

به نام خدا



دانشگاه تهران

دانشکده مهندسی برق و کامپیووتر

مدل‌های مولد عمیق

تمرین چهارم درس

محمد طاها مجلسی کوپایی	نام و نام خانوادگی
۸۱۰۱۰۱۵۰۴	شماره دانشجویی
۱۴۰۳/۱۰/۱۴	تاریخ ارسال گزارش

فهرست

مدل زبان بصری (Vision-Language Model)

۱۴	۱.۱. بخش اول – مدل PaliGemma
۱۵	۱.۱.۱. زیربخش اول – مدل VLM
۱۱	۱.۱.۲. زیربخش دوم – رمزگذار تصویر SigLIP
۲۰	۱.۱.۳. زیربخش سوم – پیشآموزش Pre-training
۲۹	۱.۱.۴. زیربخش چهارم – انتقال یادگیری Transfer Learning
۴۹	۱.۱.۵p. زیربخش پنجم – پیاده‌سازی و آموزش مدل
۸۰	۲. هم‌خوانی جریان (Flow Matching) – سوال ۲
۸۰	زیر بخش اول
۸۴	زیر بخش دوم
۸۸	زیر بخش سوم
۹۳	مراجع

بخش اول مدل : PaliGemma

مدل **PaliGemma** یک مدل پیشرفته در حوزه مدل‌های بینایی-زبانی (Vision-Language Models) است که با هدف ترکیب و یکپارچه‌سازی داده‌های بصری و متنی برای انجام وظایف پیچیده‌ای مانند پاسخ به سوالات بصری (VQA) و توصیف تصاویر طراحی شده است. این مدل با بهره‌گیری از معماری مقیاس‌پذیر خود، قادر به پردازش و

درک همزمان اطلاعات تصویری و زبانی می‌باشد که آن را به ابزاری قدرتمند در کاربردهای متتنوع هوش مصنوعی تبدیل کرده است.

معماری مدل Paligemma شامل دو جزء اصلی است: **رمزنگار تصویری SigLIP** و **رمزنگار زبانی SigLIP**. رمزنگار با استفاده از تکنیک‌های پیشرفته مانند استفاده از توابع سیگموئید برای پیش‌تربیت، ویژگی‌های کلیدی تصاویر را استخراج کرده و آنها را به فضای تعابیه مشترک منتقل می‌کند که با داده‌های متنه همسو است. این امر امکان تعامل موثر بین داده‌های بصری و زبانی را فراهم می‌آورد و به مدل اجازه می‌دهد تا به صورت هماهنگ و دقیق به سوالات مرتبط با تصاویر پاسخ دهد. از سوی دیگر، رمزنگار زبانی با استفاده از معماری ترانسفورمر، توانایی تولید متن‌های مرتبط و دقیق را بر اساس ویژگی‌های استخراج شده از تصاویر دارد.

یکی از ویژگی‌های برجسته مدل Paligemma، تنظیم دقیق با بهره‌وری پارامتری (PEFT) است که از روش‌هایی مانند LoRA و QLoRA برای بهینه‌سازی و کاهش نیازمندی‌های محاسباتی استفاده می‌کند. این روش‌ها به مدل اجازه می‌دهند تا با تغییرات کم در پارامترها، به سرعت و با کارایی بالا به وظایف خاصی مانند پاسخ به سوالات بصری تنظیم شود. علاوه بر این، Paligemma با استفاده از تکنیک‌های **تطابق جریان (Flow Matching)** توانسته است فرآیند تولید داده‌های جدید را بهینه کرده و قابلیت‌های مولد خود را افزایش دهد، که این امر باعث بهبود کیفیت و دقت پاسخ‌های زبانی مدل می‌شود.

در نهایت، مدل Paligemma با ترکیب معماری پیشرفته، تکنیک‌های بهینه‌سازی و قابلیت‌های مولد قوی، توانسته است جایگاه خود را به عنوان یکی از مدل‌های برجسته در حوزه بینایی-زبانی تثبیت کند. این مدل با ارائه پاسخ‌های دقیق و مرتبط به سوالات بصری، امکان استفاده گسترده‌ای در زمینه‌هایی مانند دستیارهای هوشمند، سیستم‌های مدیریت محتوا و تحلیل تصاویر پزشکی فراهم آورده است. همچنین، قابلیت مقیاس‌پذیری Paligemma آن را برای توسعه‌های آینده در زمینه هوش مصنوعی و یادگیری چندرسانه‌ای بسیار مناسب می‌سازد.

سوال اول:

زیربخش‌های سنتی تصویری یا متنی را توضیح دهید. همچنین، مدل‌های بینایی-زبانی (VLM) و تفاوت آن‌ها با سایر مدل‌ها را شرح دهید.

پاسخ:

۱. مدل‌های سنتی تصویری

مدل‌های سنتی تصویری عمدهاً بر پردازش و تحلیل داده‌های بصری مانند تصاویر و ویدئوها متمرکز هستند. این مدل‌ها از معماری‌های مختلفی بهره می‌برند که هر کدام برای وظایف خاصی طراحی شده‌اند.

(الف) شبکه‌های عصبی پیچشی (CNNs)

- **معماری:** شبکه‌های عصبی پیچشی شامل لایه‌های پیچشی (Convolutional Layers)، لایه‌های فعال‌سازی (Activation Layers)، و لایه‌های جمع‌کننده (Pooling Layers) هستند.
- **عملکرد:** این شبکه‌ها ویژگی‌های محلی تصاویر را استخراج کرده و به تدریج به ویژگی‌های سطح بالاتر تبدیل می‌کنند.
- **کاربردها:** تشخیص اشیاء، طبقه‌بندی تصاویر، شناسایی چهره، و تحلیل ویدئو.

(ب) ترانسفورمرهای بینایی (Vision Transformers - ViTs)

- **معماری:** برخلاف CNN‌ها که از فیلترهای محلی استفاده می‌کنند، ViTs از مکانیزم توجه (Attention) بهره می‌برند تا وابستگی‌های بلندمدت در تصاویر را مدل‌سازی کنند.
- **عملکرد:** تصاویر را به تکه‌های کوچک (patches) تقسیم کرده و هر تکه را به یک بردار تعبیه می‌کند. سپس با استفاده از ترانسفورمرها، این بردارها را پردازش می‌کند.
- **کاربردها:** طبقه‌بندی تصاویر، تشخیص اشیاء، و سایر وظایف بینایی کامپیوتري.

۲. مدل‌های سنتی متنی

مدل‌های سنتی متنی بر پردازش داده‌های زبانی مانند متن‌های نوشتاری متمرکز هستند. این مدل‌ها از معماری‌های مختلفی بهره می‌برند که هر کدام برای وظایف خاصی طراحی شده‌اند.

(الف) مدل‌های مبتنی بر RNN (شبکه‌های عصبی بازگشته)

- **معماری:** شامل لایه‌های بازگشته (Recurrent Layers) که به صورت توالی‌ای داده‌ها را پردازش می‌کنند.
- **عملکرد:** توانایی مدل‌سازی وابستگی‌های زمانی در داده‌های متنی.
- **کاربردها:** ترجمه ماشینی، پیش‌بینی کلمه بعدی، و تحلیل احساسات.

ب) مدل‌های مبتنی بر ترنسفورمر (Transformer-Based Models)

- **معماری:** استفاده از مکانیزم توجه (Attention Mechanism) برای پردازش توالی‌ها به صورت همزمان و بدون نیاز به پردازش ترتیبی.
- **عملکرد:** توانایی درک روابط پیچیده بین کلمات در یک جمله.
- **کاربردها:** ترجمه ماشینی، تولید متن، پاسخ به سوالات، و تحلیل معنایی.

۳. مدل‌های بینایی-زبانی (VLM) و تفاوت آن‌ها با مدل‌های سنتی

مدل‌های بینایی-زبانی (Vision-Language Models) یا VLMs ترکیبی از پردازش داده‌های بصری و متنی هستند که به مدل‌ها امکان می‌دهند تا به صورت همزمان به تحلیل و تولید محتوا در هر دو حوزه بپردازند. این مدل‌ها با استفاده از معماری‌های ترکیبی، قابلیت درک و تعامل میان تصاویر و متن را دارا هستند.

تفاوت‌ها با مدل‌های سنتی:

- **یکپارچگی داده‌ها:** برخلاف مدل‌های سنتی که تنها بر یک نوع داده (تصویری یا متنی) متمرکز هستند، VLM‌ها به پردازش و تعامل میان داده‌های بصری و متنی می‌پردازند.
- **فضای تعبیه مشترک:** VLM‌ها معمولاً داده‌های بصری و متنی را در فضای تعبیه مشترکی نگاشت می‌کنند که امکان تعامل و ترکیب اطلاعات از هر دو حوزه را فراهم می‌آورد.
- **کاربردهای چندرسانه‌ای:** این مدل‌ها قادر به انجام وظایفی مانند پاسخ به سوالات بصری (VQA)، توصیف تصاویر، و تولید محتوای چندرسانه‌ای هستند که نیازمند درک ترکیبی از تصویر و متن می‌باشند.

مدل‌های معروف VLM:

- .1 **DALL-E:** مدل مولد که توانایی تولید تصاویر جدید بر اساس توضیحات متنی را دارد.
- .2 **Imagen:** مدل مولد با تأکید بر کیفیت بالاتر تصاویر تولید شده.
- .3 **Paligemma Flow PEFT Matching:** مدل چندمنظوره و مقیاس‌پذیر با استفاده از تکنیک‌های پیشرفت‌ههای مانند PEFT و Flow.

سوال دوم:

رویکردهای مختلف در مدل‌های بینایی-زبانی (VLM) را توضیح دهد. یکی از این رویکردها، رویکرد **end-to-end** است و رویکرد دیگر رویکرد ماژولار می‌باشد. مدل‌های مانند DALL-E و Paligemma را با این دو رویکرد توضیح دهد و مزایا و معایب هر کدام را به طور کلی مقایسه کنید.

پاسخ:

۱. معرفی رویکردهای End-to-End و ماژولار در مدل‌های بینایی-زبانی

در حوزه مدل‌های بینایی-زبانی (VLMs) یا Vision-Language Models، دو رویکرد اصلی برای طراحی و پیاده‌سازی مدل‌ها وجود دارد: **رویکرد End-to-End** و **رویکرد ماژولار**. هر یک از این رویکردها دارای ویژگی‌ها، مزایا و معایب خاص خود هستند که در ادامه به تفصیل به آن‌ها پرداخته می‌شود.

الف) رویکرد End-to-End

تعریف:

در رویکرد End-to-End، تمامی اجزای مدل از پردازش داده‌های ورودی (متنی و تصویری) تا تولید خروجی (مثلاً تصویر یا متن) به صورت یکپارچه و در یک معماری واحد آموزش داده می‌شوند.

ویژگی‌ها:

- **یکپارچگی کامل:** تمامی بخش‌های مدل به صورت همزمان و یکپارچه آموزش داده می‌شوند، بدون نیاز به تعاملات جداگانه بین ماژول‌ها.
- **سادگی پیاده‌سازی:** طراحی و پیاده‌سازی مدل ساده‌تر است زیرا نیازی به مدیریت تعاملات میان بخش‌های مختلف مدل نیست.
- **یادگیری همزمان:** مدل قادر است به طور همزمان از داده‌های متنی و تصویری برای بهبود عملکرد استفاده کند.

مزایا:

- **عملکرد بالا در وظایف ترکیبی:** توانایی یادگیری روابط پیچیده بین داده‌های متنی و تصویری به دلیل آموزش مشترک تمامی بخش‌ها.

- خلاقیت و نوآوری: قابلیت تولید خروجی‌های خلاقانه و متنوع بر اساس ترکیب داده‌های متنی و تصویری.
- پایداری در آموزش: مدل به دلیل آموزش همزمان تمامی بخش‌ها، ممکن است در یادگیری تعمیم‌پذیری بهتری داشته باشد.

معایب:

- نیازمندی‌های محاسباتی بالا: آموزش مدل‌های End-to-End معمولاً نیازمند منابع محاسباتی بسیار زیادی است.
- انعطاف‌پذیری کمتر: تغییر یا بهبود یک بخش خاص از مدل نیازمند بازآموزی کل مدل است.
- چالش‌های تعمیم‌پذیری: ممکن است در مواجهه با داده‌های جدید یا وظایف متفاوت عملکرد مناسبی نداشته باشد.

(ب) رویکرد ماژولار

تعریف:

در رویکرد ماژولار، مدل به چندین بخش مستقل تقسیم می‌شود که هر کدام وظیفه خاصی را انجام می‌دهند و با یکدیگر از طریق رابطه‌ای مشخص تعامل می‌کنند.

ویژگی‌ها:

- تفکیک وظایف: هر ماژول به صورت مستقل برای انجام وظایف خاصی طراحی و آموزش داده می‌شود.
- انعطاف‌پذیری بالا: امکان تغییر یا بهبود هر بخش به صورت مستقل بدون نیاز به بازآموزی کل مدل.
- مدیریت بهتر منابع: می‌توان منابع محاسباتی را به طور بهینه‌تری تخصیص داد، زیرا هر ماژول می‌تواند به صورت جداگانه بهینه‌سازی شود.

مزایا:

- انعطاف‌پذیری بالا: قابلیت تغییر، بهبود یا جایگزینی هر بخش به صورت مستقل، بدون تأثیر بر سایر بخش‌ها.
- مدیریت بهینه منابع: امکان بهینه‌سازی و تخصیص منابع به هر ماژول بر اساس نیازهای خاص آن.
- قابلیت توسعه: افزودن ماژول‌های جدید برای وظایف جدید بدون تأثیر مستقیم بر سایر بخش‌ها امکان‌پذیر است.

معایب:

- **پیچیدگی پیاده‌سازی:** نیاز به طراحی رابطه‌ای مشخص و مدیریت تعامل میان مازول‌ها که می‌تواند پیچیده باشد.
- **هماهنگی بین مازول‌ها:** تضمین هماهنگی و تعامل بهینه میان بخش‌های مختلف مدل ممکن است چالش‌برانگیز باشد.
- **کارایی کمتر در برخی موارد:** عدم یکپارچگی کامل ممکن است در برخی وظایف خاص، کارایی مدل را کاهش دهد.

۲. مقایسه مدل‌های DALL-E و Paligemma و مازولار

الف) مدل E (رویکرد DALL-E)

تعریف:

یک مدل مولد بینایی-زبانی است که توسط OpenAI توسعه یافته و توانایی تولید تصاویر جدید بر اساس توضیحات متنی را دارد.

رویکرد:

از رویکرد End-to-End استفاده می‌کند که در آن تمامی بخش‌های مدل از ورودی متنی تا خروجی تصویری به صورت یکپارچه آموزش داده می‌شوند.

ویژگی‌ها:

- **معماری ترنسفورمر:** استفاده از معماری ترنسفورمر برای ترکیب داده‌های متنی و تصویری.
- **تولید مولد:** توانایی تولید تصاویر خلاقانه و منحصر به فرد بر اساس توضیحات متنی.
- **پیش‌تربیت گستردگی:** آموزش بر روی مجموعه داده‌های بزرگ تصاویر و توضیحات متنی.

مزایا:

- **خلاقیت بالا:** قابلیت تولید تصاویر نوآورانه و متنوع بر اساس توضیحات متنی.
- **یکپارچگی متن و تصویر:** توانایی درک و ترکیب دقیق بین داده‌های متنی و تصویری.
- **پایداری در آموزش:** مدل به دلیل آموزش همزمان تمامی بخش‌ها، قابلیت یادگیری روابط پیچیده بین متن و تصویر را دارا است.

معايير:

- نیازمندی‌های محاسباتی زیاد: آموزش و استنتاج مدل نیازمند منابع محاسباتی بالا است.
- انعطاف‌پذیری کم: تغییر یا بهبود یک بخش خاص از مدل نیازمند بازمی‌گیری کامل مدل است.
- پتانسیل تولید ناصحیح: گاهی اوقات تصاویر تولید شده ممکن است دقیقاً با توضیحات متین مطابقت نداشته باشند.

ب) مدل **Paligemma** (رویکرد ماژولار)

تعريف:

Paligemma یک مدل بینایی-زبانی چندمنظوره و مقیاس‌پذیر است که با استفاده از تکنیک‌های پیشرفته مانند PEFT (تنظیم دقیق با بهره‌وری پارامتری) و Flow Matching طراحی شده است.

رویکرد:

Paligemma از رویکرد ماژولار استفاده می‌کند که در آن مدل به بخش‌های مستقل مانند رمزگذار تصویری SigLIP و رمزگشا زبانی تقسیم می‌شود که هر کدام به صورت جداگانه بهینه‌سازی می‌شوند.

ویژگی‌ها:

- **معماری مقیاس‌پذیر:** قابلیت افزایش اندازه و پارامترهای مدل بدون کاهش کارایی، که امکان تعمیم‌پذیری و بهبود عملکرد را فراهم می‌آورد.
- **تنظیم دقیق با PEFT:** استفاده از تکنیک‌هایی مانند LoRA و QLoRA برای تنظیم دقیق مدل با مصرف حافظه و محاسبات کمتر.
- **تطابق جریان (Flow Matching):** استفاده از تکنیک‌هایی مانند PEFT برای بهبود کیفیت و کارایی تولید داده‌های چندرشته‌ای.

مزایا:

- **مقیاس‌پذیری بالا:** امکان توسعه و تنظیم مدل برای وظایف خاص با مصرف منابع کمتر.
- **کارایی بهینه:** استفاده از تکنیک‌های PEFT باعث کاهش نیازمندی‌های محاسباتی و حافظه می‌شود، که امکان استفاده از مدل‌های بزرگ را در محیط‌های محدود فراهم می‌آورد.
- **توانایی مولد قوی:** تطابق جریان (Flow Matching) بهبود کیفیت و کارایی تولید داده‌های مولد را تضمین می‌کند.

- انعطاف‌پذیری بالا: امکان تغییر یا بهبود هر بخش به صورت مستقل بدون نیاز به بازآموزی کل مدل.

معايير:

- پیچیدگی پیاده‌سازی: نیاز به طراحی و مدیریت رابطه‌ای مشخص میان مأذول‌ها که می‌تواند پیچیده باشد.
- هماهنگی بین مأذول‌ها: تضمین هماهنگی و تعامل موثر میان بخش‌های مختلف مدل ممکن است چالش‌برانگیز باشد.
- کمبود یکپارچگی: ممکن است در برخی وظایف خاص، کارایی کمتر از مدل‌های End-to-End داشته باشد.

۳. مقایسه کلی بین مدل‌های DALL-E و Imagen

ویژگی‌ها	DALL-E	Imagen
رویکرد	End-to-End	End-to-End
معماری	ترنسفورمر	ترنسفورمر پیشرفته‌تر
کیفیت تصاویر	خوب	بسیار بالا
(Fine-Tuning) تنظیم دقیق	محدود	محدود
پایداری	متوسط	بالا
نیازمندی‌های محاسباتی	بالا	بسیار بالا
(Flow Matching) تطبیق جریان	ندارد	ندارد
مزایا	خلاصه‌یابی، یکپارچگی متن و تصویر	کیفیت بالا، پایداری بیشتر
معايير	نیاز به منابع زیاد، تولید ناصحیح	پیچیدگی بیشتر، نیاز به داده‌های بیشتر

۴. نتیجه‌گیری

مدل‌های بینایی-زبانی مانند DALL-E و Imagen با استفاده از رویکرد End-to-End توانسته‌اند قابلیت‌های مولد قابل توجهی را ارائه دهند که شامل تولید تصاویر خلاقانه و با کیفیت بالا بر اساس توضیحات متنی است. این مدل‌ها با یکپارچگی کامل بین بخش‌های متنی و تصویری، توانایی یادگیری روابط پیچیده بین این دو نوع داده را دارا هستند. با این حال، نیازمندی‌های محاسباتی بالا و انعطاف‌پذیری محدود، از معایب اصلی این رویکردها به شمار می‌روند.

از سوی دیگر، مدل Paligemma با استفاده از رویکرد مازولار و بهره‌گیری از تکنیک‌های پیشرفته مانند PEFT و Flow Matching، توانسته است تعادل بین کارایی، مقیاس‌پذیری و کیفیت مولد برقرار کند. این مدل با امکان تنظیم دقیق و بهینه‌سازی بخش‌های مختلف به صورت مستقل، انعطاف‌پذیری بالاتری را ارائه می‌دهد و می‌تواند با منابع محاسباتی کمتر نیز کارایی مناسبی داشته باشد. با این حال، پیچیدگی پیاده‌سازی و نیاز به هماهنگی دقیق میان مازول‌ها از چالش‌های اصلی این رویکرد به شمار می‌رود.

نتیجه نهایی:

انتخاب رویکرد مناسب بین End-to-End و مازولار بستگی به نیازها، منابع موجود و ویژگی‌های خاص پروژه دارد. اگر تمرکز بر تولید تصاویر خلاقانه و با کیفیت بالا با منابع محاسباتی کافی باشد، مدل‌هایی مانند DALL-E و Imagen با رویکرد End-to-End می‌توانند گزینه‌های مناسبی باشند. اما اگر نیاز به انعطاف‌پذیری بیشتر، تنظیم دقیق و بهینه‌سازی منابع باشد، مدل‌هایی مانند Paligemma با رویکرد مازولار می‌توانند انتخاب بهتری باشند.

SIGLIP IMAGE ENCODER زیر بخش دوم

در ادامه، متن پاسخ را با جزئیات کامل، به صورت روان و قابل فهم به زبان فارسی ارائه می‌کنیم:

سؤال اول

«معماری مدل چگونه است؟ فرایند آموزش فضای مشترک تعییه شده (Joint Embedding Space) چگونه کار می‌کند؟ و از چه تابع ضرری (Loss Function) برای آموزش مدل SigLIP استفاده می‌شود؟»

پاسخ:

مدل Paligemma (SigLIP (Sign Language Image Processing) به کار می‌رود، در اصل نسخه‌ای تغییر یافته از مدل شناخته‌شده CLIP (Contrastive Language–Image Pretraining) است. هر دو مدل، تصاویر و داده‌های متنی را در یک فضای مشترک مرتب می‌کنند تا بتوان جفت‌های تصویر-متن مشابه را به راحتی مقایسه یا بازیابی کرد. تمرکز اصلی SigLIP روی تشخیص و پردازش زبان اشاره است، اما مبنای روش آن همان CLIP است.

