به نام خدا

درس یادگیری عمیق

تمرین سری چهارم

استاد درس : دکتر محمدرضا محمدی

دستیاران : رضا علیدوست ، علیرضا حقانی

و امیرحسین نمازی

دانشگاه علم و صنعت ایران، دانشکده مهندسی کامپیوتر

نیمسال دوم تحصیلی ۱۴۰۳ - ۱۴۰۴

مهلت تحویل : ۱۴۰۴/۰۲/۲۳

لطفا به نکات موجود در سند قوانین انجام و تحویل تمرین ها دقت فرمایید.

سوالات تئوری

۱.  بر اساس مقاله لطفا به سوالات زیر پاسخ دهید (۱۰ نمره):

(آ) چالش‌های اصلی در زمینه مدیریت حافظه که سیستم‌های خدمات‌دهی LLM موجود مواجه هستند، چیست؟

سیستم‌های خدمات‌دهی LLM موجود به دلیل نحوه مدیریت حافظه کش کلید-مقدار [1] با چالش‌های اساسی مواجه هستند. مشکل اصلی این است که این سیستم‌ها حافظه کش KV هر درخواست را در یک فضای حافظه پیوسته [2] ذخیره می‌کنند. این رویکرد به دلیل ویژگی‌های منحصربه‌فرد حافظه کش KV در LLMها (رشد پویا، طول عمر و اندازه نامشخص از قبل) منجر به ناکارآمدی‌های جدی می‌شود. چالش‌های اصلی عبارتند از:

. اتلاف حافظه به دلیل تکه‌تکه شدن [3]:

- تکه‌تکه شدن داخلی [4] : سیستم‌های موجود برای هر درخواست، یک قطعه حافظه پیوسته به اندازه حداکثر طول دنباله ممکن (مثلاً ۲۰۴۸ توکن) از پیش تخصیص

[1] KV Cache

[2] contiguous

[3] Fragmentation

[4] Internal Fragmentation

می‌دهند. از آنجایی که طول واقعی خروجی اغلب بسیار کوتاه‌تر از این مقدار است، بخش بزرگی از حافظه تخصیص‌داده‌شده هرگز استفاده نمی‌شود و به هدر می‌رود. (ارجاع: بخش ۱ و شکل ۳). مقاله در شکل ۲ نشان می‌دهد که در سیستم‌های موجود، تنها ۲۰ تا ۳۸ درصد از حافظه کش KV واقعاً برای ذخیره توکن‌ها استفاده می‌شود.

- تکه‌تکه شدن خارجی[۵]: زمانی که درخواست‌ها با طول‌های حداکثری متفاوت وارد سیستم می‌شوند، تخصیص‌دهنده حافظه (مانند buddy allocator) ممکن است نتواند فضاهای خالی بین بلاک‌های تخصیص‌داده‌شده را به طور مؤثر مدیریت کند و فضاهای خالی غیرقابل استفاده‌ای ایجاد می‌شود. (ارجاع: بخش ۱ و شکل ۳). تصور کنید حافظه GPU شما یک قفسه کتاب با طول مشخص است. هر درخواست[۶] یک کتاب با قطر متفاوت است. حالت سیستم‌های قدیمی (تکه‌تکه شدن خارجی): این سیستم‌ها

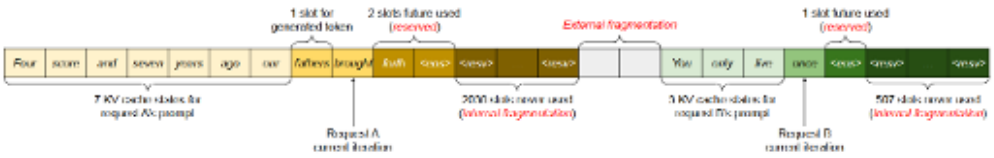

Figure 3. KV cache memory management in existing systems. Three types of memory wastes – reserved, internal fragmentation, and external fragmentation – exist that prevent other requests from fitting into the memory. The token in each memory slot represents its KV cache. Note the same tokens can have different KV cache when at different positions.

برای هر کتاب (درخواست) نیاز به یک فضای یکپارچه و پیوسته روی قفسه دارند. فرض کنید یک کتاب با قطر ۲۰ سانتی‌متر و یک کتاب با قطر ۳۰ سانتی‌متر را در قفسه قرار می‌دهید. حالا یک کتاب با قطر ۵۰ سانتی‌متر برداشته می‌شود. اکنون شما یک فضای خالی ۵۰ سانتی‌متری دارید. اگر درخواست بعدی یک کتاب با قطر ۶۰ سانتی‌متر باشد، با اینکه مجموع فضاهای خالی در کل قفسه شاید بیشتر از ۶۰ سانتی‌متر باشد، اما چون هیچ فضای خالی یکپارچه‌ای به طول ۶۰ سانتی‌متر وجود ندارد، این کتاب جدید در قفسه جا نمی‌شود. این فضاهای خالیِ کوچک و پراکنده که قابل استفاده برای درخواست‌های بزرگتر نیستند، تکه‌تکه شدن خارجی نام دارند. مقاله در شکل ۳ به خوبی این مفهوم را نشان می‌دهد که فضاهای خالی بین بلاک‌های حافظه‌ی رزرو شده برای درخواست A و B وجود دارد که قابل استفاده نیستند.

