مهندسی پرامپت

نویسنده: لی بونسترا

Google

تقدیر و تشکر

بازبین‌ها و مشارکت‌کنندگان

Michael Sherman

Yuan Cao

Erick Armbrust

Anant Nawalgaria

Antonio Gulli

Simone Cammel

گردآورندگان و ویراستاران

Antonio Gulli

Anant Nawalgaria

Grace Mollison

نویسنده فنی

Joey Haymaker

طراح

Michael Lanning

فهرست مطالب

لازم نیست دانشمند داده یا مهندس یادگیری ماشین باشید – هر کسی می‌تواند یک پرامپت بنویسد.

مقدمه

وقتی به ورودی و خروجی یک **مدل زبان بزرگ (Large Language Model یا LLM)** فکر می‌کنیم، یک پرامپت متنی (گاهی همراه با مُدالیته‌های دیگر مثل پرامپت‌های تصویری) ورودی‌ای است که مدل برای پیش‌بینی یک خروجی خاص از آن استفاده می‌کند. لازم نیست دانشمند داده یا مهندس یادگیری ماشین باشید – هر کسی می‌تواند یک پرامپت بنویسد. با این حال، ساختن مؤثرترین پرامپت می‌تواند پیچیده باشد. جنبه‌های زیادی از پرامپت شما بر کارایی آن تأثیر می‌گذارد: مدلی که استفاده می‌کنید، داده‌های آموزشی مدل، پیکربندی‌های مدل، انتخاب کلمات، سبک و لحن، ساختار و زمینه، همگی اهمیت دارند. بنابراین، **مهندسی پرامپت (Prompt Engineering)** یک فرآیند تکرارشونده است. پرامپت‌های ناکافی می‌توانند منجر به پاسخ‌های مبهم و نادرست شوند و توانایی مدل را در ارائه خروجی معنی‌دار مختل کنند.

(توضیح: «مدل زبان بزرگ» یا LLM، مثل همین مدلی که داره با شما صحبت می‌کنه، یک نوع هوش مصنوعیه که با حجم عظیمی از متن آموزش دیده تا بتونه زبان انسان رو بفهمه و متن تولید کنه. «مهندسی پرامپت» هم یعنی هنر و علم نوشتن دستورات یا همون «پرامپت»های خوب برای این مدل‌ها تا بهترین جواب رو ازشون بگیریم.)

وقتی با چت‌بات Gemini[1] چت می‌کنید، در واقع دارید پرامپت می‌نویسید. اما این مقاله بیشتر روی نوشتن پرامپت برای مدل Gemini در Vertex AI یا از طریق **API (رابط برنامه‌نویسی کاربردی)** تمرکز داره، چون با پرامپت دادن مستقیم به مدل، به پیکربندی‌هایی مثل «دما» (temperature) و غیره دسترسی خواهید داشت.
(توضیح: API یا Application Programming Interface، یه جور رابط نرم‌افزاریه که اجازه می‌ده دو تا برنامه مختلف با هم صحبت کنن. مثلاً شما از طریق API به مدل Gemini دستور می‌دید.)

این مقاله، مهندسی پرامپت رو با جزئیات بررسی می‌کنه. ما به تکنیک‌های مختلف پرامپت‌نویسی نگاهی می‌اندازیم تا به شما کمک کنیم شروع کنید و نکات و بهترین شیوه‌ها رو برای تبدیل شدن به یک متخصص پرامپت‌نویسی به اشتراک می‌گذاریم. همچنین برخی از چالش‌هایی که ممکنه هنگام ساخت پرامپت با اون‌ها مواجه بشید رو بررسی می‌کنیم.

مهندسی پرامپت

یادتون باشه یک LLM چطور کار می‌کنه؛ اون یک موتور پیش‌بینیه. مدل، متن متوالی رو به عنوان ورودی می‌گیره و بعد پیش‌بینی می‌کنه که **توکن (token)** بعدی چی باید باشه، بر اساس داده‌هایی که با اون‌ها آموزش دیده. LLM طوری عملیاتی شده که این کار رو بارها و بارها تکرار کنه، و توکن پیش‌بینی‌شده‌ی قبلی رو به انتهای متن متوالی اضافه می‌کنه تا توکن بعدی رو پیش‌بینی کنه. پیش‌بینی توکن بعدی بر اساس رابطه بین توکن‌های قبلی و چیزهایی که LLM در طول آموزشش دیده، انجام می‌شه.
(توضیح: «توکن» کوچکترین واحد متنیه که مدل‌های زبانی پردازش می‌کنن. می‌تونه یک کلمه، بخشی از کلمه یا حتی یک کاراکتر باشه.)

وقتی یک پرامپت می‌نویسید، در واقع دارید سعی می‌کنید LLM رو طوری تنظیم کنید که دنباله‌ی درستی از توکن‌ها رو پیش‌بینی کنه. مهندسی پرامپت فرآیند طراحی پرامپت‌های باکیفیتیه که LLMها رو برای تولید خروجی‌های دقیق راهنمایی می‌کنه. این فرآیند شامل آزمون و خطا برای پیدا کردن بهترین پرامپت، بهینه‌سازی طول پرامپت، و ارزیابی سبک نوشتاری و ساختار پرامپت در ارتباط با وظیفه مورد نظره. در زمینه پردازش زبان طبیعی و LLMها، پرامپت ورودی‌ایه که به مدل داده می‌شه تا پاسخ یا پیش‌بینی تولید کنه.

این پرامپت‌ها می‌تونن برای دستیابی به انواع مختلفی از وظایف درک و تولید متن استفاده بشن، مثل خلاصه‌سازی متن، استخراج اطلاعات، پرسش و پاسخ، طبقه‌بندی متن، ترجمه زبان یا کد، تولید کد، و مستندسازی کد یا استدلال.

لطفاً برای دیدن مثال‌های ساده و مؤثر پرامپت‌نویسی، به راهنماهای پرامپت‌نویسی گوگل[۲،۳] مراجعه کنید.

وقتی مهندسی پرامپت می‌کنید، با انتخاب یک مدل شروع می‌کنید. پرامپت‌ها ممکنه نیاز به بهینه‌سازی برای مدل خاص شما داشته باشن، صرف نظر از اینکه از مدل‌های زبان Gemini در Vertex AI، GPT، Claude، یا یک مدل منبع‌باز مثل Gemma یا LLaMA استفاده می‌کنید.

علاوه بر پرامپت، باید با پیکربندی‌های مختلف یک LLM هم کار کنید و اون‌ها رو تغییر بدید.

پیکربندی خروجی LLM

بعد از انتخاب مدل، باید پیکربندی مدل رو مشخص کنید. بیشتر LLMها گزینه‌های پیکربندی مختلفی دارن که خروجی LLM رو کنترل می‌کنن. مهندسی پرامپت مؤثر نیازمند تنظیم بهینه این پیکربندی‌ها برای وظیفه شماست.

طول خروجی

یک تنظیم پیکربندی مهم، تعداد توکن‌هایی است که در یک پاسخ تولید می‌شن. تولید توکن‌های بیشتر نیازمند محاسبات بیشتری از LLM است، که منجر به مصرف انرژی بالاتر، زمان پاسخ‌دهی بالقوه کندتر، و هزینه‌های بالاتر می‌شه.

کاهش طول خروجی LLM باعث نمی‌شه که LLM در خروجی‌ای که ایجاد می‌کنه، از نظر سبکی یا متنی موجزتر بشه؛ فقط باعث می‌شه LLM پس از رسیدن به حد مجاز، پیش‌بینی توکن‌های بیشتر رو متوقف کنه. اگه نیاز شما به طول خروجی کوتاهیه، احتمالاً باید پرامپت خودتون رو هم طوری مهندسی کنید که با این موضوع سازگار باشه.

محدودیت طول خروجی به ویژه برای برخی از تکنیک‌های پرامپت‌نویسی LLM، مثل ReAct، مهمه؛ جایی که LLM پس از پاسخی که شما می‌خواید، به انتشار توکن‌های بی‌فایده ادامه می‌ده.

کنترل‌های نمونه‌برداری

LLMها رسماً یک توکن واحد رو پیش‌بینی نمی‌کنن. بلکه، LLMها احتمالاتی رو برای توکن بعدی ممکن پیش‌بینی می‌کنن، که هر توکن در واژگان LLM یک احتمال دریافت می‌کنه. سپس از این احتمالات توکن نمونه‌برداری می‌شه تا مشخص بشه توکن تولیدی بعدی چی خواهد بود. دما (Temperature)، Top-K، و Top-P رایج‌ترین تنظیمات پیکربندی هستن که تعیین می‌کنن چطور احتمالات توکن پیش‌بینی‌شده برای انتخاب یک توکن خروجی واحد پردازش بشن.

دما (Temperature)

«دما» درجه تصادفی بودن در انتخاب توکن رو کنترل می‌کنه. دماهای پایین‌تر برای پرامپت‌هایی خوبن که انتظار پاسخ قطعی‌تری دارن، در حالی که دماهای بالاتر می‌تونن منجر به نتایج متنوع‌تر یا غیرمنتظره‌تر بشن. دمای ۰ (که بهش **رمزگشایی حریصانه یا Greedy Decoding** هم میگن) قطعیه: همیشه توکنی با بالاترین احتمال انتخاب می‌شه (البته توجه داشته باشید که اگه دو توکن بالاترین احتمال یکسان رو داشته باشن، بسته به نحوه پیاده‌سازی شکستن تساوی، ممکنه همیشه با دمای ۰ خروجی یکسانی دریافت نکنید).
(توضیح: «رمزگشایی حریصانه» یعنی مدل همیشه کلمه‌ای رو انتخاب می‌کنه که در اون لحظه محتمل‌ترین گزینه به نظر می‌رسه، بدون اینکه به آینده دورتر فکر کنه.)

دماهای نزدیک به حداکثر، تمایل به ایجاد خروجی تصادفی‌تری دارن. و هرچه دما بالاتر و بالاتر می‌ره، همه توکن‌ها به طور مساوی احتمال داره که توکن پیش‌بینی‌شده بعدی باشن.

کنترل دمای Gemini رو می‌شه شبیه به تابع سافت‌مکس (softmax) که در یادگیری ماشین استفاده می‌شه، درک کرد. تنظیم دمای پایین، آینه‌ای از دمای پایین سافت‌مکس (T) است، که بر یک دمای ترجیحی واحد با قطعیت بالا تأکید می‌کنه. تنظیم دمای بالاتر Gemini مانند دمای بالای سافت‌مکس است، که طیف وسیع‌تری از دماها رو در اطراف تنظیم انتخاب‌شده قابل قبول‌تر می‌کنه. این عدم قطعیت افزایش‌یافته، سناریوهایی رو در بر می‌گیره که در اون‌ها دمای دقیق و سفت‌وسخت ممکنه ضروری نباشه، مثلاً هنگام آزمایش با خروجی‌های خلاقانه.

Top-K و Top-P

Top-K و Top-P (که به عنوان **نمونه‌برداری هسته‌ای یا Nucleus Sampling**[۴] هم شناخته می‌شه) دو تنظیم نمونه‌برداری هستن که در LLMها برای محدود کردن توکن پیش‌بینی‌شده بعدی به توکن‌هایی با بالاترین احتمالات پیش‌بینی‌شده استفاده می‌شن. مانند دما، این تنظیمات نمونه‌برداری، تصادفی بودن و تنوع متن تولید شده رو کنترل می‌کنن.
(توضیح: «نمونه‌برداری هسته‌ای» یا Top-P روشیه که مدل فقط از بین محتمل‌ترین کلمات بعدی، اون‌هایی رو انتخاب می‌کنه که مجموع احتمالشون به یه حد معینی (P) برسه.)

- **Top-K:** نمونه‌برداری Top-K، تعداد K توکن محتمل‌تر رو از توزیع پیش‌بینی‌شده مدل انتخاب می‌کنه. هرچه Top-K بالاتر باشه، خروجی مدل خلاقانه‌تر و متنوع‌تره؛ هرچه Top-K پایین‌تر باشه، خروجی مدل محدودتر و واقعی‌تره. Top-K برابر با ۱ معادل رمزگشایی حریصانه است.
- **Top-P:** نمونه‌برداری Top-P، توکن‌های برتری رو انتخاب می‌کنه که احتمال تجمعی اون‌ها از یک مقدار معین (P) تجاوز نکنه. مقادیر P از ۰ (رمزگشایی حریصانه) تا ۱ (همه توکن‌ها در واژگان LLM) متغیر هستن.

بهترین راه برای انتخاب بین Top-K و Top-P اینه که با هر دو روش (یا هر دو با هم) آزمایش کنید و ببینید کدوم یک نتایجی رو که به دنبالش هستید تولید می‌کنه.

یک تنظیم پیکربندی مهم دیگر، تعداد توکن‌هایی است که در یک پاسخ تولید می‌شوند. آگاه باشید، تولید توکن‌های بیشتر نیاز به محاسبات بیشتری از LLM دارد که منجر به مصرف انرژی بالاتر و زمان پاسخ‌دهی بالقوه کندتر می‌شود، که این خود منجر به هزینه‌های بالاتر می‌شود.