در ادامه، مروری عمیق بر معماری، سازوکار آموزش، و تابع ضرر در SigLIP خواهیم داشت:

(1) معماری (Architecture)

: (Image Encoder)

- در SigLIP، انکودر تصویر معمولاً از یک شبکه‌ی عصبی کانولوشنی (CNN) یا یک ویژن تنسفورمر (ViT) استفاده می‌کند:

■ CNN‌ها معماری‌های مرسومی برای پردازش تصویر هستند و با لایه‌های

کانولوشن، ویژگی‌های سطح بالاتر را به تدریج از پیکسل‌های خام استخراج می‌کنند.

■ ViT، که رویکرد جدیدتری است، تصویر را همانند مجموعه‌ای از وصله‌ها

(Patch) در نظر می‌گیرد (مشابه توکن‌ها در پردازش زبان طبیعی) و از مکانیزم تنسفورمر برای بررسی ارتباط بین این قطعات تصویری استفاده می‌کند.

- این انکودر، تصاویر ورودی را به بردارهای پرمحتوا و با ابعاد بالا تبدیل می‌کند که بیانگر محتوای معنایی تصویر هستند.

: (Text Encoder)

- انکودر متن معمولاً بر پایه‌ی یک مدل تنسفورمری (نظیر BERT یا GPT) است. این مدل، توصیف‌های زبانی (یا برچسب‌های متنی) را پردازش کرده و آن‌ها را به یک فضای برداری با ابعاد بالا نگاشت می‌کند؛ فضایی که از نظر ابعاد با انکودر تصویر هم خوانی دارد.

- این فرایند باعث می‌شود توصیف‌های متنی (مثلًا توضیحات زبان اشاره یا برچسب‌ها) به بردارهایی تبدیل شوند که بتوان آن‌ها را با بردارهای تصویری مقایسه کرد.

: (Joint Embedding Space)

- نقطه‌ی کلیدی در CLIP و SigLIP، همین فضای مشترک است؛ جایی که هم انکودر تصویر و هم انکودر متن، ورودی‌های خود (تصویر و متن) را به فضای یکسانی نگاشتند.
 - در این فضا، تصاویری که با یک توصیف متنی خاص مرتبط هستند، باید به هم نزدیک باشند و جفت‌های غیرمرتب از هم فاصله بگیرند.
 - این ویژگی، امکان انجام کارهایی نظیر بازیابی میان‌مدلی (Cross-Modal Retrieval) را فراهم می‌کند؛ مثلاً یافتن تصاویر بر اساس متن یا یافتن متون بر اساس تصویر.
-

(۲) آموزش فضای مشترک تعبیه‌شده (Training the Joint Embedding Space)

- **داده‌های جفتی (Data Pairs):** مدل روی دیتاست‌های بزرگ آموزش می‌بیند که شامل جفت‌های تصویر-متن هستند. در SigLIP، این جفت‌ها مربوط به تصاویر زبان اشاره و توضیحات متنی (یا تفاسیر) آن‌هاست.
 - **هدف آموزش (Training Objective):** در طول آموزش، هر دو انکودر تصویر و انکودر متن هم‌زمان بهینه می‌شوند تا تضمین کنند تعبیه‌های (بردارهای) مربوط به یک جفت تصویر-متن واقعی، هم‌تراز و به هم نزدیک باشند. به بیان دیگر، جفت‌های تصویر-متن مشابه باید در فضای مشترک، بردارهای نزدیک به هم داشته باشند و جفت‌های نامشابه از هم دور شوند.
-

(۳) تابع ضرر (Loss Function)

- در SigLIP از تابع ضرر مقایسه‌ای (Contrastive Loss) استفاده می‌شود که ماهیتاً مشابه تابع ضرر مدل CLIP است.

- نحوه‌ی کارکرد تابع ضرر مقایسه‌ای:

در مدل‌های CLIP و SigLIP معمولاً از Cross-Entropy متقارن به شکل مقایسه‌ای استفاده می‌شود که دو خواسته‌ی اصلی را محقق می‌کند:

- جفت‌های مثبت (Positive Pairs): تعبیه‌های تصویر و متنی که به یکدیگر مرتبط هستند، باید در فضای تعبیه‌شده به هم نزدیک شوند. (کاهش فاصله‌ی کسینوسی یا افزایش شباهت کسینوسی)
- جفت‌های منفی (Negative Pairs): تعبیه‌های تصویر و متنی که به یکدیگر مرتبط نیستند، باید در این فضا از هم دور شوند. (افزایش فاصله‌ی کسینوسی یا کاهش شباهت کسینوسی)
- روش محاسبه (مثال ریاضی):

عموماً از توابعی مانند NT-Xent (Normalized Temperature-scaled Triplet Loss) یا Cross-Entropy استفاده می‌شود که شباهت هر جفت تصویر-متن را با تمام جفت‌های دیگر در یک بچ (Batch) مقایسه می‌کند. شکل کلی آن به صورت زیر است:

$$\mathcal{L} = -\log \frac{\exp(\text{sim}(I, T)/\tau)}{\sum_j \exp(\text{sim}(I, T_j)/\tau) + \sum_k \exp(\text{sim}(I_k, T)/\tau)}$$

که در آن:

- $\text{sim}(I, T)$ بیانگر شباهت کسینوسی بین بردار تصویر I و بردار متن T است.
- τ پارامتری برای تنظیم مقیاس (دما) است؛ هرچه τ کوچک‌تر باشد، تابع ضرر تمایل دارد تفاوت شباهت‌ها را واضح‌تر (شارپ‌تر) کند.
- در مخرج کسر، مجموع شباهت‌های همه‌ی تصویر-متن‌های دیگر در یک بچ قرار گرفته است. این کار باعث می‌شود مدل، فاصله‌ی بین جفت‌های مثبت را کم و فاصله‌ی بین جفت‌های منفی را زیاد کند.

به کمک یادگیری مقایسه‌ای، مدل می‌آموزد که شباهت بین جفت‌های واقعی (مثبت) را حداکثر و بین جفت‌های نامربوط (منفی) را حداقل کند؛ در نتیجه، یک فضای مشترک به خوبی هم‌راستا ساخته می‌شود.

سؤال دوم

«در فرایند آموزش **SigLIP**، ایده‌ای به کار گرفته شده است که باعث می‌شود آموزش سریع‌تری نسبت به **CLIP** داشته باشد. این ایده چیست و چه مزیتی دارد؟»

۱. جایگزینی تابع ضرر مبتنی بر **Softmax** با تابع ضرر سیگموئید جفتی (**Sigmoid**)

مدل **CLIP** Contrastive Language–Image Pre Training مبتنی بر **Softmax** استفاده می‌کند که نیازمند ساختن و نگهداری یک ماتریس ($N \times N$) از شباهت‌ها برای تمام تصاویر و متون در یک بچ (Batch) است. با بزرگتر شدن اندازه‌ی بچ، این رویکرد از نظر حافظه و ارتباط بین دستگاه‌ها بسیار پرهزینه می‌شود.

در مقابل، **SigLIP** Sigmoid Language–Image Pretraining از تابع ضرر سیگموئید جفتی بهره می‌گیرد و برای هر جفت تصویر-متن، به‌طور مستقل یک مسئله‌ی دودویی (مثبت یا منفی) حل می‌کند. این کار نیاز به نرمال‌سازی جهانی (Global Normalization) بر کل بچ را حذف کرده و ماتریس ($N \times N$) را کنار می‌گذارد.

۲. تابع ضرر سیگموئید جفتی (**Pairwise Sigmoid**)

در این روش، مدل تلاش می‌کند برای هر جفت تصویر (I) و متن (T)، تشخیص دهد آیا این جفت «مطابقت صحیح» دارد یا «نامرتبط» ($\text{label} = 1$) (نامرتبط). بنابراین:

{مثلاً بر اساس شباهت کسینوسی یا ضرب نقطه‌ای}

1. تابع سیگموید:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

که خروجی آن در بازه‌ی $(0, 1)$ قرار دارد و نشان‌دهنده‌ی احتمال «مثبت بودن» جفت است.

2. تابع ضرر :**(Binary Cross-Entropy)**

$$\text{loss} = \text{BCE}(\sigma(\text{sim}(I, T)), \text{label})$$

به طور مسروخ، تابع ضرر سیگموید جفته به صورت زیر نوشته می‌شود:

$$\text{loss} = - \left[y \cdot \log(\sigma(\text{sim}(I, T))) + (1 - y) \cdot \log(1 - \sigma(\text{sim}(I, T))) \right]$$

که در آن:

○ همان **label** یا برچسب جفت (0 یا 1) است.

○ $\sigma(\cdot)$ تابع سیگموئید است.

۳. مزایای استفاده از تابع ضرر سیگموئید در SigLIP

1. حذف نرمال‌سازی جهانی (Softmax) بر کل بج

در CLIP، برای هر تصویر باید احتمال تعلق به هر متن موجود در بج محاسبه شود (و بالعکس)، که نیازمند ماتریس $(N \times N)$ شباهت‌هاست.

○ در SigLIP، هر جفت مستقل بررسی می‌شود و نیازی به ماتریس بزرگ برای نرمال‌سازی نیست.

۲. کاهش شدید مصرف حافظه

با کنار گذاشتن ماتریس ($N \times N$), حافظه‌ی مورد نیاز برای پردازش شباهت‌های تصویر-متن بسیار کمتر می‌شود. این موضوع خصوصاً در اندازه‌های بچ بزرگ (۳۲K، ۶۴K، ۱۶K، و بیشتر) اهمیت حیاتی دارد.

۳. کاهش عملیات مخابراتی (Communication Overhead)

- در رویکرد Softmax، همه‌ی تعبیه‌های تصویر و متن باید بین دستگاه‌های محاسباتی مبادله شوند تا آن ماتریس کامل به دست آید.

- در روش سیگموئید جفتی، می‌توان به جای همه-برای-همه (All-Gather)، فقط به صورت بخشی (Chunked) یا تنها با دستگاه‌های همسایه تعبیه‌ها را ردوبدل کرد و جفت‌های منفی کافی را برای محاسبه‌ی ضرر به دست آورد.

۴. پیاده‌سازی آسان‌تر و پایدارتر

- محاسبه‌ی $\log \sum \exp$ در Softmax نیازمند تثبیت (Stabilization) عددی است (مثلاً کم کردن ماکزیمم لاجیت‌ها).
- در سیگموئید، هر جفت تنها یک مقدار لجستیک تولید می‌کند که به شکل مستقیم در فرمول قرار می‌گیرد و پیاده‌سازی آن ساده‌تر و پایدارتر است.

۴. نتایج عملی و تاثیر بر سرعت آموزش

• افزایش مقیاس‌پذیری:

با حذف نیاز به ماتریس ($N \times N$)، حال می‌توان اندازه‌ی بچ را بسیار بزرگ کرد (حتی تا صدها هزار یا یک میلیون) بی‌آنکه حافظه‌ی دستگاه‌ها زود تمام شود.

• بهبود سرعت و Throughput:

از آنجا که هم هزینه‌ی ارتباطی کاهش می‌یابد و هم نیازی به چندین پاس عددی نیست، SigLIP می‌تواند در مدت زمان کمتر به همان دقیقت یا حتی دقیقت بالاتر برسد.

• مدل‌های بسیار بزرگ روی سخت‌افزار محدود:

در مقاله اشاره شده که می‌توان روی تعداد محدودی TPU/GPU، اندازه‌ی بچ ۳۲K را بدون دردرس اجرا کرد. در CLIP، چنین اندازه‌ای معمولاً به خوش‌های بزرگ‌تری نیاز دارد.

۵. نتیجه‌گیری نهایی

ایده‌ی کلیدی در SigLIP این است که بهجای اتکا به تابعی که برای هر تصویر باید در برابر تمام متون (و بالعکس) نرمال‌سازی شود (همان Softmax در CLIP)، هر جفت تصویر-متن را مستقل بررسی می‌کند و از تابع سیگموید جفتی برای تشخیص مثبت یا منفی بودن جفت استفاده می‌کند. این کار:

- مصرف حافظه را بهشدت کاهش می‌دهد.
- ارتباط بین دستگاه‌ها را کم می‌کند.
- فرآیند را برای اندازه‌های بچ بسیار بزرگ مقیاس‌پذیر می‌کند.
- پیاده‌سازی و محاسبات را آسان‌تر و سریع‌تر می‌سازد.

فرمول نهایی تابع ضرر سیگموید جفتی در قالب Binary Cross-Entropy :

$$\text{loss}_{\text{sigmoid}}(I, T) = - \left[y \log(\sigma(\text{sim}(I, T))) + (1 - y) \log(1 - \sigma(\text{sim}(I, T))) \right].$$

در اینجا:

- I و T نماینده‌ی تعبیه‌های تصویر و متن هستند.
 - $\text{sim}(I, T)$ یک تابع شباهت (مثلاً ضرب نقطه‌ای یا کسینوسی).
 - y همان تابع سیگموید است.
 - $y \in \{0, 1\}$.
- برچسب مثبت (۱) یا منفی (۰) بودن جفت است.

پرسش ۱

پس از آنکه انکودر تصویر و دیکودر متن در داده‌های تک‌وجهی (uni-modal) جداگانه آموزش دیدند، در مرحله بعد این دو مدل با هم ترکیب یا در معماری کلان مدل قرار می‌گیرند و در دو مرحله دیگر دوباره آموزش (pre-train) می‌شوند. این دو مرحله آموزش چگونه انجام می‌شود و هر کدام چه ضرورتی دارند؟

مروری کلی بر مراحل (Overview)

1. آموزش تک‌وجهی - (Unimodal Pre Training)

- خلاصه: مدل بینایی (Vision Encoder) و مدل زبانی (Language Model) هرکدام جداگانه روی داده‌های وسیع مربوط به حوزه خودشان آموزش می‌بینند. مثلًا SigLIP فقط با جفت‌های تصویر-متن (برای یادگیری بازنمایی تصویری) به شکل کنتراستیو (contrastive) تمرین می‌شود و Gemma روی دیتاست‌های متنی بزرگ، مدل زبان خودبازگشته را یاد می‌گیرد.
- هدف: اطمینان از آنکه هر کدام از این بخش‌ها (تصویری یا زبانی) به تنایی قدرتمند شوند و در کار تک‌وجهی خود، عملکردی قوی ارائه دهند. در نتیجه، در مرحله بعدی هنگامی که این دو کنار هم قرار می‌گیرند، پایه اولیه‌شان خوب است و نیاز نیست همه‌چیز از صفر یاد گرفته شود.

2. پیشآموزش چندوجهی - (Multimodal Pre Training)

- خلاصه: ترکیب SigLIP (به عنوان انکودر تصویر) و Gemma-2B (به عنوان دیکودر زبانی) در یک مدل واحد. از این به بعد، مدل کلان با تسكیه‌های چندوجهی (متتنوع) آموزش می‌بیند. در این مرحله رزولوشن ورودی معمولاً 224×224 پیکسل است.
- جزئیات مهم:

■ بدون فریزکردن کامل بخش‌ها: برخلاف روالی که گاهی انکودر تصویر یا مدل زبان فریز می‌شود تا خدشهای در پارامترهای پیشآموزش دیده وارد نشود، در PaliGemma کل پارامترها (هم تصویری و هم زبانی) قابل بهروزرسانی هستند. این کار باعث می‌شود مدل بتواند یاد بگیرد که تصویر و متن را به شکلی عمیق‌تر تلفیق کند.

■ آغاز ملایم (Warm-up) برای نرخ یادگیری انکودر تصویر:

- اگر از همان ابتدا نرخ یادگیری بزرگی برای بخش تصویری بگذاریم، ممکن است «گرادیان‌های نامربوط» از سمت زبان، کیفیت انکودر تصویر را شدیداً خراب کنند. برای جلوگیری، ابتدا نرخ یادگیری انکودر تصویر خیلی کم تنظیم می‌شود و به تدریج زیادتر می‌گردد.
 - در نتیجه، سیگنال‌های ناهمخوانی در مراحل ابتدایی کمتر به انکودر آسیب می‌زنند و به تدریج مدل دو وجهی، همگرایی بهتری پیدا می‌کند.
 - استفاده از چندین تسك مختلف؛ در دیتاست‌های چندوجهی، علاوه بر کپشن‌نویسی (captioning) و پرسش‌پاسخ تصویری (VQA)، تسك‌هایی مثل تشخیص اشیا، سگمنت‌کردن بخش‌های تصویر، تشخیص متن در عکس (OCR) و غیره نیز گنجانده می‌شوند. این تنوع به مدل کمک می‌کند مهارت‌های گسترده‌ای از «درک تصویر و زبان» را به دست آورد.
 - طراحی معماری:
 - ابتدا خروجی انکودر تصویر (токن‌های تصویری) از یک لایه خطی (linear) سپس، توكن‌های تصویر + BOS + توكن‌های پیشوند (prefix) + SEP) عبور می‌کند تا از لحاظ ابعاد با توكن‌های زبانی (واژگان projection یکسان شود.
 - توكن‌های پسوند suffix به مدل زبان (به صورت یک دنباله) فرستاده می‌شود.
 - قانونِ توجه (attention) به صورت prefix-LM طراحی شده است؛ یعنی توكن‌های تصویر و توكن‌های prefix می‌توانند هم‌دیگر را ببینند (دوسو) ولی توكن‌های suffix فقط به گذشته خودشان دسترسی دارند (خودبازگشتن). این الگو برای مولد بودن مدل ضروری است (یعنی مدل بتواند عبارت نهایی را پیش‌بینی کند).
 - هدف: مدل حالا می‌تواند تصویر و متن را تواند پردازش کند و روی مجموعه‌ای عظیم از وظایف چندوجهی تمرین ببیند؛ بنابراین مهارت همبستگی بین «بردارهای بصری» و «تعابیر زبانی» شکل می‌گیرد. در پایان Stage1، مدل هم می‌تواند صفر تا صد تصویر را بخواند (توسط انکودر SigLIP) و هم می‌تواند پاسخی در قالب زبان (با Gemma) تولید کند.
- .3 (افزایش رزولوشن - Resolution Increase Stage2

- خلاصه: همان مدل آموزش دیده Stage1، حالا با رزولوشن تصویری بالاتر (معمولاً 448×448 یا 896×896) برای مدت محدودی ادامه داده می‌شود.
 - جزئیات مهم:
 - تsek‌های رزولوشن حساس: در این مرحله بیشتر داده‌های آموزشی مربوط به خواندن متن ریز (OCR)، نمودارها و دیاگرام‌ها، همچنین وظایف مربوط به اجسام کوچک یا تشخیص/سگمنت‌کردن اشیای ظریف در اولویت بالاتری قرار می‌گیرند. چون این وظایف واقعاً به پیکسل‌های بیشتری احتیاج دارند و در 224×224 ممکن است اطلاعات کافی در دسترس نباشد.
 - تمرین نسبتاً کوتاه: برخلاف Stage1 که ممکن است چند صد میلیون یا حتی میلیارد نمونه را ببیند، این مرحله فقط ده‌ها میلیون نمونه جدید می‌بیند (اما گران‌تر هستند) چون رزولوشن بالاتری دارند). ایده این است که مدل «دانش چندوجهی» کلی خود را در Stage1 فراگرفته و حالا فقط «یاد می‌گیرد» چطور با جزئیات تصویری بیشتر برخورد کند.
 - فایده افزایش توکن تصویر: وقتی تصویر بزرگ‌تر می‌شود، به تبع تعداد توکن‌های تصویری (خروجی انکودر ViT) نیز بیشتر می‌شود. این یعنی مدل ظرفیت بیشتری دارد تا اطلاعاتِ موضعی دقیق‌تری را در کد توکن‌ها بگنجاند. نمود این قضیه در وظایف OCR، سگمنت‌سازی و غیره دیده می‌شود.
 - هدف: بهترشدن مدل در وظایف ریزدانه (fine-grained). در عمل، مقاله نشان می‌دهد که عملکرد مدل روی خواندن متن‌های کوچک، دیزاین‌های آن، مسائل چالش‌برانگیز در VQA (مثلاً اطلاعات دقیق مثل اعداد روی چارت یا جزئیات شیء) با عبور از Stage2 ارتقا می‌یابد.
4. (انتقال نهایی - Transfer Stage3)

- خلاصه: نتیجه کل مراحل (انکودر + دیکودر) حالا به صورت یک مدل چندوجهی پایه در دسترس است. اما هنوز ممکن است نیاز باشد آن را برای یک تsek یا کاربرد خاص «fine-tune» کنیم.
- چرایی نیاز به Fine-tuning:
 - گاهی می‌خواهیم تsek‌های ویژه‌ای را بهتر حل کنیم؛ مثلاً Video QA، یا مثلاً دیتابست یک رشته خاص مثل تصاویر پزشکی. یا اصلاً بخواهیم مدل را اینستراکشن-تیون (instruction tuning) کنیم تا بتواند به سبک گفت‌وگویی عمل کند.

■ بنا بر نتایج مقاله، همین مدل پایه را می‌توان در مدت کوتاهی روی دیتابست تخصصی‌ای (مثلًاً COCO Caption، InfographicQA، WidgetCap و ...) تنظیم کرد و عملکردی بالا به دست آورد.

پرسش ۲

در معماری Pali Gemma، دنبالهٔ ورودی به مدل زبان (Gemma) به دو بخش اصلی «توكن‌های پیشوند (prefix)» و «توكن‌های پسوند (suffix)» تقسیم می‌شود. این تقسیم‌بندی ریشه در نحوهٔ تعامل «دادهٔ ورودی» با «سیستم تولید متن خودبازگشتی» دارد. در ادامه، بدون آوردن مثال صریح، به صورت مفهومی و فنی نقش و ضرورت هر کدام را توضیح می‌دهیم:

۱. توكن‌های پیشوند (Prefix Tokens)

الف) شناسنامهٔ تسک و شرایط ورودی

توكن‌های پیشوند عموماً برای توضیح کاری است که مدل باید انجام دهد. این کار ممکن است «پرسش‌پاسخ تصویری»، «کپشن‌نویسی»، «تشخیص اشیا»، یا هر وظیفهٔ دیگری باشد. از آنجا که در PaliGemma یک مدل زبان بزرگ (LLM) مسئول تولید متن است، باید بداند چه کاری از او خواسته شده است. پس در بخش پیشوند، کلیدواژه‌ها یا عبارت‌های مربوط به نوع وظیفه، زبان هدف، یا دیگر توضیحات آورده می‌شود.