[5] External Fragmentation

[6] request

.□□ عدم امکان اشتراک‌گذاری حافظه[۷]: الگوریتم‌های رمزگشایی پیشرفته مانند نمونه‌برداری موازی[۸] یا جستجوی پرتوئی[۹] چندین دنباله خروجی برای یک ورودی واحد تولید می‌کنند. بخش‌هایی از این دنباله‌ها (مانند پرامپت اولیه) کاملاً یکسان هستند و حافظه کش KV آن‌ها می‌تواند به اشتراک گذاشته شود. اما چون سیستم‌های موجود برای هر دنباله یک بلاک حافظه پیوسته مجزا تخصیص می‌دهند، این اشتراک‌گذاری حافظه غیرممکن یا بسیار ناکارآمد است. (ارجاع: بخش ۱ و بخش ۳، پاراگراف Complex decoding algorithms).

.□□□ حافظه کش KV و محدودیت‌های برنامه‌ریزی[۱۰]:

- حافظه کش KV برای هر درخواست می‌تواند بسیار بزرگ باشد (مثلاً تا ۱.۶ گیگابایت برای یک درخواست در مدل OPT-13B). این امر تعداد درخواست‌هایی را که می‌توانند به صورت همزمان در یک دسته[۱۱] پردازش شوند، به شدت محدود می‌کند. (ارجاع: بخش ۳، پاراگراف Large KV cache).
- طول ورودی و خروجی درخواست‌ها از قبل مشخص نیست. این عدم قطعیت، برنامه‌ریزی[۱۲] و تخصیص حافظه را پیچیده می‌کند و باعث می‌شود سیستم‌ها رویکرد محافظه‌کارانه و ناکارآمد تخصیص حداکثری را در پیش بگیرند. (ارجاع: بخش ۳، پاراگراف Scheduling for unknown input/output lengths).

(ب) PagedAttention چگونه به این چالش‌ها پاسخ می‌دهد؟

PagedAttention یک الگوریتم توجه[۱۳] جدید است که با الهام از تکنیک‌های کلاسیک حافظه مجازی[۱۴] و صفحه‌بندی[۱۵] در سیستم‌عامل‌ها طراحی شده است تا مشکلات مدیریت حافظه در سیستم‌های موجود را حل کند. راهکار اصلی PagedAttention این است که به کلیدها و مقادیر (KV Cache) اجازه می‌دهد تا در فضاهای حافظه غیرپیوسته[۱۶] ذخیره شوند. این کار از طریق مکانیزم زیر انجام می‌شود:

---

[7] Inability to Share Memory
[8] Parallel Sampling
[9] Beam Search
[10] Large KV Cache & Scheduling Complexity
[11] batch
[12] scheduling
[13] Attention
[14] Virtual Memory
[15] Paging
[16] non-contiguous

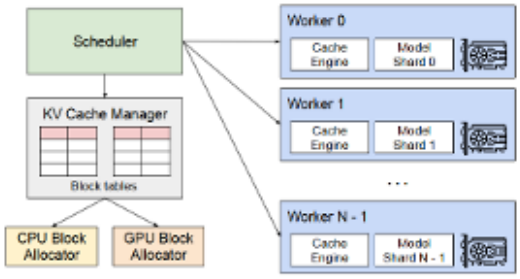

Figure 4. vLLM system overview.

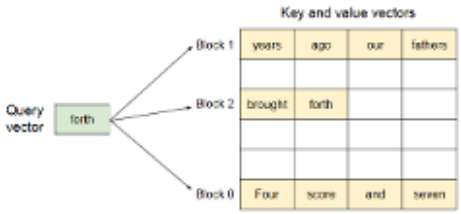

Figure 5. Illustration of the PagedAttention algorithm, where the attention key and values vectors are stored as non-contiguous blocks in the memory.

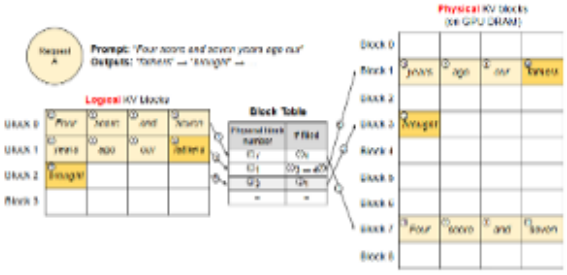

Figure 6. Block table translation in vLLM.

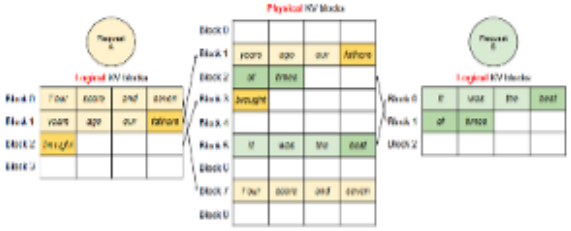

Figure 7. Storing the KV cache of two requests at the same time in vLLM.

.1 تقسیم حافظه کش به بلاک‌ها (KV Blocks): PagedAttention حافظه کش KV هر دنباله را به بلاک‌هایی با اندازه ثابت تقسیم می‌کند. هر بلاک، کلید و مقدار مربوط به تعداد ثابتی از توکن‌ها را در خود جای می‌دهد. این بلاک‌ها معادل "صفحات[17]" در حافظه مجازی هستند. (ارجاع: بخش ۴.۱).

.2 مدیریت غیرپیوسته حافظه: برخلاف سیستم‌های قبلی، این بلاک‌ها نیازی به قرارگیری پشت سر هم در حافظه فیزیکی GPU ندارند. vLLM با استفاده از جداول بلاک[18]، بلاک‌های منطقی (از دید هر دنباله) را به بلاک‌های فیزیکی (در حافظه GPU) نگاشت می‌کند. (ارجاع: بخش ۴.۲ و شکل ۵ و ۶).

