مقیاس بندی خودکار
سرویس دهی مدل‌های زبانی بزرگ

پروپوزال

۳

محمد سعید صفایی صادق

مفاهیم پایه

Prompt چیست؟

Prompt به ورودی متنی گفته می‌شود که به یک مدل زبانی مثل GPT4 یا LLaMA داده می‌شود تا بر اساس آن، پاسخی متنی output تولید کند.

Prompt به‌عنوان شروع فرآیند inference استنتاج تعریف شده است. یعنی:

An inference process starts from a request (prompt) with a "list of input tokens..."

فرآیند استنتاج با یک درخواست که همان prompt است آغاز می‌شود، که شامل فهرستی از توکن‌های ورودی است.

Prompt چیست؟

Prompt → Input Tokens → Output Tokens

مدل زبانی ابتدا prompt را به توکن‌های ورودی تبدیل می‌کند، سپس با استفاده از روش تولید خودبازگشتی autoregressive، توکن‌های خروجی را یکی‌یکی تولید می‌کند.

Prompt بر محتوای تولیدشده، سبک پاسخ، و رفتار مدل تأثیر مستقیم دارد.

Input Tokens توکن‌های ورودی:

تعریف:

عبارت است از رشته‌ای از توکن‌ها (کلمات، بخش‌هایی از کلمات، یا کاراکترها) که از prompt یا متن اولیه‌ی کاربر استخراج می‌شود و به مدل داده می‌شود.

مثال:

اگر ورودی prompt این باشد:

What is Chiron?

مدل آن را به توکن‌هایی مانند زیر تبدیل می‌کند (وابسته به tokenizer مدل):

"What", " is", " Ch", "iron", "?"

Output Tokens توکن‌های خروجی:

تعریف:

توکن‌هایی هستند که مدل به‌صورت مرحله‌به‌مرحله autoregressive بر اساس توکن‌های ورودی تولید می‌کند تا یک پاسخ کامل بسازد.

مثال:

پاسخ مدل به prompt بالا ممکن است این باشد:

"Chiron is an autoscaler for large language model serving systems..."

که خودش شامل ده‌ها توکن خروجی است.

تعداد input tokens روی هزینه‌ی پردازش اولیه prefill تأثیر می‌گذارد.

تعداد output tokens روی زمان پاسخ‌دهی، هزینه‌ی محاسباتی، و inter-token latency (ITL) اثر مستقیم دارد.

Autoregressive Generation

تولید خودبازگشتی به روشی در تولید متن گفته می‌شود که مدل زبانی هر توکن خروجی را بر اساس توکن‌های قبلی (ورودی‌ها و توکن‌های تولیدشده‌ی قبلی) تولید می‌کند.

این فرآیند به‌صورت مرحله‌به‌مرحله انجام می‌شود:

$$y_1 \leftarrow P(y_1 \mid x)$$

$$y_2 \leftarrow P(y_2 \mid x, y_1)$$

$$y_3 \leftarrow P(y_3 \mid x, y_1, y_2)$$

...

که در آن:

x: توکن‌های ورودی (Input tokens یا همان prompt)

$y_1, y_2, \ldots$: توکن‌های خروجی تولیدشده (Output tokens)

Autoregressive Generation

چرا "بازگشتی" autoregressive نامیده می‌شود؟

چون مدل توکن جدید را با استفاده از نتیجه‌ی مراحل قبلی (خروجی‌های خودش) پیش‌بینی می‌کند.

یعنی هر مرحله به تاریخ

چه‌ی تولیدشده‌ی قبلی وابسته است.

فرآیند تولید نمی‌تواند موازی انجام شود.

این محدودیت باعث می‌شود latency بین توکن‌ها ITL بالا برود.

ITL = Inter-Token Latency

ITL مدت‌زمانی است که مدل صرف می‌کند تا هر توکن خروجی جدید را پس از توکن قبلی تولید کند.

این زمان شامل:

محاسبه attention خودتوجهی

دسترسی به حافظه مثلاً از KV Cache

و سایر هزینه‌های محاسباتی در مرحله decoding است.

فرض کنید مدل GPT نیاز دارد:

۲۰۰ میلی‌ثانیه برای تولید هر توکن ← این می‌شود ITL = 200ms

اگر پاسخ شامل ۳۰ توکن باشد:

مجموع زمان پاسخ ~۶ ثانیه بدون TTFT

نکته مهم ITL vs TBT

در برخی منابع غیررسمی یا عمومی، TBT ممکنه به صورت غیررسمی به "**Time Between Tokens**" اشاره کنه.

این تعبیر، در واقع همون چیزیه که ما در مدل‌های زبانی بهش می‌گیم ITL (Inter-Token Latency).

یعنی:

TBT (Time Between Tokens) ≈ ITL

اما...

در این مقاله و در ادبیات پژوهشی رسمی مانند OSDI، SOSP، MLSys :

TBT معمولاً به معنی **Total Batch Time** → Time to process the Batch هست.

ITL به‌صورت رسمی برای **Inter-Token Latency** به کار می‌رود (که دقیقاً همون "زمان بین توکن‌ها" است).

تفاوت ITL و TBT

مفهوم	ITL	TBT
نام کامل	Inter-Token Latency	Total Batch Time
ترجمه فارسی	تاخیر بین توکن‌ها	زمان کل اجرای یک batch
تعریف	مدت‌زمان تولید یک **توکن خروجی واحد** (بعد از قبلی)	کل زمان لازم برای پردازش **تمام درخواست‌های داخل یک batch**
سطح مشاهده	**سطح توکن – ریزبینانه**	**سطح batch – کلی‌تر**
کاربرد اصلی	اندازه‌گیری تأخیر در تجربه کاربر (برای پاسخ‌های تعاملی)	ارزیابی کارایی و بهره‌وری سیستم در سطح batch
اهمیت در مقاله	معیار کلیدی برای رعایت **SLO** در پاسخ تعاملی	شاخص بهره‌وری پردازش در حضور **batching و multiplexing**
وابسته به	Self-attention cost، queueing، preemption	Batch size، تعداد درخواست‌ها، مدل انتخاب‌شده

مثال ساده برای درک تفاوت:

فرض کنیم یک batch حاوی ۱۰ درخواست است و هر کدام از آن‌ها نیاز به تولید ۵ توکن دارد.

اگر ITL = 200ms باشد

→ هر توکن از پاسخ ۵تایی با فاصله ۲۰۰ ms تولید می‌شود

اگر کل زمان اجرا برای آن batch = 2.5s باشد

→ TBT = 2.5 seconds

مفاهیم

تفاوت در اصطلاح‌گذاری:

اصطلاح	کاربرد غیررسمی	کاربرد رسمی (مثل مقاله Chiron)
TBT	ممکنه به "Time Between Tokens" اشاره کنه (غیررسمی)	Time to complete one" یا "Total Batch Time" "batch
ITL	-	مدت‌زمان بین تولید هر دو توکن – Inter-Token Latency
TTFT	-	مدت‌زمان برای تولید اولین توکن (prefill latency) – Time to First Token

Speculative Decoding چیست؟

یکی از تکنیک‌های نوآورانه در مدل‌های زبانی بزرگ LLMs است که به‌ویژه در تسریع فرایند تولید توکن‌ها و بهبود کارایی مدل‌ها به کار می‌رود.

این تکنیک در تلاش است تا زمان تاخیر latency را کاهش داده و تولید سریع‌تر توکن‌ها را در پردازش‌های پیچیده‌ی autoregressive ممکن سازد.

Speculative Decoding به معنای پیش‌بینی موازی توکن‌ها است، جایی که مدل به‌طور موازی و پیش‌بینی‌کننده چندین توکن را به صورت پیش‌فرض speculatively تولید می‌کند، حتی قبل از اینکه مدل به‌طور قطعی نتیجه‌ی آن‌ها را تایید کند.

این فرآیند می‌تواند در مواردی که مدل نیاز به زمان زیادی برای تولید هر توکن دارد، به سرعت تولید توکن‌ها کمک کند.

Speculative Decoding چیست؟

در سیستم‌های LLM مانند GPT، که به‌طور طبیعی autoregressive هستند، مدل‌ها برای تولید هر توکن به مدل قبلی وابسته‌اند.

این وابستگی باعث افزایش زمان پردازش برای هر توکن می‌شود، به ویژه زمانی که توکن‌ها زیاد باشند.

Speculative Decoding این مشکل را با پیش‌بینی توکن‌ها به‌صورت هم‌زمان حل می‌کند، که به مدل این امکان را می‌دهد که تعداد بیشتری از توکن‌ها را به‌طور سریع‌تر تولید کند.

Speculative Decoding چیست؟

KV Cache :

در سیستم‌های LLM، KV Cache برای ذخیره‌سازی داده‌های کلید-مقدار key-value توکن‌ها استفاده می‌شود.

استفاده از Speculative Decoding در کنار KV Caching می‌تواند کارایی پردازش را به حداکثر برساند.

Batching :

این تکنیک می‌تواند در پردازش دسته‌ای batching مفید باشد چون می‌تواند پیش‌بینی‌ها را برای چند درخواست به‌طور موازی انجام دهد، این کار باعث می‌شود که مدل توکن‌ها را به‌طور بهینه‌تری تولید کند.

Speculative Decoding چیست؟

مثال عملی:

فرض کنید مدل باید برای یک پرسش پاسخ دهد و در این پاسخ باید به ترتیب جملات را تولید کند.

در حالت معمول، مدل فقط یک جمله تولید می‌کند و سپس جمله بعدی را می‌سازد، که این روند زمان‌بر است.

با Speculative Decoding ، مدل می‌تواند چند جمله را به صورت پیش‌بینی شده تولید کند و در صورت نیاز، اصلاحات را روی توکن‌های تولید شده اعمال کند.

Prefix Caching چیست؟

Prefix Caching به معنای ذخیره‌سازی اطلاعات مربوط به توکن‌های قبلی (یا پیش‌فرض‌ها) است تا هنگام تولید توکن‌های بعدی، از آن‌ها برای تسریع فرآیند تولید استفاده شود.

به‌عبارت دیگر، مدل می‌تواند توکن‌هایی که قبلاً تولید کرده را به یاد بیاورد و نیازی به پردازش دوباره آن‌ها نداشته باشد.

زمانی که مدل توکن‌هایی تولید می‌کند، به‌ویژه در فرآیندهای autoregressive که هر توکن به توکن قبلی وابسته است، مدل می‌تواند از داده‌های قبلی برای تولید توکن‌های بعدی استفاده کند.

در Prefix Caching، توکن‌ها به‌عنوان یک پیشوند prefix ذخیره می‌شوند و هنگام تولید توکن‌های جدید، مدل می‌تواند از این پیشوندها برای محاسبات بعدی استفاده کند و نیاز به پردازش مجدد آنها نخواهد بود.

Prefix Caching چیست؟

مثال عملی:

فرض کنید یک مدل برای پاسخ به سوال زیر کار می‌کند:

"Explain the process of photosynthesis in plants."

مدل ابتدا توکن‌های ورودی را پردازش می‌کند.