ب) توجه دوسویه به تصویر و پیشوند

از آنجا که معماری PaliGemma برای مرحلهٔ ورودی از رویکرد «prefix-LM» استفاده می‌کند، توكن‌های پیشوند از ابتدا می‌توانند با همدیگر (و با توكن‌های تصویری) دوسویه تعامل داشته باشند؛ یعنی به یکدیگر و به داده‌های بصری نگاه کنند و لایه‌های توجه (attention) هر کدام را ببینند. به این ترتیب مدل می‌تواند بهتر تصمیم بگیرد که چه جنبه‌هایی از تصویر یا چه سبک پاسخی را باید دنبال کند.

ج) تفکیک تسک‌های مختلف

زمانی که مدل روی دیتاست‌های چندوجهی متنوع آموزش می‌بیند، این توکن‌های پیشوند به‌نوعی «برچسب» یا «مسیریاب» هستند تا به مدل نشان دهنده‌ان چه نوع وظیفه‌ای در حال انجام است. این موضوع جلوی تداخل آموزش در وظایف مختلف را می‌گیرد و از همان ابتدا، جهت‌گیری مدل را مشخص می‌کند.

د) نیاز به عدم پیش‌بینی خودبازگشتی

در روش prefix-LM، فرض بر این است که همه توکن‌های پیشوند و توکن‌های تصویری، داده‌ای هستند که «از قبل» داریم و نباید آن‌ها را به صورت خودبازگشتی پیش‌بینی کنیم. یعنی این توکن‌ها بخشی از «کانتکست» هستند و قرار است تمام‌شان را مدل ببیند تا بر اساس‌شان پاسخ نهایی را بسازد. بنابراین، توکن‌های پیشوند یک نقش زمینه‌ساز یا context-provider ایفا می‌کنند.

۲. توکن‌های پسوند (Suffix Tokens)

الف) تولید مرحله‌به‌مرحله (خودبازگشتی)

مهم‌ترین ویژگی توکن‌های پسوند آن است که مدل زبان باید آن‌ها را «مرحله‌به‌مرحله» بسازد. این تفاوت بنیادین با بخش پیشوند دارد که از قبل مشخص و ثابت بود. در بخش پسوند، مدل فقط می‌تواند به توکن‌هایی که تاکنون تولید کرده نگاه کند (همچنین به کل پیشوند و تصویر)، اما اجازه ندارد آیندهً پاسخ را بداند. این ساختار «خودبازگشتی» هسته‌ای مدل‌های زبانی مولد است.

ب) خروجی اصلی مدل

هر کاری که از مدل خواسته شود (شرح، پاسخ، لیستی از مختصات، متن ترجمه‌شده و ...) در این ناحیه شکل می‌گیرد. مدل از روی محتوای بصری و متن پیشوند، تصمیم‌می‌گیرد هر توکن بعدی چه باشد و آن را می‌نویسد. به این ترتیب، کل «پاسخ» در دنبالهٔ پسوند قرار می‌گیرد تا سرانجام به پایان برسد.

ج) عدم دید دوسویه

برخلاف پیشوند که توکن‌های آن می‌توانند به شکل دوسویه یکدیگر را ببینند، در بخش پسوند مدل اجازه ندارد به توکن‌های «آینده» نگاه کند؛ یعنی یک توکن پسوندی فقط از پیشوند + تصویر + توکن‌های پسوندی‌ای که قبلاً تولید شده‌اند، خبر دارد. این ممنوعیت «نگاه به آینده» بخشی از مکانیسم اصلی زبان‌مدل خودبازگشتی است و تصمیم‌می‌کند تولید متن پله‌پله پیش می‌رود.

د) پایان (EOS) و پَدکردن (PAD)

برای متوقف کردن فرایند تولید، نشانگر پایان (EOS) در انتهای پسوند قرار می‌گیرد. این یعنی مدل می‌گوید «کار تمام شد» و اگر لازم باشد، با توکن‌های PAD دنباله را تا طول ثابتی پر می‌کنند. بنابراین، پسوند بخشی است که مدل زبان در آن آزاد است هر چقدر نیاز دارد محتوای پاسخ را ایجاد کند تا به نتیجه برسد.

۳. چرایی تمایز پیشوند و پسوند در PaliGemma

الف) سازگاری با رویکردهای چندوجهی

مدل ابتدا توکن‌های تصویر و پیشوند را به‌طور آزاد می‌بیند، پس می‌تواند تمام اطلاعات محیط، دستوری، و تصویری را یکجا درک کند. آنگاه در قالب پسوند، به صورت پله‌پله واکنش نشان می‌دهد و متن خروجی را تولید می‌کند. این ساختار هم با ماهیت «پردازش تصویر» سازگار است (که پیشاپیش داده شده) و هم با ماهیت «زبان خودبازگشته» در بخش تولید خروجی.

ب) پشتیبانی از طیف متنوع کارکردها

زمانی که لازم باشد مدل زبان در تسک‌های مختلف (کپشن، پرسش‌پاسخ، تشخیص اشیا و ...) شرکت کند، به‌سادگی در بخش پیشوند دستور کار را تغییر می‌دهیم؛ اما بخش پسوند همچنان جایگاه تولید متن است که مطابق آن دستور کار پیش می‌رود.

ج) عدم تداخل دستورالعمل و پاسخ

اگر دستورالعمل (سؤال یا هر قالبی) و پاسخ در هم آمیخته می‌بودند، مدل باید تشخیص می‌داد چه قسمتی از ورودی «باید پیش‌بینی شود» و چه قسمتی «صرفًا ورودی ثابت» است. اما با جدا کردن پیشوند و پسوند، این معضل حل می‌شود: پیشوند همیشه ورودی است و پیش‌بینی‌نشدنی، پسوند همیشه خروجی است و باید مرحله‌به‌مرحله پیش‌بینی شود.

۴. فواید عملی

۱. همگرایی پایدار

از آنجا که ساختار ماسک‌گذاری در بخش پاسخ خود بازگشتی و در بخش پرسش/تصویر دو سوت، مدل به شکل نسبتاً ساده و قابل کنترل آموزش می‌بیند. بخشی از خطاهای ناسازگاری‌های احتمالی که در معماری‌های پیچیده پیش می‌آمد، کم می‌شود.

۲. استفاده از دانش پیشوند برای رمزگشایی بهتر

وقتی سؤالی یا دستوری در پیشوند است، همه توکن‌های پسوند می‌توانند به آن دسترسی پیدا کنند. به محض تولید هر توکن در پسوند، از محتوای پیشوند (شامل مفهوم وظیفه، زبان، دستورات خاص و ...) بهره می‌گیرد. این جریان اطلاعات باعث می‌شود پاسخ با نیاز تسلیک هماهنگ باشد.

۳. توسعه‌پذیری و افزودن وظایف جدید

در آینده، اگر وظیفه یا ساختار جدیدی تعریف کنیم، کافی است کلیدواژه یا فرمت جدیدی در پیشوند اضافه کنیم تا مدل بداند کار چیست. خروجی باز هم در پسوند خواهد بود. به این شکل، معماری انعطاف‌پذیری بالایی پیدا می‌کند.

پرسش ۳

یکی از چالش‌های معمول در چنین ساختارهایی این است که مدل زبان (Language Model) روی توکن‌های متنی تنظیم شده و وقتی با داده‌های دیداری مواجه می‌شود ممکن است کیفیتش افت کند. مدل PaliGemma چگونه این مشکل را حل کرده است؟

در بسیاری از رویکردهای چندرسانه‌ای (Multimodal) وقتی یک مدل زبان بزرگ (LLM) را مستقیماً با ورودی بصری درهم می‌آمیزیم، احتمال دارد که «دانش زبان» از قبل آموخته شده تحت تأثیر سیگنال‌های تصویری و گرادیان‌های جدید، تضعیف شود یا حتی بخشی از آن فراموش شود. اصطلاحاً این پدیده را "Catastrophic Forgetting" یا «فراموشی فاجعه‌بار» می‌نامند. در چنین شرایطی، مدل شاید هنوز بتواند اطلاعاتی از تصویر استخراج کند، اما هم‌زمان مهارت‌های زبانی از بین می‌روند و یا کیفیت کلی آن افت می‌کند.

در مدل PaliGemma (و روش‌های مشابه)، چندین راهکار هم‌زمان به کار گرفته می‌شود تا از تخریب یا نزول عملکرد LLM هنگام دریافت داده تصویری جلوگیری شود:

1. استفاده از انکودر تصویری پیشآموزش دیده قدرتمند

- **SigLIP**: این انکودر تصویر از قبل (در Stage0) با یک دیتاست عظیم از جفت‌های تصویر-متن و به روش کنتراستیو (Contrastive Learning) آموزش دیده است.
 - **مزیت اصلی**: وقتی مدل زبان بهجای تصویر خام، بردارهای باکیفیت و معنادار را دریافت می‌کند، نیاز ندارد «از صفر» رمزگشایی پیکسل‌ها را یاد بگیرد. بنابراین، سیگنال بصری ورودی، ساختارمند و به ثبات رسیده است و کمتر باعث تغییر ناگهانی در پارامترهای زبان مدل می‌شود.
-

2. طراحی "Warm-up" ملایم برای نرخ یادگیری بخش تصویری

- در مرحله چندوجهی (Stage1)، اگر از همان ابتدا نرخ یادگیری (LR) برای انکودر تصویر خیلی بزرگ باشد، مدل زبان که هنوز هماهنگ با سیگنال تصویری نیست، ممکن است گرادیان‌های عجیب و غریب به سمت انکودر تصویر بفرستد.
- **راه حل**: ابتدا LR انکودر تصویر را خیلی کم می‌گذارند و رفته‌رفته به مقدار دلخواه می‌رسانند.
- **نتیجه**:

 1. در مراحل اولیه که مدل زبان هنوز «تطابق» کافی با داده‌های تصویری ندارد، تغییرات کمتری روی بخش تصویری اعمال می‌شود (جلوگیری از تخریب).
 2. به محض آنکه مدل زبان و سیگنال چندرسانه‌ای تا حدی منسجم شدند، LR انکودر تصویر بالاتر رفته و می‌تواند در راستای وظایف پیچیده (مانند رابطه‌های فضایی یا جزئیات ظریف) یادگیری را بهبود بخشد.

3. ساده نگهداشت اتصال (Connector) بین انکودر تصویر و مدل زبان

- **لایه خطی (Linear Projection)**:
 - بسیاری از کارها، برای یکپارچه‌سازی ورودی تصویر به مدل زبان، یک شبکه MLP چندلایه یا سازوکارهای پیچیده‌تر به کار می‌گیرند.

- در PaliGemma، از یک لایه خطی ساده برای تبدیل ابعاد خروجی انکودر تصویر به همان ابعاد توکن‌های زبانی استفاده می‌شود.
 - مزیت: سادگی این لایه هم باعث پایداری بیشتر در آموزش می‌شود، هم ریسک «همه‌چیز را بهم ریختن» را کاهش می‌دهد. هرچه لایه میانی حجمی‌تر باشد، احتمال دارد مدل زبان در آن «گیر بیفت» و شکل ناهنجاری از همگرایی یا حتی افت کلی ایجاد شود.
-

4. ساختار ماسک‌گذاری از نوع Prefix-LM

- در معماری‌های «Encoder-Decoder» سنتی، ممکن است برخی توکن‌های ورودی نیز به شکل خودبازگشتنی پیش‌بینی شوند یا بخش بصری و متنی توزیع ماسک پیچیده داشته باشند.
 - در PaliGemma، به روش Prefix-LM عمل می‌شود:
 1. توکن‌های تصویر + توکن‌های پرسش/دستور (prefix) را می‌توان به صورت دوسو خودبازگشتنی مشاهده کرد. یعنی این توکن‌ها همدیگر را می‌بینند و با هم "تعمق" می‌کنند.
 2. اما توکن‌های پاسخ (suffix) به صورت خودبازگشتنی تولید می‌شود: هر توکن بعدی با توجه به توکن‌های قبلی‌اش نوشته می‌شود.
 - چرا کمک می‌کند؟
 1. توکن‌های تصویر از همان آغاز می‌توانند «بدانند» سؤال یا دستور متنی چیست (prefix). پس بردارهای تصویری به شکلی هدفمند، سیگنال لزوم توجه به قسمت‌های خاص تصویر را می‌گیرند.
 2. مدل زبان هم فقط در بخش پاسخ (suffix) هست که باید مرحله‌به‌مرحله خروجی تولید کند؛ بنابراین «مغز زبانی» دچار پردازش عجیب کل ورودی (شامل بخش تصویری) به صورت خودبازگشتنی نمی‌شود. در نتیجه، دانش زبانی کمتر لطمه می‌خورد.
-

5. عدم فریز کامل بخش زبان

- برخلاف برخی روش‌ها که می‌گویند «برای حفاظت از مدل زبان، آن را فریز کنیم و فقط بر بخش تصویری تنظیمات انجام شود»، در PaliGemma این‌طور نیست.

- اما: چون SigLIP قبلاً قدرتمند است و ضمناً warm-up ملایم دارد، گرادیان‌ها به شکلی هدایت می‌شوند که مدل زبان تضعیف نشود.
 - فایده:
 1. LLM می‌تواند از سیگنال تصویری یاد بگیرد که مثلًاً چگونه به تصاویر پیچیده پاسخ زبانی دقیق دهد (مثلًاً کجاها باید شمارش کند، کجاها باید توصیف کند...).
 2. مدل زبان توانایی‌های خود در درک معنایی و زبانی را حفظ می‌کند و حتی غنی‌تر هم می‌شود (چون حالا از تصاویر هم دانش بیشتری کسب می‌کند).
-

6. جلوگیری از «همه‌چیز را یک‌دفعه با هم تمرین دادن» (تدریج در طراحی مراحل)

- رویکرد کاملاً مرحله‌بندی شده است:
 - Stage0 (آموزش تک‌وجهی) → Stage1 (چند ساعته یا چند روزه، ولی تنوع زیاد داده‌های تصویری-متنی) → Stage2 (رزولوشن بالا، ولی مدت کمتر).
 - این خودش جلوی فشار بیش از حد را می‌گیرد که ممکن است مدل زبان با ورودی‌های خام/حجیم‌بصری از ابتدا له شود.
 - در انتها، مدلی داریم که نه تنها افت نمی‌کند بلکه در انواع وظایف بینایی-زبانی خوب عمل می‌کند.
-

زیر بخش چهارم - Transfer Learning

در این بخش ابتدا به تعریف PEFT میپردازیم :

PEFT: Parameter-Efficient Fine-Tuning

(Fine-Tuning) مخفف PEFT است که به یک رویکرد نوآورانه در تنظیم (Parameter-Efficient Fine-Tuning) مدل‌های بزرگ یادگیری عمیق اشاره دارد. این روش به جای بهروزسازی تمام پارامترهای یک مدل بزرگ، تنها زیرمجموعه‌ای کوچک از پارامترها را تغییر می‌دهد. هدف اصلی PEFT کاهش منابع محاسباتی، حافظه مورد نیاز، و هزینه‌ی تنظیم مدل است، بدون کاهش چشمگیر در دقت یا عملکرد.

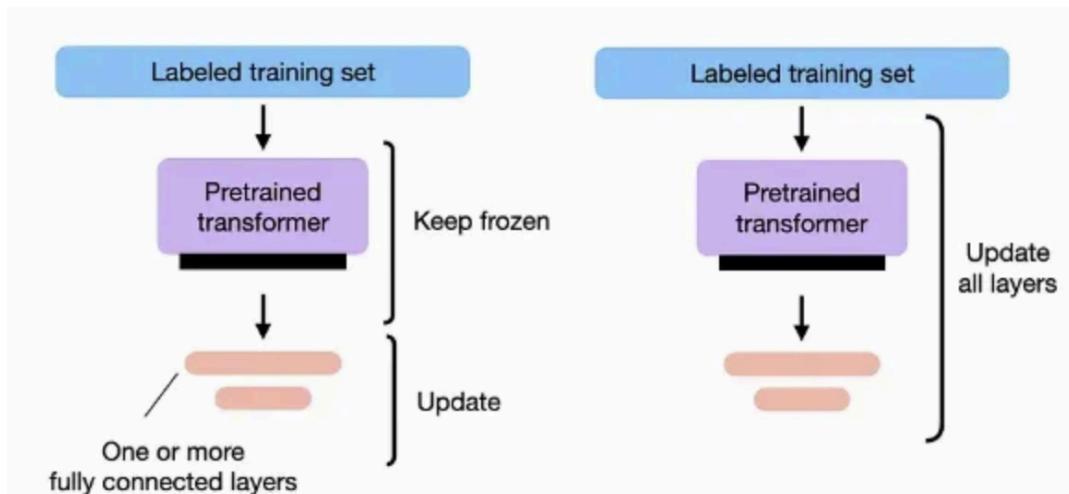
دلایل استفاده از PEFT

1. اندازه‌ی مدل‌های بزرگ:
 - مدل‌های مدرن یادگیری عمیق (مانند ViT، GPT-3، BERT) معمولاً شامل میلیاردها پارامتر هستند.
 - تنظیم این مدل‌ها به طور کامل (Full Fine-Tuning) مستلزم حافظه و محاسبات بسیار زیاد است.
2. هزینه و زمان بالا:
 - تنظیم کامل مدل‌های بزرگ برای هر وظیفه نیازمند منابع سخت‌افزاری قوی و زمان زیادی است که ممکن است برای بسیاری از کاربران عملی نباشد.
3. افزایش قابل استفاده بودن مدل‌ها:

با PEFT، می‌توان مدل‌های از پیش‌آموزش‌دیده را برای کاربردهای خاص با تغییرات جزئی تنظیم کرد، بدون نیاز به بازآموزی کل مدل.

اصول PEFT

Full finetuning vs PEFT



در PEFT، تنها بخش کوچکی از پارامترهای مدل تغییر می‌کنند یا پارامترهای اضافی به مدل اضافه می‌شوند، در حالی که بخش عمده‌ی پارامترها ثابت می‌ماند. این روش معمولاً شامل مراحل زیر است:

1. مدل اصلی را ثابت نگه دارید (Frozen Model):

- پارامترهای اصلی مدل از پیشآموزش دیده، ثابت نگه داشته می‌شوند و تنها بخشی خاص (معمولًاً کوچک) از آن‌ها تغییر می‌کند.
- اضافه کردن لایه‌های اضافی (اختیاری):

- در برخی از روش‌های PEFT، لایه‌ها یا مازولهای کوچک جدیدی (مانند **Adapters**) به مدل اضافه می‌شوند که وظیفه‌ی یادگیری تغییرات خاص وظیفه را بر عهده دارند.
- به روزرسانی انتخابی:

- به جای تغییر کل پارامترها، تنها بخش خاصی از مدل (مثلاً لایه‌های آخر، یا مازولهای خاص) به روزرسانی می‌شوند.

روش‌های معروف PEFT

1. **Adapters**:

- در این روش، لایه‌های کوچکی به بین لایه‌های اصلی مدل اضافه می‌شوند. این لایه‌ها وظیفه دارند تغییرات مربوط به وظیفه‌ی خاص را یاد بگیرند.
- پارامترهای اصلی مدل ثابت باقی می‌مانند و تنها پارامترهای Adapter به روزرسانی می‌شوند.

2. **(LoRA) (Low-Rank Adaptation)**:

- یک روش موثر در PEFT است که فرض می‌کند تغییرات مورد نیاز برای تنظیم مدل را می‌توان به یک فضای کمرتبه (Low-Rank) محدود کرد.
- تنها بخشی از پارامترهای مدل را تغییر می‌دهد، معمولاً با افزودن ماتریس‌های کمرتبه به LORA مدل.

:Prefix-Tuning .3

- در این روش، بردارهای اضافی (Prefixes) به ورودی مدل اضافه می‌شوند که مدل را برای وظیفه‌ی خاص هدایت می‌کنند.

- این بردارها آموزش داده می‌شوند، اما خود مدل ثابت باقی می‌ماند.

:Prompt-Tuning .4

- شبیه به Prefix-Tuning، اما فقط روی تنظیمات ورودی (Prompts) تمرکز دارد. مدل اصلی هیچ تغییری نمی‌کند.

:BitFit .5

- این روش تنها بایاس‌ها (Biases) مدل را تنظیم می‌کند و تمام پارامترهای دیگر را ثابت نگه می‌دارد.

:Diff Pruning .6

- این روش فرض می‌کند که تفاوت (Difference) بین مدل اولیه و مدل تنظیم شده را می‌توان به صورت پراکنده (Sparse) ذخیره کرد.

PEFT مزایای

1. کاهش منابع محاسباتی:

- چون تنها بخش کوچکی از پارامترها به روزرسانی می‌شوند، نیاز به حافظه و محاسبات کاهش می‌یابد.

2. سرعت بالاتر در تنظیم:

- تنظیم مدل‌های بزرگ زمانبر است، اما PEFT این فرآیند را سریع‌تر می‌کند.

3. صرفه‌جویی در ذخیره‌سازی:

- به جای ذخیره‌ی کل مدل تنظیم شده، تنها تغییرات کوچک ذخیره می‌شوند.

4. افزایش انعطاف‌پذیری:

- مدل اصلی می‌تواند برای چندین وظیفه‌ی مختلف، بدون تغییر کلی، استفاده شود.

5. دسترسی بیشتر به مدل‌های بزرگ:

- امکان استفاده از مدل‌های بزرگ را برای کاربران با منابع محدود فراهم می‌کند.

معایب PEFT

1. محدودیت در تنظیمات پیچیده:

- ممکن است برای وظایف پیچیده یا داده‌های کم، عملکرد آن به اندازه‌ی تنظیم کامل نباشد.
- نیاز به طراحی دقیق:

- انتخاب قسمت‌هایی از مدل که باید بهروزرسانی شوند، نیازمند دانش عمیق از مدل و وظیفه است.

3. کارایی کمتر در برخی موارد:

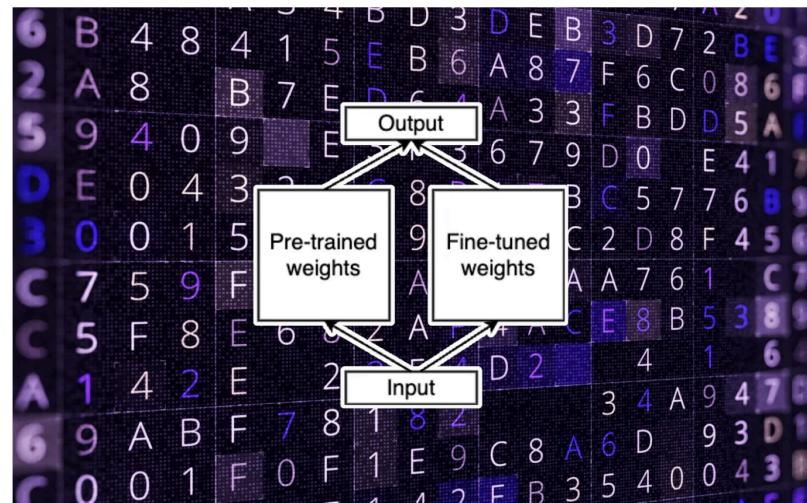
اگر تغییرات گسترده در مدل برای وظیفه‌ی خاص لازم باشد، PEFT ممکن است به اندازه‌ی Full Fine-Tuning مؤثر نباشد.

