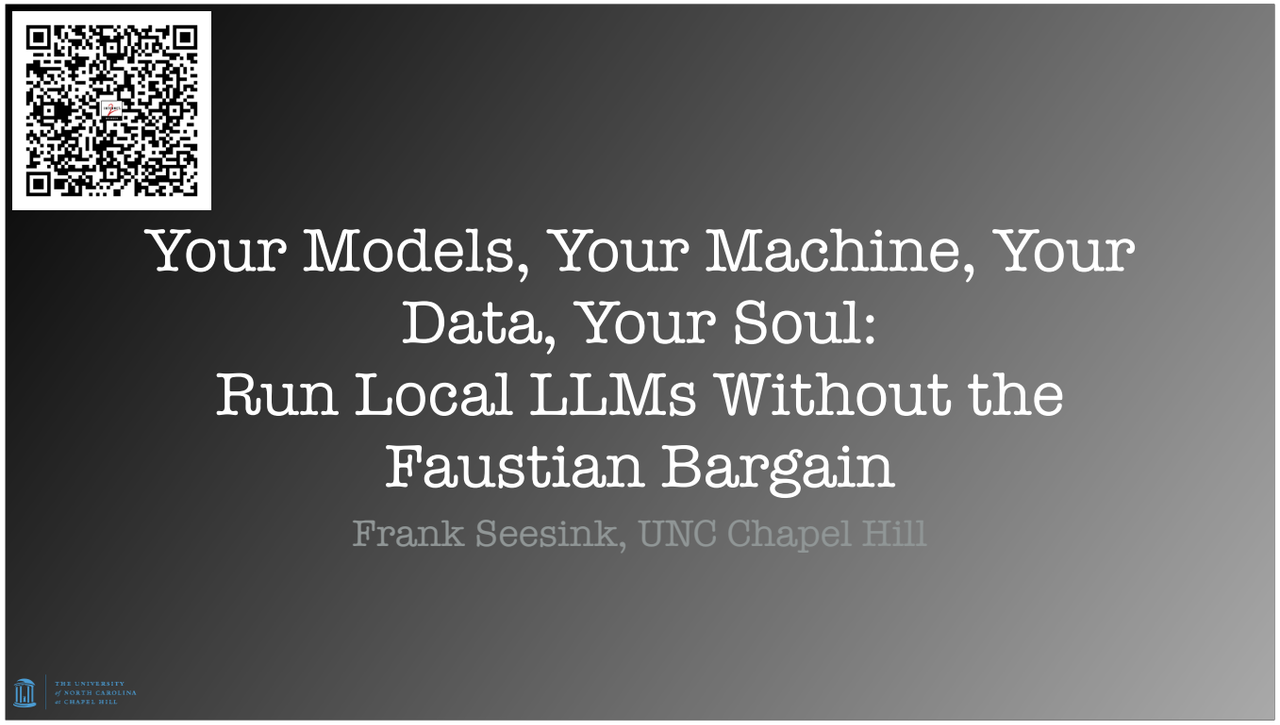


I have 25 minutes to rip through this, and I see that yet again I am standing between you and lunch. So strap in. Here we go…

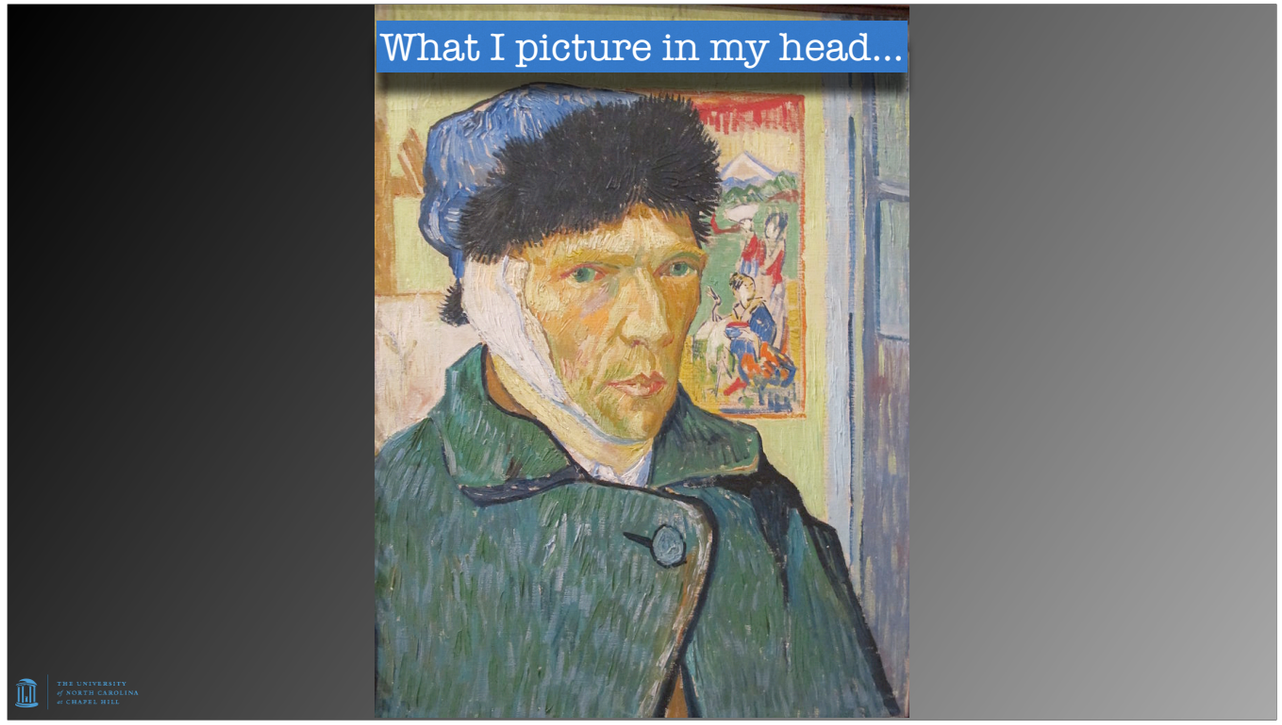
Good morning. My name is…

—

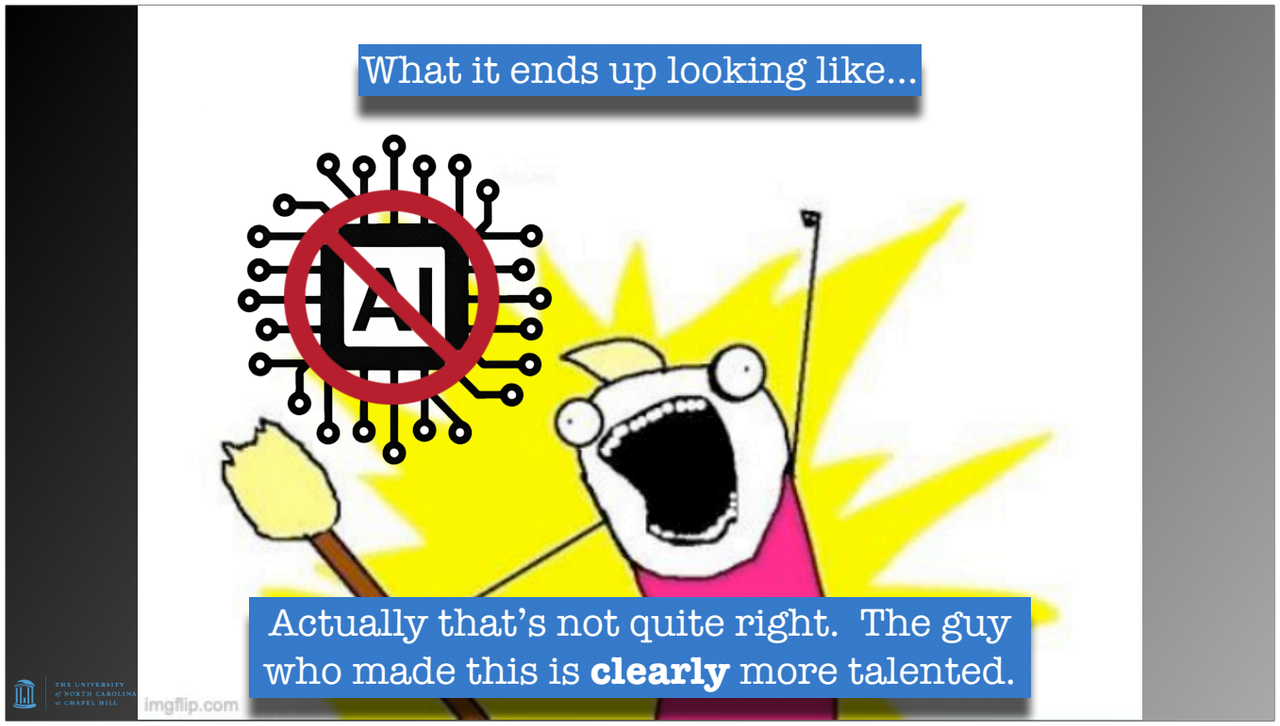
FYI: The term “Faustian Bargain” comes from the German folklore story of Johann Georg Faust, a scholar who becomes unhappy with human limits and trades his soul to an evil spirit (Mephistopheles) for years of worldly power and magic.

First, a message from
our sponsor...

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL



When I work on presentations, this is what I picture in my head.



Unfortunately, THIS is what it typically ends up looking like.

Just managing expectations.

Oh, and in this AI slop world, I need to make it clear. I cannot blame any of this on AI.

Who Am I?



Who am I?

Frank Seesink

- Senior Network Engineer, UNC Chapel Hill
- Part of network DevOps group
- Involved in network automation for years
- Love languages, both human & computer
- Programming since I was 12 years old
- Formally B.S. in Computer Science with all coursework for an M.S. in C.S.
- JOAT - databases, OSes, networking,...

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

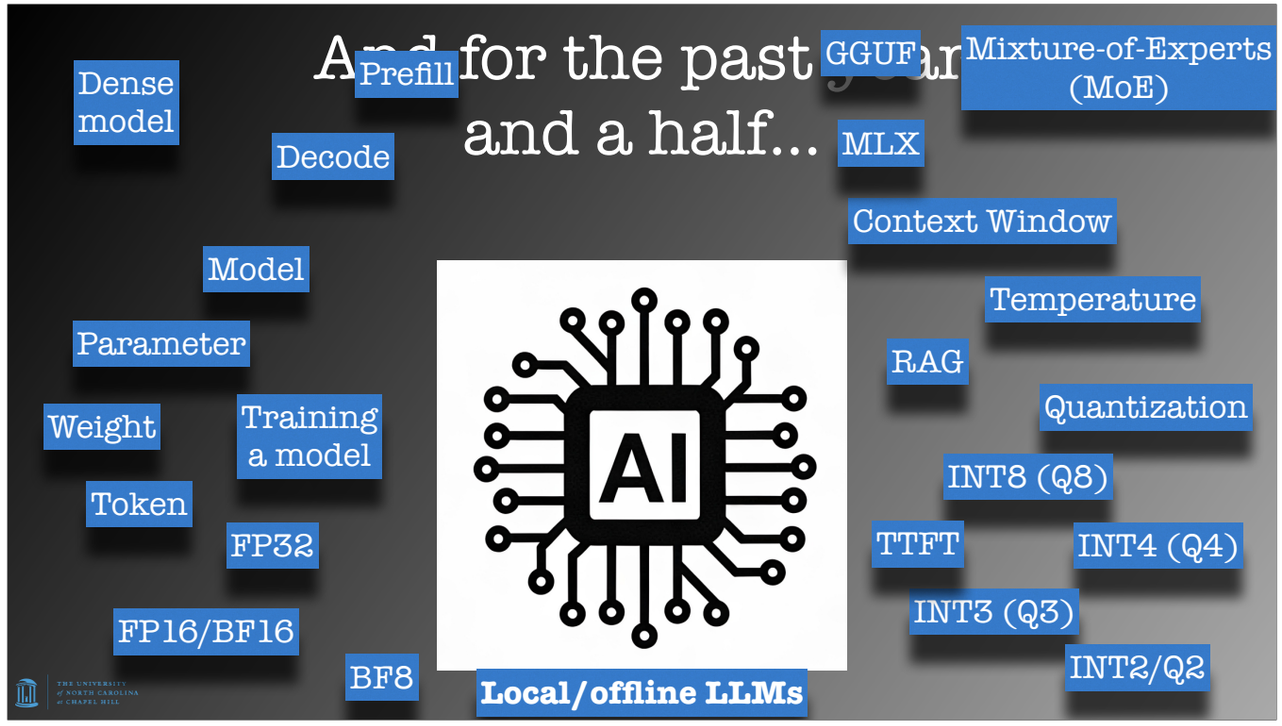
That reminds me.
For anything useful, credit goes to UNC Chapel Hill for allowing me to attend.
For any mistakes/errors/etc., that all falls on me.



—SKIP FOR PRESENTATION—
Brrrraaaaiiiinnnssss.

… No wait, that’s not right. That’s zombies.

Let me try again.

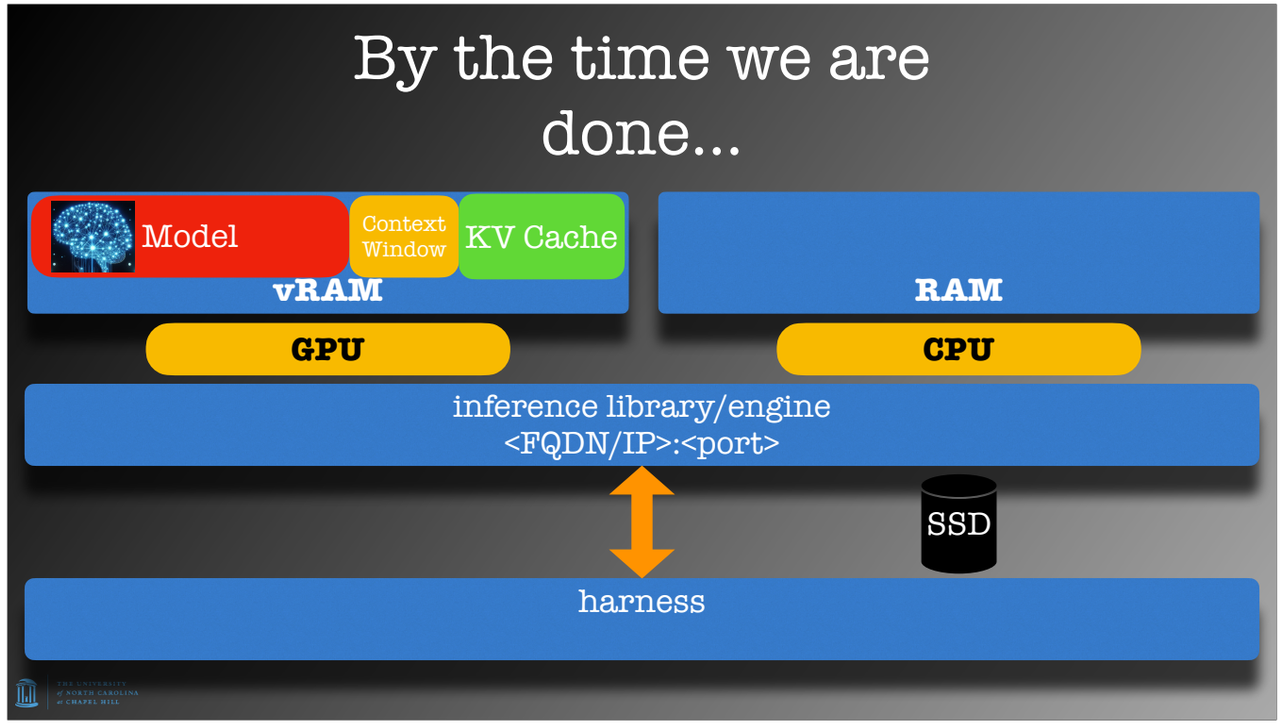


This has required me to learn a whole new vocabulary, as seen here. But by the end of this presentation, you will also know these terms.

I should explain something about myself. I am a bit of a pit bull. By which I mean that when I learn something new or troubleshoot, I tend to sink my teeth in rather deeply and don't let go until I fully wrap my head around what I am learning and understand it, or I have solved whatever issue I am tackling. Instead of just learning one tool in a given area, I tend to research the ecosystem of tools that do similar things.

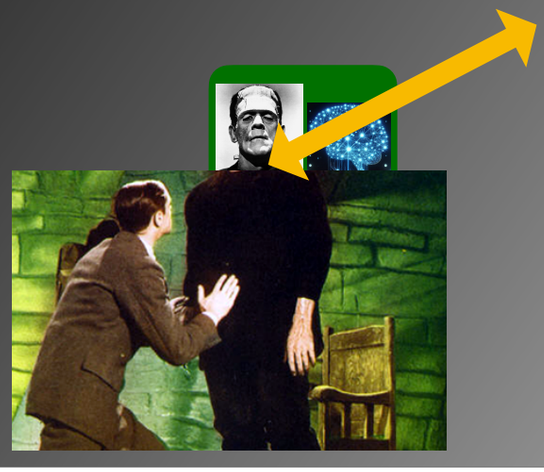
My end goal here is always the same: Dig into something until everything coalesces in my brain, where I can then take all that time and effort, chunk it down, organize it, and hopefully turn around and provide what I have learned in a condensed, structured manner that saves others from having to fumble around like I did. Basically my goal is to save others time.

With regard to AI/LLMs, this means I did not simply go find one program to run LLMs, for example. Of course I started with one, but then I went researching as many others as I could find. More on this in a minute.



By the time we are done, this diagram will make perfect sense.

And you will hopefully see LLMs...

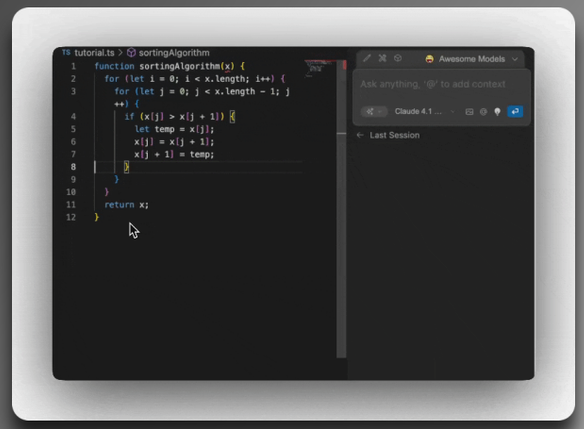


THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

And you will hopefully see LLMs simply as your own personal monster… err… assistant?… where the brain/head might be in one place and the body in another.

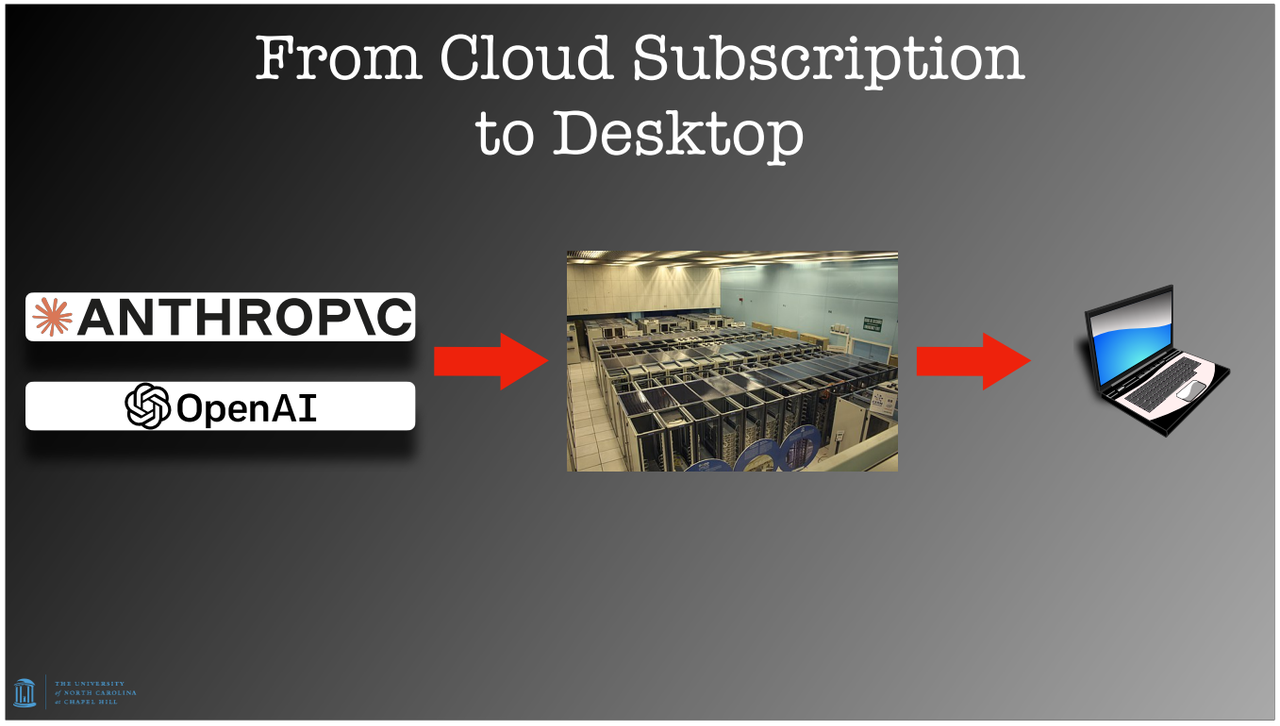
Story time...

