — refers to the data source integrations in the ChatGPT version.
- Function calls — because that's what the locally integrated tools are.

Tools are a very advanced part of an AI architecture. You don't want to worry about them in your first rodeo. But if you want to extend the capabilities of LLMs beyond whatever was in their training data set, then tools are the way to achieve this.

Types of Tools

The AI engine can use several types of tools:

- Data tools
- Action tools
- Calculation tools

Data tools are ways that new data is integrated into the AI model without extra fine-tuning. The RAG architecture is the typical way that these tools are integrated. A “retriever” component looks up relevant data and then the AI engine converts that into an answer. This is how a lot of business chatbots learn the data about their entire website, and indeed a similar architecture is probably used in the AI-enabled search available with Bing or Google.

Action tools are ways that the AI engine can change something, rather than just output some text. For example, it could integrate with your internal financials app so as to post an employee expense report (or if the integration is two-way, then it could also access data from the financials app, which is like it’s a data tool, rather than an action tool).

One way to think about this is in terms of “read-only” versus “read-write” tool interfaces. A data tool is mainly a read-only tool, whereas an action tool can also write data.

Calculation tools are computational apps that the AI model can call to generate results. For example, if it detects a mathematical computation, it might launch that tool to answer the math question. Models have to be trained on how to launch tools, and when to launch particular tools. Some examples of tools include:

- Math tools (e.g., calculators, converters)
- Clocks and calendars (e.g., for time/date computations)
- Wordsmithing tools (e.g., wordcount)

Tool Architectures

LLMs with tool capabilities used to be called Tool-Augmented Language Models (TALMs), but this terminology has largely been dropped. Tool usage is table stakes for LLMs these days, and is just one part of its training regimen, along with high-protein egg nog. LLMs are trained about tool generically in regard to issues like:

- Deciding whether to use a tool or not for a user query.
- Choosing which tool to use for a given query.
- Extracting the parameters from the user query to send to the tool.
- The input format required for the tool and its “function call” request (usually JSON or a Python script).
- The output format returned from the tool (e.g., JSON or HTML or plain text).
- How to use the tool results for a final answer.

Tool usage is intended to be hidden from the user. The normal user interaction won’t see the tool usage happening, because the LLM does everything in its own sandbox, before presenting the final results as a summary. The normal sequence is for the LLM to receive a user query, decide that a tool call is needed, wait for the tool’s results, and then summarize the tool output back into nice text. However, developers who are building LLM-based applications can view traces of tool function calls and their results.

One major limitation is that LLM tool usage is *stateless*. The LLM does not usually learn from, or even remember, the results it received from one of its tool interactions. The tool interfaces might support caching, which is a speedup, but this won’t help the LLM to make better use of tools, or to learn anything from the output results it sees from a tool.

How are Tools Integrated?

Like humans, an AI needs to learn to look at its watch if someone asks the time. Specific training data sets are required that tell the AI what tool to use, and when.

In the early days of LLMs, only a small number of internal tools were used. However, modern LLMs now often the ability to submit an interface specification for a new tool, usually via a JSON configuration file. This means that you can create and add new tools for business-specific purposes without needing to do any fine-tuning of these powerful LLMs.

During inference, the AI engine has to recognize in the LLM output that a tool must be executed. Not all queries require tools, and hence the LLM output results may or may not indicate tool usage. There are a variety of ways to do this:

- Tool-specific tokens — i.e., the LLM can emit a “trigger” token to run a tool. Note that PEFT could be used here to fine-tune new tool capabilities, by only adding a few new tool-triggering tokens to the vocabulary.
- Placeholder patterns — i.e., output something like an “--insert current time here--” special pattern is another way, and the engine then looks for these patterns, which avoids adding tool tokens to the vocabulary, but is inefficient in that there are multiple text tokens in the output).
- JSON data processing — this is the method used by OpenAI’s API, whereby some models have been trained to return JSON-formatted text that indicates a function call. In this case, the client must call the tool, rather than the OpenAI server. The models can also call tools themselves automatically on the server side, but this is less under the control over the client. The client OpenAI API supports “tools” and “tool_choice” parameters, which give some control over the tool launching processes.
- Code generation — there are various AI models that will generate code, such as in Python, that can be executed to generate the answer. This is a general solution, because Python can call various submodules and can thereby generate many tools.
- Multi-level planning — the AI first generates a plan of how to answer the query, including what tools to use, and then runs any tools, and then does another inference query to collate it into a final answer.

Tools can be valuable to any type of LLM. They can enhance the output produced by any AI application, but where they really hit their stride is with agents, by including tool capabilities in “agentic architectures” controlled by multiple agents.

Tool Usage in RAG Architectures

Tool usage is an issue in RAG architectures just as much as other LLM systems. Not all queries can be handled by LLMs even with a RAG datastore. Any dynamic queries that cannot be answered by a chunk of a stored document may need either:

- (a) General knowledge from the model itself, or
- (b) Tool execution, or
- (c) Both of these.

Yet another case is where RAG chunks are required, but none are found. The model needs to be trained to emit a failure message, or a different prompt is required for the zero-chunks case: “Sorry, I do not know the answer to that question.”

Tool integration is a general LLM issue that is not specific to RAG architectures. Examples of tools include clocks, timers, calendars, calculators, and many more. Having an LLM launch tools means training it to know:

- (a) Which queries need tools and/or RAG chunks.
- (b) What tools are needed (if any).
- (c) Whether the tools need to run on the RAG chunk as input or if they don’t require one.
- (d) How to launch the tool (with input parameters).
- (e) How to integrate the tool output results into the LLM’s answer.

Hence, an advanced LLM needs to decide on a number of issues at the same time: whether or not it needs to launch a tool, whether or not it needs to get an external RAG chunk, not to mention whether or not a user’s question is safe to answer (i.e., refusal and prompt shields). All of that in less than 200ms. That’s quite a lot to ask of the poor silicon elf.