مدل Prefix Caching را فعال می‌کند تا توکن‌های اولیه را در حافظه ذخیره کند **(مثل واژه‌ها و بخش‌های اولیه جمله).**

زمانی که مدل به مرحله تولید توکن‌های بعدی می‌رسد، به‌جای محاسبه مجدد آن توکن‌ها، از Prefix Cache استفاده می‌کند تا سرعت پردازش را افزایش دهد.

Chunked Prefill چیست؟

Chunked Prefill یکی از تکنیک‌های نوین و مهم در پردازش مدل‌های زبانی بزرگ LLMs است که به‌ویژه در تسریع فرآیند prefill پیش‌پر کردن و بهینه‌سازی حافظه کاربرد دارد.

این تکنیک به مدل‌های زبانی کمک می‌کند تا بتوانند ورودی‌های طولانی را با بهره‌وری بهتر از منابع پردازش کنند.

در حالت معمول، وقتی مدلی مانند GPT یا LLaMA یک ورودی طولانی دریافت می‌کند، تمامی ورودی باید به طور همزمان در حافظه مدل قرار گیرد و سپس پردازش شود.

این می‌تواند باعث استفاده زیاد از حافظه و کاهش کارایی شود.

در Chunked Prefill، ورودی به بخش‌های کوچک‌تر تقسیم می‌شود و مدل ابتدا هر بخش را پردازش می‌کند و سپس به بخش بعدی می‌رود.

این تقسیم‌بندی می‌تواند به طور مؤثری مقدار حافظه مورد نیاز را کاهش دهد و پردازش موازی را بهبود بخشد.

Chunked Prefill چیست؟

مثال عملی:

فرض کنید شما یک مدل زبانی دارید و ورودی شما به شکل زیر است:

"Explain the process of photosynthesis in plants. Photosynthesis is the process by which green plants and some other organisms use sunlight to synthesize foods with the help of chlorophyll..."

این ورودی ممکن است خیلی طولانی باشد و بارگذاری تمام آن در حافظه به صورت یک‌باره باعث افت کارایی شود.

در Chunked Prefill، این ورودی به بخش‌های کوچکتر تقسیم می‌شود:

Chunk 1: "Explain the process of photosynthesis in plants."

Chunk 2: "Photosynthesis is the process by which green plants and some other organisms use sunlight..."

مدل ابتدا بخش اول را پردازش می‌کند، سپس به بخش بعدی می‌رود.

این فرآیند باعث می‌شود که مدل بتواند ورودی‌های بزرگ را به‌طور مؤثرتر پردازش کند و مصرف حافظه را کاهش دهد.

StreamingLLM چیست؟

StreamingLLM به یک رویکرد در پردازش مدل‌های زبانی گفته می‌شود که به مدل اجازه می‌دهد توکن‌ها را به صورت پیوسته streaming تولید کند، بدون اینکه نیاز باشد تمام ورودی از قبل پردازش شود.

به عبارت دیگر، مدل می‌تواند به طور مداوم داده‌ها را پردازش کرده و پاسخ‌ها را در حین پردازش تولید کند.

StreamingLLM چیست؟

مدل‌های StreamingLLM به‌طور خاص طراحی شده‌اند تا بتوانند داده‌ها را به صورت جریان‌محور پردازش کنند. این کار به این صورت انجام می‌شود:

۱- **دریافت ورودی به صورت تدریجی**: به‌جای اینکه مدل یک ورودی کامل را یک‌باره دریافت کند، ورودی به‌صورت تدریجی و پیوسته به مدل وارد می‌شود.

۲- **تولید پاسخ‌ها به صورت پیوسته**: به محض دریافت هر بخش از ورودی، مدل شروع به تولید پاسخ‌ها می‌کند. مدل به‌طور پیوسته توکن‌ها را تولید کرده و بدون نیاز به انتظار برای دریافت کامل ورودی، به پاسخ‌دهی ادامه می‌دهد.

۳- **پیش‌بینی توکن‌ها**: در مدل‌های autoregressive، مدل به‌طور پیوسته توکن‌ها را تولید می‌کند، هر بار یک توکن، با استفاده از توکن‌های قبلی و داده‌های ورودی.

این فرآیند در StreamingLLM می‌تواند بدون وقفه و به‌طور همزمان با دریافت داده‌های جدید ادامه یابد.

StreamingLLM چیست؟

مثال:

کاربر:

"فرآیند فتوسنتز در گیاهان را به‌طور کامل توضیح بده."

گام اول: ورودی کاربر وارد می‌شود

در ابتدا، کاربر سوالی به مدل ارسال می‌کند. مدل، به‌جای اینکه منتظر بماند تا تمام ورودی پردازش شود، از StreamingLLM استفاده می‌کند و به‌طور پیوسته ورودی را پردازش می‌کند.

مدل بلافاصله اولین بخش از ورودی را دریافت می‌کند، یعنی:

"فرآیند فتوسنتز در گیاهان..."

StreamingLLM چیست؟

گام دوم: پردازش بخش اول ورودی

مدل شروع به پردازش این بخش می‌کند و اولین توکن‌ها را تولید می‌کند:

مدل تولید اولین بخش پاسخ را شروع می‌کند:

"فتوسنتز فرآیندی است که در آن گیاهان از نور خورشید برای تولید انرژی استفاده می‌کنند..."

در همین حال، مدل همچنان منتظر بخش‌های بعدی ورودی از کاربر است، اما پاسخ اولیه را به‌طور هم‌زمان با پردازش اطلاعات ارسال می‌کند.

StreamingLLM چیست؟

گام سوم: دریافت و پردازش بخش دوم ورودی

در این مرحله، مدل به‌طور پیوسته به پردازش ادامه می‌دهد و اطلاعات جدید وارد می‌شود:

کاربر ممکن است بخش بعدی سوال را ارسال کند:

"آن‌ها نور را از طریق کلروفیل در برگ‌های خود جذب می‌کنند."

مدل این بخش جدید را پردازش کرده و پاسخ را تکمیل می‌کند، به طوری که بخش جدید وارد پاسخ قبلی می‌شود.

مدل توکن‌های بعدی را تولید می‌کند:

"کلروفیل نور را جذب کرده و آن را به انرژی شیمیایی تبدیل می‌کند..."

StreamingLLM چیست؟

گام چهارم: پاسخ‌دهی پیوسته

در این مرحله، مدل در حال ارسال پاسخ است در حالی که همچنان به پردازش ورودی ادامه می‌دهد.

مدل به سرعت اطلاعات را پردازش کرده و پاسخ به کاربر را به‌طور مداوم به‌روزرسانی می‌کند.

پاسخ نهایی به کاربر به صورت زیر خواهد بود:

"فتوسنتز فرآیندی است که در آن گیاهان از نور خورشید برای تولید انرژی استفاده می‌کنند. کلروفیل نور را جذب کرده و آن را به انرژی شیمیایی تبدیل می‌کند که سپس برای تولید گلوکز از دی‌اکسید کربن و آب استفاده می‌شود. این فرآیند برای رشد گیاه و تولید اکسیژن ضروری است."

FlashAttention

FlashAttention یک الگوریتم بهینه برای محاسبات Attention است که در مدل‌های ترنسفورمر به کار می‌رود.

هدف این تکنیک، افزایش سرعت محاسبات و کاهش استفاده از حافظه است. FlashAttention به‌طور خاص از حافظه GPU استفاده بهینه می‌کند و از آن برای تسریع محاسبات attention به‌ویژه در مدل‌های بزرگ استفاده می‌کند.

برای مثال، در پردازش یک دنباله طولانی از کلمات یا توکن‌ها، مدل باید تمامی وزن‌های توجه attention weights را برای هر جفت توکن محاسبه کند.

در اینجا، FlashAttention با کاهش هزینه حافظه و افزایش سرعت محاسبات، این فرآیند را سریع‌تر و کارآمدتر می‌کند.

FlashAttention

در الگوریتم‌های سنتی Self-Attention :

تعداد محاسبات به صورت $O(n^2)$ است، جایی که n تعداد توکن‌ها است.

نیاز به حافظه زیاد برای ذخیره وزن‌های توجه و توکن‌ها وجود دارد.

در FlashAttention :

محاسبات توجه به طور بهینه‌تری انجام می‌شود، در نتیجه مصرف حافظه کاهش می‌یابد.

زمان پردازش کاهش یافته و سرعت بالاتری برای مدل‌های بزرگ فراهم می‌شود.

ChunkedAttention

ChunkedAttention به معنای تقسیم ورودی‌ها به قطعات کوچک chunks و انجام محاسبات توجه attention به‌صورت جداگانه برای هر قطعه است.

به‌طور کلی، این تکنیک برای کاهش پیچیدگی محاسبات $O(n^2)$ در مدل‌های ترنسفورمر استفاده می‌شود که معمولاً با افزایش تعداد توکن‌ها به شکل نمایی افزایش می‌یابد.

ChunkedAttention

مثال عملی:

فرض کنید یک مدل ترنسفورمر برای پردازش یک ورودی بسیار طولانی با ۱۰۰۰ توکن طراحی شده است. در حالت معمول، برای هر توکن، باید با تمام ۹۹۹ توکن دیگر محاسبات attention انجام شود. این باعث می‌شود که پیچیدگی محاسباتی به $O(n^2)$ برسد و به شدت زمان و مصرف حافظه افزایش یابد.

اما با استفاده از ChunkedAttention :

ورودی ۱۰۰۰ توکنی به بخش‌های کوچکتر (مثلاً هر بخش ۱۰۰ توکن) تقسیم می‌شود.

محاسبات attentionفقط برای هر ۱۰۰ توکن انجام می‌شود، که پیچیدگی محاسباتی را به $O(n)$کاهش می‌دهد.

این روش باعث می‌شود که مدل قادر به پردازش ورودی‌های طولانی‌تر با منابع کمتر و زمان پردازش کوتاه‌تر باشد.

ChunkedAttention

چالش‌ها و محدودیت‌ها:

۱- از دست دادن دقت:

ChunkedAttention ممکن است دقت را کاهش دهد زیرا بخش‌ها به‌طور مستقل پردازش می‌شوند و رابطه معنایی بین بخش‌ها ممکن است از دست برود.

۲- پیچیدگی در هماهنگی بخش‌ها:

هماهنگ کردن چند بخش به‌طور مستقل می‌تواند به چالش تبدیل شود و به دقت بیشتری در طراحی نیاز دارد.

۳- نیاز به تنظیمات دقیق:

مدل باید به‌طور دقیق تنظیم شود تا بخش‌ها به‌طور بهینه پردازش شوند و از کاهش عملکرد یا از دست دادن اطلاعات جلوگیری شود.

RelayAttention

RelayAttention یک تکنیک بهینه‌سازی محاسبات توجه است که هدف آن کاهش مصرف حافظه و افزایش کارایی در پردازش داده‌های طولانی است.

این روش به‌ویژه در مدل‌های Self-Attention استفاده می‌شود، جایی که مدل باید برای هر توکن ورودی با تمامی توکن‌های دیگر ارتباط برقرار کند و وزن‌های توجه را محاسبه کند.