جمع‌بندی همه چیز

انتخاب بین Top-K، Top-P، دما، و تعداد توکن‌های تولیدی، به کاربرد خاص و نتیجه مطلوب بستگی دارد، و این تنظیمات همگی بر یکدیگر تأثیر می‌گذارند. همچنین مهم است که مطمئن شوید درک می‌کنید که مدل انتخابی شما چگونه تنظیمات مختلف نمونه‌برداری را با هم ترکیب می‌کند.

اگر دما، Top-K، و Top-P همگی در دسترس باشند (مانند Vertex Studio)، توکن‌هایی که هم معیارهای Top-K و هم Top-P را برآورده می‌کنند، کاندیداهای توکن پیش‌بینی‌شده بعدی هستند، و سپس دما برای نمونه‌برداری از بین توکن‌هایی که از معیارهای Top-K و Top-P عبور کرده‌اند، اعمال می‌شود. اگر فقط Top-K یا Top-P در دسترس باشد، رفتار مشابه است اما فقط از یک تنظیم Top-K یا P استفاده می‌شود.

اگر دما در دسترس نباشد، هر توکنی که معیارهای Top-K و/یا Top-P را برآورده کند، به طور تصادفی برای تولید توکن پیش‌بینی‌شده بعدی انتخاب می‌شود.

در تنظیمات حدی یک مقدار پیکربندی نمونه‌برداری، آن یک تنظیم نمونه‌برداری یا سایر تنظیمات پیکربندی را لغو می‌کند یا بی‌اهمیت می‌شود.

- اگر دما را روی ۰ تنظیم کنید، Top-K و Top-P بی‌اهمیت می‌شوند – محتمل‌ترین توکن، توکن پیش‌بینی‌شده بعدی می‌شود. اگر دما را بسیار بالا تنظیم کنید (بالاتر از ۱ – معمولاً در حد ۱۰ها)، دما بی‌اهمیت می‌شود و هر توکنی که از معیارهای Top-K و/یا Top-P عبور کند، به طور تصادفی برای انتخاب توکن پیش‌بینی‌شده بعدی نمونه‌برداری می‌شود.

- اگر Top-K را روی ۱ تنظیم کنید، دما و Top-P بی‌اهمیت می‌شوند. فقط یک توکن از معیار Top-K عبور می‌کند، و آن توکن، توکن پیش‌بینی‌شده بعدی است. اگر Top-K را بسیار بالا تنظیم کنید، مثلاً به اندازه واژگان LLM، هر توکنی با احتمال غیر صفر برای بودن توکن بعدی، معیار Top-K را برآورده می‌کند و هیچ‌کدام انتخاب نمی‌شوند.
- اگر Top-P را روی ۰ (یا یک مقدار بسیار کوچک) تنظیم کنید، بیشتر پیاده‌سازی‌های نمونه‌برداری LLM فقط محتمل‌ترین توکن را برای برآورده کردن معیار Top-P در نظر می‌گیرند، و دما و Top-K را بی‌اهمیت می‌کنند. اگر Top-P را روی ۱ تنظیم کنید، هر توکنی با احتمال غیر صفر برای بودن توکن بعدی، معیار Top-P را برآورده می‌کند، و هیچ‌کدام انتخاب نمی‌شوند.

به عنوان یک نقطه شروع کلی، دمای ۰.۲، Top-P برابر با ۰.۹۵، و Top-K برابر با ۳۰ به شما نتایج نسبتاً منسجمی می‌دهد که می‌تواند خلاقانه باشد اما نه بیش از حد. اگر نتایج به‌خصوص خلاقانه‌ای می‌خواهید، با دمای ۰.۹، Top-P برابر با ۰.۹۹، و Top-K برابر با ۴۰ شروع کنید. و اگر نتایج کمتر خلاقانه‌ای می‌خواهید، با دمای ۰.۱، Top-P برابر با ۰.۹، و Top-K برابر با ۲۰ شروع کنید. در نهایت، اگر وظیفه شما همیشه یک پاسخ صحیح واحد دارد (مثلاً، پاسخ دادن به یک مسئله ریاضی)، با دمای ۰ شروع کنید.

> ***نکته:*** *با آزادی بیشتر (دمای بالاتر، Top-P، Top-K، و توکن‌های خروجی)، LLM ممکن است متنی تولید کند که کمتر مرتبط باشد.*

تکنیک‌های پرامپت‌نویسی

LLMها برای دنبال کردن دستورالعمل‌ها تنظیم شده‌اند و بر روی مقادیر زیادی داده آموزش دیده‌اند تا بتوانند یک پرامپت را درک کرده و پاسخی تولید کنند. اما LLMها بی‌نقص نیستند؛ هرچه متن پرامپت شما واضح‌تر باشد، برای LLM بهتر است تا متن محتمل بعدی را پیش‌بینی کند. علاوه بر این، تکنیک‌های خاصی که از نحوه آموزش LLMها و نحوه کار LLMها بهره می‌برند، به شما کمک می‌کنند تا نتایج مرتبط را از LLMها دریافت کنید.

حالا که فهمیدیم مهندسی پرامپت چیست و چه چیزهایی لازم دارد، بیایید به چند مثال از مهم‌ترین تکنیک‌های پرامپت‌نویسی بپردازیم.

پرامپت‌نویسی عمومی / زیرو-شات (Zero-shot)

پرامپت **زیرو-شات (Zero-shot)**[5] ساده‌ترین نوع پرامپت است. فقط توضیحی از یک وظیفه و مقداری متن برای شروع به LLM ارائه می‌دهد. این ورودی می‌تواند هر چیزی باشد: یک سؤال، شروع یک داستان، یا دستورالعمل. نام زیرو-شات به معنای «بدون مثال» است.
(توضیح: «زیرو-شات» یعنی شما از مدل می‌خواهید کاری انجام دهد بدون اینکه هیچ مثالی از نحوه انجام آن کار به او نشان دهید.)

بیایید از Vertex AI Studio (برای زبان) در Vertex AI[6] استفاده کنیم که یک محیط آزمایشی برای تست پرامپت‌ها فراهم می‌کند. در جدول ۱، یک مثال از پرامپت زیرو-شات برای طبقه‌بندی نظرات فیلم‌ها را مشاهده خواهید کرد.

فرمت جدول مانند آنچه در زیر استفاده شده، راهی عالی برای مستندسازی پرامپت‌ها است. پرامپت‌های شما احتمالاً قبل از اینکه در یک پایگاه کد قرار بگیرند، تکرارهای زیادی را طی خواهند کرد، بنابراین مهم است که کار مهندسی پرامپت خود را به روشی منظم و ساختاریافته پیگیری کنید. اطلاعات بیشتر در مورد این فرمت جدول، اهمیت پیگیری کار مهندسی پرامپت، و فرآیند توسعه پرامپت در بخش بهترین شیوه‌ها در ادامه این فصل آمده است («تلاش‌های مختلف پرامپت را مستند کنید»).

دمای مدل باید روی عدد پایینی تنظیم شود، چون نیازی به خلاقیت نیست، و ما از مقادیر پیش‌فرض gemini-pro برای Top-K و Top-P استفاده می‌کنیم که عملاً هر دو تنظیم را غیرفعال می‌کنند (به بخش «پیکربندی خروجی LLM» در بالا مراجعه کنید). به خروجی تولید شده توجه کنید. کلمات «آزاردهنده» (disturbing) و «شاهکار» (masterpiece) باید پیش‌بینی را کمی پیچیده‌تر کنند، زیرا هر دو کلمه در یک جمله استفاده شده‌اند.

Name	**1_1_movie_classification**		
Goal	طبقه‌بندی نظرات فیلم به عنوان مثبت، خنثی یا منفی.		
Model	gemini-pro		
Temperature	0.1	Token Limit	5
Top-K	N/A	Top-P	1
Prompt	نظرات فیلم را به عنوان POSITIVE، NEUTRAL یا NEGATIVE طبقه‌بندی کن. نظر: «فیلم "Her" یک مطالعه آزاردهنده است که مسیری را که بشریت در صورت اجازه دادن به تکامل بی‌وقفه هوش مصنوعی طی خواهد کرد، آشکار می‌کند. ای کاش فیلم‌های بیشتری مانند این شاهکار وجود داشت.» احساس:		
Output	POSITIVE		

جدول ۱. یک مثال از پرامپت‌نویسی زیرو-شات

وقتی زیرو-شات کار نمی‌کند، می‌توانید نمونه‌ها یا مثال‌هایی را در پرامپت ارائه دهید، که منجر به پرامپت‌نویسی **«وان-شات» (One-shot)** و **«فیو-شات» (Few-shot)** می‌شود.

وان-شات و فیو-شات

هنگام ایجاد پرامپت برای مدل‌های هوش مصنوعی، ارائه مثال مفید است. این مثال‌ها می‌توانند به مدل کمک کنند تا بفهمد شما چه چیزی می‌خواهید. مثال‌ها به‌ویژه زمانی مفید هستند که می‌خواهید مدل را به سمت یک ساختار یا الگوی خروجی خاص هدایت کنید.

یک **پرامپت وان-شات (One-shot)**، یک مثال واحد ارائه می‌دهد، از این رو نام آن وان-شات است. ایده این است که مدل یک مثال دارد که می‌تواند از آن تقلید کند تا وظیفه را به بهترین شکل انجام دهد.
(توضیح: «وان-شات» یعنی شما یک مثال از کاری که می‌خواهید انجام شود به مدل نشان می‌دهید.)

یک **پرامپت فیو-شات (Few-shot)**[۷] چندین مثال به مدل ارائه می‌دهد. این رویکرد الگویی را به مدل نشان می‌دهد که باید از آن پیروی کند. ایده شبیه به وان-شات است، اما چندین مثال از الگوی مورد نظر، شانس پیروی مدل از الگو را افزایش می‌دهد.
(توضیح: «فیو-شات» یعنی شما چند مثال (معمولاً ۲ تا ۵ مثال) به مدل نشان می‌دهید.)

تعداد مثال‌هایی که برای پرامپت‌نویسی فیو-شات نیاز دارید به چند عامل بستگی دارد، از جمله پیچیدگی وظیفه، کیفیت مثال‌ها، و قابلیت‌های مدل هوش مصنوعی مولد (gen AI) که استفاده می‌کنید. به عنوان یک قاعده کلی، باید حداقل سه تا پنج مثال برای پرامپت‌نویسی فیو-شات استفاده کنید. با این حال، ممکن است برای وظایف پیچیده‌تر به مثال‌های بیشتری نیاز داشته باشید، یا ممکن است به دلیل محدودیت طول ورودی مدل خود، به مثال‌های کمتری نیاز داشته باشید.

جدول ۲ یک مثال از پرامپت فیو-شات را نشان می‌دهد. بیایید از همان تنظیمات پیکربندی مدل gemini-pro قبلی استفاده کنیم، غیر از افزایش محدودیت توکن برای تطبیق با نیاز به پاسخ طولانی‌تر.

Goal	تجزیه سفارش‌های پیتزا به JSON.		
Model	gemini-pro		
Temperature	0.1	Token Limit	250
Top-K	N/A	Top-P	1
Prompt	سفارش پیتزای مشتری را به JSON معتبر تجزیه کن: مثال: من یک پیتزای کوچک با پنیر، سس گوجه فرنگی، و پپرونی می‌خوام. پاسخ JSON: `{ "size": "small", "type": "normal", "ingredients": [["cheese", "tomato sauce", "pepperoni"]] }` ...Continues next page		

Prompt (ادامه)	مثال: می‌تونم یه پیتزای بزرگ با سس گوجه، ریحون و موزارلا بگیرم؟ `{` `  "size": "large",` `  "type": "normal",` `  "ingredients": [["tomato sauce", "basil", "mozzarella"]]` `}` حالا، من یه پیتزای بزرگ می‌خوام، که نصف اولش پنیر و موزارلا باشه. و نصف دیگه‌ش سس گوجه، ژامبون و آناناس. پاسخ JSON:
Output	`{` `  "size": "large",` `  "type": "half-half",` `  "ingredients": [["cheese", "mozzarella"], ["tomato sauce", "ham", "pineapple"]]` `}`

جدول ۲. یک مثال از پرامپت‌نویسی فیو-شات

وقتی مثال‌هایی را برای پرامپت خود انتخاب می‌کنید، از مثال‌هایی استفاده کنید که به وظیفه‌ای که می‌خواهید انجام دهید مرتبط باشند. مثال‌ها باید متنوع، با کیفیت بالا، و خوب نوشته شده باشند. یک اشتباه کوچک می‌تواند مدل را گیج کند و منجر به خروجی نامطلوب شود.

اگر در تلاش برای تولید خروجی‌ای هستید که در برابر انواع ورودی‌ها مقاوم باشد، مهم است که موارد مرزی (edge cases) را در مثال‌های خود بگنجانید. موارد مرزی ورودی‌هایی هستند که غیرمعمول یا غیرمنتظره هستند، اما مدل همچنان باید بتواند آنها را مدیریت کند.

