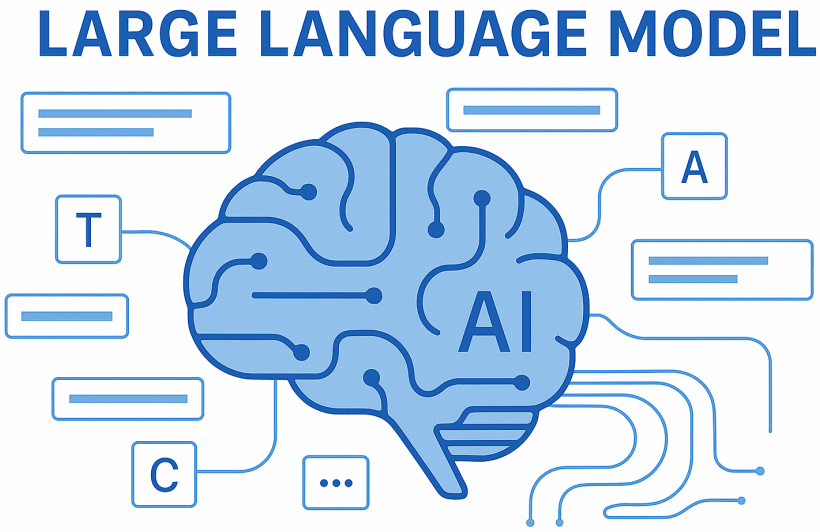
مقیاس بندی خودکار
سرویس دهی مدل‌های زبانی بزرگ



پروپوزال

۱

محمد سعید صفایی صادق

مفاهیم پایه

محاسبات ابری Cloud Computing

محاسبات ابری به معنای استفاده از منابع سخت‌افزاری و نرم‌افزاری (مثل سرورها، ذخیره‌سازی، دیتابیس‌ها) از طریق اینترنت است. شما نیازی به نگهداری یا مدیریت سرورهای فیزیکی ندارید و فقط برای استفاده از این منابع هزینه می‌پردازید.

مثال: فرض کنید شما یک اپلیکیشن موبایل دارید که نیاز به پردازش داده‌های زیادی دارد. به جای اینکه یک سرور فیزیکی بخرید و راه‌اندازی کنید، می‌توانید از سرویس‌هایی مانند Amazon Web Services (AWS) یا Google Cloud استفاده کنید و منابع پردازشی را بر اساس نیاز خود اجاره کنید.

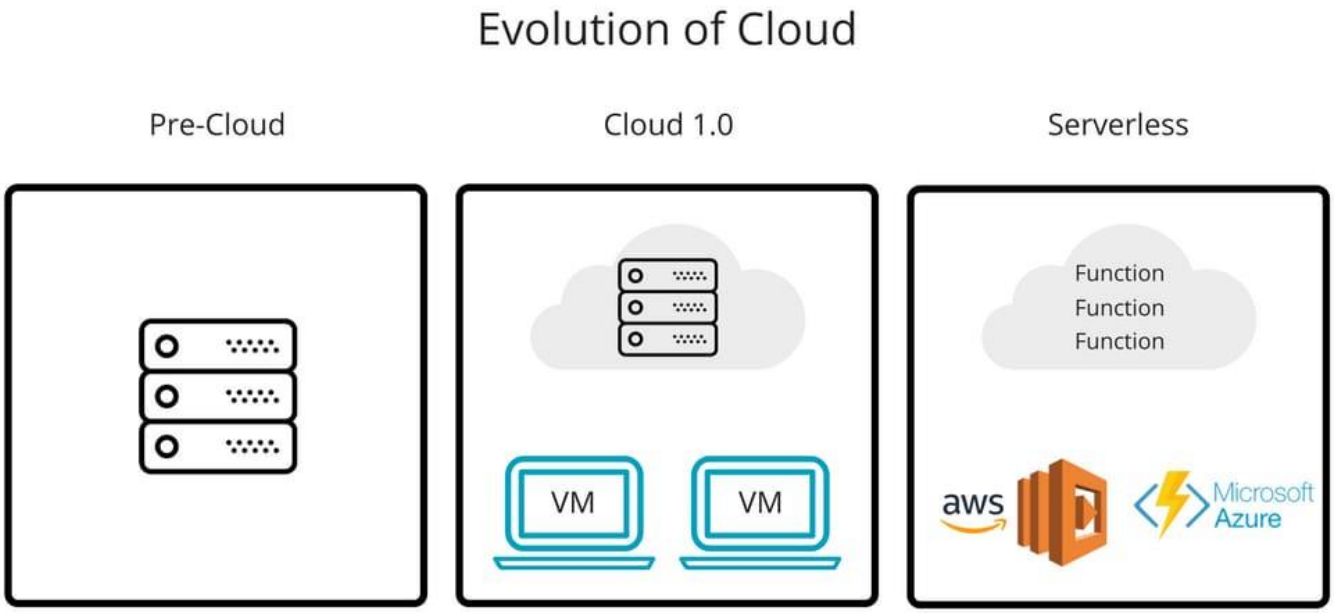
سرورلس Serverless

سرورلس یک مدل محاسباتی است که در آن شما نیازی به مدیریت و نگهداری سرورها ندارید.

سرویس‌دهنده ابری مثل AWS Lambda به طور خودکار منابع را تخصیص می‌دهد و شما تنها برای استفاده از این منابع هزینه می‌پردازید.

مثال: شما می‌خواهید یک تابع ساده در برنامه خود اجرا کنید که هر بار که یک پیام جدید در سیستم ارسال شد، یک ایمیل ارسال کند. با استفاده از AWS Lambda، این تابع به طور خودکار اجرا می‌شود و شما نیازی به مدیریت سرور ندارید، فقط زمانی که تابع اجرا می‌شود هزینه پرداخت می‌کنید.

تفاوت رایانش ابری و سرورلس



رایانش ابری به شما امکان می‌دهد که منابع ابری را مدیریت کنید و برای استفاده از آن‌ها هزینه بدهید.

سرورلس به شما این امکان را می‌دهد که بدون نیاز به مدیریت سرورها فقط برای اجرای کد و منابع استفاده‌شده هزینه کنید.

تفاوت رایانش ابری و سرورلس

۱- مدیریت سرور:

در رایانش ابری، شما مسئول انتخاب و مدیریت سرورها هستید. باید سرورها را تنظیم و مقیاس‌دهی کنید تا با نیازهای خود هماهنگ شوند.

در سرورلس، نیازی به مدیریت سرور ندارید. تمامی زیرساخت‌ها و منابع توسط سرویس‌دهنده ابری به صورت خودکار مدیریت می‌شود.

تفاوت رایانش ابری و سرورلس

۲- مقیاس‌پذیری:

در رایانش ابری، شما باید مقیاس منابع را به صورت دستی تنظیم کنید. یعنی اگر نیاز به منابع بیشتری دارید، باید خودتان سرورهای بیشتری راه‌اندازی کنید.

در سرورلس، مقیاس‌پذیری به طور خودکار انجام می‌شود. سرویس‌دهنده ابری به صورت خودکار تعداد سرورها یا منابع لازم را بر اساس حجم درخواست‌ها تنظیم می‌کند.

تفاوت رایانش ابری و سرورلس

۳- پرداخت به ازای استفاده:

در رایانش ابری، شما برای استفاده از منابع (مثل سرورها) هزینه می‌پردازید، حتی اگر سرورها در حالت بیکار باشند.

در سرورلس، شما فقط برای زمانی که کد شما اجرا می‌شود، هزینه می‌پردازید. بنابراین، هزینه‌ها معمولاً کمتر از مدل رایانش ابری سنتی است، زیرا سرورها تنها زمانی که لازم است فعال می‌شوند.

تفاوت رایانش ابری و سرورلس

۴- مناسب برای چه نوع برنامه‌هایی؟

رایانش ابری بیشتر برای برنامه‌هایی مناسب است که نیاز به منابع ثابت و مدیریت‌شده دارند. این نوع سرویس برای سیستم‌های پیچیده‌تر و بزرگ‌تر که به کنترل دقیق‌تری بر زیرساخت‌ها نیاز دارند، مناسب است.

سرورلس بیشتر برای توابع کوچک و مقیاس‌پذیر که نیاز به مدیریت سرور ندارند، مناسب است. مثلاً اگر شما فقط به پردازش داده‌های ساده و درخواست‌های API نیاز دارید، سرورلس بهترین گزینه است.

تفاوت رایانش ابری و سرورلس

رایانش ابری: شما یک وب‌سایت دارید که نیاز به یک سرور اختصاصی برای اجرای پایگاه داده و اپلیکیشن دارد. شما سرور EC2 را در AWS انتخاب می‌کنید و برای آن تنظیمات خاصی انجام می‌دهید (مثلاً از حافظه بیشتر یا پردازشگر سریع‌تر استفاده می‌کنید).

سرورلس: شما یک وب‌سایت دارید که فقط به تابعی نیاز دارد که هر بار یک کاربر وارد می‌شود، یک ایمیل ارسال کند. شما از AWS Lambda استفاده می‌کنید که به طور خودکار منابع مورد نیاز برای اجرای تابع شما را تخصیص می‌دهد و فقط زمانی که تابع شما اجرا می‌شود، هزینه می‌پردازید.

مدل‌های زبانی بزرگ LLM

مدل‌های زبانی بزرگ، مدل‌های یادگیری ماشینی هستند که برای پردازش زبان طبیعی طراحی شده‌اند.

