Gmail **David Baek <davidbaek92@gmail.com>**

**Scaling ChatGPT: Five Real-World Engineering Challenges**

**The Pragmatic Engineer** <pragmaticengineer_at_substack.com_davidbaek92@duck.com> Tue, Feb 20, 2024 at 9:15 AM
Reply-To: The Pragmatic Engineer
<reply+2cgnye&ytrc6&&0beb62ba74a99fdd3a383e0134ce902286e5469711dfe38a45bd3fa24697856a_at_mg1.substack.com_davidbaek92@duck.com>
To: davidbaek92@duck.com

DuckDuckGo removed trackers from Mailgun, and one other. More Report Spam

View in browser

👋*Hi, this is Gergely with a free issue of the Pragmatic Engineer Newsletter. In every issue, I cover challenges at Big Tech and startups through the lens of engineering managers and senior engineers.*

# Scaling ChatGPT: Five Real-World Engineering Challenges

Just one year after its launch, ChatGPT had more than 100M weekly users. In order to meet this explosive demand, the team at OpenAI had to overcome several scaling challenges. An exclusive deepdive.

GERGELY OROSZ
FEB 20

READ IN APP ↗

*Before we start, a request: I'm researching what it's like to work with GenZ software developers, and what they think of the "older" generation. Please help by sharing your observations here. This is all uncharted territory – and should be an interesting one, and helpful for all of us!*

*'Real-world engineering challenges' is a series in which I interpret interesting software engineering or engineering management case studies from tech companies.*

In November 2022, ChatGTP took the world by storm, in what it's safe to say was the biggest "wow!" moment yet for applied Artificial Intelligence (AI,) and the catalyst for the continuing surge of investment in the technology, today. CEO Sam Altman announced the product's unprecedented growth in November 2023.

We previously covered how OpenAI and the ChatGPT team ships so quickly. In that article, we look into the setup of the organization, the Applied team's integration with Research, a strategy of uncoupled and incremental releases, and more.

But what about the very real scaling challenges that come with explosive growth? To learn more, I've again turned to Evan Morikawa, who led the Applied engineering team as ChatGPT launched and scaled.

Today, Evan reveals the engineering challenges and how the team tackled them. We cover:

1. A refresher on OpenAI, and an introduction to Evan
2. Importance of GPUs
3. How does ChatGPT4 work?
4. Five scaling challenges:
   - KV Cache & GPU RAM
   - Optimizing batch size
   - Finding the right metrics to measure
   - Finding GPUs wherever they are
   - Inability to autoscale
5. Lessons learned

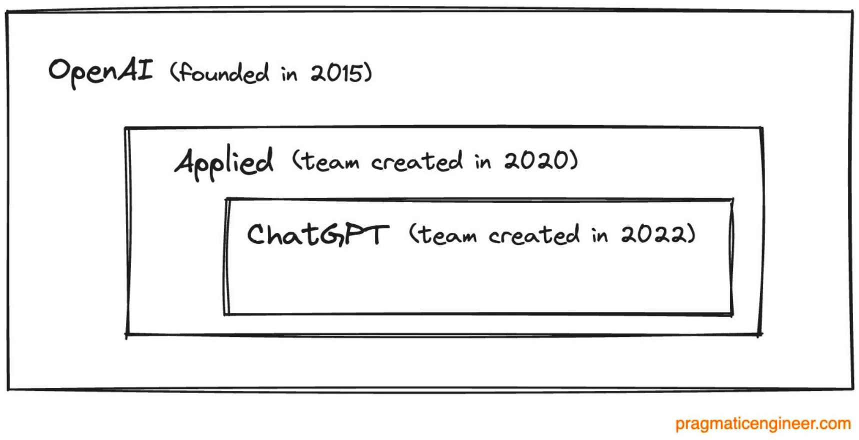
With that, it's over to Evan.