LLM Computer Usage

Computers can be a useful tool! Also, you know, your phone is a computer and it’s a lot more powerful than the one they had on Apollo 13. Could an LLM use your computer?

This idea of having the LLM use your phone or your computer is the latest hot area of tool usage and agentic architectures. Research has shown that LLMs can access the screen of your device in two ways:

- (a) Screenshot analysis (image models), or
- (b) Hierarchical internal views.

Either of these methods is possible with existing multimodal LLMs. Arguably, the view method might even only need to use text analysis if it represents the logical layout of images (e.g., logical windows and vector image formats).

As I write this, there are two major industry launches getting attention for putting this type of functionality live:

- Apple Intelligence — training on-device LLMs to view your screen and manipulate your apps.
- Anthropic — generalized LLM computer usage via mouse and keyboard.

Apple Intelligence involves two separate facets. First, the on-device LLMs can learn about what's on your screen, so it can respond in-context depending on what app you're currently using (e.g., texting versus searching the web). Secondly, Apple is rushing to have all its own apps, and those from third parties, add LLM integrations called "intents" to their apps, so that the AI phone can do actions using those apps.

Anthropic has trained a model with a similar goal, but it's focused on full computer usage without any modifications to apps. Not only can it view the screen, but it can take control of the mouse and keyboard. Theoretically, the LLM can then do anything that a human could do with the GUI (for better or worse), using the apps as they already exist today.

References

1. Jerry Huang, Prasanna Parthasarathi, Mehdi Rezagholizadeh, Sarath Chandar, 14 Apr 2024, *Towards Practical Tool Usage for Continually Learning LLMs*, <https://arxiv.org/abs/2404.09339>
2. Amy Marks, Jun 11, 2024, *Clarifying Function Calling / Tool Use in LLMs*, <https://medium.com/@aevalone/clarifying-function-calling-tool-use-in-llms-6511af510f99>
3. Yicheng Fu, Raviteja Anantha, Prabal Vashisht, Jianpeng Cheng, Etai Littwin, 6 Sep 2024, *UI-JEPA: Towards Active Perception of User Intent through Onscreen User Activity*, <https://www.arxiv.org/abs/2409.04081>
4. V Adrakatti, 2024, *Exploring screen summarization with large language and multimodal models*, Masters Thesis, University of Illinois Urbana-Champaign, Urbana, Illinois, USA, <https://www.ideals.illinois.edu/items/131510>
5. Anthropic, 23 Oct 2024, *Developing a computer use model*, <https://www.anthropic.com/news/developing-computer-use>
6. Anthropic, 23 Oct 2024, *Introducing computer use, a new Claude 3.5 Sonnet, and Claude 3.5 Haiku*, <https://www.anthropic.com/news/3-5-models-and-computer-use>
7. Anirban Ghoshal, 23 Oct 2024, *How Anthropic's new 'computer use' ability could further AI automation*, <https://www.cio.com/article/3583260/how-anthropics-new-computer-use-ability-could-further-ai-automation.html>

21. Agentic Architectures

Single Agent Architectures

The idea of agents tends to evoke the idea of AI engines that take over the planet. After all, agents are the AI apps that “do” things, whereas LLMs are supposed to just sit there and write poetry.

Actually, agents are just software, and aren’t necessarily an architecture you need to shy away from. These architectures are already well established in various industry frameworks. The features that are needed include:

- Data sources (e.g., read your email inbox, search the internet).
- Software integrations (e.g., your company database, HR system, internal financials, etc.)
- LLM queries (the basic level of intelligence).
- Combining it together (i.e., planning, scheduling, delegating, etc.)

Although single agents have the potential to be powerful, it’s becoming clear that the main industry direction is “agentic architectures” and the use of multiple agents with specially trained LLMs for particular tasks. For example, Salesforce has multiple agents for different activities, and Apple Intelligence has a “multi-LoRA” architecture with many small on-phone LLMs, backed up by a larger LLM framework in the cloud called Private Cloud Compute (PCC).

Types of Agents

There are, in fact, several different types of agents, each with their requisite levels of difficulty. There’s not really any widely accepted categorization of agents, but rather than be discouraged, I’ll invent my own:

- Read-only agents (report agents)
- Read-write agents (action agents)

And we can further sub-categorize based on how the agent gets kicked off:

- Manual agents
- Scheduled agents
- Triggered agents

And another dimension is whether the agent needs approval for its “write” actions:

- Supervised agents (human-in-the-loop approval needed)
- Unattended agents (automated or “autonomous agents”)

So, that’s about 12 distinct species of agents, but they’re not that different. If you’re building an agent, it needs several components:

- Integration to its own LLM
- Data source integration (to “read”)
- Output integration (to “report”)
- Action integration (to “write” or “act”)

This is getting quite tricky, so some examples might help.

- Imagine an agent that integrates with the email subsystem, and launches whenever it received an incoming email, and then integrates with outgoing email API, so as to let you set up an automatic reply to people sending you emails, so that you can tell them you’re on vacation.
- An agent software runs in the background and is only triggered when a text comes into your phone, and the agent then accesses the speaker integration, an “action,” so that the phone goes: *Ping!*
- Another type of manually-launched agent could, once it receives your query request, scour the internet and return you the top ten blue links about that topic.
- Imagine the safety improvement of having a fully autonomous software agent that watches the wheels in your car (“read”), detecting the signs of slippage, and automatically turns your brakes on and off in quick succession (an “action”), so as to re-gain traction, without the driver doing anything.

Oh, wait! Those are things we’ve had for years. We’ve had AI agents all around us, and never knew it?