مثال کاربردی

فرض کنید شما یک مدل بزرگ مانند GPT-3 دارید و می‌خواهید آن را برای تحلیل احساسات (Sentiment) تنظیم کنید. با استفاده از PEFT (Analysis

1. مدل اصلی ثابت می‌ماند و فقط لایه‌های اضافه‌شده یا بایاس‌ها بهروزرسانی می‌شوند.
2. حافظه و محاسبات بسیار کمتری نیاز دارید، زیرا تنها تعداد کمی از پارامترها تغییر می‌کنند.
3. مدل نهایی سبک‌تر و سریع‌تر خواهد بود، و به راحتی می‌توانید آن را برای سایر وظایف نیز استفاده کنید.

LoRA



LoRA approach. Source:[6]

پرسش اول :

در این پاسخ، هر کدام از بخش‌های مطرح شده درباره LoRA، QLoRA، پارامتر رنک (Rank) در LoRA، و دیتاتایپ NF4 را با جزئیات بیشتر تشریح می‌کنیم تا روشن شود در عمل چه اتفاقی می‌افتد و هر کدام چه اثری دارد.

LoRA Low-Rank Adaptation .۱

۱.۱. ایده اصلی در LoRA

- وقتی می خواهیم یک مدل بزرگ زبانی (LLM) یا مدل ویژن-زبان را برای یک تسک تازه **فاینتیون** کنیم، معمولاً نیاز به بهروزرسانی همه پارامترهای مدل داریم که هزینه حافظه و محاسباتی بالایی دارد.
- در روش **LoRA**، تصمیم می گیریم که وزن های اصلی مدل (مثلاً ماتریس های بزرگ لایه توجه) را ثابت نگه داریم و به جای آن در بخش های هدف (مثلاً لایه های q_proj , k_proj در self-attention) این ماتریس هایی کم رتبه اضافه کنیم که فقط همین ماتریس ها طی یادگیری آپدیت شوند.
- این ماتریس های کم رتبه معمولاً از ابعاد $r \times fan_in^*$ و fan_out^* تشکیل می شوند؛ جایی که r همان رنک (Rank) است.

۱.۲. چرا «کم رتبه»؟

- ایده «کم رتبه» (low-rank) این است که تغییر مورد نیاز در وزن های بزرگ مدل، عموماً ساختار یافته و با بعد موثر پایین تر از کل ابعاد آن لایه است. مثلاً اگر لایه ای $65k$ پارامتر داشته باشد، می توان با یک ماتریس کوچک تر (رنک پایین) تغییر مطلوب را پیاده کرد.
- بدین ترتیب، فضایی که مدل می تواند «نسبت به وزن های پایه» حرکت کند محدود تر ولی کافی است تا دانش تازه در تسک جدید را کسب نماید.

۱.۳. جزئیات مزایا

- صرفه جویی در حافظه:
 - در فاین تیون کامل، هر وزن مدل باید گرادیان، مومنتوم و واریانس اوپتیمایزر (مثلاً در آدم) داشته باشد که حجم زیادی ایجاد می کند.
 - در LoRA، به جای همه وزن ها، تنها آدابتور هایی با ابعاد کوچک + گرادیان + مومنتوم + واریانس مربوط به آنها ذخیره می شوند.
- پارامترهای کم و ساختار مازولار:
 - پس از یادگیری، دیگر نیازی نیست مدل را به طور کامل ذخیره کنیم. می توانیم همان وزن های پایه را نگه داریم + آدابتورهای کم حجم را به صورت افزونه (plug-in) داشته باشیم.
- سرعت:
 - چون تعداد پارامترهای در حال بهروزرسانی کوچک است، پس انتشار (backprop) سریع تر انجام می شود و تبادل حافظه کمتری صورت می گیرد.

۱.۴. محدودیت‌ها و نکات طراحی

۱. ظرفیت محدود:

- اگر رنک خیلی کوچک باشد یا مازول‌های نامناسبی برای افزودن LoRA انتخاب شوند، شاید مدل نتواند همهٔ پیچیدگی‌های تسك را بیاموزد (underfitting).
 - به همین دلیل گاهی نیاز است رنک را بزرگ‌تر بگیریم یا مازول‌های بیشتری را LoRA کنیم تا کیفیت بالا بماند.
۲. طراحی نیازمند تجربه:
- بسته به مدل، گاهی فقط در مازول‌های q_proj و v_proj LoRA قرار می‌دهند، گاهی همهٔ پروژکشن‌های توجه یا حتی فیدفوروارد را پوشش می‌دهند. این تصمیم مستقیماً روی عملکرد و حجم نهایی تاثیر دارد.

QLoRA Quantized LoRA .۲

۱. ایدهٔ اصلی در QLoRA

- در QLoRA، همان آداتورهای کمرتبه LoRA را داریم، اما با این تفاوت که وزن‌های پایهٔ مدل را به فرم ۴بیتی (Quantized) نگهداری می‌کنیم.
- بنابراین، اندازهٔ حافظهٔ اشغال‌شده توسط مدل اصلی بسیار کم می‌شود؛ ولی آداتورهای LoRA همچنان با دقت بالاتر (مثلًا FP16 یا BF16) ذخیره و آپدیت می‌شوند.

۲. چرا کوانتیزهٔ ۴بیتی در پایهٔ مدل؟

1. کاهش حافظه و هزینه سخت‌افزار:
 - مثلًا اگر یک مدل ۱۰ میلیارد پارامتری را FP16 ذخیره کنید، نیاز به ۲۰ گیگابایت برای فقط وزن‌ها دارید. اما در QLoRA می‌توان همین را به یک‌چهارم (یا حتی کمتر) تقلیل داد.
2. ترکیب با LoRA:
 - هرچند وزن‌ها کوانتیزه شده‌اند، اما یادگیری اصلی روی آداتورهای کمرتبه است که دقت بالایی دارند و اجازه می‌دهند کیفیت حفظ شود.

۲.۳. محدودیت‌ها و ظرایف

• پیچیدگی کوانتیزاسیون:

اگر اجرای ۴ بیتی به درستی صورت نگیرد، ممکن است مدل پایه دچار افت دقت قابل توجه شود؛ به ویژه اگر نحوه نرمال‌سازی یا نقشه‌برداری عددی اشتباہ باشد.

• باقی ماندن محدودیت‌های LoRA:

در نهایت QLoRA هم دقیقاً مانند LoRA به انتخاب رنک، ماثول‌ها و ... بستگی دارد. صرفاً مزیت بیشتری در کاهش حافظه فراهم می‌کند.

LoRA Rank .۳

۳.۱. تعریف Rank

- هنگامی که یک لایه مثلثی W دارد و ما می‌خواهیم با LoRA تغییرش دهیم، دو ماتریس کمرتبه اضافه می‌کنیم $\mathbf{B} \in \mathbb{R}^{r \times d}$ و $\mathbf{A} \in \mathbb{R}^{d \times r}$ (به صورت خلاصه). پارامتر r همان Rank است.
- در عمل، وزنی به شکل $\mathbf{W} + \alpha \cdot \mathbf{A}\mathbf{B}$ شکل می‌گیرد (که α معمولاً یک ضریب مقیاس کوچک است).

۳.۲. رنک بالا در مقابل رنک پایین

1. رنک بالا:
 - ماتریس کمرتبه AB فضای بزرگ‌تری را می‌پوشاند؛ بنابراین می‌تواند پیچیدگی‌های گوناگون تر تسک جدید را یاد بگیرد.
 - اما پارامترهای اضافی بیشتری دارد (تقریباً $r^2 \times d \times 2$) و حافظه و زمان بیشتری برای آپدیت لازم دارد.
2. رنک پایین:
 - سبک‌تر است. سریع‌تر تمرين می‌شود.
 - اما اگر تسک جدید خیلی متفاوت یا پیچیده باشد، مدل با رنک پایین امکان دارد نتواند نمایشی دقیق از آن بسازد (افت دقت).

۳.۳. یافتن «رنک بهینه»

- یک اصل کلی این است که از عدهای متوسط (مانند ۸، ۱۶ یا ۳۲) استفاده می‌کنند و تجربه نشان داده برای بسیاری از تسکهای زبانی کافی است.
 - گاهی نیاز به تجربه و ارزیابی روی مجموعه‌داده اعتبارسنجی است تا رنک مناسب انتخاب شود؛ توازن بین کیفیت و هزینه.
-

۴. دیتا تایپ؛ NF4

۴.۱. تفاوت با int4 ساده

- در روش int4 ممکن است وزن‌ها به‌شکل خطی بین \min و \max نگاشت شوند؛ یا همه آن‌ها با یک مقیاس ثابت ذخیره شوند.
- در NF4 Normalized Float 4-bit، معمولاً رویکردی هوشمندتر به‌کار می‌رود:
 - برای هر بلاک کانال یا هر بلوک چند ده پارامتری، یک ضریب مقیاس (scale factor) محاسبه می‌شود تا آن بخش نرم‌الگردد.
 - سپس اعداد در محدوده ۴‌بیتی شناور ثبت می‌شوند.

۴.۲. مزیت NF4

- دقت بهتر در توزیع وزن‌ها:
 - چون دامنه عددی هر بلوک به‌شکل جداگانه تنظیم می‌شود، خطای کوانتیزه‌کردن برای محدوده‌های بزرگ یا کوچک کاهش می‌یابد.
- حفظ ویژگی‌های مهم وزن‌ها:
 - مثلاً اگر برخی وزن‌ها مثبت و برخی منفی باشند یا بردارهای مختلف دامنه‌های متفاوت داشته باشند، NF4 بهتر از int4 ساده عمل می‌کند.
- کاهش پهنای باند:
 - در حین پیش‌پردازش (forward pass)، خواندن وزن‌ها از حافظه سبک‌تر می‌شود و منابع محاسباتی آزادتر می‌گردد.

۴.۳. کاربرد در QLoRA

- از NF4 برای نگهداری وزن‌های فریز شده‌ی مدل بهره می‌گیرد؛ یعنی مدل پایه را با NF4 ذخیره می‌کند.

- بعد در هنگام آموزش و استنتاج، این وزن‌ها به فرمت بالاتر (مثلًا FP16 داخلی) تبدیل می‌شوند تا محاسبات انجام گردد، ولی ذخیره‌سازی و انتقال در حافظه با فرمت ۴بیتی است.

ماژول‌های مورد هدف در سوال (q_proj, k_proj, v_proj, o_proj, gate_proj.) : (down_proj و up_proj

ماژول‌های مورد هدف در سوال (down_proj, q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj) اجزای اصلی معماری مدل‌های مبتنی بر ترنسفورمر (مانند GPT، BERT یا LLaMA) هستند. این ماژول‌ها مسئول محاسبات و تغییراتی هستند که در مکانیزم توجه و لایه‌های پیش‌برنده (Feed-Forward) انجام می‌شود. در ادامه، هر یک را به‌طور کامل توضیح می‌دهیم:

۱. ماژول‌های مکانیزم توجه (Self-Attention Mechanism)

مکانیزم توجه بخشی از مدل است که روابط بین توکن‌های یک جمله (یا توکن‌های ورودی) را محاسبه می‌کند. این ماژول‌ها شامل موارد زیر هستند:

q (پروژکشن کوئری یا پرسش)

- **هدف:** ورودی‌های مدل (بردارهای تعبیه‌شده یا حالت‌های مخفی) را به فضای کوئری (پرسش) تبدیل می‌کند.
- **نقش:** مشخص می‌کند که هر توکن می‌خواهد به چه چیزی در توکن‌های دیگر توجه کند.
- **فرمول ریاضی:** $Q = XWq$ که در آن X ورودی و W ماتریس وزن کوئری است.

(Key) k_proj (پروژکشن کلید یا k)

- **هدف:** ورودی‌ها را به فضای کلید تبدیل می‌کند.

- نقش: نشان می‌دهد که هر توکن چه اطلاعاتی برای دیگر توکن‌ها فراهم می‌کند تا به آن توجه کنند.
- فرمول ریاضی: $K = XW_k K = XW_k$ ماتریس وزن کلید است.

(پروجکشن مقدار یا v_{proj})

- هدف: ورودی‌ها را به فضای مقدار تبدیل می‌کند.
- نقش: اطلاعات واقعی را که پس از محاسبات توجه منتقل می‌شوند، شامل می‌شود.
- فرمول ریاضی: $V = XWvV = XW_v$ ماتریس وزن مقدار است.

(پروجکشن خروجی یا o_{proj})

- هدف: مقادیر وزنی (که نتیجه مکانیزم توجه هستند) را ترکیب کرده و به فضای ابعاد اولیه بازمی‌گرداند.
- نقش: اطمینان می‌دهد که خروجی مکانیزم توجه همان ابعاد ورودی را دارد.
- فرمول ریاضی: $O = AW_0 = AW_0$ ماتریس وزن خروجی است.

2. مازول‌های لایه‌های پیش‌برنده (Feed-Forward Layers)

بعد از مکانیزم توجه، لایه‌های پیش‌برنده (FFN) برای پردازش و تبدیل خروجی مکانیزم توجه به کار می‌روند. این مازول‌ها شامل موارد زیر هستند:

(پروجکشن دروازه‌ای یا $gate_{proj}$)

- هدف: کنترل جریان اطلاعات در لایه پیش‌برنده.
- نقش: مشابه مکانیزم‌های دروازه‌ای در شبکه‌های دیگر (مانند LSTM)، این مازول قبل از پردازش اطلاعات در لایه، آن را تغییر شکل می‌دهد.

(پروجکشن صعودی یا up_{proj})

- هدف: افزایش ابعاد داده به طور موقت در طول فرآیند پردازش.
- نقش: فضای ویژگی را گسترش می‌دهد تا امکان انجام محاسبات پیچیده‌تر فراهم شود.

- فرمول ریاضی:

داده‌ها از بعد اصلی dd به بعد بالاتر $\text{upd}_{\backslash \text{text}\{up\}}$ منتقل می‌شوند.

(Downward Projection down_proj

- هدف: کاهش ابعاد داده به مقدار اولیه پس از پروژکشن صعودی.

- نقش: اطمینان می‌دهد که خروجی نهایی لایه پیش‌برنده همان ابعاد ورودی لایه ترانسفورمر را دارد.

- فرمول ریاضی:

داده‌ها از $\text{up}_{\backslash \text{text}\{up\}}$ به dd بازمی‌گردند.

چگونگی ارتباط این مازول‌ها

1. در مکانیزم توجه:

- v_{proj} و q_{proj} ، k_{proj} برای محاسبه کوئری‌ها، کلیدها و مقادیر استفاده می‌شوند.

- این مقادیر برای محاسبه امتیاز توجه و تولید خروجی وزنی استفاده می‌شوند.

- o_{proj} نتیجه نهایی را به فضای اصلی بازمی‌گرداند.

2. در لایه‌های پیش‌برنده:

- اطلاعات را پیش از پردازش مدیریت می‌کند.

- ابعاد داده را افزایش داده و $down_{proj}$ آن را به مقدار اولیه بازمی‌گرداند.

چرا این مازول‌ها هدف LoRA قرار می‌گیرند؟

- این مازول‌ها شامل بزرگترین ماتریس‌های وزنی در مدل هستند و تعداد زیادی پارامتر دارند.

- تنظیم این مازول‌ها تأثیر قابل توجهی بر عملکرد مدل دارد.

- روش LoRA فقط این ماتریس‌ها را تغییر داده و سایر پارامترهای مدل را ثابت نگه می‌دارد، که باعث

کاهش نیاز به حافظه و زمان محاسباتی می‌شود.

با تمرکز روی این مازول‌ها، LoRA به طور کارآمد مدل را با استفاده از تعداد کمی از پارامترهای قابل آموزش تنظیم می‌کند.

فرمول کلی تعداد پارامترها در هر مازول بدین صورت محاسبه خواهند شد :

برای هر مازول:

$$\{\text{تعداد پارامترها}\} = \{\text{ابعاد ورودی}\} * \{\text{ابعاد خروجی}\}$$

برای محاسبه تعداد پارامترهای مصرفی هر یک از مازول‌های ذکر شده (`q_proj`, `k_proj`, `v_proj`, `o_proj`, `down_proj` و `gate_proj`, `up_proj`), باید به ابعاد ورودی و خروجی هر مازول توجه کنیم. در اینجا، محاسبه تعداد پارامترها بر اساس معماری مدل‌های ترنسفورمری با ابعاد مشخص انجام می‌شود.

فرمول کلی تعداد پارامترها در هر مژول

برای هر مژول:

$$\text{ابعاد خروجی} \times \text{ابعاد ورودی} = \text{تعداد پارامترها}$$

اگر LoRA برای مژول‌ها اعمال شود، ماتریس‌های A و B اضافه می‌شوند:

$$\text{تعداد پارامترهای LoRA} = 2 \times (\text{رتبه} \times \text{ابعاد اصلی})$$

مژول‌های مکانیزم توجه

q_proj, k_proj, v_proj .1

- این مژول‌ها ابعاد ورودی را از فضای d (ابعاد مخفی) به همان فضای مخفی کاهش یا افزایش می‌دهند.
- تعداد پارامترها (بدون LoRA):
$$\text{تعداد پارامترها} = d \times d$$

o_proj .2

- این مژول خروجی مکانیزم توجه را از فضای d به فضای مخفی برمی‌گرداند.
- تعداد پارامترها (بدون LoRA):
$$\text{تعداد پارامترها} = d \times d$$

مژول‌های لایه‌های پیش‌برنده

gate_proj .3

- این مژول معمولاً یک لایه پیش‌برنده است که اطلاعات را از فضای مخفی عبور می‌دهد.
- تعداد پارامترها (بدون LoRA):
$$\text{تعداد پارامترها} = d \times d_{\text{ffn}}$$

up_proj .4

- این مژول ابعاد را از d به d_{ffn} افزایش می‌دهد.
- تعداد پارامترها (بدون LoRA):
$$\text{تعداد پارامترها} = d \times d_{\text{ffn}}$$

down_proj .5

- این مژول ابعاد را از d_{ffn} به d کاهش می‌دهد.
- تعداد پارامترها (بدون LoRA):
$$\text{تعداد پارامترها} = d_{\text{ffn}} \times d$$

محاسبات برای LoRA

در LoRA، هر ماتریس دو ماتریس A و B اضافه می‌کند:

$$\text{LoRA} = 2 \times (d \times r)$$

مثال با مقادیر مشخص

فرض کنیم:

- $d = 4096$ (ابعاد فضای مخفی).

$$d_{\text{ffn}} = 4 \times d = 16384 \quad \bullet$$

- رتبه LoRA.

محاسبات برای هر ماتریس

:q_proj, k_proj, v_proj, o_proj .1

- تعداد پارامترها (بدون LoRA):

$$4096 \times 4096 = 16,777,216$$

- تعداد پارامترهای LoRA

$$2 \times (4096 \times 16) = 131,072$$

:up_proj و gate_proj .2

- تعداد پارامترها (بدون LoRA):

$$4096 \times 16384 = 67,108,864$$

- تعداد پارامترهای LoRA

$$2 \times (4096 \times 16) = 131,072$$

:down_proj .3

- تعداد پارامترها (بدون LoRA):

$$16384 \times 4096 = 67,108,864$$

- تعداد پارامترهای LoRA

$$2 \times (4096 \times 16) = 131,072$$

خلاصه تعداد پارامترها

ماژول	(پارامترها) بدون LoRA	(پارامترهای اضافه) با LoRA
q_proj	16,777,216	131,072
k_proj	16,777,216	131,072
v_proj	16,777,216	131,072
o_proj	16,777,216	131,072
gate_proj	67,108,864	131,072
up_proj	67,108,864	131,072
down_proj	67,108,864	131,072

در این پاسخ، تلاش می‌کنیم روش CoPrompt و شیوه مقابله آن با فراموشی فاجعه‌بار (وقتی مدل پس از یادگیری یک وظیفه جدید، عملکرد کلی خود را از دست می‌دهد) را گام‌به‌گام و با جزئیات بیشتر توضیح دهیم تا کاملاً روش شود چطور با ایجاد تغییرات ساختاری در مدل، این مشکل را کاهش می‌دهد و عملکرد مدل را در هر دو بعد (یادگیری وظایف جدید و حفظ/افزایش توانایی عمومی) بهبود می‌بخشد.

۱. ایده کلی CoPrompt: حفظ انسجام بین مدل پیش‌تمرین‌شده و مدل

فاین‌تیون‌شده

مدل‌های پیش‌تمرین‌شده (مثل CLIP) توانایی کلی و قدرت شناختی بالایی دارند زیرا روی دیتاست‌های بسیار بزرگ آموزش دیده‌اند. اگر ما برای یک تسک خاص (مثلاً تشخیص یک دسته تصاویر جدید) بخواهیم مدل را فاین‌تیون کنیم، معمولاً برعی از پارامترهای مدل را تغییر می‌دهیم یا پارامترهای جدیدی اضافه می‌کنیم. مشکل را اینجاست که در داده‌های کم (few-shot)، مدل ممکن است «دانش» گسترده خود را فدا کند و صرفاً روی آن داده‌ها بیش‌برازش کند که نتیجه‌اش فراموشی فاجعه‌بار است.

برای حل این مشکل، یک محدودیت انسجامی (Consistency Constraint) تعریف می‌کند که در واقع می‌گوید:

«پارامترهایی که برای یادگیری وظیفه جدید (few-shot) تغییر می‌کنند، نباید بازنمایی‌های نهایی مدل را زیاد از بازنمایی‌های قبلی (مدل اصلی) دور کنند.»