PagedAttention با این رویکرد به چالش‌های ذکر شده در سوال (آ) به شکل زیر پاسخ می‌دهد:

- حل مشکل تکه‌تکه شدن:

.1 تکه‌تکه شدن داخلی: حافظه به صورت پویا و بلاک به بلاک تخصیص داده می‌شود. یعنی تنها زمانی یک بلاک جدید تخصیص می‌یابد که بلاک قبلی پر شده باشد. این کار اتلاف حافظه داخلی را به حداکثر یک بلاک برای هر دنباله محدود می‌کند که

[17] Pages
[18] Block Tables

بسیار ناچیز است. (ارجاع: بخش ۴.۳).

- تکه‌تکه شدن خارجی: از آنجایی که تمام بلاک‌ها اندازه یکسانی دارند، مشکل تکه‌تکه شدن خارجی به طور کامل از بین می‌رود. (ارجاع: بخش ۱).

- امکان‌پذیر کردن اشتراک‌گذاری حافظه: از آنجا که هر بلاک به صورت مستقل مدیریت می‌شود، چندین دنباله منطقی می‌توانند به یک بلاک فیزیکی واحد اشاره کنند. این قابلیت، اشتراک‌گذاری حافظه را به سادگی ممکن می‌سازد. (ارجاع: بخش ۴.۴).

در نتیجه، PagedAttention با حذف اتلاف حافظه، به سیستم اجازه می‌دهد تا درخواست‌های بیشتری را در یک بچ قرار دهد و توان عملیاتی[19] را به شدت افزایش دهد. (ارجاع: شکل ۲ که نشان می‌دهد vLLM نزدیک به صفر اتلاف حافظه دارد).

(ج) اهمیت اشتراک‌گذاری حافظه کش KV در سیستم‌های خدمات‌دهی LLM را مورد بحث قرار دهید. vLLM چگونه اشتراک‌گذاری حافظه را تسهیل می‌کند و این موضوع چه پیامدهایی برای توان عملیاتی کلی سیستم دارد؟ پاسخ خود را با جزئیات موجود در مقاله بیان کنید.

- اهمیت اشتراک‌گذاری حافظه کش KV: اشتراک‌گذاری حافظه کش KV در سناریوهای رایج سرویس‌دهی LLM بسیار حیاتی است، زیرا مستقیماً منجر به صرفه‌جویی در مصرف حافظه می‌شود. سناریوهای کلیدی عبارتند از:

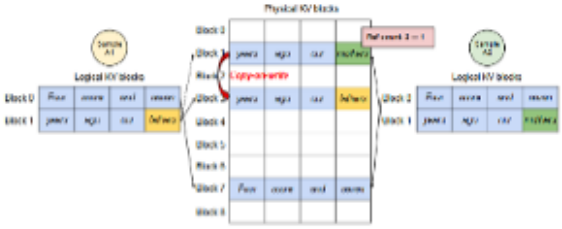

Figure 8. Parallel sampling example.

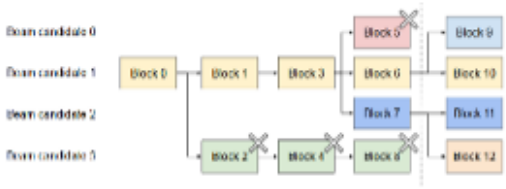

Figure 9. Beam search example.

sample space. The algorithm relies on the *beam width* pa-

- نمونه‌برداری موازی[20]: زمانی که برای یک پرامپت ورودی، چندین خروجی مستقل تولید می‌شود (مثلاً برای ارائه گزینه‌های مختلف به کاربر). در این حالت، حافظه کش KV مربوط به پرامپت اولیه بین تمام خروجی‌ها مشترک است. (ارجاع: بخش ۴.۴).

[19] throughput

[20] Parallel Sampling

- جستجوی پرتوئی[۲۱]: در این الگوریتم، چندین "کاندیدا" برای بهترین خروجی به صورت همزمان بررسی می‌شوند. این کاندیداها نه تنها در پرامپت، بلکه در بخش‌های ابتدایی توالی تولید شده نیز اشتراک دارند. الگوی اشتراک به صورت پویا در هر مرحله تغییر می‌کند. (ارجاع: بخش ۴.۴ و شکل ۹).
- پیشوند مشترک[۲۲]: در بسیاری از کاربردها، یک پیشوند طولانی (مانند دستورالعمل‌ها یا مثال‌ها) به تمام درخواست‌ها اضافه می‌شود. با ذخیره کردن حافظه کش KV این پیشوند، می‌توان از محاسبات تکراری جلوگیری کرد. (ارجاع: بخش ۴.۴ و شکل ۱۰).