Report Agents

Report agents are “read-only” agents that only create output such as a report after some analysis. The agent finds some data that you want, and creates a report on it. Example use cases would include:

- Email inbox summary
- News headline summarization
- Company stock online research
- Research paper literature reviews

Let’s say we want an AI engine that reads the news headlines in the morning, and shows you a summary. In order to implement this idea, the report agent needs:

1. Scheduler (to wake it up in the early hours).
2. Data source integrations (to download the news headlines from somewhere).
3. LLM query interface (to send the news headlines to the LLM to summarize).
4. Display the text summary as its “report”

Note that, without the scheduler, this is effectively a RAG architecture with an integrated data source. If we had to launch this report manually, it’s not really an agent.

Action Agents

Action agents are general agents that can do something that changes the outside world, with an “action” or “read-write” capability. Some example use cases include:

- Sending an email or text on our behalf
- Trading your own stocks using an automated algorithm
- Booking flights or an entire vacation
- Completing and filing a tax return (yeah, right!)

Imagine if we extended our news headlines report agent so that it not only summarized the news (i.e., “read”), but also then emailed us a report every morning.

This adds one more step, which is the “action” or the “write” operation of sending the email. This needs one more component, which is an integration with the email service, so that the agent can send out an email.

In this case, the agent is actually running “unattended” because the scheduler wakes it up, and then it sends an email without needing human approval. Other types of AI agents could be programmed to require a human to approve the actions.

Another variation would be a “triggered agent” where some event starts the agent running, rather than a scheduler. For example, an incoming email might trigger some type of email-responding agent, or a summary notification to pop up on your phone.

Agentic Architectures

The term “agentic architectures” is a hot area that is all the rage at the moment on Arxiv. It’s a somewhat vague concept, but generally encompasses the use of LLM agent technologies with these important aspects:

- Multiple agents
- Cooperation of multiple types of agents
- Workflow pathways
- Planning
- Scheduling, sequencing, and chaining
- Retrievers (i.e., data source “read” capabilities)
- Actions (“write” capabilities)

In the most basic form, it's very much like RAG, except the agent goes and fetches the data from somewhere, typically from something that's not a database, such as the web.

The good thing is that it requires no work to keep it up-to-date. The bad thing is, it has access to the web, but there are ways to constrain the scope of a web search.

As an example, if you're trying to produce a chatbot that is an expert on "recipes," the agent might be specifically aware of only a few "cooking" websites and may even use the search capabilities of those sites. This is an agentic architecture with "retrievers" and you can see the analogy to RAG retrievers.

You could characterize RAG as an agent that knows how to retrieve data from your vector store or other stores. However, in general agentic architectures are often more diverse, and there are typically multiple agents involved.

As another example, perhaps you want to have a website which generates 5-course meals where each course complements each other. Perhaps also catering to some special needs.

You might end up with a couple of agents that search for recipes, and an agent that has some knowledge about what foods complement each other, and another that knows about wine pairings, and so on. These agents are often arranged in a structure whereby the output of one agent feeds into another agent and the results are further refined.

Each "agent" is also running an LLM query, which focuses the agent to a specific "role" but also has knowledge of the original "question". Once the structure has all been traversed, the result is generated by a final LLM summary.

An agentic architecture can be interactive, and there are often multiple places in the sequence where the user can be involved. For example, deep down, it's possible that there may be a choice between a "beef dish" or a "chicken dish" and it's possible for the LLM to ask a followup question (if trained to do so), whereby the user can indicate which food they prefer.

More complex agentic architectures are not a static structure or single toolchain. Advanced agentic structures can contain loops, decision points, interactivity or approval choices, feedback points, and steps for the user to fulfil.

Many of the software development AI tools that generate entire fully-coded applications are agentic systems. They can have agents that make up a typical software team: there is a PM Agent, a coding agent, an architect agent, a testing agent, a documentation agent, and more.

The user describes the software they want built, in as much detail as possible, and the PM agent will refine the requirements, often in a loop with the user, then each requirement will be “designed” by the architect agent, coded by the coding agent, tested and documented by the respective agents.

Along the way, bugs can occur, so a “debugging agent” can come into the mix.

At the end of a full cycle of auto-development, the user will be brought back in to test the generated application. Behind the scenes there is yet another agent, one which builds the code and deploys the executable. Once the user accepts a requirement as complete, the agentic system loops around to the next requirement.

Overall, the “structure” of an agentic architecture is very dynamic. Each agent can have access to a different LLM. The coding agent might be a code-completing LLM, whereas the “architect agent” might be a coding agent with perhaps a design pattern RAG-based LLM. Agentic architectures can get very expensive fast!

Security of Agents

A common concern with agents on everyone’s computer (or phone) is that they are a security vulnerability. If an AI engine can send emails, then so can the hackers, after they take over your device.

This is true, and security must be reviewed carefully, but it’s important to note that this is nothing new. It’s always been the case that hackers, if they gained control of your device, could send out emails. There’s nothing about AI agent architectures that inherently makes the situation either more or less insecure than it has been in the past.

But what about the agent itself going rogue? Could it without permission start sending out lots of emails?

Technically, it could, yes, but what would cause it to? Why would it be more likely to go rogue than plain old Microsoft Outlook? Maybe it has some potential to do so because we've made automation easier, but it would require a human failure in the software design. And programmers never write any bugs, so that should be reassuring.