به عبارت ساده، خروجی مدل فاینتیون شده باید «شبیه» خروجی همان مدل پیش‌تمرین شده باقی بماند. در نتیجه:

1. توانایی یا دانش عمومی (general knowledge) مدل حفظ می‌شود.
 2. از بیش‌برازش روی دادهٔ جدید جلوگیری می‌کند.
-

۲. واردکردن اغتشاش در ورودی‌ها برای آموزش بهتر (Inputs)

۲.۱. متن

- در حالت عادی، برای نام کلاس مثل «cat»، مدل پیش‌تمرین شده CLIP از یک عبارت ساده مثل «a photo of a cat» استفاده می‌کند.
- در **CoPrompt**، برای ایجاد تنوع و جلوگیری از «یادگیری تک‌جمله‌ای»، از یک مدل زبان (مثل GPT) استفاده می‌شود تا یک جملهٔ غنی‌تر بسازد (مثل «یک موجود چهارپای خذار با گوش‌های تیز»).
- سپس انسجام بین بردار این توصیف جدید و توصیف قدیمی بررسی و کنترل می‌شود.

۲.۲. تصویر

- از افزایش‌های تصویری (Image Augmentations) مثل برش تصادفی، چرخش، یا flip استفاده می‌شود تا چند تصویر تغییریافته از یک ورودی بسازیم.
- مدل باید خروجی (embedding) خود را برای این تصاویر اغتشاش‌یافته نزدیک به خروجی‌های مدل پیش‌تمرین شده نگه دارد.

این کار سبب می‌شود که مدل فاینتیون شده یاد بگیرد حتی اگر ورودی تصاویر/متن کمی عوض شد، همچنان به شبک «داده‌های گستردگی» مدل اصلی عمل کند و از مسیر اصلی دور نشود.

۳. ترکیب دو روش مهم: پرامپت‌ها + آدابتورها

۳.۱. پرامپت‌ها (Prompts)

- پرامپت‌ها قطعاتی از بردارهای قابل آموزش هستند که در ورودی متن یا تصویر اضافه می‌شوند تا به مدل «راهنمایی» بیشتری دربارهٔ وظیفه جدید بدهند (مثلاً توضیح اضافی در بخش متنهای ساختار جدید در ورودی تصویری).
- پیش از این روش‌ها، گاهی پرامپت‌ها فقط برای متن بودند یا فقط برای تصویر؛ برخی روش‌ها (مثل MaPLe) هردو را داشتند اما ممکن بود مشکل بیش‌برازش به وجود بیايد.

۳.۲. آدابتورها (Adapters)

- آدابتورها مازولهای کوچکی هستند که در لایه‌های آخر مدل قرار می‌گیرند و بردار خروجی را تنظیم می‌کنند.
- مثلاً یک MLP دولایه که بردار ویژگی‌های مدل را می‌گیرد و خروجی را کمی دستکاری می‌کند تا به تسكیج جدید بهتر پاسخ دهد.

۳.۳. مزیت ترکیب

- CoPrompt می‌گوید: «چرا از هردو روش با هم استفاده نکنیم؟». چون پرامپت‌ها در سطح ورودی (text) و (image) عمل می‌کنند، اما آدابتورها در سطح خروجی (feature) قرار دارند. بنابراین مجموعاً انعطاف‌بیشتری داریم.
 - البته اضافه کردن پارامتر زیاد ممکن است خطر بیش‌برازش را افزایش دهد. اما محدودیت انسجامی (Consistency Constraint) کمک می‌کند این پارامترها همچنان در راستای مدل پیش‌تمرين شده باقی بمانند و از هدایت اشتباه جلوگیری می‌کند.
-

۴. چرا این کار جلو فراموشی فاجعه‌بار را می‌گیرد؟

1. محدودیت انسجامی مدل را وادار می‌کند که «دانش کلی» (General Knowledge) خود را کنار نگذارد و تنها در محدوده‌ای مشخص اجازه می‌دهد تا برای تسک جدید تنظیم شود.
 2. اغتشاش در ورودی باعث می‌شود مدل مجبور شود روی یک طیف وسیع‌تری از جملات یا تصاویر خودش را «مطابق» کند. در نتیجه، صرفاً به داده‌های اندک بسته نمی‌کند و قابلیت‌های پیشینش را حفظ می‌نماید.
 3. پرامپت‌ها + آداتورها می‌توانند آزادی عمل بیشتری برای یادگیری تسک جدید بدهند (بهتر فاین‌تیون شوند)، اما به خاطر قانون انسجام، نباید راه "غلط" را بروند. پس هم قدرت یادگیری بالا می‌رود و هم فراموشی رخ نمی‌دهد.
-

۵. دستاوردها و نتایج عملی

- طبق آزمایش‌های گزارش‌شده در مقاله:
 - عملکرد مدل در وظیفه جدید (few-shot) افزایش چشمگیری داشته است.
 - توانایی zero-shot (یعنی عمومی‌سازی مدل روی وظایف جدید بدون داده آموزشی) هم گاهی بیشتر از خود CLIP اولیه شده است؛ که بسیار جالب است چون معمولاً فاین‌تیونینگ سبب افت عملکرد zero-shot می‌شود.
 - در آزمون‌های cross-dataset (مدل روی یک دیتابست فاین‌تیون می‌شود ولی روی دیتابست دیگر تست می‌شود)، نتایج بسیار خوبی داشته است.
-

۶. جمع‌بندی کوتاه

:CoPrompt

1. یک محدودیت انسجامی تعریف می‌کند تا خروجی مدل فاین‌تیون شده نزدیک به خروجی مدل اصلی بماند.

2. از ورودی‌های اغتشاش‌یافته (جمله‌های توصیفی از LLM و تصاویر افزایشی) بهره می‌گیرد تا این انسجام
محکم‌تر شود.

3. پرامپت‌ها (در ورودی متن و تصویر) و آداتورها (در لایه‌های بالایی) را با هم ترکیب می‌کند؛ اما به کمک
آن محدودیت انسجامی، مانع بیش‌برازش می‌شود.

نتیجهٔ نهایی: مدل حتی پس از یادگیری تسك‌های خاص (با دادهٔ کم) همچنان دانش کلی خود را از دست نداده و می‌تواند در وظایف گسترده (zero-shot) عملکرد خوبی— حتی بهتر— داشته باشد؛ یعنی مشکل فراموشی فاجعه‌بار حل می‌شود یا بسیار کاهش می‌یابد.

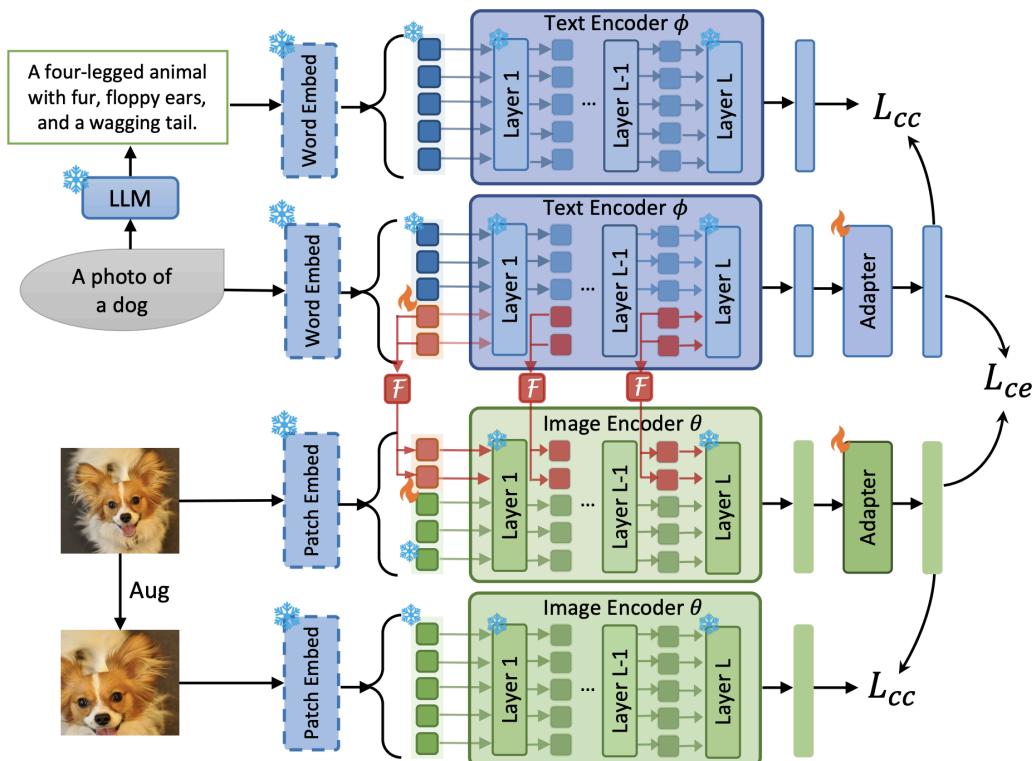


Figure 2: Overview of the proposed CoPrompt.

زیر بخش پنجم : پیاده سازی مدل ها

در این قسمت هر بخش از مدل ای را که داریم را توضیح میدهیم :

```
import os
import numpy as np

import logging
import torch
from datasets import load_dataset, load_from_disk
from google.colab import drive

from peft import LoraConfig, get_peft_model

from transformers import (
    PaliGemmaProcessor,
    PaliGemmaForConditionalGeneration,
    Trainer,
    TrainingArguments,
    DataCollatorForSeq2Seq,
    EarlyStoppingCallback,
    AutoConfig,
    BitsAndBytesConfig,
)
```

این بخش کد با هدف آماده سازی پروژه نوشته شده است. ابتدا کتابخانه های عمومی مانند `os` و `numpy` برای مدیریت فایل ها و محاسبات عددی، و `logging` برای ثبت گزارش ها فراخوانی می شوند. همچنین، از `torch` برای استفاده از امکانات PyTorch و `datasets` برای بارگذاری داده ها از منابع آنلاین یا محلی استفاده می شود. اتصال به Google Drive نیز برای مدیریت ذخیره سازی در Google Colab تنظیم شده است.

در بخش تخصصی تر، از کتابخانه PEFT برای تنظیم بهینه مدل ها (مانند LORA) و از `transformers` برای پردازش مدل های پیشرفته زبان طبیعی (مانند `PaliGemmaForConditionalGeneration`) استفاده شده است. علاوه بر این، ابزارهایی برای مدیریت آموزش مدل ها، مانند `TrainingArguments` و `Trainer` و `BitsAndBytesConfig` و توابعی برای مدیریت داده ها و تنظیمات پیشرفته مانند `BitsAndBytesConfig`، وارد شده اند.

```
def free_gpu_memory():
    import gc
    gc.collect()
    torch.cuda.empty_cache()

free_gpu_memory()
```

این قطعه کد برای آزادسازی حافظه GPU استفاده می‌شود و شامل دو مرحله اصلی است:

1. **جمعآوری حافظه استفاده نشده توسط Python**: با استفاده از `gc.collect()`، تمام اشیاء بدون استفاده (garbage) در حافظه پاکسازی می‌شوند. این به مدیریت حافظه کمک می‌کند و ممکن است فضای را برای اشیاء جدید باز کند.

2. **خالی کردن کش GPU در PyTorch**: با استفاده از `torch.cuda.empty_cache()`، حافظه GPU که توسط PyTorch کش شده است، آزاد می‌شود. این دستور حافظه را به‌طور فیزیکی آزاد نمی‌کند، اما آن را برای استفاده مجدد آماده می‌سازد.

تابع `free_gpu_memory` این دو عمل را انجام می‌دهد تا از مشکلات حافظه در GPU جلوگیری کند و به برنامه اجازه دهد تا از حافظه آزاد شده برای عملیات جدید استفاده کند.

```

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

if torch.cuda.is_available():
    device = torch.device("cuda")
    logger.info(f"Using device: {torch.cuda.get_device_name(0)}")
else:
    device = torch.device("cpu")
    logger.info("GPU not available, using CPU instead.")

train_save_path = '/content/drive/My Drive/train_subset'
eval_save_path = '/content/drive/My Drive/eval_subset'

if not os.path.exists(train_save_path):
    logger.info("Loading the CLEVR dataset...")
    dataset = load_dataset("HuggingFaceM4/clevr", "default")

    train_size = int(0.01 * len(dataset["train"]))
    eval_size = int(0.01 * len(dataset["validation"]))

    train_subset = dataset["train"].shuffle(seed=42).select(range(train_size))
    eval_subset = dataset["validation"].shuffle(seed=42).select(range(eval_size))

    logger.info(f"Training subset size: {len(train_subset)}")
    logger.info(f"Evaluation subset size: {len(eval_subset)}")

    train_subset.save_to_disk(train_save_path)
    eval_subset.save_to_disk(eval_save_path)

    logger.info(f"Training subset saved to {train_save_path}")
    logger.info(f"Evaluation subset saved to {eval_save_path}")
else:
    logger.info(f"Training subset already exists at {train_save_path}, loading...")
    train_subset = load_from_disk(train_save_path)

    logger.info(f"Evaluation subset already exists at {eval_save_path}, loading...")
    eval_subset = load_from_disk(eval_save_path)

```

این قطعه کد برای مدیریت و پردازش مجموعه داده **CLEVR** طراحی شده است و شامل عملیات بارگذاری، تقسیم‌بندی و ذخیره‌سازی داده‌ها است. ابتدا اتصال به Google Drive با استفاده از `drive.mount` انجام می‌شود تا امكان ذخیره و بازیابی داده‌ها از فضای ذخیره‌سازی ابری فراهم شود. همچنین، با تنظیم `logging` پیام‌ها و وضعیت اجرای کد ثبت می‌شود. دستگاه محاسباتی (CPU) یا GPU نیز شناسایی و انتخاب می‌شود و جزئیات آن در گزارش ثبت می‌گردد.

در مرحله بعد، بررسی می‌شود که آیا مسیرهای ذخیره زیرمجموعه‌های آموزشی و ارزیابی (eval_save_path و train_save_path) قبلاً ایجاد شده‌اند یا خیر. اگر مسیرها موجود نباشند، مجموعه داده **CLEVR** از مخزن HuggingFaceM4/clevr بارگذاری می‌شود. سپس، زیرمجموعه‌ای کوچک (1٪) از داده‌های آموزشی و ارزیابی به صورت تصادفی انتخاب و ذخیره می‌شود. این زیرمجموعه‌ها برای کاهش

حجم پردازش داده‌ها و تمرکز بر آزمایشات سریع طراحی شده‌اند. اندازه این زیرمجموعه‌ها نیز در گزارش ثبت می‌شود.

در صورتی که مسیرهای ذخیره قبلً وجود داشته باشند، زیرمجموعه‌های ذخیره‌شده به جای بارگذاری مجدد از اینترنت، مستقیماً از دیسک بارگذاری می‌شوند. این روش سرعت اجرای کد را افزایش می‌دهد و از مصرف منابع اضافی جلوگیری می‌کند. هدف اصلی این کد تضمین دسترسی سریع و آسان به مجموعه داده‌های آماده برای مدل‌سازی و کاهش زمان پردازش اولیه است.

```
logger.info("Loading model configuration...")
config = AutoConfig.from_pretrained("google/paligemma-3b-pt-224")

config.hidden_activation = "gelu_pytorch_tanh"

input_max_length = 128
label_max_length = 384

free_gpu_memory()

if not hasattr(config, 'decoder_start_token_id') or config.decoder_start_token_id is None:
    config.decoder_start_token_id = config.pad_token_id

logger.info("Loading processor and model with updated configuration...")
processor = PaliGemmaProcessor.from_pretrained([
    "google/paligemma-3b-pt-224",
    config=config
])
```

این بخش از کد برای بارگذاری و پیکربندی یک مدل از پیشآموزش‌دیده از کتابخانه Transformers استفاده می‌شود. توضیحات به صورت دقیق‌تر در سه بند ارائه می‌شود:

1. **بارگذاری و پیکربندی مدل:** ابتدا پیکربندی مدل با استفاده از `AutoConfig.from_pretrained` برای مدل `google/paligemma-3b-pt-224` بارگذاری می‌شود. سپس، یک ویژگی پیکربندی مدل، یعنی `hidden_activation`، به مقدار `"gelu_pytorch_tanh"` تغییر داده می‌شود. این ویژگی عملکرد لایه‌های فعال‌سازی مدل را کنترل می‌کند. مقادیر `input_max_length` و `label_max_length` نیز تعیین می‌شوند تا طول حداکثر ورودی و خروجی مدل محدود شود.

2. **مدیریت حافظه GPU:** تابع `free_gpu_memory` در دو نقطه مختلف فراخوانی می‌شود، یکی پس از پیکربندی مدل و دیگری پس از بارگذاری پردازشگر. این کار تضمین می‌کند که حافظه GPU از هر گونه

اطلاعات غیرضروری خالی شود تا فضای کافی برای بارگذاری مدل و پردازش‌ها فراهم گردد.

.3. بارگذاری پردازشگر و مدل: با استفاده از `PaliGemmaProcessor.from_pretrained`، پردازشگر و مدل همراه با پیکربندی به روزرسانی شده بارگذاری می‌شوند. اگر در پیکربندی موجود نباشد یا مقدار آن تعریف نشده باشد، مقدار `decoder_start_token_id` به جای آن تنظیم می‌شود. این کار برای اطمینان از تنظیمات مناسب برای توکن‌های `pad_token_id` و بیژه در مدل است. پیام‌های مناسب نیز برای نظارت و گزارش وضعیت ثبت می‌شوند.

```
bnb_config = BitsAndBytesConfig(  
    load_in_8bit=True,  
    llm_int8_threshold=6.0,  
    bnb_4bit_compute_dtype=torch.bfloat16  
)  
  
model = PaliGemmaForConditionalGeneration.from_pretrained(  
    "google/paligemma-3b-pt-224",  
    config=config,  
    device_map="auto",  
    load_in_8bit=True  
)  
  
lora_config = LoraConfig(  
    r=8,  
    lora_alpha=32,  
    target_modules=["q_proj", "o_proj"],  
    lora_dropout=0.1,  
    bias="none",  
    task_type="CAUSAL_LM"  
)
```

این بخش از کد به تنظیمات بهینه‌سازی مدل و بارگذاری آن برای کاهش مصرف حافظه و افزایش کارایی می‌پردازد. ابتدا، تنظیمات `BitsAndBytesConfig` تعریف شده است که به مدل اجازه می‌دهد در حالت دقیق پایین‌تر (8 بیت و 4 بیت) بارگذاری شود. این تنظیمات شامل `load_in_8bit=True` برای بارگذاری

مدل در حالت 8 بیتی و کاهش مصرف حافظه، و `bnb_4bit_compute_dtype=torch.bfloat16` برای انجام محاسبات داخلی با دقت `bfloat16` است. این تنظیمات عملکرد مدل را در دستگاههای با منابع محدود بهبود می‌دهد.

مدل `google/paligemma-3b-pt-224` با `PaliGemmaForConditionalGeneration` از مخزن 3B از `AutoModelForConditionalGeneration` استفاده از پیکربندی بهروزرسانی شده بارگذاری می‌شود. در اینجا از `device_map="auto"` استفاده شده تا توزیع مدل بین دستگاههای موجود (مانند GPU و CPU) به صورت خودکار انجام شود. همچنین `load_in_8bit=True` حافظه مورد نیاز برای مدل را کاهش می‌دهد و امکان اجرای آن بر روی دستگاههایی با محدودیت حافظه را فراهم می‌کند.

در ادامه، تنظیمات `LoraConfig` برای استفاده از تکنیک LORA (Low-Rank Adaptation) در بهینه‌سازی مدل تعریف شده است. این تنظیمات شامل پارامترهای کلیدی مانند `r=8` (مرتبه کاهش)، `lora_alpha=32` (وزن یادگیری)، و انتخاب ماژول‌های هدف (مانند `q_proj` و `o_proj`) است. این تنظیمات به کاهش تعداد پارامترهای قابل آموزش کمک می‌کنند و برای بهبود عملکرد و صرفه‌جویی در منابع در مدل‌های زبان علی `(CAUSAL_LM)` طراحی شده‌اند.

```
model = get_peft_model(model, lora_config)
model.print_trainable_parameters()
```

این بخش از کد برای اعمال تنظیمات LORA بر روی مدل و چاپ پارامترهای قابل آموزش آن است. توضیحات در سه بند:

1. اعمال LORA به مدل:

- با استفاده از تابع `get_peft_model`، تنظیمات تعریف شده در `lora_config` به مدل اصلی اضافه می‌شود. این فرآیند تغییراتی در مدل ایجاد می‌کند که باعث کاهش تعداد پارامترهای قابل آموزش و بهینه‌سازی منابع محاسباتی می‌شود. تکنیک LORA به جای

به روزرسانی تمام پارامترهای مدل، تنها تغییرات مربوط به برخی لایه‌ها (مانند `q_proj` و `o_proj`) را در نظر می‌گیرد.

2. چاپ پارامترهای قابل آموزش:

با استفاده از تابع `print_trainable_parameters`، تعداد و جزئیات پارامترهایی که برای آموزش مدل تنظیم شده‌اند، چاپ می‌شود. این کار به کاربر اجازه می‌دهد تا مطمئن شود که تنها بخش‌های خاصی از مدل برای آموزش تنظیم شده‌اند و بقیه قسمت‌ها ثابت باقی می‌مانند. این اطلاعات می‌تواند برای ناظرت و درک بهتر از مصرف منابع مفید باشد.