- This all started with putting in an RFP back in the Spring because I wanted to share how to configure **Visual Studio Code** (**VSCode**) + the **Continue.dev** extension to use a local LLM (**Qwen3 Coder**) and avoid paying for an AI subscription!
- The irony: I don't even use any of these anymore!



—SKIP FOR PRESENTATION—

This is simply to show how fast things are moving in this space.



Most people, once they get past basic playing with ChatGPT, if they do anything more, they go and pay a subscription of $10, $20, up to $200/month to one or more of the Frontier AI companies where all they use is the tool that each company provides. For example, getting a subscription with Anthropic, then downloading and using Claude Code. Or subscribing to Open AI and using Codex.

Some might then move to getting a subscription with a third party that hosts frontier or what are called “open weight” models to try and reduce their costs. Basically moving the LLM from the frontier AI company datacenters to others who run various models at different rates.

Others still might run such models in their own datacenters if they have the compute capacity. And this is where the line crosses from running LLMs you do not control to those over which you do have some or all control.

And this goes all the way down to running LLMs directly on your own PC/laptop.

The “Why”

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

I always prefer covering the “why” of something before the “how”. So why would anyone want to run their own LLMs—locally or otherwise—vs. simply paying a subscription to an AI frontier lab like OpenAI or Anthropic for the likes of ChatGPT/Codex or Claude?

Moral/Ethical Concerns

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Some would argue that the Frontier AI Labs are guilty of the largest copyright theft in human history. That they stole the content from millions of creators without permission, without attribution, and without compensation, to train their models.

This is a topic which could take up the entire day all by itself. But this is not the purpose of this session.

Data Governance / Privacy

Many folks, for various reasons—personal, professional, legal (compliance/regulations/laws)—do not wish to or simply cannot share data/information, whether simply outside their organization, their country, or their region; e.g.,

- Privacy laws such as the GDPR (General Data Protection Regulations) in Europe
- Corporate governance rules
- NDAs

Privacy matters. And as we will see, using the cloud-based AI services is fraught with major concerns.

Security

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Coupled with privacy is security. Concerns extend well beyond simple privacy to

- leaking corporate trade secrets
- account credentials / passwords
- credit card / banking information
- all manner of information which could be used against you, your organization, or even your country

(Very abridged) History Lesson

1818

Mary Shelley's "Frankenstein; or, The Modern Prometheus"



THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

—SKIP FOR PRESENTATION—

Quick note: This presentation is a gross oversimplification of cherry picked items condensed down into hopefully more easily digestible bits.

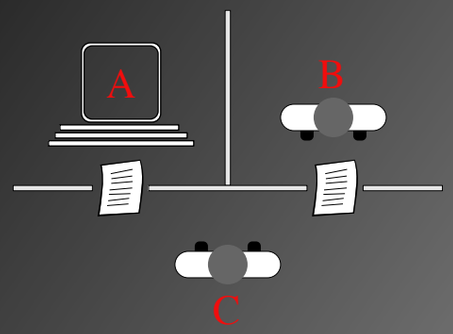
Humanity has had a fascination with the human brain for a very long time. We could back to the early Egyptians and their embalming rituals. But let's keep things semi-recent.

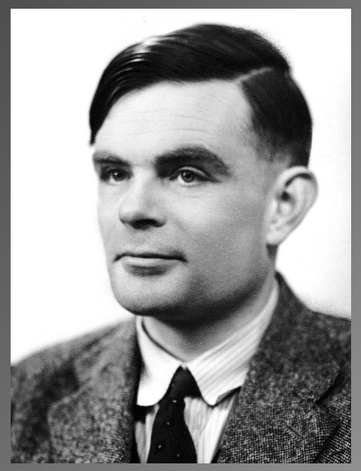
In Mary Shelley's book "Frankenstein; or, The Modern Prometheus", a mad scientist puts dead human body parts together and re-animates them. While the original story does not contain specific reference to the brain, when the book was adapted to a movie in 1931, that bit was added.

(Very abridged) History Lesson

1950

The Turing Test





THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

—SKIP FOR PRESENTATION—

OVER 75 YEARS AGO… “The Turing test was designed by Alan Turing to assess a machine's ability to exhibit intelligent behaviour equivalent to that of a human by imitating interactive dialogue. In the modern version, a human evaluator judges a text transcript of a natural-language conversation between a human and a machine.”

“Player C, the interrogator, is tasked with determining which player – A or B – is a computer and which is a human using only the responses to written questions”

- https://en.wikipedia.org/wiki/Turing_test

(Very abridged) History Lesson

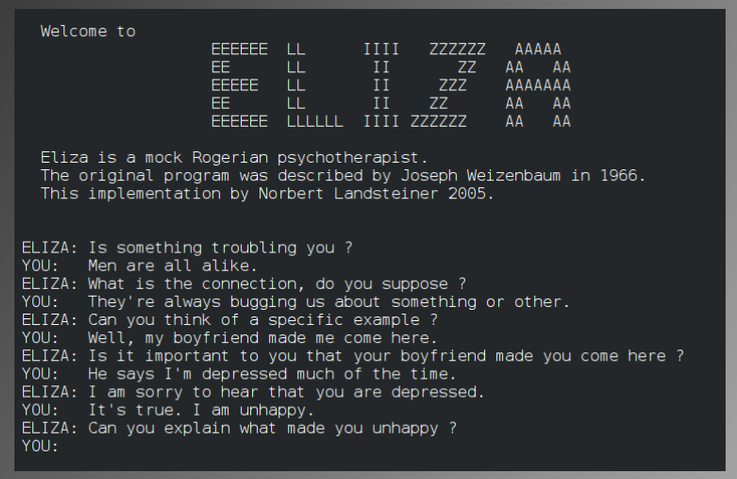
1964-1967

ELIZA

Developed at MIT

by

Joseph Weizenbaum



THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

“Weizenbaum first implemented ELIZA in his own SLIP list-processing language, where, depending upon the initial entries by the user, the illusion of human intelligence could appear, or be dispelled through several interchanges.[2] **Some of ELIZA's responses were so convincing that Weizenbaum and several others have anecdotes of users becoming emotionally attached to the program, occasionally forgetting that they were conversing with a computer.[3][page needed] Weizenbaum's own secretary reportedly asked Weizenbaum to leave the room so that she and ELIZA could have a real conversation. Weizenbaum was surprised by this, later writing: "I had not realized … that extremely short exposures to a relatively simple computer program could induce powerful delusional thinking in quite normal people.” (Mind you, he wrote that in 1976!)**

- https://en.wikipedia.org/wiki/ELIZA

Now imagine the total computing power in those days. Even the most powerful computers could not match what you have in any of your IoT devices, let alone the computers sitting in front of you.

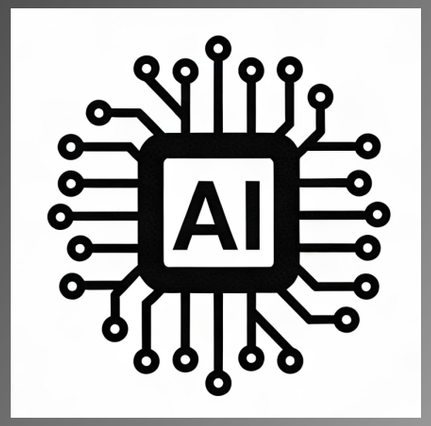
And if someone such as his secretary could react as she did then, imagine with the computing power we have today…

(Very abridged) History Lesson

1980s

Neural Networks





THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

In the 1980s, the term you most often heard was “neural networks”. And often they would describe it this way…

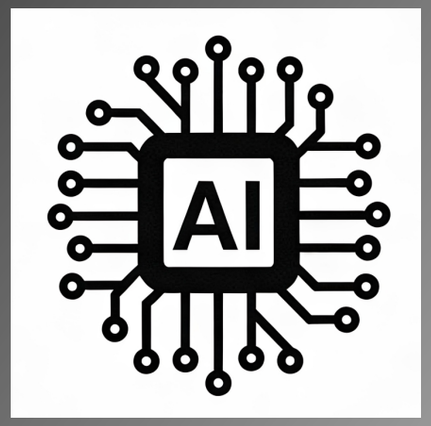
The brain is made up of neurons and synapses. Using the image here, the neurons would be the dots, and the synapses would be the lines/links between the dots.

(Very abridged) History Lesson

1980s

Neural Networks





THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

So let’s talk about that image…

LLM Terminology

token = Neuron (dot)

parameter = Synapse (line)

weight = line thickness

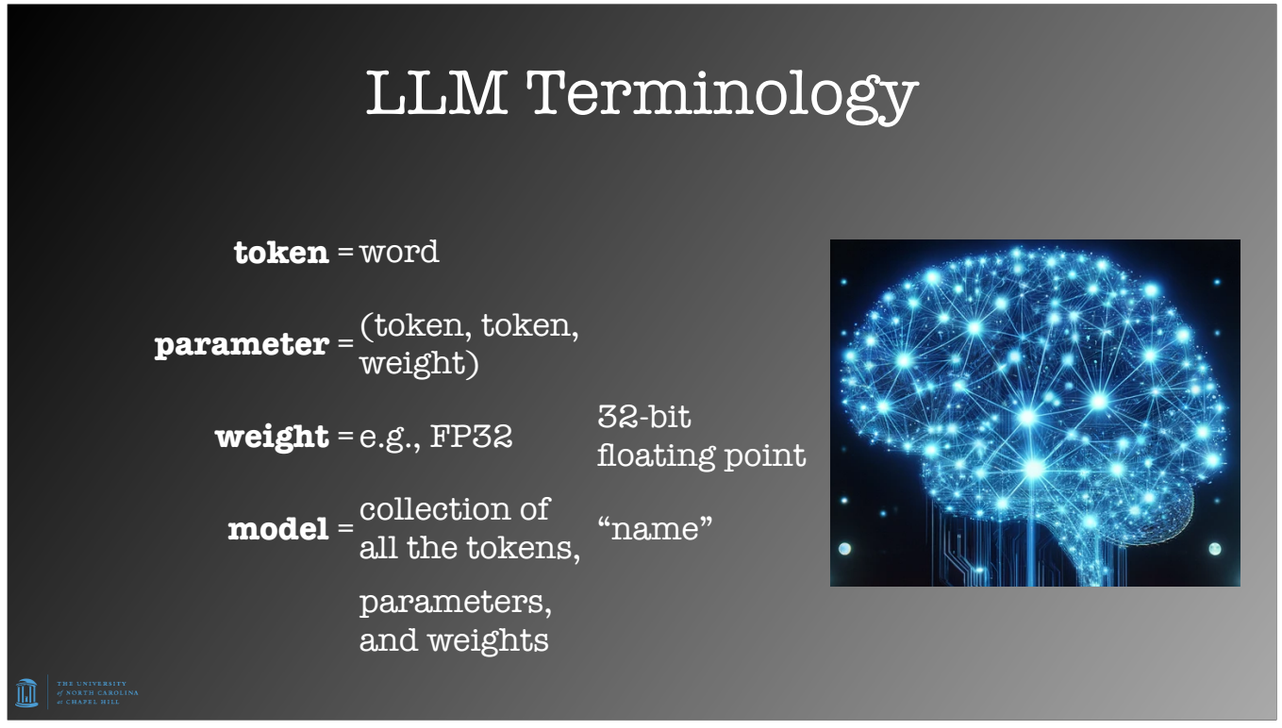
model = brain



THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

In that image we see the neurons as dots and the synapses as the lines between the dots. But those lines vary in thickness. And, of course, the whole collection of neurons and synapses make up the entire brain.

So now let's learn our first LLM vocabulary words.



[~7.5m in]

To put it another way, when working with Large LANGUAGE Models, we can think of **tokens** as **words**. (Technically tokens may be words are just parts of words, but for our purposes, let’s just imagine each token is a unique word.)

We can think of **parameters** as a tuple consisting of two (2) tokens and a weight, which represents the thickness of the line between those two tokens.

The **weight** can be a 32-bit floating point number which represents the thickness of the line, or the strength of the association if you will, between two tokens.

And the **model**—which is the entire collection of all the tokens, parameters, and weights—is given a name, which we will discuss in a moment.

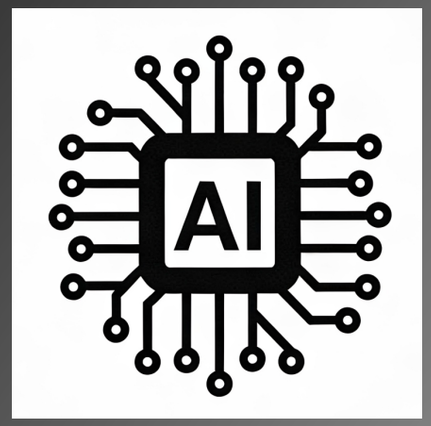
LLM Terminology

Model

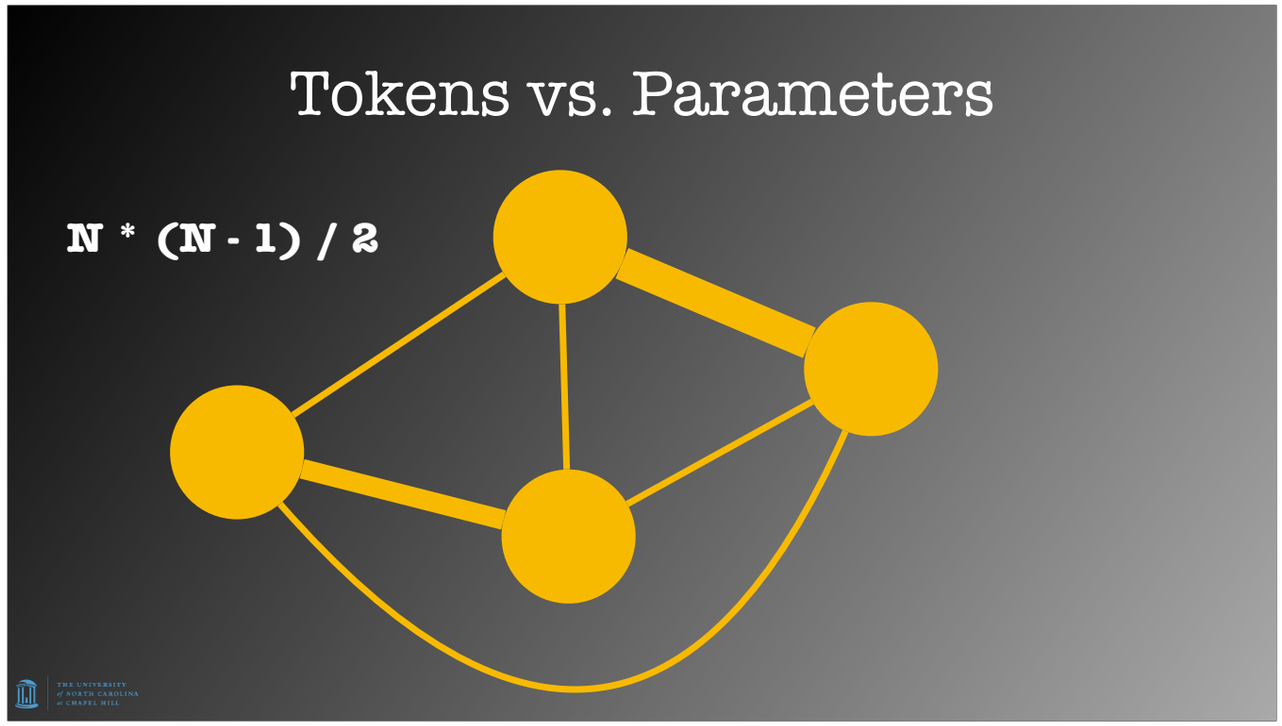
Parameter

Weight

Token



Now with those 4 terms, you have the basics of understanding folks discussing LLMs. But let’s go further.



So we now have tokens (neurons) and parameters with weights (synapses). Just as a point of reference, if we think of this like graph theory, let's see what happens as the number of tokens/neurons increases.

With just 2 tokens/neurons, there can really only be 1 parameter/synapse. Add 1 more token/neuron, and you can have up to 2 more parameters/synapses. Add 1 more again and you can have up to 3 more for a total of 6 parameters/synapses.

Basically n*(n-1)/2 for n tokens/neurons. You can see how this will grow very quickly.

Biology vs. Technology

	Neurons	Synapses
Fly (insect)	140K	50M
Human	86-100 BILLION	100-500 TRILLION
Largest Open Weight Models		< 3 TRILLION parameters

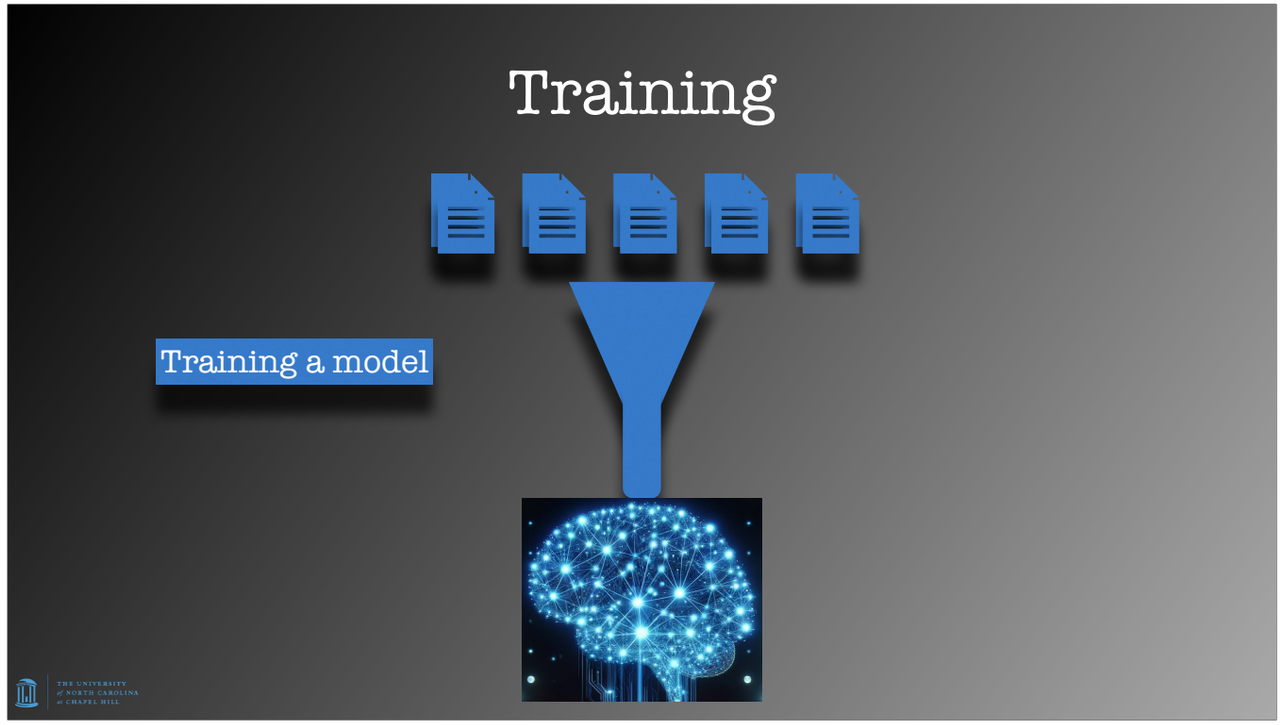
THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Now in the real world, generally speaking, brains with more neurons and synapses are considered more intelligent.

For example, a fly's brain has ~140K neurons and ~50M synapses. (A full mesh would have nearly 10B synapses!)
By comparison, the human brain has somewhere between 86B and 100B neurons and roughly 100 to 500 TRILLION synapses!

To put things in perspective, currently the largest open weight models have somewhere on the order of < 3T parameters. (Kimi K3 weighs in at ~2.4T.)
And that is for LLMs running on datacenters worth of compute!

Fun fact: the human brain runs on about 20W of metabolic power. Keep that in mind when listening to how much power these LLMs consume.



—SKIP FOR PRESENTATION—

So what does it mean to train a model? While we do not have time to go into the specifics here, just look at it this way. A model is simply a set of files which contain

- all the tokens
- all the parameters
- all the weights

Imagine a system wherein you feed an immense amount of data into an algorithm that chews on that data, splitting up the content into tokens (tokenization), then running a lot of math to determine the weights of the parameters between a set of tokens, adjusting those weights as more content is fed in and patterns repeat.

Feed a certain amount of data in, and you get a certain sized set of tokens and parameters. If you add more content with more unique words, the set of tokens and parameters continues to grow. Even once there are no new tokens to be had, because you have already provided all the words, new parameters might still be possible. And those parameters which already exist can have their weights adjusted over time to reflect how strongly they are connected/ associated.

LLM Example

- “It was the best of times. It was the worst …”?
- “It was the best of times. It was the **worst** of …”?
- “It was the best of times. It was the **worst of** times.”?

Congratulations! You are an LLM.

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

For those of you familiar with Charles Dickens, you will recognize this as the opening sentence of “A Tale of Two Cities”.

Congratulations. You are an LLM! This is how LLMs work.