References

1. Arun Shankar, Oct 2024, *Designing Cognitive Architectures: Agentic Workflow Patterns from Scratch*, <https://medium.com/google-cloud/designing-cognitive-architectures-agentic-workflow-patterns-from-scratch-63baa74c54bc>
2. Anita Kirkovska, David Vargas, Jul 11, 2024, *Agentic Workflows in 2024: The ultimate guide*, <https://www.vellum.ai/blog/agentic-workflows-emerging-architectures-and-design-patterns>
3. Shuofei Qiao, Runnan Fang, Zhisong Qiu, Xiaobin Wang, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, Huajun Chen, 10 Oct 2024, *Benchmarking Agentic Workflow Generation*, <https://arxiv.org/abs/2410.07869>
4. A. Singh, A. Ehtesham, S. Kumar and T. T. Khoei, 2024, *Enhancing AI Systems with Agentic Workflows Patterns in Large Language Model*, 2024 IEEE World AI IoT Congress (AIIoT), Seattle, WA, USA, 2024, pp. 527-532, doi: 10.1109/AIIoT61789.2024.10578990. <https://ieeexplore.ieee.org/abstract/document/10578990>
5. Chawla, Chhavi; Chatterjee, Siddharth; Gadadinni, Sanketh Siddanna; Verma, Pulkit; Banerjee, Sourav, 2024, *Agentic AI: The building blocks of sophisticated AI business applications*, Journal of AI, Robotics & Workplace Automation, Volume 3 / Number 3 / Summer 2024, pp. 1-15(15), Henry Stewart Publications, DOI: <https://doi.org/10.69554/XEHZ1946> <https://www.ingentaconnect.com/content/hsp/airwa/2024/00000003/00000003/art00001>
6. Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, Chenglin Wu, 14 Oct 2024, *AFlow: Automating Agentic Workflow Generation*, <https://arxiv.org/abs/2410.10762> <https://github.com/geekan/MetaGPT>
7. Ruixuan Xiao, Wentao Ma, Ke Wang, Yuchuan Wu, Junbo Zhao, Haobo Wang, Fei Huang, Yongbin Li, 21 Jun 2024, *FlowBench: Revisiting and Benchmarking Workflow-Guided Planning for LLM-based Agents*, <https://arxiv.org/abs/2406.14884>
8. Dawei Gao, Zitao Li, Xuchen Pan, Weirui Kuang, Zhijian Ma, Bingchen Qian, Fei Wei, Wenhao Zhang, Yuexiang Xie, Daoyuan Chen, Liuyi Yao, Hongyi Peng, Zeyu Zhang, Lin Zhu, Chen Cheng, Hongzhu Shi, Yaliang Li, Bolin Ding, Jingren Zhou, 20 May 2024 (v2), *AgentScope: A Flexible yet Robust Multi-Agent Platform*, <https://arxiv.org/abs/2402.14034> <https://github.com/modelscope/agentscope>

9. Bryson Masse, October 31, 2024, *Microsoft's agentic AI tool OmniParser rockets up the open source charts*, <https://venturebeat.com/ai/microsofts-agentic-ai-tool-omniparser-rockets-up-the-open-source-charts/>

22. AI Research Overview

The Three S's of AI Research

There are three main types of AI research, which I've taken to calling "The Three S's" when I categorize them. The areas are:

- Smartness
- Speed
- Safety

I'm really a fan of speed. That's our main area of research at Aussie AI, with expertise and a few patents filed in low-level kernel optimizations, on-device inference, and accelerator add-on components. So, this might be a long chapter if I let myself go.

Oh, wait! We've already written a whole book on AI speedups, which is titled *Generative AI in C++: Coding Transformers and LLMs*. It's all about how to code up Transformer internals, and the many types of kernel optimizations to use for the components (e.g., KV caching, kernel fusion, memory efficiency, etc.). But that book was written in March 2024, and there are about five new types of KV caching in the research papers, so there are some parts that need updating. I'll try to be brief.

Smartness Research

The overall goal of AI researchers is, you know, artificial intelligence. Since we've got the "artificial" part well covered, there's a ton of research on "intelligence" and I call it "smartness" research, just to fit in with my alliterative fun. There are many subareas of smartness research, such as:

- Artificial General Intelligence (AGI)
- Artificial Super-Intelligence (ASI)
- Use cases
- Prompt engineering
- Reasoning
- Mathematics

The above research is mostly about making super-smart AI models, no matter what the cost in electrons, and reducing this expense is delegated to other AI researchers. Some of the hotter areas in AI “smartness” research include:

- Trillion-parameter models
- Mixture-of-experts (MoE) and other “ensemble” architectures
- Multimodal “omni” models
- Multi-step reasoning algorithms (e.g., Chain-of-Thought)
- Small Language Model (SLM) capability improvements
- Multi-agent architectures (“agentic architectures”)

There’s lots of research happening, some of which appears in papers, and the rest is hidden away behind swinging doors. The capabilities of models are astounding and increasing, but we’re not that close to AGI yet.

Multi-step reasoning. The hottest area at the moment is the use of multiple inference steps to improve overall reasoning of an LLM. This has received a surge of interest since OpenAI released its “o1” model, which was code-named “strawberry” in reference to a well-known reasoning problem: LLMs could not correctly count the number of the letter “r” in “strawberry” (they would say two rather than three). This model used the “Chain-of-Thought” (CoT) method of multiple steps of inference to improve its results.

This is a very fundamental change of focus. Until this point, the main way to make an LLM smarter was to give it more parameters, and better training data. This was based on the “scaling laws” that AI would be smarter if you scaled the parameters. However, this has been somewhat superseded by the “inference scaling laws” which says that the LLM can be smarter if you give it more time to run better inference analysis in multiple steps.

There are many subtypes of this multi-step inference approach to reasoning. Chain-of-Thought is obviously getting the most attention because of its use by OpenAI. Hence, the subfields of AI reasoning research areas includes:

- Chain-of-Thought (CoT)
- Self-reflection
- LLM as Judge
- Tree of Thoughts (ToT)
- Best-of-N (BoN)
- Skeleton of Thoughts (SoT)
- Graph of Thoughts (GoT)
- Agentic architectures

There is also a lot of crossover between these ideas and prompt engineering. The two areas are mostly orthogonal, so there is a combinatorial explosion of options when you consider that all of the above multi-step algorithms can also use different prompting optimizations at every step.

On the other hand, some recent research put out by Apple has cast doubts on whether LLMs are doing any type of reasoning at all. Their paper asserts that most of the results of LLMs are due to memorization and pattern matching, rather than using any generalized reasoning analysis.