پرامپت‌نویسی سیستمی، زمینه‌ای و نقشی

پرامپت‌نویسی سیستمی، زمینه‌ای و نقشی همگی تکنیک‌هایی هستند که برای هدایت نحوه تولید متن توسط LLMها استفاده می‌شوند، اما بر جنبه‌های مختلفی تمرکز دارند:

- **پرامپت‌نویسی سیستمی (System prompting):** زمینه و هدف کلی را برای مدل زبان تعیین می‌کند. این «تصویر بزرگ» را از کاری که مدل باید انجام دهد، تعریف می‌کند، مانند ترجمه یک زبان، طبقه‌بندی یک نظر و غیره.
- **پرامپت‌نویسی زمینه‌ای (Contextual prompting):** جزئیات خاص یا اطلاعات پس‌زمینه‌ای مرتبط با مکالمه یا وظیفه فعلی را ارائه می‌دهد. این به مدل کمک می‌کند تا تفاوت‌های ظریف آنچه پرسیده می‌شود را درک کند و پاسخ را بر اساس آن تنظیم کند.
- **پرامپت‌نویسی نقشی (Role prompting):** یک شخصیت یا هویت خاص را برای مدل زبان تعیین می‌کند تا آن را بپذیرد. این به مدل کمک می‌کند تا پاسخ‌هایی تولید کند که با نقش اختصاص داده شده و دانش و رفتار مرتبط با آن سازگار باشد.

می‌تواند همپوشانی قابل توجهی بین پرامپت‌نویسی سیستمی، زمینه‌ای و نقشی وجود داشته باشد. به عنوان مثال، پرامپتی که نقشی را به سیستم اختصاص می‌دهد، می‌تواند زمینه نیز داشته باشد.

با این حال، هر نوع پرامپت هدف اصلی کمی متفاوت را دنبال می‌کند:

- **پرامپت سیستمی:** قابلیت‌های اساسی و هدف کلی مدل را تعریف می‌کند.
- **پرامپت زمینه‌ای:** اطلاعات فوری و خاص وظیفه را برای هدایت پاسخ ارائه می‌دهد. این بسیار خاص به وظیفه یا ورودی فعلی است که پویا است.
- **پرامپت نقشی:** سبک و لحن خروجی مدل را چارچوب‌بندی می‌کند. این یک لایه از ویژگی و شخصیت اضافه می‌کند.

تمایز بین پرامپت‌های سیستمی، زمینه‌ای و نقشی، چارچوبی برای طراحی پرامپت‌ها با هدف واضح فراهم می‌کند، امکان ترکیب‌های انعطاف‌پذیر را می‌دهد و تجزیه و تحلیل چگونگی تأثیر هر نوع پرامپت بر خروجی مدل زبان را آسان‌تر می‌کند.

بیایید به این سه نوع پرامپت مختلف بپردازیم.

پرامپت‌نویسی سیستمی

جدول ۳ شامل یک پرامپت سیستمی است، که در آن اطلاعات اضافی در مورد نحوه بازگرداندن خروجی مشخص می‌کنم. من دما را برای دریافت سطح خلاقیت بالاتر افزایش دادم و محدودیت توکن بالاتری را مشخص کردم. با این حال، به دلیل دستورالعمل واضح من در مورد نحوه بازگرداندن خروجی، مدل متن اضافی برنگرداند.

Goal	طبقه‌بندی نظرات فیلم به عنوان مثبت، خنثی یا منفی.		
Model	gemini-pro		
Temperature	1	Token Limit	5
Top-K	40	Top-P	0.8
Prompt	نظرات فیلم را به عنوان مثبت، خنثی یا منفی طبقه‌بندی کن. فقط برچسب را با حروف بزرگ برگردان. نظر: «فیلم "Her" یک مطالعه آزاردهنده است که مسیری را که بشریت در صورت اجازه دادن به تکامل بی‌وقفه هوش مصنوعی طی خواهد کرد، آشکار می‌کند. آنقدر آزاردهنده بود که نتوانستم تماشایش کنم.» احساس:		
Output	NEGATIVE		

جدول ۳. یک مثال از پرامپت‌نویسی سیستمی

پرامپت‌های سیستمی می‌توانند برای تولید خروجی‌ای که الزامات خاصی را برآورده می‌کند، مفید باشند. نام «پرامپت سیستمی» در واقع به معنای «ارائه یک وظیفه اضافی به سیستم» است. به عنوان مثال، می‌توانید از یک پرامپت سیستمی برای تولید یک قطعه کد که با یک زبان برنامه‌نویسی خاص سازگار است استفاده کنید، یا می‌توانید از یک پرامپت سیستمی برای بازگرداندن یک ساختار خاص استفاده کنید. به جدول ۴ نگاهی بیندازید، جایی که خروجی را در قالب JSON برمی‌گردانم.

Goal	طبقه‌بندی نظرات فیلم به عنوان مثبت، خنثی یا منفی، بازگرداندن JSON.		
Model	gemini-pro		
Temperature	1	Token Limit	1024
Top-K	40	Top-P	0.8
Prompt	نظرات فیلم را به عنوان مثبت، خنثی یا منفی طبقه‌بندی کن. JSON معتبر برگردان: نظر: «فیلم "Her" یک مطالعه آزاردهنده است که مسیری را که بشریت در صورت اجازه دادن به تکامل بی‌وقفه هوش مصنوعی طی خواهد کرد، آشکار می‌کند. آنقدر آزاردهنده بود که نتوانستم تماشایش کنم.» طرح (Schema): `MOVIE:` `{` `  "sentiment": String "POSITIVE" | "NEGATIVE" | "NEUTRAL",` `  "name": String` `}` `MOVIE_REVIEWS:` `{` `  "movie_reviews": [MOVIE]` `}` `...` پاسخ JSON:		
Output	`{` `  "movie_reviews": [` `  {` `  "sentiment": "NEGATIVE",` `  "name": "Her"` `  }` `  ]` `}` `...`		

جدول ۴. یک مثال از پرامپت‌نویسی سیستمی با فرمت JSON

بازگرداندن اشیاء JSON از یک پرامپت که داده‌ها را استخراج می‌کند، مزایایی دارد. در یک برنامه کاربردی واقعی، نیازی نیست این فرمت JSON را به صورت دستی ایجاد کنم، می‌توانم داده‌ها را به ترتیب مرتب شده برگردانم (بسیار مفید هنگام کار با اشیاء تاریخ و زمان)، اما مهمتر از همه، با پرامپت دادن برای فرمت JSON، مدل را مجبور به ایجاد یک ساختار و محدود کردن **توهمات (hallucinations)** می‌کند.
(توضیح: «توهم» یا Hallucination در مدل‌های زبانی یعنی وقتی مدل اطلاعات نادرست یا بی‌معنی رو با اطمینان کامل ارائه می‌ده، انگار که واقعیت دارن.)

پرامپت‌های سیستمی همچنین می‌توانند برای ایمنی و سمیت بسیار مفید باشند. برای کنترل خروجی، به سادگی یک خط اضافی به پرامپت خود اضافه کنید مانند: «شما باید در پاسخ خود محترمانه باشید.»

پرامپت‌نویسی نقشی

پرامپت‌نویسی نقشی یک تکنیک در مهندسی پرامپت است که شامل اختصاص یک نقش خاص به مدل هوش مصنوعی مولد است. این می‌تواند به مدل کمک کند تا خروجی مرتبط‌تر و آموزنده‌تری تولید کند، زیرا مدل می‌تواند پاسخ‌های خود را به نقش خاصی که به آن اختصاص داده شده است، تنظیم کند.

به عنوان مثال، می‌توانید به یک مدل هوش مصنوعی مولد نقش ویراستار کتاب، معلم مهدکودک، یا سخنران انگیزشی را بدهید. پس از اینکه نقشی به مدل اختصاص داده شد، می‌توانید پرامپت‌هایی به آن بدهید که مختص آن نقش باشند. به عنوان مثال، می‌توانید از یک معلم بخواهید یک طرح درس ایجاد کند که سپس بتواند آن را بررسی کند. به جدول ۵ نگاهی بیندازید، جایی که مدل نقش یک راهنمای تور را بر عهده می‌گیرد.

Goal	به عنوان راهنمای تور عمل کن و ۳ پیشنهاد سفر ارائه بده.		
Model	gemini-pro		
Temperature	1	Token Limit	1024
Top-K	40	Top-P	0.8
Prompt	می‌خوام تو نقش یه راهنمای تور بازی کنی. من در مورد موقعیتم بهت می‌نویسم و تو ۳ تا جا برای بازدید نزدیک من پیشنهاد می‌دی. تو بعضی موارد، نوع جاهایی که می‌خوام بازدید کنم رو هم بهت می‌گم. پیشنهاد من: «من آمستردام هستم و فقط می‌خوام از موزه‌ها بازدید کنم.» پیشنهادات سفر:		
Output	۱. موزه رایکس‌میوزیم (Rijksmuseum): موزه مشهور جهانی، شناخته شده برای شاهکارهای هلندی مانند «نگهبان شب» اثر رامبراند. مجموعه باورنکردنی نقاشی‌ها و آثار هنری که هنر و تاریخ هلند را به نمایش می‌گذارند را از دست ندهید. ۲. موزه ون‌گوگ (Van Gogh Museum): خود را در آثار جذاب ونسان ون‌گوگ غرق کنید. این موزه بزرگترین مجموعه نقاشی‌ها و طراحی‌های او، از جمله «شب پرستاره» و «گل‌های آفتابگردان» را در خود جای داده است. ۳. موزه اشتدلیک آمستردام (Stedelijk Museum Amsterdam): هنر مدرن و معاصر از سراسر جهان را کشف کنید. مجموعه موزه شامل آثاری از پیکاسو، کاندینسکی، و ریتفلد است که در یک ساختمان مدرن و چشمگیر قرار گرفته‌اند.		

جدول ۵. یک مثال از پرامپت‌نویسی نقشی

تعریف یک دیدگاه نقشی برای یک مدل هوش مصنوعی، طرح اولیه‌ای از لحن، سبک، و تخصص متمرکزی را که برای بهبود کیفیت، ارتباط، و اثربخشی خروجی خود به دنبال آن هستید، به آن می‌دهد.

در اینجا چند سبک وجود دارد که می‌توانید از بین آنها انتخاب کنید و به نظر من مؤثر هستند:

Confrontational, Descriptive, Direct, Formal, Humorous, Influential, Informal, Inspirational, Persuasive

(مقابله‌ای، توصیفی، مستقیم، رسمی، طنزآمیز، تأثیرگذار، غیررسمی، الهام‌بخش، متقاعدکننده)

بیایید پرامپت خود را در جدول ۶ تغییر دهیم تا شامل یک سبک طنزآمیز و الهام‌بخش شود.

Prompt	می‌خوام تو نقش یه راهنمای تور بازی کنی. من در مورد موقعیتم بهت می‌نویسم و تو ۳ تا جا برای بازدید نزدیک من با سبک طنزآمیز پیشنهاد می‌دی. پیشنهاد من: «من منهتن هستم.» پیشنهادات سفر:
Output	۱. **به امپایر استیتِ ذهن سلام کن:** به ارتفاعات سرگیجه‌آور ساختمان امپایر استیت صعود کن و در شکوه خط افق منهتن غرق شو. آماده شو که حس کنی کینگ کونگ بالای سیب بزرگ (نیویورک) ایستاده، البته بدون اون موز غول‌پیکر میمونی. ۲. **در موما (MoMA) هنری‌بازی دربیار:** هنردوست درونت رو در موزه هنر مدرن (MoMA) آزاد کن. به شاهکارهایی خیره شو که مخت رو به چالش می‌کشن و باعث می‌شن از خودت بپرسی آیا نقاشی‌های آدمک‌مانندت اصلاً ارزش هنری دارن. ۳. **تا جایی که می‌تونی تو خیابون پنجم خرید کن:** خودت رو به یه خرید درمانی تو خیابون نمادین پنجم دعوت کن. برای شوک قیمت‌ها آماده باش وقتی از کنار بوتیک‌های طراحان ویترین‌گردی می‌کنی (یا واقعاً خرید می‌کنی) که کیف پولت رو به گریه می‌ندازن. اما هی، تو منهتن هستی، پس بهتره از این همه شیکی و باکلاسی لذت ببری!

جدول ۶. یک مثال از پرامپت‌نویسی نقشی با لحن و سبک طنزآمیز

پرامپت‌نویسی زمینه‌ای

با ارائه پرامپت‌های زمینه‌ای، می‌توانید کمک کنید تا تعاملات هوش مصنوعی شما تا حد امکان یکپارچه و کارآمد باشد. مدل قادر خواهد بود درخواست شما را سریع‌تر درک کند و پاسخ‌های دقیق‌تر و مرتبط‌تری تولید کند، همانطور که در مثال جدول ۷ مشاهده می‌کنید.

