GPT-4 Architecture, Infrastructure, Training Dataset, Costs, Vision, MoE

Dylan Patel, Gerald Wong : : 2023/7/11

Demystifying GPT-4: The engineering tradeoffs that led OpenAI to their architecture.

If you will be in Hawaii for ICML, let us know, let's hang out!

/PAID CONTENT

OpenAI is keeping the architecture of GPT-4 closed not because of some existential risk to humanity but because what they've built is replicable. In fact, we expect Google, Meta, Anthropic, Inflection, Character, Tencent, ByteDance, Baidu, and more to all have models as capable as GPT-4 if not more capable in the near term.

Don't get us wrong, OpenAI has amazing engineering, and what they built is incredible, but the solution they arrived at is not magic. It is an elegant solution with many complex tradeoffs. Going big is only a portion of the battle. OpenAI's most durable moat is that they have the most real-world usage, leading engineering talent, and can continue to race ahead of others with future models.

We have gathered a lot of information on GPT-4 from many sources, and today we want to share. This includes model architecture, training infrastructure, inference infrastructure, parameter count, training dataset composition, token count, layer count, parallelism strategies, multi-modal vision adaptation, the thought process behind different engineering tradeoffs, unique implemented techniques, and how they alleviated some of their biggest bottlenecks related to inference of gigantic models.

The most interesting aspect of GPT-4 is understanding why they made certain architectural decisions.

Furthermore, we will be outlining the cost of training and inference for GPT-4 on A100 and how that scales with H100 for the next-generation model architectures.

First off, with the problem statement. From GPT-3 to 4, OpenAI wanted to scale 100x, but the problematic lion in the room is cost. Dense transformers models will not scale further. A dense transformer is the model architecture that OpenAI GPT-3, Google PaLM, Meta LLAMA, TII Falcon, MosaicML MPT, etc use. We can easily name 50 companies training LLMs using this same architecture. It's a good one, but it's flawed for scaling.

See our discussion training cost from before the GPT-4 announcement on the upcoming AI brick wall for dense models from a **training cost** standpoint. There we revealed what OpenAI is doing at a high-level for GPT-4's architecture as well as training cost for a variety of existing models.

Over the last 6 months we realized that **training cost are irrelevant**.

Sure, it seems nuts on the surface, tens of millions if not hundreds of millions of dollars of compute time to train a model, but that is trivial to spend for these firms. It is effectively a Capex line item where scaling bigger has consistently delivered better results. The only limiting factor is scaling out that compute to a timescale where humans can get feedback and modify the architecture.

Over the next few years, multiple companies such as Google, Meta, and OpenAI/Microsoft will train models on supercomputers **worth over one hundred billion dollars**. Meta is burning over $16 billion a year on the "Metaverse", Google waste's $10 billions a year on a variety of projects that will never come to fruition. Amazon has lost over $50+ billion on Alexa. Cryptocurrencies wasted over $100 billion on nothing of value.

These firms and society in general can and will spend over one hundred billion on creating supercomputers that can train single massive model. These massive models can then be productized in a variety of ways. That effort will be duplicated in multiple counties and companies. It's the new space race. The difference between those prior wastes and now is that with AI there is tangible value that will come from the short term from human assistants and autonomous agents.

The much more important issue with scaling AI, the real AI brick wall, is **inference.** The goal is to decouple training compute from inference compute. This is why it makes sense to train well past Chinchilla optimal for any model that will be deployed. This is why you do sparse model architecture; every parameter is not activated during inference.

The real battle is that scaling out these models to users and agents costs far too much. The costs of inference exceed that of training by multiple folds. This is what OpenAI's innovation targets regarding model architecture and infrastructure.

Inference of large models is a multi-variable problem in which model size kills you for dense models. We have discussed this regarding the edge in detail here, but the problem statement is very similar for datacenter. The quick rundown is that devices can never have enough memory bandwidth for large language models to achieve certain levels of throughput. Even if they have enough bandwidth, utilization of hardware compute resources on the edge will be abysmal.