Personally, I think it's another case of the "bitter lesson" whereby human researchers always think that advancements must come by coding up human-like reasoning algorithms, but the best solution for computers is often simply brute-force computations.

I guess time will tell who's right!

References on Reasoning

Below are a number of relevant research papers, and the full list is available at <https://www.aussieai.com/research/reasoning>.

1. Jason Wei and Denny Zhou, May 11, 2022, *Language Models Perform Reasoning via Chain of Thought*, <https://research.google/blog/language-models-perform-reasoning-via-chain-of-thought/>
2. Cameron R. Wolfe, Jul 24, 2023, *Chain of Thought Prompting for LLMs: A practical and simple approach for "reasoning" with LLMs*, <https://towardsdatascience.com/chain-of-thought-prompting-for-llms-33c963eead38>
3. Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, Karthik Narasimhan, 3 Dec 2023 (v2), *Tree of Thoughts: Deliberate Problem Solving with Large Language Models*, <https://arxiv.org/abs/2305.10601> Code: <https://github.com/princeton-nlp/tree-of-thought-llm>
4. Cameron R. Wolfe, Aug 21, 2023, *Tree of Thoughts Prompting. Solving multi-step problems with LLMs via deliberate planning and exploration*, <https://cameronwolfe.substack.com/p/tree-of-thoughts-prompting>
5. M.Besta, N. Blach, A. Kubicek, R. Gerstenberger, M. Podstawski, L. Gianinazzi, J. Gajda, T. Lehmann, H. Niewiadomski, P. Nyczek et al., 2024, *Graph of thoughts: Solving elaborate problems with large language models*, Proceedings of the AAAI Conference on Artificial Intelligence, vol. 38, no. 16, pp. 17682–17690. <https://arxiv.org/abs/2308.09687>
6. Cameron R. Wolfe, Jan 3, 2024, *Graph-Based Prompting and Reasoning with Language Models. Understanding graph of thoughts prompting and several*

variants, <https://towardsdatascience.com/graph-based-prompting-and-reasoning-with-language-models-d6acbcd6b3d8>

7. Xuefei Ning, Zinan Lin, November 17, 2023 *Skeleton-of-Thought: Parallel decoding speeds up and improves LLM output*, Microsoft Research Blog, <https://www.microsoft.com/en-us/research/blog/skeleton-of-thought-parallel-decoding-speeds-up-and-improves-llm-output/> Code: <https://github.com/imagination-research/sot/>
8. Cogni Down Under, Sep 2024, *Reflection 70B: The AI That Thinks Before It Speaks*, <https://medium.com/@cognidownunder/reflection-70b-the-ai-that-thinks-before-it-speaks-8a70d3a0e38a>
9. Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, Aman Chadha, 5 Feb 2024, *A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications*, <https://arxiv.org/abs/2402.07927>
10. Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei Zaharia, James Zou, 4 Jun 2024 (v2), *Are More LLM Calls All You Need? Towards Scaling Laws of Compound Inference Systems*, <https://arxiv.org/abs/2403.02419>
11. Zehui Chen, Kuikun Liu, Qiuchen Wang, Jiangning Liu, Wenwei Zhang, Kai Chen, Feng Zhao, 29 Jul 2024, *MindSearch: Mimicking Human Minds Elicits Deep AI Searcher*, <https://arxiv.org/abs/2407.20183> Code: <https://github.com/InternLM/MindSearch> Project: <https://mindsearch.netlify.app>
12. Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, Azalia Mirhoseini, 31 Jul 2024, *Large Language Monkeys: Scaling Inference Compute with Repeated Sampling*, <https://arxiv.org/abs/2407.21787> (Generating multiple answers by repeated inference queries, and then using a verifier to choose the best one, which is shown to greatly increase overall accuracy.)
13. Justin Chih-Yao Chen, Archiki Prasad, Swarnadeep Saha, Elias Stengel-Eskin, Mohit Bansal, 18 Sep 2024, *MAGICoRe: Multi-Agent, Iterative, Coarse-to-Fine Refinement for Reasoning*, <https://arxiv.org/abs/2409.12147> <https://github.com/dinobby/MAGICoRe>
14. Xiaohan Xu, Chongyang Tao, Tao Shen, Can Xu, Hongbo Xu, Guodong Long, Jian-guang Lou, 29 Feb 2024 (v2), *Re-Reading Improves Reasoning in Large Language Models*, <https://arxiv.org/abs/2309.06275>

Safety Research

There's a lot of research about making AIs safer. Well, actually, there's a lot more research on smartness and speed than there is on safety (about 2% of all AI papers), but with 250,000 research papers published per year on AI, there's still plenty of safety papers to fill your weekends.

We've covered a whole host of safety topics in Chapter 10, and every bullet point in that chapter is a whole research field in itself.

Some of the main research areas are:

- Hallucinations
- Bias
- Fairness
- Explainability
- Adversarial attacks

Apple Intelligence. In addition to speedup, safety has also been a priority factor in training of the Apple AI models for both on-device and cloud-based inference. Here are some of the approaches they used:

- Data preparation (of training data)
- Filtering profanity
- Filtering personal details — e.g., credit card numbers.
- Refusal-specific training
- Model evaluation on safety benchmarks

I must admit not being fully versed with AI safety research. I’m more focused on speed, especially for advanced software kernels and on-device inference.

Speed Research

AI research tends to use the word “performance” to mean “intelligence” or “smartness” in the vernacular. Hence, you have to look for more specific words like “efficiency” or “latency” or “throughput” to find all the speed papers. Even the word “optimization” can mean optimizing the performance, but can actually appear in the titles of both types of papers.

The basic categorization of speed research papers goes like this:

- Hardware acceleration
- Software acceleration
- Some combination thereof

Personally, as a software engineer, I tend to skim over all the hardware papers, because they’re in the “too hard basket” for me. But hardware systems and silicon chips is the area where the greatest speed advances have been made in the past, and this continues to be the case into the foreseeable future.