در RelayAttention، برخلاف روش‌های معمول که ممکن است از حافظه زیاد برای ذخیره تمام وزن‌های توجه استفاده کنند، این تکنیک **به جای نگهداری تمام داده‌های توجه به‌طور همزمان، اطلاعات را به‌صورت پیوسته و بهینه انتقال می‌دهد** تا استفاده از حافظه و محاسبات را کاهش دهد.

RelayAttention

در مدل‌های ترنسفورمر، یکی از مشکلات اصلی در محاسبات Self-Attention ، نیاز به ذخیره و پردازش تمام وزن‌های توجه attention weights برای جفت توکن‌ها است که باعث می‌شود پیچیدگی محاسبات به صورت $O(n^2)$ باشد.

RelayAttention این مشکل را با انتقال تدریجی اطلاعات بین توکن‌ها و بهینه‌سازی استفاده از حافظه حل می‌کند.

این تکنیک در حقیقت به‌جای پردازش تمامی توکن‌ها به‌طور هم‌زمان، از یک مدل انتقالی relay model استفاده می‌کند که اطلاعات را به‌صورت پیوسته بین توکن‌ها منتقل می‌کند.

به عبارت دیگر، **RelayAttention به‌جای ذخیره تمام وزن‌های توجه برای تمام توکن‌ها، آن‌ها را به‌طور موقت و بهینه در فواصل زمانی مختلف پردازش می‌کند.**

مفاهیم

در این مقاله، FlashAttention، ChunkedAttention و RelayAttention به‌طور خلاصه به‌صورت زیر تعریف شده‌اند:

FlashAttention: یک تکنیک بهینه‌سازی است که به طور خاص برای کاهش مصرف حافظه و افزایش سرعت محاسبات در مدل‌های Self-Attention طراحی شده است. این روش از ویژگی‌های خاص حافظه GPU استفاده می‌کند تا عملیات‌های مربوط به محاسبات توجه را سریع‌تر و بهینه‌تر انجام دهد.

ChunkedAttention: این تکنیک ورودی‌ها را به بخش‌های کوچکتر تقسیم کرده و محاسبات توجه را به‌طور جداگانه برای هر بخش انجام می‌دهد. این کار باعث کاهش پیچیدگی محاسباتی و بهینه‌سازی مصرف حافظه در هنگام پردازش داده‌های طولانی می‌شود.

RelayAttention: در این روش، اطلاعات مربوط به توجه به‌جای ذخیره‌سازی همه وزن‌های توجه به‌طور همزمان، به‌طور پیوسته و بهینه بین بخش‌ها منتقل می‌شود. این تکنیک مصرف حافظه را کاهش داده و سرعت پردازش را بهبود می‌بخشد.

این سه تکنیک به‌طور کلی برای بهینه‌سازی عملکرد سیستم‌های مدل‌های زبانی بزرگ LLM و کاهش مصرف منابع طراحی شده‌اند.

مفاهیم

ویژگی	FlashAttention	ChunkedAttention	RelayAttention
هدف اصلی	افزایش سرعت و کاهش مصرف حافظه	کاهش پیچیدگی محاسباتی با تقسیم ورودی به بخش‌ها	کاهش مصرف حافظه با انتقال تدریجی اطلاعات
پردازش ورودی	تمام ورودی به‌طور همزمان پردازش می‌شود	ورودی به بخش‌های کوچکتر تقسیم می‌شود	اطلاعات بین بخش‌ها به‌طور پیوسته منتقل می‌شود
حافظه	استفاده بهینه از حافظه GPU	کاهش مصرف حافظه با تقسیم ورودی به بخش‌ها	کاهش مصرف حافظه با انتقال تدریجی اطلاعات
کارایی	سریع‌تر شدن پردازش به دلیل بهینه‌سازی GPU	کاهش زمان پردازش با پردازش بخش‌های مجزا	بهینه‌سازی حافظه و پردازش پیوسته
مناسب برای	ورودی‌های بزرگ و نیاز به پردازش موازی سریع	ورودی‌های طولانی که مصرف حافظه زیادی دارند	پردازش ورودی‌ها به‌صورت پیوسته و کاهش حافظه
پیچیدگی محاسباتی	کاهش پیچیدگی با بهینه‌سازی حافظه و GPU	کاهش پیچیدگی محاسباتی با تقسیم ورودی به بخش‌ها	پیچیدگی کمتری در ذخیره‌سازی و پردازش حافظه

Hierarchical Autoscaling

در این مقاله، Hierarchical Autoscaling به‌عنوان یک مکانیسم مقیاس‌پذیری چندسطحی برای زیرسیستم‌های مختلف در زنجیره‌ی پاسخ‌دهی مدل‌های زبانی معرفی شده است.

این سیستم به‌طور خاص برای Chiron طراحی شده؛ پلتفرمی که inference pipeline را به مراحل مجزایی تقسیم می‌کند.

مقیاس‌پذیری در هر مرحله به‌صورت مستقل انجام می‌شود.

به‌جای در نظر گرفتن یک سطح کلی مقیاس‌پذیری برای تمام پردازش‌ها، سیستم Chiron هر مرحله را به‌صورت جداگانه مانیتور و مقیاس‌دهی می‌کند.

برای مثال:

اگر در یک بازه زمانی، تعداد زیادی درخواست دارای context طولانی باشند، سیستم prefill stage را سریع‌تر مقیاس می‌دهد.

اگر سرعت تولید توکن در حال افت است، سیستم روی decode stage مقیاس‌پذیری اعمال می‌کند.

مراحل Pipeline :

۱- **Prefill Stage**: اولین مرحله از pipeline .

کل prompt به مدل داده می‌شود و خروجی‌های اولیه (hidden statesو attention) محاسبه می‌شود. سنگین‌ترین مرحله از نظر محاسباتی است (زیرا input ممکن است بسیار طولانی باشد). نیاز به GPUهای بزرگ دارد.

۲- **Decode Stage** : در این مرحله، مدل توکن‌های خروجی را به‌صورت autoregressive و تکراری تولید می‌کند. نسبت به prefill محاسبات سبک‌تری دارد، ولی چون تکرارشونده است، latency زیادی تولید می‌کند. می‌تواند با GPU های سبک‌تر اجرا شود.

۳- **Postprocess Stage**: خروجی مدل به قالب نهایی تبدیل می‌شود (مثل متن، JSON یا API response .

این مرحله بیشتر CPU-bound است، نه GPU-bound .

چرا استفاده از Pipeline مهم است؟

در مدل‌های LLM، اگر همه چیز یکجا انجام شود:

۱- Latency بالا می‌رود.

۲- منابع بیش از حد مصرف می‌شوند.

۳- throughput کاهش می‌یابد.

اما وقتی مراحل جدا می‌شوند و به‌صورت یک pipeline اجرا می‌گردند:

- ✅ چندین درخواست می‌توانند به‌طور همزمان در مراحل مختلف باشند
- ✅ هر مرحله می‌تواند به‌طور جداگانه مقیاس‌پذیر شود Hierarchical Autoscaling
- ✅ بهره‌وری سیستم بسیار بیشتر می‌شود.

تعریف Backpressure ؟

Backpressure به سیستمی گفته می‌شود که ورود درخواست‌های جدید را محدود می‌کند یا به تأخیر می‌اندازد تا زمانی که ظرفیت پردازشی آزاد شود.

یعنی وقتی یکی از مراحل pipeline مثلاً prefill یا decode به ظرفیت کامل خود برسد و دیگر نتواند درخواست جدیدی را بپذیرد،

backpressure به مراحل قبلی یا کل سیستم منتقل می‌شود تا از بارگذاری بیش از حد جلوگیری شود.

مکانیزم Backpressure در Chiron ؟

در Chiron، هر مرحله از pipeline : prefill ، decode، postprocess :

دارای صف ورودی queue مشخص با حداکثر ظرفیت است.

وقتی صف پر شود، مرحله قبلی به‌جای ادامه کار:

یا متوقف می‌شود،

یا درخواست را reject یا delay می‌کند.

این انتقال فشار به عقب = Backpressure

مثال عملی Backpressure در Chiron ؟

فرض کنید:

GPU های decode فقط می‌توانند همزمان ۱۰۰ توکن را تولید کنند.

۱۵۰ درخواست همزمان به مرحله prefill رسیده‌اند.

Prefill خیلی سریع‌تر از decode است.

🔻 بدون backpressure :

Prefill همه را انجام می‌دهد.

Decode اشباع می‌شود.

حافظه سرریز می‌شود، latency بالا می‌رود.

✅ با backpressure :

وقتی decode پر می‌شود، prefill متوقف یا کند می‌شود.

Frontend هم ممکن است درخواست‌های جدید را reject کند یا در صف قرار دهد.

سیستم پایدار باقی می‌ماند.

ارتباط Backpressure با Hierarchical Autoscaling

یکی از ویژگی‌های مهم Hierarchical Autoscaling در Chiron این است که:

وقتی یک مرحله به‌صورت مداوم backpressure تولید می‌کند،

سیستم آن مرحله را به‌صورت پویا scale-up می‌کند (مثلاً decode GPU اضافه می‌کند)،

تا گلوگاه bottleneck برطرف شود.

Latency-based Backpressure (LBP)

LBP نوع خاصی از backpressure است که به‌جای صرفاً تکیه بر ظرفیت صف‌ها یا منابع سخت‌افزاری، تصمیمات خود را بر اساس میزان تأخیر latency در مراحل مختلف pipeline اتخاذ می‌کند.

به بیان ساده:

اگر latency واقعی یک مرحله افزایش یابد و از آستانه‌ی تعیین‌شده فراتر رود، LBP به‌طور خودکار به مراحل قبلی فشار می‌آورد تا بار ورودی کاهش یابد یا جریان کنترل شود.

Throughput-based Backpressure (TBP)

TBP یک مکانیزم کنترل جریان backpressure است که بر اساس نرخ عبور واقعی درخواست‌ها throughput تصمیم می‌گیرد که آیا باید به مرحله قبل از خود فشار بیاورد یا نه.

به بیان ساده:

اگر تعداد درخواست‌های پردازش‌شده در یک بازه زمانی کاهش یابد (یعنی throughput افت کند)، TBP فعال می‌شود و به مراحل قبلی سیگنال می‌دهد که:

"من نمی‌توانم به اندازه کافی سریع پردازش کنم، فعلاً درخواست نفرست."

Interactive Backpressure (IBP)

IBP یک نوع backpressure است که در شرایط تعاملی interactive serving به کار می‌رود جایی که مدل LLM به صورت streaming یا در پاسخ‌گویی real-time با کاربر در تعامل است.