On Device AI – Double-Edged Sword

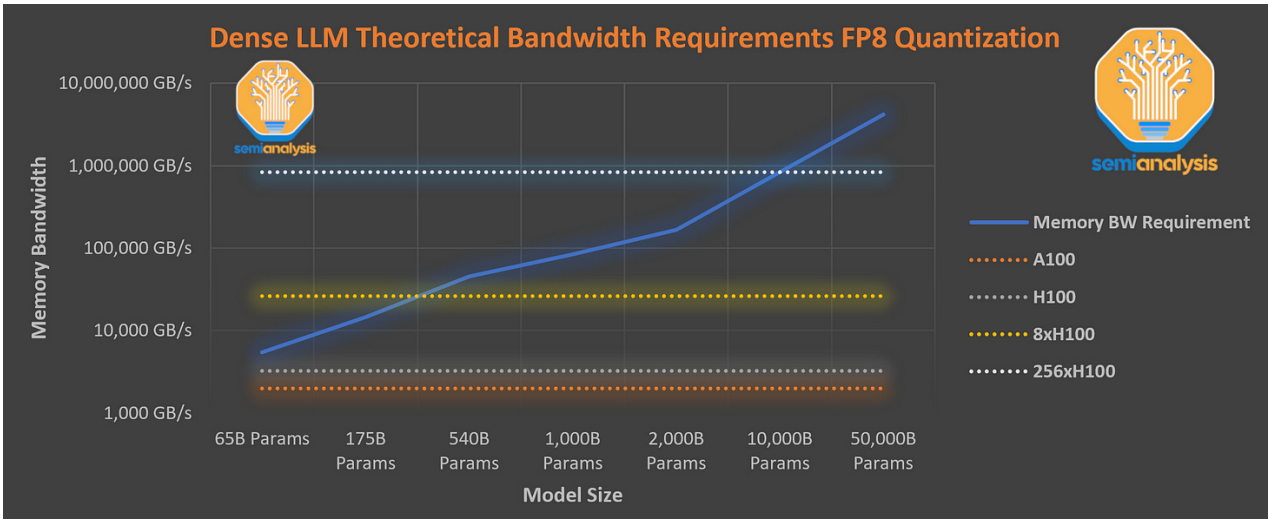
Read full story→

In the datacenter, in the cloud, utilization rates are everything. Half the reason Nvidia is lauded for software excellence is because over a GPU's generations lifespan, Nvidia is constantly updating low level software that pushes FLOPS utilization rates up with smarter movement of data around a chip, between chips, and memory.

LLM inference in most current use cases is to operate as a live assistant, meaning it must achieve throughput that is high enough that users can actually use it. Humans on average read at ~250 words per minute but some reach as high as ~1,000 words per minute. This means you need to output at least 8.33 tokens per second, but more like 33.33 tokens per second to cover all corner cases.

A trillion-parameter dense model mathematically cannot achieve this throughput on even the newest Nvidia H100 GPU servers due to memory bandwidth requirements. Every generated token requires every

parameter to be loaded onto the chip from memory. That generated token is then fed into the prompt and the next token is generated. Furthermore, additional bandwidth is required for streaming in the KV cache for the attention mechanism.



> This chart assumes inefficiencies from not being able to fuse every op, memory bandwidth required for the attention mechanism, and hardware overhead are equivalent to parameter reads. In reality, even with "optimized" libraries such as Nvidia's FasterTransformer library, the total overhead is even larger.

The chart above demonstrates the memory bandwidth required to inference an LLM at high enough throughput to serve an individual user. It shows that even 8x H100 cannot serve a 1 trillion parameter dense model at 33.33 tokens per second. Furthermore, the FLOPS utilization rate of the 8xH100's at 20 tokens per second would still be under 5%, resulting is horribly high inference costs. Effectively there is an inference constraint around ~300 billion feed-forward parameters for an 8-way tensor parallel H100 system today.

Yet OpenAI is achieving human reading speed, with A100s, with a model larger than 1 trillion parameters, and they are offering it broadly at a low price of only $0.06 per 1,000 tokens. That's because it is sparse, IE not every parameter is used.

Enough waffling about, let's talk about GPT-4 model architecture, training infrastructure, inference infrastructure, parameter count, training dataset composition, token count, layer count, parallelism strategies, multi-modal vision encoder, the thought process behind different engineering tradeoffs, unique implemented techniques, and how they alleviated some of their biggest bottlenecks related to inference of gigantic models.

Model Architecture

GPT-4 is more than 10x the size of GPT-3. We believe it has a total of ~1.8 trillion parameters across 120 layers versus the ~175 billion parameters of GPT-3.

OpenAI was able to keep costs reasonable by utilizing a mixture of experts (MoE) model. If you are unfamiliar with MoE, read our post about the broad GPT-4 architecture and training cost from 6 months

ago.

Furthermore, OpenAI utilizes 16 experts within their model, each is about ~111B parameters for MLP. 2 of these experts are routed to per forward pass.