3. هدف کلی:

هدف اصلی این مرحله کاهش پیچیدگی و منابع مورد نیاز برای آموزش مدل است. این کار با تمرکز بر روی پارامترهای کلیدی و استفاده از روش LORA انجام می‌شود که برای مدل‌های بزرگ با منابع محدود، کارایی و عملکرد بالاتری ارائه می‌دهد.

```

def preprocess_function(batch):
    from PIL import Image
    questions = batch["question"]
    images = batch["image"]
    answers = batch["answer"]

    processed_images = []
    texts_with_image = []

    for q, img in zip(questions, images):
        try:
            if isinstance(img, Image.Image):
                pil_img = img.convert("RGB").resize((224, 224))
            else:
                pil_img = Image.open(img).convert("RGB").resize((224, 224))
            arr_img = np.array(pil_img)
        except Exception as e:
            logger.warning(f"Error processing image: {e}")
            arr_img = np.zeros((224, 224, 3), dtype=np.uint8)

        processed_images.append(arr_img)
        texts_with_image.append("<image> " + q)

    processed_images = np.array(processed_images)
    processed_images = processed_images.transpose(0, 3, 1, 2)
    processed_images = processed_images / 255.0

    encoder_inputs = processor(
        images=processed_images,
        text=texts_with_image,
        padding="max_length",
        truncation=True,
        max_length=input_max_length,
        return_tensors="pt",
        do_rescale=False
    )

    decoder_inputs = processor.tokenizer(
        answers,
        padding="max_length",
        truncation=True,
        max_length=label_max_length,
        return_tensors="pt"
    )

    labels_ids = decoder_inputs["input_ids"].clone()
    padding_mask = decoder_inputs["attention_mask"] == 0
    labels_ids[padding_mask] = -100

    encoder_inputs["labels"] = labels_ids

    return encoder_inputs

```

۱. ورودی‌های تابع و آماده‌سازی اولیه:

- تابع `preprocess_function` برای پردازش دسته‌ای از داده‌ها طراحی شده است و سه ورودی کلیدی از یک `batch` دریافت می‌کند:
 - شامل یک لیست از پرسش‌ها به صورت متن.
 - شامل یک لیست از تصاویر مرتبط با هر پرسش که می‌تواند به صورت فایل یا اشیاء تصویری (مانند `PIL.Image`) باشد.
 - شامل لیستی از پاسخ‌های متنی مرتبط با هر پرسش.
 - دو لیست خالی به نامهای `texts_with_image` و `processed_images` ایجاد می‌شود:
 - برای ذخیره تصاویر پردازش شده.
 - `texts_with_image` برای ترکیب متن پرسش‌ها با اطلاعات تصویری (با اضافه کردن `<image>` برچسب).
-

2. پردازش تصاویر:

- حلقه‌ای روی لیست‌های پرسش‌ها و تصاویر اجرا می‌شود:
 - برسی نوع تصویر: اگر تصویر از نوع `PIL.Image` باشد، با استفاده از `("convert("RGB` باشد، با استفاده از `PIL.Image` به فرمت RGB تبدیل و اندازه آن با متند `resize` به 224x224 تغییر داده می‌شود. این ابعاد استاندارد برای بسیاری از مدل‌های پیش‌آموزش دیده هستند.
 - باز کردن تصویر از فایل: اگر تصویر فایل باشد، با استفاده از `Image.open` باز و سپس همان فرایند تبدیل به RGB و تغییر اندازه انجام می‌شود.
 - مدیریت خطا: اگر پردازش تصویر با خطأ مواجه شود (مثلاً تصویر خراب باشد)، یک تصویر خالی (آرایه‌ای از صفرها با ابعاد 224x224x3) به جای تصویر اصلی ذخیره می‌شود و یک پیام هشدار ثبت می‌شود.
 - تصویر پردازش شده به آرایه NumPy تبدیل شده و در لیست `processed_images` ذخیره می‌شود.
 - متن پرسش به همراه برچسب `<image>` به `texts_with_image` اضافه می‌شود تا مدل اطلاعات متنی و تصویری را ترکیب کند.
-

3. تبدیل تصاویر به فرمت مناسب برای مدل:

- تصاویر پردازش شده که در قالب یک لیست ذخیره شده اند، به یک آرایه NumPy با ابعاد `(batch_size, 224, 224, 3)` تبدیل می شوند.
 - تغییر ترتیب ابعاد: ترتیب ابعاد تصاویر به `(batch_size, channels, height, width)` تا با فرمت ورودی استاندارد PyTorch `(batch_size, 3, 224, 224)` تبدیل می شود (با استفاده از `transpose(0, 3, 1, 2)`) تا با فرمت ورودی استاندارد سازگار شود.
 - نرمال سازی پیکسل ها: مقادیر پیکسل ها از بازه $[0, 255]$ به $[1, 0]$ تغییر می کنند (با تقسیم بر 255) تا مقادیر مقیاس شده مناسب برای مدل باشند.
-

4. پردازش ورودی ها برای انکودر (Encoder):

- با استفاده از پرداشگر `(processor)`، تصاویر نرمال شده و متن های ترکیب شده با برچسب `<image>` پردازش می شوند.
 - تنظیمات زیر برای انکودر اعمال می شود:
 - : متن ها تا طول مشخص شده با توکن پرکننده `(padding="max_length")` پر می شوند.
 - : متن هایی که طولشان بیشتر از مقدار تعیین شده است، بریده `(truncation=True)` می شوند.
 - : حداکثر طول متن ها تنظیم می شود `(max_length=input_max_length)`.
 - : خروجی به فرمت PyTorch Tensor `("return_tensors="pt")` تبدیل می شود.
 - : مشخص می کند که تصاویر قبل از نرمال سازی شده اند و نیازی به تنظیم مجدد نیست `(do_rescale=False)`.
-

5. پردازش ورودی ها برای دیکودر (Decoder):

- پاسخ های متنی `(answers)` با استفاده از توکنایزر مدل پردازش می شوند:
- مشابه انکودر، متن ها بریده یا پر می شوند و به طول `label_max_length` محدود می گردند.

- خروجی شامل `input_ids` (شناسه‌های توکن‌ها) و `attention_mask` (ماسک توجه) است.
 - این مقادیر مستقیماً به عنوان ورودی دیکودر استفاده می‌شوند.
-

6. تنظیم برچسب‌ها :`(Labels)`

- مقدار `input_ids` که نمایانگر توکن‌های پاسخ‌ها است، برای برچسب‌ها `labels` کپی می‌شود.
 - جایگزینی مقادیر `padding`: مقادیر مربوط به توکن‌های پرکننده (که در `attention_mask` با مقدار 0 مشخص شده‌اند) با مقدار -100 جایگزین می‌شوند. این کار تضمین می‌کند که مقادیر پرکننده در محاسبه `loss` مدل تأثیر نگذارند.
-

7. خروجی نهایی:

- یک دیکشنری شامل موارد زیر بازگردانده می‌شود:
 - و تصاویر نرمال‌شده به عنوان ورودی‌های انکودر.
 - `input_ids`, `attention_mask` و `labels` به عنوان برچسب‌ها برای محاسبه `loss`.
 - این دیکشنری آماده است تا به عنوان ورودی به مدل یادگیری عمیق برای آموزش یا پیش‌بینی داده شود.
-

هدف کلی:

اینتابع داده‌ها را به فرمتی استاندارد و مناسب برای مدل‌های یادگیری عمیق تبدیل می‌کند. این شامل مدیریت ترکیب اطلاعات متنی و تصویری، نرمال‌سازی داده‌ها، و تنظیم دقیق ورودی‌ها و برچسب‌ها است تا آموزش مدل بهینه و بدون خطأ باشد. همچنین مدیریت خطاهای مرتبط با داده‌های ناقص تضمین می‌کند که فرآیند پردازش متوقف نشود.

```

logger.info("Preprocessing the training dataset...")
num_proc = os.cpu_count() or 1 |

train_subset = train_subset.map(
    preprocess_function,
    batched=True,
    num_proc=num_proc,
    remove_columns=["image", "question", "answer"]
)
logger.info("Preprocessing the evaluation dataset...")
eval_subset = eval_subset.map(
    preprocess_function,
    batched=True,
    num_proc=num_proc,
    remove_columns=["image", "question", "answer"]
)

```

این بخش از کد وظیفه پیشپردازش دادههای آموزشی و ارزیابی را بر عهده دارد و شامل چندین گام کلیدی است که به شرح زیر توضیح داده میشود:

1. پیشپردازش دادههای آموزشی:

- **ثبت پیام شروع:** با استفاده از `logger.info`, فرآیند پیشپردازش دادههای آموزشی ثبت میشود تا در صورت نیاز وضعیت پردازش قابل پیگیری باشد.
- **تعداد پردازندهها:** تعداد پردازندههای موجود در سیستم با استفاده از `(os.cpu_count)` مشخص میشود. اگر مقدار مشخص وجود نداشته باشد، مقدار پیشفرض `1` در نظر گرفته میشود.
- **:map تابع**
- **تابع پردازشی:** دادهها با استفاده از تابع `preprocess_function` که قبلاً تعریف شده بود، پردازش میشوند. این تابع تصاویر را پردازش کرده، متنها را آماده میکند و دادهها را برای مدل به فرمت مناسب تبدیل میکند.

- پردازش دسته‌ای (`batched=True`): داده‌ها به صورت دسته‌ای پردازش می‌شوند تا سرعت عملکرد افزایش یابد.
 - پردازش موازی (`num_proc`): تعداد پردازنده‌ها برای اجرای موازی عملیات مشخص می‌شود، که باعث بهبود عملکرد در سیستم‌هایی با هسته‌های پردازشی بیشتر می‌شود.
 - حذف ستون‌های غیرضروری: ستون‌های اصلی (مانند `image, question, answer`) که دیگر نیازی به آن‌ها نیست، حذف می‌شوند تا داده‌های پردازش شده کوچک‌تر و ساده‌تر شوند.
 - خروجی این عملیات به روزشده در `train_subset` ذخیره می‌شود.
-

2. پیش‌پردازش داده‌های ارزیابی:

- ثبت پیام شروع: مشابه داده‌های آموزشی، پیام شروع پیش‌پردازش داده‌های ارزیابی ثبت می‌شود.
 - تابع `:map` • فرآیند مشابه داده‌های آموزشی است و شامل پردازش دسته‌ای، استفاده از پردازنده‌های موازی، و حذف ستون‌های غیرضروری می‌باشد.
 - خروجی این عملیات در `eval_subset` ذخیره می‌شود.
-

3. هدف کلی این بخش:

- این کد اطمینان حاصل می‌کند که داده‌های آموزشی و ارزیابی به طور کامل و بهینه پردازش می‌شوند و به فرمتی مناسب برای مدل تبدیل می‌شوند.
- با حذف ستون‌های غیرضروری، فضای ذخیره‌سازی کاهش یافته و سرعت عملیات افزایش می‌یابد.
- استفاده از پردازش موازی (چندپردازنده‌ای) و پردازش دسته‌ای سرعت پیش‌پردازش داده‌ها را بهینه می‌کند، به ویژه برای مجموعه داده‌های بزرگ.

```

train_subset.set_format("torch")
eval_subset.set_format("torch")

logger.info("Preprocessing completed.")

logger.info("Setting up the data collator...")
from torch.utils.data import default_collate

def custom_data_collator(features):
    for feat in features:
        feat.pop("decoder_input_ids", None)
        feat.pop("decoder_attention_mask", None)

    batch = default_collate(features)
    return batch

data_collator = custom_data_collator

logger.info("Setting up training arguments...")
from transformers import TrainingArguments

data_collator = custom_data_collator

```

این بخش کد به مرحله نهایی آماده‌سازی داده‌ها، تنظیم آرگومان‌های آموزش، و تعریف collator سفارشی می‌پردازد. جزئیات هر بخش به شرح زیر است:

۱. تنظیم فرمت داده‌ها:

- : ("set_format": "torch" •
 - با استفاده از این دستور، داده‌های آموزشی (train_subset) و ارزیابی (eval_subset) را در فرمت PyTorch تبدیل می‌شوند.
 - این تغییر به مدل اجازه می‌دهد تا داده‌ها را مستقیماً در قالب Tensors دریافت کند، که برای پردازش با PyTorch ضروری است.
 - ثبت پیام تکمیل پیش‌پردازش:

-
- پیام Preprocessing completed ثبت می‌شود تا پایان موفقیت‌آمیز عملیات پیش‌پردازش داده‌ها مشخص شود.

2. تعریف collator سفارشی برای داده‌ها:

- هدف collator داده‌ها:
 - وظیفه دارد داده‌ها را از فرمت دیکشنری‌های جداگانه به یک دسته (batch) تبدیل کند که مدل بتواند با آن کار کند.
 - استفاده شده است که به طور پیش‌فرض داده‌ها را به صورت دسته‌ای ترکیب می‌کند.
- :**(custom_data_collator)** سفارشی collator
 - در هر ویژگی (feature) از داده‌ها، کلیدهای `decoder_input_ids` و `decoder_attention_mask` (در صورت وجود) حذف می‌شوند. این مقادیر ممکن است برای این مرحله غیرضروری باشند و با حذف آن‌ها داده‌ها ساده‌تر شوند.
 - سپس باقی‌مانده ویژگی‌ها با استفاده از `default_collate` ترکیب می‌شوند تا به صورت دسته‌ای آماده شوند.
- تخصیص collator به متغیر `:data_collator`
 - این collator برای مراحل آموزش و ارزیابی مدل استفاده خواهد شد.

3. تنظیم آرگومان‌های آموزش:

- ثبت پیام تنظیم آرگومان‌ها:
 - پیام Setting up training arguments ثبت می‌شود تا نشان دهد مرحله تنظیم پارامترهای آموزش آغاز شده است.
- از مازول TrainingArguments در کتابخانه Transformers استفاده می‌شود (در این بخش تعریف جزئیات آرگومان‌ها دیده نمی‌شود، اما فرض می‌شود که در ادامه انجام خواهد شد).
- سفارشی collator که در مرحله قبل تعریف شده (`data_collator`), به عنوان collator داده برای این آرگومان‌ها تنظیم می‌شود.

هدف کلی این بخش:

- این کد داده‌ها را به فرمت PyTorch تبدیل کرده و فرآیند آماده‌سازی آن‌ها برای مدل را تکمیل می‌کند.
- با تعریف collator سفارشی، داده‌های ورودی به مدل به صورت دسته‌ای و با حذف اطلاعات غیرضروری سازمان‌دهی می‌شوند.
- این مراحل، پایه‌ای برای تنظیم آموزش مدل و اجرای فرآیندهای بعدی مانند آموزش و ارزیابی هستند.

```
logger.info("Setting up training arguments...")
training_args = TrainingArguments(
    output_dir=".//results",
    run_name="paligemma_clevr_run",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=2,
    per_device_eval_batch_size=4,
    gradient_accumulation_steps=2,
    num_train_epochs=1,
    weight_decay=0.01,
    logging_dir=".//logs",
    logging_steps=10,
    fp16=True,
    load_best_model_at_end=True,
    report_to=["tensorboard"],
    dataloader_num_workers=num_proc,
    disable_tqdm=False,
    optim="adamw_bnb_8bit",
    ddp_find_unused_parameters=False,
)
```

این بخش کد برای تنظیم آرگومانهای آموزش مدل طراحی شده است. با استفاده از کلاس `Training`، پارامترهای مختلفی برای آموزش و ارزیابی مدل تعریف می‌شود. توضیحات کامل هر پارامتر در زیر آورده شده است:

1. پیکربندی عمومی و ذخیره نتایج:

- مسیر خروجی برای ذخیره مدل‌های آموزش‌دیده و نتایج ذخیره می‌شود.
 - یک نام برای این اجرا (`run`) تعیین می‌شود که در ابزارهای گزارش‌گیری (مانند TensorBoard) قابل استفاده است.
-

2. استراتژی‌های ارزیابی و ذخیره‌سازی:

- ارزیابی مدل پس از هر epoch انجام می‌شود. این تضمین می‌کند که پس از پردازش کامل تمام داده‌های آموزشی، عملکرد مدل بررسی شود.
 - مدل پس از هر epoch ذخیره می‌شود. این به بازیابی بهترین نسخه مدل کمک می‌کند.
-

3. تنظیمات بهینه‌سازی و یادگیری:

- نرخ یادگیری اولیه برای بهینه‌سازی مدل. این مقدار کوچک انتخاب شده تا از نوسانات در پارامترهای مدل جلوگیری شود.
- برای جلوگیری از overfitting، کاهش وزن‌ها اعمال می‌شود که به منظم‌سازی کمک می‌کند.
- از بهینه‌ساز AdamW با نسخه 8 بیتی استفاده می‌شود که حافظه کمتری مصرف کرده و برای مدل‌های بزرگ مناسب است.

4. تنظیمات دسته‌بندی داده‌ها:

- اندازه دسته (batch size) برای داده‌های آموزشی روی `per_device_train_batch_size=2` هر دستگاه تنظیم می‌شود. مقدار کوچک انتخاب شده است تا با محدودیت حافظه GPU سازگار باشد.
 - اندازه دسته برای داده‌های ارزیابی `per_device_eval_batch_size=4` تنظیم شده تا ارزیابی سریع‌تر انجام شود.
 - با انباشتن گرادیان‌ها طی دو مرحله، یک دسته مؤثر `gradient_accumulation_steps=2` بزرگ‌تر شبیه‌سازی می‌شود. این تنظیم برای دستگاه‌هایی با محدودیت حافظه مفید است.
-

5. تعداد epoch‌ها و تنظیمات آموزش:

- تعداد epoch‌های آموزش به 1 تنظیم شده است. این ممکن است برای آزمایش اولیه باشد.
 - تعداد پردازندۀ‌های موازی برای بارگذاری داده‌ها مشخص می‌شود. این مقدار از تعداد هسته‌های CPU گرفته شده است.
-

6. ورود اطلاعات و گزارش‌گیری:

- مسیر ذخیره لاغ‌ها برای ابزارهای گزارش‌گیری مانند TensorBoard: `logging_dir=". /logs"`
 - پیام‌های لاغ هر 10 مرحله ثبت می‌شوند. این تنظیم برای مشاهده وضعیت آموزش در زمان واقعی مفید است.
 - ابزار گزارش‌دهی به TensorBoard تنظیم می‌شود تا کاربر بتواند پیشرفت مدل را مشاهده کند. ["report_to=["tensorboard
-

7. بهینه‌سازی حافظه و عملکرد:

- **fp16=True**: فعالسازی آموزش با دقت ترکیبی (16 بیت) برای بهبود سرعت و کاهش مصرف حافظه.
 - **Distributed**: این گزینه برای آموزش توزیع شده (ddp_find_unused_parameters=False) استفاده می شود و پردازش پارامترهای بدون استفاده را غیرفعال می کند.
-

8. مدیریت مدل در طول آموزش:

- **load_best_model_at_end=True**: بهترین مدل بر اساس معیار ارزیابی در انتهای آموزش بارگذاری می شود. این تضمین می کند که کاربر مدل بهینه را دریافت کند.
 - **disable_tqdm=False**: نوار پیشرفت (progress bar) غیرفعال نمی شود و در هنگام اجرا نمایش داده خواهد شد.
-

هدف کلی این بخش:

این تنظیمات بهینه‌ترین روش برای آموزش مدل را فراهم می‌کنند. از مدیریت منابع سخت‌افزاری (مانند GPU) گرفته تا استراتژی‌های ارزیابی و ذخیره‌سازی، همه به‌گونه‌ای طراحی شده‌اند که کارایی و دقت مدل را به حداقل برسانند. همچنین، گزارش‌گیری و تنظیمات لاغ‌ها باعث می‌شوند که پیشرفت آموزش به صورت شفاف قابل مشاهده باشد.

```

logger.info("Initializing the Trainer...")
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_subset,
    eval_dataset=eval_subset,
    tokenizer=processor.tokenizer,
    data_collator=data_collator,
    callbacks=[EarlyStoppingCallback(early_stopping_patience=2)],
)

logger.info("Starting training...")
try:
    trainer.train()
except Exception as e:
    logger.error(f"Training failed with error: {e}")
    raise e

logger.info("Training completed. Starting evaluation...")
eval_results = trainer.evaluate()
print("Final Evaluation Results:", eval_results)

```

این بخش از کد مسئول ایجاد و اجرای فرآیند آموزش مدل با استفاده از کلاس `Trainer` از کتابخانه Transformers است. توضیحات کامل هر قسمت:

1. ایجاد `Trainer`

- کلاس `Trainer` تمام مراحل آموزش و ارزیابی را به صورت خودکار مدیریت می‌کند. در اینجا پارامترهای زیر به آن ارسال می‌شوند:
 - مدل اصلی که با تنظیمات قبلی (مانند LORA و 8 بیتی) آماده شده است.
 - آرگومان‌های آموزش که قبلًا در `TrainingArguments` تعریف شده بودند (مانند نرخ یادگیری، تعداد epochها، و غیره).
 - مجموعه داده آموزشی که قبلًا پردازش شده و آماده است.

- مجموعه داده ارزیابی برای ارزیابی مدل پس از هر epoch
 - توکنایزر مدل برای پردازش متن‌ها (پرسش‌ها و پاسخ‌ها).
 - سفارشی که داده‌ها را به صورت دسته‌ای برای مدل آماده می‌کند.
 - callbacks=[EarlyStoppingCallback(early_stopping_patience=2] : یک callback برای توقف زودهنگام آموزش در صورتی که عملکرد مدل در داده‌های ارزیابی بهبود نیابد. پارامتر early_stopping_patience=2 تعیین می‌کند که اگر مدل پس از 2 مرحله متوالی بهبود نداشته باشد، آموزش متوقف شود.
-

2. شروع فرآیند آموزش:

- ثبت پیام شروع: با logger.info پیام شروع فرآیند آموزش ثبت می‌شود.
 - فرآیند آموزش مدل آغاز می‌شود. این دستور تمام تنظیمات از جمله داده‌ها، آرگومان‌های آموزش، و استراتژی‌های ارزیابی را اجرا می‌کند.
 - مدیریت خطاهای:
 - در صورت بروز خطا در فرآیند آموزش، خطا ثبت می‌شود و دوباره به بالا منتقل می‌شود (با raise e). این تضمین می‌کند که خطاهای مهم شناسایی و رفع شوند.
-

3. ارزیابی مدل پس از آموزش:

- ثبت پیام پایان آموزش و شروع ارزیابی: پیام مربوط به پایان موفقیت‌آمیز آموزش و آغاز ارزیابی ثبت می‌شود.
- فرآیند ارزیابی مدل با استفاده از داده‌های ارزیابی (eval_dataset) (trainer.evaluate) اجرا می‌شود.
- نتایج ارزیابی:

- نتایج نهایی ارزیابی مدل به صورت یک دیکشنری بازگردانده و چاپ می‌شوند. این دیکشنری شامل معیارهای کلیدی مانند دقت (accuracy)، از دست دادن (loss)، و معیارهای دیگر ارزیابی مدل است.
-

هدف کلی این بخش:

این کد با استفاده از `Trainer` فرآیند آموزش و ارزیابی مدل را ساده و خودکار می‌کند. این شامل آماده‌سازی داده‌ها، مدیریت توکنایزر، اعمال callbacks (مانند توقف زودهنگام)، و ارزیابی عملکرد مدل می‌شود. مدیریت خطاهای تضمین می‌کند که مشکلات احتمالی در طول آموزش به درستی ثبت و رفع شوند. در نهایت، نتایج ارزیابی به عنوان بازخوردی از عملکرد مدل در داده‌های ارزیابی ارائه می‌شود.

```
logger.info("Training completed. Starting evaluation...")
eval_results = trainer.evaluate()
logger.info(f"Final Evaluation Results: {eval_results}")

save_path = "./my_pali_lora_clevr_model"
logger.info(f"Saving the model to {save_path}...")
trainer.save_model(save_path)
processor.tokenizer.save_pretrained(save_path)

logger.info("Model and tokenizer saved successfully!")
```

این بخش کد مسئول ارزیابی نهایی مدل، ذخیره مدل آموزش‌دیده و توکنایزر مرتبط با آن است. توضیحات کامل:

۱. ارزیابی نهایی مدل:

- `:()trainer.evaluate` •
◦ مدل با استفاده از داده‌های ارزیابی `eval_dataset` (که در زمان تعریف `Trainer` تنظیم شده بود، ارزیابی می‌شود).