## 1. A refresher on OpenAI, and on Evan

*How did you join OpenAI, and end up heading the Applied engineering group – which also builds ChatGPT?*

OpenAI is the creator of ChatGPT, which is just one of the business' products. Other shipped things include DALL·E 3 (image generation,) GPT-4 (an advanced model,) and the OpenAI API which developers and companies use to integrate AI into their processes. ChatGPT and the API each expose several classes of model: GPT-3, GPT-3.5, and GPT-4.

The engineering, product, and design organization that makes and scales these products is called "Applied," and was founded in 2020 when GPT-3 was released. It's broadly chartered with safely bringing OpenAI's research to the world. OpenAI itself was founded in 2015 and at the company's core today is still a research lab with the goal of creating a safe and aligned artificial general intelligence (AGI.)


OpenAI (founded in 2015)
Applied (team created in 2020)
ChatGPT (team created in 2022)
pragmaticengineer.com

The Applied group and ChatGPT within OpenAI

I joined OpenAI in October 2020 when Applied was brand new. I do *not* have a PhD in Machine Learning, and was excited by the idea of building APIs and engineering teams. I managed our entire Applied Engineering org from its earliest days through the launch and scaling of ChatGPT. *We cover more on Evan's story in Inside OpenAI: how does ChatGPT ship so quickly?*

## 2. How does ChatGPT work? A refresher.

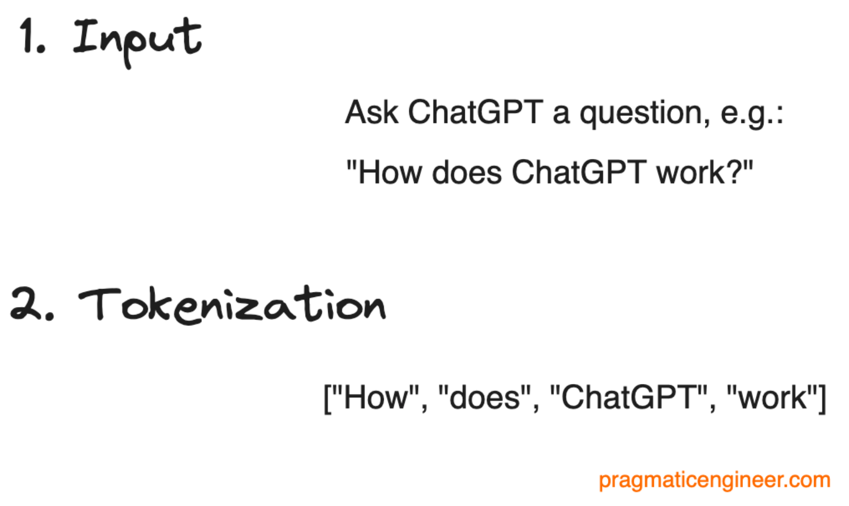
*For those of us who have not spent the past few years building ChatGPT from the ground up, how does it work?*

In order to introduce some scaling topics, let me give a quick introduction about how these AI models work. If you're familiar with the basics, skip to section 3.

When you ask ChatGPT a question, several steps happen:

1. **Input.** We take your text from the text input.
2. **Tokenization.** We chunk it into tokens. A token roughly maps to a couple of unicode characters. You can think of it as a word.
3. **Create embeddings.** We turn each token into a vector of numbers. These are called embeddings.
4. **Multiply embeddings by model weights.** We then multiply these embeddings by hundreds of billions of model weights.
5. **Sample a prediction.** At the end of this multiplication, the vector of numbers represents the probability of the next most likely token. That next most likely token are the next few characters that spit out of ChatGPT.