While the literature talks a lot about advanced routing algorithms for choosing which experts to route each token to, OpenAI's is allegedly quite simple, for the current GPT-4 model.

Furthermore, there are roughly ~55B shared parameters for attention.

Each forward pass inference (generation of 1 token) only utilizes ~280B parameters and ~560 TFLOPs. This contrasts with the ~1.8 trillion parameters and ~3,700 TFLOP that would be required per forward pass of a purely dense model.

Dataset Composition

OpenAI trained GPT-4 on ~13 trillion tokens. This makes sense, given CommonCrawl for RefinedWeb contains ~5 trillion tokens high-quality tokens. For reference, Deepmind's Chinchilla and Google's PaLM model were trained with ~1.4 trillion tokens and ~0.78 trillion tokens, respectively. Even PaLM 2 is allegedly trained on ~5 trillion tokens.

This dataset is not 13 trillion unique tokens. Instead, the dataset contains multiple epochs due to a lack of high-quality tokens. There were 2 epochs for text-based data and 4 for code-based data. Interestingly, this is far short of Chinchilla optimal, indicating the need to train the model on double the token count. This indicates there is a lack of easy-to-source tokens on the web. There are 1,000x more high-quality text tokens out there and even more audio and visual, but sourcing them isn't as simple as a web scrape.

There is millions of rows of instruction fine-tuning data from ScaleAI as well as internally. Unfortunately, we could not find much out on their RLHF data.

There was an 8k context length (seqlen) for the pre-training phase. The 32k seqlen version of GPT-4 is based on fine-tuning of the 8k after the pre-training.

The batch size was gradually ramped up over a number of days on the cluster, but by the end, OpenAI was using a batch size of 60 million! This, of course, is "only" a batch size of 7.5 million tokens per expert due to not every expert seeing all tokens.

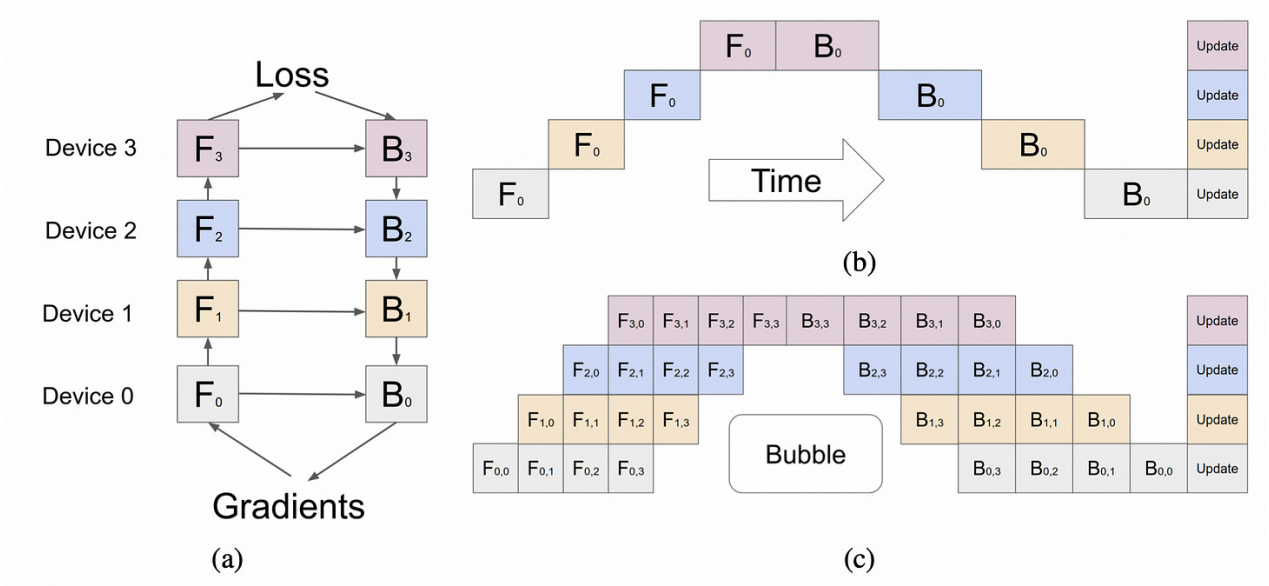
Parallelism Strategies

The strategy for parallelizing across all their A100 GPUs is critically important. They utilized 8-way tensor parallelism as that is the limit for NVLink. Beyond that, we hear they are using 15-way pipeline parallelism. Theoretically, this is too many pipelines when considering data-comm vs compute time, but if they are memory capacity bound, then it makes sense.