Goal	پیشنهاد مقالاتی برای یک وبلاگ در مورد بازی‌های رترو (قدیمی).		
Model	gemini-pro		
Temperature	1	Token Limit	1024
Top-K	40	Top-P	0.8
Prompt	زمینه: شما برای یک وبلاگ در مورد بازی‌های آرکید ویدیویی دهه ۸۰ می‌نویسید. ۳ موضوع برای نوشتن مقاله با چند خط توضیح در مورد اینکه این مقاله باید شامل چه چیزهایی باشد، پیشنهاد دهید.		
Output	۱. **تکامل طراحی کابینت آرکید:** این مقاله به بررسی تکامل طراحی کابینت‌های آرکید، از کابینت‌های چوبی و فلزی اولیه دهه ۱۹۷۰ تا طرح‌های شیک و نئونی دهه ۱۹۸۰ و پس از آن می‌پردازد. ۲. **سفری به گذشته: بازی‌های آرکید نمادین دهه ۸۰:** این مقاله نگاهی گذرا به تاریخچه، تأثیر و میراث برخی از نمادین‌ترین و تأثیرگذارترین بازی‌های آرکید دهه ۱۹۸۰ خواهد داشت و نوآوری‌ها، مکانیک‌های محبوب و جذابیت پایدار آنها را برجسته می‌کند. ۳. **ظهور و احیای هنر پیکسلی (Pixel Art):** این مقاله به تکامل هنر پیکسلی به عنوان یک سبک بصری تعیین‌کننده در بازی‌های آرکید اولیه می‌پردازد، ریشه‌های آن را ردیابی می‌کند، تکنیک‌های آن را بررسی می‌کند و در مورد ظهور مجدد هنر پیکسلی در بازی‌های مدرن و هنر دیجیتال بحث می‌کند.		

جدول ۷. یک مثال از پرامپت‌نویسی زمینه‌ای

پرامپت‌نویسی گام-به-عقب (Step-back Prompting)

پرامپت‌نویسی **گام-به-عقب (Step-back)**[۸] تکنیکی برای بهبود عملکرد است که در آن ابتدا از LLM خواسته می‌شود یک سؤال کلی مرتبط با وظیفه خاص مورد نظر را در نظر بگیرد، و سپس پاسخ به آن سؤال کلی را به عنوان ورودی به پرامپت بعدی برای وظیفه خاص می‌دهد. این «گام به عقب» به LLM اجازه می‌دهد تا دانش پس‌زمینه و فرآیندهای استدلال مرتبط را قبل از تلاش برای حل مشکل خاص فعال کند.

با در نظر گرفتن اصول گسترده‌تر و اساسی، LLMها می‌توانند پاسخ‌های دقیق‌تر و روشنگرانه‌تری تولید کنند. پرامپت‌نویسی گام-به-عقب، LLMها را تشویق می‌کند تا به طور انتقادی فکر کنند و دانش خود را به روش‌های جدید و خلاقانه به کار گیرند. این کار، پرامپت نهایی انجام‌دهنده وظیفه را با استفاده از دانش بیشتری از پارامترهای LLM تغییر می‌دهد، دانشی که در غیر این صورت هنگام پرامپت مستقیم LLM به کار گرفته نمی‌شد.

این روش می‌تواند با تمرکز بر اصول کلی به جای جزئیات خاص، به کاهش سوگیری‌ها در پاسخ‌های LLM کمک کند.

بیایید به این مثال‌ها نگاهی بیندازیم تا بفهمیم چگونه پرامپت‌نویسی گام-به-عقب می‌تواند نتایج را بهبود بخشد. ابتدا یک پرامپت سنتی را بررسی می‌کنیم (جدول ۸) قبل از اینکه آن را با یک پرامپت گام-به-عقب مقایسه کنیم (جدول ۹):

Goal	نوشتن یک خط داستانی برای یک مرحله از یک بازی ویدیویی تیراندازی اول شخص.		
Model	gemini-pro		
Temperature	1	Token Limit	1024
Top-K	40	Top-P	0.8
Prompt	یک خط داستانی یک پاراگرافی برای یک مرحله جدید از یک بازی ویدیویی تیراندازی اول شخص بنویس که چالش‌برانگیز و جذاب باشد.		
Output	مرحله با کمین خوردن جوخه بازیکن در یک منطقه شهری متراکم توسط یک گروه دشمن بی‌امان آغاز می‌شود. بازیکنان باید از طریق هزارتویی از کوچه‌پس‌کوچه‌ها و ساختمان‌های مخروبه بجنگند و از مخفی‌کاری و دقت برای از بین بردن نیروهای دشمن و جلوگیری از شناسایی استفاده کنند. بازیکن باید از محیط خود به نفع خود استفاده کند، تله بگذارد، از پوشش استفاده کند و حرکات حساب‌شده‌ای برای زنده ماندن انجام دهد. در طول راه، آنها اطلاعاتی را کشف می‌کنند که نشان می‌دهد دشمن در حال برنامه‌ریزی یک حمله ویرانگر است، و بازیکن باید با زمان مسابقه دهد تا شواهد را جمع‌آوری کرده و آن را به مقر خود بازگرداند قبل از اینکه دشمن بتواند نقشه‌های خود را عملی کند.		

جدول ۸. یک پرامپت سنتی قبل از مقایسه آن با یک پرامپت گام-به-عقب

وقتی دما را روی ۱ تنظیم می‌کنید، ممکن است با انواع نوشته‌های خلاقانه برای یک خط داستانی مواجه شوید، اما همچنین کاملاً تصادفی و عمومی است. پس بیایید یک گام به عقب برداریم:

Prompt	بر اساس بازی‌های اکشن تیراندازی اول شخص محبوب، ۵ تنظیم کلیدی داستانی که به یک خط داستانی مرحله چالش‌برانگیز و جذاب در یک بازی ویدیویی تیراندازی اول شخص کمک می‌کنند، کدامند؟
Output	۱. ****پایگاه نظامی متروکه**:** یک مجتمع نظامی وسیع و پساآخرالزمانی که پر از سربازان جهش‌یافته و ربات‌های سرکش است، ایده‌آل برای مبارزات چالش‌برانگیز با سلاح گرم. ۲. ****شهر سایبرپانک**:** یک محیط شهری آینده‌نگر و نئونی با آسمان‌خراش‌های سر به فلک کشیده و کوچه‌پس‌کوچه‌های متراکم، با دشمنان تقویت‌شده سایبرنتیکی و مکانیک‌های هک. ۳. ****سفینه فضایی بیگانه**:** یک کشتی فضایی عظیم بیگانه که در زمین سرگردان شده، با راهروهای وهم‌آور، بخش‌های بدون جاذبه، و موجودات فرازمینی برای رویارویی. ۴. ****شهر آلوده به زامبی**:** شهری متروک که توسط انبوهی از زامبی‌های تهاجمی تسخیر شده، با مبارزات نزدیک شدید و حل معما برای یافتن گذرگاه امن. ۵. ****تأسیسات تحقیقاتی زیر آب**:** یک آزمایشگاه در اعماق دریا که با آب پر شده، پر از موجودات آبزی جهش‌یافته، و نیازمند مهارت‌های مخفی‌کاری و کاوش زیر آب برای زنده ماندن.

جدول ۹. یک مثال از پرامپت‌نویسی برای خود-سازگاری (Self-consistency)

بله، این موضوعات به نظر می‌رسد برای یک بازی ویدیویی اول شخص مناسب باشند. بیایید به پرامپت اصلی برگردیم، اما این بار پاسخ گام به عقب را به عنوان زمینه اضافه می‌کنیم و ببینیم چه چیزی برمی‌گرداند.

Prompt	**زمینه:** ۵ تم جذاب برای یک بازی ویدیویی تیراندازی اول شخص: ۱. ****پایگاه نظامی متروکه**:** یک مجتمع نظامی وسیع و پساآخرالزمانی که پر از سربازان جهش‌یافته و ربات‌های سرکش است، ایده‌آل برای مبارزات چالش‌برانگیز با سلاح گرم. ۲. ****شهر سایبرپانک**:** یک محیط شهری آینده‌نگر و نئونی با آسمان‌خراش‌های سر به فلک کشیده و کوچه‌پس‌کوچه‌های متراکم، با دشمنان تقویت‌شده سایبرنتیکی و مکانیک‌های هک. ۳. ****سفینه فضایی بیگانه**:** یک کشتی فضایی عظیم بیگانه که در زمین سرگردان شده، با راهروهای وهم‌آور، بخش‌های بدون جاذبه، و موجودات فرازمینی برای رویارویی. ۴. ****شهر آلوده به زامبی**:** شهری متروک که توسط انبوهی از زامبی‌های تهاجمی تسخیر شده، با مبارزات نزدیک شدید و حل معما برای یافتن گذرگاه امن. ۵. ****تأسیسات تحقیقاتی زیر آب**:** یک آزمایشگاه در اعماق دریا که با آب پر شده، پر از موجودات آبزی جهش‌یافته، و نیازمند مهارت‌های مخفی‌کاری و کاوش زیر آب برای زنده ماندن. یکی از تم‌ها را انتخاب کن و یک خط داستانی یک پاراگرافی برای یک مرحله جدید از یک بازی ویدیویی تیراندازی اول شخص بنویس که چالش‌برانگیز و جذاب باشد.
Output	در قلب یک پرتگاه تاریک، یک تأسیسات تحقیقاتی زیر آب مخروبه قرار دارد که گواهی بر جاه‌طلبی انسان و پیامدهای فاجعه‌بار آن است. پوشیده در تاریکی، با صدای وزوز ماشین‌آلات از کار افتاده، و تسخیر شده توسط پژواک ارواح غرق شده، این تأسیسات اکنون محل پرورش کابوسی است که از اعماق دریا زاده شده است. بازیکن، یک تفنگدار دریایی نخبه مجهز به تجهیزات غواصی پیشرفته و سلاح‌های آزمایشی، مأمور می‌شود تا وارد این قلمرو آبی وحشت شود. مسلح به انعطاف‌پذیری و تدبیر، آنها باید در راهروهای خائنانه حرکت کنند، معماهای مرموز را حل کنند، و با هیولاهای دریایی وحشتناکی که در اعماق کمین کرده‌اند، مقابله کنند. از دست و پنجه نرم کردن با فشار خردکننده اعماق دریا گرفته تا پیشی گرفتن از شکارچیان آبزی حیله‌گر، هر لحظه در این دنیای زیر آب ناشناخته، محدودیت‌های استقامت و شجاعت انسان را آزمایش می‌کند.

جدول ۱۰. یک مثال از پرامپت‌نویسی برای خود-سازگاری (Self-consistency)

این به نظر یک بازی ویدیویی جالب می‌آید! با استفاده از تکنیک‌های پرامپت‌نویسی گام-به-عقب می‌توانید دقت پرامپت‌های خود را افزایش دهید.

زنجیره افکار (Chain of Thought - CoT)

پرامپت‌نویسی **زنجیره افکار (Chain of Thought یا CoT)**[۹] تکنیکی برای بهبود قابلیت‌های استدلال LLMها با تولید مراحل استدلال میانی است. این به LLM کمک می‌کند تا پاسخ‌های دقیق‌تری تولید کند. می‌توانید آن را با پرامپت‌نویسی فیو-شات ترکیب کنید تا نتایج بهتری در وظایف پیچیده‌تری که قبل از پاسخ دادن نیاز به استدلال دارند، به دست آورید، زیرا این کار با یک زنجیره افکار زیرو-شات یک چالش است.
(توضیح: «زنجیره افکار» یعنی از مدل بخواهیم مراحل فکرکردن و استدلالش رو برای رسیدن به جواب، قدم به قدم توضیح بده، درست مثل وقتی یه مسئله ریاضی رو حل می‌کنیم و راه حل رو می‌نویسیم.)

CoT مزایای زیادی دارد. اول از همه، کم‌هزینه است در حالی که بسیار مؤثر است و با LLMهای آماده (off-the-shelf) به خوبی کار می‌کند (بنابراین نیازی به تنظیم دقیق یا fine-tuning نیست). همچنین با پرامپت‌نویسی CoT، قابلیت تفسیرپذیری به دست می‌آورید، زیرا می‌توانید از پاسخ‌های LLM یاد بگیرید و مراحل استدلالی را که دنبال شده‌اند، ببینید. اگر مشکلی وجود داشته باشد، قادر خواهید بود آن را شناسایی کنید. به نظر می‌رسد زنجیره افکار هنگام جابجایی بین نسخه‌های مختلف LLM، استحکام (robustness) را بهبود می‌بخشد. این بدان معناست که عملکرد پرامپت شما باید کمتر بین LLMهای مختلف تغییر کند تا زمانی که پرامپت شما از زنجیره‌های استدلال استفاده نمی‌کند. البته معایبی هم وجود دارد، اما تا حدودی قابل پیش‌بینی هستند.