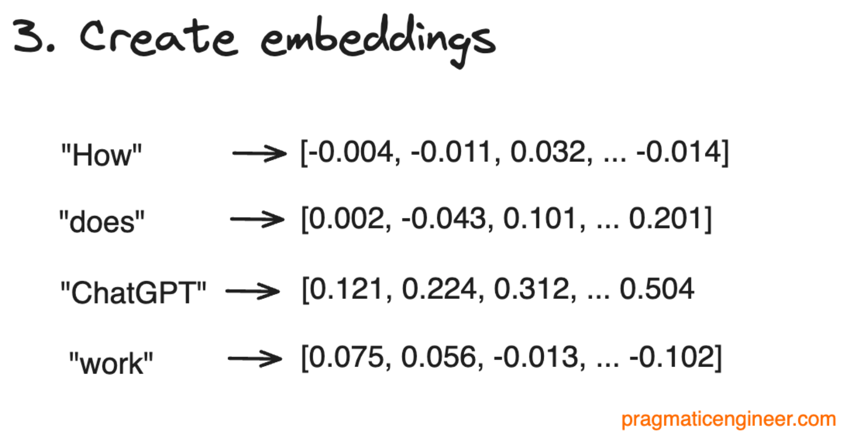
Let's visualize these steps. The first two are straightforward:



Steps 1 and 2 of *what happens when you ask ChatGPT a question*

Note that tokenization doesn't necessarily mean splitting text into words; tokens can be subsets of words as well.

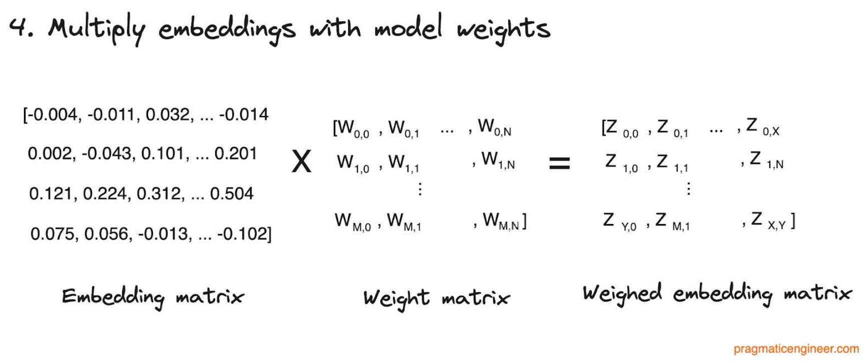
**Embeddings** are at the heart of large language models (LLM), and we create them from tokens in the next step:


3. Create embeddings
"How" → [-0.004, -0.011, 0.032, ... -0.014]
"does" → [0.002, -0.043, 0.101, ... 0.201]
"ChatGPT" → [0.121, 0.224, 0.312, ... 0.504
"work" → [0.075, 0.056, -0.013, ... -0.102]
pragmaticengineer.com

Step 3 of *what happens when you ask ChatGPT a question. Embeddings represent tokens as vectors. The values in the above embedding are examples*

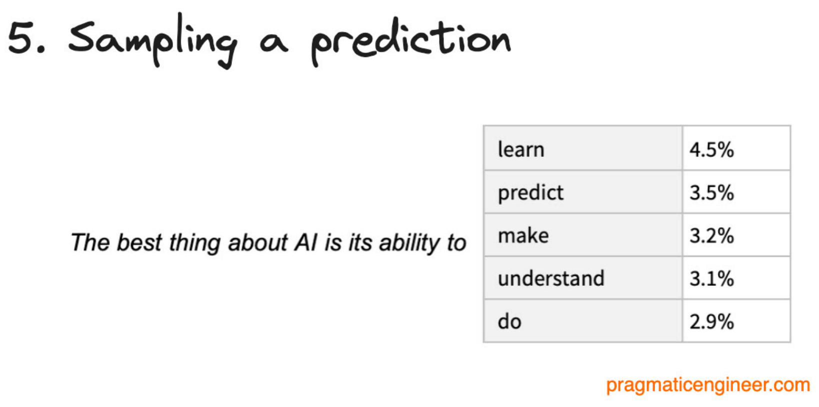
An embedding is a multi-dimensional representation of a token. We explicitly train some of our models to explicitly allow the capture of semantic meanings and relationships between words or phrases. For example, the embedding for "dog" and "puppy" are closer together in several dimensions than "dog" and "computer" are. These multi-dimensional embeddings help machines understand human language more efficiently.

**Model weights** are used to calculate a weighted embedding matrix, which is used to predict the next likely token. For this step, we need to use OpenAI's weight matrix, which consists of hundreds of billions of weights, and multiply it by a matrix we construct from the embeddings. This is a compute-intensive multiplication.