به‌طور خاص، IBP طراحی شده تا:

۱- از تجمع بیش از حد درخواست‌های decoding مرحله‌به‌مرحله جلوگیری کند

۲- جریان پاسخ‌دهی به کاربر را پایدار نگه دارد

۳- با تأخیر پایین و روان پاسخ دهد، حتی وقتی سیستم تحت بار است

Interactive Backpressure (IBP)

در شرایطی مانند:

افزایش ناگهانی تعداد کاربران تعاملی

کاهش منابع (مثلاً GPUدرگیر دیگر مراحل شده)

افزایش ITL مثلاً بیش از ۲۰۰ ms بین توکن‌ها

✅ در این حالت:

IBP فعال می‌شود و:

برخی درخواست‌های decode صف می‌شوند

برخی تولید توکن‌ها با گام‌های تنظیم‌شده انجام می‌شود

جریان به‌طور موقتی کندتر اما پایدارتر می‌شود

Batch Backpressure (BBP)

BBP یا «فشار برگشتی دسته‌ای»، مکانیزمی است که برای کنترل اندازه و نرخ تولید batch در مرحله‌ی Decode استفاده می‌شود.

این مکانیزم از آن جهت مهم است که:

در LLM serving ، اجرای دسته‌ای batching برای افزایش بهره‌وری GPU ضروری است، اما اگر batch بیش‌ازحد بزرگ یا خیلی کوچک باشد، می‌تواند عملکرد را مختل کند.»

بنابراین، BBP کمک می‌کند تا تعادل بین latency و throughput در batching حفظ شود.

اگر batch خیلی بزرگ باشد → زمان اجرای هر مرحله زیاد می‌شود → latency زیاد می‌شود.

اگر batch خیلی کوچک باشد → GPU به‌درستی استفاده نمی‌شود → throughput پایین می‌آید.

Batch Backpressure (BBP)

BBP از مشاهده وضعیت اجرا و ظرفیت پردازشی GPU استفاده می‌کند تا:

۱- زمان‌بندی اجرای batch ها را کنترل کند.

۲- در صورت overload شدن decode stage، تولید batch جدید را به تأخیر بیندازد.

۳- اگر نیاز باشد، batch ها را با درخواست‌های بعدی ترکیب کند delayed batching .

۴- از ارسال بیش‌ازحد درخواست به decode جلوگیری کند.

مفاهیم

مقایسه‌ی چهار نوع Backpressure در Chiron

ویژگی / نوع BP	LBP Latency-Based	TBP Throughput-Based	IBP Interactive	BBP Batch
هدف اصلی	جلوگیری از افزایش بیش از حد تأخیر (latency)	جلوگیری از افت در نرخ پاسخ‌دهی (throughput)	حفظ تجربه‌ی روان در تعاملات زنده و streaming	کنترل اندازه و زمان‌بندی batchها در decode
زمان فعال شدن	وقتی latency از آستانه عبور کند	وقتی throughput افت قابل‌توجه کند	وقتی کاربران زیاد در حالت تعاملی هستند و ITL زیاد شود	وقتی batchهای decode بهینه نیستند یا GPU اشباع شده
بر چه چیزی نظارت می‌کند؟	میانگین یا لحظه‌ای latency در هر مرحله	تعداد توکن یا درخواست خروجی در ثانیه	نرخ و کیفیت جریان پاسخ توکن‌به‌توکن (ITL)	حجم batchها، وضعیت GPU و تأخیر ساخت batch
مرحله‌ی هدف	همه مراحل، به‌ویژه prefill و decode	همه مراحل، به‌ویژه decode	مرحله‌ی **Decode** در حالت streaming	مرحله‌ی **Decode** در حالت batching
واکنش سیستم	کند کردن ورودی به مرحله اشباع‌شده	محدود کردن ورودی‌ها برای بازیابی throughput	pause یا throttle کردن توکن‌های streaming	تعویق ساخت batch، ترکیب با درخواست‌های بعدی
ارتباط با Autoscaling	بله – ممکن است باعث trigger autoscaling شود	بله – نشانهٔ ناکارآمدی منابع	گاهی – بیشتر برای حفظ تعامل	نه مستقیماً، بیشتر برای استفاده بهینه از GPU
مناسب برای	جلوگیری از spike در latency و حفظ SLA	حفظ بهره‌وری کلی سیستم	تعاملات زنده مثل چت یا voice	بهینه‌سازی پردازش دسته‌ای توکن‌ها
مثال ساده	latency در prefill از 1.5s بیشتر شود → ورود کند می‌شود	throughput از 500 به 200 tok/s افت کند → ورودی‌ها محدود می‌شوند	چند کاربر همزمان در حال دریافت جواب → تولید توکن‌ها یکی‌یکی کند می‌شود	GPU مشغول است ولی batch جدید آماده شده → صبر می‌کنیم یا batch را ادغام می‌کنیم

Overprovisioning

«اختصاص دادن بیش از حد منابع محاسباتی (مثل GPU، CPU، حافظه) به یک سیستم، برای اطمینان از این‌که در هیچ لحظه‌ای کمبود منابع وجود نداشته باشد.»

در واقع، برای جلوگیری از افت عملکرد در مواقع پیک (بار زیاد)، بسیاری از سیستم‌ها از پیش منابع زیادی رزرو می‌کنند حتی اگر در اکثر زمان‌ها بی‌استفاده بمانند.

یکی از اهداف اصلی مقیاس‌بندی Autoscaling، جلوگیری از همین است

وقتی بار سیستم بالا می‌رود ← منابع بیشتری اختصاص یابد scale-up

وقتی بار کاهش می‌یابد ← منابع آزاد شوند یا حذف شوند scale-down

مقیاس‌بندی دقیقاً برای حل مشکل Overprovisioning طراحی شده است.

چرا Multiplexing مهم است؟

در مرحله‌ی Decode، مدل LLM به‌صورت autoregressive توکن‌ها را یکی‌یکی تولید می‌کند. اگر برای هر درخواست فقط یک token پردازش شود، GPU بخش زیادی از زمانش idle می‌ماند.

با multiplexing، می‌توان:

چندین درخواست را هم‌زمان روی یک GPU اجرا کرد

در هر گام، برای چند درخواست token جدید تولید کرد

بهره‌وری GPU را به شدت افزایش داد

latencyبین توکن‌ها ITL را پایین نگه داشت

Hysteresis در مقیاس‌پذیری

ایجاد یک بازه‌ی امن (آستانه‌ی بالا و پایین) برای تصمیمات scale-up و scale-down ، به‌طوری‌که از نوسان flapping مکرر در تخصیص منابع جلوگیری شود.

فرض کنید:

اگر تعداد درخواست‌ها از ۱۰۰ بیشتر شود، سیستم یک GPU اضافه کند scale up

اگر کمتر از ۱۰۰ شود، یک GPU حذف کند scale down

حالا اگر بار نوسان کند بین ۹۹ و ۱۰۱:

سیستم مدام GPU اضافه و حذف می‌کند

این رفتار نوسانی و ناپایدار است

باعث هزینه، تاخیر، و بی‌ثباتی در عملکرد می‌شود

Hysteresis در مقیاس‌پذیری

با Hysteresis، دو آستانه تعریف می‌شود:

اگر بار > ۱۲۰ ← scale up

اگر بار < ۸۰ ← scale down

- ✅ بین ۸۰ تا ۱۲۰ ← هیچ اقدامی نمی‌شود
- ✅ فقط وقتی واقعا تغییر بار پایدار باشد، سیستم وارد عمل می‌شود
- ✅ این باعث ثبات در رفتار سیستم و جلوگیری از overreaction می‌شود

Hysteresis در مقیاس‌پذیری

راهکار مشابه hysteresis در Chiron	توضیح
Time smoothing	فقط اگر latency یا throughput در چند بازه زمانی پشت سر هم بالا باشد، scale-up انجام شود
Cooldown window	بعد از هر scale-up/down، تا مدت زمان مشخصی تغییر جدیدی اعمال نمی‌شود
Dual thresholds	آستانه‌های جداگانه برای بالا رفتن و پایین آمدن تعریف می‌شود (مثلاً 120 و 80)

Hysteresis در مقیاس‌پذیری

ایجاد یک بازه‌ی امن (آستانه‌ی بالا و پایین) برای تصمیمات scale-up و scale-down ، به‌طوری‌که از نوسان مکرر در تخصیص منابع جلوگیری شود.

فرض کنید:

اگر تعداد درخواست‌ها از ۱۰۰ بیشتر شود، سیستم یک GPU اضافه کند scale up

اگر کمتر از ۱۰۰ شود، یک GPU حذف کند scale down

حالا اگر بار نوسان کند بین ۹۹ و ۱۰۱:

سیستم مدام GPU اضافه و حذف می‌کند

این رفتار نوسانی و ناپایدار است

باعث هزینه، تاخیر، و بی‌ثباتی در عملکرد می‌شود

تعریف vLLM

vLLM مخفف Virtualized LLM Serving

یک موتور inference مدرن برای مدل‌های زبانی بزرگ است که:

با استفاده از تکنیک‌های پیشرفته‌ای مثل PagedAttention و مدیریت کارآمد حافظه، امکان اجرای چندین درخواست هم‌زمان را با latency پایین و throughput بالا فراهم می‌کند.

۱-افزایش بهره‌وری GPU

۲-پشتیبانی از batch کردن پیشرفته

۳-کاهش تأخیر latency در پاسخ‌دهی

۴-پردازش هم‌زمان چند درخواست multi-query serving

تعریف PagedAttention

PagedAttention در vLLM یعنی:

مدل به‌جای اینکه تمام context را در حافظه GPU نگه دارد، آن را به صفحات کوچک‌تر pages تقسیم می‌کند و فقط صفحات مورد نیاز را در لحظه بارگذاری می‌کند.

✅ نتیجه:

۱- استفاده‌ی بسیار بهتر از حافظه

۲- امکان اجرای درخواست‌های بسیار طولانی

۳- multi-tenancy راحت‌تر

تعریف Multi Tenancy

Multi-tenancy به این معناست که یک سیستم inference بتواند به‌طور هم‌زمان به چندین کاربر tenant یا اپلیکیشن مختلف خدمت‌رسانی کند، بدون اینکه آن‌ها منابع کاملاً جداگانه نیاز داشته باشند.

در LLM serving هم:

چندین کاربر/برنامه/API هم‌زمان از یک مدل استفاده می‌کنند

منابع مثل GPU، حافظه، و زمان اجرا بین آن‌ها اشتراکی ولی کنترل‌شده است

تعریف Multi Tenancy

مثال کاربردی

فرض کنید:

۵ اپلیکیشن مختلف (ترجمه، چت، خلاصه‌سازی، QA، برنامه‌نویسی)

هم‌زمان به یک مدل LLM نیاز دارند

به‌جای ایجاد ۵ نسخه جداگانه از مدل و ۵ GPU

یک GPU shared با multi-tenancy می‌تواند:

Context هر اپلیکیشن را جدا نگه دارد

توکن‌ها را با multiplexing تولید کند

پاسخ‌دهی را سریع، ایزوله و مقرون‌به‌صرفه حفظ کند

Llumnix چیست؟

Llumnix یک سیستم سرویس‌دهی برای مدل‌های زبانی بزرگ LLM است که به‌طور خاص برای مدیریت هوشمند و دینامیک درخواست‌های هم‌زمان در چند نمونه مدل عمل می‌کند

این سیستم به‌عنوان یک لایه‌ی scheduling روی موتورهایی مانند vLLM قرار می‌گیرد.