The best software acceleration algorithms for AI engines and models tend to be in the range of ten-fold improvement in speed, whereas hardware speedup is hundred-fold and above. And there are plenty of software papers that give improvements of ten or twenty percent, which is obviously still valuable when you consider the cost of running AI platforms, but it's not the kind of revolution possible with hardware.

SOTA Speed Research

What is the state-of-the-art for speedup optimizations in AI? Mainly I'm going to focus on inference, although there are speedups for training and fine-tuning as well. There are two main areas:

- Data center inference (i.e., lots of GPUs)
- On-device inference (GPU-free phones and PCs)

In terms of data center optimizations, the main focus is all that parallelization capacity available from multiple GPUs. Some of the newer multi-GPU optimizations include:

- Multi-GPU scheduling with preemption
- Serving optimizations
- Batching optimizations
- Disaggregating prefill and decoding phases
- Offloading

There are several software improvements that can be used in both on-device and data center inference. Memory reductions or using fewer computations are techniques for any inference platform. Some software optimizations that have garnered traction in both open-source and commercial platforms include:

- 4-bit quantization (of weights, activations, and/or KV cache)
- Grouped-Query Attention (GQA) — beyond Multi-Query Attention
- KV caching in autoregressive decoding (this is basic table stakes now)
- Flash attention
- Flash decoding
- Paged attention
- Paged Flash attention (combined)
- Prefix KV caching (session-based or generic)
- Chunked prefills
- Multi-LoRA
- Continuous batching

There are several software optimizations that are specific to parallelization, and thus more beneficial for GPU-based data centers (although NPUs are increasingly making this comment incorrect!). Examples include:

- Kernel optimizations
- Speculative decoding
- Prompt lookup decoding
- Distributed tensor parallelism

Commercial AI Platform Speedups

But what are the big companies using? Well, it's hard to say because the big US companies have “gone green” and aren't putting out many research papers. Most of the best papers now are coming out of China. Maybe I should look in the US companies' patent filings, but there's a big lag time between filing them and their public availability. Nevertheless, here are some examples.

Character.AI platform: This is the company that does AI companions online, co-founded by prominent AI researcher Noam Shazeer, so it should be using some advanced stuff. They recently put out a research blog article describing their data center platform, which is obviously GPU-based. Apparently, they're doing 20,000 queries per second, which is astounding. The techniques that they mentioned included:

- GPUs
- INT8 inference quantization (for weights, attention, and KV cache data)
- INT8 quantized training
- Multi-Query Attention (MQA)
- Hybrid Local/Global Attention — interleaving layers of local attention and global attention.
- Cross-layer KV sharing — a type of “layer fusion” in the KV cache data.
- “Stateful caching” — session-based caching.
- Session-based multi-turn KV caching (prefix KV caching)
- “Sticky sessions” — each network session returns to the same box, so its prefix KV cache data is there.

According to the blog post, their estimate is a 13x cost reduction by using these techniques versus a more naive commercial platform, and a 30-fold efficiency improvement since inception.

Apple Intelligence: At the other end of the spectrum, Apple recently announced details of their on-device capabilities, planned for late 2024 and early 2025. Their main methods include:

- M-series hardware (i.e., NPUs)
- Small language models — a 3B on-device foundation model.
- Multiple LoRA adapters (fine-tuning of small models)

Apple’s strategy here is to use a small-ish model for the foundation model, with a 3B on-device model and a “larger” server-based model for cloud execution. Interestingly, their other main on-device strategy is to use fine-tuned versions of this small on-device model, but doing so via multiple LoRA adapters.

LoRA adapters are a way to do fine-tuning by adding a small number of extra parameter, but leaving the main foundational model’s parameter “frozen” (unchanged during fine-tuning). This is a type of Parameter-Efficient Fine-Tuning (PEFT).

LoRA adapters are advantageous in a few ways for on-device inference. Apple mentions using different LoRA adapters per use case, and also a size in the “tens of millions of parameters.” Hence, the LoRA adapters are much smaller than the 3B model, making them easier to switch in and out of memory, while still offering more specialized models for different activities. Apple calls this “on-the-fly specialization” of the foundation model.

This LoRA approach is better than having multiple fine-tuned versions of the 3B foundation model, and trying to swap gigabytes of weights in and out of memory. Instead, just load up the 3B model once, and leave it in the memory permanently, while swapping the much smaller adapters. This may also simplify the process of providing software updates or fixes to the models over the internet.

The details of the LoRA adapters are also included somewhat. These are applied to attention matrices, the attention projection matrix, and the fully connected layers. It is a little unclear since the document mentions 2-bit to 4-bit configurations, but also 16-bit representations of LoRA weights.

The size of the models is the main factor for faster on-device inference, but some additional software acceleration techniques are also used. For inference speedup, in addition to LoRA adapters, these techniques are mentioned:

- Grouped-query attention (memory-efficient)
- Shared embedding/unembedding tensors
- Smaller on-device vocabulary (49k local versus 100k for server-based)
- LoRA adapters (see above)
- Activation quantization
- Embedding quantization
- Efficient KV cache update method for neural engines (details undisclosed!)

Apple has not disclosed the details of its KV cache methods as yet. I wonder whether it is based on session-based prefix KV caching, which would make sense for on-device inference, since every device effectively has only one session.

The only downside to all of this is that most of the early iPhone versions won't have the hardware to run these features. For iPhone 15 and beyond, Apple reports latency measurements of 0.6ms per prompt token (i.e., prefill) and 30 tokens-per-second of decoding.