.□□ نحوه تسهیل اشتراک‌گذاری حافظه در vLLM: vLLM با استفاده از مکانیزم‌های الهام‌گرفته از سیستم‌عامل، اشتراک‌گذاری حافظه را به طور کارآمد پیاده‌سازی می‌کند:

- جداول بلاک و نگاشت چند به یک: vLLM به هر دنباله یک جدول بلاک منطقی اختصاص می‌دهد. این سیستم اجازه می‌دهد که چندین بلاک منطقی از دنباله‌های مختلف به یک بلاک فیزیکی واحد در حافظه GPU نگاشت شوند. (ارجاع: بخش ۴.۲ و ۴.۴).
- شمارش ارجاع[۲۳]: برای هر بلاک فیزیکی یک شمارنده ارجاع نگهداری می‌شود. این شمارنده تعداد دنباله‌های منطقی را که به آن بلاک اشاره می‌کنند، ثبت می‌کند. یک بلاک فیزیکی تنها زمانی آزاد می‌شود که شمارنده ارجاع آن به صفر برسد. (ارجاع: بخش ۴.۴، زیربخش Parallel sampling).
- کپی در زمان نوشتن[۲۴]: زمانی که یک دنباله نیاز به تغییر محتوای یک بلاک مشترک دارد (مثلاً با اضافه کردن یک توکن جدید)، [۲۵] به جای تغییر بلاک اصلی، یک کپی جدید از آن بلاک ایجاد می‌کند، آن را به دنباله مورد نظر اختصاص می‌دهد و شمارنده ارجاع بلاک اصلی را کاهش می‌دهد. این کار از کپی‌های غیرضروری جلوگیری کرده و تنها در مواقع لزوم انجام می‌شود. (ارجاع: بخش ۴.۴ و شکل ۸).

.□□□ پیامدها برای توان عملیاتی[۲۶]: اشتراک‌گذاری حافظه به طور مستقیم توان عملیاتی کل سیستم را افزایش می‌دهد. این تأثیر به شکل زیر است:

- کاهش مصرف حافظه: با اشتراک‌گذاری، حافظه مورد نیاز برای هر گروه از درخواست‌ها

[21] Beam Search
[22] Shared Prefix
[23] Reference Counting
[24] Copy-on-Write - CoW
[25] vLLM
[26] Throughput

(مثلاً یک درخواست با چندین خروجی موازی) به شدت کاهش می‌یابد. مقاله در شکل ۱۵ نشان می‌دهد که این روش می‌تواند در Parallel Sampling تا ۳۰٪ و در Beam Search تا ۶۶٪ حافظه را صرفه‌جویی کند.

- افزایش اندازه بچ[۲۷]: صرفه‌جویی در حافظه به vLLM اجازه می‌دهد تا تعداد بسیار بیشتری درخواست را به صورت همزمان در یک بچ قرار دهد. (ارجاع: شکل ۱۳ که افزایش چشمگیر تعداد درخواست‌های بچ‌شده را نشان می‌دهد).
- افزایش توان عملیاتی: افزایش اندازه بچ به معنای استفاده بهینه‌تر از قدرت محاسباتی GPU و در نتیجه، افزایش قابل توجه توان عملیاتی (تعداد درخواست‌های پردازش شده در ثانیه) است. مقاله در بخش ۶.۳ و شکل ۱۴ نشان می‌دهد که مزیت توان عملیاتی vLLM نسبت به سیستم‌های دیگر در سناریوهای Beam Search (که اشتراک‌گذاری بیشتری دارند) بسیار بارزتر است. برای مثال، برتری vLLM نسبت به Orca از ۱.۳ برابر در نمونه‌برداری عادی به ۲.۳ برابر در Beam Search با عرض ۶ افزایش می‌یابد.

ویدیوی ارائه نویسندگان مقاله در یک کنفرانس

۲.  با توجه به Multi-Head Attention به پرسش‌های زیر پاسخ دهید (۱۰ نمره):

(آ) چرا در مدل‌های ترنسفورمر از توجه چندسری (Multi-Head Attention) استفاده می‌شود؟ و این سرهای توجه چه نوع اطلاعاتی را می‌توانند یاد بگیرند؟

مدل‌های ترنسفورمر برای یادگیری بهتر و دقیق‌تر از توجه چندسری استفاده می‌کنند. دلایل اصلی آن عبارتند از:

- نمایش‌های متنوع[۲۸]: هر سر توجه می‌تواند بر بخش متفاوتی از دنباله ورودی تمرکز کند و روابط و الگوهای مختلفی را بیاموزد. این تنوع در توجه، باعث افزایش قدرت مدل در درک داده‌های پیچیده می‌شود.
- افزایش ظرفیت مدل[۲۹]: با داشتن چندین سر توجه، مدل می‌تواند اطلاعات بیشتری را به‌صورت همزمان پردازش کند. هر سر می‌تواند در ویژگی خاصی تخصص پیدا کند و در

[27] Batch Size
[28] Diverse Representations
[29] Increased Capacity

نتیجه، نمایش نهایی غنی‌تری حاصل شود.

- پویایی بهتر در یادگیری[۳۰]: یادگیری جنبه‌های مختلف داده توسط سرهای مختلف، به مدل کمک می‌کند تا از بیش‌برازش[۳۱] جلوگیری کرده و عملکرد بهتری روی داده‌های نادیده از خود نشان دهد.
- درک بهتر زمینه[۳۲]: هر سر توجه می‌تواند به زمینه یا جنبه متفاوتی از ورودی توجه کند، که موجب درک بهتر مفاهیم و ظرایف زبانی یا سایر داده‌های ترتیبی می‌شود.
- پردازش تانسوری و موازی[۳۳]: محاسبات مستقل هر سر توجه امکان استفاده از عملیات تانسوری و اجرای موازی را فراهم می‌سازد که موجب افزایش سرعت آموزش و پیش‌بینی می‌شود.
- الگوهای توجه انعطاف‌پذیر[۳۴]: سرهای مختلف می‌توانند به انواع مختلفی از وابستگی‌ها در داده‌ها (مثل وابستگی‌های محلی و سراسری) توجه کنند و این امر موجب افزایش تطبیق‌پذیری مدل در مواجهه با مسائل مختلف می‌شود.