هدف آن بهبود latency مخصوصاً (tail latency، throughput و کاهش fragmentation حافظه است.

پشتیبانی از preemptive rescheduling و live migration درخواست‌ها بین نمونه‌ها برای رفع گلوگاه‌ها و حفظ سرویس با کیفیت.

تعریف Fragmentation

حافظه GPU یا منابع سیستمی به‌صورت «پراکنده و ناکارآمد» در اختیار درخواست‌ها قرار می‌گیرد، به‌طوری که اگرچه ظرفیت آزاد وجود دارد، ولی نمی‌توان از آن استفاده کرد چون این فضای آزاد به‌صورت تکه‌تکه و غیرپیوسته است.

فرض کنید یک GPU دارید با ۱۰ گیگابایت حافظه.

درخواست 4 GB → A می‌گیرد

درخواست 3 GB → B می‌گیرد

سپس B تمام می‌شود → ۳ GB آزاد می‌شود

حالا درخواست C می‌آید که به ۴ GB نیاز دارد

❌ ولی چون فضای آزاد به‌صورت ۳ + GB باقی‌مانده است (غیرپیوسته)،

C نمی‌تواند اجرا شود

حتی با اینکه حافظه کافی در مجموع وجود دارد!

⬅ این یعنی fragmentation .

Ray Serve

Ray Serve یک چارچوب framework مقیاس‌پذیر و انعطاف‌پذیر برای سرویس‌دهی مدل‌های هوش مصنوعی ML/LLM در محیط‌های production است که بخشی از پروژه‌ی متن‌باز Ray است.

Ray Serve یک سیستم توزیع‌شده برای سرویس‌دهی مدل‌های ML، مدل‌های زبانی بزرگ (LLMها)، و برنامه‌های پیچیده inference است که بر پایه‌ی موتور Ray ساخته شده و از مقیاس‌پذیری افقی، autoscaling، و composability پشتیبانی می‌کند.

Composability یعنی توانایی ترکیب ماژول‌های مستقل یا مراحل مختلف در یک سیستم، به‌طوری‌که بتوان آن‌ها را به‌صورت جدا توسعه، تست، و مقیاس‌پذیر کرد، ولی همچنان در کنار هم یک pipeline کامل بسازند.

TensorRT-LLM

TensorRT-LLM یک چارچوب inference مبتنی بر TensorRT است که به‌طور خاص برای مدل‌های transformer-based و LLMها مانند LLaMA، GPT، Falcon و Mistral و غیره بهینه‌سازی شده است.

تکنیک	توضیح
✅ INT8 و FP8 quantization	کاهش حجم مدل بدون افت دقت محسوس
✅ KV-Cache Management	مدیریت بهینه حافظه cache در مرحله decode
✅ Fused operations	ترکیب چند عملگر ریاضی به یک کرنل برای سرعت بیشتر GPU
✅ Paginated Prefill	برای جلوگیری از حافظه بالا chunked اولیه به‌صورت context پر کردن
✅ Multi-stream batching	multi-tenant از درخواست‌ها هم‌زمان، بهینه برای stream اجرای چند

TensorRT-LLM

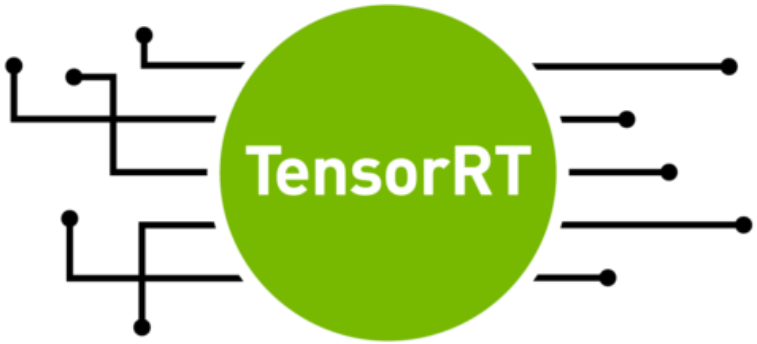
TensorRT-LLM مناسب است برای:

شرکت‌هایی که inference را روی NVIDIA GPUs در محیط production اجرا می‌کنند

نیاز به latency بسیار پایین real-time دارند مثلاً voice assistants، agents

مدل‌های سفارشی quantized یا fine-tuned

Batch های بزرگ یا multi-user serving



مقایسه با سایر سرویس‌دهنده‌های LLM

موتور / ویژگی	TensorRT-LLM	vLLM	HuggingFace TGI
Performance	🚀 بسیار بالا (NVIDIA optimized)	بالا (PagedAttention)	متوسط
Latency (ITL/TBT)	بسیار کم	کم	بالاتر
Memory Efficiency	بالا (Fused kernels)	بالا (paging)	معمولی
Hardware	فقط NVIDIA GPU	NVIDIA + Others	عموم GPUها
Streaming	✅ پشتیبانی دارد	✅ پشتیبانی دارد	✅ پشتیبانی دارد
Quantization	FP8, INT8, GPTQ	پشتیبانی جزئی	پشتیبانی محدود

QLM (Queue Length Modeling)

در زمینه‌ی LLM Serving و به‌طور کلی مدیریت منابع برای مدل‌های یادگیری عمیق، عبارت QLM (Queue Length Modeling) به یک تکنیک آماری/تحلیلی اشاره دارد که برای پیش‌بینی طول صف درخواست‌ها و کمک به تصمیم‌گیری در زمان‌بندی scheduling یا مقیاس‌پذیری autoscaling به کار می‌رود.

QLM یعنی ساخت مدلی از رفتار صف درخواست‌ها queue در سیستم برای تخمین طول صف در آینده یا ارزیابی تأخیر احتمالی بر اساس بار ورودی، ظرفیت سرویس‌دهی، و ویژگی‌های مدل.

QLM (Queue Length Modeling)

در سیستم‌های LLM serving مانند Chiron، Shepherd، FastServe یا vLLM، درخواست‌ها به‌صورت پیوسته از کاربران مختلف وارد سیستم می‌شوند. اما:

منابع محدودند (مثلاً تعداد GPU یا تعداد مدل‌های فعال)

تأخیر مجاز برای کاربران باید کنترل شود (latency SLA)

اولویت‌بندی بین درخواست‌های مختلف مهم است

بنابراین سیستم باید تصمیم بگیرد:

کدام درخواست فوراً اجرا شود؟

کدام صف انتظار تشکیل دهد؟

آیا نیاز به scale-up است؟

آیا باید یک درخواست را migrate کند به مدل دیگر؟

QLM (Queue Length Modeling)

اجزای اصلی QLM

مؤلفه	نقش
λ (Lambda)	نرخ ورود درخواست‌ها (arrival rate)
μ (Mu)	نرخ سرویس‌دهی (service rate) هر مدل/worker
Lq	طول صف پیش‌بینی‌شده
Wq	زمان انتظار متوسط در صف
ρ = λ / μ	نرخ بهره‌برداری (utilization) سیستم
SLA	سطح توافق‌شده زمان پاسخ برای درخواست‌ها

QLM (Queue Length Modeling)

مثال ساده

فرض کنید:

هر مدل LLM توانایی پاسخ به ۲ درخواست بر ثانیه دارد $\mu = 2$

نرخ ورود درخواست‌ها $\lambda = 3$ است

نرخ بهره‌برداری: $\rho = 3 / 2 = 1.5$

صف بی‌نهایت رشد می‌کند ← باید scale-up انجام شود یا بار تقسیم شود.

در QLM :

با محاسبه نرخ رشد Lq، می‌توان تشخیص داد که delay از آستانه SLAعبور خواهد کرد

در نتیجه: یا درخواست reject شود، یا منابع اضافه شوند، یا redirectشود

QLM (Queue Length Modeling)

چرا صف بی‌نهایت رشد می‌کند؟

وقتی نرخ ورود درخواست‌ها λ بیشتر از نرخ سرویس‌دهی مدل‌ها μ باشد:

- سیستم نمی‌تواند با سرعت کافی پاسخ دهد
- درخواست‌های جدید سریع‌تر وارد می‌شوند از آن‌چه که سرویس‌دهی می‌شود
- بنابراین صف پشت مدل شروع به رشد می‌کند و اگر هیچ کاری انجام نشود، به‌صورت نامحدود رشد می‌کند

QLM (Queue Length Modeling)

اگر $\lambda > \mu$ سیستم وارد حالت اشباع overload می‌شود

حتی اگر λ فقط کمی بیشتر از μ باشد، delay شروع به افزایش نمایی می‌کند

به‌خصوص در سیستم‌های LLM که هر inference زمان‌بر است، این رشد صف بسیار محسوس خواهد بود

راه‌حل‌ها: چطور جلوی رشد بی‌نهایت صف را بگیریم؟

۱. **Scale Up مقیاس‌پذیری افزایشی**

۲. **Load Balancing / Routing**

۳. **Backpressure / Rejection**

مفاهیم

Hetero یا Heterogeneous

Heterogeneous هتروژن یعنی ناهمگن بودن منابع یا اجزا یعنی سیستم از ترکیبی از GPU، CPU، TPU، یا مدل‌های مختلف برای اجرای درخواست‌ها استفاده می‌کند.

در زمینه‌ی LLM Serving یا سیستم‌های ML inference، واژه‌ی hetero یا heterogeneous به ساختارهایی اشاره دارد که شامل انواع مختلف منابع سخت‌افزاری یا مدل‌های متنوع هستند.