Step 4 of what happens when you ask ChatGPT a question. The weight matrix contains hundreds of billions of model weights

**Sampling a prediction** is done after we do billions of multiplications. The final vector represents the probability of the next most likely token. Sampling is when we choose the next most likely token and send it back to the user. Each word that spits out of ChatGPT is this same process repeated over and over again many times per second.



| | |
|---|---|
| learn | 4.5% |
| predict | 3.5% |
| make | 3.2% |
| understand | 3.1% |
| do | 2.9% |

*Step 5. We end up with the probability of the next most likely token (roughly a word). We* ***sample*** *the next most probable word, based on pre-trained data, the prompt, and the text generated so far. Image source: What is ChatGPT doing and why does it work? by Stehen Wolfram*

### Pretraining and inference

How do we generate this complex set of model weights, whose values encode most of human knowledge? We do it through a process called **pretraining**. The goal is to build a model that can predict the next token (which you can think of as a word), for all words on the internet.

During pretraining, the weights are gradually updated via **gradient descent**, which is is a mathematical optimization method. An analogy of gradient descent is a hiker stuck up a mountain, who's trying to get down. However, they don't have a full view of the mountain due to heavy fog which limits their view to a small area around them. Gradient descent would mean to look at the steepness of the incline from the hiker's current position, and proceed in the direction of the steepest descent. We can assume steepness is not obvious from simple observation, but luckily, this hiker has an instrument to measure steepness. However, it takes time to do a single measurement and they want to get down before sunset. So, this hiker needs to decide how frequently to stop and measure the steepness, so they still can get down before sunset.

Once we have our model we can run **inference** on it, which is when we prompt the model with text. For example, the prompt could be: "write a guest post for the Pragmatic Engineer." This prompt then asks the model to **predict the next most likely token (word)**. It makes this prediction based on past input, and it happens repeatedly, token by token, word by word, until it spits out your post!

*Note from Gergely: just to say, not this post or any other I publish are made by ChatGPT! But it's a fun experiment to learn how the tech works. Here's what ChatGPT generates in response to this command, by always generating the next most likely token (word.) For more detail on this prediction technique, check out What is ChatGPT doing and why does it work? by Stehen Wolfram.*

### Scalability challenge from self-attention

Under the hood, we use the Transformer architecture, a key characteristic of which is that each token is aware of *every other token*. This approach is known as self-attention. A consequence is that the longer your text is – or context – the more math is needed.

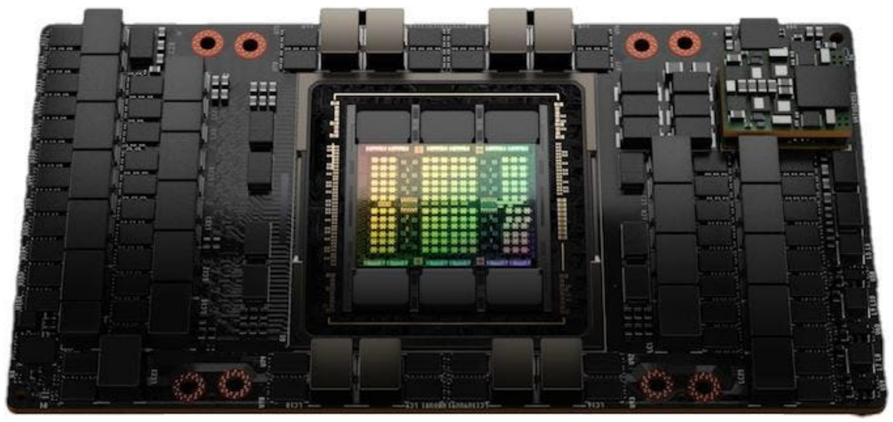
Unfortunately, self attention scales quadratically. If you want the model to predict the 100th token, it needs to do about 10,000 operations. If you want the model to predict the 1,000th token, it needs to do about 1 million operations.