(ب) فرض کنید یک مدل آموزش‌دیده داریم که بر پایه‌ی توجه چندسری (Multi-Head Attention) ساخته شده است و می‌خواهیم برای افزایش سرعت پیش‌بینی، سرهای توجه کم‌اهمیت‌تر را حذف (Prune) کنیم. چگونه می‌توانیم آزمایش‌هایی طراحی کنیم تا اهمیت هر سر توجه را اندازه‌گیری کنیم؟

برخی از روش‌ها عبارتند از:

- آزمون حذف تدریجی[۳۵]: در این روش، هر سر توجه به‌صورت جداگانه غیرفعال می‌شود (خروجی‌اش صفر یا حذف می‌شود) و عملکرد مدل روی داده‌های اعتبارسنجی[۳۶] اندازه‌گیری می‌شود. مراحل:
  - یکی از Head ها را حذف کن.
  - مدل را اجرا کن و دقت یا معیار ارزیابی (Accuracy، BLEU، F1 و ...) را ثبت کن.
  - همین کار را برای تک‌تک Head ها تکرار کن.

[30] Improved Learning Dynamics
[31] Overfitting
[32] Enhanced Contextual Understanding
[33] Tensor Processing & Parallelism
[34] Flexibility in Attention Patterns
[35] Ablation Study
[36] Validation

- کاهش زیاد در عملکرد = اهمیت بالا و کاهش کم یا بدون تأثیر = اهمیت پایین

- تحلیل بر اساس گرادیان‌ها[۳۷]: در این روش، اندازه گرادیان‌های مربوط به وزن‌های هر Head بررسی می‌شود. گرادیان‌های کم نشان می‌دهند که head در طول آموزش به‌روزرسانی خاصی دریافت نکرده و احتمالاً کم‌اهمیت است. مراحل:
  - در طول آموزش یا inference، گرادیان وزن‌های ماتریس‌های Q، K،V برای هر Head را ذخیره کن.
  - میانگین یا نُرم L2 این گرادیان‌ها را محاسبه کن.
  - Head هایی با گرادیان کوچک → احتمالاً بلااستفاده یا کم‌اهمیت هستند.
- آنتروپی توجه[۳۸]: سرهایی که توجه یکنواخت و پخش دارند (توجه به همه توکن‌ها به یک میزان)، اطلاعات خاصی منتقل نمی‌کنند. می‌توان با محاسبه آنتروپی distribution توجه، به این موضوع پی برد. مراحل:
  - ماتریس attention weights هر Head را بگیر.
  - برای هر سطر (query)، آنتروپی آن را محاسبه کن.
  - آنتروپی بالا = توجه پخش Head → کم‌اهمیت
  - آنتروپی پایین = توجه متمرکز Head → مهم‌تر

(ج) حذف سرهای توجه چه اثری روی وظایف پایین‌دستی (مثل طبقه‌بندی یا ترجمه) دارد؟ از چه معیارهایی برای ارزیابی تأثیر حذف سرها استفاده کنیم؟

- کاهش دقت مدلPerformance Degradation: اگر head هایی حذف شوند که اطلاعات مهمی از ورودی را منتقل می‌کردند، ممکن است دقت مدل کاهش یابد، مخصوصاً در وظایف حساس به زمینه[۳۹] مانند ترجمه.
- افزایش سرعت پیش‌بینی[۴۰]: با کاهش تعداد head ها، محاسبات matrix-multiplication برای Q،K و V کاهش یافته و سرعت inference بیشتر می‌شود.
- کاهش مصرف حافظه[۴۱]: حذف head ها باعث می‌شود حافظه GPU/CPU کمتر مصرف شود، خصوصاً در مدل‌های بزرگ مانند BERT یا GPT.

[37] Gradient-based Analysis
[38] Attention Entropy
[39] context-sensitive tasks
[40] Faster Inference
[41] Memory & Resource Efficiency

- احتمال بهبود تعمیم‌پذیری[۴۲]: در برخی موارد، حذف head های غیرمفید می‌تواند باعث کاهش بیش‌برازش و بهبود تعمیم‌پذیری مدل روی داده‌های جدید شود.
- ریسک حذف اطلاعات حیاتی: اگر فرآیند pruning به‌درستی انجام نشود، ممکن است head های مهم حذف شوند و این منجر به افت شدید عملکرد شود.

برای ارزیابی اینکه حذف سرهای توجه چه تأثیری داشته، می‌توان از معیارهای زیر استفاده کرد:

- معیارهای عملکرد مدل:
  - در طبقه‌بندی:
    - Accuracy
    - F1 Score
    - Precision / Recall
  - در ترجمه ماشینی:
    - BLEU Score
    - METEOR / ROUGE
- معیارهای منابع محاسباتی:
  - زمان پیش‌بینی: مقایسه زمان پردازش یک نمونه بین مدل pruning‌شده و مدل اصلی.
  - تعداد پارامترها: بررسی میزان کاهش پارامترهای مدل پس از حذف head ها.
  - تعداد عملیات شناور[۴۳]: مقایسه حجم محاسبات قبل و بعد از pruning.
  - میزان استفاده از حافظه: در مدل‌های بزرگ، حذف head ها می‌تواند مصرف حافظه را کاهش دهد.
- معیارهای تحلیلی و ساختاری
  - تحلیل آنتروپی توجه[۴۴]: بررسی تغییرات در الگوهای توجه پس از حذف head ها.
  - تغییر در بردارهای ویژگی[۴۵]: تحلیل اینکه حذف head ها چه اثری بر نمایش نهایی داده‌ها می‌گذارد.