It is ~30GB just for the parameters at FP16 per GPU when purely pipeline + tensor parallel. Once you add on KV cache and overhead, this theoretically makes sense if a large portion of OpenAI's GPUs are 40GB A100s. They likely used ZeRo Stage 1. It is possible they used block-level FSDP, or hybrid shared data parallel.

As for why they didn't use full model FSDP, it could be because of the higher communication overhead. While OpenAI has high-speed networking between most nodes, it may not be between all their nodes. We believe at least some of the clusters are connected at much lower bandwidth than the other clusters.

We don't understand how they avoid having huge bubbles for every batch with such high pipeline parallelism. It's likely they just ate the cost.



Training Cost

OpenAI's training FLOPS for GPT-4 is ~2.15e25, on ~25,000 A100s for 90 to 100 days at about 32% to 36% MFU. Part of this extremely low utilization is due to an absurd number of failures requiring checkpoints that needed to be restarted from. The above mentioned bubbles are extremely costly.

Another reason is that all-reduce between that many GPUs is extremely costly. The is especially true if, as we suspect, the cluster is really a bunch of smaller clusters with much weaker networking between them IE 800G/1.6T non-blocking between segments of the cluster, but those segments are only connected at 200G/400G.

If their cost in the cloud was about $1 per A100 hour, the training costs for this run alone would be about $63 million. This ignores all the experimentation, failed training runs, and other costs such as data gathering, RLHF, staff, etc. The true cost is much higher due to these factors. Furthermore, it implies you have someone to buy the chips/networking/datacenter, absorb the Capex, and rent them to you.

Today, the pre-training could be done with ~8,192 H100 in ~55 days for $21.5 million at $2 per H100 hour. Note that we believe there are 9 firms that will have more H100s by the end of this year. Not all of these firms will dedicate all of them to a single training run, but those that do will have a much larger model. Meta will have over 100,000 H100's by the end of the year, but a significant number will be distributed across their datacenters for inference. Their largest individual cluster will still be **well over** 25k H100s.

Many firms will have the compute resources to train a GPT-4 size model by the end of this year.

Mixture of Expert Tradeoffs

MoE is a great way to reduce parameter count during inference while still pumping up parameter count, which is required to encode more information per training token. This is necessary as acquiring enough high-quality tokens is extremely difficult. OpenAI would have had to train on 2x the tokens if they were actually trying to go Chinchilla optimal.

With that said, there are multiple tradeoffs that OpenAI has made. For example, MoE is incredibly difficult to deal with on inference because not every part of the model is utilized on every token generation. This means parts may sit dormant when other parts are being used. When serving users, this really hurts utilization rates.

Researchers have shown that using 64 to 128 experts achieves better loss than 16 experts, but that's purely research. There are multiple reasons to go with fewer experts. One reason for OpenAI choosing 16 experts is because more experts are difficult to generalize at many tasks. More experts can also be more difficult to achieve convergence with. With such a large training run, OpenAI instead chose to be more conservative on the number of experts.

Furthermore, running with fewer experts also helps with their inference infrastructure. There are a variety of difficult tradeoffs when moving to a mixture of experts' inference architecture. Let's start with the basic tradeoffs with inference for LLMs before moving to what OpenAI faces and the choices they made.

Inference Tradeoffs

Before starting, as an aside, we want to point out that every LLM company we have spoken with thinks Nvidia's FasterTransformer inference library is quite bad, and that TensorRT is even worse. The lack of ability to take Nvidia's template and modify it means that people create their own solutions from scratch. For those of you at Nvidia reading this, you need to get on this ASAP for LLM inference, or else the defacto will become an open tool, which can add 3rd party hardware support much more easily. A wave of huge models is coming. If there is no software advantage in inference, and handwritten kernels are required anyways, then there is a much larger market for AMD's MI300 and other hardware.

There are 3 main tradeoffs for inference of large language models that occur along the batch size (number of concurrent users served) dimension and the number of chips used.

1. **Latency** – The model must respond in reasonable latency. Humans don't want to wait many seconds before waiting for their output to start streaming in a chat application. Prefill (input tokens) and decode (output tokens) take varying amounts of time to process.

2. **Throughput** – The model must output a certain number tokens outputted per second. Somewhere around 30 tokens per second is what is needed for humans use. Lower and higher throughput are okay for various other usecases.