این مدل‌ها می‌توانند متون پیچیده را تجزیه و تحلیل کنند، پاسخ دهند و یا حتی متن جدید تولید کنند. این مدل‌ها معمولاً بسیار بزرگ هستند و می‌توانند صدها گیگابایت اطلاعات داشته باشند.

مثال: یکی از مدل‌های معروف GPT-3 است. این مدل می‌تواند به سوالات شما پاسخ دهد، متن‌های مختلف را ترجمه کند یا حتی داستان بنویسد.

مدل‌های زبانی بزرگ LLM

برای درک مفهوم مدل‌های زبانی بزرگ LLM و نحوه عملکرد آن‌ها، لازم است که با برخی مفاهیم پایه‌ای در زمینه یادگیری ماشین و پردازش زبان طبیعی NLP آشنا شوید.

این مفاهیم به شما کمک می‌کنند تا بهتر درک کنید که مدل‌های زبانی بزرگ چگونه کار می‌کنند و چرا برای پردازش و تولید متن‌های پیچیده به این اندازه پیشرفته هستند.

در ادامه مفاهیم کلیدی که باید بدانید آمده است

یادگیری ماشین Machine Learning

یادگیری ماشین یکی از شاخه‌های هوش مصنوعی است که به سیستم‌ها این امکان را می‌دهد که از داده‌ها یاد بگیرند و به‌طور خودکار بدون نیاز به برنامه‌نویسی دقیق، تصمیمات و پیش‌بینی‌هایی انجام دهند.

مثال: به‌جای اینکه به یک کامپیوتر دقیقاً بگوییم که چگونه یک تصویر را شناسایی کند، از داده‌های آموزشی استفاده می‌کنیم و به سیستم می‌گوییم که خود از آن داده‌ها الگوها را یاد بگیرد.

پردازش زبان طبیعی

Natural Language Processing - NLP

شاخه‌ای از هوش مصنوعی است که به کامپیوترها کمک می‌کند زبان انسانی را درک کرده، تحلیل کنند و با آن تعامل داشته باشند.

این شامل کارهایی مانند ترجمه زبان‌ها، خلاصه‌سازی متن، شناسایی موجودیت‌ها و پاسخ به سوالات می‌شود.

مثال: وقتی از یک دستیار صوتی مانند سیری یا الکسای آمازون استفاده می‌کنید، این سیستم‌ها از پردازش زبان طبیعی برای درک دستورات و پاسخ‌دهی به آن‌ها استفاده می‌کنند.

شبکه عصبی مصنوعی

Artificial Neural Network – ANN

مدل‌هایی هستند که الهام گرفته از ساختار مغز انسان ساخته شده‌اند.

این شبکه‌ها از مجموعه‌ای از گره‌ها (یا نورون‌ها) تشکیل شده‌اند که می‌توانند داده‌ها را پردازش کنند و از طریق یادگیری، خود را بهبود دهند.

مثال: زمانی که شما از یک شبکه عصبی برای شناسایی تصویر استفاده می‌کنید، این شبکه از داده‌های تصویری برای یادگیری ویژگی‌های مهم هر تصویر و طبقه‌بندی آن استفاده می‌کند.

مدل‌های ترنسفورمر Transformer Models

مدل‌های ترنسفورمر یکی از معماری‌های جدید و قدرتمند در یادگیری ماشین و پردازش زبان طبیعی هستند.

این مدل‌ها به طور خاص برای پردازش زبان‌های طبیعی طراحی شده‌اند و ویژگی‌های مهمی مانند توجه Attention را برای پردازش داده‌های متوالی ارائه می‌دهند.

مثال: مدل‌هایی مانند GPT-3 و BERT از معماری ترنسفورمر استفاده می‌کنند. این معماری به مدل‌ها کمک می‌کند تا وابستگی‌های طولانی‌مدت در متن‌های پیچیده را بهتر درک کنند.

وابستگی های طولانی مدت

وابستگی‌های طولانی‌مدت Long-Term Dependencies در پردازش زبان طبیعی NLP به ارتباطات و روابطی اطلاق می‌شود که در متن‌های طولانی‌تر وجود دارند و برای درک کامل معنی و مفهوم جمله یا متن، باید این ارتباطات را به درستی شبیه‌سازی کرد.

وابستگی های طولانی مدت

فرض کنید جمله‌ای به این صورت داریم:

"وقتی علی به خانه آمد، تصمیم گرفت که در اتاق نشیمن استراحت کند."

در این جمله، برای اینکه معنی کامل و صحیح جمله را درک کنیم، باید به این نکته توجه داشته باشیم که "علی" که در ابتدا ذکر شده، همان کسی است که "تصمیم گرفت" و "در اتاق نشیمن استراحت کند". این وابستگی بین بخش‌های مختلف جمله (یعنی رابطه علی با تصمیمش) به‌طور مستقیم در کلمات ابتدایی و انتهایی جمله وجود دارد.

این نوع ارتباط‌ها به‌عنوان وابستگی‌های طولانی‌مدت شناخته می‌شوند.

وابستگی های طولانی مدت

فرض کنید جمله‌ای مانند زیر داریم:

"مریم از دوستش خواسته بود که کتابی را که در سفر به فرانسه خریده بود، برایش بیاورد."

در این جمله، برای اینکه معنای صحیح را درک کنیم، باید رابطه بین "کتابی" که "در سفر به فرانسه خریده بود" و "مریم" که از دوستش خواسته بود آن کتاب را بیاورد، به‌درستی درک شود.

این نوع وابستگی‌ها معمولاً در متن‌های طولانی‌تر و پیچیده‌تر اتفاق می‌افتد.

وابستگی های طولانی مدت

مدل‌های ترنسفورمر مانند GPT-3 و BERT به‌گونه‌ای طراحی شده‌اند که قادرند وابستگی‌های طولانی‌مدت را به‌طور مؤثر شبیه‌سازی کنند.

این معماری از مکانیزم توجه Attention Mechanism استفاده می‌کند که به مدل این امکان را می‌دهد که هنگام پردازش هر کلمه، به تمام کلمات قبلی در جمله یا متن توجه کند، نه فقط کلمات نزدیک به آن.

این باعث می‌شود که مدل بتواند ارتباطات و وابستگی‌های طولانی‌مدت را در متن‌های پیچیده بهتر درک کند.

به طور ساده‌تر، ترنسفورمرها به‌طور خودکار از تمام اطلاعات متن برای فهمیدن معنی و ارتباط بین قسمت‌های مختلف استفاده می‌کنند، بدون اینکه نیاز به پردازش خطی یا محدودیت‌های دیگر داشته باشند.

پردازش داده‌های متنی Text Data Processing

برای اینکه یک مدل زبانی مانند GPT-3 بتواند متنی را تجزیه و تحلیل کند، ابتدا باید داده‌های متنی پردازش شوند.

این پردازش‌ها شامل تبدیل کلمات به فرمت‌هایی است که کامپیوترها بتوانند آن‌ها را درک کنند (مثلاً از طریق تبدیل به توکن‌ها).

مثال: قبل از اینکه یک مدل زبان طبیعی مانند GPT-3 شروع به یادگیری کند، تمام متونی که وارد می‌شود به قسمت‌های کوچکتری به نام توکن تبدیل می‌شوند.

این کار به مدل کمک می‌کند تا کلمات و جملات را به‌طور مؤثر پردازش کند.

توکن‌سازی Tokenization

توکن‌سازی فرایندی است که در آن یک متن بزرگ به بخش‌های کوچکتر (توکن‌ها) تقسیم می‌شود.

این توکن‌ها می‌توانند کلمات، نشانه‌ها یا حتی بخش‌های کوچکتر از کلمات باشند.

مثال: فرض کنید جمله‌ای به این شکل داریم: "مدل‌های زبانی بزرگ برای پردازش زبان طبیعی طراحی شده‌اند." این جمله می‌تواند به توکن‌های زیر تقسیم شود:

"مدل‌های" "زبانی" "بزرگ" "برای" "پردازش" "زبان" "طبیعی" "طراحی" "شده‌اند"

توکن‌سازی Tokenization

نکته:

توکن‌سازی Tokenization کار ترنسفورمر نیست، بلکه یک مرحله پیش‌پردازش است که قبل از ورود داده‌ها به مدل‌های ترنسفورمر (مثل GPT-3 یا BERT) انجام می‌شود.

توکن‌سازی در واقع فرایندی است که متن طولانی را به واحدهای کوچکتری به نام توکن‌ها تقسیم می‌کند تا مدل‌ها بتوانند آن‌ها را پردازش کنند.

ترنسفورمرها سپس به این توکن‌ها پرداخته و ارتباطات بین آن‌ها را برای درک و تولید متن پردازش می‌کنند.

توکن‌سازی Tokenization

نکته:

توکن‌سازی Tokenization کار ترنسفورمر نیست، بلکه یک مرحله پیش‌پردازش است که قبل از ورود داده‌ها به مدل‌های ترنسفورمر (مثل GPT-3 یا BERT) انجام می‌شود.

توکن‌سازی در واقع فرایندی است که متن طولانی را به واحدهای کوچکتری به نام توکن‌ها تقسیم می‌کند تا مدل‌ها بتوانند آن‌ها را پردازش کنند.

ترنسفورمرها سپس به این توکن‌ها پرداخته و ارتباطات بین آن‌ها را برای درک و تولید متن با استفاده از مکانیزم توجه پردازش می‌کنند.

توجه یعنی چی؟

تصور کنید که دارید یک جمله طولانی می‌خوانید. در این جمله، برخی کلمات به یکدیگر ارتباط دارند، مثلاً در جمله "او به کتابخانه رفت زیرا نیاز به کتاب داشت"، کلمه "نیاز" بیشتر به "کتاب" ارتباط دارد تا به "کتابخانه".