[42] Generalization
[43] FLOPs
[44] Attention Entropy
[45] Embedding Drift

(د) آیا می‌توان از یادگیری تقویتی [۴۶] برای انتخاب دینامیک سرهای توجه استفاده کرد؟

بله، می‌توان از یادگیری تقویتی برای انتخاب دینامیک (پویا) سرهای توجه در مدل‌های ترنسفورمر استفاده کرد. این ایده در برخی پژوهش‌ها و مقالات نیز پیاده‌سازی شده و به نتایج جالبی منجر شده است.

(اختیاری: می‌توانید از مقاله بهره بگیرید.)

۳. در رابطه با Additive Attention به پرسش‌های زیر پاسخ دهید (۱۰ نمره):

(آ) آیا ایده‌ی خوبی است که در مدل ترنسفورمر، توجه ضرب نقطه‌ای مقیاس‌شده (Scaled Dot-Product Attention) را با توجه جمعی (Additive Attention) جایگزین کنیم؟ چرا؟

در اکثر موارد خیر، ایده‌ی خوبی نیست که در مدل ترنسفورمر، توجه ضرب نقطه‌ای مقیاس‌شده را با توجه جمعی جایگزین کنیم، مگر در موارد خاص. دلیل اصلی آن، بازده محاسباتی پایین‌تر و پیچیدگی بیشتر توجه جمعی است. در کل در مدل‌های ترنسفورمر توجه ضرب نقطه‌ای مقیاس‌شده هم ساده‌تر، هم سریع‌تر و هم بهینه‌تر است.

(ب) آیا می‌توان ترکیبی از این دو نوع توجه استفاده کرد؟

بله، از نظر تئوری و حتی در برخی پژوهش‌ها می‌توان ترکیبی از توجه ضرب نقطه‌ای مقیاس‌شده [۴۷] و توجه جمعی [۴۸] را استفاده کرد. با این حال، چنین ترکیبی باید با هدف خاصی صورت بگیرد و به دقت طراحی شود، زیرا هزینه محاسباتی و پیچیدگی مدل افزایش می‌یابد. زمانی ترکیب می‌تواند مفید باشد که: داده‌ها نویزی یا کم‌ساختار هستند و توجه Additive می‌تواند جزئیات بیشتری را استخراج کند. می‌خواهیم توازن بین سرعت و قدرت نمایش ایجاد کنیم. در تنظیمات Low-Resource (داده کم) هستیم و نیاز به توجه حساس‌تر داریم. قصد طراحی مدل جدید یا پژوهش در معماری‌های ترکیبی را داریم.

(ج) یک توجه چند سر additive با ۳ سر را در نظر بگیرید. ابعاد query، key و value را به ترتیب ۱۰، ۲۰، ۳۰ در نظر بگیرید فرض کنید هر کدام از سرها به ابعاد ۱۰۰ تبدیل شوند. همچنین در نظر داشته باشید که خروجی نهایی ۵۰ میباشد. با فرض اینکه دنباله ورودی ۶۴ تایی باشد، تعداد پارامترها را مشخص کنید.

[46] Reinforcement Learning
[47] Scaled Dot-Product
[48] Additive

برای هر سر، ماتریس‌های تبدیل خطی برای key، query و value دارای ابعاد زیر خواهند بود:

ماتریس key برابر است با:

10×100

ماتریس query برابر است با:

20×100

ماتریس value برابر است با:

30×100

خروجی هر سر دارای ابعاد:

100×64

هر سر در توجه چندسری دارای مجموعه‌ای از پارامترهای قابل یادگیری برای نمایش key، query و value است. تعداد پارامترهای هر سر به صورت زیر محاسبه می‌شود:

ماتریس key:

$100 \times 10 = 1000$ پارامتر

ماتریس query:

$100 \times 20 = 2000$ پارامتر

ماتریس value:

$100 \times 30 = 3000$ پارامتر

تعداد کل پارامترها در هر سر:

$1000 + 2000 + 3000 = 6000$ پارامتر

با داشتن 3 سر، تعداد کل پارامترها برای همه سرها برابر است با:

$6000 \times 3 = 18000$ پارامتر

در نهایت، باید پارامترهایی را برای لایه‌ی پروجکشن خروجی در نظر بگیریم (که خروجی‌های به‌هم‌پیوسته‌ی همه‌ی سرها را گرفته و به ابعاد خروجی نهایی نمایش می‌دهد):

$3 \times 100 \times 50 = 15000$ پارامتر

بنابراین، تعداد کل پارامترها برای توجه چندسری (با 3 سر)، با توجه به ابعاد مشخص‌شده و طول توالی ورودی، برابر است با:

(پروجکشن خروجی) $15000 +$ (توجه چندسری) $18000 = 33000$ پارامتر

۴.  در رابطه با کاربرد مدل‌های transformer در سری‌های زمانی به سوالات زیر پاسخ دهید (۲۰ نمره):

(آ) چه زمانی استفاده از ترنسفورمر در سری زمانی مناسب‌تر از استفاده از LSTM است؟

) استفاده از ترنسفورمرها به‌جای LSTM ها در سری‌های زمانی زمانی مناسب‌تر است که برخی از شرایط زیر برقرار باشد:

- زمانی‌که روابط بلندمدت[۴۹] اهمیت زیادی دارند.
- زمانی‌که طول دنباله زیاد باشد. (Long Sequences)
- زمانی‌که داده‌ی زیادی در دسترس باشد. (Data-Hungry Model)
- زمانی‌که ویژگی‌های متعددی در هر گام زمانی وجود دارد. (Multivariate Time Series)
- در صورت نیاز به تفسیر مدل. (Interpretability)

(ب) چگونه داده‌های سری زمانی باید برای ورودی به ترنسفورمر پیش‌پردازش شوند؟

برای استفاده از ترنسفورمر در سری‌های زمانی، باید داده‌ها را به‌گونه‌ای پیش‌پردازش کنیم که با ساختار ورودی ترنسفورمر (که در اصل برای متن طراحی شده) سازگار شود.