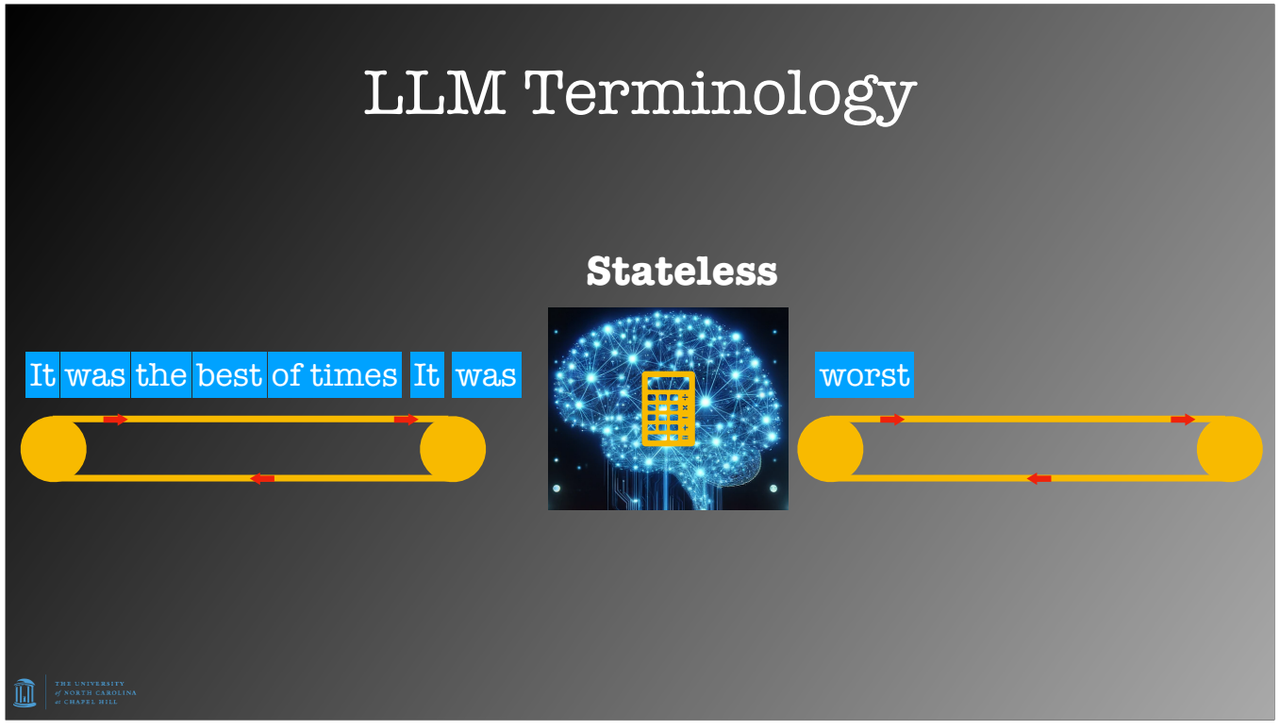
How LLMs Work

- Fancy word probability engine
- You provide an LLM with a sequence of tokens (a “prompt”), and the LLM takes in that prompt, does a lot of math using the model’s parameters and weights, and it spits out the next most likely token.
- It then takes the original prompt + the output token and does it all over again. Rinse/repeat.

LLM Terminology

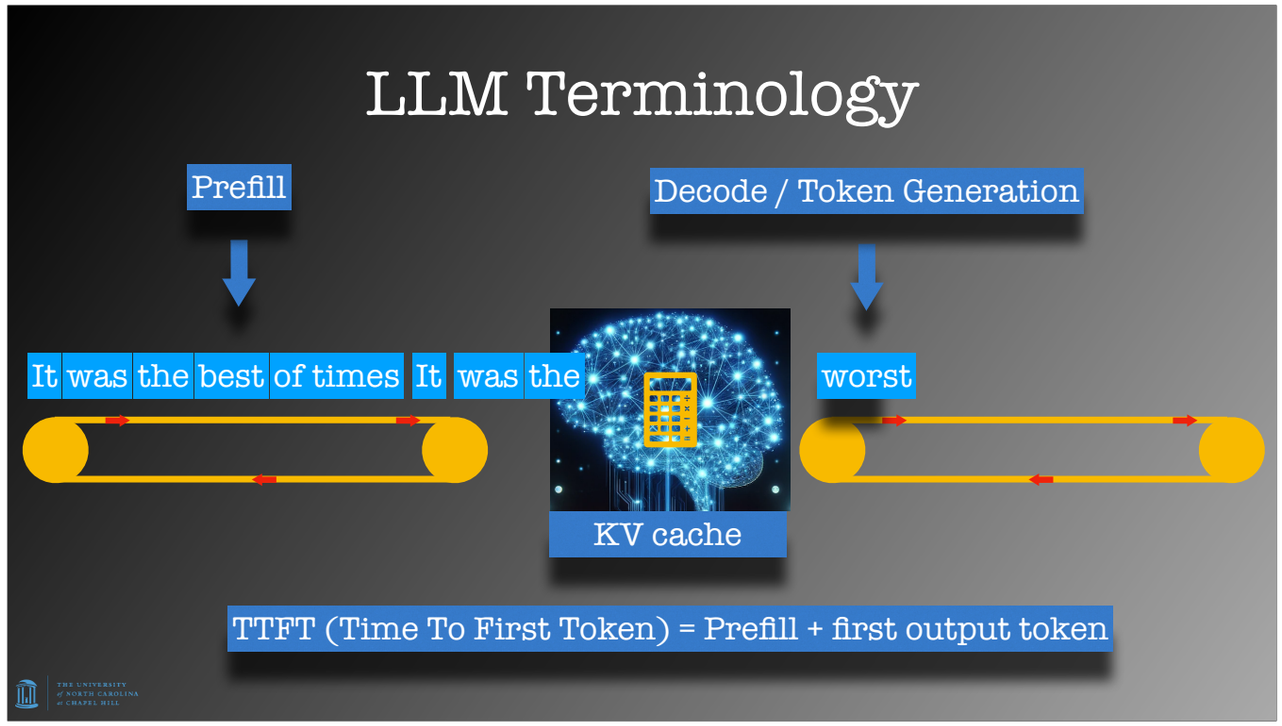


Think of it like a conveyor belt. Tokens/words are loaded up on the belt and fed into the LLM.
The LLM performs a bunch of math on all the tokens and figures out what the next token should be.
The LLM then spits out the next token.



Next, the LLM takes your original text + the output token and feeds it right back in again. Rinse/repeat for each token generated.

It should be noted that all LLMs are stateless. That is, once they process a prompt and generate their single output token, the process to create the next output token is like starting all over again for every.single.token.



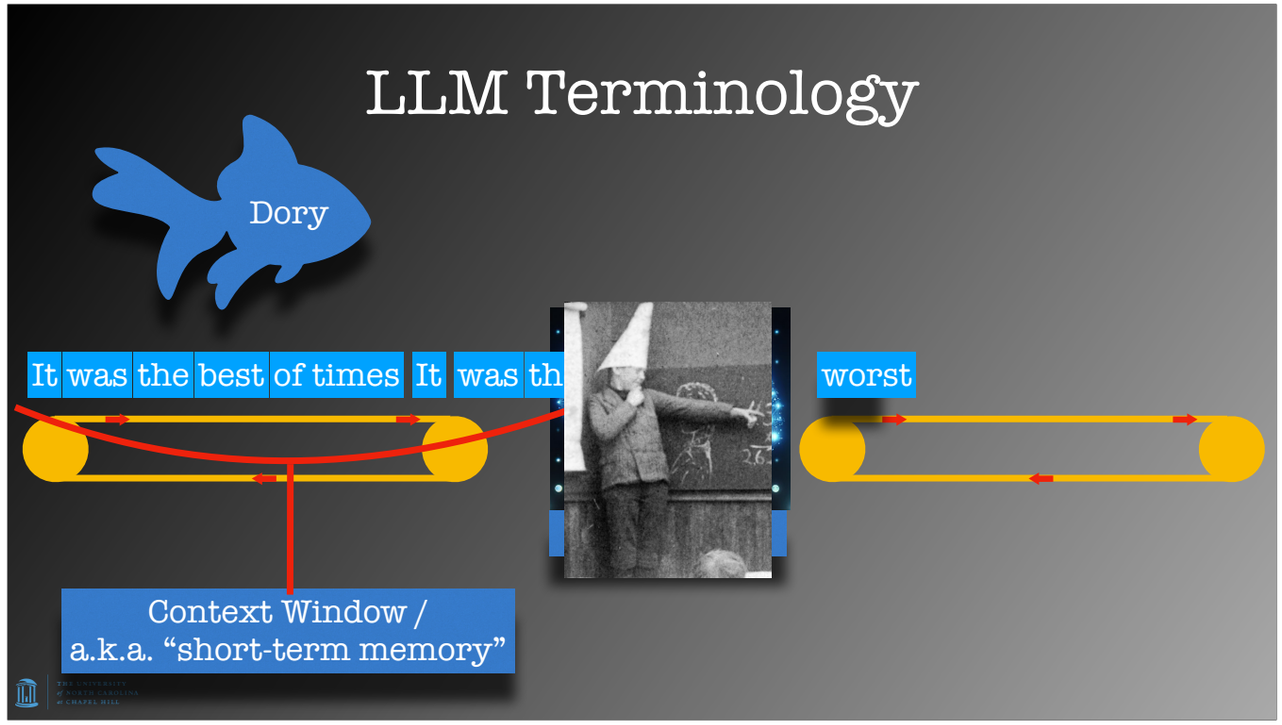
The stage of loading the prompt/tokens into the LLM is called “prefill” and is done in one parallel pass. This stage computes what is called the Key-Value (KV) cache, which stores the internal attention states of the prompt.

Next is the decode stage, also referred to as token generation, where the model generates a response one token at a time. When you hear about most LLMs with respect to performance, they typically talk about tokens per second (tps) which is this decode stage. This stage is memory-bandwidth-bound, since generating each output token requires, for each new token, transferring the entire model into the GPU for processing. The larger the model, the slower the decode. You can calculate the max. theoretical tps with

Max Tokens/sec = Memory Bandwidth (bytes/sec) / Active Model Size (bytes)

This is why the memory bandwidth of a GPU is so important, and why high end systems tend to use HBM (High Bandwidth Memory).

The TTFT (Time To First Token) is equal to the time it takes to perform the prefill + generate the first output token.



[~15m]

The context window is the max. number of tokens the model can hold, process, and reference in a single request.

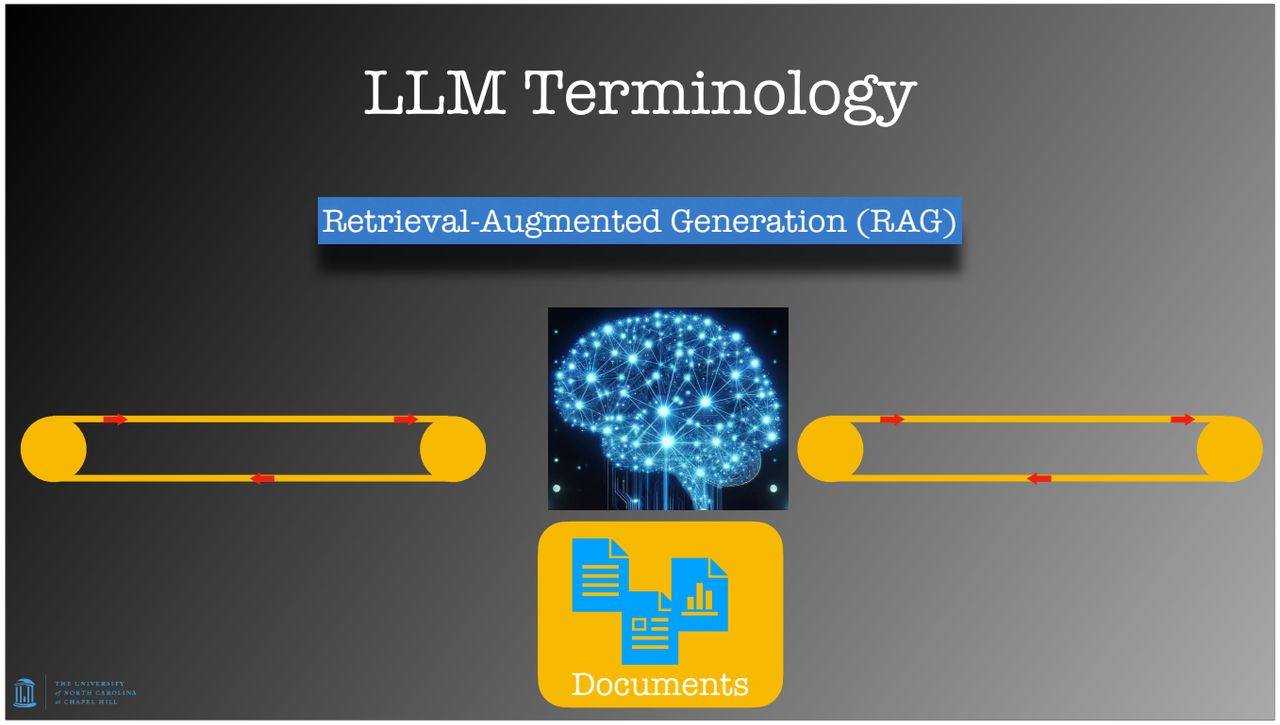
For those familiar with the Pixar animation “Finding Nemo”, think of LLMs like Dory, the blue tang fish with the very short attention span.

The context window is a fixed size, FIFO memory block. This can be anything from a very small window such as 8K tokens to 32K, 128K, 256K, or even 1M or 2M tokens, though the latter is often only found on frontier models running on major compute platforms.

Using the conveyor belt, think of the context window length as how many tokens can fit on the belt. Once the context window is full, as new tokens are added on the right, the oldest tokens on the left “fall off”.

(For those who remember the famous “I Love Lucy” episode where Lucy and Ethel work in a chocolate wrapping factory, remember how once Lucy could not keep up with the wrapping, chocolates starting falling off the end, where she tried to scoop them up, put them in her pockets and elsewhere? Only here once a token “falls out”, it is simply gone.)

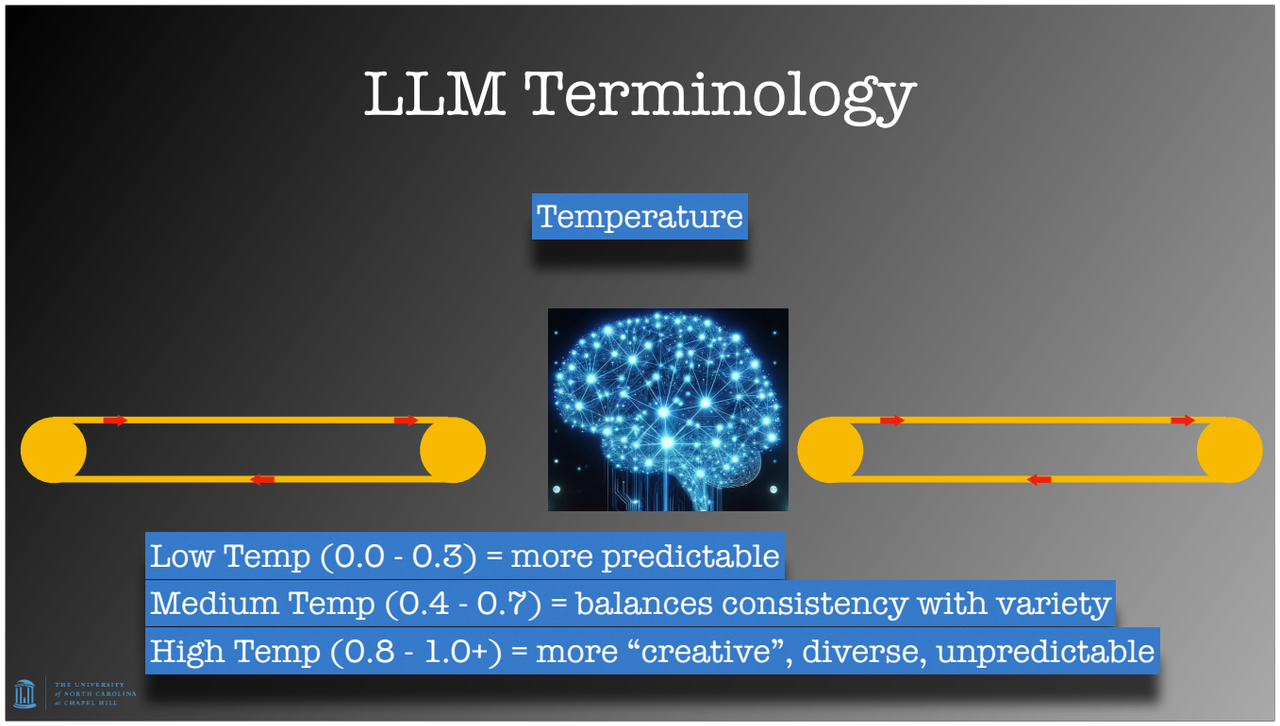
Think of this as a digital form of Alzheimer’s. In fact, when folks talk about LLMs suddenly becoming “stupid”, this is often the reason that occurs. There are various techniques for preventing this, such as compressing the context window content every so often, or otherwise summarizing what was in the context window in a shorter form to free up enough space for new tokens. But ultimately this is a limiting factor, much as it is for humans. When asked to remember a sequence of numbers, most people can only really remember 7-13 single digit numbers at any given time.



Let's say you are a network engineer who works on Cisco gear. And you want to use an LLM to help you troubleshoot an issue. But you have all these Cisco PDF manuals and documents, and you'd like the LLM to be more than just a "generic brain" and rather have it read/use that information to help inform its answer. This is referred to as Retrieval-Augmented Generation (or RAG for short).

RAG lets you make your LLM more focused / "smarter". The process involves taking your documentation, processing it, typically into the form of what is known as a vector database, and then letting you have your LLM access those documents while doing its calculations. Doing so results in the output often being more tuned to the task at hand.

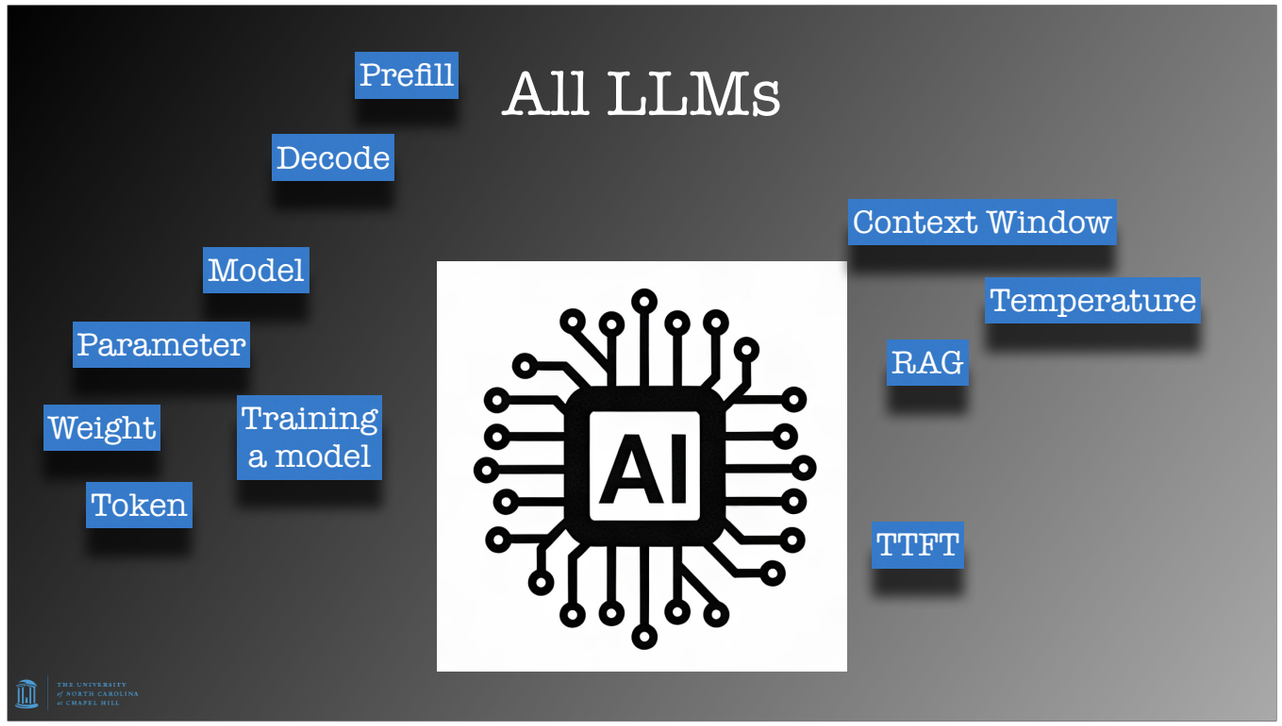
For those familiar with Google's NotebookLM, which I believe now is called Gemini Notebook, this is an example of RAG.