Heterogeneous یا Hetero

انواع ناهمگنی (Heterogeneity) در inference systems:

نوع ناهمگنی	توضیح
🖥️ سخت‌افزاری	استفاده از CPU، GPU، TPU، AWS Inferentia، Habana Gaudi و غیره در یک خوشه
🧠 مدلی	اجرای مدل‌هایی با دقت و اندازه‌ی متفاوت (مثلاً BERT-small، BERT-base، BERT-large)
🎛️ قابلیت‌ها	برخی منابع مناسب batch بزرگ‌اند، برخی مناسب streaming؛ برخی quantized هستند، برخی full-precision

چهارچوب های Inference Serving

InferLine

سیستی برای پرداختن به پیکربندی هوشمند pipeline inference است؛ با هدف بهینه‌سازی latency انتهایی از طریق تنظیمات SQL-likeبرای تنظیم batch sizes و مقیاس‌بندی افقی هر مرحله pipeline

Clipper

یکی از سیستم‌های اولیه serving مدل ML با تمرکز بر batching ساده و routing هوشمند. معمولاً از مدل‌های از پیش تعیین‌شده استفاده می‌شود و مدل‌های staticرا اجرا می‌کند (بدون انتخاب یا switch dynamic)

INFaaS

یک چارچوب model-less برای inference که خودکار متغیرهای مدل model-variants را انتخاب، autoscaling، و schedulingدر خوشه‌های هتروژن CPU/GPU/Inferentia انجام می‌دهد. هدف آن تأمین SLOهای latency/daccuracyبا هزینه پایین است

Cocktail

سیستمی برای ensemble-based serving که با انتخاب داینامیک مدل‌های مناسب در هر لحظه، ادغام weighted voting، و autoscalingفعال روی خوشه‌های ابری، دقت بالا را در latencyکاهش یافته و هزینه بهینه فراهم می‌کند

چهارچوب های Inference Serving

معیار	InferLine	Clipper	INFaaS	Cocktail
هدف اصلی	latency-aware pipeline scaling	ساده batching و routing	مدل‌لس، انتخاب خودکار variantمدل	انتخاب ensemble هوشمند + autoscaling
انتخاب مدل	ثابت یا دستی	ثابت (static)	داینامیک بر اساس SLO و cost	داینامیک و بر اساس accuracy+latency+cost
مدیریت منابع	optimization heuristic برای batch و scaling	ساده	مدل / متغیر و autoscaling منابع hetero	بسته به هر مدل pool با importance sampling
نوع پردازش	multi-stage pipeline	single-stage	single-stage	ensemble multi-model
کنترل هزینه	معمولاً ثابت	متوسط	کاهش هزینه ~1.2× نسبت به Clipper	کاهش هزینه تا ~1.4× نسبت به InFaas
درصد پاسخ در SLA	حفظ latency low	بستگی به مدل‌ها دارد	کاهش violations تا 1.6× کمتر از Clipper	تا 2× کاهش latency، 96% درخواست‌ها accurate
استفاده منابع	ساده بهینه‌سازی pipeline stages	معمولاً low utilization	heterogeneous usage ≈2.8× بهتر	بالا با ensemble و autoscale هوشمند

INFaaS

INFaaS مخفف Inference-as-a-Service است؛ یک سیستم پیشرفته برای ارائه‌ی سرویس‌های inference مدل‌های یادگیری ماشین (به‌ویژه LLMها و DNNها) به‌صورت مقیاس‌پذیر، بهینه و هوشمند در محیط‌های ابری.

INFaaS یک سیستم Model-less Inference Serving است که:

به کاربران اجازه می‌دهد بدون انتخاب مستقیم مدل، درخواست ارسال کنند

سیستم به‌طور خودکار مدل مناسب و سخت‌افزار بهینه را انتخاب می‌کند

تضمین می‌کند که SLO مانند latency و دقت رعایت شود

و در عین حال هزینه inference را بهینه می‌کند

مفاهیم

INFaaS

هدف	توضیح
🎯 پنهان‌سازی پیچیدگی مدل‌ها از کاربر	کاربر به‌جای گفتن "کدام مدل را اجرا کن؟"، می‌گوید "پاسخ دقیق بده، با تأخیر کمتر از ۱ ثانیه"
💡 انتخاب خودکار مدل و سخت‌افزار	سیستم تصمیم می‌گیرد چه مدل (BERT-base؟ DistilBERT؟) روی چه سخت‌افزاری (CPU؟ GPU؟) اجرا شود
💸 بهینه‌سازی هزینه و مصرف منابع	بر اساس میزان دقت/تأخیر موردنیاز و وضعیت منابع
📈 مقیاس‌پذیری در مقیاس کلان	در سطح دیتاسنتر، با پشتیبانی از منابع هتروژن و مدل‌های متعدد

INFaaS

سیستم	آیا یک INFaaS کامل است؟	توضیح
INFaaS	بله ✅	تعریف‌شده برای model-less inference با SLA و cost optimization
Cocktail	⚠️ تقریباً (نزدیک به INFaaS)	با selection خودکار مدل و focus روی دقت + latency، ولی محدودتر از INFaaS
InferLine	نه ❌	تنظیم pipelineهای inference، با تمرکز روی batching و latency—not model selection
Clipper	نه ❌	سیستم پایه inference serving با قابلیت ensemble ساده، بدون انتخاب دینامیک مدل یا hardware
vLLM	نه (موتور سطح پایین است) ❌	موتور سریع LLM serving، بدون routing، SLA-awareness یا model variation

مفاهیم

در ادامه به معرفی و مقایسه سه سیستم FastServe، Andes و Vidur می‌پردازیم.

هر یک از این سیستم‌ها بر روی بخش خاصی از چالش‌های LLM Serving تمرکز دارند:

۱- **FastServe** برای بهبود عملکرد و جریان واکنش سریع توکن‌ها با schedulingهوشمند طراحی شده است.

۲- **Andes** تمرکز دارد بر رضایت کاربر و تجربه روان streaming به کمک QoE-aware scheduling .

۳- **Vidur** یک ابزار تحلیلی و شبیه‌سازی است که به مدیران سیستم‌ها کمک می‌کند بدون اجرای واقعی پیکربندی‌های پیچیده را به‌صرفه پیدا کنند.

مفاهیم

مقایسه کلی: FastServe vs Andes vs Vidur

ویژگی	FastServe	Andes	Vidur
هدف اصلی	کاهش JCT + preemptive scheduling	بهبود QoE کاربر در streaming	شبیه‌ساز high-fidelity برای tuning
سطح اجرای preemption	نرخ توکن به توکن	نرخ توکن به توکن	-
معیارهای بهینه‌سازی	JCT (job completion time)	QoE (TTFT و جریان روان توکن‌ها)	latency، throughput، TTFT ،TBT
مدیریت حافظه	proactively offloading KV cache	با زمان‌بندی برای جلوگیری از congestion	بررسی configuration برای حافظه
نوع خروجی سیستم	مولتی‌دستوری (server/system)	سرویس‌دهی real-time به کاربران	ابزار طراحی و سناریو برای طراحان
کاربرد پیشنهادی	API با throughput بالا و latency پایین	چت، ویدیو-چت و UX محور	بهینه‌سازی پیکربندی cluster و deployment

QoE چیست؟ Quality of Experience

QoE = میزان درک و رضایت کاربر نهایی از کیفیت خدماتی که دریافت می‌کند

(مخصوصاً در زمینه‌هایی مثل پاسخ‌دهی real-time، چت، تولید متن و ویدئو)

تفاوت QoE با QoS

اصطلاح	تعریف	تمرکز
QoS (Quality of Service)	ویژگی‌های فنی و سیستمی مثل latency، throughput، error rate	از نگاه زیرساخت
QoE (Quality of Experience)	تجربه‌ی واقعی کاربر از دریافت پاسخ، سرعت، روان بودن و دقت	از نگاه کاربر نهایی

سناریوهای QoE در LLM

✅ تجربه خوب (QoEبالا):

TTFT < 300ms

هر توکن با فاصله ۵۰–۱۵۰ msمی‌رسد

پاسخ سریع، بدون مکث و بدون قطع

❌ تجربه بد (QoEپایین):

TTFT > 1s

توکن‌ها با تأخیر زیاد یا ناپیوسته می‌رسند

کاربر حس کند سیستم "کند" یا "گیر کرده" است

GPU Efficiency / GPU Utilization

GPU Utilization چیست؟

درصد زمانی که GPU در حال فعالیت است (مشغول انجام کار محاسباتی) نسبت به کل زمان در دسترس.

اگر یک مدل در مدت ۱۰ ثانیه اجرا شود و GPU فقط ۳ ثانیه واقعاً مشغول باشد:

Utilization = 30%

بالا بودن این عدد یعنی:

GPU مشغول است

احتمالاً batching و pipeline بهینه اجرا شده‌اند

GPU Efficiency چیست؟

نسبت مفید بودن کار انجام‌شده روی GPU به مقدار انرژی/زمانی که GPU مصرف کرده.

GPU Efficiency / GPU Utilization

نکته کاربردی:

ممکن است GPU Utilization بالا باشد ولی Efficiency پایین، اگر مثلاً:

توکن‌ها کند تولید شوند

GPU برای prefill سنگین مشغول باشد ولی پاسخ مفید ندهد

head-of-line blocking رخ داده باشد

Head-of-Line Blocking یعنی وقتی یک درخواست کند یا بزرگ در ابتدای صف قرار گرفته و باعث انتظار تمام درخواست‌های بعدی (حتی درخواست‌های سریع‌تر) می‌شود.

[درخواست 3 (متوسط)] ← [درخواست 2 (سبک)] ← [درخواست 1 (خیلی سنگین)]

اما چون زمان‌بندی به‌ترتیب ورود **FIFO** است:

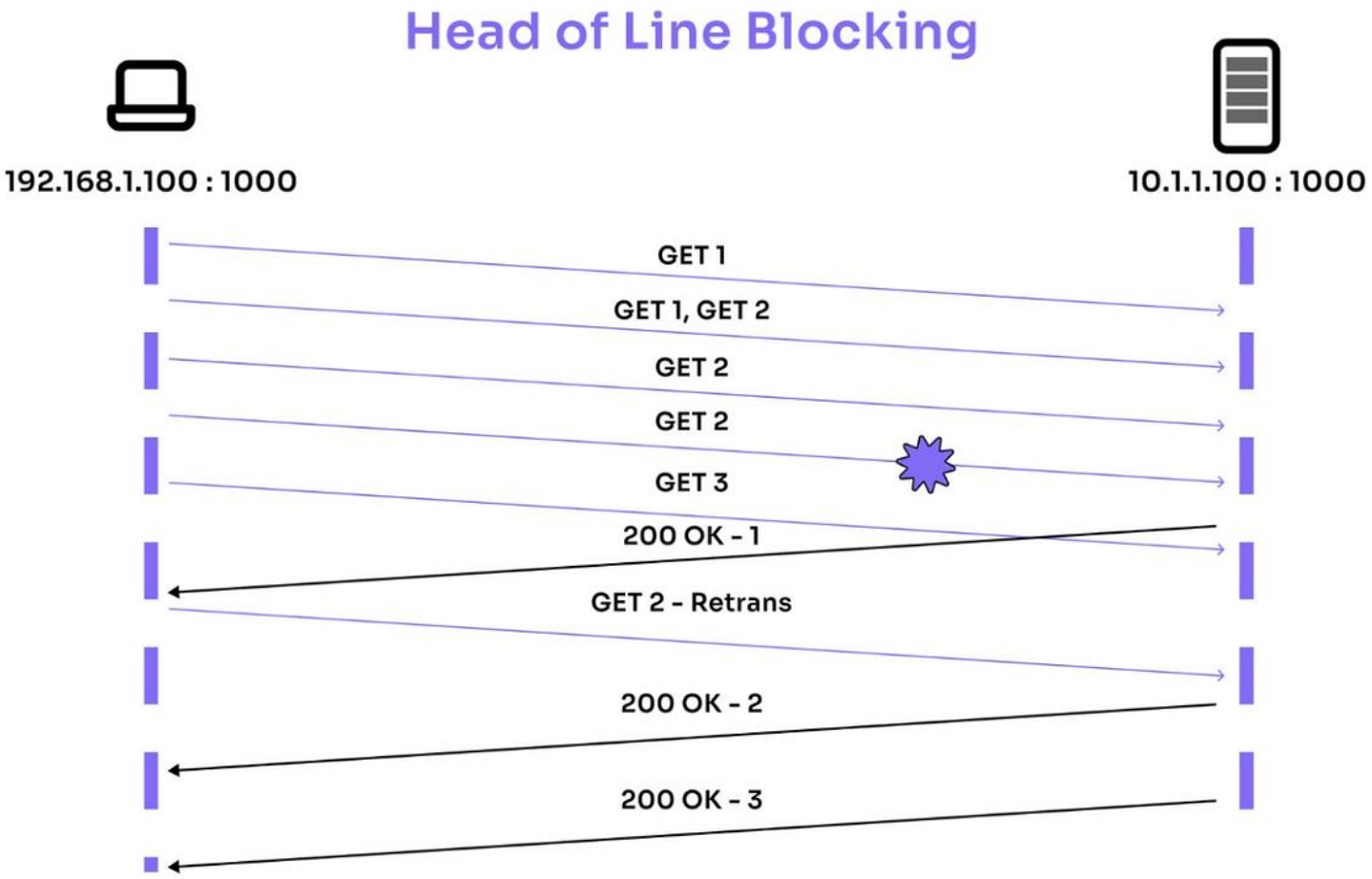
GPU باید اول درخواست ۱ را اجرا کند

حتی اگر درخواست ۲ فقط نیاز به ۱۰۰ msدارد، باید منتظر بماند تا ۱ تمام شود مثلاً ۲s

این یعنی درخواست سبک قربانی درخواست سنگین شده

و این، Head-of-Line Blocking است.