مدل ترنسفورمر به این صورت عمل می‌کند:

وقتی که مدل در حال پردازش کلمه "نیاز" است، به کلمات دیگر جمله نگاه می‌کند و بررسی می‌کند که کدام کلمات برای فهمیدن معنی کلمه "نیاز" اهمیت دارند.

این که مدل به کدام کلمات بیشتر توجه می‌کند، بستگی به وزن‌های توجه دارد. این وزن‌ها مشخص می‌کنند که کدام کلمات در جمله باید بیشتر مورد توجه قرار بگیرند.

مثال ساده برای توضیح وزن:

تصور کنید شما یک معلم هستید و در حال ارزیابی یک پروژه از دانش‌آموزان خود هستید.

برای برخی پروژه‌ها، بیشتر به خلاقیت اهمیت می‌دهید و برای پروژه‌های دیگر، بیشتر به دقت علمی توجه می‌کنید.

در این حالت، شما به خلاقیت وزن بیشتری می‌دهید تا دقت علمی.

یعنی اگر یک پروژه از نظر خلاقیت عالی باشد، حتی اگر دقت علمی کمتری داشته باشد، شما بیشتر به آن توجه می‌کنید.

توضیح دقیق‌تر با مکانیزم توجه:

در مکانیزم توجه، برای هر کلمه (توکن) که مدل در حال پردازش آن است، مدل به کلمات دیگر در جمله نگاه می‌کند و وزن‌های توجه را محاسبه می‌کند.

این وزن‌ها نشان می‌دهند که هر کلمه چقدر باید بر کلمات دیگر تأثیر بگذارد یا به آن‌ها توجه کند.

برای مثال، اگر شما جمله‌ای مانند "او به کتابخانه رفت زیرا نیاز به کتاب داشت" داشته باشید، مدل باید ببیند که کلمه "نیاز" بیشتر به "کتاب" مربوط است تا به "کتابخانه". برای این منظور، مدل به "کتاب" وزن بالاتری می‌دهد و به "کتابخانه" وزن کمتری می‌دهد.

چطور مدل ترنسفورمر این کار را انجام می‌دهد؟

برای هر کلمه (توکن)، مدل به‌طور همزمان به تمام کلمات دیگر در جمله توجه می‌کند.

مدل به کمک سه بردار اصلی که برای هر کلمه ایجاد می‌کند (که به نام Query، Key و Value شناخته می‌شوند)، محاسبه می‌کند که هر کلمه باید چقدر به کلمات دیگر توجه کند.

در نهایت، مدل یک نمایش جدید از هر کلمه تولید می‌کند که شامل اطلاعات مهم از سایر کلمات در جمله است.

توجه خود Self-Attention

در ترنسفورمرها، از توجه خود Self-Attention برای درک ارتباطات بین توکن‌ها در همان دنباله استفاده می‌شود.

این یعنی هر توکن در یک دنباله به سایر توکن‌های آن دنباله توجه می‌کند.

توجه خود Self-Attention

چطور کار می‌کند؟

۱- **ورودی‌ها**: ورودی‌های مدل ترنسفورمر، توکن‌هایی هستند که پس از توکن‌سازی به مدل داده می‌شوند.

۲- **سه بردار اصلی**: برای هر توکن، مدل سه بردار اصلی تولید می‌کند: Queryپرسش، Keyکلید و Valueمقدار.

این بردارها برای محاسبه وزن توجه بین توکن‌ها استفاده می‌شوند.

۳- **محاسبه وزن‌های توجه**: مدل با استفاده از این بردارها، میزان اهمیت یا وزن توجه هر توکن را نسبت به سایر توکن‌ها محاسبه می‌کند. برای این کار، مدل از ضرب داخلی بین بردارهای Query و Key استفاده می‌کند.

توجه خود Self-Attention

۴- **تولید نمایه‌ها**: پس از محاسبه وزن‌ها، مدل از آن‌ها برای ایجاد نمایه‌های جدید (توکن‌های پردازش شده) استفاده می‌کند که تمام اطلاعات مهم از توکن‌های دیگر را در خود دارند.

مثال ساده:

در جمله "او به کتابخانه رفت زیرا نیاز به کتاب داشت"، برای پردازش کلمه "نیاز"، مکانیزم توجه باید بفهمد که این کلمه بیشتر به "کتاب" مرتبط است تا به "کتابخانه". برای این کار، مدل محاسبه می‌کند که توکن "کتاب" چقدر باید به توکن "نیاز" توجه کند و از این طریق روابط بین توکن‌ها را درک می‌کند.

چند لایه توجه Multi-Head Attention

یکی از ویژگی‌های مهم ترنسفورمرها استفاده از چند لایه توجه Multi-Head Attention است.

این بدان معناست که مدل به‌طور همزمان چندین مجموعه از Query، Key و Valueرا برای محاسبه توجه در نظر می‌گیرد.

این باعث می‌شود که مدل بتواند از اطلاعات مختلف و ارتباطات متفاوت میان توکن‌ها استفاده کند.

به جای اینکه فقط یکبار برای هر توکن توجه محاسبه شود، مدل چندین بار این کار را انجام می‌دهد و سپس نتایج را ترکیب می‌کند تا بهترین نمایه برای هر توکن تولید شود.

چند لایه توجه Multi-Head Attention

مدل‌های ترنسفورمر به جای اینکه فقط یکبار برای هر کلمه تصمیم بگیرند که به کدام کلمات توجه کنند، چندین بار این کار را انجام می‌دهند.

به این فرآیند چند لایه توجه می‌گویند.

چندین لایه توجه باعث می‌شود که مدل از چندین زاویه مختلف به جمله نگاه کند و روابط متفاوت میان کلمات را بهتر درک کند.

مراحل توجه Attention Mechanism

فرض کنید جمله زیر داریم:

"او به کتابخانه رفت زیرا نیاز به کتاب داشت."

ما می‌خواهیم به کلمه "نیاز" توجه کنیم و ببینیم که این کلمه به کدام بخش‌های دیگر جمله مرتبط است.

مراحل توجه Attention Mechanism

مرحله اول: ایجاد سه بردار: Query، Keyو Value

برای هر کلمه در جمله، مدل سه بردار Query، Keyو Valueایجاد می‌کند.

Query پرسش: برای کلمه "نیاز" (که می‌خواهیم توجه کنیم)، مدل یک Query ایجاد می‌کند.

Key کلید: مدل برای سایر کلمات جمله مانند "او"، "کتابخانه"، "داشت" و غیره Key ایجاد می‌کند.

Value مقدار: برای هر کلمه در جمله، مدل یک Value ایجاد می‌کند که شامل اطلاعات مربوط به آن کلمه است.

مراحل توجه Attention Mechanism

مرحله دوم: محاسبه شباهت‌ها (مقیاس‌گذاری توجه)

حالا مدل باید محاسبه کند که کلمه "نیاز" چقدر باید به کلمات دیگر توجه کند. برای این کار:

مدل ضرب داخلی Query کلمه "نیاز" را با Key هر کلمه دیگر در جمله انجام می‌دهد.

این ضرب داخلی به ما یک عدد می‌دهد که نشان‌دهنده میزان شباهت یا ارتباط بین "نیاز" و سایر کلمات است.

مثال: ضرب داخلی Query نیاز و Key کتابخانه: اگر مدل تشخیص دهد که کلمه "نیاز" بیشتر به "کتاب" مربوط است، این عدد بالا خواهد بود.

مراحل توجه Attention Mechanism

یادگیری از طریق ارتباطات معنایی:

مدل از طریق یادگیری‌های پیچیده در شبکه‌های عصبی روابط معنایی میان کلمات را یاد می‌گیرد.

این به این معناست که مدل، با پردازش داده‌های مختلف، از هر کلمه و عبارت، ویژگی‌های معنایی استخراج می‌کند و می‌فهمد که کلمات در چه زمینه‌هایی به هم مرتبط هستند.

مثال: مدل متوجه می‌شود که وقتی کلمه "نیاز" به کلمه "کتاب" می‌رسد، معمولاً در جملاتی مانند "نیاز به کتاب" یا "کتاب مورد نیاز" آمده است، که نشان‌دهنده ارتباط معنایی میان این دو است.

مراحل توجه Attention Mechanism

مرحله سوم: نرمال‌سازی و محاسبه وزن‌ها

مدل سپس نتایج ضرب داخلی را نرمال‌سازی می‌کند (یعنی این که همه نتایج را به مقداری بین ۰ و ۱ می‌رساند) تا از آن‌ها وزن‌های توجه بسازد.

این وزن‌ها نشان می‌دهند که هر کلمه چقدر برای فهمیدن "نیاز" مهم است.

مثال:

فرض کنید که کلمه "کتاب" بیشترین ارتباط را با "نیاز" دارد، بنابراین وزن توجه به "کتاب" زیاد می‌شود.

کلمه "کتابخانه" ارتباط کمتری با "نیاز" دارد، بنابراین وزن توجه به آن کمتر می‌شود.

مراحل توجه Attention Mechanism

مرحله چهارم: محاسبه مقدار نهایی (Attention Score)

در این مرحله، مدل از وزن‌های توجه که در مرحله قبلی محاسبه کرده است، استفاده می‌کند تا مقدار نهایی (یا پاسخ) برای هر کلمه را محاسبه کند.

مدل به کلمات دیگر نگاه می‌کند و از وزن‌های توجه برای ترکیب Values آن‌ها استفاده می‌کند.