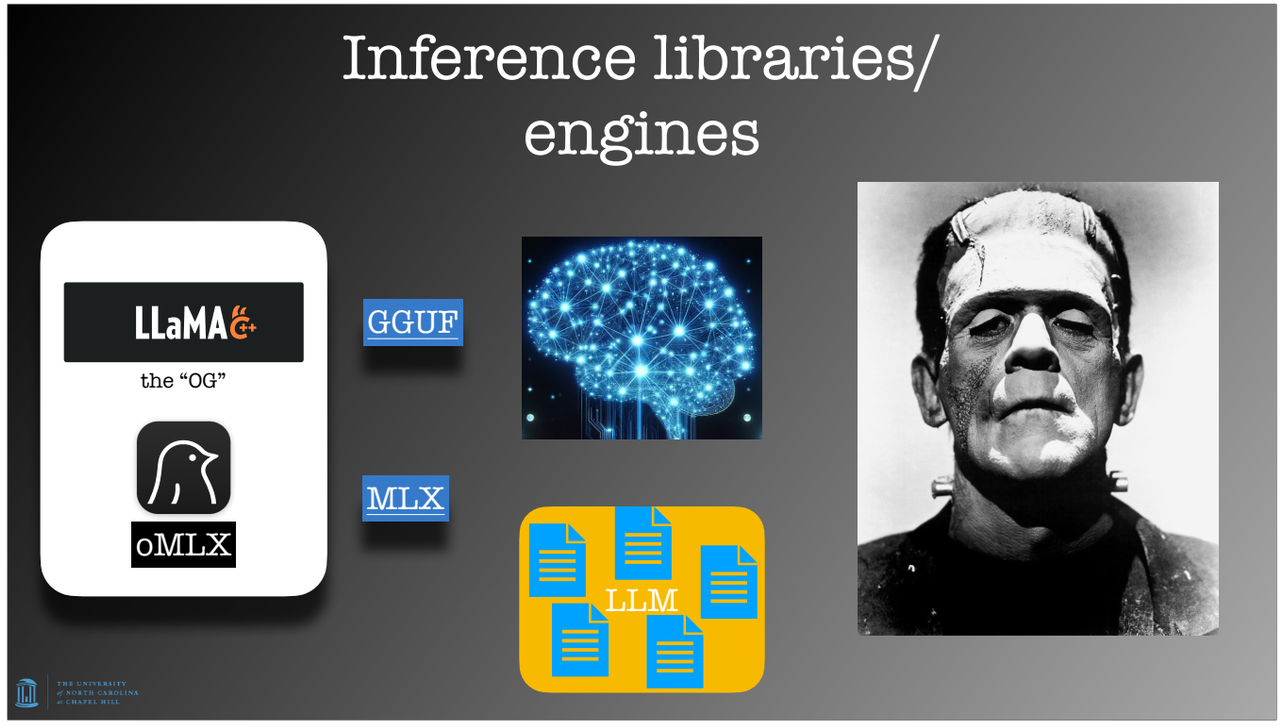
An LLM temperature is a hyperparameter that controls the randomness and creativity of text generated by a large language model.

LLMs basically perform token prediction based on a probability distribution. Think of temperature as a randomization factor.
A low temperature (0.0 to 0.3) means the outcomes tend to be more predictable, repeatable, and closer to deterministic.
A medium temperature (0.4 to 0.7) balances consistency with natural conversational variety.
A high temperature (0.8 to 1.0+) flattens the distribution, giving less likely tokens/words a stronger chance to be chosen. Outputs become more “creative”, diverse, and unpredictable.

Why would you choose a higher temperature? Well if you are using an LLM to help you write a fictional story…



Up until now, everything we have covered applies to any LLM, whether cloud-based, local, or otherwise.



An LLM is nothing but a set of files containing tokens, parameters, and weights. By itself it can do nothing. It is like a brain without a head. Think a pile of MS Word documents. They are useless without a program like MS Word or LibreOffice to open/use them.

So going back to our monster scenario, what we need is some kind of “head” to put the “brain” into. This is the role of an inference library or engine. Just a few examples here are things like **llama.cpp**—the “OG” of local LLM inferencing—and **oMLX**—which runs LLMs designed to run on Apple Silicon’s Metal GPU framework.

And since we mentioned those, let’s knock out a few more terms. When working with offline LLMs, you will often see references to

- **GGUF** (**GG**ML **U**niversal **F**ile), where GGML == Georgi Gerganov Machine Learning. Introduced by llama.cpp project. Think of this as the standard in LLM model file formats.
- **MLX** (Apple’s Machine Learning framework)

These are binary file formats that store both tensors and metadata in a single file and are designed for fast saving and loading of model data.

“Harness”





THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

But even a head with a brain is a bit useless. I mean you can talk to it, like talking to a head in a jar in some sci-fi movie. But that’s about it.

What we really need is a “body” for that “brain” to control.

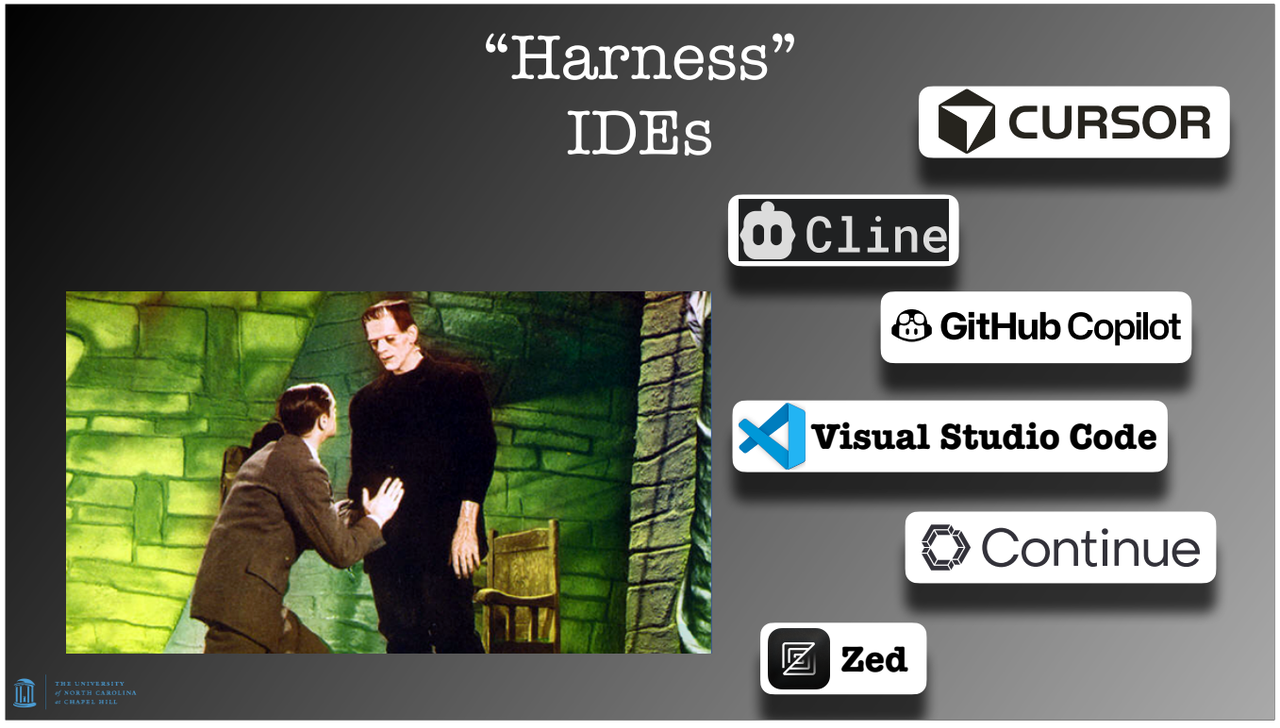
This is what is referred to as a “harness.”



This is what most people are initially familiar with: a web interface to interact with an LLM.

Examples of a web-based harness include

- **ChatGPT**
- **Claude**
- **Google Gemini**
- **Perplexity**
- **LiteLLM** (self-hosted)
- **Open WebUI** (self-hosted)



Examples of an IDE harness include

- **Cursor**
- **Cline**
- **GitHub Copilot**
- **Visual Studio Code (VSCode)** + GitHub Copilot or Cline or **Continue.dev** (now part of Cursor)
- **Zed** (my personal favorite; written in Rust by creators of Atom)



Examples of a harness include

CLI tools: **Claude Code**, **Ollama**, **OpenCode**, **Crush**
Desktop tools: **Claude Desktop**, **ChatGPT for Desktop**, and **Google Antigravity**; **Block Goose**, **Unsloth Desktop**, **LM Studio & Bionic**, **AnythingLLM**
Web interface tools: **LLama** (formerly LLamaBarn; macOS only)

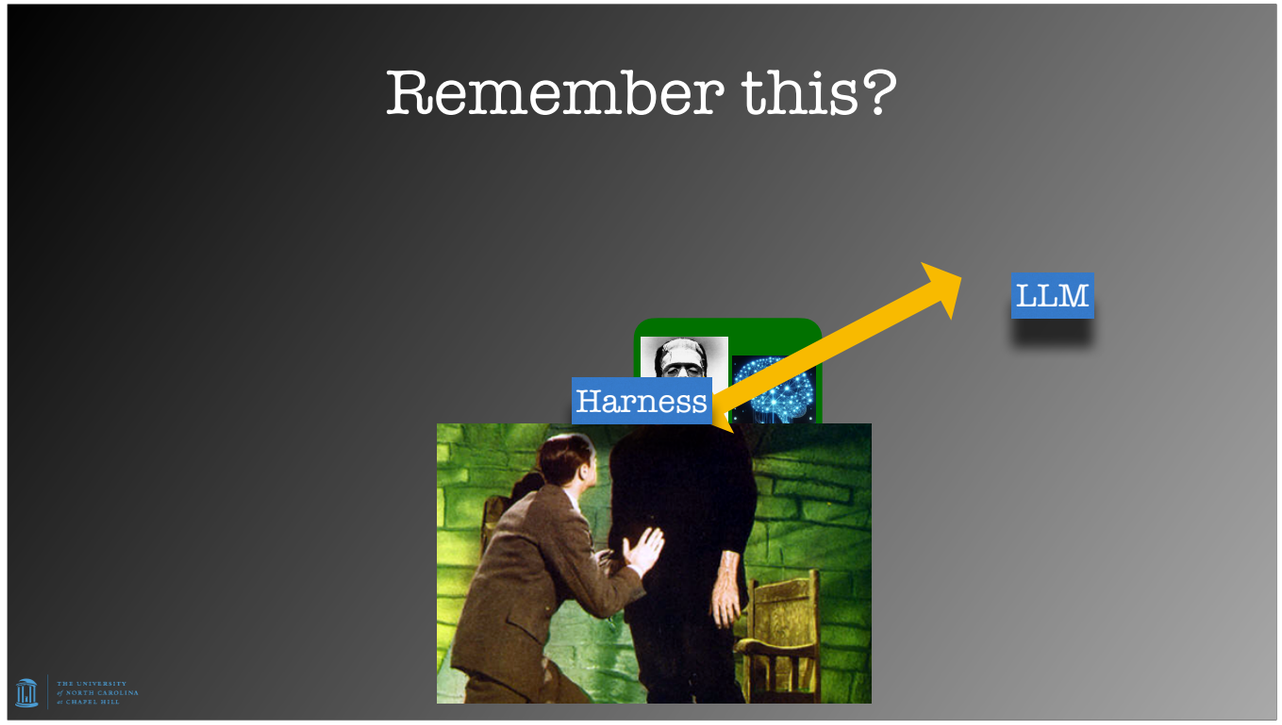
NOTE: Some of these programs act as both inference engine (loading/running models) AND harness (e.g., providing a chat interface, RAG features, coding agent, etc.).



Examples of an AI agent harness (i.e., “an agentic harness” where the agent is like your assistant) include but surely are not limited to

- OpenClaw (written in TypeScript) — the “OG” of agentic AI tools
- nanobot (written in Python)
- PicoClaw (written in Go)
- NanoClaw (written in TypeScript)
- MicroClaw (written in Rust)
- ZeroClaw (written in Rust)
- Pi (written in TypeScript) — what OpenClaw is wrapped around. A minimal agent loop.
- Hermes Agent (written in Python)

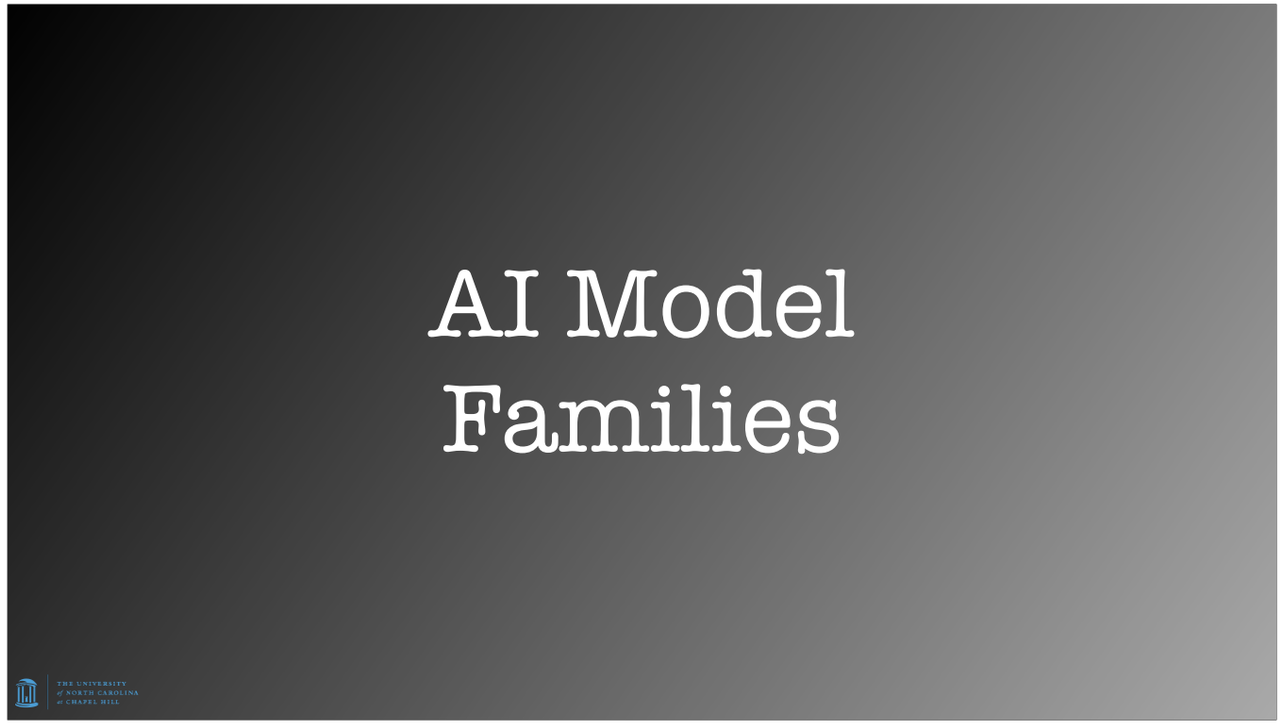
In the news recently you may have seen information/ads for Meta Muse or Grok Bot. The big companies are, in fact, behind in this regard, as OpenClaw, written by one person as an open-source project, is the “OG” in this space. In fact, OpenClaw’s creator, Peter Steinberger, was hired by OpenAI to work on personal agents.



So now you can see how an AI setup is like your own personal ~~monster~~ AI assistant, with a brain (LLM) and a body (harness). And where each of those are located varies.

But what is important to remember is that it is the “body” that does the actions. Anything/everything that is done such as reading/writing files, connecting through a network, etc., that is all from the harness. If you run a harness on your computer, keep that in mind as it could access your files/etc.

And it is the “brain” that processes all that data. So it matters where the LLM is located, as all information flows through it, even though it is the harness that performs the actions. So if you use a coding agent on your laptop that ties into Anthropic or OpenAI’s cloud-based LLMs, then anything/everything your coding agent has access to / “sees” is being transmitted to those LLM servers out in the cloud.



[~25m]

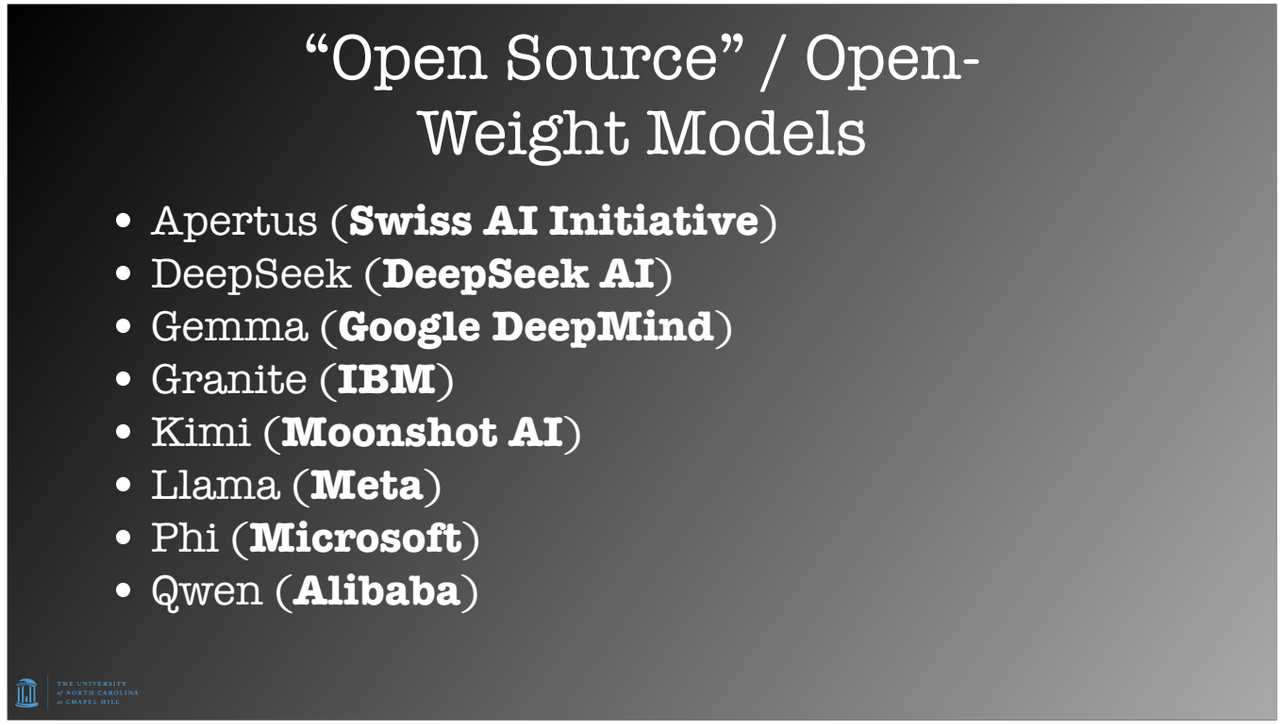
Next up we need to talk about the various AI model families.

AI Model Families (Proprietary)

- Claude (**Anthropic**)
  - Fable
  - Haiku
  - Opus
  - Sonnet
- Gemini (**Google DeepMind**)
  - Flash
  - Pro
- Grok (**xAI**)
- GPT (**OpenAI**)
  - Microsoft Copilot
- Mistral (**Mistral AI**)
- Muse (**Meta**)
  - Spark
  - Wonder
  - Wonder Full

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Most folks know the names of the major companies involved. The key thing to note here is when you hear of one of these families of models, each is tied to a specific vendor.



Remember when I mentioned that models have names? Well these are the names you will most likely see when working with LLMs locally. So if someone says “I am running Qwen3.8 27B”, they are saying that that they are using a model from Alibaba. The name typically reveals a bit about the model, including the version and the number of parameters it has.

Now the term “open source” in the LLM world is a bit misleading. With the exception of Apertus, which is a specific initiative to create a truly open source LLM where anyone not only has access to the weights and parameters but also all of the original content used to train the model, most LLMs you can download are really what are called “open weight” models, meaning you have access to the weights and parameters. But you do NOT have access to the actual content used to train the model, meaning you have no idea what was used to “teach” that “brain”.

But again here the point is simply to say that when you hear someone saying they use a certain model, you understand that the model comes from a specific vendor.

Frontier AI Models

- TRILLIONS of parameters
- TERABYTES of files just containing the parameters, weights, etc.
- Training costs range from tens of millions to over $300M, with upcoming models projected to cost over $1B!

Now when it comes to these frontier AI models, the numbers are staggering. Which then begs the question…

Wait, why would AI companies offer open-weight models?

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

—SKIP FOR PRESENTATION—

Hang on. The “majors” like OpenAI and Anthropic spend BILLIONS of dollars to train their models. Estimates for Anthropic’s Mythos model cite

- training costs around $10B
- training data sets in the range of 15.5 TRILLION tokens
- a storage requirement of “several terabytes” just to hold the raw weights

Why would anyone who spends that kind of money and effort then offer up versions of their models for others to run/use?

Key Reasons for Sharing Models

- Ecosystem Adoption and Lock-in
- Hardware and Infrastructure Sales
- Shifting Cost and Responsibility
- Competitive Pressure and Commoditization
- Community-Driven Improvement and Safety
- Enterprise Control and Data Privacy

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

—SKIP FOR PRESENTATION—
Suffice it to say there are many reasons why a company might share their models as outlined here.

Consider “First mover advantage”, for example.

And sometimes the competitive advantage is in offering something for free simply to drive down the cost in the market to make it harder for your competitors to charge as much as they do.

Example: APIs

- Open AI API
- Anthropic (Claude API)
- Google Gemini API (Google AI Studio / Vertex AI)
- DeepSeek API

These are used by “harnesses” to hook into their LLMs. Think of it like “developer mindshare.” If everyone uses your API, then any software developed likely will use it as well.