HOL Head of Line Blocking



مفاهیم

EWMA

EWMA یا Exponentially Weighted Moving Average به فارسی «میانگین متحرک با وزن نمایی» یک تکنیک آماری و محاسباتی است که برای ردیابی تغییرات نرم و سریع در داده‌های سری‌زمانی (مثل تأخیر، نرخ درخواست، طول صف، و ...) به کار می‌رود.

این روش به داده‌های جدید وزن بیشتری می‌دهد و به داده‌های قدیمی‌تر وزن کمتری، ولی آن‌ها را به طور کامل حذف نمی‌کند.

FCFS (First-Come-First-Serve)

FCFS یا First-Come-First-Serve به فارسی: «اولین واردشده، اولین سرویس‌گرفته» یک روش ساده و پایه‌ای برای زمان‌بندی Scheduling است که در بسیاری از سیستم‌ها از جمله:

صف‌های پردازنده CPU scheduling

صف‌های inference در GPU

سرویس‌دهی LLMها

به کار می‌رود.

FCFS (First-Come-First-Serve)

- مزایا و معایب FCFS

✅ مزیت‌ها	❌ معایب
ساده‌ترین الگوریتم ممکن	ممکن است **درخواست‌های سبک یا فوری را بی‌دلیل معطل کند**
بدون نیاز به تحلیل یا اولویت	می‌شود **Head-of-Line Blocking** دچار
عادلانه از نظر ترتیب	را تضمین کند QoE یا QoS نمی‌تواند
مناسب در سیستم‌های سبک	بالا ایجاد می‌کند latency، در بار زیاد

جایگزین های FCFS

روش جایگزین	مزیت نسبت به FCFS
Shortest Job First (SJF)	اولویت به درخواست‌های سبک → کاهش متوسط latency
Priority Scheduling	بر اساس نوع درخواست یا کاربر تصمیم می‌گیرد
Multi-Level Feedback Queue (MLFQ)	ترکیبی از اولویت + تطبیق‌پذیری
Token-level scheduling (مثل Andes)	زمان‌بندی بر اساس توکن، نه درخواست کامل
Preemptive Scheduling	اجازه توقف و اجرای دیگر درخواست‌ها در میانه راه

Coefficient of Variation (CV)

Coefficient of Variation (CV) یا به فارسی ضریب تغییرات، یک معیار آماری برای اندازه‌گیری **پراکندگی نسبی داده‌ها نسبت به مقدار میانگین است.**

این معیار به ما می‌گوید که داده‌ها چقدر نسبت به میانگینشان متغیر هستند، و معمولاً به صورت درصدی بیان می‌شود.

Coefficient of Variation (CV)

- تفسیر مقدار CV

مقدار CV	تفسیر
کم (مثلاً < 0.1)	داده‌ها نزدیک به میانگین هستند (پراکندگی کم)
متوسط (مثلاً 0.1–0.3)	پراکندگی متوسط
زیاد (> 0.3)	داده‌ها به‌شدت متغیر و پراکنده‌اند

CV برای latency / response time

اگر CV برای latency پایین باشد ← رفتار مدل پیش‌بینی‌پذیر است

اگر CV بالا باشد ← سیستم در شرایط مختلف رفتار ناپایدار دارد

← ممکن است نیاز به بازطراحی scheduler یا batching باشد

Coefficient of Variation (CV)

CV برای batch size یا توکن در درخواست‌ها

اگر CVبرای تعداد توکن‌ها در هر درخواست خیلی بالا باشد

→ یعنی بار کاری بسیار غیر یکنواخت است

→ سیستم باید از token-level یا priority scheduling استفاده کند تا از HoL blockingجلوگیری شود

CV برای GPU throughput

پایین بودن CV نشان‌دهنده ثبات و بهره‌وری خوب GPU است

بالا بودن CV ممکن است نشانه سوء استفاده منابع یا نوسانات بار باشد

بررسی مقاله سوم

Hierarchical Autoscaling for Large Language Model Serving with Chiron

مقیاس‌پذیری سلسله‌مراتبی برای سرویس‌دهی مدل‌های زبان بزرگ با استفاده از Chiron

کلیات مقاله

این مقاله به Chiron اشاره دارد که یک سیستم autoscaling برای سرویس‌دهی مدل‌های زبانی بزرگ LLM است.

هدف این مقاله بهبود عملکرد سیستم‌های سرویس‌دهی LLM از طریق مقیاس‌گذاری خودکار سلسله‌مراتبی است.

این سیستم برای درخواست‌های تعاملی و دسته‌ای به طور مؤثری منابع را مقیاس‌گذاری می‌کند و تلاش می‌کند تا SLO ها Service Level Objectives را در شرایط بار بالا حفظ کند.

نکات کلیدی

۱- دسته‌بندی درخواست‌ها:

درخواست‌های تعاملی: نیاز به پاسخ‌دهی سریع دارند (در عرض چند ثانیه).

درخواست‌های دسته‌ای: می‌توانند زمان بیشتری بگیرند (دقیقه‌ها تا ساعت‌ها).

Chiron این دو نوع درخواست (Interactive و Batch) را به‌طور جداگانه زمان‌بندی و مقیاس‌گذاری می‌کند، به‌طوری که منابع برای درخواست‌های تعاملی به سرعت تخصیص داده می‌شود، در حالی که درخواست‌های دسته‌ای می‌توانند زمان بیشتری بگیرند و منابع بیشتری را اشغال کنند.

نکات کلیدی

۱- دسته‌بندی درخواست‌ها:

درخواست‌های تعاملی: نیاز به پاسخ‌دهی سریع دارند (در عرض چند ثانیه).

درخواست‌های دسته‌ای: می‌توانند زمان بیشتری بگیرند (دقیقه‌ها تا ساعت‌ها).

Chiron این دو نوع درخواست (Interactive و Batch) را به‌طور جداگانه زمان‌بندی و مقیاس‌گذاری می‌کند، به‌طوری که منابع برای درخواست‌های تعاملی به سرعت تخصیص داده می‌شود، در حالی که درخواست‌های دسته‌ای می‌توانند زمان بیشتری بگیرند و منابع بیشتری را اشغال کنند.

حاالا در ادامه میگویم این دسته بندی ها چگونه صورت میگیرد:

نحوه دسته بندی Batching

۱- دسته‌بندی بر اساس نوع درخواست Interactive vs Batch

۲- دسته‌بندی بر اساس اولویت Priority-Based Scheduling

اولویت	توضیح
اولویت بالا	درخواست‌هایی که برای کاربران مهم‌تر هستند یا SLA سخت‌گیرانه‌تری دارند (مثلاً درخواست‌های مشتریان با اشتراک گران‌قیمت)
اولویت پایین	درخواست‌هایی که زمان پاسخ‌دهی طولانی‌تری می‌پذیرند یا کاربرانی که اولویت کمتری دارند

۳- دسته‌بندی بر اساس منابع مورد نیاز Resource-Based Classification

نوع درخواست	منابع مورد نیاز	توضیح
پردازش سنگین	GPU یا TPU	درخواست‌هایی که نیاز به پردازش پیچیده دارند، مانند Decoder در LLMها
پردازش سبک	CPU	درخواست‌هایی که نیاز به پردازش ساده دارند، مانند Prefill یا توکن‌سازی ابتدایی

۴- دسته‌بندی بر اساس تأخیر و طول مدت پردازش Latency-Driven Classification: Real-time یا Batchable

چگونه Chiron دسته‌بندی را مدیریت می‌کند؟

برای دسته‌بندی مؤثر، Chiron از الگوریتم‌های autoscaling و backpressure استفاده می‌کند تا درخواست‌ها به‌طور مؤثر در سطح محلی batch size و سطح سراسری instances مدیریت شوند.

Backpressure:

در صورت بالا بودن بار سیستم، Chiron می‌تواند درخواست‌های کم‌اولویت را به طور موقت متوقف کند یا رد کند تا منابع به درخواست‌های اولویت‌دار اختصاص یابند.

Autoscaling:

Chiron به‌طور دینامیک تعداد نمونه‌ها یا مقیاس‌گذاری منابع را برای درخواست‌های دسته‌ای و تعاملی تنظیم می‌کند.

نکات کلیدی

۲- مشکلات سیستم‌های قبلی:

۱- Overestimation of backpressure: برخی از سیستم‌های قبلی over-scaling می‌کنند که باعث استفاده ناکارآمد از منابع می‌شود.

۲- HoL (Head-of-Line) Blocking: برخی از سیستم‌ها از FIFO (First-Come-First-Serve) استفاده می‌کنند که باعث مسدود شدن درخواست‌های سبک به دلیل درخواست‌های سنگین می‌شود.

Overestimation of Backpressure

وقتی گفته می‌شود که overestimation of backpressure اتفاق می‌افتد، منظور این است که:

سیستم پیش‌بینی می‌کند که بار بیشتر از آنچه که هست است و به اشتباه منابع را زودتر از زمان لازم محدود یا رد می‌کند.

این می‌تواند منجر به اتلاف منابع یا پاسخ‌دهی به درخواست‌های غیرضروری شود.

به عبارت دیگر، سیستم زیاد از حد محتاطانه عمل می‌کند.

Overestimation of Backpressure

مشکل	توضیح
❌ اتلاف منابع	را پیش‌ازموعد از دست بدهد و نتواند از ظرفیت کامل خود (GPU مثلاً) سیستم ممکن است منابع استفاده کند.
⚠️ پاسخ‌دهی نامناسب	درخواست‌ها به‌طور غیرضروری رد می‌شوند یا تأخیر ایجاد می‌شود، حتی اگر سیستم به اندازه کافی توانایی پاسخ‌دهی دارد.
⬅️ END کاهش throughput	عملکرد کلی سیستم کاهش می‌یابد زیرا بیشتر درخواست‌ها رد می‌شوند، حتی در شرایطی که می‌شد پردازش شوند.
⬅️ END عدم مقیاس‌پذیری	سیستم ممکن است نتواند به درستی مقیاس‌پذیری پویا را اعمال کند زیرا بارهای احتمالی بیش از حد پیش‌بینی می‌شوند.