هر کلمه‌ای که وزن توجه بالاتری داشته باشد، تأثیر بیشتری در نتیجه نهایی خواهد داشت.

مثال:

اگر مدل متوجه شده باشد که کلمه "کتاب" با "نیاز" ارتباط زیادی دارد، مقدار نهایی Value برای "کتاب" بیشتر در محاسبات مدل لحاظ می‌شود.

مراحل توجه Attention Mechanism

مرحله پنجم: نتیجه نهایی

در نهایت، مدل تمام این محاسبات را انجام می‌دهد و نمایش جدیدی از کلمه "نیاز" تولید می‌کند که شامل اطلاعات از سایر کلمات (مثل "کتاب") است. این به مدل کمک می‌کند که مفهوم کامل‌تری از جمله درک کند.

نتیجه:

در این مثال، مدل "نیاز" را با توجه به ارتباط‌های آن با سایر کلمات (مانند "کتاب") درک می‌کند و اطلاعات مفیدی از بقیه جمله استخراج می‌کند تا مفهوم کلی جمله بهتر فهمیده شود.

پیش‌بینی توکن بعدی Next Token Prediction

مدل‌های زبانی بزرگ مانند GPT-3 از یک رویکرد پیش‌بینی توکن بعدی استفاده می‌کنند.

به این معنی که مدل‌ها برای تولید متن، توکن‌ها را یکی پس از دیگری پیش‌بینی می‌کنند.

مثال: وقتی شما از GPT-3 می‌خواهید که یک جمله را ادامه دهد، این مدل کلمات بعدی را پیش‌بینی می‌کند.

مثلاً اگر جمله‌ای شروع شود با "آسمان آبی است و"، مدل ممکن است پیش‌بینی کند که ادامه جمله به این صورت خواهد بود: "ابرها در حال حرکت هستند."

یادگیری عمیق Deep Learning

یادگیری عمیق یک زیرشاخه از یادگیری ماشین است که به استفاده از شبکه‌های عصبی پیچیده برای یادگیری الگوهای پیچیده از داده‌ها اشاره دارد.

مثال: در مدل‌های زبانی بزرگ مانند GPT-3، یادگیری عمیق به این معناست که شبکه عصبی مدل از لایه‌های مختلف برای پردازش و درک داده‌های زبانی استفاده می‌کند.

بازیابی اطلاعات Information Retrieval

بازیابی اطلاعات به فرآیند جستجو و استخراج اطلاعات از پایگاه‌های داده یا مجموعه‌های متنی اشاره دارد.

مثال: وقتی که از GPT-3 یا هر مدل زبانی مشابه برای جستجو و پاسخ به سوالات استفاده می‌کنید، مدل به‌طور داخلی از تکنیک‌های بازیابی اطلاعات برای یافتن پاسخ صحیح به سؤال شما استفاده می‌کند.

GPU و پردازش موازی

GPU واحد پردازش گرافیکی دستگاهی است که برای انجام محاسبات موازی و سنگین طراحی شده است.

برخلاف CPU که برای انجام وظایف متوالی طراحی شده، GPU می‌تواند هزاران محاسبه را به صورت همزمان انجام دهد.

مثال: فرض کنید شما می‌خواهید یک مدل یادگیری ماشین را آموزش دهید که نیاز به پردازش تعداد زیادی داده دارد.

اگر از CPU استفاده کنید، زمان زیادی خواهد برد، اما با استفاده از GPU، این پردازش‌ها به‌طور همزمان انجام می‌شوند و زمان آموزش مدل به شدت کاهش می‌یابد.

راه‌اندازی سرد Cold Start

Cold start زمانی است که یک تابع یا سرویس سرورلس برای اولین بار یا بعد از مدتی بیکار بودن اجرا می‌شود.

در این حالت، منابع (مثل حافظه و پردازشگر) باید دوباره بارگذاری شوند که این کار می‌تواند زمان‌بر باشد.

مثال: فرض کنید شما یک وب‌سایت دارید که از AWS Lambda برای اجرای توابع استفاده می‌کند.

اگر یک مدت طولانی از این توابع استفاده نکرده باشید و کاربری از آن‌ها درخواست کند، AWS باید تابع را دوباره راه‌اندازی کند که این باعث ایجاد تأخیر در پاسخ‌دهی می‌شود.

حافظه چندسطحی Multi-tier Storage

در حافظه چندسطحی از چند نوع ذخیره‌سازی با سرعت‌های مختلف استفاده می‌شود.

داده‌ها ابتدا در ذخیره‌سازی سریع‌تر مثل حافظه DRAM قرار می‌گیرند و اگر نیاز به فضای بیشتری باشد، به ذخیره‌سازی کندتر مثل SSD یا هارد دیسک منتقل می‌شوند.

مثال: فرض کنید شما در حال بارگذاری یک مدل یادگیری ماشین بزرگ هستید.

داده‌های مدل ابتدا در حافظه سریع DRAM قرار می‌گیرند. اگر فضای کافی نداشته باشید، داده‌ها به SSD انتقال می‌یابند تا حافظه سریع‌تر برای پردازش بیشتر آزاد شود.

مهاجرت زنده Live Migration

مهاجرت زنده فرآیندی است که در آن داده‌ها یا وظایف از یک سرور به سرور دیگر منتقل می‌شود، بدون اینکه پردازش‌ها قطع شوند.

مثال: فرض کنید شما یک مدل یادگیری ماشین در یک سرور در حال اجرا دارید و نیاز به جابجایی به سرور دیگری دارید.

در فرآیند مهاجرت زنده، مدل بدون توقف اجرا، از سرور قدیمی به سرور جدید منتقل می‌شود.

زمان‌بندی Scheduling

زمان‌بندی فرآیندی است که در آن منابع محاسباتی به وظایف مختلف تخصیص داده می‌شود.

این کار به بهینه‌ترین شکل انجام می‌شود تا کمترین تأخیر را در پردازش‌ها داشته باشیم.

مثال: فرض کنید چندین کار مختلف در یک سرور نیاز به پردازش دارند. سیستم زمان‌بندی تصمیم می‌گیرد که هر کار چه زمانی باید اجرا شود و منابع سرور چگونه بین کارها تقسیم شوند تا تأخیر در پردازش به حداقل برسد.

FaaS (Function as a Service)

FaaS یک مدل سرویس‌دهی در رایانش ابری است که به شما اجازه می‌دهد تنها برای اجرا شدن توابع هزینه پرداخت کنید و نیازی به مدیریت سرور ندارید.

مثال: فرض کنید شما می‌خواهید یک تابع ساده بنویسید که هر بار کاربری به سایت شما وارد می‌شود، یک خوشامدگویی ارسال کند.

شما از AWS Lambda یا Google Cloud Functions برای نوشتن این تابع استفاده می‌کنید و تنها زمانی که این تابع اجرا می‌شود، هزینه پرداخت می‌کنید.

مقیاس‌بندی خودکار Auto Scaling

مقیاس‌بندی خودکار Auto Scaling به فرآیندی اطلاق می‌شود که به سیستم اجازه می‌دهد به‌طور خودکار تعداد منابع محاسباتی (مثل سرورها یا پردازنده‌ها) را بر اساس نیازهای لحظه‌ای خود افزایش یا کاهش دهد.

در سیستم‌های رایانش ابری و سرورلس، وقتی تقاضای پردازش افزایش می‌یابد (مثلاً تعداد کاربران یا درخواست‌ها بیشتر می‌شود)، سیستم به‌طور خودکار منابع بیشتری را اضافه می‌کند تا از ایجاد تأخیر و افت کیفیت خدمات جلوگیری کند.

برعکس، وقتی تقاضا کاهش می‌یابد، سیستم منابع غیرضروری را خاموش کرده یا کاهش می‌دهد تا از مصرف اضافی منابع و هزینه‌های بالا جلوگیری شود.

مقیاس‌بندی خودکار Auto Scaling

مقیاس‌بندی خودکار می‌تواند به دو صورت عمل کند:

۱- مقیاس‌بندی عمودی Vertical Scaling : در این حالت، منابع موجود مانند پردازنده یا حافظه را افزایش می‌دهیم، به‌طوری که سرور قوی‌تر می‌شود.

اما این روش محدودیت‌هایی دارد و در بسیاری از مواقع به اندازه مقیاس‌بندی افقی مؤثر نیست.

مقیاس‌بندی خودکار Auto Scaling

مقیاس‌بندی خودکار می‌تواند به دو صورت عمل کند:

۲- مقیاس‌بندی افقی Horizontal Scaling : در این حالت، تعداد سرورها یا منابع پردازشی به‌طور افقی افزایش می‌یابد. این به این معناست که به جای اینکه سرور موجود را قوی‌تر کنیم، تعداد سرورها را افزایش می‌دهیم. این روش به‌ویژه در محیط‌های ابری یا سرورلس مانند AWS Lambda، Google Cloud Functions و سایر سرویس‌های مشابه استفاده می‌شود.

مقیاس‌بندی خودکار Auto Scaling

در سیستم‌های مدل‌های زبانی بزرگ LLM مانند GPT-3 یا BERT، تقاضا ممکن است به‌طور ناگهانی و در مقیاس بزرگ افزایش یابد.

این به این معناست که باید توان پردازشی سیستم به سرعت افزایش یابد تا بتواند درخواست‌ها را با کیفیت بالا و بدون تأخیر پردازش کند.

مقیاس‌بندی خودکار به سیستم‌ها این امکان را می‌دهد که بدون نیاز به مداخله انسانی و به‌طور پویا، منابع خود را مدیریت کنند.