At first, this sounds like bad news. However, there are clever workarounds we can use to circumvent the quadratic scale problem. Before we get into how we solve it, we need to talk about the infrastructure powering ChatGPT.

## 3. Importance of GPUs

GPUs (Graphics Processing Units) are the lifeblood of ChatGPT and the APIs it's built on. The extremely short supply of GPUs, their quirks, and cost, dominate how we operate and scale.

To set the stage, let me introduce you to one of the GPUs we use.



The NVIDIA H100. Image source: NVIDIA

This is the NVIDIA H100. It has a special High Bandwidth Memory (HBM) memory attached to each GPU. The GPUs can talk to each other with a high bandwidth interconnect called NVLink, and they can talk to the outside world with a special ethernet alternative called Infiniband. We pack 8 of these in a single node. Nvidia sells a similar configuration called the DGX H100.

For us, every bit of added compute or bandwidth *directly* impacts the ChatGPT user experience.

## 4. Five scaling challenges

There's a lot to talk about with scaling challenges from GPUs. We focus on:

- #1: GPU RAM and KV Cache
- #2: Batch size, ops:bytes, and arithmetic intensity
- #3: The right metrics to measure
- #4: Finding GPUs, wherever they are
- #5: Autoscaling; lack thereof

Understanding these was critical while we scaled ChatGPT. Let’s get into them!

### Challenge #1: KV cache & GPU RAM

Our big challenge is that self-attention scales quadratically, meaning that the more tokens we generate, we need quadratically more operations (about ~10,000 operations for the 100th token, but about 1,000,000 for the 1,000th.) How can we improve performance?

An initial workaround is to cache the math we did for all prior tokens; the **KV cache**. We use the attention mechanism in our machine learning (ML) model. With this model, we use three vectors:

- Q: related to what we include
- K: related to what we use as input to output
- V: learned vector; the output of the calculations

We can cache *both* K and V, hence the name "KV cache." However, we cannot cache Q because this vector changes every time.

We have to store KV Cache in GPU RAM. This is because moving data in GPU RAM has a speed of around 3TB/sec, when we use High Bandwidth Memory (HBM). However, if we push data across a PCIe bus (PCI Express: a high-speed bus standard, common on motherboards to transfer data,) we get about 30GB/sec. Using HBM is around two orders of magnitude faster (around 100 times) than PCIe!

**Why is HBM so fast?** It's physically bonded to the GPUs in stacked layers, with thousands of pins for massively parallel data throughput.

This GPU HBM RAM is *very* expensive and quite limited. Most HBM RAM is also spent holding the model weights. As with any cache, when it fills up, we free up space by "expiring" the oldest data.

**"Cache misses" are expensive for our use case.** A cache miss is not locating a cached value needed for our calculation; typically because we had to "expire" this value, and kick it out of our HBM memory to free up space. If there's a cache miss, we need to recompute a *whole* ChatGPT conversation again! And we're at the 1,000th character, so that could be closer to 1 million operations!

Our infrastructure shares GPU RAM across users. What this means is that as a user, your conversation may get evicted from the system (the GPU cache) if it has been idle for too long because we need to free up space for other conversations.

This caching approach has several implications:

- **GPU RAM is a most valuable commodity**. In practice, GPU RAM, not compute resource, is the most frequent bottleneck for LLM operations.
- **Cache miss rates are *very* important!** Cache miss refers to events when the system tries to retrieve data from the cache – like the KV cache – but nothing is cached. Cache misses have massive, nonlinear influence on how much work the GPUs do. The higher the cache miss rate, the more that the GPUs need to work quadratically, not linearly.

This means that when scaling ChatGPT, there wasn't some simple CPU utilization metric to look at. We had to look at KV Cache utilization, and how to maximize GPU RAM.

## Challenge #2: Optimizing batch size

Batch size is a second metric to balance when scaling ChatGPT. Roughly speaking, batch size is the number of concurrent requests we run through a GPU.

An H100 GPU can move 3.35TB of RAM per second into its registers, at most. GPU registers are the fast on-chip memories which store operands for the operations that the computing core will then execute.