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

—SKIP FOR PRESENTATION—

Examples:

- https://developers.openai.com/api/reference/overview
- https://platform.claude.com/docs/en/cli-sdks-libraries/libraries/openai-sdk

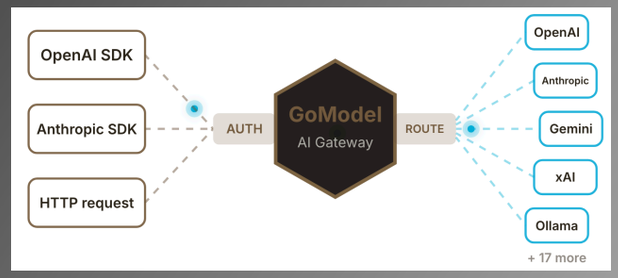
AI Gateways

OpenRouter

LiteLLM

GoModel

e.g.,



THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

—SKIP FOR PRESENTATION—

Side note: This led to an entire new category called AI Gateways. One of the most popular is OpenRouter, which we will discuss later. But there are also self-hosted ones such as LiteLLM, GoModel, and others.

The “How”

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

But even if some vendors (notably the Chinese AI companies) offered a stripped down version of their frontier models, how could anyone USE them? The storage and compute requirements alone are massive! Well…

Story time (part 2)

- Similar to most, began tinkering with AI/LLMs by playing with ChatGPT
  - “Rewrite this text as the Swedish Chef”
- Around May 2025, began using LLMs locally

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

—SKIP FOR PRESENTATION—

I began tinkering with AI/LLMs like most, trying out ChatGPT a few years ago. Basically asking it simple things, like “Rewrite this text as the Swedish Chef from The Muppet Show”. 😀

Then around May 2025, I began running LLMs locally. And this led me down this rabbit hole.

Where Can I Get These Models?

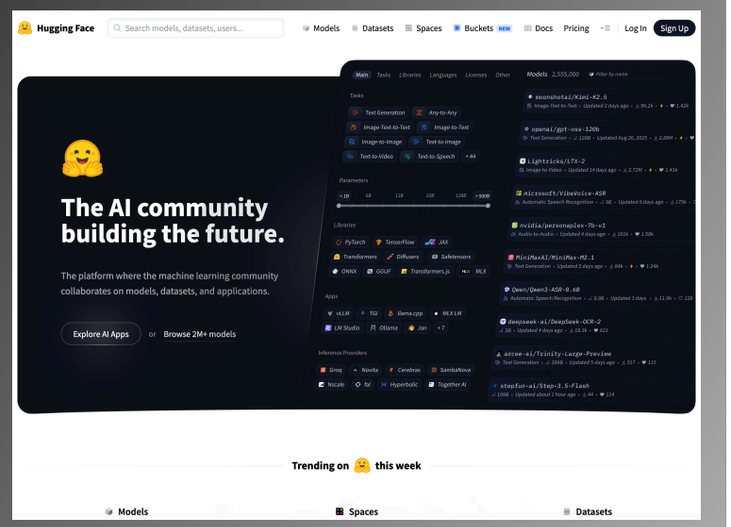
THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Now let’s REALLY get into the “How”…

Hugging Face

https://huggingface.co/

“The Docker Hub / GitHub / Amazon.com of open LLM models.”



Hugging Face is “the” Go-To location for finding/downloading/sharing LLMs for local use. There is an entire ecosystem of folks out there who take the open-weight frontier models, then crunch them down into something capable of being run on smaller systems. This is where the term “quantization” comes in, which we will cover in a minute.

But for now simply note that this is the site almost every local LLM tool out there goes to in order to download models that you can use.

Side note: In September 2026, Nvidia announced that they had agreed to acquire Hugging Face for nearly $13B!

But How Can I Run These?

But How Can I Run These?

You can’t. At least not the frontier models.





https://creativecommons.org/licenses/by/3.0/

The real answer is “You can’t.” You need datacenters worth of GPUs to be able to run true frontier models at full precision.

But How Can I Run These?

You can't. At least not the frontier models.

So somehow they need to take this…

and turn it into this



THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

So we need something that can take this frontier-sized "brain"… and turn it into something small like this…



So that we can run it on local hardware.

This might range from, say,

- an **Nvidia H100** (~$30K originally)
- to a **DGX Spark**, which is powered by an NVIDIA GB10 Grace Blackwell Superchip featuring a 20-core Arm CPU (10 Cortex-X925 and 10 Cortex-A725) and a Blackwell GPU (originally ~$4K but now selling for ~$7K a year later for the same hardware!)
- or an **Apple Mac Studio** w/ M5 MAX or M5 Ultra chip
- or a PC with one or more of the various **RTX GPU cards** like the 3090, 4090, 5090, etc.
- down to an **Apple Mac Mini**
- or even a **laptop**

Frontier vs. Open-Weight Models

	Frontier Model	Open-Weight Model
Parameters	TRILLIONS	BILLIONS
Weights (precision)	16 or 32-bit floating point	8-, 6-, 5-, 4-, 3-, 2-, or 1-bit integer
Context Window ("short-term memory")	1-2M	up to 256K
e.g.	Qwen3.8-2.4T-A95B (Qwen3.8-Max)	Qwen3.8-27B

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Here is a quick comparison of large frontier AI models vs. smaller open-weight models with just one example.

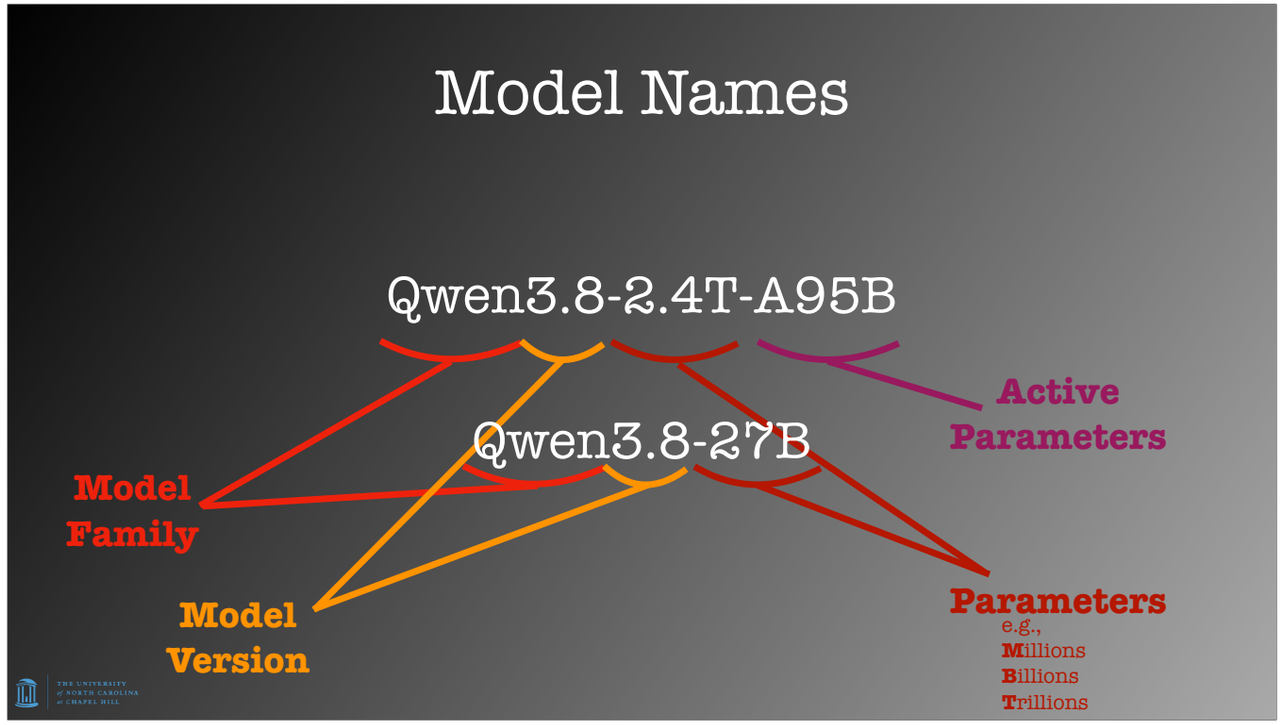
As you can see here, frontier models are measured in TRILLIONS of parameters vs. open-weight models in BILLIONS of parameters. This in essence means the frontier models are a bit "lobotomized" in order to help bring their size down.

Also, the frontier model weights are stored as 16- or 32-bit floating point precision numbers, while the open-weight models typically store them as 8- down to 1- bit integers. So regarding the "thickness" of the synapses/lines, we are going from a range of up to 4 billion possible values down to 256 or less. Obviously this means a loss of specificity.

Next each model has a context window limit. Frontier models typically tap out at 1-2M tokens, while open-weight models typically have a max. of 256K tokens at best.

And as just one example, let's take the Qwen3.8-Max family frontier model here vs. the currently popular Qwen3.8-27B model. The frontier model has 2.4 TRILLION parameters vs. the open-weight model having just 27 BILLION. That is, the frontier model is nearly 100X larger, or the open-weight model is just 1% of the frontier model!

And a quick note. Those names all have meanings, and we will get to that.



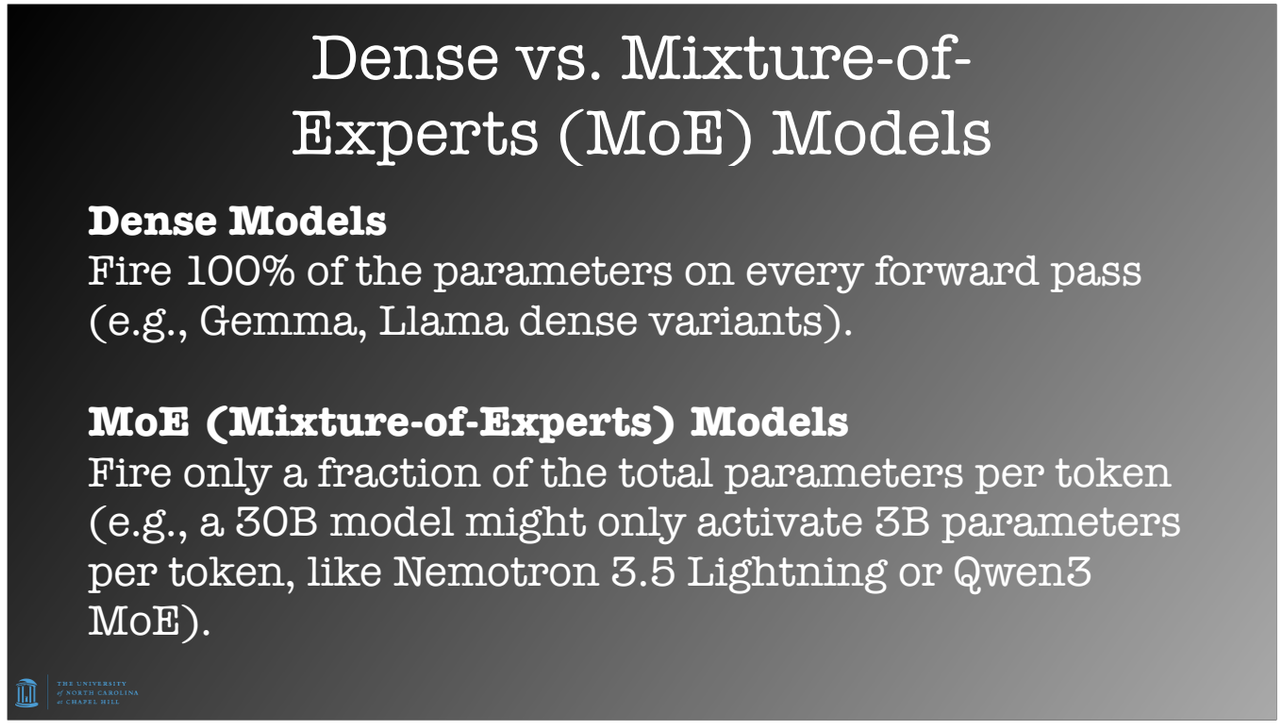
Model names follow a pattern which, once seen, become much easier to digest.

First you have the model family (e.g., Qwen from Alibaba, Gemma from Google, LLama from Meta, etc.)
Next you have the model version. So if you see Qwen 3.5 and Qwen 3.8, you know the latter is newer than the former.
Then you have the total number of parameters, represented as a number followed by “M”, “B”, or “T”, to indicate millions, billions, or trillions respectively.
And finally, if there, you get the total number of active parameters per forward pass. This only appears when dealing with a Mixture-Of-Experts (MoE) model.

Dense vs. Mixture-of-Experts (MoE) Models

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Which brings us to two types of models: Dense and Mixture-of-Experts (MoE)

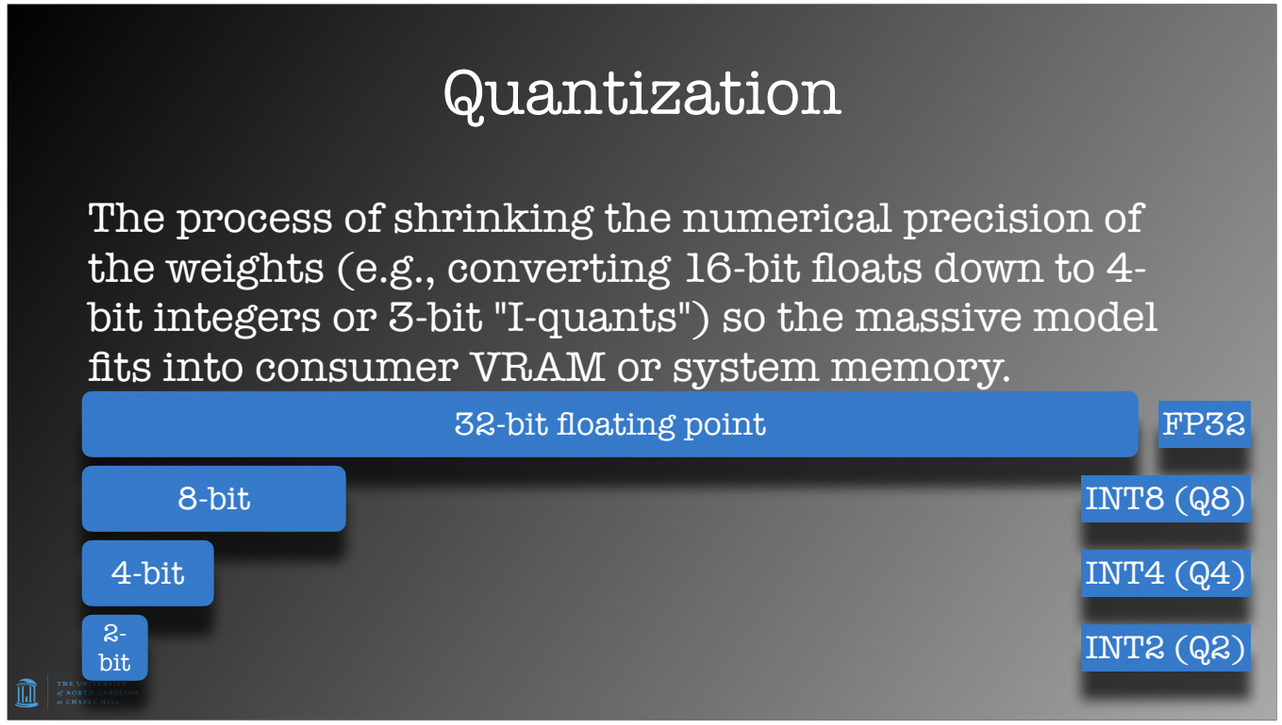


The key thing to note is that, regardless of type, any model requires enough memory to load it in its entirely. So whether you are using the Qwen3.8-27B dense model or the Qwen35B-A3B (MoE model that only activates 3B parameters for each token), first you must load the entire model into vRAM. Then it comes down to how many parameters you “fire” (i.e., do calculations on) for each token generated.

Quantization

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

AND NOW for one of the biggest terms you will hear when talking about open-weight models that you run locally: “quantization”.

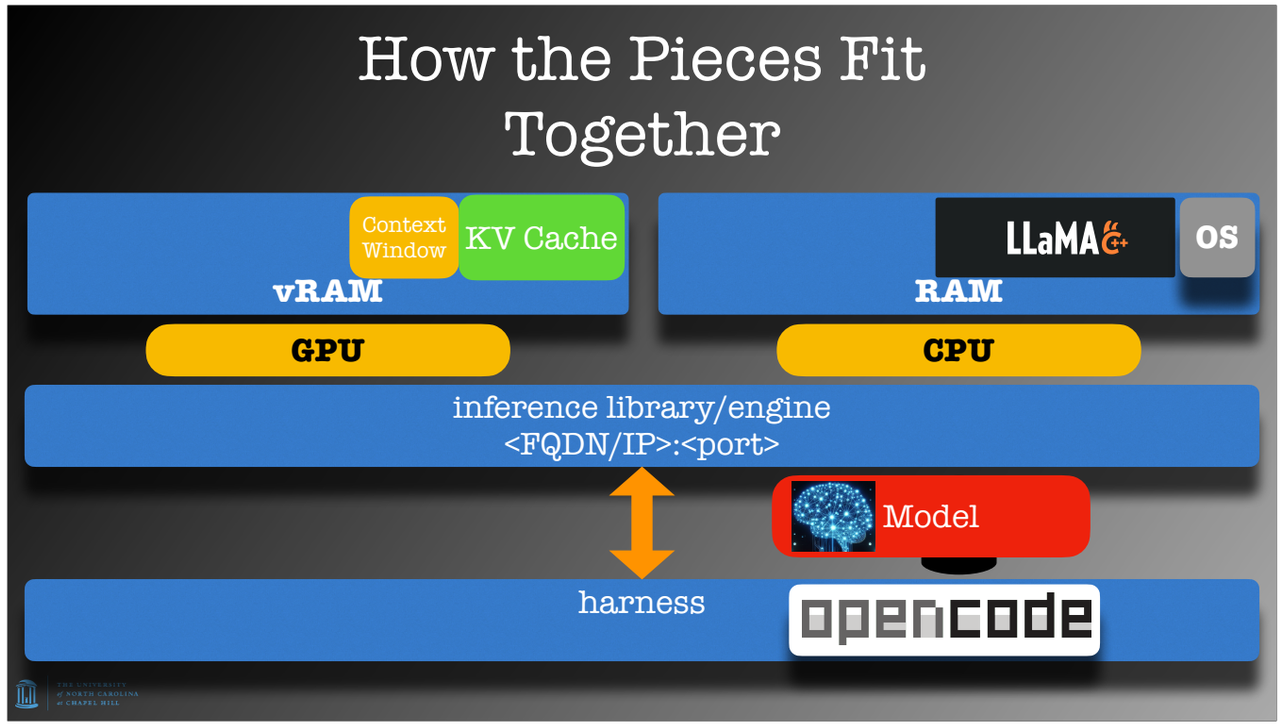


Quantizing a 32-bit FP number down to an 8-bit integer means going from a granularity of 4 billion possible values to a max. of 256. You might also quantize it down to 6-, 4-, 2-, or even 1-bit depending.

The key thing here is to notice how this impacts the total storage requirement of the model. Reducing all the parameters from 32-bits to just 8-bits means you reduce the storage to just 25% (or reduced it from the frontier size by 75%). Quantize to 4-bits and you've reduced the parameter storage by 87.5%. Quantize to 2-bits and you've reduced storage by nearly 94%!

Of course with each reduction you lose some specificity. But amazingly, some models quantized down to just 4-bits perform remarkably well vs. their frontier versions.

And now you can see that when you see Q8, Q4, etc., the 'Q' refers to quantizing, and the number represents how many bits.

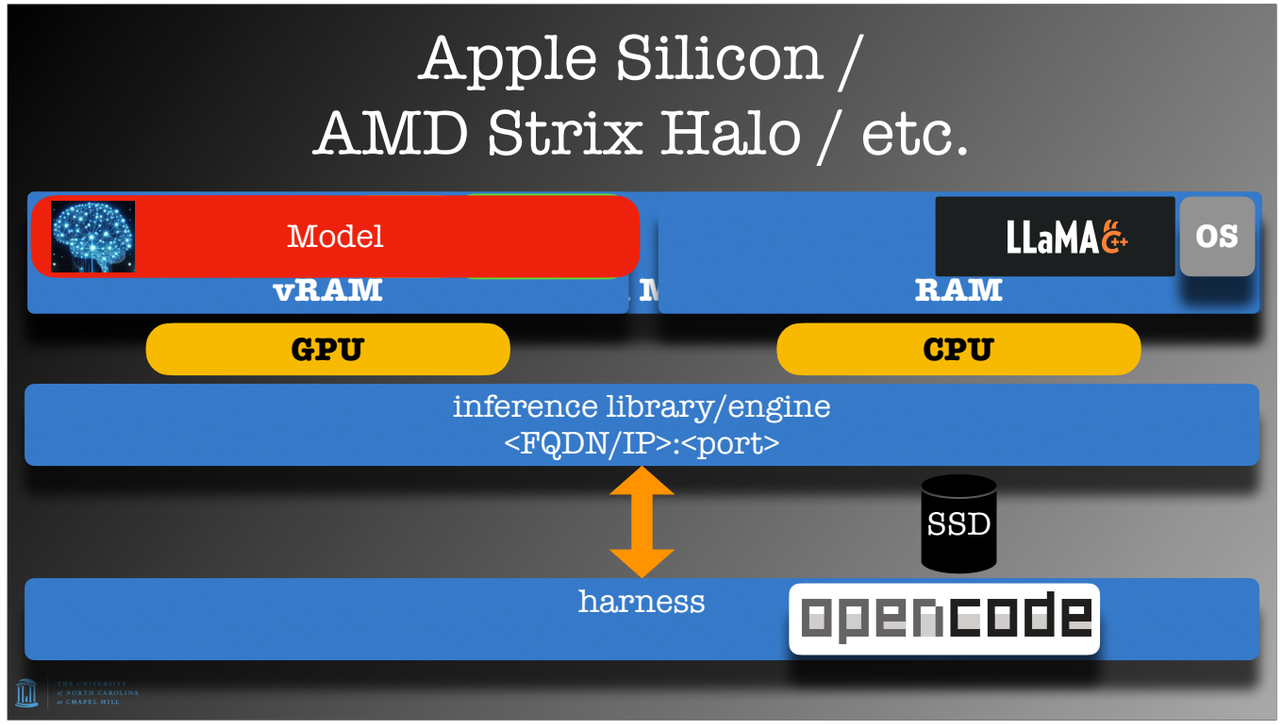


Generally speaking, we think of computers as having CPU, RAM, and maybe storage disk space (HDDs, SSDs). But for the purposes of running LLMs, the focus shifts to GPUs and the total vRAM (video memory, or in some cases high-bandwidth memory (HBM)).