مقیاس‌بندی خودکار Auto Scaling

در Amazon Web Services (AWS)، از سرویس‌هایی مانند Auto Scaling Groups برای مقیاس‌بندی منابع استفاده می‌شود.

به این معنا که:

اگر تعداد درخواست‌ها از یک حد معین بیشتر شود (مثلاً تعداد کاربرانی که از یک مدل زبانی بزرگ استفاده می‌کنند افزایش یابد)، Auto Scaling به‌طور خودکار سرورهای جدیدی اضافه می‌کند تا بار پردازش به‌طور مؤثر تقسیم شود.

وقتی تعداد درخواست‌ها کاهش می‌یابد، سرورهای اضافی خاموش می‌شوند یا منابع کاهش می‌یابند تا هزینه‌های اضافی به حداقل برسد.

تفاوت‌های اصلی بین Auto Scaling و Scheduling :

مقیاس‌بندی خودکار به این معناست که سیستم به‌طور خودکار و بدون دخالت انسان تعداد منابع مورد نیاز خود را بر اساس تقاضای واقعی تنظیم می‌کند.

زمان‌بندی Scheduling به معنای تخصیص منابع در زمان‌های خاص و پیش‌بینی شده است.

این یعنی شما به سیستم می‌گویید که در زمان‌های خاصی منابع را افزایش یا کاهش دهد.

تفاوت‌های اصلی بین Auto Scaling و Scheduling :

ویژگی	Auto Scaling	Scheduling
پاسخ به بار	به‌طور خودکار و در لحظه پاسخ می‌دهد	منابع در زمان‌های خاص از پیش تعیین‌شده تنظیم می‌شوند
نیاز به دخالت انسان	نیازی به دخالت انسان ندارد	نیاز به برنامه‌ریزی قبلی دارد
زمان‌بندی	به‌صورت داینامیک و در زمان واقعی انجام می‌شود	منابع بر اساس زمان‌بندی مشخص تنظیم می‌شوند
مناسب برای	تغییرات ناگهانی در بار و تقاضا	بار ثابت و قابل پیش‌بینی (مثلاً ساعات پیک ترافیک)

در بسیاری از سیستم‌ها، ترکیبی از این دو روش استفاده می‌شود تا هم از تغییرات ناگهانی بار و هم از زمان‌های پیک ترافیک به خوبی پشتیبانی شود.

بررسی مقاله اول

Enabling Efficient Serverless Inference Serving for LLMs (Large Language Models) in the Cloud

امکان‌پذیر ساختن ارائه استنتاج به‌طور کارآمد برای مدل‌های زبانی بزرگ در محیط ابری

مقاله ۱

این مقاله به چالش‌های مقیاس‌بندی و حل مشکل تأخیر شروع سرد **Cold Start** در سیستم‌های سرورلس برای پردازش مدل‌های زبانی بزرگ **LLMs** پرداخته است.

خلاصه اهداف اصلی:

کاهش تأخیر شروع سرد Cold Start Latency

مقیاس‌بندی خودکار منابع پردازشی

معرفی راهکار ServerlessLLM برای مدل‌های زبانی بزرگ

بهینه‌سازی تخصیص منابع و مدیریت حافظه

ارزیابی عملکرد و بهبود کارایی سیستم

مفاهیم:

در این مقاله، علاوه بر موضوعات مقیاس‌بندی خودکار Auto Scaling و سرورلس، چندین مفهوم دیگر به تفصیل بررسی شده است.

این مفاهیم به طور خاص به **چالش‌ها و راهکارهای کاهش تأخیر شروع سرد Cold Start Latency در مدل‌های زبانی بزرگ LLM در محیط‌های سرورلس** مربوط می‌شود.

مفهوم LLM Checkpoints

به نسخه‌های ذخیره‌شده‌ای از مدل‌های زبانی بزرگ اطلاق می‌شود که در طول فرآیند آموزش یا استنتاج مدل‌های پیچیده مانند GPT-3، BERT و سایر مدل‌های مشابه، ذخیره می‌شوند.

چک‌پوینت‌ها در واقع شامل پارامترهای مدل، وزن‌ها، و سایر اطلاعات ضروری هستند که وضعیت مدل را در یک لحظه خاص ذخیره می‌کنند.

این اطلاعات به مدل این امکان را می‌دهند که در مراحل بعدی آموزش یا پردازش، از جایی که در چک‌پوینت متوقف شده، ادامه دهد.

مفهوم LLM Checkpoints

آموزش مدل: در طول آموزش، چک‌پوینت‌ها به‌طور مداوم ذخیره می‌شوند تا در صورت توقف، آموزش از همان نقطه ادامه یابد.

استنتاج مدل: در پردازش داده‌ها، چک‌پوینت‌ها اجازه می‌دهند که مدل به‌طور دقیق و سریع پردازش انجام دهد.

چالش‌ها: حجم زیاد چک‌پوینت‌ها در مدل‌های بزرگ می‌تواند باعث تأخیر بارگذاری و نیاز به مدیریت منابع شود.

تکنیک بارگزاری چند لایه LLM Checkpoints

در این مقاله، سیستم بارگذاری چک‌پوینت چندلایه که توسط ServerlessLLM معرفی شده، یک روش نوآورانه برای کاهش تأخیر شروع سرد و بهبود زمان بارگذاری مدل‌های زبانی بزرگ است.

این سیستم با استفاده از یک ساختار ذخیره‌سازی چندلایه که شامل حافظه GPU، DRAMو SSD است، زمان بارگذاری مدل‌ها را تسریع می‌کند.

عملکرد بارگزاری چند لایه LLM Checkpoints

۱- فرمت بهینه‌شده چک‌پوینت‌ها:

مدل‌ها به‌طور بخش‌بندی‌شده ذخیره می‌شوند و داده‌ها به صورت دسته‌ای و ترتیبی خوانده می‌شوند.

این روش از عملیات I/O زیاد جلوگیری کرده و سرعت انتقال داده‌ها را افزایش می‌دهد.

۲- ساختار ذخیره‌سازی چندلایه:

GPU حافظه برای پردازش نهایی مدل‌ها استفاده می‌شود.

DRAM به‌عنوان کش میانه برای دسترسی سریع به GPU و SSD به‌عنوان ذخیره‌سازی بلندمدت برای چک‌پوینت‌ها به‌کار می‌رود.

عملکرد بارگزاری چند لایه LLM Checkpoints

۳- بارگذاری موازی و تقسیم داده‌ها:

داده‌ها به‌طور موازی در سطوح مختلف ذخیره‌سازی بارگذاری می‌شوند و از چندنخ‌گذاری برای بیشینه‌سازی پهنای باند استفاده می‌شود.

۴- آدرس‌دهی مؤثر به تنسورها:

آدرس حافظه هر **تنسور** از قبل محاسبه می‌شود که این کار عملیات کپی حافظه غیرضروری را حذف کرده و سرعت بارگذاری را افزایش می‌دهد.

مفهوم تنسور

تنسور Tensor یک ماتریس چند بعدی است که در علم داده‌ها و یادگیری ماشین برای نمایش و پردازش داده‌ها استفاده می‌شود.

تنسور می‌تواند یک عدد (که به آن اسکالر گفته می‌شود)،

یک بردار (آرایه یک‌بعدی)،

یا یک ماتریس (آرایه دوبعدی) باشد،

اما در یادگیری عمیق، تنسورها معمولاً به شکل ماتریس‌های چند بعدی در نظر گرفته می‌شوند که برای نمایش داده‌های پیچیده و مدل‌ها به کار می‌روند.

چالش‌های شروع سرد Cold Start Latency

مشکل شروع سرد زمانی رخ می‌دهد که سیستم‌های سرورلس پس از مدت‌ها بیکار بودن، شروع به بارگذاری مدل‌های بزرگ مانند LLM می‌کنند که این فرآیند زمان‌بر است.

این تأخیر به دلیل نیاز به بارگذاری مدل‌های عظیم و فعال‌سازی منابع پردازشی مانند GPU به وجود می‌آید.

چالش‌های شروع سرد Cold Start Latency

در سیستم‌های سرورلس، زمانی که سرویس‌ها یا توابع به مدت طولانی استفاده نمی‌شوند، منابع مورد نیاز آن‌ها (مثل سرورها یا ماشین‌های مجازی) به طور موقت خاموش می‌شوند یا به حالت بیکار می‌روند.

این موضوع به دلایل زیر اتفاق می‌افتد:

۱. صرفه‌جویی در هزینه‌ها

در سرویس‌های سرورلس، شما فقط برای استفاده از منابع محاسباتی هزینه می‌پردازید. وقتی هیچ درخواستی برای پردازش وجود نداشته باشد، سرورهای مربوط به آن سرویس به طور خودکار خاموش می‌شوند تا از هزینه‌های اضافی جلوگیری شود.

این کار باعث می‌شود که فقط زمانی که واقعاً به منابع نیاز دارید، هزینه‌ها اعمال شود.

چالش‌های شروع سرد Cold Start Latency

۲. مدیریت منابع بهینه

در محیط‌های سرورلس، مدیریت منابع به‌طور خودکار انجام می‌شود. سرویس‌دهندگان ابری مانند AWS Lambda، Google Cloud Functions یا Azure Functions از **مقیاس‌بندی خودکار** استفاده می‌کنند. یعنی زمانی که تقاضا کم می‌شود، منابع اضافی به حالت بیکار در می‌آیند تا منابع بهینه‌تر استفاده شوند.