- معیارهایی مانند `loss` و دقت (`accuracy`) یا معیارهای دیگر ارزیابی می‌شوند و نتایج در قالب یک دیکشنری بازگردانده می‌شوند.
 - ثبت نتایج ارزیابی:
 - نتایج ارزیابی با استفاده از `logger.info` ثبت می‌شوند تا به صورت دقیق مستند شوند و قابل مشاهده باشند.
-

2. ذخیره مدل آموزش‌دیده:

- `:trainer.save_modelsave_path` •
 - مدل آموزش‌دیده در مسیر مشخص شده (`save_path`) ذخیره می‌شود. این شامل وزن‌های مدل و اطلاعات مربوط به تنظیمات آن است.
 - `:Processor.tokenizer.save_pretrained save_path` •
 - توکنایزر مرتبط با مدل (که برای پردازش متون ورودی استفاده می‌شود) نیز در همان مسیر ذخیره می‌شود. این تضمین می‌کند که مدل و توکنایزر همزمان برای بارگذاری مجدد و استفاده در آینده آماده هستند.
-

3. ثبت موفقیت‌آمیز ذخیره‌سازی:

- ثبت پیام‌های مربوط به ذخیره‌سازی:
 - پیام‌ها با استفاده از `logger.info` برای آغاز و اتمام موفقیت‌آمیز فرآیند ذخیره‌سازی مدل و توکنایزر ثبت می‌شوند.
 - این پیام‌ها اطمینان می‌دهند که فرآیند ذخیره‌سازی بدون خطا تکمیل شده است.
-

هدف کلی این بخش:

این کد مدل آموزش دیده را ارزیابی کرده و معیارهای عملکرد آن را ثبت می کند. سپس مدل و توکنایزر ذخیره می شوند تا بتوان از آنها در مراحل بعدی، مانند پیش بینی یا آموزش مجدد، استفاده کرد. این مرحله نهایی تضمین می کند که نتایج و مدل به طور کامل حفظ شده و آماده استفاده در پروژه های آینده هستند.

حال در این قسمت به سوالات مربوطه پاسخ خواهیم داد :

سوال دوم معیار : RELU score

امتیاز ROUGE (Recall-Oriented Understudy for Gisting Evaluation) مجموعه ای از معیارهای ارزیابی کیفیت متون تولید شده خودکار (مانند خلاصه ها، ترجمه ها یا پاسخ های چت بات ها) است. این معیار، با مقایسه متون تولید شده توسط ماشین با متون مرجع انسانی، شباهت میان آنها را می سنجد. به عبارت دیگر، هدف ROUGE ارزیابی این است که یک متن تولید شده تا چه حد محتوای اصلی را حفظ کرده و چقدر با متن مرجع شباهت دارد.

انواع معیارهای ROUGE

ROUGE-N .1

این معیار بر اساس n-gram ها (توالی های n تایی از کلمات) عمل می کند و میزان همپوشانی n-gram های مشترک را می سنجد.

- ROUGE-1: مقایسه یک تایی ها (unigrams). این معیار نشان دهنده همپوشانی کلمات منفرد بین متن تولید شده و مرجع است.
- ROUGE-2: مقایسه دوتایی ها (bigrams). این معیار دقیق تر بیشتری بر ترتیب کلمات دارد.
- ROUGE-3 و بیشتر: این معیارها برای ارزیابی n-gram های بلندتر استفاده می شوند و برای متون با ساختار پیچیده تر مفید هستند.

ROUGE-L .2

این معیار به طولانی ترین دنباله مشترک (LCS یا Longest Common Subsequence) بین متن تولید شده و مرجع می پردازد. این دنباله نیازی به حفظ ترتیب متوالی کلمات ندارد و ساختار کلی جملات را بهتر منعکس می کند. ROUGE-L برای سنجش شباهت های ساختاری و معنایی متن مناسب است.

ROUGE-S .3

این معیار یا Skip-Bigram برای ارزیابی حفتکلماتی است که ترتیب آنها در متن ثابت است اما ممکن است کلمات دیگری میان آنها قرار گرفته باشند. این معیار انعطاف بیشتری در مقایسه با ROUGE-2 دارد.

ROUGE-W .4

این نسخه از ROUGE وزن بیشتری به دنباله‌های طولانی‌تر از کلمات مشترک می‌دهد و برای متونی که انسجام معنایی مهم است کاربرد دارد.

:ROUGE محاسبه

محاسبات ROUGE معمولاً بر سه مفهوم کلیدی استوار است:

- **یادآوری (Recall)**

نسبت تعداد n-گرام‌های مشترک با مرجع به تعداد کل n-گرام‌های مرجع.

$$Recall = \frac{\text{تعداد های مشترک n-gram}}{\text{تعداد های مرجع n-gram}}$$

- **دقت (Precision)**

نسبت تعداد n-گرام‌های مشترک به تعداد کل n-گرام‌های متن تولیدشده.

$$Precision = \frac{\text{تعداد های مشترک n-gram}}{\text{تعداد های تولیدشده n-gram}}$$

- **F1 امتیاز**

میانگین هارمونیک یادآوری و دقت.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

کاربردهای ROUGE

1. ارزیابی سیستم‌های خلاصه‌سازی خودکار:

ROUGE برای ارزیابی این‌که خلاصه تولیدشده تا چه حد محتوای اصلی متن را بازتاب می‌دهد، بسیار مفید است.

2. ارزیابی ترجمه ماشینی:

در ترجمه ماشینی، ROUGE می‌تواند کیفیت ترجمه را با مقایسه آن با ترجمه انسانی بسنجد.

3. ارزیابی چتباتها و مدل‌های زبانی:

ROUGE برای سنجش کیفیت پاسخ‌های مدل‌های زبانی، مانند ChatGPT، استفاده می‌شود.

4. تحلیل شباهت متون:

این معیار می‌تواند برای سنجش شباهت میان هر دو متن، مانند متون دانشجویی با مقاله‌های مرجع، کاربرد داشته باشد.

مزایا و محدودیت‌ها:

مزایا:

- ساده و سریع: محاسبه ROUGE به ابزارهای پیچیده نیاز ندارد و می‌تواند به سرعت بر روی متون اعمال شود.

- معیار استاندارد: در بسیاری از پژوههای NLP و تحقیقات دانشگاهی به عنوان معیار استاندارد پذیرفته شده است.

محدودیت‌ها:

1. عدم درک معنایی: ROUGE فقط شباهت کلمات را می‌سنجد و توانایی درک معنای عمیق متن را ندارد. بنابراین اگر یک متن به خوبی پارافریز شده باشد ولی از کلمات متفاوتی استفاده کند، ROUGE ممکن است امتیاز پایینی بدهد.

2. حساسیت به خلاصه‌های مرجع: کیفیت و تنوع متون مرجع می‌تواند بر امتیازات ROUGE تأثیر زیادی بگذارد.

3. عدم ارزیابی انسجام: این معیار توانایی سنجش روانی، انسجام و ساختار منطقی متن را ندارد.
-

جمع‌بندی:

ابزاری قدرتمند برای ارزیابی خودکار متن تولید شده توسط ماشین است، اما نباید به تنها ای برای سنجش کیفیت کلی متن استفاده شود. برای بهبود نتایج، بهتر است ROUGE در کنار معیارهای معنایی یا ارزیابی انسانی به کار گرفته شود.

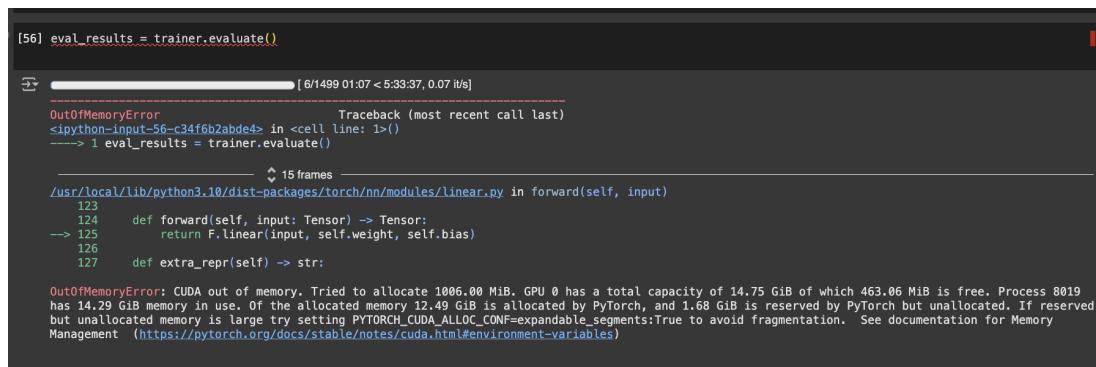
در این بخش به خاطر وجود زمان کم نتوانستیم به خوبی مدل را ترین کنیم :

پس تنها در دو epoch به یادگیری مدل پرداخته بودیم که به خاطر کمبود سخت افزار ما نتوانستیم مدل را باز دوم بر روی آن ایپاک ها اجرا کنیم و مدل تنها یک بار بر روی کل داده ها آموزش داده شد

بدین صورت تنها ما یک بار داده هارا آموزش دادیم که حدود یک ساعت زمان برد بود و در انتهای آن را در یک ریپازیتوری در [hugging face](#) ذخیره سازی کردیم [به این آدرس](#).

و بعد در پایین کد ها به evaluate کردن آن ها پرداختیم

در زمان evaluate کردن به خاطر حجم بالای مدل نمیتوانیم آن را evaluate کنیم .



```
[56] eval_results = trainer.evaluate()

[ 6/1499 01:07 < 5:33:37, 0.07 It/s]

OutOfMemoryError: Traceback (most recent call last)
<ipython-input-56-c34f6b2abde4> in <cell line: 1>()
--> 1 eval_results = trainer.evaluate()

          ▾ 15 frames
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/linear.py in forward(self, input)
 123
 124     def forward(self, input: Tensor) -> Tensor:
--> 125         return F.linear(input, self.weight, self.bias)
 126
 127     def extra_repr(self) -> str:

OutOfMemoryError: CUDA out of memory. Tried to allocate 1006.00 MiB. GPU 0 has a total capacity of 14.75 GiB of which 463.06 MiB is free. Process 8019 has 14.29 GiB memory in use. Of the allocated memory 12.49 GiB is allocated by PyTorch, and 1.68 GiB is reserved by PyTorch but unallocated. If reserved but unallocated memory is large try setting PYTORCH_CUDA_ALLOC_CONF=expandable_segments=True to avoid fragmentation. See documentation for Memory Management (https://pytorch.org/docs/stable/notes/cuda.html#environment-variables)
```

۱. مرور مدل پایه (CLIP)

- در روش CoPrompt، فرض بر این است که ما یک مدل ویژن-زبان از نوع CLIP (یا مشابه آن) داریم که شامل:
1. یک انکودر تصویر با پارامترهای θ .
 2. یک انکودر متن با پارامترهای ϕ .

CLIP معمولاً به طور پیش-تمرین شده (pre-trained) در دسترس است و می‌تواند به صورت صفرشات (zero-shot) بر اساس عبارت‌های متنی و تصاویر، طبقه‌بندی انجام دهد.

۱.۱. فرمول خروجی مدل CLIP به شکل صفرشات

برای یک ورودی تصویر x و یک مجموعه متن‌های مربوط به هر کلاس $w_k = \{w_k\}_{k=1}^C$ (مثلًا "a photo" می‌تواند "of [class k]" باشد)، بردار تصویری z چنین است:

$$z = \theta(x)$$

همچنین هر متن w_k با انکودر متن تبدیل می‌شود به:

$$w_k = \phi(\text{فراخوانی جمله‌ی مربوط به کلاس } k)$$

سپس احتمال خروجی مدل برای کلاس y (از بین C کلاس) طبق تابع سافت‌مکس روی شباهت کسینوسی است:

$$p(y|x) = \frac{\exp(\text{sim}(z, w_y)/\tau)}{\sum_{k=1}^C \exp(\text{sim}(z, w_k)/\tau)}$$

که در آن:

- $\text{sim}(a, b)$ نشان‌دهنده شباهت کسینوسی (cosine similarity) بین دو بردار a و b است.
- τ دمای (temperature) اسکالر است.

۲. ایده‌ی یادگیری پرامپت (Prompt Learning)

در رویکردهای جدید (مانند CoOp)، به جای جمله‌ی ثابتی مثل "a photo of [class]"، یک سری بردارهای قابل آموزش (u_1, u_2, \dots, u_m) به ابتدای توکن‌های کلاسی اضافه می‌شوند. مثلاً اگر نام کلاس c_k باشد، متن نهایی:

$$t_k = \{u_1, u_2, \dots, u_m, c_k\}$$

و آنگاه متن ورودی به انکودر متن ϕ داده می‌شود:

$$\phi(t_k)$$

در CoPrompt، پرامپت‌ها هم برای تصویر و هم برای متن در نظر گرفته می‌شوند.

۳. مشکل فراموشی فاجعه‌بار و راه حل

مدل‌های پیش‌تمرین شده (مانند CLIP) وقتی برای داده‌های کم (few-shot) فاین‌تیون می‌شوند، دچار بیش‌برازش (Overfitting) می‌شوند و ظرفیت "zero-shot" قبلی خود را از دست می‌دهند که به آن فراموشی (Consistency Constraint) معروفی می‌کند. روش CoPrompt برای رفع این مشکل، یک قیود انسجامی (Consistency Constraint) معرفی می‌کند.

۱. تابع زیان پایه (Cross-Entropy)

ابتدا تابع زیان عادی ناظارت شده (Supervised) را در نظر می‌گیریم. فرض کنید CE تابع کراس‌انتروپی باشد که مدل را وادار می‌کند نمونه‌ها را به کلاس درست بچسباند. اگر خروجی تصویر را (i) و خروجی متن مربوط به کلاس k را (t_k) ϕ بنامیم، مدل احتمال زیر را می‌دهد:

$$p(y|x) = \frac{\exp(\text{sim}(\theta(i), \phi(t_y))/\tau)}{\sum_{k=1}^C \exp(\text{sim}(\theta(i), \phi(t_k))/\tau)}$$

سپس کراس‌انتروپی برای برجسب y چنین است:

$$L_{ce} = -\log \left(\frac{\exp(\text{sim}(\theta(i), \phi(t_y))/\tau)}{\sum_{k=1}^C \exp(\text{sim}(\theta(i), \phi(t_k))/\tau)} \right)$$

۳.۲. قیود انسجامی (Consistency Constraint)

۳.۲.۱. ایده کلی

CoPrompt می‌گوید خروجی مدل فاین‌تیون شده (که شامل پرامپت و آداتورهای قابل‌آموزش است) نباید فاصله زیادی از خروجی مدل پیش‌تمرین شده داشته باشد. برای این منظور:

- انکودر پیش‌تمرین شده (فریز) = θ برای تصویر، ϕ برای متن
- مدل قابل‌آموزش = $\theta_a(\theta(i))$ برای تصویر (جایی که θ_a آداتور است)، $(\phi_a(\phi(t)))$ برای متن (جایی که ϕ_a آداتور است)، و پرامپتهايی که در ورودی دخیل‌اند.

به این ترتیب، در روش CoPrompt باید embeddings نهایی شبیه هم بمانند.

۳.۲.۲. ورودی‌های متنوع (Perturbed Inputs)

۱. در بخش متن: به جای این که تنها از جمله‌ی ساده ("a photo of a dog") استفاده شود، با یک LLM (مثلًا GPT) جمله‌ی توصیفی تری (s_y) ایجاد می‌شود. پس دو متن داریم:

- t_y : جمله با پرامپت قابل‌آموزش
- s_y : جمله توصیفی تولیدشده توسط GPT (مدل فریز) آنگاه باید بردارهای متن (s_y) و $\phi_a(\phi(t_y))$ با هم منسجم باشند.

۲. در بخش تصویر: تصویر اصلی را مثلًا x می‌نامیم. آن را با تابع δ تغییر (Augment) می‌دهیم و $x' = \delta(x)$ می‌سازیم. سپس باید $\theta_a(\theta(i))$ و $\theta_a(\theta(x'))$ با هم منسجم باشند.

۳.۲.۳. فرمول زیان انسجامی

در مقاله، از فاصله کسینوسی (Cosine Distance) استفاده شده است. به طور ساده:

$$\text{sim}(a, b) = \frac{a \cdot b}{\|a\| \|b\|}$$

و برای حفظ نزدیکی، از فرمولی شبیه زیر استفاده می‌شود (در مقاله به صورت (3) و (4) و (5) آمده است):

$$L_{cc} = 2 - \frac{\phi(s_y) \cdot \phi_a(\phi(t_y))}{\|\phi(s_y)\| \|\phi_a(\phi(t_y))\|} - \frac{\theta(x') \cdot \theta_a(\theta(i))}{\|\theta(x')\| \|\theta_a(\theta(i))\|}$$

- این عبارت در حقیقت می‌گوید دو جفت بردار باید کسینوسشان را تا حد ممکن بالا ببرند (یعنی فاصله کسینوسی تا حد ممکن کم شود).
- عبارت $-2 \cdot (\text{sim} + \text{sim})$ به این خاطر است که هر sim در بازه‌ی $[1, -1]$ است و ما می‌خواهیم مقدارش را حداکثر کنیم. با این نگارش، کمینه کردن L_{cc} یعنی بیشینه کردن کسینوس شباهت.

۴. ترکیب زیان کلی

در نهایت، زیان **CoPrompt** از دو بخش تشکیل می‌شود:

- .1: زیان نظارت شده (کراس انترپری) روی داده‌های جدید.
- .2: زیان انسجامی.

با ضریب وزنی λ ، فرمول نهایی:

$$L = L_{ce} + \lambda L_{cc}$$

که در مقاله به صورت (7) آمده است.

۵. آداتورها (Adapters) و پرامپتها (Prompts) همراه با انسجام

علاوه بر پرامپتها که در ورودی متن/تصویر استفاده می‌شوند، **آداتورها** را هم معرفی می‌کند تا در خروجی هریک از انکودرها، تبدیل‌های قابل آموزش انجام دهد:

- ϕ_a : آداتور برای خروجی انکودر متن
- θ_a : آداتور برای خروجی انکودر تصویر

بدین ترتیب، وقتی مدل می‌خواهد یک نمونه تصویر x را پردازش کند:

$$\theta(i) \rightarrow \theta_a(\theta(i))$$

و همین‌طور برای متن:

$$\phi(t_k) \rightarrow \phi_a(\phi(t_k))$$

محدودیت انسجامی هم اعمال می‌شود تا این بردارهای "تغییرشده" به بازنمایی اصلی نزدیک بمانند.

۶. چرا فراموشی فاجعه‌بار حل می‌شود؟

1. انسجام با مدل اصلی: چون ϕ_a و θ_a و نیز پرامپتها قابل آموزش اجازه ندارند هرچقدر می‌خواهند از مدل قدیمی دور شوند (به خاطر L_{cc} ، مدل جدید، دانش عمومی‌سازی اش را حفظ می‌کند).
2. ورودی‌های متنوع: نمونه‌های متنی غنی و افزوده‌های مختلف تصویری همگی سبب می‌شوند مدل فقط به چند نمونه محدود بسته نکند، بلکه بازه‌ای از توصیف‌ها/تصاویر را یاد بگیرد.
3. جمع‌شدن پرامپتها و آداتورها: مدل آزادی عمل بیشتری دارد تا با داده جدید بهتر هماهنگ شود، اما محدودیت انسجامی جلوی بیش‌برازش را می‌گیرد.

سوال دوم : Flow Matching

در این بخش، هر یک از سه شرط اصلی را که مقاله برای برقراری برابری گرادیان‌ها در Flow Matching و Conditional Flow Matching مطرح می‌کند، با جزئیات بیشتری توضیح می‌دهیم. به یاد بیاورید که این شرایط برای آن هستند که

$$\nabla_{\theta} L_{\text{FM}}(\theta) = \nabla_{\theta} L_{\text{CFM}}(\theta)$$

برقرار شود.

۱. مرور کلی از ایده‌ی Conditional Flow Matching و Flow Matching

۱.۱. مسئله اصلی: یادگیری یک مسیر انتقال بین نویز و داده

در مدل‌های مولد (Generative Models) بر پایه جریان‌های پیوسته یا همان Continuous Normalizing Flows (CNF)، ایده این است که داده‌ها را در طول زمان $t \in [0, 1]$ از توزیعی ساده (مثلًا نویز گاوی) به توزیع داده واقعی هدایت کنیم. به عبارت دیگر، می‌خواهیم توزیع π_0 (نویز) طی یک مسیر احتمالاتی به π_1 (داده واقعی) برسد.

- در رویکرد Flow Matching (FM): ما می‌گوییم یک میدان برداری $v_{\theta}(x, t)$ داریم که در هر لحظه از زمان و در هر نقطه‌ی فضا، مشخص می‌کند «چطور نقطه‌ها باید حرکت کنند». همچنین یک مسیر «هدف» $u_t(x)$ هست که مسیر درست را نشان می‌دهد. زیان L_{FM} براساس فاصله بین $(v_{\theta}(x, t) \text{ و } u_t(x))$ تعریف می‌شود.
- در رویکرد Conditional Flow Matching (CFM): در عوض، برای هر نمونه واقعی (x_1) ، یک مسیر احتمالاتی شرطی $(x_1 | x)$ تعریف می‌شود که داده‌ها را از نویز به همان نمونه x_1 می‌رساند. سپس میدان برداری شرطی $(x_1 | x)$ به شکل جداگانه برای هر نمونه تعریف می‌گردد. زیان L_{CFM} بر پایه فاصله بین v_{θ} و میدان برداری شرطی $u_t(x | x_1)$ ساخته می‌شود.

مقاله نشان می‌دهد که اگر تمام این مسیرهای شرطی را به شکل مناسب ترکیب (یا به بیان مقاله، کنیم، دقیقاً همان مسیر کلی (یعنی مسیر از نویز تا داده‌های همه نمونه‌ها) به دست می‌آید.