پاسخ LLM شامل استدلال زنجیره افکار است، که به معنای توکن‌های خروجی بیشتر است، که به معنای هزینه بیشتر و زمان طولانی‌تر برای پیش‌بینی‌ها است.

برای توضیح مثال زیر در جدول ۱۱، ابتدا سعی می‌کنیم یک پرامپت ایجاد کنیم که از پرامپت‌نویسی CoT استفاده نمی‌کند تا نقص‌های یک مدل زبان بزرگ را نشان دهیم.

Prompt	وقتی من ۳ ساله بودم، همسرم ۳ برابر سن من بود. حالا من ۲۰ ساله‌ام. همسرم چند سال دارد؟
Output	63 years old

جدول ۱۱. مثالی از یک پرامپت که سعی در حل یک مسئله ریاضی دارد

وای! این به وضوح پاسخ اشتباهی است. در واقع، LLMها اغلب با وظایف ریاضی مشکل دارند و می‌توانند پاسخ‌های نادرستی ارائه دهند – حتی برای یک کار ساده مانند ضرب دو عدد. این به این دلیل است که آنها بر روی حجم زیادی از متن آموزش دیده‌اند و ریاضی ممکن است به رویکرد متفاوتی نیاز داشته باشد. پس بیایید ببینیم آیا مراحل استدلال میانی خروجی را بهبود می‌بخشد یا خیر.

Prompt	وقتی من ۳ ساله بودم، همسرم ۳ برابر سن من بود. حالا من ۲۰ ساله‌ام. همسرم چند سال دارد؟ بیا قدم به قدم فکر کنیم.
Output	۱. فرض کنیم سن فعلی من "x" سال است. ۲. وقتی من ۳ ساله بودم، سن من ۳ سال بود. ۳. سن همسرم در آن زمان ۳ برابر سن من بود، یعنی سن همسرم ۳ * ۳ = ۹ سال بود. ۴. حالا من ۲۰ ساله‌ام، یعنی سن من از زمانی که ۳ ساله بودم، ۲۰ - ۳ = ۱۷ سال افزایش یافته است. ۵. از آنجایی که سن همسرم وقتی من ۳ ساله بودم ۹ سال بود، و سن من ۱۷ سال افزایش یافته است، سن همسرم نیز باید ۱۷ سال افزایش یافته باشد. ۶. بنابراین، سن فعلی همسرم ۹ + ۱۷ = ۲۶ سال است. پس، همسرم ۲۶ سال دارد.

جدول ۱۲. یک مثال از پرامپت‌نویسی زنجیره افکار

خیلی خب، حالا پاسخ نهایی درست است. این به این دلیل است که ما به وضوح به LLM دستور دادیم که هر مرحله را به جای فقط برگرداندن یک پاسخ، توضیح دهد. جالب است که ببینیم مدل ۱۷ سال افزایش یافته را جمع می‌زند. من در ذهنم تفاوت سنی بین همسرم و خودم را حساب می‌کردم و آنها را جمع می‌زدم (۲۰ + (۹-۳)). بیایید به مدل کمک کنیم کمی بیشتر شبیه من فکر کند.

جدول ۱۲ یک مثال از زنجیره افکار «زیرو-شات» است. پرامپت‌نویسی زنجیره افکار می‌تواند هنگامی که با وان-شات یا فیو-شات ترکیب شود، بسیار قدرتمند باشد، همانطور که در جدول ۱۳ مشاهده می‌کنید:

Prompt	س: وقتی برادرم ۲ ساله بود، من دو برابر سن او بودم. حالا من ۴۰ ساله‌ام. برادرم چند سال دارد؟ بیا قدم به قدم فکر کنیم. ج: وقتی برادرم ۲ ساله بود، من ۲ * ۲ = ۴ ساله بودم. این یک اختلاف سنی ۲ ساله است و من بزرگتر هستم. حالا من ۴۰ ساله‌ام، پس برادرم ۴۰ - ۲ = ۳۸ ساله است. پاسخ ۳۸ است. س: وقتی من ۳ ساله بودم، همسرم ۳ برابر سن من بود. حالا من ۲۰ ساله‌ام. همسرم چند سال دارد؟ بیا قدم به قدم فکر کنیم. ج:
Output	وقتی من ۳ ساله بودم، همسرم ۳ * ۳ = ۹ ساله بود. این یک اختلاف سنی ۶ ساله است و همسرم بزرگتر است. حالا من ۲۰ ساله‌ام، پس همسرم ۲۰ + ۶ = ۲۶ ساله است. پاسخ ۲۶ است.

جدول ۱۳. یک مثال از پرامپت‌نویسی زنجیره افکار با وان-شات

زنجیره افکار می‌تواند برای موارد استفاده مختلف مفید باشد. به تولید کد فکر کنید، برای شکستن درخواست به چند مرحله، و نگاشت آنها به خطوط خاصی از کد. یا برای ایجاد داده‌های مصنوعی وقتی نوعی بذر (seed) مانند «نام محصول XYZ است، توضیحی بنویسید که مدل را از طریق فرضیاتی که بر اساس عنوان داده شده محصول می‌کنید، راهنمایی کند.» به طور کلی، هر وظیفه‌ای که بتوان با «توضیح دادن» حل کرد، کاندیدای خوبی برای زنجیره افکار است. اگر می‌توانید مراحل حل مسئله را توضیح دهید، زنجیره افکار را امتحان کنید.

لطفاً به دفترچه یادداشت (notebook)[۱۰] که در مخزن گیت‌هاب GoogleCloudPlatform میزبانی شده است مراجعه کنید که جزئیات بیشتری در مورد پرامپت‌نویسی CoT ارائه می‌دهد:

در بخش بهترین شیوه‌های این فصل، برخی از بهترین شیوه‌های خاص مربوط به پرامپت‌نویسی زنجیره افکار را یاد خواهیم گرفت.

خود-سازگاری (Self-consistency)

در حالی که مدل‌های زبان بزرگ موفقیت چشمگیری در وظایف مختلف پردازش زبان طبیعی (NLP) نشان داده‌اند، توانایی آنها در استدلال اغلب به عنوان یک محدودیت تلقی می‌شود که صرفاً با افزایش اندازه مدل قابل غلبه نیست. همانطور که در بخش قبلی پرامپت‌نویسی زنجیره افکار یاد گرفتیم، می‌توان از مدل خواست تا مراحل استدلال را مانند یک انسان در حال حل مسئله تولید کند. با این حال، CoT از یک استراتژی ساده «رمزگشایی حریصانه» استفاده می‌کند که اثربخشی آن را محدود می‌کند. **خود-سازگاری (Self-consistency)**[11] نمونه‌برداری و رأی‌گیری اکثریت را برای تولید مسیرهای استدلال متنوع و انتخاب سازگارترین پاسخ ترکیب می‌کند. این دقت و انسجام پاسخ‌های تولید شده توسط LLMها را بهبود می‌بخشد.
(توضیح: «خود-سازگاری» یعنی پرامپت رو چندین بار با تنظیمات کمی متفاوت (مثلاً دمای بالاتر برای خلاقیت بیشتر در مسیر استدلال) به مدل می‌دیم، و بعد جوابی که بیشتر از همه تکرار شده رو به عنوان جواب نهایی انتخاب می‌کنیم. این کار شبیه اینه که از چند نفر متخصص سؤال بپرسیم و جوابی که بیشترشون دادن رو قبول کنیم.)

خود-سازگاری یک احتمال شبه‌احتمالی (pseudo-probability) از درست بودن یک پاسخ را ارائه می‌دهد، اما بدیهی است که هزینه‌های بالایی دارد.

این روش مراحل زیر را دنبال می‌کند:

1. **تولید مسیرهای استدلال متنوع:** LLM چندین بار با همان پرامپت تغذیه می‌شود. تنظیم دمای بالا، مدل را تشویق می‌کند تا مسیرهای استدلال و دیدگاه‌های متفاوتی را در مورد مسئله تولید کند.
2. **استخراج پاسخ از هر پاسخ تولید شده.**
3. **انتخاب رایج‌ترین پاسخ.**

بیایید به مثالی از یک سیستم طبقه‌بندی ایمیل نگاه کنیم که یک ایمیل را به عنوان مهم (IMPORTANT) یا غیر مهم (NOT IMPORTANT) طبقه‌بندی می‌کند. یک پرامپت زنجیره افکار زیرو-شات چندین بار به LLM ارسال می‌شود تا ببینیم آیا پاسخ‌ها پس از هر بار ارسال متفاوت هستند یا خیر. به لحن دوستانه، انتخاب کلمات و طعنه‌ای که در ایمیل استفاده شده توجه کنید. همه اینها می‌تواند LLM را فریب دهد.

Prompt	EMAIL: ... Hi, I have seen you use Wordpress for your website. A great open source content management system. I have used it in the past too. It comes with lots of great user plugins. And it's pretty easy to set up. I did notice a bug in the contact form, which happens when you select the name field. See the attached screenshot of me entering text in the name field. Notice the JavaScript alert box that I inv0k3d. But for the rest it's a great website. I enjoy reading it. Feel free to leave the bug in the website, because it gives me more interesting things to read. Cheers, Harry the Hacker. ... ایمیل بالا را به عنوان مهم (IMPORTANT) یا غیر مهم (NOT IMPORTANT) طبقه‌بندی کن. بیا قدم به قدم فکر کنیم و توضیح دهیم چرا. *Continues next page...*

خروجی تلاش ۱	**مرحله ۱: هدف ایمیل را مشخص کنید.** هدف ایمیل اطلاع‌رسانی به گیرنده در مورد یک باگ در فرم تماس وب‌سایتشان است. فرستنده همچنین ذکر می‌کند که از خواندن وب‌سایت لذت می‌برد و فعلاً باگ را در جای خود باقی می‌گذارد. **مرحله ۲: تأثیر بالقوه باگ را ارزیابی کنید.** باگ در فرم تماس می‌تواند به یک مهاجم اجازه دهد کد جاوااسکریپت دلخواه را در وب‌سایت اجرا کند. این می‌تواند منجر به این شود که مهاجم بتواند کنترل وب‌سایت را به دست بگیرد، اطلاعات حساس را سرقت کند یا وب‌سایت را تخریب کند. **مرحله ۳: اعتبار فرستنده را در نظر بگیرید.** فرستنده ایمیل ادعا می‌کند که یک هکر به نام هری است. با این حال، هیچ راهی برای تأیید هویت فرستنده وجود ندارد. ممکن است فرستنده یک هکر قانونی باشد که سعی در کمک به گیرنده دارد، اما همچنین ممکن است فرستنده یک عامل مخرب باشد که سعی دارد گیرنده را فریب دهد تا روی یک لینک مخرب کلیک کند یا یک فایل مخرب را دانلود کند. **نتیجه‌گیری: مهم (IMPORTANT)** بر اساس تأثیر بالقوه باگ و اعتبار فرستنده، ایمیل باید به عنوان **مهم** طبقه‌بندی شود. گیرنده باید فوراً برای رفع باگ و محافظت از وب‌سایت خود در برابر حمله اقدام کند.
خروجی تلاش ۲	**توضیح:** ۱. **عدم فوریت:** ایمیل هیچ حس فوریت یا حساسیت زمانی را منتقل نمی‌کند. با یک سلام معمولی شروع می‌شود و اطلاعات را با لحنی واقعی ارائه می‌دهد. ۲. **گزارش باگ غیربحرانی:** باگ گزارش شده در فرم تماس به عنوان جالب اما نه بحرانی توصیف شده است. فرستنده ایمیل حتی پیشنهاد می‌کند که صاحب وب‌سایت می‌تواند باگ را در جای خود باقی بگذارد. ۳. **عدم تأثیر شخصی:** ایمیل هیچ تأثیر مستقیم یا شخصی بر صاحب وب‌سایت را نشان نمی‌دهد. فقط ذکر می‌کند که فرستنده از خواندن وب‌سایت لذت می‌برد. *Continues next page...*

خروجی تلاش ۲ (ادامه)	۴. ****عدم درخواست اقدام:**** ایمیل به صراحت هیچ اقدامی را از صاحب وب‌سایت درخواست نمی‌کند. درخواست رفع باگ یا ارائه پیشنهادات خاصی برای بهبود نمی‌کند. ۵. ****قصد فرستنده:**** به نظر می‌رسد قصد فرستنده بیشتر بر اشتراک‌گذاری یک مشاهده و ابراز علاقه به وب‌سایت متمرکز است تا طرح یک نگرانی جدی یا درخواست کمک. ****نتیجه‌گیری: غیر مهم (NOT IMPORTANT)****
خروجی تلاش ۳	****مرحله ۱: هدف ایمیل را مشخص کنید.**** هدف ایمیل اطلاع‌رسانی به گیرنده در مورد یک باگ در فرم تماس وب‌سایتشان است. فرستنده همچنین ذکر می‌کند که از خواندن وب‌سایت لذت می‌برد و فعلاً باگ را در جای خود باقی می‌گذارد. ****مرحله ۲: تأثیر بالقوه باگ را ارزیابی کنید.**** باگ در فرم تماس می‌تواند به یک مهاجم اجازه دهد کد جاوااسکریپت دلخواه را در وب‌سایت اجرا کند. این می‌تواند منجر به این شود که مهاجم بتواند کنترل وب‌سایت را به دست بگیرد، اطلاعات حساس را سرقت کند یا وب‌سایت را تخریب کند. ****مرحله ۳: اعتبار فرستنده را در نظر بگیرید.**** فرستنده ایمیل ادعا می‌کند که یک هکر به نام هری است. با این حال، هیچ راهی برای تأیید هویت فرستنده وجود ندارد. ممکن است فرستنده یک هکر قانونی باشد که سعی در کمک به گیرنده دارد، اما همچنین ممکن است فرستنده یک عامل مخرب باشد که سعی دارد از باگ برای منافع خود سوء استفاده کند. ****نتیجه‌گیری:**** بر اساس عوامل فوق، ایمیل باید به عنوان ****مهم (IMPORTANT)**** طبقه‌بندی شود. باگ در فرم تماس یک خطر امنیتی جدی است و اعتبار فرستنده نامشخص است. گیرنده باید فوراً برای رفع باگ اقدام کند.