- تقسیم سری زمانی به پنجره‌های هم‌طول
- نرمال‌سازی داده‌ها
- اضافه کردن اطلاعات زمانی (Positional Encoding)
- تنظیم فرمت [batch, sequence_length, features]
- آماده‌سازی خروجی/برچسب برای آموزش
- ساخت ماسک‌ها برای جلوگیری از نگاه به آینده در صورت نیاز

(ج) چه تفاوتی بین ترنسفورمر استاندارد و ترنسفورمر مخصوص سری زمانی (مانند Time Series Transformer یا Informer) وجود دارد؟

در ترنسفورمر استاندارد هدف اصلی طراحی مدل‌سازی توالی‌ها در NLP (مانند ترجمه، خلاصه‌سازی و غیره) می‌باشد در حالی که در ترنسفورمر مخصوص سری زمانی هدف اصلی پیش‌بینی داده‌های سری زمانی[۵۰]، است و ترنسفورمر استاندارد هیچ سازوکار خاصی برای جدا کردن trend و seasonality ندارد. همچنین به دلیل تفاوت آن‌ها در نوع مکانیزم توجه اغلب ترنسفورمرهای مخصوص سری زمانی پیچیدگی زمانی کمتری از ترنسفورمر استاندارد دارند.

[49] Long-Term Dependencies

[50] Forecasting

(د) چگونه می‌توان از ترنسفورمر برای پیش‌بینی چند مرحله‌ای (multi-step forecasting) در سری‌های زمانی استفاده کرد؟

برای پیش‌بینی چند مرحله‌ای با ترنسفورمر، می‌توان از دو استراتژی کلی استفاده کرد:

- Encoder-Decoder Structure:
  - Encoder: تاریخچه داده‌ها (مثلاً ۹۶ تایم‌استپ گذشته) را می‌گیرد.
  - Decoder: به جای یک مقدار، کل دنباله خروجی (مثلاً ۲۴ تایم‌استپ آینده) را به‌صورت یکجا پیش‌بینی می‌کند.

  این روش موازی بوده و برای مسائل multi-horizon forecasting بسیار مؤثر است.
- Direct Multi-Output: مدل یک بار اجرا شده و چند مقدار آینده را به‌صورت مستقیم خروجی می‌دهد (vector output). مناسب برای سرعت و کاهش خطاهای انباشته‌شده [51] در پیش‌بینی‌های autoregressive

(ه) نحوه‌ی عملکرد مدل iTransformer جهت وظیفه‌ی Time Series Forecasting را توضیح دهید. (میتوانید از مقاله بهره بجویید.)

iTransformer یکی از مدل‌های پیشرفته برای پیش‌بینی سری زمانی است که بر پایه ایده‌ی Instance-based Patch Attention طراحی شده است.

ویژگی‌های کلیدی iTransformer:

- Patch Embedding: به‌جای استفاده مستقیم از نقاط زمانی، iTransformer داده‌ها را به پچ‌هایی (مثلاً با طول ۱۶ یا ۳۲) تقسیم کرده و مانند تصویر یا متن، آن‌ها را به صورت embedding درمی‌آورد.
- Instance-based Attention: برخلاف ترنسفورمرهای سنتی که اطلاعات را بین کل batch به اشتراک می‌گذارند، iTransformer تمرکز خود را فقط روی هر نمونه جداگانه (instance-wise) نگه می‌دارد تا تعمیم‌پذیری و دقت پیش‌بینی را افزایش دهد.
- تطبیق بهتر با تغییرات روند[52]: با استفاده از ساختار Patch و نداشتن وابستگی ترتیبی مستقیم، iTransformer توانایی درک بهتر نوسانات پیچیده را دارد.
- پیش‌بینی چند مرحله‌ای: مدل در خروجی چند مقدار (مثلاً ۲۴ گام زمانی آینده) را به‌صورت مستقیم و بدون ساختار autoregressive پیش‌بینی می‌کند.

[51] accumulated errors

[52] Trend

سوالات عملی

۵.  در این تمرین با مدل ViT برای دسته‌بندی تصاویر آشنا خواهید شد. شما یک مدل pretrained را با استفاده از Hugging Face بارگذاری می‌کنید، وزن‌های attention آن را تجزیه و تحلیل می‌کنید و برای درک بهتر مکانیزم توجه و رفتار آن در مدل پیش آموخته شده، وزن‌های یادگیری شده را نمایش خواهید داد. در بخش بعدی آن را روی دیتاست CIFAR-10 آموزش خواهید داد (fine-tune) و در نهایت نقش attention headها را بررسی می‌کنید. در این سوال از نوتبوک Q5.ipynb استفاده کنید.(۲۵ نمره)

(آ) از کتابخانه transformers در Hugging Face برای بارگذاری مدل pretrained استفاده کنید (ترجیحا مدل google/vit-base-patch16-224). در این بخش پس از انجام پیش‌پردازش تصویر مورد نظر، با استفاده از مدل پیش‌آموزش دیده خروجی مدل را بدست آورده و ۵ کلاس برتر پیش‌بینی شده را همراه با احتمالات پیش‌بینی محاسبه کنید.