Training efficiency. Although not directly relevant to users, Apple has also detailed some of its training improvements in efficiency and safety. For its training capabilities, Apple mentions:

- TPUs and on-premise GPUs
- Data parallelism
- Tensor parallelism
- Pipeline parallelism
- Fully Sharded Data Parallel (FSDP)

Model Compression

Model compression is where you make the LLM smaller. Using a smaller LLM is a simple way to go faster because it reduces both memory usage and the total number of computations needed to run inference. There are several types of “model compression” you can consider:

- Small Language Models (SLMs)
- Quantization
- Pruning

Quantization is commonplace, and only the most stringent requirements for accuracy would have you eschew it. Typically, quantization uses sizes such as 16-bit integer or floating point (INT16 or FP16), 8-bit integers (INT8), and even 4-bit integer quantization (INT4) is commonly used as a good speed-versus accuracy trade-off. Note that 32-bit floating point is the basic non-quantized size, so 16-bit is half-size, 8-bit is a quarter, and 4-bit is an eighth. For more about quantization research, see <https://www.aussieai.com/research/quantization>.

Pruning is where you discard small weights by making them zero. You can do “unstructured pruning” where small weights are zeroed no matter where they are. This is related to “sparsity” where there are mostly zeros, but they are not quite the same thing.

Alternatively, you can do “structured pruning” where you cut out only big structures. There are literally four dimensions on which you can prune structures, which is usually done dynamically:

- Depth — layers of the model (e.g., early exit, layer pruning, layer skipping).
- Width — the neurons across a layer (e.g., attention head pruning, filter pruning, channel pruning).
- Length — the input token sequence (e.g., input prompt pruning, token pruning, token merging).
- Internal dimension — embeddings vector pruning methods.

However, pruning is not as commonplace as quantization, and many of the above methods are still mostly in research papers, rather than widespread industry practice. For example, you’ll note that both Character.AI and Apple Intelligence mention quantization, but neither mentions pruning. For more research on model pruning, see <https://www.aussieai.com/research/model-pruning>.

Finally, note that there are other types of model compression, such as knowledge distillation, weight sharing, layer fusion, and weight clustering. For example, Character.AI mentions layer fusion, but it’s a little different because it is in relation to KV cache data rather than weights.

Kernel Optimizations

The low-level code that runs the Transformer inference steps is called a “kernel.” The main types of kernels that have the most optimizations are:

- Matrix multiplications
- Attention
- Decoding algorithms
- KV cache

The main computations in LLMs are matrix multiplications (“MatMul”), which are generalized to “tensor products.” In the early days, a lot of brain power went into making matrices and tensors compute faster. Surprisingly, there are still some breakthroughs happening in MatMul, especially as on-device platforms have different characteristics. For more about MatMul research, see: <https://www.aussieai.com/research/matmul>.

Attention kernel optimizations. Attention is the special algorithm that made Transformers famous. However, it’s also slow, and has now been optimized by dozens of variations, most of which aim to either: (a) reduce total computations by paying less attention to some tokens, or (b) make it more “memory efficient” in its computation pathways. Here’s a list:

- Multi-Head Attention (MHA) — the basic idea from 2017.
- Multi-Query Attention (MQA)
- Group-Query Attention (GQA)
- Local Attention
- Sliding Window Attention
- Linear Attention (other types)
- Flash Attention — there’s also a Flash Attention version 2.
- Paged Attention — the claim to fame of the vLLM open-source platform.
- Paged Flash attention (combination of methods)

For more research on attention optimization, see <https://www.aussieai.com/research/attention>.

Parallel decoding algorithms. The non-autoregressive decoding method is a bottleneck that enforces sequential execution of an LLM, one token at a time.

Various attempts to parallelize this idea have been discovered:

- Speculative decoding
- Retrieval lookup decoding
- Prompt lookup decoding
- Self-speculative decoding
- Tree speculative decoding
- Multi-token prediction
- Aggressive decoding

For more information on speculative decoding and other parallel decoding advances, see <https://www.aussieai.com/research/speculative-decoding>.

KV caching. The KV cache was an innovation at the time, but it's a basic method nowadays. When the Transformer computes the 10th token, it uses computed data from the first 9 tokens, and someone noticed that you could store that “Key-Value” (KV) data in memory, and avoid re-doing some computations for those first 9 tokens. That's the basic “KV cache” method.

Unfortunately, it has become too much of a good thing. If you write a Tolstói novel, about 700,000 words, which is about a million tokens, then you have a KV cache that uses memory for each of those one million tokens. And I mean, a lot per token, with a whole slice of a tensor per token. So, the memory needed for the KV cache actually gets bigger than the whole LLM, and those things aren't small.

The first solution was to limit the length of the “context window” so that you cannot track such long texts. Early models had context lengths of 2048 or 4096, and then gradually improved to 8k and 32k. But that's not very good for understanding a long book, or for processing big images or video, so researchers have developed two types of optimizations: (a) attention optimizations, as above, and (b) KV cache compression.

The idea with “KV cache compression” is to use less memory for the KV cache. Amusingly, almost every method researchers ever tried for model compression for LLM weights can also be applied to compressing the KV cache data — quantization, pruning, and so on. Quantization of the KV cache data is quite commonplace, such as in the Character.AI platform, and this also uses layer fusion of KV cache data (which is similar to depthwise pruning of the KV cache). There are various newer research papers on techniques such as lengthwise per-token KV cache data pruning.

Maybe we could do a KV cache for the KV cache? KV-squared cache. Now there's a patentable idea!

For more research on the various KV caching methods, see:

<https://www.aussieai.com/research/caching>.

An important limitation of the basic KV cache, whether compressed or not, is that it only works within the current query for one user. Optimizing inference across multiple queries from multiple users is where other accelerators come in.

AI Accelerators

The main way to speed up an AI engine is to use better silicon, and find an inference engine that supports the right chipsets. Once you’ve maxed that out, you have to look at software, which means kernel optimizations as already discussed above, and then other add-on accelerator components, which is what this section is about.

The first point is don’t just optimize your LLM and its Transformer, no matter how much fun that is. Any production architecture has other components, such as a basic web server, utility servers, DNS server, identity validation, and so on. You should measure end-to-end response time and optimize the whole system.