پیشبینی بار در این مقاله

در مقاله Chiron برای پیش‌بینی بار از روشی به نام Estimating Batch Backpressure (BBP) استفاده شده است.

در این روش، زمان انتظار درخواست‌ها در صف‌ها queue برای گروه‌های مختلف درخواست پیش‌بینی می‌شود تا مشخص شود که آیا زمان انتظار درخواست‌ها به TTFT SLO نزدیک است یا نه.

این پیش‌بینی به‌صورت آماری و با استفاده از مدل‌های QLM انجام می‌شود.

پیشبینی بار در این مقاله

۱- **انتظار صف درخواست‌ها Queue Waiting Time**: این زمان از تعداد توکن‌های باقی‌مانده در صف و throughput توکن که به‌صورت ثابت فرض می‌شود، محاسبه می‌شود.

۲- **استفاده از قانون حد مرکزی Central Limit Theorem**: از این قانون برای پیش‌بینی توزیع زمان انتظار و محاسبه دقیق‌تر برای تعداد درخواست‌هایی که احتمالاً به SLO ها نخواهند رسید استفاده می‌شود.

۳- **پیش‌بینی تعداد درخواست‌های با تأخیر بالا BBP**: تعداد درخواست‌هایی که زمان انتظارشان به حد SLO می‌رسد به عنوان BBP محاسبه می‌شود.

این پیش‌بینی به سیستم کمک می‌کند تا مقیاس‌گذاری منابع به طور دقیق‌تری انجام گیرد و از overprovisioning یا underutilization جلوگیری شود.

در مجموع، پیش‌بینی با استفاده از داده‌های تاریخی و آمارهای صف به سیستم اجازه می‌دهد تا تصمیمات بهینه‌تری در زمینه تخصیص منابع و مقیاس‌گذاری بگیرد.

نتیجه مقاله

در مقاله Chiron ، سیستم توانست با استفاده از مقیاس‌گذاری پویا و backpressure سلسله‌مراتبی تا ٪۳۰۰ throughput را افزایش دهد و زمان تأخیر را کاهش دهد.

همچنین توانست ٪۷۰ منابع GPU را صرفه‌جویی کند و از overprovisioning جلوگیری کند.

با پیش‌بینی دقیق‌تر بار، منابع به‌طور بهینه تخصیص داده شدند و SLOها حفظ شدند.

این سیستم عملکرد بهتری نسبت به دیگر سیستم‌ها مانند Clipper و vLLM نشان داد.

در نهایت، Chiron توانست از اتلاف منابع جلوگیری کند و بهبود چشمگیری در پاسخ‌دهی سریع به درخواست‌ها ایجاد کند.

نقاط ضعف

۱. بارهای ناگهانی و غیرقابل پیش‌بینی

Chiron بیشتر بر اساس پیش‌بینی بار با استفاده از داده‌های تاریخی و الگوریتم‌های پیش‌بینی عمل می‌کند.

در شرایطی که بار ناگهانی و غیرقابل پیش‌بینی باشد (مثلاً در زمان‌های پیک یا تغییرات غیرمنتظره)، این سیستم ممکن است کارایی پایین‌تری داشته باشد چون پیش‌بینی دقیق در این شرایط سخت است.

نقاط ضعف

۲- مدل‌های بسیار بزرگ با نیاز به منابع زیاد

اگر سیستم‌های LLM بسیار بزرگ با تعداد پارامترهای بالا و نیاز به پردازش موازی زیاد (مثل مدل‌های چندین ترلیون پارامتر) در نظر گرفته شوند، ممکن است Chiron در مدیریت مقیاس‌گذاری افقی و منابع با چالش‌های بیشتری روبه‌رو شود.

سیستم‌هایی با چنین مقیاس‌هایی ممکن است نیاز به راهکارهای خاص و متفاوت برای مدیریت فشار بار و تخصیص منابع داشته باشند.

نقاط ضعف

۳- سیستم‌هایی با نیاز به پردازش بلادرنگ

در درخواست‌های تعاملی که نیاز به پاسخ‌دهی بلادرنگ real-time دارند، Chiron ممکن است با مشکل مواجه شود.

چرا که سیستم بیشتر به درخواست‌های دسته‌ای توجه دارد و پردازش‌های فوری می‌تواند باعث تاخیر در زمان پاسخ‌دهی شود.

در چنین سناریوهایی زمان‌بندی پویا و تخصیص منابع ممکن است کارایی لازم را نداشته باشد.

نقاط ضعف

۴- بارهای متغیر و پیچیده که به راحتی پیش‌بینی نمی‌شوند

Chiron به پیش‌بینی دقیق بار و تنظیم منابع بر اساس آن متکی است.

در شرایطی که بار به صورت غیرخطی یا بسیار متغیر است (مثلاً در شرایطی که تغییرات شدید و غیرقابل پیش‌بینی وجود داشته باشد)، پیش‌بینی‌ها می‌توانند دقت پایین‌تری داشته باشند و سیستم ممکن است منابع را به‌طور بهینه تخصیص ندهد.

نقاط ضعف

در مقاله Chiron برای پیش‌بینی بار و فشار سیستم از یادگیری ماشین به‌طور مستقیم استفاده نشده است.

به جای آن، سیستم از مدل‌های آماری و توزیع‌های احتمالاتی برای پیش‌بینی بار و تخصیص منابع استفاده می‌کند.

به طور خاص، در این مقاله بیشتر بر استفاده از پیش‌بینی بر اساس تاریخچه بار سیستم و استفاده از قانون حد مرکزی Central Limit Theorem برای تخمین زمان انتظار درخواست‌ها در صف‌ها و پیش‌بینی فشار بار (backpressure) تمرکز شده است.

نقاط ضعف

۱- **پیش‌بینی با استفاده از داده‌های تاریخی**: مقاله بیشتر به تحلیل و پیش‌بینی بار سیستم بر اساس داده‌های تاریخی پرداخته است و از مدل‌های آماری برای تخمین زمان انتظار و تخصیص منابع استفاده کرده است.

۲- **مدل‌های آماری ساده**: به جای استفاده از مدل‌های پیچیده یادگیری ماشین، از روش‌هایی مانند توزیع‌های نرمال و Central Limit Theorem استفاده شده است تا از پیچیدگی‌های اضافی اجتناب شود و سیستم به‌طور سریع و کارآمد به پیش‌بینی فشار بار بپردازد.

۳- **تمرکز بر عملکرد بهینه در مقیاس بزرگ**: یکی از اهداف مقاله Chiron این بوده که یک سیستم مقیاس‌پذیر و سریع ایجاد کند که بتواند منابع را به‌طور پویا تخصیص دهد. استفاده از یادگیری ماشین ممکن است پیچیدگی و زمان پردازش را افزایش دهد.

جمع بندی

جمع بندی

مقدمه

سیستم‌های LLM مانند GPT-3و GPT-4 به شدت در حال گسترش هستند و توانایی پردازش و تولید متن با دقت بالا را دارند.

این سیستم‌ها نیاز به مدیریت منابع بهینه برای پردازش درخواست‌ها دارند، به خصوص زمانی که بار زیاد و تعداد زیاد درخواست‌ها وارد سیستم می‌شود.

در این تحقیق، Chironبه عنوان یک سیستم مقیاس‌گذاری خودکار ارائه شده که توانسته عملکرد سیستم‌های LLM را با حفظ SLOها و مقیاس‌گذاری پویا بهبود بخشد.

جمع بندی

ادبیات

چالش‌های مدیریت منابع در سیستم‌های LLM :شامل زمان‌بندی درخواست‌ها، تخصیص منابع GPU، پیش‌بینی بار و فشار سیستم.

سیستم‌های مشابه : مانند Clipperو vLLM که به مدیریت منابع و مقیاس‌گذاری پرداخته‌اند.

محدودیت‌ها: بسیاری از این سیستم‌ها نمی‌توانند پیش‌بینی دقیقی از بار سیستم داشته باشند و آستانه‌های backpressureآن‌ها ثابت است که باعث کاهش کیفیت خدمات می‌شود.

Chiron به‌عنوان یک سیستم نوآورانه که از backpressure سلسله‌مراتبی و مقیاس‌گذاری پویا برای تخصیص منابع بهینه استفاده می‌کند، معرفی می‌شود.

جمع بندی

هدف

هدف کلی تحقیق: هدف از این تحقیق ارزیابی و بهبود عملکرد سیستم‌های LLM از طریق مقیاس‌گذاری پویا و مدیریت دقیق منابع با استفاده از Chironاست.

اهداف فرعی:

تحلیل و بهبود پیش‌بینی بار در سیستم‌های LLM .

توسعه روش‌های زمان‌بندی پویا برای درخواست‌های تعاملی و دسته‌ای.

مقیاس‌گذاری منابع به‌طور بهینه با استفاده از backpressure سلسله‌مراتبی.

روش شناسی

استفاده از داده‌های تاریخی: برای پیش‌بینی بار و فشار سیستم، از داده‌های تاریخی سیستم استفاده خواهد شد.

مدل‌های پیش‌بینی آماری: استفاده از قانون حد مرکزی و توزیع‌های احتمالاتی برای پیش‌بینی زمان انتظار و میزان بار سیستم.

زمان‌بندی پویا: تخصیص منابع به‌طور داینامیک بر اساس نوع درخواست (تعامل‌گرا یا دسته‌ای) و وضعیت بار.

استفاده از الگوریتم‌های یادگیری ماشین (در صورت نیاز): مانند مدل‌های رگرسیونی یا یادگیری تقویتی برای بهبود پیش‌بینی بار و تخصیص منابع.

پیاده‌سازی و شبیه‌سازی: شبیه‌سازی بار واقعی و بررسی عملکرد Chironدر مقایسه با سیستم‌های مشابه.

روش شناسی

استفاده از داده‌های تاریخی: برای پیش‌بینی بار و فشار سیستم، از داده‌های تاریخی سیستم استفاده خواهد شد.

مدل‌های پیش‌بینی آماری: استفاده از قانون حد مرکزی و توزیع‌های احتمالاتی برای پیش‌بینی زمان انتظار و میزان بار سیستم.

زمان‌بندی پویا: تخصیص منابع به‌طور داینامیک بر اساس نوع درخواست (تعامل‌گرا یا دسته‌ای) و وضعیت بار.

استفاده از الگوریتم‌های یادگیری ماشین (در صورت نیاز): مانند مدل‌های رگرسیونی یا یادگیری تقویتی برای بهبود پیش‌بینی بار و تخصیص منابع.

پیاده‌سازی و شبیه‌سازی: شبیه‌سازی بار واقعی و بررسی عملکرد Chiron در مقایسه با سیستم‌های مشابه.

با تشکر از همراهی شما

محمد سعید صفایی صادق

www.SaeidSafaei.ir