۳. افزایش انعطاف‌پذیری و مقیاس‌پذیری

در این مدل‌ها، سیستم به‌طور خودکار بر اساس نیازهای واقعی بار (مانند تعداد درخواست‌ها یا حجم پردازش) مقیاس می‌شود. به همین دلیل، اگر هیچ درخواستی برای پردازش وجود نداشته باشد، منابع به حالت بیکار می‌روند.

چالش‌های شروع سرد Cold Start Latency

۴. پاسخ به درخواست‌های لحظه‌ای

وقتی که تقاضا یا درخواست جدیدی برای پردازش ایجاد شود (مثل وقتی که کاربر جدیدی بخواهد از سیستم استفاده کند)، سیستم به‌طور خودکار منابع مورد نیاز را راه‌اندازی کرده و پردازش آغاز می‌شود.

این ویژگی یکی از مزایای سیستم‌های سرورلس است که موجب صرفه‌جویی در منابع می‌شود.

در سیستم‌های سرورلس، به دلیل اینکه منابع بر اساس نیاز تخصیص داده می‌شوند، زمانی که سرویس‌ها یا توابع بیکار باشند (یعنی درخواست جدیدی نباشد)، منابع به‌طور خودکار خاموش می‌شوند تا از مصرف بی‌دلیل منابع و هزینه‌ها جلوگیری شود. به همین دلیل، پس از مدتی بیکار بودن، زمانی که سیستم دوباره نیاز به پردازش داشته باشد، باید مدل‌ها و منابع دوباره بارگذاری شوند که این امر می‌تواند باعث ایجاد تأخیر به نام شروع سرد Cold Start شود.

چالش‌های شروع سرد Cold Start Latency

تاثیرات منفی:

۱- **تأخیر پاسخ‌دهی**: برای کاربردهایی که نیاز به پاسخ‌دهی سریع دارند، مانند چت‌بات‌ها، ترجمه متن یا دستیارهای صوتی، تأخیر ناشی از شروع سرد می‌تواند تجربه کاربری را تحت تأثیر قرار دهد.

۲- **هزینه‌ها**: با توجه به اینکه مقیاس‌بندی در سرورلس به‌طور پویا انجام می‌شود، زمان طولانی بارگذاری مدل‌ها می‌تواند موجب مصرف منابع زیاد در زمان‌های غیرضروری شود.

مقیاس‌بندی خودکار برای مدل‌های زبانی بزرگ:

مقیاس‌بندی خودکار یکی از اهداف دیگر این مقاله است.

مقاله بر چگونگی بهینه‌سازی تخصیص منابع و استفاده بهینه از سرورهای GPU، حافظه‌ها و دیگر منابع پردازشی برای مدل‌های زبانی بزرگ تأکید دارد.

مقیاس‌بندی خودکار به معنای افزایش یا کاهش منابع پردازشی به‌طور خودکار بر اساس بار درخواست‌ها است.

هدف، کاهش هزینه‌ها و بهبود کارایی سیستم در پاسخ‌دهی به درخواست‌ها است.

مقیاس‌بندی خودکار برای مدل‌های زبانی بزرگ:

در مقدمه مقاله، به طور خلاصه اشاره شده که سیستم‌های سرورلس به طور خودکار منابع را تخصیص می‌دهند.

این سیستم‌ها به‌ویژه در مدل‌های زبانی بزرگ با چالش‌هایی مانند تأخیر شروع سرد مواجه هستند.

مشکلات مقیاس‌بندی به دلیل نیاز به پردازش سنگین و حجم زیاد داده‌ها ایجاد می‌شود.

این مشکلات به‌ویژه در مدل‌های زبانی بزرگ LLMs که نیاز به محاسبات پیچیده و داده‌های حجیم دارند، بیشتر خود را نشان می‌دهند.

ضرورت مقیاس‌بندی خودکار برای LLM

۱- نیاز به پردازش سنگین:

مشکل: مدل‌های زبانی بزرگ برای پردازش به منابع پردازشی زیادی نیاز دارند.

پردازش این مدل‌ها نیازمند GPU های قدرتمند، پردازنده‌های چند هسته‌ای و حافظه‌های زیاد است.

اهمیت: این پردازش‌های سنگین و پیچیده می‌توانند به مشکلات مقیاس‌بندی منجر شوند و زمان پردازش را افزایش دهند.

ضرورت مقیاس‌بندی خودکار برای LLM

۲- حجم زیاد داده‌ها:

مشکل: مدل‌های زبانی بزرگ به داده‌های بسیار زیادی برای آموزش نیاز دارند.

این داده‌ها ممکن است از منابع مختلفی مانند متن‌های کتابخانه‌ای، مقالات و داده‌های وب جمع‌آوری شوند.

اهمیت: حجم زیاد داده‌ها می‌تواند به مشکلات ذخیره‌سازی و پردازش منجر شود، که این مشکلات در مقیاس‌های بزرگتر، پیچیدگی بیشتری پیدا می‌کنند.

ضرورت مقیاس‌بندی خودکار برای LLM

۳- مدیریت منابع در مقیاس‌های بزرگ:

مشکل: سیستم‌های سرورلس باید بتوانند منابع را به‌طور مؤثر و دینامیک مدیریت کنند.

با افزایش تقاضا یا تغییرات در بار پردازشی، نیاز به تخصیص منابع مانند پردازنده‌ها و حافظه‌ها به‌طور خودکار وجود دارد.

اهمیت: بدون تخصیص درست و خودکار منابع، سیستم‌ها ممکن است با تأخیر یا از دست دادن عملکرد مواجه شوند.

ضرورت مقیاس‌بندی خودکار برای LLM

۴- مقیاس‌پذیری سیستم‌ها:

مشکل: مقیاس‌پذیری سیستم‌ها در محیط‌های سرورلس به‌ویژه زمانی که بار پردازشی به طور ناگهانی افزایش می‌یابد، به چالش تبدیل می‌شود.

اهمیت: این مشکل باعث می‌شود که نیاز به مقیاس‌بندی خودکار منابع پردازشی بیشتر احساس شود.

استراتژی‌های موجود برای مقابله با شروع سرد و تخصیص خودکار منابع

۱- حافظه موقت Warm Containers :

یکی از روش‌های مورد استفاده برای کاهش تأخیر، استفاده از حافظه‌های موقت است.

در این روش، مدل‌های زبانی بزرگ در حافظه‌های کش ذخیره می‌شوند تا در صورت نیاز، سریع‌تر بارگذاری شوند.

این کار باعث می‌شود که مدل‌ها برای درخواست‌های بعدی آماده‌تر باشند.

۱- حافظه موقت Warm Containers :

Warm Containers به سرورهای یا ماشین‌های مجازی اطلاق می‌شود که از قبل آماده و در حالت فعال هستند و به محض دریافت درخواست، قادر به پردازش آن بدون تأخیر زیاد خواهند بود.

به عبارت دیگر، این کانتینرها به‌طور پیش‌فرض در حالت آماده به کار warm نگه‌داشته می‌شوند تا به محض دریافت درخواست، بارگذاری جدیدی لازم نباشد.

۱- حافظه موقت Warm Containers :

نحوه عملکرد warm containers :

وقتی یک سرویس یا مدل در یک محیط سرورلس مانند AWS Lambda بارگذاری می‌شود، سرور یا کانتینر برای پردازش‌های آینده در حالت آماده به کار باقی می‌ماند.

این به معنای این است که به‌جای بارگذاری مجدد کامل مدل‌ها، تنها نیاز است که از حافظه کش شده یا سرور فعال برای پردازش استفاده شود.

۲- کشینگ مدل Model Caching

کشینگ مدل به فرآیند ذخیره‌سازی داده‌های مدل‌های زبانی بزرگ در حافظه‌های سریع مانند DRAM یا SSD گفته می‌شود، به‌طوری که مدل‌های زبانی بزرگ یا چک‌پوینت‌ها پس از بارگذاری اولیه به سرعت در دسترس قرار گیرند.

هدف اصلی کشینگ مدل این است که زمان بارگذاری مجدد مدل‌ها را به حداقل برساند و در نتیجه تأخیر ناشی از شروع سرد cold start کاهش یابد.

۲- کشینگ مدل Model Caching

نحوه عملکرد کشینگ مدل: پس از بارگذاری اولیه مدل، نسخه‌ای از آن در حافظه ذخیره می‌شود (مثلاً در حافظه DRAMیا حافظه کش سریع‌تر).

این مدل به‌طور مستقیم از حافظه کش خوانده می‌شود و نیازی به بارگذاری مجدد ندارد، بنابراین دسترسی به آن سریع‌تر می‌شود.

مزایا: کاهش زمان تأخیر و بهبود کارایی در پاسخ‌دهی سریع به درخواست‌ها.

مقاله ۱

ارتباط بین کشینگ مدل و Warm Containers :

در واقع، کشینگ مدل و warm containersاز منظر هدف مشابه هستند: هر دو به‌دنبال کاهش زمان تأخیر شروع سرد هستند.

کشینگ مدل داده‌های مدل را در حافظه ذخیره می‌کند تا از بارگذاری مجدد جلوگیری شود، در حالی که warm containers سرور یا محیط اجرایی را آماده نگه می‌دارد تا به محض دریافت درخواست، پردازش شروع شود بدون اینکه به زمان بارگذاری نیاز باشد.

کشینگ مدل تمرکز بیشتری روی ذخیره‌سازی داده‌ها و مدل‌ها دارد، در حالی که warm containers بیشتر به آماده‌سازی و فعال نگه داشتن محیط‌های اجرایی اشاره دارد.

۳- پیش‌بارگذاری مدل‌ها Model Preloading

پیش‌بارگذاری مدل‌ها به معنای بارگذاری مدل‌های زبانی بزرگ LLM یا داده‌ها قبل از شروع پردازش درخواست‌ها است.