In the same second that the data is moved into the registers, the H100 can multiply 1.98 quadrillion 8-bit floating point numbers. Let's divide this 1.98 quadrillion operations by the 3.35TB of data moved; the H100 can do 591 floating point operations in the time it takes to move 1 byte.

**The H100 has a 591:1 ops:bytes ratio.** If you're going to spend time moving around 1GB, you should do *at least* 591 billion floating-point operations per second (FLOPs.) Doing fewer than this means wasting GPU FLOPS. However, if you do more, you're waiting on memory bandwidth.

In our models, the amount of memory we move around is relatively fixed at roughly the size of our model weights.

We get some control over the process by adjusting our batch size, which changes the volume of mathematical calculations involved.

When scaling ChatGPT, we needed to fully "saturate" GPUs, meaning to monitor our batch sizes so we had the right FLOPS, so the GPUs weren't under-utilized for compute, but also not over-utilized so as to wait on memory bandwidth.

In reality, it's effectively impossible to truly sustain the flop numbers printed on the box. We can use the math to get close, but a lot of experimentation in prod was required to fine tune everything.

## Challenge #3: The right metrics to measure

The combination of batch size and KV cache utilization was the primary metric we focused on. These are two numbers we used to determine how loaded our servers were. We didn't get to this metric from the start; it took time to figure out it works best for us.

Initially, we had a simpler gpu utilization metric, similar to standard cpu utilization metrics. However, it turned out to be misleading, as simple utilization *only* said whether or not the gpu was doing math in a time period. It didn't tell us:

- Whether we had saturated **arithmetic intensity**; the ratio of floating point operations to total data movement: FLOPs/byte. Basically, GPU utilization didn't tell us if we were under-utilizing compute, or being starved of memory.

- Whether we were running out of KV cache. This is a problem because with KV cache misses, the GPU needs to do quadratically more computation to calculate uncached results.

**Other bottlenecks, too.** KV cache and batch size were not the only bottlenecks we found. In the real world, we've had bottlenecks from:

- Memory bandwidth
- Network bandwidth between GPUs
- Network bandwidth between nodes (machines)
- …to name just three!

To make it more complicated, the locations of bottlenecks change frequently. For example, asking ChatGPT to summarize an essay has vastly different performance characteristics from asking it to write one. We are constantly tweaking specific details on the LLM transformer model architecture and these changes affect where bottlenecks occur.

There's an unexpectedly wide variability in the constraints you run up against in the LLMs' problem space. This makes it hard for us to solve, and for chip manufacturers to design chips that achieve the optimal balance.

For example, NVidia's new H100 chip is the generation after the A100 chip. The H100 increases compute (aka FLOPs) by about 6x, while memory size stays constant and memory bandwidth increases only 1.6x. Since we're frequently bottlenecked by memory, the dramatic increase in FLOPs (and cost) tends to go underutilized.

The reality is that a top-of-the-line chip today like the NVIDIA H100 has a design locked in years ago. NVIDIA had no way to know of discoveries on the importance of memory back when the H100 chip was designed, as future architectures are hard to predict. The recently announced H200 chip has increased memory; however it took the industry a while to better understand these unique bottlenecks to running LLMs at scale.

## Challenge #4: Finding GPUs wherever they are

Another challenge with GPUs was where to find them! Our company – and the AI space in general – has grown much faster than suppliers like NVIDIA, or the complete TSMC supply chain, can produce GPUs. With a supply chain as complex as semiconductors and data centers, bottlenecks happen in many places.

One solution was to get GPUs from wherever we could. We use Microsoft Azure as our cloud provider, which has GPUs in many different geographic regions and data centers. As a result, we quickly found ourselves in many regions all over the world.

From day one, our tiny team was forced to go multi-region, multi-cluster, and globally distributed. Our kubernetes clusters very quickly went from being treated like pets, to more like cattle.

**A well-balanced GPU fleet became more of a priority than allocating nearby GPUs.** For ChatGPT, the biggest bottleneck in response time is how quickly GPUs can stream out one token at a time. This means that GPUs geographically closer to end users is less of a priority because the round-trip time a network request takes matters less, than GPUs being “ready to go.” My career has taught me the importance of computing at the edge for latency reasons. This ended up being less relevant for these workloads, especially when GPUs are so constrained.