Simply put, the total amount of vRAM available determines how large a model you can load/run. If you have a PC laptop with 128GB RAM and a GPU with only 8GB vRAM, then you can only load a model that fits in 8GB. And you also need to account for the fact that the context window and KV cache will also consume some of that vRAM.

You load/fire up your OS and an inference library/engine in your standard RAM. And the inference engine, in turn, loads up your model into vRAM. That also includes space for your context window and KV cache, the latter of which grows as you do more with the LLM.

Now if you are using an inference engine, it typically then presents a port to which you connect your harness, which may or may not be running on that same system.

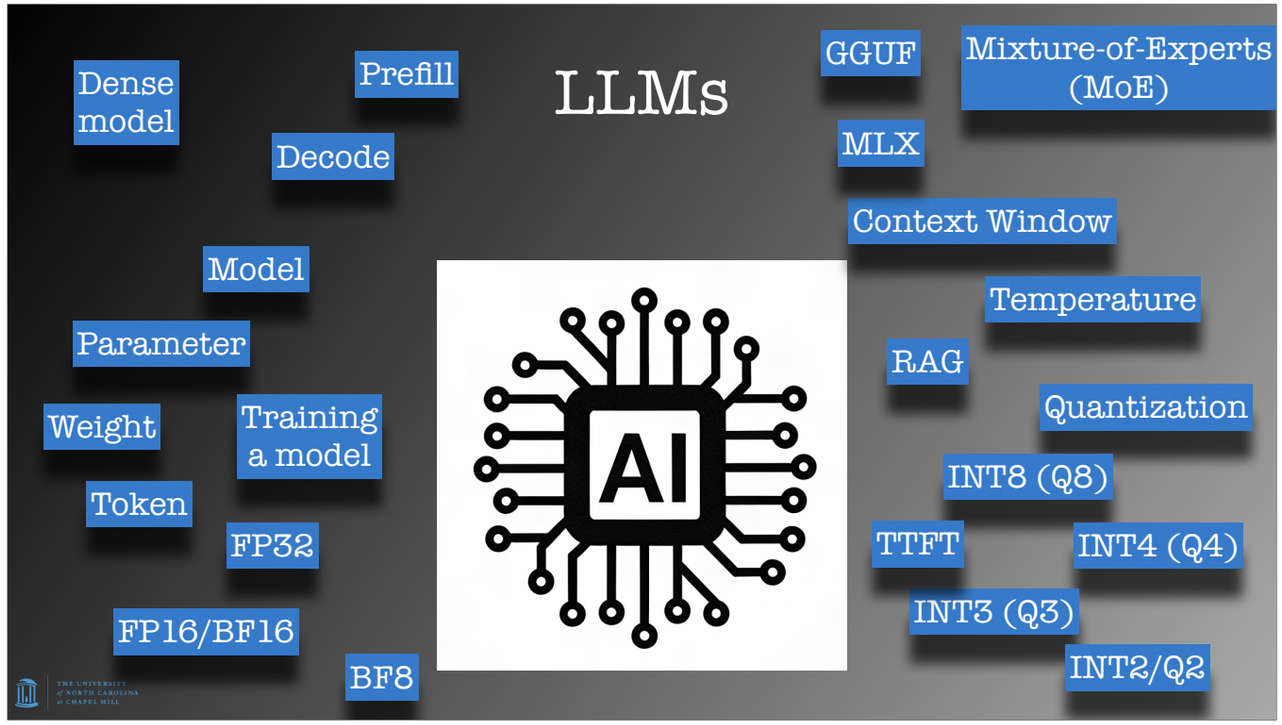


Now let's look at Apple Silicon and similar architectures like the AMD Strix Halo.

These systems have what is called "unified memory." That is, the same memory is shared/accessed between the CPU and the GPU. This has one distinct advantage. It means it is possible to load larger models.

For example, given the choice between a PC with 256GB RAM and a GPU with 8GB RAM or an AMD Strix Halo with 128GB of unified memory, the latter would let you load far larger models even though it has only half the total RAM. (But it has 8x the vRAM.)

The second is cost. Typically unified memory is more like standard RAM, vs. the HBM used on graphics card. However, the key here is the memory bandwidth, as this determines the max. decode/token generation possible.



At this point we have covered all of these terms!

Pop Quiz!

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

—SKIP FOR PRESENTATION—

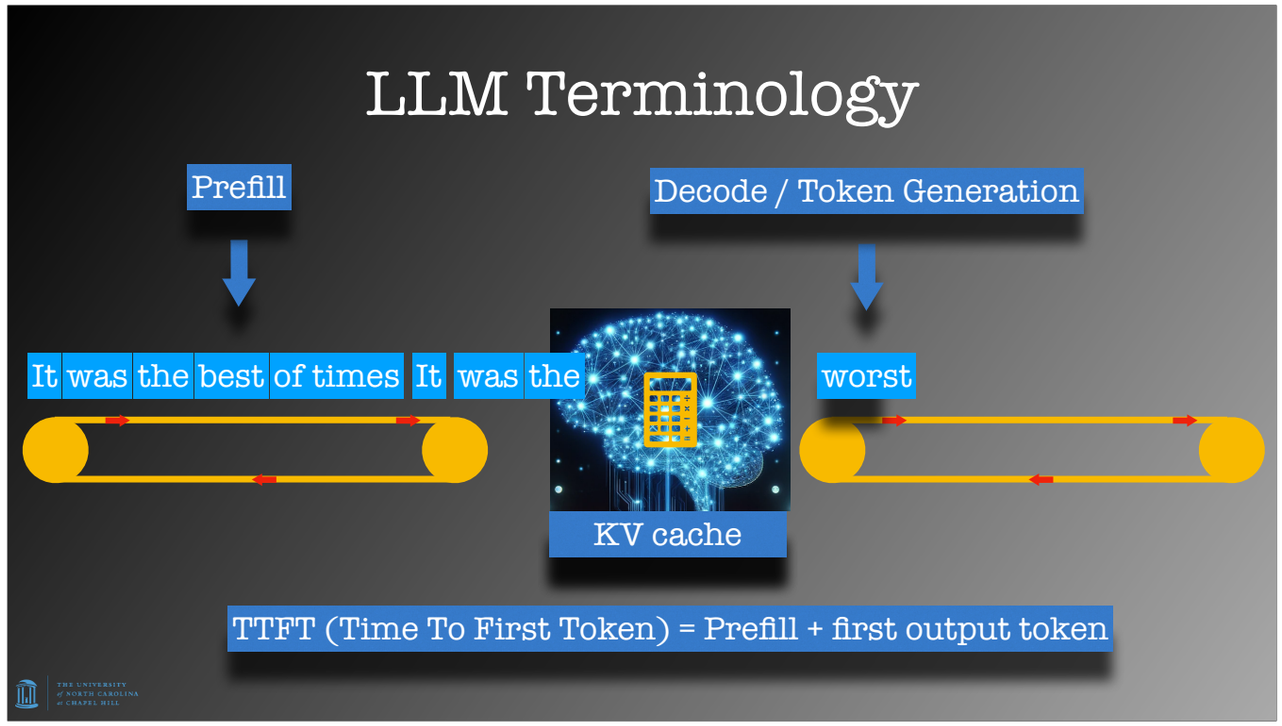
Model Names

Qwen3-235B-A22B-MLX-4bit

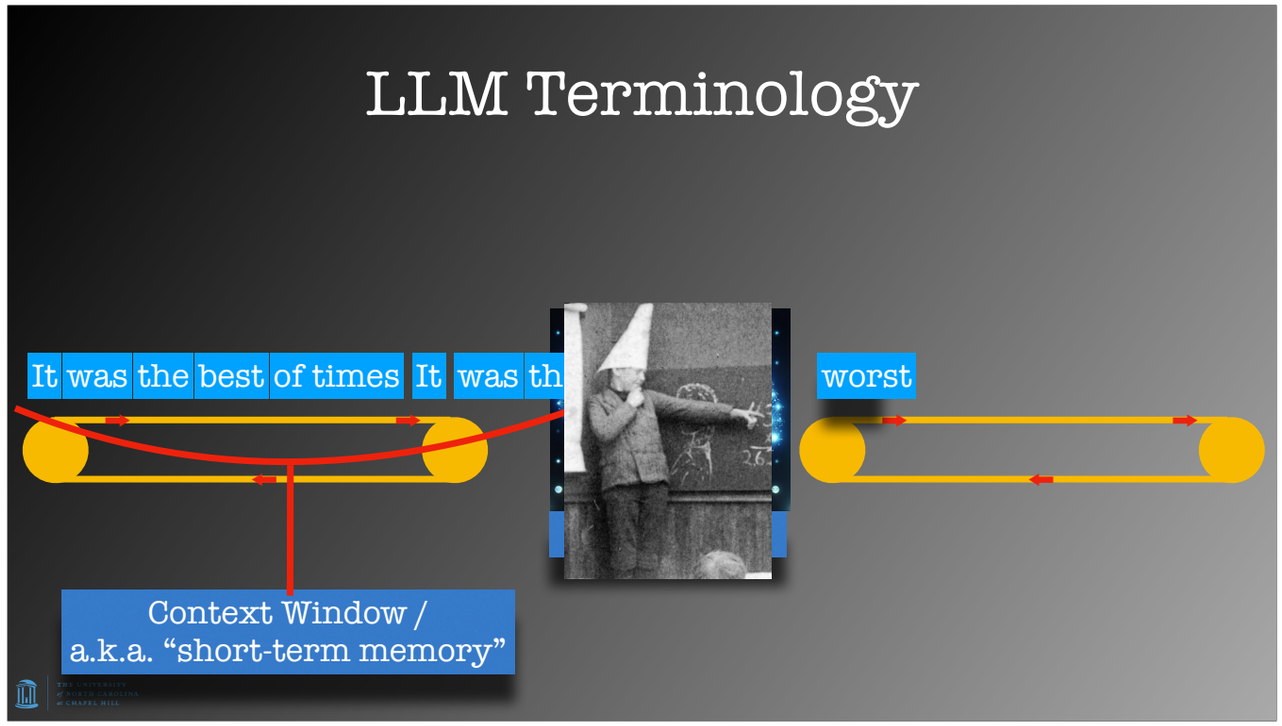
- Which Frontier AI lab did this model come from?
- How many total parameters does this model have?
- Is this a dense model or a MoE?
- If this is an MoE model, how many parameters activate for each forward pass?
- Is this model tuned for Apple Silicon?
- Is this model quantized, and if so, down to how many bits?

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

—SKIP FOR PRESENTATION—



—SKIP FOR PRESENTATION—
What do we call the fundamental piece of information (hint: “neuron”) in LLMs? A token.
What do we call the sequence of tokens we load into an LLM? A prompt.
What do we call the first stage where we load the prompt into the LLM? Prefill
And what do we call the place where we store the internal attention state of the prompt? KV cache.
What does “KV” stand for? Key-Value.
What do we call the second stage where the LLM does the math and spits out the next token? The decode/token generation stage.
And what do we call the total time it takes for the prefill + the first output token? TTFT (Time To First Token).



—SKIP FOR PRESENTATION—

What do we call the total amount of space that an LLM can hold as a prompt (i.e., the “conveyor belt”)? The context window.

What happens when the context window is full? The oldest tokens drop out in FIFO order.

Installing Local AI Tools

Story time… again…

Around May 2025, began using LLMs locally

1. **LM Studio** (1) — nice GUI; easiest to use; checks your hardware
2. **Ollama** (1) — CLI; offers a Docker-like experience
3. **llama.cpp** (1) — binary; 1st to support concurrent sessions
4. **LLamaBarn** (2) (now simply **Llama**) — simple menubar icon
5. back to **LM Studio**…

(1) macOS, Windows, Linux
(2) macOS only

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

—SKIP FOR PRESENTATION—
I began tinkering with AI/LLMs like most, trying out ChatGPT a few years ago.

Then around May 2025, I began running LLMs locally. Now I try to be “OS agnostic” and aim for programs which run on all 3 major OSes—MS Windows, macOS, and Linux—whenever possible. So first I downloaded **LM Studio**, as it was the simplest/easiest GUI to work with. It hooks into Hugging Face, recommends models, checks your hardware and lets you know whether the model can fully offload into vRAM or not, etc. The only thing is that LM Studio, while free, is NOT open-source like the others.

Then I tried **Ollama**, a command line tool which is akin to using **docker** commands. If you know Docker, you will feel right at home using Ollama.

Eventually I went to pure **llama.cpp**, “the OG of inference engines” written in… surprise surprise… C++. It offered concurrent session support, which at the time neither Ollama nor LM Studio did. And they had added a web UI. **llama.cpp** is just a binary, meaning the onus is on you to find/download the LLM model files, and to clean them up after the fact. At one point I had written scripts to streamline the process.

In fact, almost all of the other apps such as Ollama and LM Studio use llama.cpp under the hood. And they exist specifically to handle the things that llama.cpp did/does not do itself.

Then I found **LlamaBarn** (now simply **Llama**), a small macOS-only app which made my scripts unneeded. It is a simple program that sits in the macOS menubar, includes its own instance of llama.cpp, and offers a curated set of models.

Later I went back to LM Studio once they added concurrent session report. And since then they added MLX support and came out with another product, Bionic, which is like their coding agent/assistant.

The reason I explain all this is that just in the past year and a half, things have moved so quickly that I have changed inference engines 4-5x, along with changing my IDE and other aspects of my work. So just understand that things move pretty quickly in this space.

OpenAI-compatible APIs

Software/Tool	Default Port
LLama (macOS) formerly LlamaBarn	9931
llama.cpp	8080
LM Studio	1234
Ollama	11434
vLLM	8000

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

—SKIP FOR PRESENTATION—

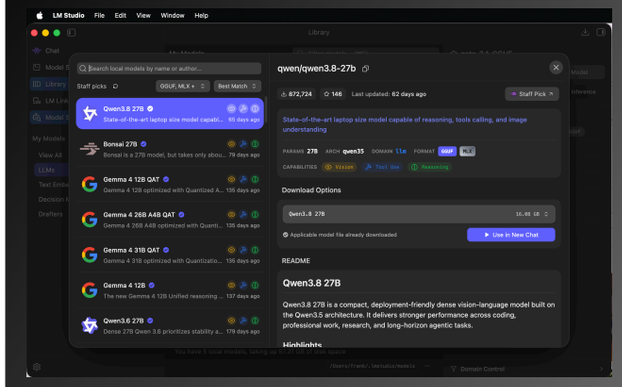
Just a quick note regarding inferencing engines. Almost all provide an OpenAI-compatible API. But each tends to listen on a different default port. You can, of course, change that if needed.

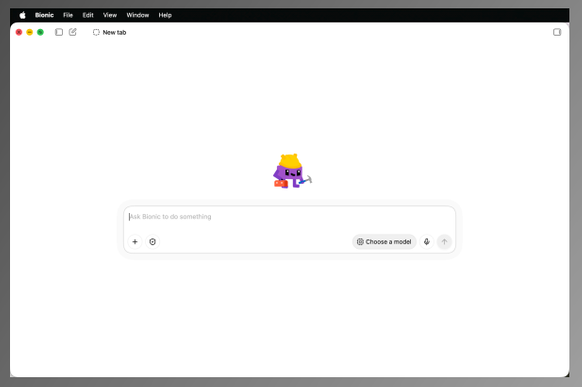
LM Studio / Bionic

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

So here we go. The first program I almost always recommend to people is LM Studio.

LM Studio / Bionic





Simply download & install: https://lmstudio.ai/download

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

LM Studio is both an inference engine and a chat harness. It offers more granular control over LLM settings. Think temperature, context window size, etc. LM Studio presents the models from HuggingFace in a nice UI where you can filter by recommended picks, file formats like GGUF and MLX, and more. For each model, it shows how large it is on disk, as well as whether that model can fully offload into vRAM on your hardware. It weighs in ~1.6GB fully installed.

Bionic is "LM Studio's agent for open models." It offers basic inference engine features like LM Studio, will use any models you have installed for LM Studio (i.e., they share the same directory for models), and offers a basic ability to download models from HuggingFace. But it offers more features as a harness, going beyond chat to doing more agentic coding work, etc. It weighs in ~2.2GB fully installed.

They do not so much as compete as complement, though their overlapping features can make folks wonder if they only need one or the other. The real question lies in what your needs are.

Ollama

Next we have Ollama for those who prefer working from the shell/terminal.

Ollama

As a CLI tool, **Ollama** gets installed as one might expect: from the command line/terminal.

Go to https://ollama.com/download and follow the instructions.

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Ollama works very similarly to the Docker CLI tool. If you are comfortable using Docker in the terminal, you will be right at home with Ollama.

OpenCode



OpenCode

OpenCode began as a terminal coding agent. And as such, you install it from the command line/terminal if using it this way. That said, they now offer not only a CLI version, but a desktop (GUI) version, IDE extensions, and more.

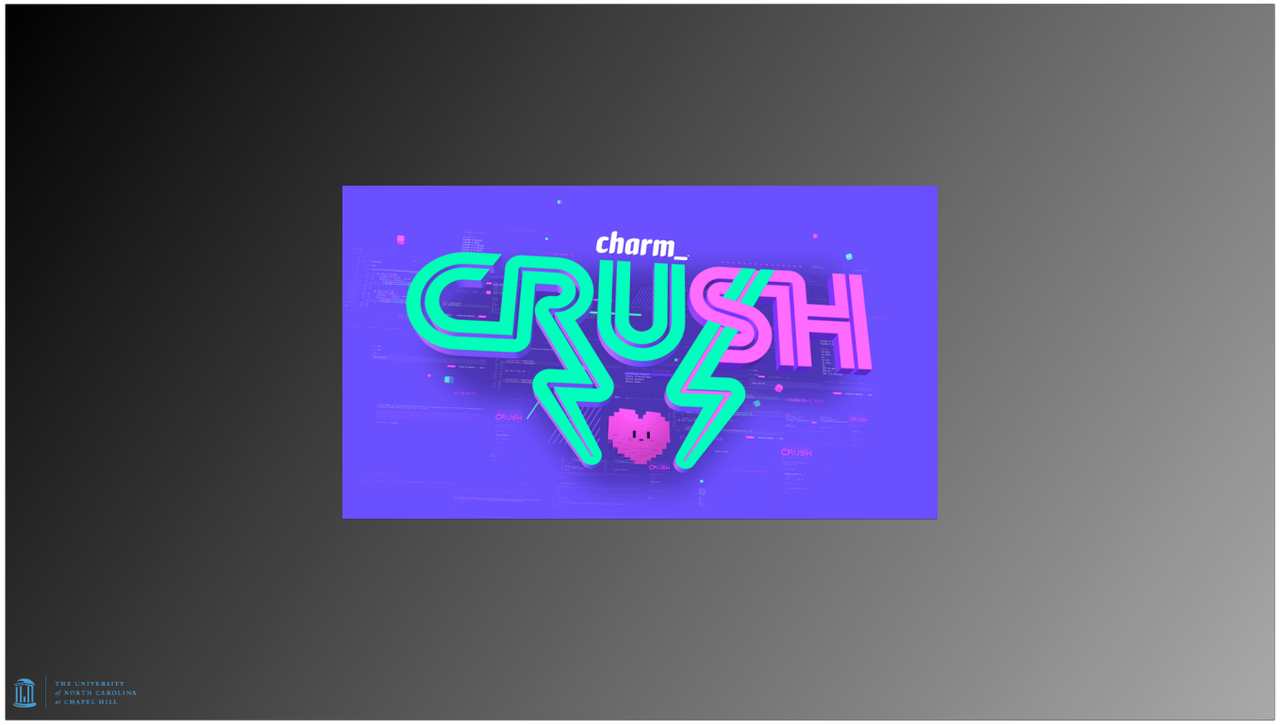
Visit https://opencode.ai/download and follow the instructions.

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

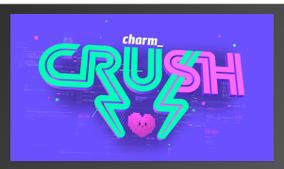
Actually, what today is called OpenCode is NOT the original OpenCode at all. The original OpenCode program was written in Go by a single developer. But there is a bit of Internet drama around this.