این فرآیند در واقع یک نوع آماده‌سازی پیش‌رو برای استفاده از مدل‌هاست که قبل از دریافت درخواست‌های واقعی انجام می‌شود.

با این روش، مدل از ابتدا بارگذاری می‌شود و به‌طور آماده برای پردازش درخواست‌های بعدی قرار می‌گیرد.

نحوه عملکرد پیش‌بارگذاری: در پیش‌بارگذاری، **مدل‌ها** به طور پیش‌فرض و پیش از استفاده در حافظه یا سرورهای فعال قرار می‌گیرند.

این کار معمولاً قبل از شروع به کار یک سیستم یا برنامه انجام می‌شود.

تفاوت‌های اصلی بین پیش‌بارگذاری و کشینگ مدل:

ویژگی	پیش‌بارگذاری مدل (Model Preloading)	کشینگ مدل (Model Caching)
هدف	بارگذاری مدل‌ها یا داده‌ها پیش از درخواست‌ها	ذخیره‌سازی نتایج یا مدل‌ها در حافظه برای استفاده مجدد
زمان انجام	قبل از شروع استفاده از مدل‌ها (ابتدای اجرای سیستم)	پس از پردازش اولین درخواست یا پردازش‌های مشابه
تأثیر	کاهش تأخیر بارگذاری اولیه مدل	کاهش تأخیر پردازش مجدد و استفاده از نتایج ذخیره‌شده
محل ذخیره‌سازی	حافظه، سرور یا محیط اجرایی برای آماده‌سازی	حافظه کش سریع‌تر (DRAM، SSD)

۴- مهاجرت زنده Live Migration:

استفاده از مهاجرت زنده برای انتقال مدل‌ها از سرورهای دیگر به سرورهای فعال بدون اینکه پردازش متوقف شود، یکی دیگر از روش‌های کاهش تأخیر شروع سرد است.

با این روش، مدل‌ها می‌توانند بدون وقفه به سرور جدید منتقل شوند.

به‌جای اینکه مدل دوباره بارگذاری شود و منتظر بماند، مهاجرت زنده باعث می‌شود که مدل‌ها به سرعت و بدون تأخیر به سرور جدید منتقل شوند.

۴- مهاجرت زنده Live Migration:

مثال:

تصور کنید شما یک سیستم سرورلس دارید که از مدل‌های زبانی بزرگی مانند GPT-3 برای پردازش درخواست‌ها استفاده می‌کند.

این سیستم به‌طور خودکار منابع محاسباتی را اضافه یا کاهش می‌دهد بر اساس میزان تقاضا.

حالا بیایید سناریوهایی را بررسی کنیم:

۴- مهاجرت زنده Live Migration:

سناریو ۱: درخواست‌های کم: در ابتدا، سیستم شما تعداد کمی از کاربران دارد و پردازش‌ها به‌آرامی انجام می‌شود.

در این زمان، منابع پردازشی مانند GPU و RAM به اندازه کافی هستند و سیستم به راحتی کار می‌کند.

۴- مهاجرت زنده Live Migration:

سناریو ۲: افزایش تقاضا: حالا تعداد کاربران به‌طور ناگهانی افزایش می‌یابد.

سیستم نیاز به سرورهای جدید برای پردازش درخواست‌های جدید دارد، اما چون مدل‌های زبانی بزرگ نیاز به منابع زیادی دارند، بارگذاری مدل‌ها از ابتدا (با استفاده از منابع جدید) زمان زیادی می‌برد که این باعث تأخیر در پردازش می‌شود.

حالا چطور مهاجرت زنده به کمک می‌آید؟

مهاجرت زنده به این معناست که مدل‌ها به‌طور زنده و بدون وقفه از سرورهای قبلی به سرورهای جدید منتقل می‌شوند.

یعنی بدون اینکه پردازش متوقف شود یا تأخیری در پردازش پیش بیاید، مدل از سرور قبلی به سرور جدید منتقل می‌شود.

۴- مهاجرت زنده Live Migration:

نحوه عملکرد مهاجرت زنده:

انتقال بدون وقفه: فرض کنید که سیستم شما یک سرور قدیمی دارد که مدل GPT-3 در آن بارگذاری شده است و پردازش درخواست‌ها در حال انجام است.

حالا سیستم نیاز دارد که سرور جدیدی را برای پردازش بار اضافی به‌طور خودکار اضافه کند.

با استفاده از مهاجرت زنده، مدل GPT-3 بدون اینکه پردازش‌ها متوقف شود، به سرور جدید منتقل می‌شود.

مدل در سرور جدید راه‌اندازی می‌شود و از همان جایی که سرور قبلی متوقف شده بود، ادامه می‌دهد. این فرآیند در مدت‌زمان بسیار کوتاهی و بدون وقفه صورت می‌گیرد.

مقیاس‌بندی: وقتی که سیستم نیاز به افزایش منابع پیدا می‌کند، مهاجرت زنده به‌طور خودکار منابع جدید را تخصیص می‌دهد و بدون وقفه بارگذاری مدل را در سرورهای جدید انجام می‌دهد.

۵- الگوریتم‌های تخصیص منابع هوشمند

Smart Resource Allocation Algorithms

الگوریتم‌های تخصیص منابع هوشمند به سیستم‌ها این امکان را می‌دهند که به‌طور خودکار و بهینه منابع محاسباتی را تخصیص دهند.

این الگوریتم‌ها می‌توانند به تغییرات ناگهانی در بار پاسخ دهند، منابع را به‌طور پویا مقیاس‌بندی کنند و از پیش‌بینی بارهای آینده استفاده کنند تا سیستم در تمام شرایط بهینه عمل کند.

این ویژگی‌ها در سیستم‌های مدل‌های زبانی بزرگ و سیستم‌های سرورلس برای کاهش تأخیر و بهینه‌سازی مصرف منابع بسیار حیاتی هستند.

۵- الگوریتم‌های تخصیص منابع هوشمند

مثال‌هایی از استفاده الگوریتم‌های تخصیص منابع هوشمند:

AWS Auto Scaling :

در AWS ,سرویس Auto Scaling منابع را به‌طور خودکار افزایش یا کاهش می‌دهد.

به‌عنوان مثال، اگر ترافیک سایت یا تعداد درخواست‌های API افزایش یابد، Auto Scaling به‌طور خودکار سرورهای بیشتری اضافه می‌کند تا از تأخیر جلوگیری کند.

۵- الگوریتم‌های تخصیص منابع هوشمند

مثال‌هایی از استفاده الگوریتم‌های تخصیص منابع هوشمند:

Google Cloud Functions :

در Google Cloud Functions، منابع به‌طور خودکار مقیاس‌بندی می‌شوند. اگر تعداد درخواست‌های کاربران افزایش یابد، سیستم به‌طور خودکار تعداد بیشتری از توابع را اجرا می‌کند تا از تأخیر جلوگیری شود.

Azure Functions :

در Microsoft Azure, Azure Functions از الگوریتم‌های مقیاس‌بندی خودکار برای تخصیص منابع استفاده می‌کند.

در این حالت، تعداد توابع به‌طور پویا بسته به حجم درخواست‌ها افزایش می‌یابد.

۶- زمان‌بندی بهینه منابع Startup-Time-Optimized Scheduling

زمان‌بندی بهینه منابع یکی دیگر از راهکارها است که به کاهش تأخیرهای شروع سرد کمک می‌کند.

این الگوریتم‌ها به‌طور خاص زمان‌هایی که مدل‌های زبانی بزرگ نیاز به بارگذاری دارند را پیش‌بینی کرده و منابع را پیش از زمان تخصیص می‌دهند.

این روش به سیستم کمک می‌کند که منابع به‌طور هوشمندانه و پیشگیرانه تخصیص داده شوند و تأخیرهای ناشی از شروع سرد کاهش یابد.

مقاله ۱

الگوریتم پیشنهادی مقاله (ServerlessLLM)

در این مقاله، الگوریتم پیشنهادی برای کاهش تأخیر شروع سرد Cold Start Latency و مقیاس‌بندی خودکار منابع پردازشی در مدل‌های زبانی بزرگ LLMs به‌طور خاص معرفی شده است.

الگوریتم پیشنهادی این مقاله تحت عنوان **ServerlessLLM** معرفی می‌شود.

این الگوریتم به‌طور خاص برای مدل‌های زبانی بزرگ در سیستم‌های سرورلس طراحی شده تا مشکلات مقیاس‌بندی و تأخیرهای شروع سرد را حل کند.

شرح الگوریتم پیشنهادی: ServerlessLLM

ServerlessLLM یک راهکار مبتنی بر سیستم سرورلس است که به منظور بهبود کارایی و کاهش تأخیر در پردازش مدل‌های زبانی بزرگ طراحی شده است.

این الگوریتم از تکنیک‌های مختلفی برای بهینه‌سازی زمان بارگذاری مدل‌ها و تخصیص منابع استفاده می‌کند.

الگوریتم ServerlessLLM که در این مقاله پیشنهاد شده است، ترکیبی از مهاجرت زنده مدل‌ها، پیش‌بارگذاری مدل‌ها، کشینگ مدل‌ها و مقیاس‌بندی خودکار منابع است.

این الگوریتم به‌طور مؤثر چالش‌های مربوط به تأخیر شروع سرد و مقیاس‌بندی منابع پردازشی را حل کرده و عملکرد سیستم‌های پردازش مدل‌های زبانی بزرگ را بهینه می‌کند.