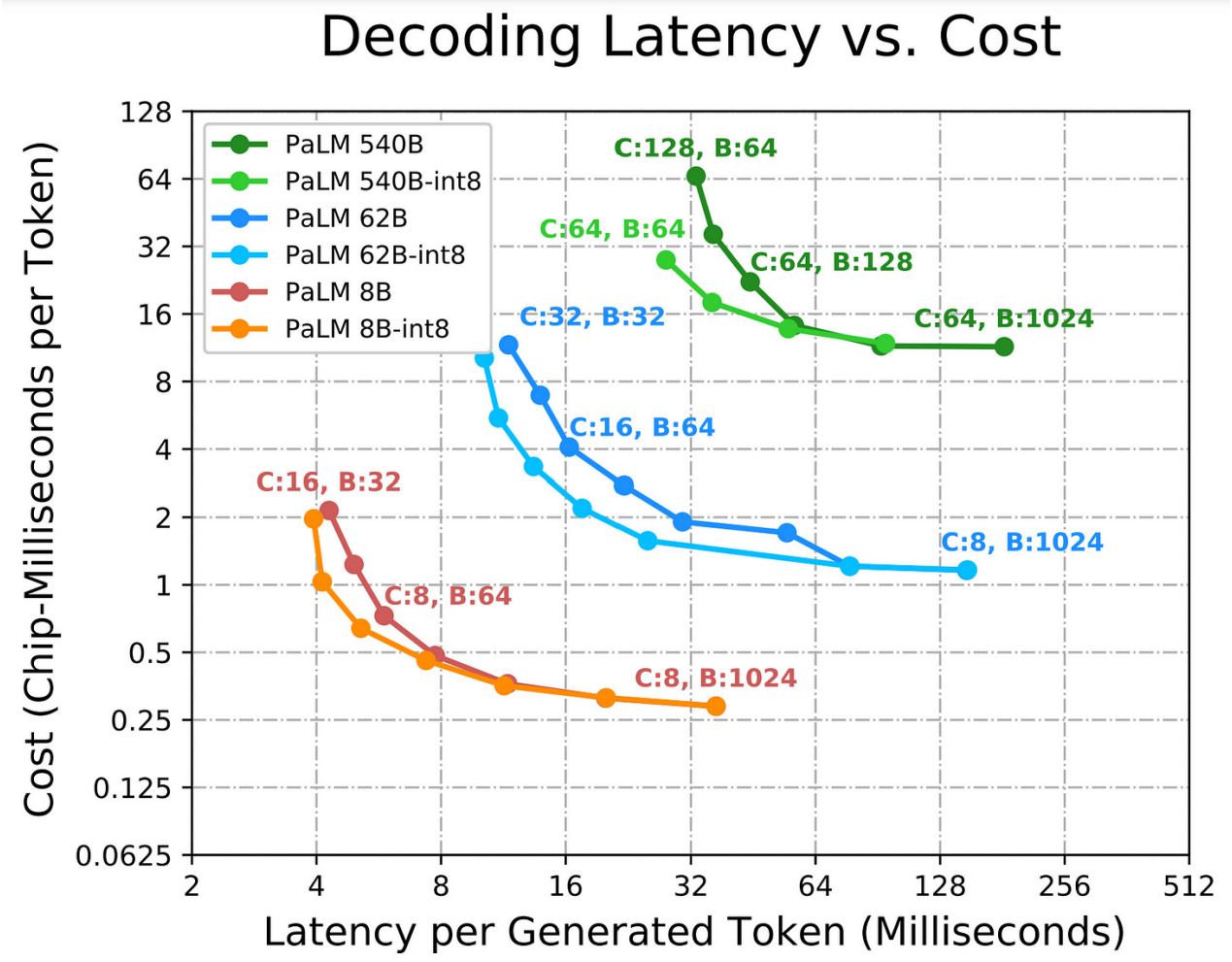
3. **Utilization** – The hardware running the model must achieve high utilization, or else it will be too costly. While higher latency and lower throughput can be used to group more user requests together

and achieve much higher utilization rates, they make it more difficult.

LLM inference is all about balancing 2 main points, memory bandwidth and compute. In the most oversimplified terms, each parameter must be read, and it has 2 FLOPs associated to it. As such, the ratio of most chips, (H100 SXM only has 3TB/s of memory bandwidth, but 2,000 TFLOP/s of FP8), is completely imbalanced for inference at batch size 1. If only 1 user is being served, batch size 1, then the memory bandwidth required to stream in every parameter for every token generation dominates inference time. Compute time is near nothing.