۲. فرمول‌های ریاضی CFM و FM

۱. زیان در (L_{FM}) Flow Matching

در FM، فرض بر این است که یک میدان برداری $u_t(x)$ می‌شناسیم که توزیع $p_t(x)$ را از نویز به داده می‌برد؛ اما می‌خواهیم آن را با یک شبکهٔ قابل‌آمورش $v_\theta(x, t)$ تقریب بزنیم. برای این کار، زیان زیر را تعریف می‌کنیم:

$$L_{\text{FM}}(\theta) = \mathbb{E}_{t, p_t(x)} \left\| v_\theta(x, t) - u_t(x) \right\|^2,$$

که در آن

- t از توزیع یکنواخت روی $[0, 1]$ نمونه گرفته می‌شود،
- x از توزیع احتمالاتی $(p_t(x))$ (مسیری که نویسنده‌ها دوست دارند از آن تعییت کنند)،
- $\|\cdot\|$ هم یک نرم (معمولًاً اقلیدسی).

در اینجا مشکل این است که یافتن (x) و $u_t(x)$ در عمل ساده نیست؛ ما توزیع داده واقعی را داریم، اما شکل بستهٔ مسیر کامل در فضا و زمان را نداریم.

۲. زیان در (L_{CFM}) Conditional Flow Matching

در روش CFM، مسیرهای شرطی تعریف می‌کنیم: برای هر مثال واقعی x_1 ، می‌گوییم یک مسیر احتمالاتی $(p_t(x | x_1))$ داشته باشیم که آغازش نویز ساده باشد و انتهایش حول و حوش x . همچنین میدان برداری شرطی $(u_t(x | x_1))$ را طوری تعریف می‌کنیم که همین مسیر را در زمان هدایت کند. سپس زیان زیر ساخته می‌شود:

$$L_{\text{CFM}}(\theta) = \mathbb{E}_{t, q(x_1), p_t(x|x_1)} \left\| v_\theta(x, t) - u_t(x | x_1) \right\|^2,$$

که در آن:

- x_1 از توزیع داده (q) نمونه می‌شود،
- x نیز از توزیع شرطی $((p_t(x | x_1))$.

نکتهٔ کلیدی مقاله نشان می‌دهد که درست است که تعریف CFM ظاهراً ساده‌تر و جدأگانه برای هر نمونه است، اما وقتی برای کل نمونه‌ها میانگین می‌گیریم، همان مسیر کلی (مارجینال) به دست می‌آید که FM دنبالش بود. یعنی مقاله ثابت می‌کند که زیان L_{CFM} و زیان L_{FM} ، از دید θ یکسان عمل می‌کنند (جز یک جملهٔ ثابت که به θ وابسته نیست).

۳. بیان دقیق دلایل برابری گرادیان

در مقاله، دو قضیه (Theorem 1 و Theorem 2) مطرح شده که خلاصه‌شان چنین است:

۱. قضیه‌ی ۱: اگر برای هر داده x_1 میدان برداری شرطی $(x | x_1) u_t(x)$ تعریف شود و آن مسیر شرطی $\{p_t(x | x_1)\}$ را بسازد، آنگاه وقتی این مسیرهای شرطی و میدان‌های مربوطه را مناسب ترکیب کنیم (با انتگرال‌گیری روی x_1 ، مسیر کلی (مارجینال) $p_t(x)$ و میدان کلی $u_t(x)$ ساخته می‌شود. فرمول دقیق این ترکیب این است:

$$p_t(x) = \int p_t(x | x_1) q(x_1) dx_1, \quad u_t(x) = \int u_t(x | x_1) \frac{p_t(x | x_1) q(x_1)}{p_t(x)} dx_1.$$

درنتیجه میدان برداری مارجینال $(x | x_1) u_t(x)$ دقیقاً همان می‌شود که مسیر کلی را می‌سازد.

۲. قضیه‌ی ۲: وقتی از این ساختار استفاده می‌کنیم، زیان CFM و زیان FM فقط به اضافه/کم کردن یک جمله ثابت (نسبت به θ) با هم تفاوت دارند. درنتیجه گرادیانشان عیناً برابر است. بهزیان دیگر:

$$L_{\text{FM}}(\theta) = L_{\text{CFM}}(\theta) + \text{جمله ثابت در مورد } \theta.$$

بنابراین

$$\nabla_\theta L_{\text{FM}}(\theta) = \nabla_\theta L_{\text{CFM}}(\theta).$$

مقاله این نکته را به صورت تشریحی و با جابه‌جایی انتگرال‌ها نشان می‌دهد. ایده اصلی این است که در زیان FM یک انتگرال مارجینال (روی $p_t(x)$) وجود دارد؛ در زیان CFM یک انتگرال دوگانه (روی $(x | x_1)$ و $q(x_1)$) هست که معادل آن است.

۴. شرایطی که حتماً باید برقرار باشند

در متن مقاله، برای اینکه این برابری بیمشکل برقرار شود، فرض‌هایی نظیر زیر مطرح می‌شوند:

۱. مثبت بودن چگالی $(x_t p)$ در سراسر دامنه: یعنی هیچ نقطه‌ای نباشد که چگالی صفر بشود؛ تا بتوانیم تقسیم بر $(x_t p)$ انجام دهیم و از جابه‌جایی انتگرال استفاده کنیم.
۲. وجود داشتن میدان‌های برداری شرطی: باید روشی باشد که میدان برداری $\{u_t(x | x_1) \}$ واقعاً مسیر شرطی بین نویز و x_1 را تولید کند.
۳. همگرایی مناسب: اطمینان از اینکه جابه‌جایی ترتیب انتگرال‌ها (Fubini's theorem) و نتیجه‌گیری گرادیان مشترک، از لحاظ تئوری مجاز باشد.

۵. چرا این نتیجه مهم است؟

- از دید عملی، یادگیری $(x u_t)$ (مسیر کلی) به صورت مستقیم بسیار سخت است؛ چون برای هر x باید مسیر را بدانیم.
- اما با سازوکار شرطی (CFM) می‌توانیم "به‌ازای هر نمونه x " یک مسیر خیلی ساده‌تر بسازیم (مثلاً مسیری که نویز را به "اطراف" همان x_1 ببرد) و میدان برداری‌اش را حساب کنیم.
- مقاله نشان می‌دهد که همین "یادگیری شرطی ساده" (CFM) دقیقاً معادل آن است که ما میدان کلی (FM) را یاد می‌گیریم.
- به بیان دیگر، هزینه یادگیری کم می‌شود و محاسبات مقیاس‌پذیر می‌گردد، بدون اینکه نتیجه تغییر کند.

زیر بخش دوم :

این سوال را بدین صورت می‌توان اثبات کرد :

در ادامه، یک اثبات گام به گام و توضیحات مربوطه را می بینید که چگونه با ارجاع به معادله (11) در مقاله Flow Matching و تئورم 3 (که می گوید بردار میدان $(x \mid x_1) u_t(x)$ همان مسیر احتمالاتی گاووسی شرطی را تولید می کند)، به فرمول

$$u_t(x \mid x_1) = \frac{\sigma'_t(x_1)}{\sigma_t(x_1)} (x - \mu_t(x_1)) + \mu'_t(x_1)$$

می رسیم.

۱. مرور چارچوب مسئله (Setup)

هدف ما بررسی یک مسیر (Path) احتمالاتی گاووسی است که شرطی بر نمونه x_1 است. به شکل زیر فرض کنید:

$$p_t(x \mid x_1) = \mathcal{N}(x \mid \mu_t(x_1), \sigma_t(x_1)^2 I),$$

در این توزیع شرطی:

- $\mu_t(x_1)$ تابع میانگین (Mean) است که به زمان t و نمونه x_1 وابسته است.
- $\sigma_t(x_1)$ نیز تابع انحراف معیار (اسکالر) است که به t و x_1 وابستگی دارد.
- t در بازه $[0, 1]$ تغییر می کند.

ایده ای اصلی این است که در $t = 0$ یک توزیع نویزی ساده (مثلاً گاووسی معیار) داشته باشیم و هنگام $t = 1$ داده ما حول نقطه x_1 (به صورت گاووسی کوچک) باشد. می خواهیم یک جریان یا فلوی زمانی بسازیم تا نمونه ها طی زمان از «نویز» به «داده» منتقل شوند.

در مقاله، معادله (11) یک نگاشت زمانمند (time-dependent map) را چنین تعریف می کند:

$$\psi_t(x) = \sigma_t(x_1)x + \mu_t(x_1),$$

به طوری که اگر $p_0(x)$ توزیع گاووسی معیار $\mathcal{N}(0, I)$ باشد، آنگاه نگاشت ψ_t در هر لحظه t ، این توزیع را به $p_t(x \mid x_1)$ می برد؛ یعنی:

$$[\psi_t]_* p_0(x) = p_t(x \mid x_1).$$

اما سؤال این است که میدان برداری $(x \mid x_1) u_t(x)$ که متناظر با این ψ_t است (یعنی همان چیزی که معادله دیفرانسیل عادی $d\psi_t/dt$ را توجیه می کند) چیست؟

در تئورم 3 مقاله آمده است:

$$u_t(x \mid x_1) = \frac{\sigma'_t(x_1)}{\sigma_t(x_1)} (x - \mu_t(x_1)) + \mu'_t(x_1),$$

و ما می خواهیم نشان دهیم که این دقیقاً همانی است که ψ_t را به صورت یک جریان زمانی به دست می دهد.

۲. تعریف ψ_t و مشتق گرفتن از آن

ا. بر پایه معادله (11) مقاله:

$$\psi_t(x) = \sigma_t(x_1)x + \mu_t(x_1).$$

۲. اکنون از ψ_t نسبت به زمان t مشتق می‌گیریم. برای نمایش مشتق زمانی، از نماد ψ'_t استفاده می‌کنیم؛
یعنی $\psi'_t = \frac{d}{dt}\psi_t$. لذا:

$$\psi'_t(x) = \frac{d}{dt} [\sigma_t(x_1)x + \mu_t(x_1)] = \sigma'_t(x_1)x + \mu'_t(x_1).$$

در اینجا:

- $\sigma'_t(x_1)$ مشتق تابع σ_t نسبت به t است (با ثابت بودن x_1)
- $\mu'_t(x_1)$ هم مشتق تابع μ_t نسبت به t است.

۳. تعیین میدان برداری ($u_t(\cdot | x_1)$)

حال می‌خواهیم یک میدان برداری $u_t(\cdot | x_1)$ تعریف کنیم به طوری که:

$$\frac{d}{dt}\psi_t(x) = u_t(\psi_t(x) | x_1).$$

به بیان دیگر، اگر $y = \psi_t(x)$ نام‌گذاری شود، آنگاه ODE زیر برقرار باشد:

$$\psi'_t(x) = u_t(y | x_1), \quad \text{با } y = \psi_t(x).$$

۳.۱. وارون‌پذیری ψ_t

نگاشت ψ_t چون یک تبدیل خطی+انتقال است ($\sigma_t(x_1)$ اسکالر مثبت)، قطعاً معکوس دارد. مشخصاً:

$$\psi_t^{-1}(y) = \frac{y - \mu_t(x_1)}{\sigma_t(x_1)}.$$

۳.۲. مقداردهی میدان در نقطه y

برای اینکه بدانیم $u_t(y | x_1)$ چیست، به این نکته توجه می‌کنیم:

$$u_t(\psi_t(x) | x_1) = \psi'_t(x).$$

اگر $y = \psi_t^{-1}(y)$ باشد، پس $y = \psi_t(x)$ لذا:

$$u_t(y | x_1) = \psi'_t(\psi_t^{-1}(y)).$$

۳. جایگذاری وارون در مشتق

از مرحله‌ی قبل دیدیم:

$$\psi'_t(x) = \sigma'_t(x_1)x + \mu'_t(x_1).$$

اگر $x = \psi_t^{-1}(y)$ را جایگزین کنیم:

$$\psi'_t(\psi_t^{-1}(y)) = \sigma'_t(x_1)\psi_t^{-1}(y) + \mu'_t(x_1).$$

ولی از رابطه‌ی وارون:

$$\psi_t^{-1}(y) = \frac{y - \mu_t(x_1)}{\sigma_t(x_1)}.$$

در نتیجه:

$$\psi'_t(\psi_t^{-1}(y)) = \sigma'_t(x_1) \frac{y - \mu_t(x_1)}{\sigma_t(x_1)} + \mu'_t(x_1).$$

حالا عامل $\frac{\sigma'_t(x_1)}{\sigma_t(x_1)}$ را می‌توانیم بیرون بکشیم:

$$\psi'_t(\psi_t^{-1}(y)) = \frac{\sigma'_t(x_1)}{\sigma_t(x_1)} [y - \mu_t(x_1)] + \mu'_t(x_1).$$

این دقیقاً مقدار $u_t(y | x_1)$ است. اگر بخواهیم به جای y دوباره (x) بنویسیم، می‌توانیم:

$$u_t(x | x_1) = \frac{\sigma'_t(x_1)}{\sigma_t(x_1)} (x - \mu_t(x_1)) + \mu'_t(x_1).$$

۴. نتیجه: همان فرمول مورد اشاره در تئورم ۳

از این روند متوجه می‌شویم که:

$p_t(x | x_1)$ مسیر (Flow) مورد نظر است که توزیع گاوی معیار را به $\psi_t(x) = \sigma_t(x_1)x + \mu_t(x_1)$ ۱. می‌برد.

۲. مشتق زمانی (Equation 11) مقاله آن، به فرمول ساده‌ی بالا ختم می‌شود.

۳. پس میدان برداری (ODE) همانی است که تئورم ۳ تصریح می‌کند:

$$u_t(x | x_1) = \frac{\sigma'_t(x_1)}{\sigma_t(x_1)} (x - \mu_t(x_1)) + \mu'_t(x_1).$$

و این میدان برداری دقیقاً معادله‌ی جریان (ODE) را محقق می‌کند که مسیر شرطی گاوی را به طور صحیح تولید می‌نماید.

جمع‌بندی

این چند مرحله ساده (۱) تعریف ψ به عنوان یک نگاشت خطی+انتقال، (۲) مشتق‌گیری از آن نسبت به زمان، و (۳) جایگزینی وارون (ψ_t^{-1}) نشان می‌دهد که میدان برداری $u_t(x | x_1)$ به صورت

$$u_t(x | x_1) = \frac{\sigma'_t(x_1)}{\sigma_t(x_1)} (x - \mu_t(x_1)) + \mu'_t(x_1)$$

پاسخ سوال سوم :

در این توضیح، سعی می‌کنیم جزئیات بیشتری از مسئله حمل و نقل بهینه (Optimal Transport, OT) و نحوه بهکارگیری آن در Flow Matching ارائه دهیم. سپس مزایا و معایب استفاده از OT در این چارچوب را بیشتر باز می‌کنیم.

۱. مرور کلی بر حمل و نقل بهینه (Optimal Transport)

۱.۱. بیان مسئله

در مسئله حمل و نقل بهینه (OT)، هدف آن است که دو توزیع (مثلًا α و β) را در یک فضای \mathbb{R}^d (یا منیفولد دیگر) با کمترین «هزینه» به هم مرتبط سازیم.

- توزیع α می‌تواند به عنوان «توزیع مبدأ» و β «توزیع مقصد» تلقی شود.
- "هزینه" معمولاً یک تابعی از فاصله (مثل $\|x - y\|^2$ یا $\|x - y\|$) است که بیان می‌کند جایه‌جایی جرم از نقطه x در α به نقطه y در β چه میزان هزینه دارد.

۱.۲. فرم ریاضی

یکی از توابعی که اغلب در OT استفاده می‌شود:

$$\min_T \mathbb{E}_{x \sim \alpha} [c(x, T(x))], \quad \text{if } T_* \alpha = \beta,$$

که در آن:

$$\mathbb{R}^d \rightarrow \mathbb{R}^d \quad \text{نگاشت (Transport Map) است،}$$

- $T_* \alpha = \beta$ به این معنی است که اگر از α نمونه‌برداری کنیم و آنگاه آن را توسط T ببریم، در نهایت به β برسد.

۱.۳. نکته کلیدی

وقتی تابع هزینه مربع فاصله باشد ($|x - y|^2$)، در برخی حالات (مثلاً تبدیل گاووسی ساده به گاووسی دیگر)، پاسخ OT منحصر به فرد و معمولاً خطی (یا «خط مستقیم» بین نقاط متناظر) خواهد بود. این نکته بسیار مهمی است، زیرا در کارهای اخیر **Flow Matching** می‌توان از این خطی بودن برای ساده‌سازی یادگیری استفاده کرد.

۲. استفاده از OT در مدل‌های Flow Matching

مدل‌های **Flow Matching** (FM) اساساً می‌خواهند یک مسیر احتمالاتی زمانمند بسازند که توزیع ساده‌ی نویز (مانند $\mathcal{N}(0, I)$) را در $t=0$ به توزیع داده در $t=1$ برسانند. در مقاله‌های رایج، این مسیر را اغلب با ایدهٔ دیفیوژن

پیاده‌سازی می‌کنند؛ یعنی نویز به تدریج تبدیل به داده می‌شود. اما یکی از پیشنهادهای جدید، استفاده از مسیر حمل و نقل بهینه است:

۱. طراحی مسیر شرطی (Conditional OT)

- برای هر نمونهٔ واقعی x_1 ، می‌توان مسیر $\{p_t(x | x_1)\}$ تعریف کرد که در $t=0$ ، یک نویز ساده باشد و در $t=1$ ، یک گوسین کوچک دور و بر x_1 .
- با رویکرد OT، این تغییر می‌تواند یک خط مستقیم در فضای ویژگی باشد (یعنی هر ذره (x) به صورت مستقیم به سمت (x_1) حرکت می‌کند).

۲. میدان برداری ساده‌تر

- اگر نگاشت $t \mapsto p_t(x_1)$ بین دو توزیع گاووسی (نویز و داده) ساخته شود، در عمل سرعت حرکت ذرات (میدان برداری) به شکل نسبتاً ساده‌ای درمی‌آید؛ ذرات با سرعت ثابت یا خط مستقیم به هدفشان می‌روند.
- در FM، کافی است ما آن میدان برداری را (در معادلهٔ CFM) به صورت بسته (closed-form) داشته باشیم و شبکهٔ عصبی هم آن را با θ تقریب بزنند.

۳. ارتباط با Flow Matching

- در Flow Matching، هدف نهایی این است که میدان برداری $\{v_\theta\}$ یاد گرفته شود تا مسیر احتمالاتی خروجی، داده‌ها را از نویز به داده برساند.
- اگر مسیر OT را انتخاب کنیم، یادگیری این میدان برداری راحت‌تر می‌شود؛ چون مسیرها معمولاً مستقیم‌اند و افت و خیز کمتری دارند.

از این‌رو، در مقالهٔ پیشنهاد می‌شود به جای مسیرهای "دیفیوژن" که ممکن است پیچیده باشند، مسیرهای مبتنی بر OT را بسازیم که کوتاه‌تر و سریع‌تر قابل یادگیری هستند.

۴. مزایای استفاده از OT در Flow Matching

۱. مسیرهای خطی (آسان‌تر شدن یادگیری)

- مسیر OT در بسیاری از حالات گاوی مسیر خطی یا نزدیک به خطی است که یادگیری آن برای مدل ساده‌تر است.
- در دیفیوژن، گاه مسیر پرپیچ و خمی داریم که شبکه باید آن را فرا بگیرد (در زمان طولانی‌تری).

۲. آموزش سریع‌تر

- چون میدان برداری (Velocity Field) ساده است، گرادیان‌ها و فاز یادگیری سریع‌تر به نتیجه می‌رسد.
- مقاله نشان می‌دهد که مدل OT-Based اغلب زودتر همگرا می‌شود.

۳. نمونه‌گیری کارآمد

- هنگامی که مدل آموزش دید، برای تولید نمونه (Sampling) باید معادله ODE را از $t=0$ تا $t=1$ حل کنیم.
- اگر مسیر ساده باشد، حلگر ODE با گام‌های کمتری (NFE) می‌تواند به دقت و کیفیت قابل قبولی برسد؛ زیرا لازم نیست چرخش‌ها یا انحراف‌های غیرضروری را دنبال کند.

۴. کیفیت یا تعمیم بهتر

- در برخی تست‌ها، متريک‌هایی مثل log-likelihood مثل FID (کیفیت تصویر) بهتر از روش دیفیوژن گزارش شده است. علت احتمالی: مدل بردارهای "اضافی" را یاد نمی‌گیرد و مستقیماً بر قسمت اصلی مسیر تمرکز می‌کند.

۵. معایب یا محدودیت‌های OT

۱. در توزیع‌های بسیار پیچیده، OT ممکن است ایده آل نباشد

- اگر توزیع داده بسیار چندوجهی یا غیر محدب باشد، مسیر OT ممکن است برای شبکه سخت‌تر شود یا نتواند برخی "اثرات نویز گونه" (که در دیفیوژن ممکن است مفید باشد) را ایجاد کند.

۲. نیاز به طراحی مسیر شرطی

- ما در این روش باید برای هر $x_1 \times x_2$, یک گوسین کوچک دورش (σ_{\min}) در نظر بگیریم؛ اگر خوب انتخاب نشود، ممکن است تنوع نمونه پایین بباید یا فرآیند یادگیری مختلف شود.

۳. فرم بسته (closed-form) همیشه در دسترس نیست

- وقتی دو توزیع (نویز و داده) خیلی پیچیده باشند، پیدا کردن نگاشت OT به صورت صریح (مثل همان خطی بودن در حالت گاوی) ممکن است نشدنی یا بسیار پیچیده باشد. مزیت سادگی در چنین مواردی از بین می‌رود.
-

جمع‌بندی:

- روشی کلاسیک برای تبدیل "حداقلی هزینه" بین دو توزیع است.
- در Flow Matching، استفاده از OT اجازه می‌دهد مسیرهای خطی یا نسبتاً ساده بسازیم که ضمن حفظ خواص مطلوب، سرعت یادگیری و تولید نمونه را بالا ببرد.
- در عین حال، اگر توزیع بسیار پیچیده باشد یا طراحی مسیر شرطی نامناسب باشد، ممکن است OT نتواند مزیتش را نشان دهد یا حتی در دسرساز شود.

در نتیجه، OT در Flow Matching یک گزینه جذاب و کارآمد است؛ بهویژه اگر توزیع اولیه و هدف (نویز و داده) ساده یا نزدیک به حالت گاوی باشند و ما بتوانیم فرم بسته‌اش را بیابیم.

مراجع :

PaliGemma: A versatile 3B VLM for transfer

Sigmoid Loss for Language Image Pre-Training

LoRA: Low-Rank Adaptation of Large Language Models

Consistency-guided Prompt Learning for Vision-Language Models

Flow Matching for Generative Modeling