Two other developers convinced the original developer that they could bring more visibility to the project. Fast forward a bit, and the original developer was offered and took a job at Charm, who makes many of the TUI libraries that were used in the project. The other two developers forked the project, and later there was dispute over who owned the project name.

In the end, the original developer changed the name of his project from OpenCode to Crush. And the two other developers ended up taking what HAD been OpenCode and rewriting it entirely in TypeScript (the same languages used in Claude Code). So what today is called OpenCode is nothing like what started as OpenCode.



Which brings us to the original OpenCode, now renamed Crush.

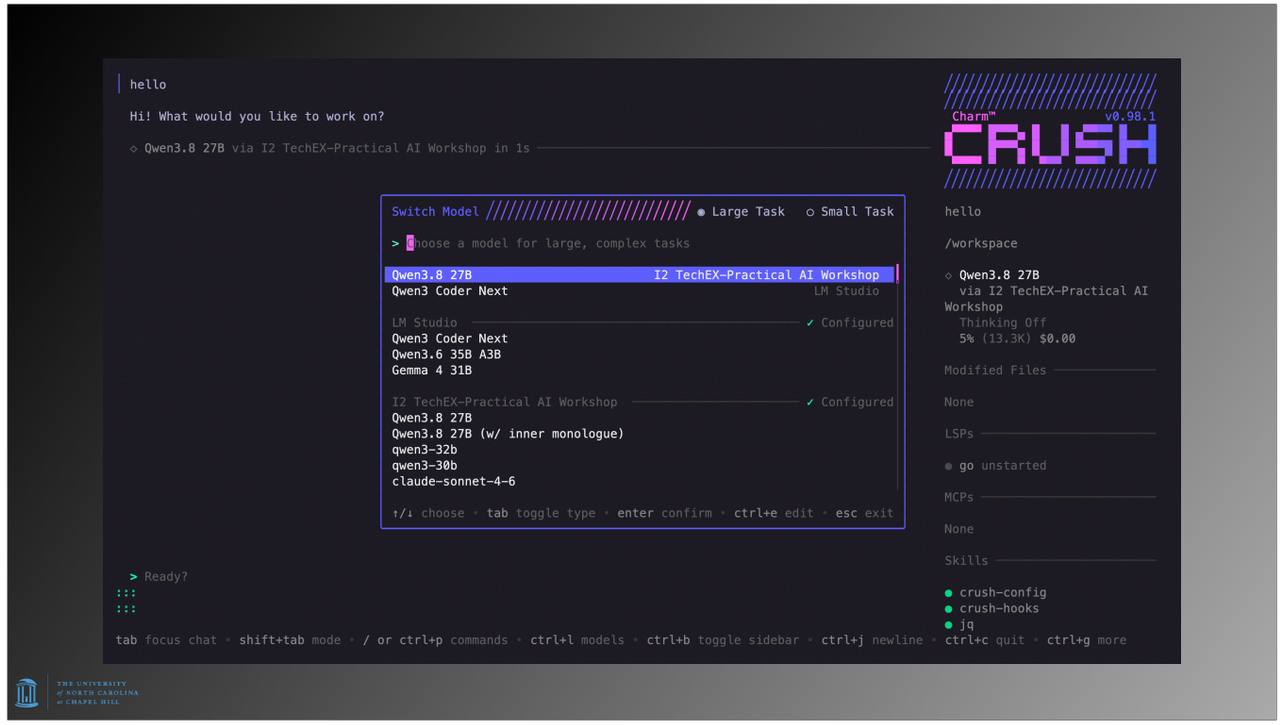


Crush is another CLI coding agent. Written in Go, it uses the Charm TUI libraries, providing a more polished feel.

They offer various ways to install it here: https://github.com/charmbracelet/crush

though I prefer to simply manually download the latest release from GitHub and decompress it to a directory within my PATH.

“The original OpenCode”, I personally find Crush refreshing to use. It is open source, written in Go, has a very polished TUI, and I just prefer it.



Here is an example screenshot of Crush. When you first enter, if you click the slash (‘/‘) key, it brings up the settings. If you then select “Switch Model”, you land on this dialog. If you look carefully, you will see that it shows “(*) Large Task ( ) Small Task” in the upper right. Looking across the bottom of the dialog, you see what you need to do to switch between configuring the model for large tasks vs. the model for small tasks.

This is a nice feature in Crush. You typically have help right in front of you, thus not requiring memorizing all the key combinations needs to perform tasks.

Example **crushrc**

```
### Global crushrc ####################

### GO LSP ############################
lsp add go \
  --command "gopls" \
  --env "GOTOOLCHAIN" "go1.27.2"

### LM Studio #########################
provider add lmstudio \
  --name "LM Studio" \
  --type lmstudio \
  --base-url "http://192.168.152.31:1234/v1/"
\
  --discover-models true

## Add specific configs for some models
model add lmstudio/qwen3.6-35b-a3b \
  --name "Qwen3.6 35B A3B" \
  --context-window 262144 \
  --default-max-tokens 20000
model add lmstudio/gemma-4-31b \
  --name "Gemma 4 31B" \
  --context-window 262144 \
  --default-max-tokens 20000
```

```
### 2026 I2 TechEX – Practical AI Workshop #######
provider add vLLM \
  --name "I2 TechEX-Practical AI Workshop" \
  --type openai-compat \
  --base-url “https://practicalai.ns.internet2.edu/llm” \
  --api-key "$vLLM_KEY" \
  --discover-models true

## Add specific configs for some models
model add vLLM/qwen3.8-27b \
  --name "Qwen3.8 27B" \
  --context-window 262144 \
  --default-max-tokens 20000 \
  --can-reason true
model add vLLM/qwen3.8-27b-thinking \
  --name "Qwen3.8 27B (w/ inner monologue)" \
  --context-window 262144 \
  --default-max-tokens 20000 \
  --can-reason true

## large = coding model, small = titles/summaries.
model large vLLM/qwen3.8-27b
model small vLLM/qwen3.8-27b
```


“Crush is configured with Bash via a set of Crush-specific builtin commands.
…
Since it’s Bash, so you can use logic, source other files, and so on. It’s really handy.
…
Why Bash?

Two reasons:
1. Crush ships with a first-class Bash interpreter, so we get the logic for free.
2. Ultimately, Crush needs to be able to configure itself, and command-based config allows both users and the agent to use the same tools.”

From https://github.com/charmbracelet/crush/tree/main/docs/config (See page for more details on Crush’s crushrc file settings.)

As you can see in the example, you can load in LSPs (Language Server Protocols) for any programming languages you use. And you can add different LLM providers, whether local or hosted elsewhere.

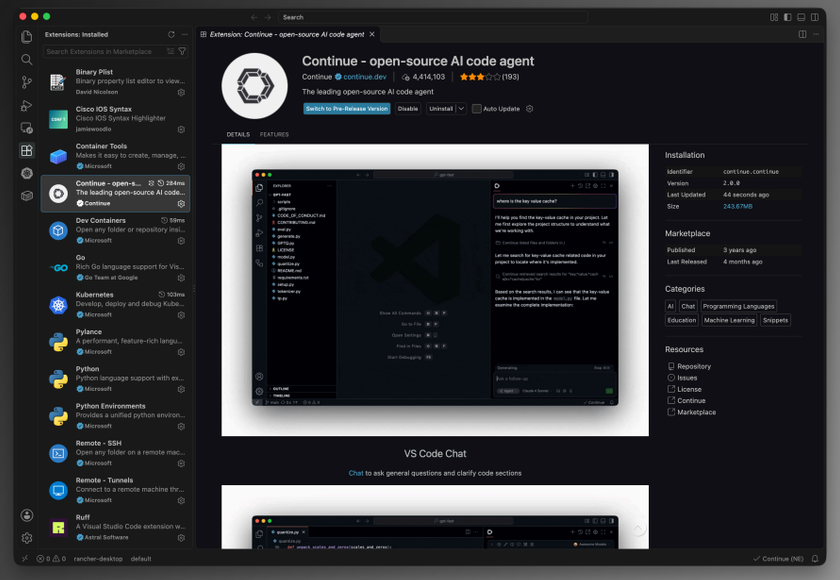
Regarding the I2 vLLM setup, note a few things:
- **$vLLM_KEY** is an environment variable, meaning the config doesn’t have to have hard-coded secrets. You could use your various password managers to populate such keys as well.
- It was discovered that **openai-compat** models arrive with an unknown (0) context window, which hides Crush's context meter and disables auto-summarize. Both GPUs serve --max-model-len 262144 (verified 2026-10-16), so we simply declare the model context windows as such.

Visual Studio Code (VS Code)

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

For IDEs, no one can deny that Microsoft's Visual Studio Code (VS Code) is one of the most popular out there.

VS Code + Continue.dev



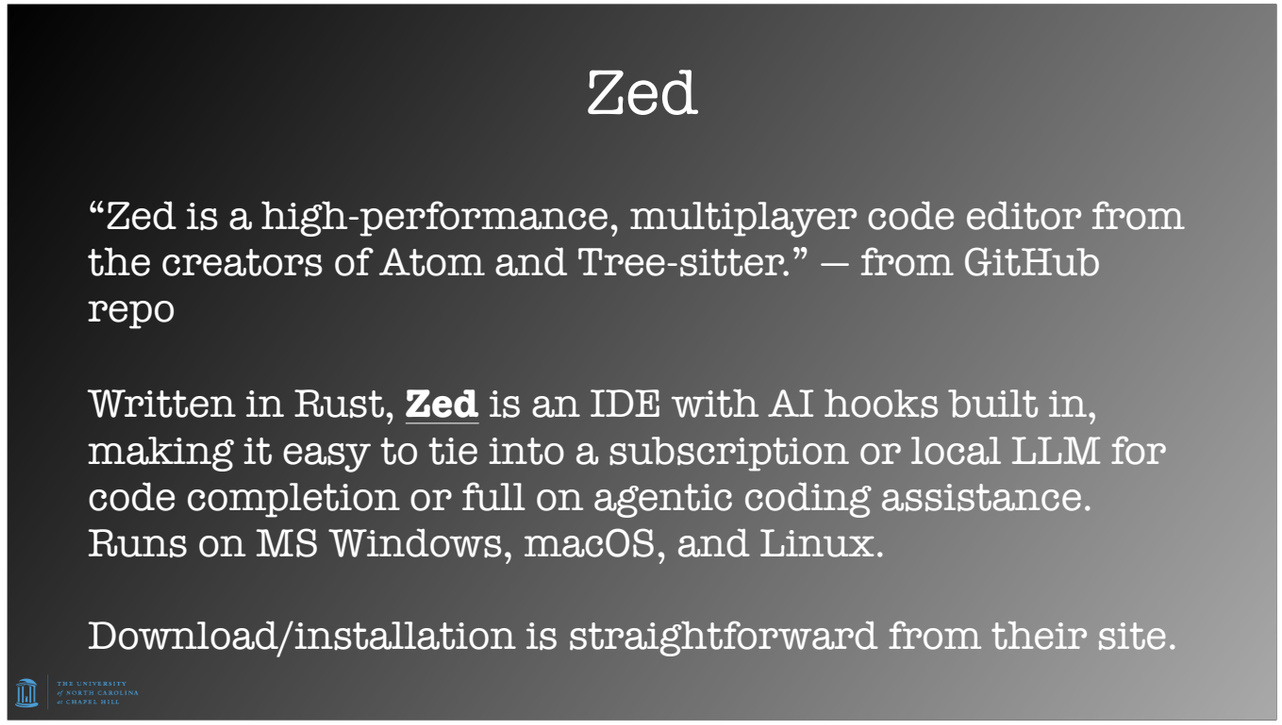
THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

And coupled with an extension like Continue.dev, you can use VS Code connected to almost any LLM provider you like and have an experience similar to using Cursor or other IDEs built with AI/LLMs in mind.

Zed

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

For those looking for an IDE who do not want to use VS Code, my preferred one these days is Zed.



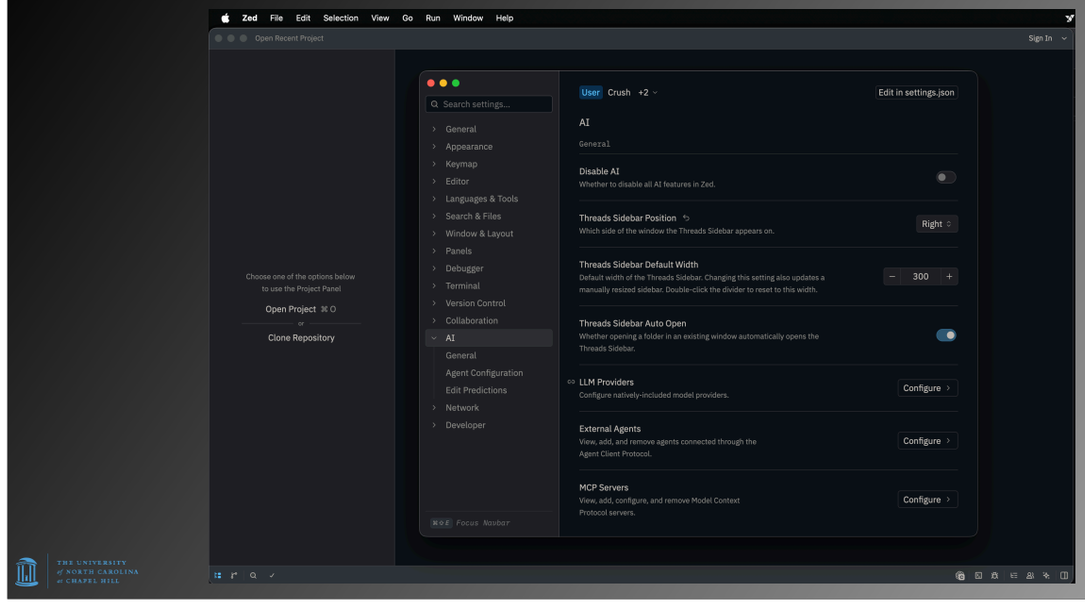
In the past year there have been several stories of supply chain attacks against programming language repos. Among the worst hit has been **npm**, the Node Package Manager, with repeated attacks from versions of the Shai-Hulud worm. Noting that npm packages offer pre- and post-script installers which were exploited in these attacks, the simple act of installing a package could infect your computer.

Noting that VSCode is written in TypeScript and relies extensively on npm, notably for all the extensions in their ecosystem, it occurred to me that the simple act of VSCode updating one of my extensions, where one of those extensions had been compromised, could infect my machine without my even realizing it. This set me down a path to finding an alternative IDE.

The end result was **Zed**. Zed was written in Rust by the same developers who originally created the GitHub Atom IDE (which itself was where the Electron framework got its start). Atom was the IDE I used before moving to VS Code many years ago because I found Atom to be just too slow (and they both used the same “LEGO pieces” of the Electron framework and JavaScript). But the point is these developers know what they’re doing. And this new editor, besides being blazingly fast, has many features built in that you would need to install various extensions for in VSCode and others.

In fact, currently I have only have 3 extensions installed: HTML, TOML, and XML. For everything else, I just use what is built into Zed. And after giving myself a few weeks to get used to small muscle memory adjustments, I found no reason to go back. But YMMV.

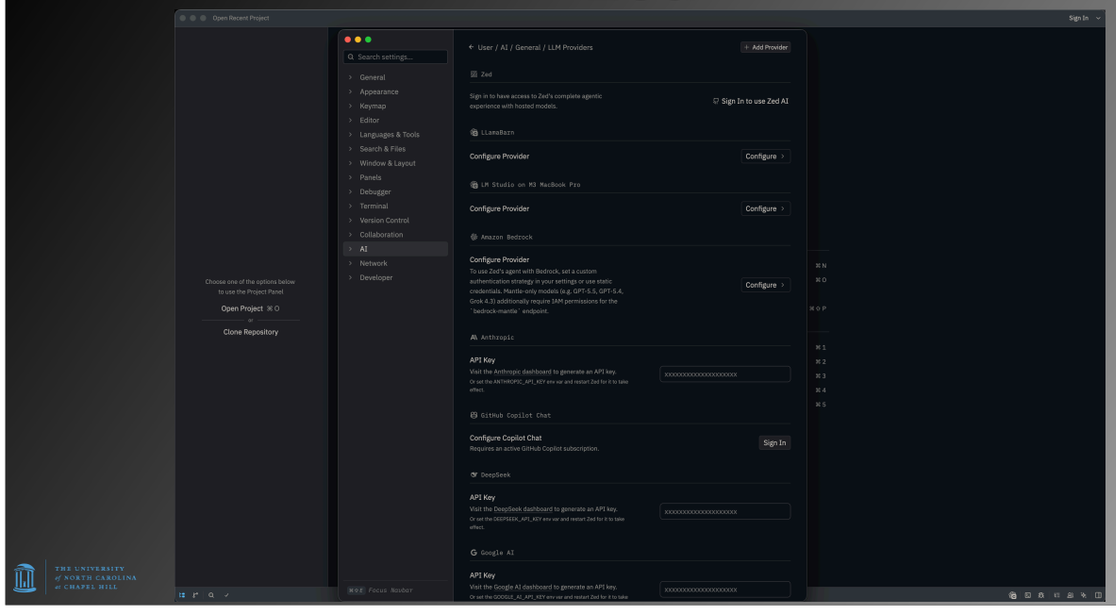
Zed



When you first install and run Zed, it walks you through a setup process where you can even designate that you want to use things like VSCode key bindings, for example.

Once running, if you go into the settings, you will see that as part of the basic IDE, they have an entire section devoted to AI tooling. This includes everything from code completion and LLM providers to using ACP (Agent Client Protocol) with external agents or MCP (Model Context Protocol) servers.

Zed



Clicking the LLM Providers “Configure” button, we are taken to a section where you can see they have options for almost every known LLM provider. This includes

- **Zed**’s own subscription offering (i.e., pay subscription, get code completion/etc.)
- All the frontier labs such as **Anthropic** (API key), **DeepSeek**, **Google** AI (API key), **Mistral**, **OpenAI**, **ChatGPT subscription**, **xAI**, **SuperGrok**, etc.
- **GitHub Copilot subscription**
- **llama.cpp**
- **LM Studio**
- **Ollama**
- **OpenCode**
- **OpenRouter**
- **Vercel AI Gateway**

If you cannot run an LLM locally…

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

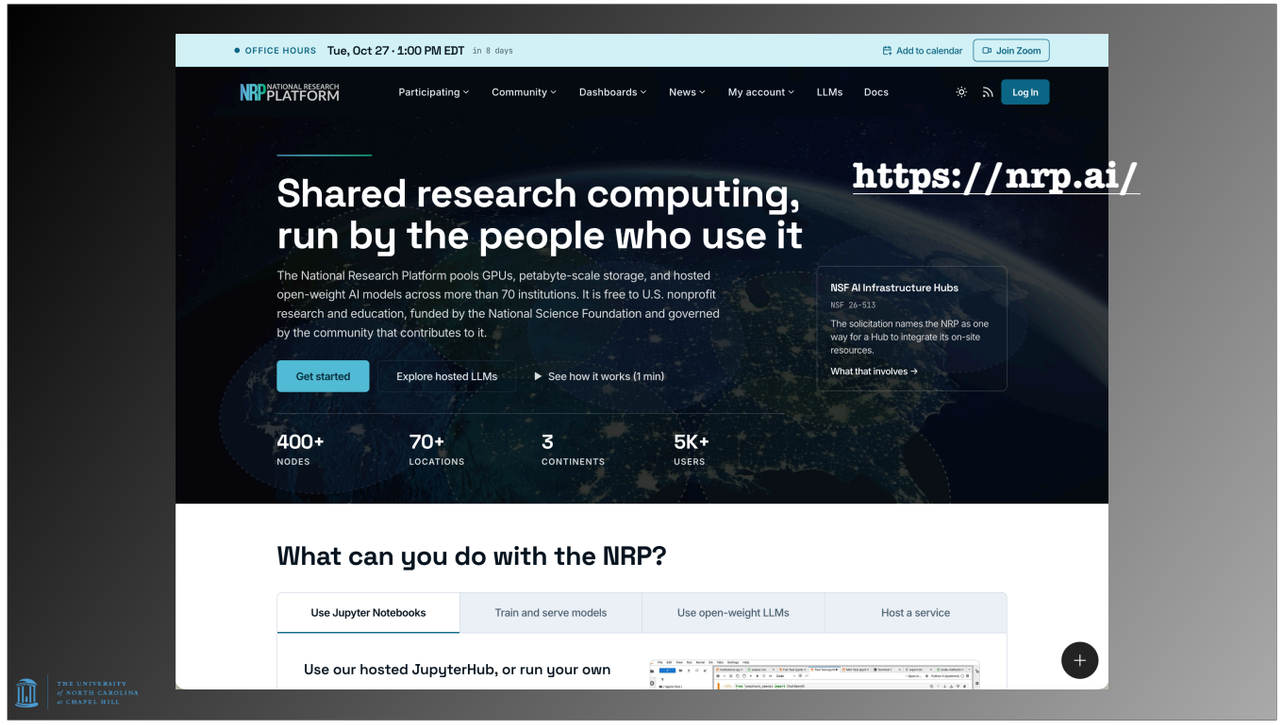
If you do not have the GPU capacity to run the LLMs you want locally, there are still options available. The following are just a few possibilities.