To efficiently scale a large language model out to many users, batch size must exceed 1. Multiple users amortize the parameter read cost. For example at batch size of 256 or 512, there are 512 FLOP/s or 1024 FLOP/s for each byte of memory that read in. This ratio more closely matches the H100's memory bandwidth versus FLOPS. This helps achieve much higher utilization, but that comes the drawback of higher latency.

Many point to memory capacity as a major bottleneck for LLM inference due to the size of the model that can fit on a number of chips, but that is incorrect. While large models require multiple chips to inference and higher memory capacity leads to them fitting on fewer chips, it is actually better to use more chips than is required capacity-wise so latency can be brought back down, throughput can be increased, and larger batch sizes can be used for increasingly higher utilization rates.



Google showed off these tradeoffs in their PaLM inference paper. However, it's noteworthy that this was for a dense model like PaLM, not a sparse model like GPT-4.

> If an application requires the lowest possible latency, we need to apply more chips and partition the model in as many ways as we profitably can. Lower latency can often be achieved with smaller batch sizes, but smaller batch sizes also result in worse MFU [utilization], resulting in a higher total cost (in terms of chip-seconds or dollars) per token.
>
> If an applications requires offline inference and latency is not a concern, the primary goal is to maximize per-chip throughput (i.e., minimize total cost per token). It is most efficient to increase the batch size because larger batches typically result in better MFU [utilization], but certain partitioning strategies that are not efficient for small batch sizes become efficient as the batch size grows larger.

More chips and higher batch sizes are the cheapest because the increase utilization, but then that also introduces a 3rd variable, networking time. Some methods of splitting a model across chips are more efficient for latency, but tradeoff with utilization.

> Both the weight loading part of the memory time and the non-attention compute time are proportional to the model size and inversely proportional to the number of chips. However, for a given partitioning layout, the time needed for chip-to-chip communication decreases less quickly (or not at all) with the number of chips used, so it becomes an increasingly important bottleneck as the chip count grows.

While we will only briefly discuss it today, it should be noted that the memory requirements for KV cache explode in capacity as batch size and seqlen grow.

> If an application requires generating text with long attention contexts, it substantially increases the inference time. For a 500B+ model with multihead attention, the attention KV cache grows large: **for batch size 512 and context length 2048, the KV cache totals 3TB, which is 3 times the size of the model's parameters**. The on-chip memory needs to load this KV cache from off-chip memory once for every token generated during which the computational core of the chip is essentially idle.

Longer sequence lengths are particularly nasty on memory bandwidth and memory capacity. OpenAI's 16k seqlen GPT 3.5 turbo and 32k seqlen GPT 4 are so much more expensive because they cannot utilize larger batch sizes due to memory constraints. Lower batch sizes leads to lower hardware utilization. Furthermore, with larger sequence length, the KV cache balloons. KV cache cannot be shared between users, so that requires individual memory reads, further bottlenecking memory bandwidth. More on MQA in a bit.

GPT-4 Inference Tradeoffs And Infrastructure

All of the above is difficult with GPT-4 inference, but the model architecture being a Mixture of Experts (MoE) introduces a whole new set of difficulties. Each token generation forward pass can be routed to a different set of experts. This throws a wrench into the tradeoffs that are achieved along the axis of throughput, latency, and utilization at higher batch sizes.

OpenAI's GPT-4 has 16 experts, with 2 routed to per forward pass. This means that if there is a batch size of 8, the parameter read for each expert could be at only batch size 1. Worse yet, it could mean 1 expert could be at a batch size of 8 and others could be at 4 or 1 or 0. Every single token generation, the routing algorithm will send the forward pass in a different direction, leading to significant variation in token-to-token latency as well as expert batch size.

Inference infrastructure is of the primary reasons OpenAI went with a much smaller number of experts. If they went with an even larger number of experts, memory bandwidth would bottleneck inference even more. OpenAI regularly hits a batch size of 4k+ on their inference clusters, which means even with optimal load balancing between experts, the experts only have batch sizes of ~500. This requires very large amounts of usage to achieve.

Our understanding is that OpenAI runs inference on a cluster of 128 GPUs. They have multiple of these clusters in multiple datacenters and geographies. The inference is done at 8-way tensor parallelism and 16-way pipeline parallelism. Each node of 8 GPUs has only ~130B parameters, or less than 30GB per GPU at FP16 and less than 15GB at FP8/int8. This enables inference to be run on 40GB A100's as long as the KV cache size across all batches doesn't balloon too large.

Individual layers containing various experts are not broken up across different nodes because that would make network traffic too irregular and recalculating KV cache between each token generation would be far too costly. The biggest difficulty for any future MoE model scaling and conditional routing is how to deal with routing around the KV cache.