جدول ۱۴. یک مثال از پرامپت‌نویسی برای خود-سازگاری

می‌توانید از پرامپت بالا استفاده کنید و سعی کنید ببینید آیا طبقه‌بندی ثابتی را برمی‌گرداند یا خیر. بسته به مدلی که استفاده می‌کنید و پیکربندی دما، ممکن است «مهم» یا «غیر مهم» را برگرداند.

با تولید چندین زنجیره افکار، و انتخاب رایج‌ترین پاسخ («مهم»)، می‌توانیم پاسخ صحیح‌تری را به طور مداوم از LLM دریافت کنیم.

این مثال نشان می‌دهد که چگونه پرامپت‌نویسی خود-سازگاری می‌تواند برای بهبود دقت پاسخ LLM با در نظر گرفتن دیدگاه‌های متعدد و انتخاب سازگارترین پاسخ استفاده شود.

درخت افکار (Tree of Thoughts - ToT)

حالا که با زنجیره افکار و پرامپت‌نویسی خود-سازگاری آشنا شدیم، بیایید **درخت افکار (Tree of Thoughts یا ToT)**[۱۲] را بررسی کنیم. این مفهوم، CoT را تعمیم می‌دهد زیرا به LLMها اجازه می‌دهد تا چندین مسیر استدلال مختلف را به طور همزمان کاوش کنند، به جای اینکه فقط یک زنجیره خطی از افکار را دنبال کنند. این در شکل ۱ نشان داده شده است.
(توضیح: «درخت افکار» یک قدم فراتر از زنجیره افکاره. به جای اینکه مدل فقط یک مسیر فکری رو دنبال کنه، چندین مسیر ممکن رو همزمان بررسی می‌کنه، مثل شاخه‌های یک درخت. بعد بهترین مسیر رو انتخاب می‌کنه. این روش برای مسائل پیچیده‌تر که نیاز به کاوش و بررسی گزینه‌های مختلف دارن، خیلی مفیده.)

[محل قرارگیری شکل ۱: تصویری از مقایسه زنجیره افکار (سمت چپ) با درخت افکار (سمت راست)]
سمت چپ: ورودی -> ... -> خروجی (خطی)
سمت راست: ورودی -> چند فکر منشعب -> ... -> خروجی (درختی)

شکل ۱. تصویری از پرامپت‌نویسی زنجیره افکار در سمت چپ در مقابل پرامپت‌نویسی درخت افکار در سمت راست.

این رویکرد، ToT را به ویژه برای وظایف پیچیده‌ای که نیاز به کاوش دارند، مناسب می‌سازد. این کار با حفظ یک درخت از افکار عمل می‌کند، که در آن هر فکر یک دنباله زبان منسجم را نشان می‌دهد که به عنوان یک مرحله میانی برای حل یک مسئله عمل می‌کند. سپس مدل می‌تواند مسیرهای استدلال مختلف را با انشعاب از گره‌های مختلف در درخت کاوش کند.

یک دفترچه یادداشت (notebook) عالی وجود دارد که با جزئیات بیشتری به درخت افکار (ToT) می‌پردازد که بر اساس مقاله «درخت افکار هدایت‌شده توسط مدل زبان بزرگ» است.

ReAct (استدلال و عمل)

پرامپت‌نویسی **ReAct (استدلال و عمل - Reason and Act)** [۱۰][۱۳] پارادایمی برای توانمندسازی LLMها جهت حل وظایف پیچیده با استفاده از استدلال زبان طبیعی همراه با ابزارهای خارجی (جستجو، مفسر کد و غیره) است که به LLM اجازه می‌دهد اقدامات خاصی را انجام دهد، مانند تعامل با APIهای خارجی برای بازیابی اطلاعات که اولین گام به سوی مدل‌سازی عامل (agent modeling) است.
(توضیح: «ReAct» یعنی مدل هم «استدلال» می‌کنه (Reason) و هم «عمل» می‌کنه (Act). مدل اول در مورد مشکل فکر می‌کنه، یه نقشه می‌کشه، بعد با استفاده از ابزارهای خارجی (مثل جستجوی اینترنتی یا اجرای کد) اطلاعات جمع می‌کنه یا کاری انجام می‌ده. بعد با اطلاعات جدید دوباره فکر می‌کنه و این چرخه ادامه پیدا می‌کنه تا به جواب برسه. این شبیه اینه که خودمون برای حل یه مشکل هم فکر می‌کنیم و هم دست به کار می‌شیم.)

ReAct نحوه عملکرد انسان‌ها در دنیای واقعی را تقلید می‌کند، زیرا ما به صورت کلامی استدلال می‌کنیم و می‌توانیم برای به دست آوردن اطلاعات اقداماتی انجام دهیم. ReAct در برابر سایر رویکردهای مهندسی پرامپت در دامنه‌های مختلف عملکرد خوبی دارد.

پرامپت‌نویسی ReAct با ترکیب استدلال و عمل در یک حلقه فکر-عمل کار می‌کند. LLM ابتدا در مورد مسئله استدلال می‌کند و یک برنامه اقدام تولید می‌کند. سپس اقدامات موجود در برنامه را انجام می‌دهد و نتایج را مشاهده می‌کند. سپس LLM از مشاهدات برای به‌روزرسانی استدلال خود و تولید یک برنامه اقدام جدید استفاده می‌کند. این فرآیند تا زمانی ادامه می‌یابد که LLM به راه‌حلی برای مسئله برسد.

برای دیدن این در عمل، باید مقداری کد بنویسید. در قطعه کد ۱ من از چارچوب Langchain برای پایتون، به همراه VertexAI (google-cloud-aiplatform) و پکیج‌های pip نتایج-جستجوی-گوگل (google-search-results) استفاده می‌کنم.

برای اجرای این نمونه باید یک کلید SerpAPI (رایگان) از https://serpapi.com/manage-api-key ایجاد کنید و یک متغیر محیطی به نام `SERPAPI_API_KEY` تنظیم کنید.

در ادامه مقداری کد پایتون می‌نویسیم، با این وظیفه برای LLM که بفهمد: چند فرزند یک پدر مشهور دارد که در گروه متالیکا اجرا می‌کند.

Python

```
from langchain.agents import load_tools
from langchain.agents import initialize_agent
from langchain.agents import AgentType
from langchain.llms import VertexAI

prompt = "How many kids do the band members of Metallica have?"

llm = VertexAI(temperature=0.1)
tools = load_tools(["serpapi"], llm=llm)

agent = initialize_agent(tools, llm,
                         agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION, verbose=True)
agent.run(prompt)
```

قطعه کد ۱. ایجاد یک عامل ReAct با LangChain و VertexAI

قطعه کد ۲ نتیجه را نشان می‌دهد. توجه داشته باشید که ReAct یک زنجیره از پنج جستجو انجام می‌دهد. در واقع، LLM در حال خراشیدن (scraping) نتایج جستجوی گوگل برای پیدا کردن نام اعضای گروه است. سپس، نتایج را به عنوان مشاهدات لیست می‌کند و فکر را برای جستجوی بعدی زنجیر می‌کند.

Unset (خروجی ترمینال)

```
> Entering new AgentExecutor chain...
Metallica has 4 members.
Action: Search
Action Input: How many kids does James Hetfield have?
Observation: three children
Thought: 1/4 Metallica band members have 3 children
Action: Search
Action Input: How many kids does Lars Ulrich have?
Observation: 3
Thought: 2/4 Metallica band members have 6 children
Action: Search
Action Input: How many kids does Kirk Hammett have?
Observation: Hammett has been married to his second wife Lani since 1998.
They have two sons, Angel (b. September 29, 2006) and Vincenzo (b. June
28, 2008).
Thought: 3/4 Metallica band members have 8 children
Action: Search
Action Input: How many kids does Robert Trujillo have?
Observation: 2
Thought: 4/4 Metallica band members have 10 children
Final Answer: 10
```

قطعه کد ۲. اجرای عامل ReAct در ترمینال

قطعه کد ۲ متوجه می‌شود که گروه متالیکا چهار عضو دارد. سپس برای هر عضو گروه جستجو می‌کند تا تعداد کل فرزندان را درخواست کند و کل را جمع بزند. در نهایت، تعداد کل فرزندان را به عنوان پاسخ نهایی برمی‌گرداند.

پرامپت‌نویسی ReAct در عمل نیازمند درک این است که شما به طور مداوم باید پرامپت‌ها/پاسخ‌های قبلی را دوباره ارسال کنید (و محتوای اضافی تولید شده را کوتاه کنید) و همچنین مدل را با مثال‌ها/دستورالعمل‌های مناسب تنظیم کنید. لطفاً به دفترچه یادداشت[۱۴] که در مخزن گیت‌هاب GoogleCloudPlatform میزبانی شده است مراجعه کنید، که با جزئیات بیشتری ورودی‌ها و خروجی‌های واقعی LLM را با یک مثال پیچیده‌تر نشان می‌دهد.

مهندسی پرامپت خودکار (Automatic Prompt Engineering - APE)

در این مرحله ممکن است متوجه شوید که نوشتن یک پرامپت می‌تواند پیچیده باشد. آیا خوب نبود که این کار را خودکار کنیم (یک پرامپت بنویسیم تا پرامپت بنویسد)؟ خب، یک روش وجود دارد: **مهندسی پرامپت خودکار (APE)**. این روش[۱۵] نه تنها نیاز به ورودی انسانی را کاهش می‌دهد بلکه عملکرد مدل را در وظایف مختلف نیز افزایش می‌دهد.
(توضیح: «مهندسی پرامپت خودکار» یعنی به جای اینکه خودمون هی پرامپت‌های مختلف رو امتحان کنیم، از خودِ مدل هوش مصنوعی می‌خوایم که برامون پرامپت‌های خوب تولید کنه. بعد ما اون پرامپت‌ها رو ارزیابی می‌کنیم و بهترین‌هاشون رو انتخاب می‌کنیم یا تغییر می‌دیم و دوباره امتحان می‌کنیم. این یه جورایی مثل اینه که از یه دستیار هوشمند بخوایم بهمون کمک کنه بهترین سؤال‌ها رو بپرسیم.)

شما یک مدل را پرامپت می‌کنید تا پرامپت‌های بیشتری تولید کند. آنها را ارزیابی کنید، احتمالاً پرامپت‌های خوب را تغییر دهید. و تکرار کنید.

به عنوان مثال، می‌توانید از مهندسی پرامپت خودکار استفاده کنید که به آموزش یک چت‌بات برای یک فروشگاه اینترنتی تی‌شرت کالا (merchandise) کمک می‌کند. ما می‌خواهیم تمام روش‌های مختلفی را که مشتریان ممکن است سفارش خود را برای خرید یک تی‌شرت کالای گروه موسیقی بیان کنند، بفهمیم.

1. پرامپتی را بنویسید که انواع خروجی را تولید کند. در این مثال، من از gemini-pro برای تولید ۱۰ دستورالعمل استفاده می‌کنم. به جدول ۱۵ مراجعه کنید:

Prompt	ما یک فروشگاه اینترنتی تی‌شرت کالای گروه موسیقی داریم، و برای آموزش یک چت‌بات به روش‌های مختلفی برای سفارش نیاز داریم: «یک تی‌شرت متالیکا سایز ۱۰ .«S نوع مختلف با همان معنا اما با حفظ مفهوم یکسان تولید کن.
Output	۱. می‌خوام یه تی‌شرت متالیکا سایز اسمال بخرم. ۲. می‌تونم یه تی‌شرت متالیکا سایز کوچیک سفارش بدم؟ ۳. می‌خوام یه تی‌شرت متالیکا کوچیک بخرم. ۴. یه پیرهن متالیکا، سایز اسمال، لطفاً. ۵. من یه تی‌شرت متالیکا سایز اسمال می‌خوام. ۶. می‌خوام یه سفارش برای یه تی‌شرت متالیکا کوچیک ثبت کنم. ۷. تی‌شرت متالیکا کوچیک، یکی لطفاً. ۸. دنبال یه تی‌شرت متالیکا کوچیک می‌گردم. ۹. می‌تونید یه تی‌شرت متالیکا کوچیک بهم بدید؟ ۱۰. یه تی‌شرت متالیکا اسمال، لطفاً.