### Challenge #5: Inability to autoscale

A final, dominant challenge was the inability to scale up our GPU fleet. There are simply no more GPUs to buy or rent, and therefore no GPUs to autoscale into. The difficulty of acquiring GPUs continues to this day and shows no signs of easing up. The exponent on demand currently appears larger than the exponent on supply. Not only are there more users, but larger models require ever-more compute, and new techniques like agents take substantially more compute per user.

*Note from Gergely: we recently covered Meta’s commitment to spend $7-9B on 350,000 NVIDIA H100 GPUs by the end of 2024 – around 15% of global supply!*

So, if you ever saw the "We are at capacity" message from ChatGPT, this was a correct statement! It meant we had nowhere to scale into, as we’d utilized all our GPUs at that time.

Unfortunately, models like GPT-4 take so much compute that GPUs are our only option right now, in order to offer our service at scale. Plain CPUs would be orders of magnitude slower.

## 5. Lesson Learned

Scaling ChatGPT is certainly not your average software engineering scaling problem. However, it has been a crash course in engineering scaling, with plenty of learnings to apply in other situations. Here are key lessons we learned.

**In early scaling, both the forest AND the trees are important.** Low level details (trees,) like KV cache optimization and CUDA kernels, were as important as higher level system details (forest,) such as the global data center strategy. Our team's ability to jump across the stack – from the lowest levels of the GPUs, all the way to product decisions – was critical.

**Take your system's constraints into account in an adaptive way.** Before joining OpenAI, I was used to doing things like:

- Tweaking systems to achieve around 80% CPU utilization metrics – generally considered "good." Once this metric is hit, sit back while the metric stays constant.
- Autoscale into an "infinitely big" cloud; meaning more machines can always be provisioned.
- Prioritize edge computing: move it very close to where users are, to reduce latency of roundtrip requests, and improve user experience.

At OpenAI, none of these practices applied! We needed to come up with new approaches that were both practical to measure and scale.

And just when we'd figure out something that worked, a model architecture would shift, or a new inference idea would be proposed, or product decisions would change usage of the system. So, we needed to adapt, again and again.

**Dive deep.** In the problem space in which we operate, the lowest-level implementation details *really* matter. It's tempting to think of ChatGPT as a "black box" that takes text as input, and spits out slightly smarter text at the other end. However, the more people dove into the smallest details of how exactly the "black box" works, the better we became as a team and product.

**It's still early days.** The scale of challenges will only grow. With every jump between GPT-2, 3, and 4, we needed entirely new ways to train and run the models at scale. This shall continue in future versions. All our new modalities like vision, images, and speech, require re-architecting systems, while unlocking new use cases.

The pace of development at OpenAI – and the ecosystem as a whole – is accelerating. We'll see what challenges the next 10x of scale holds!

## Takeaways

*This is Gergely again.*

Thanks Evan for drawing back the curtain on the challenges of scaling ChatGPT, and giving an overview of how ChatGPT works. Here are my takeaways from Evan's article:

**How ChatGPT works isn't magic, and is worth understanding.** Like most people, my first reaction to trying out ChatGPT was that it felt *magical*. I typed in questions and got answers that felt like they could have come from a human! ChatGPT works impressively well with human language, and has access to more information than any one human could handle. It's also good at programming-related questions, and there was a point when I questioned if ChatGPT could be *more capable* than humans, even in areas like programming, where humans have done better, until now?

For a sense of ChatGPT's limitations, you need to understand how it works. ChatGPT and other LLMs do not "think" and "understand" like humans. ChatGPT does, however, generate words based on what the next most likely word should be, looking at the input, and everything generated so far.