The layer count is 120, so it is simple to dive that among 15 different nodes, but because the first node needs to do dataloading and embedding, it would make sense to put fewer layers on the head node of the inference cluster. Furthermore, there are some murmurs of speculative decoding, which we will discuss later, but we aren't sure if we believe them. That would also explain why the head node needs to contain so many fewer layers.

GPT-4 Inference Cost

GPT-4 costs 3x that of the 175B parameter Davinchi model despite being only 1.6x the feed-forward parameters. This is largely due to the larger clusters required for GPT-4 and much lower utilization achieved.

We believe that it costs $0.0049 cents per 1k tokens for 128 A100s to inference GPT-4 8k seqlen and $0.0021 cents per 1k tokens for 128 H100's to inference GPT-4 8k seqlen. It should be noted, we assume decent high utilization, and keeping batch sizes high.

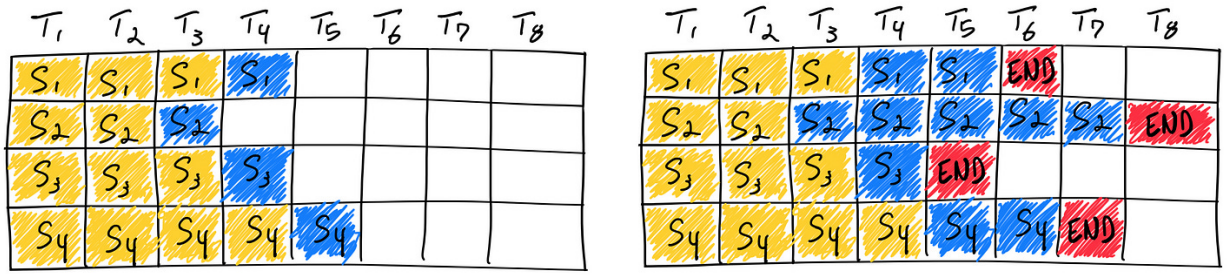
This may be an erroneous assumption as it's pretty clear OpenAI sometimes has very poor utilization at times. We assume OpenAI takes clusters down during the trough hours and repurposes those nodes to resume training from checkpoints for smaller test models trying a variety of new techniques. This helps keep inference costs low. If OfpenAI does not do this, their utilization rates would be even lower, and our cost estimates would more than double.

Multi-Query Attention

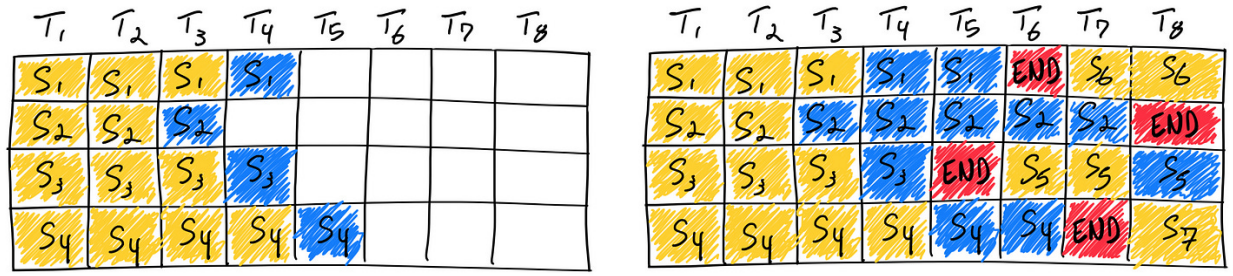
MQA is something everyone else is doing, but we wanted to note OpenAI is as well. Long story short, only 1 head is needed and memory capacity can be significantly reduced for the KV cache. Even then, the 32k seqlen GPT-4 definitely cannot run on 40GB A100s, and the 8k is capped on max batch size. Without it, the 8k would be significantly capped on max batch size to the point where it's uneconomical.

Continuous batching

OpenAI implements both variable batch sizes and continuous batching. This is so as to allow some level of maximum latency as well optimizing the inference costs. This page by AnyScale is worth a read if you are unfamiliar with the concept.



Completing four sequences using static batching. On the first iteration (left), each sequence generates one token (blue) from the prompt tokens (yellow). After several iterations (right), the completed sequences each have different sizes because each emits their end-of-sequence-token (red) at different iterations. Even though sequence 3 finished after two iterations, static batching means that the GPU will be underutilized until the last sequence in the batch finishes generation (in this example, sequence 2 after six iterations).