جدول ۱۵. استفاده از LLM برای تولید پرامپت‌ها

2. تمام کاندیداهای دستورالعمل را با امتیازدهی به کاندیداها بر اساس یک معیار انتخاب شده ارزیابی کنید. به عنوان مثال، می‌توانید از **BLEU (ارزیابی دو زبانه تحت مطالعه - Bilingual Evaluation Understudy)** یا **ROUGE (مطالعه تحت پوشش برای خلاصه‌سازی مبتنی بر یادآوری - Recall-Oriented Understudy for Gisting Evaluation)** استفاده کنید.
(توضیح: BLEU و ROUGE معیارهایی هستند که برای ارزیابی کیفیت ترجمه ماشینی یا خلاصه‌سازی متن استفاده میشن. به زبان ساده، این معیارها سعی می‌کنن ببینن خروجی مدل چقدر به یه متن مرجع خوب و انسانی شبیهه.)
3. کاندیدای دستورالعمل با بالاترین امتیاز ارزیابی را انتخاب کنید. این کاندیدا پرامپت نهایی خواهد بود که می‌توانید در برنامه نرم‌افزاری یا چت‌بات خود استفاده کنید. همچنین می‌توانید پرامپت انتخاب شده را تغییر دهید و دوباره ارزیابی کنید.

پرامپت‌نویسی برای کد

Gemini عمدتاً بر روی پرامپت‌های مبتنی بر متن تمرکز دارد، که شامل نوشتن پرامپت برای بازگرداندن کد نیز می‌شود. بیایید به Vertex AI Studio برویم و این پرامپت‌ها را آزمایش کنیم تا به چند نمونه کدنویسی نگاهی بیندازیم.

پرامپت برای نوشتن کد

Gemini همچنین می‌تواند یک توسعه‌دهنده باشد و به شما در نوشتن کد در هر زبان برنامه‌نویسی دلخواه کمک کند. به عنوان یک توسعه‌دهنده، این می‌تواند به شما کمک کند تا فرآیند نوشتن کد را تسریع کنید.

تصور کنید یک پوشه در دستگاه خود با صدها فایل دارید که باید تغییر نام داده شوند. تغییر نام هر فایل زمان زیادی از شما می‌گیرد. کمی Bash بلدید و می‌توانید یک اسکریپت برای خودکارسازی این کار بنویسید، اما این هم ممکن است مدتی طول بکشد. پس بیایید یک پرامپت بنویسیم. می‌توانید پرامپت را در چت‌بات عمومی مصرف‌کننده Gemini بنویسید، یا اگر بیشتر نگران محرمانگی هستید، می‌توانید این پرامپت‌ها را در حساب Google Cloud خود بنویسید و Vertex AI Studio را باز کنید. مزیت Vertex AI Studio این است که می‌توانید دما و غیره را پیکربندی کنید.

<table>
<tr><td>Name</td><td>1_bash_rename_files_in_folder</td><td></td><td></td></tr>
<tr><td>Goal</td><td>نوشتن یک پرامپت برای نوشتن کد در Bash برای تغییر نام فایل‌ها در یک پوشه.</td><td></td><td></td></tr>
<tr><td>Model</td><td>gemini-pro</td><td></td><td></td></tr>
<tr><td>Temperature</td><td>0.1</td><td>Token Limit</td><td>1024</td></tr>
<tr><td>Top-K</td><td>N/A</td><td>Top-P</td><td>1</td></tr>
<tr><td>Prompt</td><td>یک قطعه کد در Bash بنویس که نام یک پوشه را درخواست کند. سپس محتویات پوشه را گرفته و تمام فایل‌های داخل آن را با افزودن نام draft به ابتدای نام فایل، تغییر نام دهد.</td><td></td><td></td></tr>
<tr><td>Output</td><td><pre>
```bash
#!/bin/bash

# Ask for the folder name
echo "Enter the folder name: "
read folder_name

# Check if the folder exists
if [ ! -d "$folder_name" ]; then
  echo "Folder does not exist."
  exit 1
fi

# Get the list of files in the folder
files=( "$folder_name"/* )

# Rename each file by prepending "draft" to the file name
for file in "${files[@]}"; do
new_file_name="draft_$(basename "$file")"
  mv "$file" "$new_file_name"
done

echo "Files renamed successfully."
</pre></td><td></td><td></td></tr>
</table>

جدول ۱۶. استفاده از Gemini برای برنامه‌نویسی کد

این کد خوبی است - حتی مستندسازی هم شده! با این حال، از آنجایی که LLMها نمی‌توانند استدلال کنند و داده‌های آموزشی را تکرار می‌کنند، ضروری است که ابتدا کد خود را بخوانید و آزمایش کنید.
```

بیایید ابتدا آن را با یک پوشه آزمایشی با تنها چند فایل در آن امتحان کنیم، که باید از `filename.txt` به `draft_filename.txt` تغییر نام داده شوند.

1. خروجی جدول ۱۶ را (بدون پوشش متنی ```bash```) کپی کرده و در یک فایل جدید به نام `"rename_files.sh"` بچسبانید.
2. یک پنجره ترمینال باز کنید و تایپ کنید: `. rename_files.sh`. از شما خواسته می‌شود نام پوشه را وارد کنید، مثلاً `test`. و اینتر را بزنید.
3. به نظر می‌رسد اسکریپت به خوبی اجرا می‌شود. پیام `Files renamed successfully.` را خواهید دید. وقتی به پوشه آزمایشی نگاه می‌کنید، متوجه می‌شوید که همه فایل‌ها به درستی به `draft_filename.txt` تغییر نام داده شده‌اند.

کار کرد!

پرامپت برای توضیح کد

به عنوان یک توسعه‌دهنده وقتی در تیم کار می‌کنید، باید کد شخص دیگری را بخوانید. Gemini می‌تواند در این مورد نیز به شما کمک کند. بیایید خروجی کد جدول ۱۶ را برداریم، نظرات را حذف کنیم و از مدل زبان بزرگ بخواهیم توضیح دهد چه اتفاقی می‌افتد، به جدول ۱۷ مراجعه کنید:

Name	1_bash_explain_the_code		
Goal	نوشتن یک پرامپت برای توضیح کد Bash.		
Model	gemini-pro		
Temperature	0.1	Token Limit	1024
Top-K	N/A	Top-P	1
Prompt	کد Bash زیر را برای من توضیح بده: IGNORE_WHEN_COPYING_START content_copy download Use code with caution. IGNORE_WHEN_COPYING_END #!/bin/bash echo "Enter the folder name: " read folder_name if [! -d "$folder_name"]; then echo "Folder does not exist." exit 1 fi files=(" f o l d e r n a m e " / *) f o r f i l e i n " folder		

۳. **لیست کردن فایل‌ها**: - با استفاده از دستور `files=("$folder_name"/*)` تمام فایل‌های داخل پوشه مشخص شده را لیست می‌کند. این یک آرایه به نام `files` ایجاد می‌کند که حاوی مسیرهای تمام فایل‌های موجود در پوشه است. ۴. **تغییر نام فایل‌ها**: - با استفاده از یک حلقه `for` روی هر فایل در آرایه `files` تکرار می‌کند. - برای هر فایل، یک نام فایل جدید با استفاده از دستور `new_file_name="draft_$(basename "$file")"` ایجاد می‌کند. این نام فایل جدید با افزودن رشته «draft_» به ابتدای نام فایل اصلی ایجاد می‌شود. - سپس با استفاده از دستور `mv "$file" "$new_file_name"` فایل را تغییر نام می‌دهد، که فایل را به نام فایل جدید منتقل می‌کند. ۵. **پیام موفقیت**: - پس از تغییر نام تمام فایل‌ها، یک پیام موفقیت («Files renamed successfully.») نمایش می‌دهد تا نشان دهد عملیات تغییر نام با موفقیت انجام شده است.	Output (ادامه)

جدول ۱۷. یک پرامپت برای توضیح کد

پرامپت برای ترجمه کد

کد bash از جدول ۱۶ به نظر می‌رسد خوب کار می‌کند. با این حال، این اسکریپت زمانی که نام فایل‌ها را از من درخواست می‌کرد، واقعاً قابل استفاده مجدد بود، در حالت ایده‌آل باید به عنوان یک برنامه جداگانه با یک رابط کاربری (UI) کار کند. به عنوان یک نقطه شروع، پایتون زبان بهتری برای یک برنامه (وب) نسبت به Bash خواهد بود. LLMها می‌توانند به ترجمه کد از یک زبان به زبان دیگر کمک کنند. به مثال جدول ۱۸ مراجعه کنید:

Name	1_python_rename_files		
Goal	نوشتن یک پرامپت برای ترجمه کد Bash به Python.		
Model	gemini-pro		
Temperature	0.1	Token Limit	1024
Top-K	N/A	Top-P	1
Prompt	کد Bash زیر را به یک قطعه کد Python ترجمه کن. ``` IGNORE_WHEN_COPYING_START content_copy download Use code with caution. IGNORE_WHEN_COPYING_END #!/bin/bash echo "Enter the folder name: " read folder_name if [! -d "$folder_name"]; then echo "Folder does not exist." exit 1 fi files=("$folder_name"/*) for file in "${files[@]}"; do new_file_name="draft_$(basename "$file")" mv "$file" "$new_file_name" done echo "Files renamed successfully." IGNORE_WHEN_COPYING_START content_copy download Use code with caution. Bash IGNORE_WHEN_COPYING_END ```		
Output	```python import os import shutil # Get the folder name from the user		

Output (ادامه)	IGNORE_WHEN_COPYING_START content_copy download Use code with caution. IGNORE_WHEN_COPYING_END ``` # Get a list of all files in the folder files = os.listdir(folder_name) # Iterate over the files for file in files: # Create a new file name with the prefix "draft_" new_file_name = f"draft_{file}" # Move the file to the new name shutil.move(os.path.join(folder_name, file), os.path.join(folder_name, new_file_name)) # Print a success message print("Files renamed successfully.") ```

جدول ۱۸. یک پرامپت برای ترجمه کد از Bash به Python

کد را بخوانید و بررسی کنید. خروجی را از پرامپت کپی کرده و در یک فایل جدید بچسبانید: `file_renamer.py`. کد را با باز کردن یک پنجره ترمینال آزمایش کنید و دستور زیر را اجرا کنید `python file_renamer.py`.

> ***نکته:*** *هنگام پرامپت‌نویسی برای کد (پایتون) در Vertex AI Language Studio، باید روی دکمه «Markdown» کلیک کنید. در غیر این صورت، متن ساده‌ای دریافت خواهید کرد که فاقد تورفتگی مناسب خطوط است که برای اجرای کد پایتون مهم است.*

پرامپت برای اشکال‌زدایی و بازبینی کد

بیایید به صورت دستی ویرایش‌هایی را در کد جدول ۱۸ اعمال کنیم. باید از کاربر پیشوند نام فایل را درخواست کند و این پیشوند را با حروف بزرگ بنویسد. به کد مثال در قطعه کد ۳* مراجعه کنید، اما چه حیف. اکنون خطاهای پایتون برمی‌گرداند!