Your Home Institutions

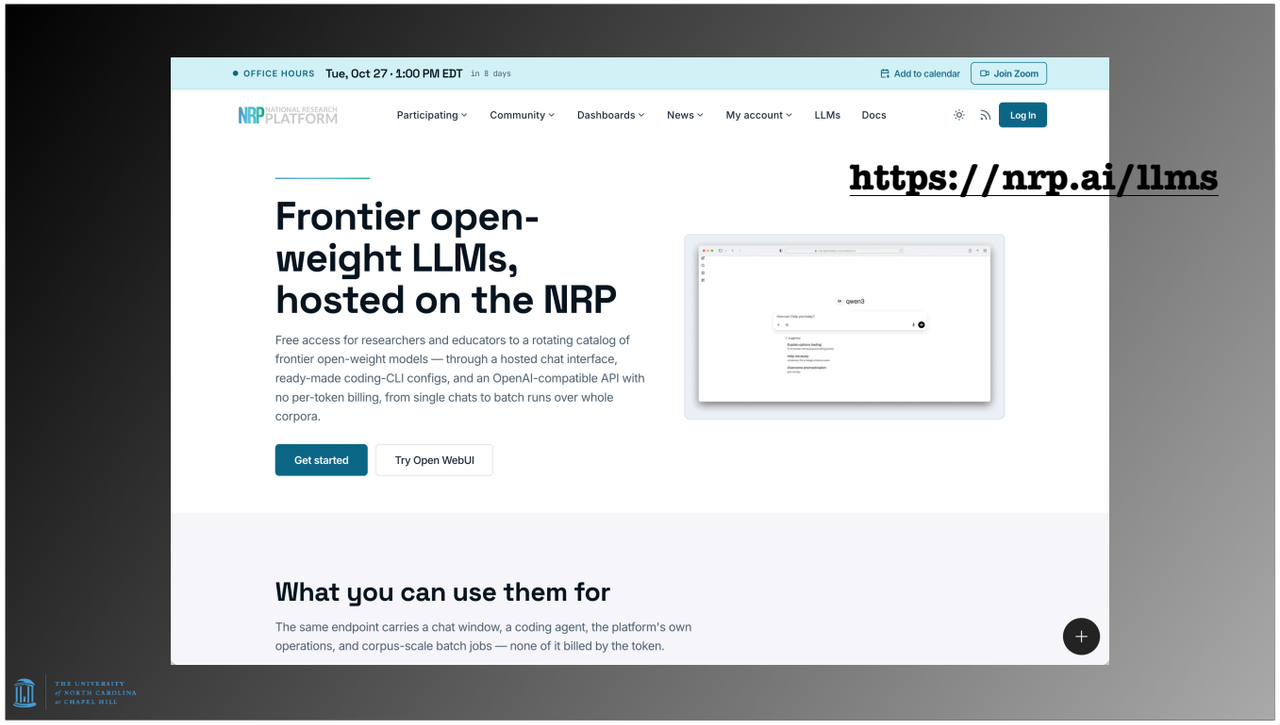
Many universities have signed up with one or more of the the major AI companies. While this does not provide any of the benefits of offline LLM use, most universities have a form of “enterprise license” with those companies where the latter “pinky promise” not to use/train on the data.

However, many research institutions have their own GPU clusters. And it may well be possible for you to get access to one of those. That at least may bring the LLM on-prem, where you have more control over things.

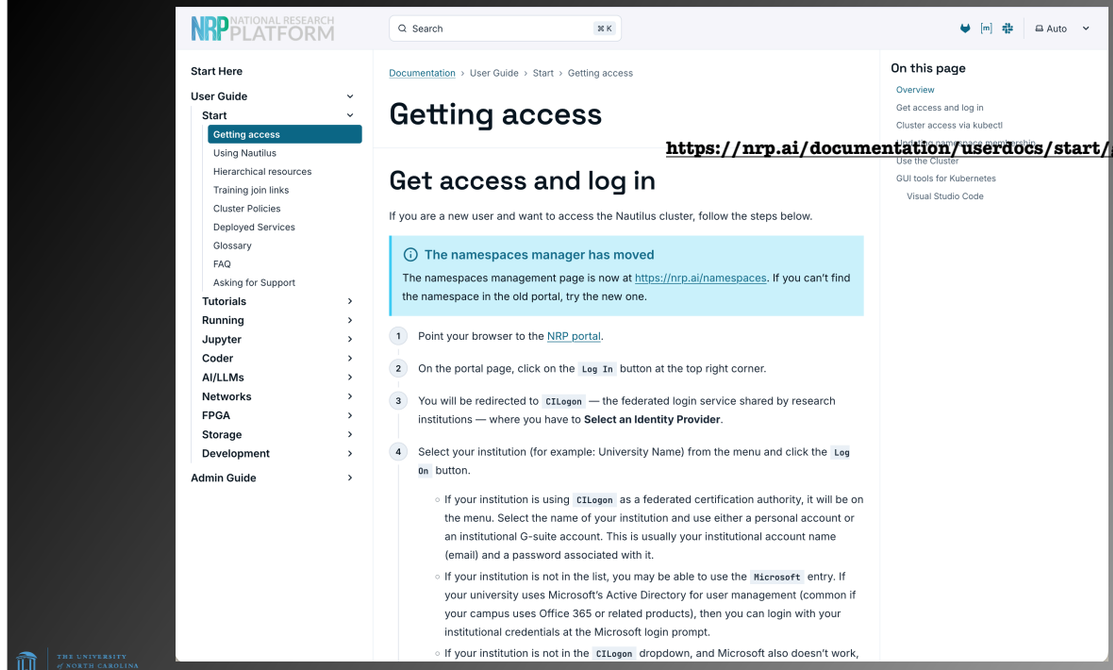
National Research Platform (NRP)



The National Research Platform is as the site says. Funded by the NSF, if the work you're doing fits in with what they list, be sure to tap into this. No, it's not an on-prem, local LLM setup. But this is compute hosted by folks whose goals/motivations are to advance research, not make a profit.



They have a page dedicated to hosting frontier, open-weight LLMs on their platform.

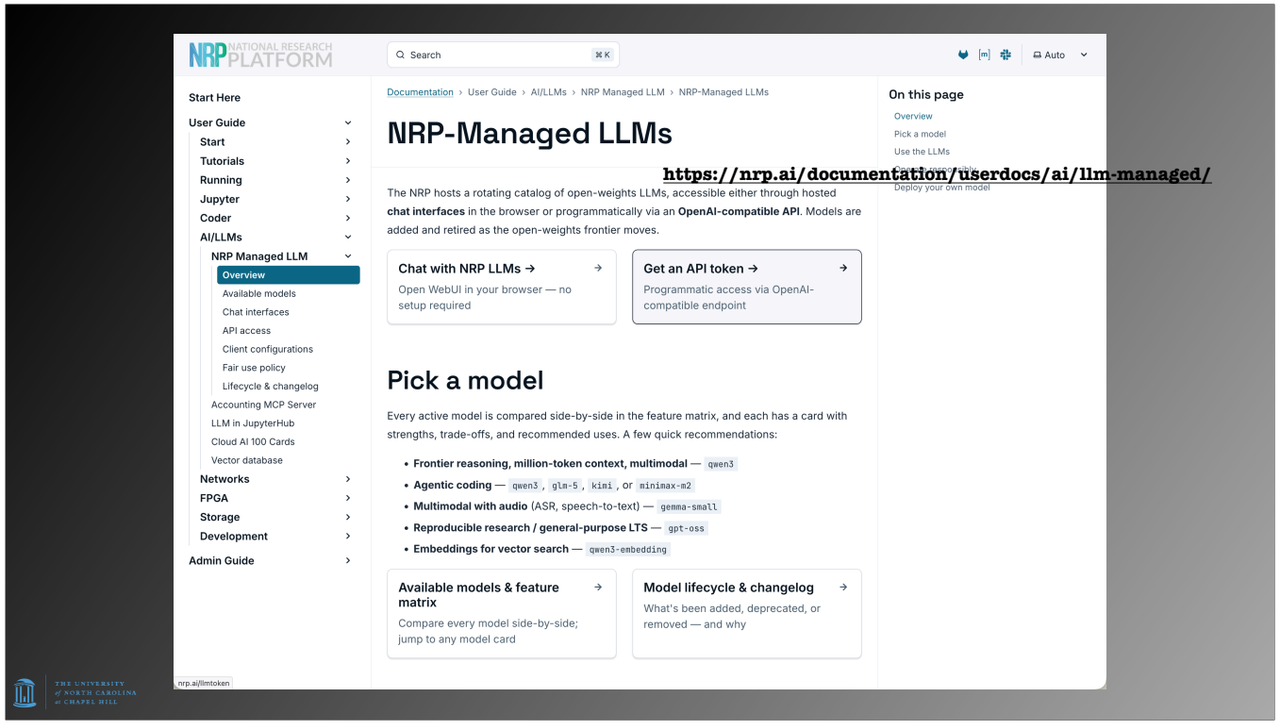


THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

—SKIP FOR PRESENTATION—
They also have robust documentation. Begin by getting access to the NRP itself by following their instructions.

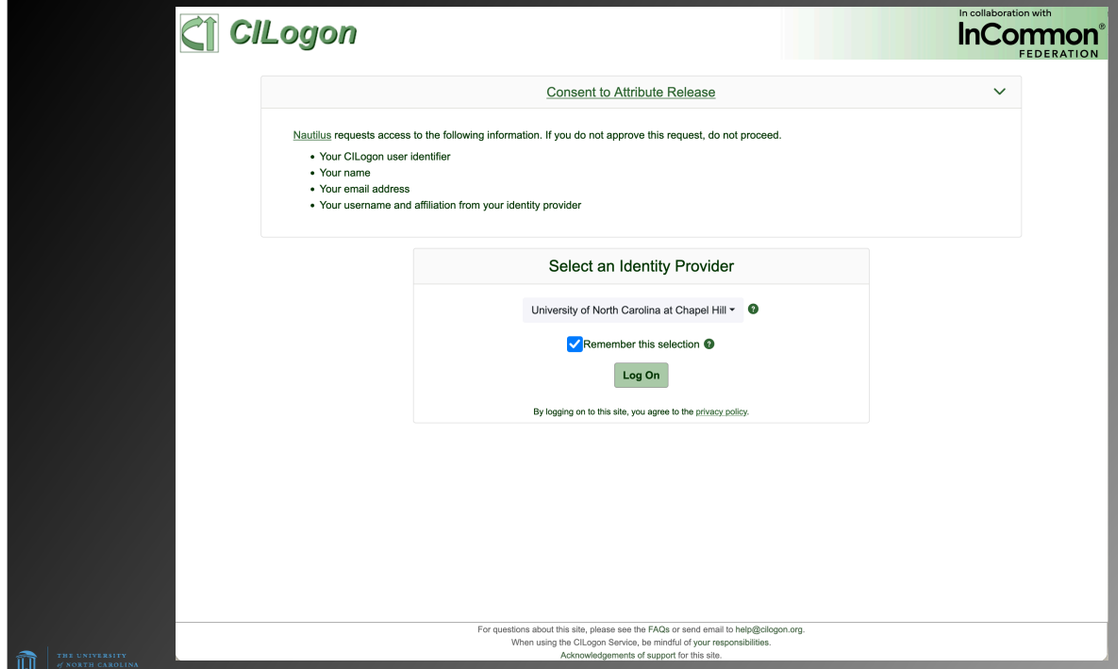
I highly recommend joining their Slack workspace, as the folks there are very supportive. I had a response within hours of trying to setup an account.

I won't lie. Once you have an account created, the process by which you go about setting up an API token is not the most obvious. But then I have not yet gone through the tutorials as linked later here, which I am sure would help.



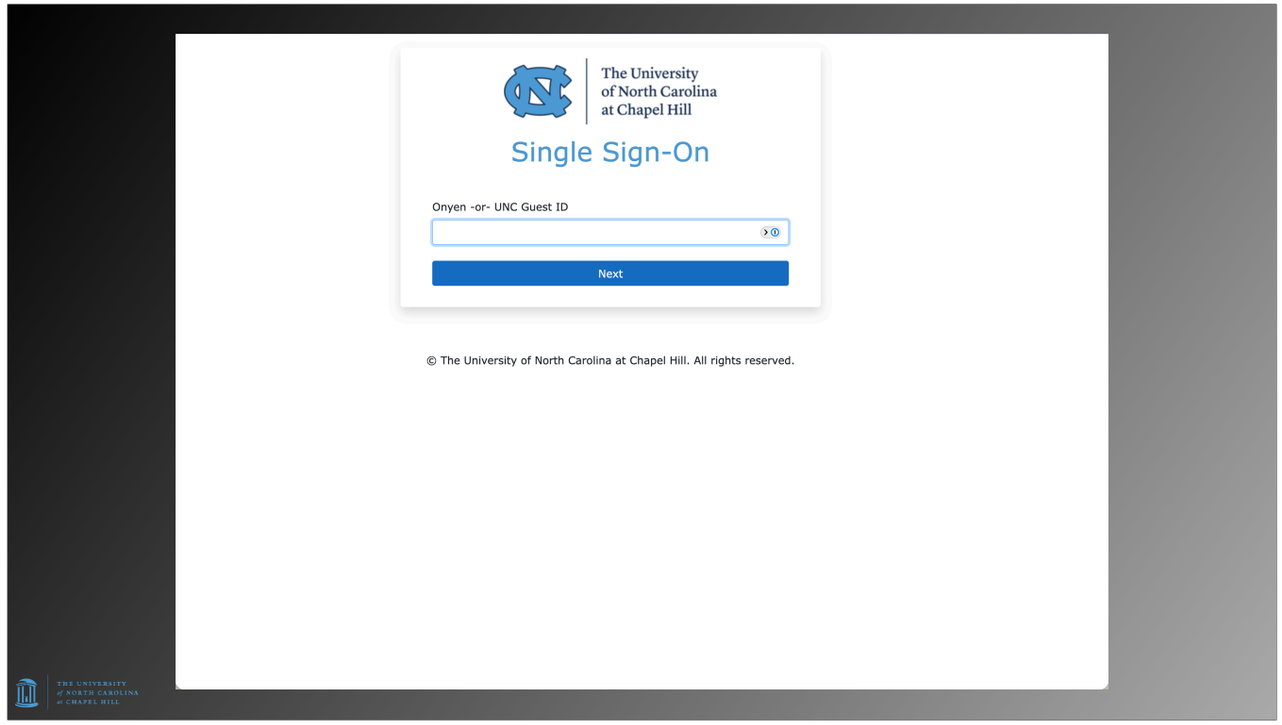
—SKIP FOR PRESENTATION—

And here is the beginning of the section of documentation specific to LLMs.

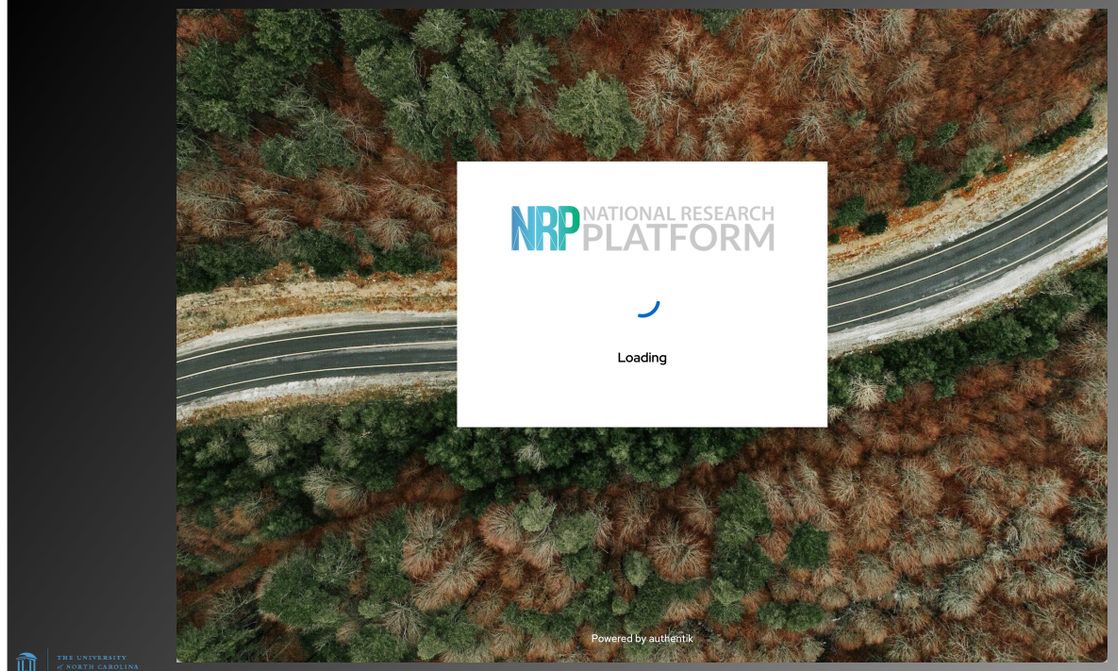


—SKIP FOR PRESENTATION—

When you log into the NRP, you will be taken to a page like this, where, assuming your institution is part of InCommon, you start typing in your institution's name. Once selected, you can have it remember this for future logins. Then you click the "Log On" button, which takes you to…



—SKIP FOR PRESENTATION—
your institution’s SSO page.



—SKIP FOR PRESENTATION—

Be patient here, as it can take a bit to login. Once completed, you will be back on the home page, only now you should see in the upper right corner that you are logged in.

National Research Platform (NRP) Resources

- Main site: **https://nrp.ai**
- Information specific to running **Frontier** open-weight LLMs, hosted on the NRP: **https://nrp.ai/llms**
- NRP-Managed LLMs | NRP Documentation

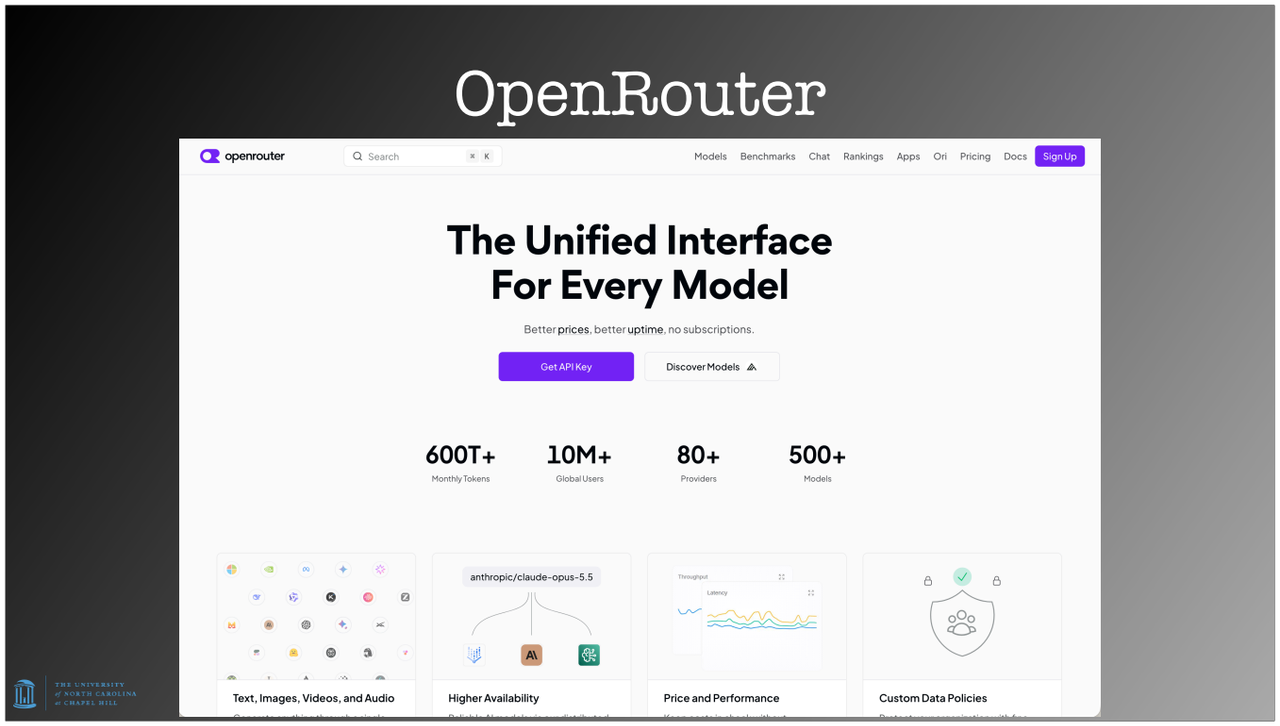
Training links provided by Daniel Diaz (UCSD) in the NRP Slack on 2026-09-30:

- Home — CLARIPHY — AI agents and Large Language Models for Scientific Discovery
- Home — LPC HATS -- NRP as a Resource for LLMs

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Here are some of the main links involved for the NRP.

I would like to thank Daniel Diaz of UCSD for providing the training links via the NRP Slack.



Another option is OpenRouter. Probably the most popular AI gateway out there, you can sign up with OpenRouter for free. OpenRouter offers some 25 free models (with token limits) that lets you get a taste for how they work. Then it's a pay-as-you-go setup for access to models across the spectrum. Interestingly enough, one of the free models recently was Qwen3.8 27B.

There are a lot of nuances to using an AI gateway which we can't get into right now. But used properly, it can be a cheaper option than many of the other API setups.

Thank You

https://frank.seesink.com/presentations/Internet2TechEx-Fall2026/

Frank Seesink
frank@seesink.com
frank@unc.edu

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Bonus Material

Additional Resources

- Proton Lumo AI - For those looking for a confidential AI chat assistant
-