Completing seven sequences using continuous batching. Left shows the batch after a single iteration, right shows the batch after several iterations. Once a sequence emits an end-of-sequence token, we insert a new sequence in its place (i.e. sequences S5, S6, and S7). This achieves higher GPU utilization since the GPU does not wait for all sequences to complete before starting a new one.

Speculative Decoding

We have heard from some reliable folks that OpenAI uses speculative decoding on GPT-4 inference. We aren't sure if we believe it to be clear. The general variation in token to token latency and difference when doing simple retrieval tasks vs more complex tasks seems to indicate it's possible, but there are way too many variables to know. Just in case, we will explain it here by using some of the text in "Accelerating LLM Inference with Staged Speculative Decoding" and modifying a bit/adding some color.

Using an LLMs is generally split into two phases,. First, prefill , the prompt is run through the model to generate the KV cache and the first output logits (probability distribution of possible token outputs). This is usually fast, as the entire prompt can be handled in parallel.

The second phase is decoding. A token is selected from the outputted logits and fed back into the model, which produces logits for the following token. This is repeated until the desired number of tokens is produced. Because decoding must be done sequentially weights streamed through the compute units each time in order to generate a single token, the arithmetic intensity (that is, FLOP of compute / byte of memory bandwidth) of this second phase is extremely low when run in small batches. As such, decoding is usually the most expensive part of autoregressive generation.

This is why input tokens are much cheaper than output tokens in OpenAI's API calls.

The basic idea of speculative decoding is to use a smaller, faster draft model to decode several tokens in advance, and then feeds them into the oracle model as a single batch. If the draft model was right about its predictions – the larger model agrees – one can decode several tokens with a single batch, which saves considerable memory bandwidth, and thus time, per token.

However, if the larger model rejects the tokens predicted by the draft model, then the rest of the batch is discarded and the algorithm naturally reverts to standard token-by-token decoding. Speculative decoding may also be accompanied by a rejection sampling scheme to sample from the original distribution. Note this is only useful in small-batch settings where bandwidth is the bottleneck.

Speculative decoding trades compute for bandwidth. There are two key reasons why speculative decoding is an attractive performance engineering target. First, it does not degrade model quality at all. Second, the gains it provides are generally orthogonal to other methods, because its performance comes from converting sequential execution to parallel execution.

Current speculative methods predict a single sequence for the batch. However, **this doesn't scale well to large batch sizes or low draft model alignments**. Intuitively**, the probability that two models agree for long consecutive sequences of tokens is exponentially low**, which means that **speculative decoding has rapidly diminishing returns as one scales its arithmetic intensity**.

We believe that if OpenAI is using speculative decoding, they are likely only using it for sequences of ~4 tokens. As an aside the whole conspiracy of lowering quality of GPT-4 could just be because they are letting the oracle model accept lower probability sequences from the speculative decoding model. Another aside is that some folks speculate that bard uses speculative decoding because Google waits for the entire sequence to be produced before sending it to users, but we don't believe this speculation is true.

Vision Multi-Modal

The vision multimodal capabilities are the least impressive part of GPT-4, at least compared to leading research. Of course, no one has commercialized any of the research into multi-modal LLMs yet.

It is a separate vision encoder from the text encoder, but there is cross-attention. We hear the architecture is similar to Flamingo. This adds more parameters on top of the 1.8T of GPT-4. It is fine-tuned with another ~2 trillion tokens, after the text only pre-training.

On the vision model, OpenAI wanted to train it from scratch, but it wasn't mature enough, so they wanted to derisk it by starting with text.

The next model, GPT-5, they train will supposedly be from scratch for vision and be able to generate images on its own too. Furthermore, it will be able to do audio as well.

One of the primary purposes of this vision capability is for autonomous agents able to read web pages and transcribe what's in images and video. Some of the data they train on is joint data (rendered LaTeX/text), screen shots of web page, youtube videos: sampling frames, and run Whisper around it to get transcript.

One interesting thing about all this over-optimization for LLMs is that the IO cost of a vision model differs from that of text. On a text model, it is extremely cheap as we described in the Amazon Cloud Crisis piece. On vision, the data-loading is ~150x higher IO. 600 bytes per token rather than 4 like with text. There is a lot of work being done on image compression.

This is extremely relevant for hardware vendors who are optimizing hardware 2-3 years out from now around the usecase and ratios of LLMs. They may find themselves in a world where every model has robust vision and audio capabilities. They may find their architecture poorly adapted. In general, the architecture will definitely evolve past the current simplified text-based dense and/or MoE models we see today.

/PAID CONTENT