In order to further confuse the issue, let us note that a lot of these “accelerators” for LLMs actually plug into the KV cache mechanism. However, the idea is a “global KV cache” that works across multiple queries, rather than the basic KV cache within a single query. But first, let’s pretend we don’t know anything about the letters K and V.

Basic Inference Caching. If you have an AI application with lots of users pounding it with queries, how would you speed it up? Well, one thing to think about is that users often ask the same questions. If it’s your internal HR chatbot, here’s one:

How do I sign up for 401(k)?

Presumably, you’ve got a policy document for that, and you can just return an answer that links to that document. Hence, every person that asks that question can receive the *same answer*. Thus, the idea of an “Inference Cache” is found: put a basic text-to-text cache mechanism between the query input and the LLM. Any answers that get a cache hit are never seen again by the LLM.

Non-Cacheable Queries. Like everything in AI, there are exceptions to this method. Not all queries can be cached. For example, think about this one:

What time is it?

Any query that cannot be answered by just the LLM weights cannot be cached. If the answer requires any external data access or any tool usages (e.g., a clock in this example), then the cache must be voided for that query. Note that LLMs already have mechanisms to detect when a query needs to use external data sources and computation tools (e.g., “trigger tokens” and “function calls”), so you can extend those interfaces to also void the cache for that query whenever they are triggered.

Semantic Cache. Another problem with the basic inference cache is that the tokens must be identical. Slightly different wording of the same question will cause a cache miss. The generalization is therefore to detect queries that have different tokens, but the same meaning, using a semantic cache. This method uses embeddings and a vector database, and works better than a basic text-to-text inference cache. As with the inference cache, you insert it between the input and the LLM. The same restrictions about non-cacheable queries with data sources or tools also apply to the semantic cache.

RAG Accelerators. If you have a RAG architecture, there are other components to speed up. For example, the performance depends on the datastore and retrieval mechanism. Speeding up databases and indexed retrieval methods sounds like something we’ve done before, like for the last 50 years or more.

Also, you can put an inference cache or a semantic cache component in front of a RAG query architecture. This will work provided that the RAG retriever isn’t accessing external data, or doing anything time-dependent. If you add new data to the RAG datastore, you’ll need to clear some or all of the cache.

Note that caching of results for recurring queries is a technique for database optimization, too. In this case, it refers to basic database querying where you cache the chunks of text that a RAG retriever returns, rather than having any unsavory interactions with the KV cache.

But there’s also a global KV cache optimization for RAG architectures. No doubt, you’re pleased to hear that. Instead of storing text chunks, you can store pre-calculated KV cache data. This idea is a variant of prefix global KV caching and fused global KV caching.

High-level accelerators. So, if you've been skimming, here's my summary of all the different types of accelerators mentioned above. Here are the high-level ones that can plug externally to the Transformer's inference engine:

- Inference cache
- Semantic cache
- RAG retriever cache (database cache)
- RAG inference cache
- RAG semantic cache
- Prompt compression (token pruning/token merging)
- Prompt shield (block some queries, more for safety than speed)

Low-level KV cache accelerators. And here's the really fun ones that plug deeply into the KV cache mechanism inside the Transformer kernels:

- Global KV cache (inference cache)
- Semantic global KV cache (semantic cache)
- Prefix global KV cache
- Session global KV cache
- Fused global KV cache
- RAG prefix global KV cache
- RAG fused global KV cache

And for extra coding fun, any of these KV cache methods can be further optimized by the various "KV cache compression" methods:

- KV cache quantization
- KV cache layer fusion (depth dimension)
- KV cache token pruning (length dimension)
- KV cache head fusion (width dimension)

Those ones are my favorites!

References on Inference Optimization

We have catalogued literally over 500 different techniques for speeding up LLM inference, from activation quantization to zero skipping. Some of these techniques are widely-used in industry (e.g., quantization, KV cache compression), whereas others are very obscure (e.g., zero-multiplication models, logarithmic systems).

Here are the links for our inference optimization research literature review:

1. Aussie AI, Nov 2024, *AI Research Overview*, <https://www.aussieai.com/research/overview>.
2. David Spuler, September 2024, *500+ LLM Inference Optimization Techniques*, Aussie AI Blog, <https://www.aussieai.com/blog/llm-inference-optimization>.
3. David Spuler, August 2024, *Hot Inference Optimization Techniques*, Aussie AI Blog, <https://www.aussieai.com/blog/hot-inference-research>.
4. Aussie AI, Nov 2024, *Inference Optimization Techniques*, <https://www.aussieai.com/research/list>.

This is our extensive AI literature survey, focused on inference optimizations, along with other areas such as reasoning algorithms. There are subpages on this site with research paper lists for almost any inference optimization technique you care to examine. If you prefer looking at code, or reading the above research in e-book or print formats, there is our book *Generative AI in C++* published in March 2024.

Green AI

Environmental impact is a concern for AI architectures, because they consume so much GPU juice. The environmental concerns include usage of precious resources:

- Electricity to run GPUs.
- Water to cool them.

In fact, the papers I've read tend to say it costs ten times more per query to run an AI-based query than a regular search using Google or Bing. It may get worse. We're not at the end of this AI ride, obviously, and multi-AI architectures and multi-step reasoning may increase the load. On the other hand, small language models, kernel optimizations, and on-device inference may reduce costs.

There are a lot of “green AI” research papers, with broad analyses about the overarching measurement of environmental impact, metrics to use, etc.; see also:

<https://www.aussieai.com/research/green>.

The point I like the most is that this area has a huge overlap with “speed” research. Faster AI engines means less electricity required, and I'm rather good at making AI run faster, so the work I'm doing is environmentally friendly!

And that's a nice way to finish out this book. Apparently, profiling my vectorized CPU code with `gprof` and tweaking GPU kernels has some good karma.