مقاله ۱

شرح الگوریتم پیشنهادی: ServerlessLLM

۱- کاهش تأخیر شروع سرد: با استفاده از **مهاجرت زنده** و **پیش‌بارگذاری مدل‌ها**، این الگوریتم زمان تأخیر ناشی از بارگذاری مجدد مدل‌ها را به حداقل می‌رساند.

۲- بهینه‌سازی تخصیص منابع: مقیاس‌بندی خودکار منابع به‌طور پویا انجام می‌شود و منابع به‌طور بهینه برای پردازش مدل‌های بزرگ تخصیص داده می‌شود.

۳- کاهش هزینه‌ها: با استفاده از **کشینگ مدل‌ها** و **مقیاس‌بندی خودکار**، از مصرف منابع غیرضروری جلوگیری می‌شود و هزینه‌ها کاهش می‌یابد.

۴- افزایش کارایی: تخصیص منابع به‌طور هوشمند و بهینه باعث بهبود سرعت پردازش و کاهش تأخیر می‌شود.

مقاله ۱

ترکیب تکنیک‌ها در ServerlessLLM :

۱- Model Preloading + Live Migration :

مهاجرت زنده مدل‌ها و پیش‌بارگذاری مدل‌ها در کنار هم باعث می‌شوند که مدل‌ها سریعاً از سرورهای قبلی به سرورهای جدید منتقل شوند و برای پردازش آماده شوند.

به‌طور مثال، وقتی تعداد درخواست‌ها به‌طور ناگهانی افزایش می‌یابد، سیستم به‌طور خودکار مدل‌ها را از سرورهای کم‌بار به سرورهای جدید منتقل می‌کند. این فرآیند بدون توقف پردازش انجام می‌شود و از تأخیرهای ناشی از شروع سرد جلوگیری می‌کند.

همزمان، پیش‌بارگذاری مدل‌ها در حافظه باعث می‌شود که زمانی که سرور جدید بارگذاری می‌شود، مدل‌ها بلافاصله در دسترس باشند و نیاز به بارگذاری مجدد مدل از ابتدا نباشد.

ترکیب تکنیک‌ها در ServerlessLLM :

۲- Model Caching + Auto-scaling :

کشینگ مدل‌ها و مقیاس‌بندی خودکار منابع به‌طور همزمان باعث می‌شوند که سیستم قادر به ذخیره‌سازی نتایج پردازش شده باشد و در صورت درخواست مشابه، از آن‌ها استفاده کند.

این کار باعث کاهش زمان پردازش می‌شود.

زمانی که بار پردازشی افزایش می‌یابد و منابع بیشتری نیاز است، سیستم به‌طور خودکار منابع اضافی مانند GPU یا حافظه را تخصیص می‌دهد تا پردازش‌های جدید و کش‌شده به سرعت انجام شوند.

این ترکیب از کشینگ و مقیاس‌بندی به سیستم کمک می‌کند که منابع بهینه‌تر استفاده شوند و زمان پردازش به حداقل برسد.

ترکیب تکنیک‌ها در ServerlessLLM :

۳- Optimized Scheduling + مقیاس‌بندی خودکار منابع:

زمان‌بندی بهینه برای تخصیص منابع به‌گونه‌ای انجام می‌شود که قبل از بروز نیاز به پردازش، سیستم بتواند پیش‌بینی کند که چه منابعی نیاز خواهد بود.

به این ترتیب، منابع از پیش تخصیص داده می‌شوند و سیستم به‌طور بهینه‌تر عمل می‌کند.

مقیاس‌بندی خودکار منابع پس از پیش‌بینی دقیق منابع و زمان‌بندی انجام می‌شود.

در اینجا، زمان‌بندی پیش‌گیرانه کمک می‌کند که منابع اضافی قبل از بالا رفتن تقاضا به‌طور مؤثر تخصیص یابند، و زمانی که بار پردازش افزایش می‌یابد، سیستم از مقیاس‌بندی خودکار استفاده می‌کند تا منابع بیشتری را تخصیص دهد.

ترکیب تکنیک‌ها در ServerlessLLM :

۴- Memory Optimization + پیش‌بارگذاری مدل‌ها:

مدیریت حافظه بهینه کمک می‌کند که مدل‌ها و داده‌ها به‌طور مؤثری در حافظه‌های سریع DRAM، SSD ذخیره شوند.

این امر باعث می‌شود که زمان بارگذاری مدل‌ها به حداقل برسد و سیستم بتواند داده‌ها را سریع‌تر پردازش کند.

این حافظه‌های سریع به‌طور ویژه برای مدل‌های زبانی بزرگ که نیاز به حجم زیادی از داده‌ها دارند، ضروری هستند.

به‌طور همزمان، پیش‌بارگذاری مدل‌ها در این حافظه‌ها باعث می‌شود که زمان پردازش مدل‌ها به طور چشمگیری کاهش یابد.

ترکیب تکنیک‌ها در ServerlessLLM :

۴- Memory Optimization + پیش‌بارگذاری مدل‌ها:

مدیریت حافظه بهینه کمک می‌کند که مدل‌ها و داده‌ها به‌طور مؤثری در حافظه‌های سریع DRAM، SSD ذخیره شوند.

این امر باعث می‌شود که زمان بارگذاری مدل‌ها به حداقل برسد و سیستم بتواند داده‌ها را سریع‌تر پردازش کند.

این حافظه‌های سریع به‌طور ویژه برای مدل‌های زبانی بزرگ که نیاز به حجم زیادی از داده‌ها دارند، ضروری هستند.

به‌طور همزمان، پیش‌بارگذاری مدل‌ها در این حافظه‌ها باعث می‌شود که زمان پردازش مدل‌ها به طور چشمگیری کاهش یابد.

نقاط ضعف مقاله :

۱- کاربرد محدود به مدل‌های خاص

الگوریتم ServerlessLLM و راهکارهای پیشنهادی بیشتر برای مدل‌های زبانی بزرگ طراحی شده‌اند و ممکن است برای مدل‌های کوچک‌تر یا وظایف دیگر کاربرد کمتری داشته باشند.

این یعنی که راهکارهایی که در مقاله مطرح شده‌اند بیشتر به مدل‌های پیچیده و بزرگ مانند GPT-3 یا BERT مرتبط هستند و ممکن است در مدل‌های کوچک‌تر با مشکلات مشابه مواجه نشوند یا به منابع کمتری نیاز داشته باشند.

نقاط ضعف مقاله :

۲- وابستگی به منابع ابری و هزینه‌ها

بسیاری از راهکارهای مقاله، از جمله مقیاس‌بندی خودکار و مهاجرت زنده مدل‌ها، به منابع ابری وابسته هستند.

این وابستگی به ابرهای بزرگ می‌تواند هزینه‌ها را افزایش دهد و مشکلاتی مانند محدودیت‌های دسترسی به داده‌ها یا هزینه‌های بالای پردازش را به همراه داشته باشد.

این مسأله به‌ویژه در سیستم‌های سرورلس که هزینه‌ها بر اساس مصرف منابع تعیین می‌شود، بیشتر به چشم می‌آید.

نقاط ضعف مقاله :

۳- تأخیرهای مربوط به مهاجرت زنده و کشینگ مدل‌ها

مهاجرت زنده مدل‌ها یک راهکار مفید است، اما انتقال مدل‌ها به سرورهای جدید یا نگهداری آن‌ها در کش همچنان ممکن است باعث تأخیر شود، به‌ویژه در زمان‌هایی که حجم داده‌ها یا درخواست‌ها به‌طور ناگهانی افزایش یابد.

همچنین ممکن است کشینگ مدل‌ها برای مدل‌های زبانی بزرگ نیاز به فضای ذخیره‌سازی بسیار زیاد داشته باشد که خود می‌تواند به مشکلات مقیاس‌پذیری و هزینه‌های اضافی منجر شود.

نقاط ضعف مقاله :

۴- پیش‌بینی و زمان‌بندی منابع به‌طور پیشرفته

یکی از راهکارهای مقاله پیش‌بینی تقاضا و زمان‌بندی منابع بهینه است.

این پیش‌بینی و زمان‌بندی ممکن است بر اساس الگوریتم‌های پیشرفته باشد که در برخی مواقع به دقت بالا و داده‌های پیشین نیاز دارند.

در صورتی که داده‌ها یا بار پردازشی به‌طور ناگهانی تغییر کنند، پیش‌بینی‌ها ممکن است غلط باشند و تخصیص منابع نتواند به‌طور مؤثر انجام شود.

پیشنهادات بهبود مقاله:

پشتیبانی از محیط‌های ترکیبی Hybrid یا Edge: گسترش الگوریتم‌ها برای استفاده از زیرساخت‌های محلی و ابری به‌طور هم‌زمان، که هزینه‌ها را کاهش دهد و انعطاف‌پذیری را افزایش دهد.

بهینه‌سازی مهاجرت زنده مدل‌ها: بهبود مهاجرت زنده با انتقال بخش‌های مورد نیاز مدل به سرورهای جدید به جای انتقال کامل مدل.

استفاده از مدل‌های فشرده‌شده: اعمال quantization یا pruning برای کاهش حجم مدل‌ها و زمان بارگذاری.

استفاده از حافظه‌های سریع‌تر: بهره‌گیری از حافظه‌های NVMe و حافظه‌های توزیع‌شده برای بارگذاری سریع‌تر مدل‌ها.

پشتیبانی از مدل‌های با منابع محدود: گسترش الگوریتم‌ها برای مدل‌های کوچکتر که بتوانند از مقیاس‌بندی خودکار بهره‌مند شوند.

با تشکر از همراهی شما

محمد سعید صفایی صادق