(ب) با فراخوانی مدل می‌توانید به وزن‌های مکانیزم توجه برای تصویر موردنظر دسترسی داشته باشید. در این بخش attention weights مربوط به توکن [CLS] را استخراج کنید و نقشه‌های توجه این توکن را برای تمام لایه و headها به صورت جداگانه نمایش دهید.

(ج) Attention Rollout روشی برای مصورسازی و تفسیر مکانیزم توجه در مدل‌های Transformer است. در این روش با ضرب تجمعی ماتریس‌های attention در لایه‌ها، مسیر توجه از ورودی تا خروجی مدل به‌صورت یکپارچه نمایش داده می‌شود. با استفاده از روش attention rollout و توضیحات داخل نوتبوک جریان تاثیر هر patch از تصویر روی توکن cls را در طول لایه‌ها نمایش دهید.

(د) مدل پیش‌آموخته شده را بر روی دیتاست CIFAR-10 آموزش (fine-tune) دهید و دقت آن را گزارش کنید.

(ه) در این بخش پس از آموزش روی دادگان بررسی کنید که دور ریختن یک یا چند head چه تاثیری در عملکرد مدل ایجاد می‌کند. با استفاده از داده validation تحلیل کنید کدام headها نقش مهمتری در تصمیم‌گیری مدل دارند.

برای استفاده از پاسخنامه به پوشه Q5-HW4-solution مراجعه کنید که از پاسخ آقای پولایی استفاده شده است.

۶.  در این تمرین، مدل ترجمه ماشینی مبتنی بر توجهی را آموزش خواهید داد تا کلمات را از انگلیسی به Pig-Latin ترجمه کنید.Pig-Latinیک بازی زبانی است که در آن قوانین به صورت مستقل برای هر کلمه اعمال می شود: (۲۵ نمره)

- اگر اولین حرف یک کلمه، حرف بی صدای انگلیسی باشد، آن حرف به انتهای کلمه منتقل شده و حروف ayبه انتهای کلمه اضافه می شوند: team → eamtay .
- اگر اولین حرف، یک حرف صدادار انگلیسی باشد، کلمه بدون تغییر باقی می ماند و حروف way به انتهای کلمه اضافه می شوند: impress → impressway.
- برخی از جفت حروف مانند sh به عنوان یک بلوک در نظر گرفته میشوند و به صورت کل به انتهای رشته منتقل میشوند: shopping → oppingshay

هدف این است که مدل ترجمه ماشینی قوانین را به طور ضمنی از طریق جفت های کلمات (English، Pig-Latin) که source کلمه انگلیسی و target ترجمه آن به Pig-Latin است، یاد بگیرد.

داده ها:

در این تمرین از دو مجموعه داده استفاده خواهید کرد:

- واژگان مجموعه داده کوچک شامل ۲۹ نشانه است: ۲۶ حرف استاندارد الفبا (همه با حروف کوچک)، نماد خط تیره "-" و دو نشانه <SOS> و <EOS> که به ترتیب شروع و پایان یک دنباله را نشان می‌دهند. مجموعه داده شامل ۳۱۹۸ جفت (English، Pig-Latin) منحصر به فرد است.
- مجموعه داده بزرگ‌تر، شامل ۲۰،۰۰۰ کلمه انگلیسی پرکاربردتر است که با مجموعه داده قبلی ترکیب می‌شود و ۲۲۴۰۲ کلمه منحصر به فرد به دست می‌آید

(آ) به بخش scaled dot product attention در نوتبوک pigLatin مراجعه کرده و بخش‌های مشخص‌شده را تکمیل کنید.

(ب) مدل Transformer را با استفاده از hidden size های ۳۲ و ۶۴ و با استفاده از مجموعه داده کوچک و بزرگ (در مجموع ۴ اجرا) اجرا کنید و اثرات افزایش ظرفیت مدل از طریق hidden size و افزایش اندازه مجموعه داده را گزارش کنید.

(ج) به معماری Transformer در شکل زیر نگاه کنید. در هر لایه ابتدا CausalScaledDotAttention را به ورودی‌های decoder و سپس ScaledDotAttention را به encoder annotations اعمال می‌کنیم. __init__ بخش decoder را طوری تغییر دهید که فقط از ScaledDotAttention استفاده کند. نتایج خود را حالت قبلی مقایسه کنید.

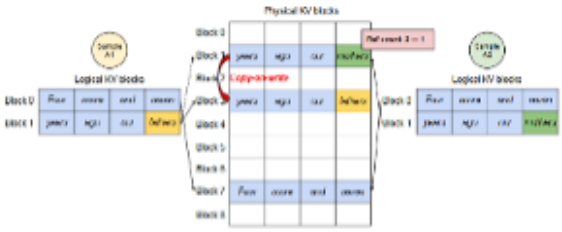

Figure 8. Parallel sampling example.

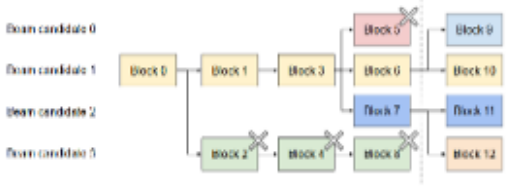

Figure 9. Beam search example.

sample space. The algorithm relies on the *beam width* pa-

برای استفاده از پاسخنامه به پوشه Q6-HW4-solution مراجعه کنید که از پاسخ آقای حسین زاده استفاده شده است.