In an excellent deepdive about how ChatGPT works, Stephen Wolfram – creator of the expert search engine WolframAlpha – summarizes ChatGPT:

> "The basic concept of ChatGPT is at some level rather simple. Start from a huge sample of human-created text from the web, books, etc. Then train a neural net to generate text that's "like this". And in particular, make it able to start from a "prompt" and then continue with text that's "like what it's been trained with".
>
> As we've seen, the actual neural net in ChatGPT is made up of very simple elements—though billions of them. And the basic operation of the neural net is also very simple, consisting essentially of passing input derived from the text it's generated so far "once through its elements" (without any loops, etc.) for every new word (or part of a word) that it generates.
>
> But the remarkable—and unexpected—thing is that this process can produce text that's successfully "like" what's out there on the web, in books, etc. (...)
>
> The specific engineering of ChatGPT has made it quite compelling. But ultimately (at least until it can use outside tools) ChatGPT is "merely" pulling out some "coherent thread of text" from the "statistics of conventional wisdom" that it's accumulated. But it's amazing how human-like the results are."

**Engineering challenges of scaling ChatGPT are relatable.** When scaling a system, these are common techniques to use:

- Improve performance by trading memory for repeated, expensive compute operations (memorization,) or vice versa; using compute to save on memory if memory is low, and there's lots of unused compute
- Buy more hardware to scale horizontally
- Figure out the right things to measure, and make improvements so those metrics trend in the right direction; rinse and repeat

The engineering team at OpenAI employed all these techniques to scale the product.

**Some scaling challenges can be reduced to solving a math problem.** In Challenge #2 – optimizing for batch size – solving for maximum utilization came down to maximizing the number of operations while also not waiting on memory bandwidth. This becomes an optimization exercise once you know the values on operations and memory bandwidth.

When faced with the challenge to optimize efficiency, take inspiration from this approach. Figure out the characteristics of your system – throughput, latency, CPU utilization, memory utilization, and other relevant values – and figure out how changing one will change another. What would higher CPU utilization (using fewer machines) mean? What about adding or removing memory from each machine? And so on.

**Even OpenAI struggles to source GPU hardware.** Large language models need massive amounts of compute, and GPU chips are the best suited for this use case. Today, NVIDIA's H100 – a high-performance GPU – is the hardware of choice for most companies investing heavily in AI.

Demand for H100s outstrips supply, especially with companies like Meta spending eye-watering sums on sourcing them. We recently covered how Meta wants to have 350,000 H100 units by the end of 2024.

Despite its strong partnership with Microsoft and large amounts of funding, OpenAI also has the challenge of getting enough GPUs.

NVIDIA should profit massively from the demand, but we can expect other players to step up and design chips to compete with it. Indeed, Meta is building its own custom AI chips, as is AWS. The major chip makers are also not idle: AMD launched the MI300X processor in December 2023 and early benchmarks show better performance than the H100. Intel is working on the Gaudi3 processor that is meant to rival and surpass the H100 with an expected launch later in 2024.

Still, even with new GPUs coming to the market, ramping up mass production will take more time. Companies in the AI space can likely expect a scarcity of GPUs for the next year or two.

**Engineering challenges related to hardware scarcity tend to lead to clever workarounds.** Few software engineers today are familiar with hardware scarcity because there is no shortage of compute or storage capacity, thanks to cloud providers. If more CPUs or storage are needed, the question is how much it will cost, not whether or not capacity is available.

At the same time, we've seen the challenges of industry-wide hardware scarcity before. Such constraints nudge engineers to come up with clever efficiency wins. For example, virtualization was born from efforts to utilize hardware resources better, as we cover in The past and future of modern backend practices.

I hope to see more clever workarounds born out of necessity by companies like OpenAI, which result in lower compute needs for even the more sophisticated AI applications. I'm excited about the future that applied AI applications bring, but hope practical AI applications do *not* require compute infrastructure that consumes hundreds of times more energy to operate, than current computing infrastructure does. Computers are efficient machines that get a lot of work done with relatively little energy required. Hopefully, we keep it this way, even as the race to build better, more capable AI applications continues.

I hope you enjoyed this deepdive into how the world's best-known AI chatbot works.

*If you could help sharing details on what it's like to work with GenZ developers – or the other way around – please share your observations here.*

---