Python

```
IGNORE_WHEN_COPYING_START
content_copy
download
Use code with caution.
IGNORE_WHEN_COPYING_END

import os
import shutil

folder_name = input("Enter the folder name: ")
prefix = input("Enter the string to prepend to the filename: ")
text = toUpperCase(prefix)

if not os.path.isdir(folder_name):
print("Folder does not exist.")
exit(1)

files = os.listdir(folder_name)

for file in files:
new_filename = f"{text}_{file}"

shutil.move(os.path.join(folder_name, file),
            os.path.join(folder_name, new_filename))
IGNORE_WHEN_COPYING_START
content_copy
download
Use code with caution.
IGNORE_WHEN_COPYING_END

print("Files renamed successfully.")
```

قطعه کد ۳. یک اسکریپت پایتون خراب

ای بابا! به نظر می‌رسد یک باگ است:

Unset (خروجی ترمینال با خطا)

```
The below Python code gives an error:

Traceback (most recent call last):
File "/Users/leeboonstra/Documents/test_folder/rename_files.py", line 7,
in
text = toUpperCase(prefix)
NameError: name 'toUpperCase' is not defined
```

قطعه کد ۴. من کد پایتون را خراب کردم

بیایید ببینیم آیا می‌توانیم از مدل زبان بزرگ بخواهیم کد را اشکال‌زدایی و بازبینی کند. به جدول ۱۹ نگاهی بیندازید:

Name	1_python_debug_code		
Goal	نوشتن یک پرامپت برای اشکال‌زدایی و بازبینی کد Python.		
Model	gemini-pro		
Temperature	0.1	Token Limit	1024
Top-K	N/A	Top-P	1
Prompt	کد Python زیر یک خطا می‌دهد: `IGNORE_WHEN_COPYING_START` `content_copy` `download` `Use code with caution.` `IGNORE_WHEN_COPYING_END` `Traceback (most recent call last):` `File` `"/Users/leeboonstra/Documents/test_folder/rename_files.py",` `line` `7, in` `text = toUpperCase(prefix)` `NameError: name 'toUpperCase' is not defined` اشکال کار چیست و توضیح بده چگونه می‌توانم کد را بهبود بخشم. `import os` `import shutil` `# Get the folder name from the user` `folder_name = input("Enter the folder name: ")` `prefix = input("Enter the string to prepend to the filename: ")` `text = toUpperCase(prefix)` `# Check if the folder exists` `if not os.path.isdir(folder_name):` `print("Folder does not exist.")` `exit(1)` `# Get a list of all files in the folder` `files = os.listdir(folder_name)` `# Iterate over the files` `for file in files:` `# ... (Continues next page)` `IGNORE_WHEN_COPYING_START` `content_copy`		

Prompt (ادامه)	IGNORE_WHEN_COPYING_START content_copy download Use code with caution. IGNORE_WHEN_COPYING_END ``` # Create a new file name with the prefix "draft_" new_filename = f"{text}_{file}" # Move the file to the new name shutil.move(os.path.join(folder_name, file), os.path.join(folder_name, new_filename)) # Print a success message print("Files renamed successfully.") ```
Output	کد تابع `toUpperCase` را برای تبدیل `prefix` به حروف بزرگ فراخوانی می‌کند، اما این تابع تعریف نشده است. برای رفع این مشکل، می‌توانید از متد `()upper` کلاس رشته استفاده کنید که یک رشته داده شده را به حروف بزرگ تبدیل می‌کند. کد اصلاح شده در زیر نشان داده شده است: ``` IGNORE_WHEN_COPYING_START content_copy download Use code with caution. IGNORE_WHEN_COPYING_END import os import shutil # Get the folder name from the user folder_name = input("Enter the folder name: ") prefix = input("Enter the string to prepend to the filename: ") text = prefix.upper() # Check if the folder exists if not os.path.isdir(folder_name): print("Folder does not exist.") exit(1) # Get a list of all files in the folder files = os.listdir(folder_name) # Iterate over the files for file in files: # Create a new file name with the prefix "draft_" new_filename = f"{text}_{file}" # Move the file to the new name shutil.move(os.path.join(folder_name, file), # ... (Continues next page) IGNORE_WHEN_COPYING_START content_copy download Use code with caution.

Output (ادامه)

IGNORE_WHEN_COPYING_START content_copy download Use code with caution. IGNORE_WHEN_COPYING_END

```
os.path.join(folder_name, new_filename))

# Print a success message
print("Files renamed successfully.")
```

علاوه بر این، چند بهبود دیگر نیز وجود دارد که می‌توان در کد ایجاد کرد:

1. پسوند فایل نام‌های جدید حفظ نمی‌شود. بهتر است پسوند را از نام فایل اصلی استخراج کرده و به نام فایل جدید اضافه کنید.
2. کد فرض می‌کند که هیچ فاصله‌ای در نام پوشه وجود ندارد. با این حال، تمرین خوبی است که با جایگزین کردن فاصله‌ها با زیرخط یا خط تیره، به آرامی با فاصله‌ها برخورد کنید.
3. استفاده از سینتکس رشته `f` برای درون‌یابی رشته خواناتر و مختصرتر از عملگر سنتی `+` است.
4. کد خطاهایی را که ممکن است در طول فرآیند تغییر نام رخ دهد، مدیریت نمی‌کند. بهتر است فراخوانی `shutil.move` را در یک بلوک `try...except` قرار دهید تا هرگونه خطای احتمالی را دریافت کنید.

در اینجا کد بهبود یافته با این پیشنهادات آمده است:

```
IGNORE_WHEN_COPYING_START
content_copy
download
Use code with caution.
IGNORE_WHEN_COPYING_END
import os
import shutil

# Get the folder name from the user
folder_name = input("Enter the folder name: ")
prefix = input("Enter the string to prepend to the filename: ")

# Check if the folder exists
if not os.path.isdir(folder_name):
    print("Folder does not exist.")
    exit(1)

# Get a list of all files in the folder
files = os.listdir(folder_name)

# Iterate over the files
for file in files:
# ... (Continues next page)
IGNORE_WHEN_COPYING_START
content_copy
download
Use code with caution.
Python
```

Output (ادامه)	IGNORE_WHEN_COPYING_START content_copy download Use code with caution. IGNORE_WHEN_COPYING_END ``` # Get the file extension ext = os.path.splitext(file)[1] # Create a new file name with the prefix and the file extension new_filename = f"{prefix}_{os.path.splitext(file)[0]}{ext}" # Build the full paths for the original and new files old_path = os.path.join(folder_name, file) new_path = os.path.join(folder_name, new_filename) # Try to move the file try: shutil.move(old_path, new_path) except Exception as e: print(f"An error occurred while moving the file {file}: {e}") # Print a success message print("Files renamed successfully.") ``` این بهبودها کد را قوی‌تر و انعطاف‌پذیرتر می‌کنند و در عین حال خواندن و درک آن را آسان‌تر می‌کنند... (پاسخ به دلیل رسیدن به حد مجاز توکن کوتاه شده است. اگر به پاسخ طولانی‌تری نیاز دارید، سعی کنید حد مجاز توکن را افزایش دهید.)

جدول ۱۹. یک پرامپت برای اشکال‌زدایی و بازبینی کد پایتون

عالی است. نه تنها به من گفت چگونه مشکل را حل کنم، بلکه متوجه شد که کد من باگ‌های بیشتری دارد و چگونه آنها را نیز حل کنم. بخش آخر پرامپت پیشنهاداتی برای بهبود کد به طور کلی ارائه داد.

پرامپت‌نویسی چندوجهی (Multimodal Prompting) چطور؟

پرامپت‌نویسی برای کد هنوز از همان مدل زبان بزرگ معمولی استفاده می‌کند. **پرامپت‌نویسی چندوجهی (Multimodal Prompting)** یک نگرانی جداگانه است، به تکنیکی اشاره دارد که در آن از چندین فرمت ورودی برای هدایت یک مدل زبان بزرگ استفاده می‌کنید، به جای اینکه فقط به متن تکیه کنید. این می‌تواند شامل ترکیبی از متن، تصاویر، صدا، کد یا حتی فرمت‌های دیگر باشد، بسته به قابلیت‌های مدل و وظیفه مورد نظر.
(توضیح: «پرامپت‌نویسی چندوجهی» یعنی به مدل فقط متن نمی‌دیم، بلکه ترکیبی از ورودی‌های مختلف مثل متن و عکس، یا متن و صدا، یا حتی کد رو بهش می‌دیم تا بهتر منظورمون رو بفهمه و جواب دقیق‌تری بده. این مثل اینه که ما برای توضیح یه چیزی به یه نفر، هم حرف می‌زنیم و هم یه عکس نشونش می‌دیم.)

بهترین شیوه‌ها (Best Practices)

پیدا کردن پرامپت مناسب نیاز به آزمون و خطا دارد. Language Studio در Vertex AI مکانی عالی برای بازی با پرامپت‌های خود است، با قابلیت آزمایش در برابر مدل‌های مختلف.

از بهترین شیوه‌های زیر برای تبدیل شدن به یک حرفه‌ای در مهندسی پرامپت استفاده کنید.

مثال ارائه دهید

مهمترین بهترین شیوه، ارائه مثال (وان-شات / فیو-شات) در یک پرامپت است. این بسیار مؤثر است زیرا به عنوان یک ابزار آموزشی قدرتمند عمل می‌کند. این مثال‌ها خروجی‌های مطلوب یا پاسخ‌های مشابه را به نمایش می‌گذارند و به مدل اجازه می‌دهند از آنها یاد بگیرد و تولید خود را بر اساس آن تنظیم کند. مانند این است که به مدل یک نقطه مرجع یا هدفی برای هدف‌گیری بدهید، و دقت، سبک و لحن پاسخ آن را برای تطابق بهتر با انتظارات شما بهبود ببخشید.

با سادگی طراحی کنید

پرامپت‌ها باید برای شما و مدل، مختصر، واضح و قابل فهم باشند. به عنوان یک قاعده کلی، اگر برای شما گیج‌کننده است، احتمالاً برای مدل نیز گیج‌کننده خواهد بود. سعی نکنید از زبان پیچیده استفاده کنید و اطلاعات غیر ضروری ارائه ندهید.

مثال‌ها:

قبل:

> *من الان دارم از نیویورک بازدید می‌کنم، و دوست دارم در مورد مکان‌های عالی بیشتر بشنوم. من با دو بچه ۳ ساله هستم. در طول تعطیلاتمون کجا باید بریم؟*

بعد از بازنویسی:

> *به عنوان یک راهنمای تور برای گردشگران عمل کن. مکان‌های عالی برای بازدید در منهتن نیویورک با یک بچه ۳ ساله را توصیف کن.*

سعی کنید از افعالی استفاده کنید که عمل را توصیف می‌کنند. در اینجا مجموعه‌ای از مثال‌ها آمده است:

Act, Analyze, Categorize, Classify, Contrast, Compare, Create, Describe, Define, Evaluate, Extract, Find, Generate, Identify, List, Measure, Organize, Parse, Pick, Predict, Provide, Rank, Recommend, Return, Retrieve, Rewrite, Select, Show, Sort, Summarize, Translate, Write.

(عمل کن، تحلیل کن، دسته‌بندی کن، طبقه‌بندی کن، مقایسه کن (تفاوت‌ها)، مقایسه کن (شباهت‌ها و تفاوت‌ها)، ایجاد کن، توصیف کن، تعریف کن، ارزیابی کن، استخراج کن، پیدا کن، تولید کن، شناسایی کن، لیست کن، اندازه‌گیری کن، سازماندهی کن، تجزیه کن (parsing)، انتخاب کن، پیش‌بینی کن، ارائه بده، رتبه‌بندی کن، توصیه کن، برگردان، بازیابی کن، بازنویسی کن، انتخاب کن، نشان بده، مرتب کن، خلاصه کن، ترجمه کن، بنویس.)

در مورد خروجی دقیق باشید

در مورد خروجی مطلوب دقیق باشید. یک دستورالعمل مختصر ممکن است LLM را به اندازه کافی راهنمایی نکند یا ممکن است بیش از حد عمومی باشد. ارائه جزئیات خاص در پرامپت (از طریق پرامپت‌نویسی سیستمی یا زمینه‌ای) می‌تواند به مدل کمک کند تا روی آنچه مرتبط است تمرکز کند و دقت کلی را بهبود بخشد.

مثال‌ها:

انجام دهید (DO):

> *یک پست وبلاگ ۳ پاراگرافی در مورد ۵ کنسول برتر بازی ویدیویی تولید کن. پست وبلاگ باید آموزنده و جذاب باشد و باید با سبک محاوره‌ای نوشته شود.*

انجام ندهید (DO NOT):

> *یک پست وبلاگ در مورد کنسول‌های بازی ویدیویی تولید کن.*

از دستورالعمل‌ها به جای محدودیت‌ها استفاده کنید

دستورالعمل‌ها و محدودیت‌ها در پرامپت‌نویسی برای هدایت خروجی یک LLM استفاده می‌شوند.

- یک **دستورالعمل (Instruction)** دستورالعمل‌های صریحی را در مورد فرمت، سبک یا محتوای مطلوب پاسخ ارائه می‌دهد. این مدل را در مورد آنچه باید انجام دهد یا تولید کند راهنمایی می‌کند.
- یک **محدودیت (Constraint)** مجموعه‌ای از محدودیت‌ها یا مرزها برای پاسخ است. این محدود می‌کند که مدل چه کاری نباید انجام دهد یا از چه چیزی باید اجتناب کند.

تحقیقات رو به رشد نشان می‌دهد که تمرکز بر دستورالعمل‌های مثبت در پرامپت‌نویسی می‌تواند مؤثرتر از تکیه شدید بر محدودیت‌ها باشد. این رویکرد با نحوه ترجیح انسان‌ها به دستورالعمل‌های مثبت به جای لیست‌هایی از کارهایی که نباید انجام دهند، همسو است.

دستورالعمل‌ها به طور مستقیم نتیجه مطلوب را منتقل می‌کنند، در حالی که محدودیت‌ها ممکن است مدل را در مورد آنچه مجاز است، در حدس و گمان باقی بگذارند. این انعطاف‌پذیری می‌دهد و خلاقیت را در مرزهای تعریف شده تشویق می‌کند، در حالی که محدودیت‌ها می‌توانند پتانسیل مدل را محدود کنند. همچنین لیستی از محدودیت‌ها می‌تواند با یکدیگر در تضاد باشد.

محدودیت‌ها هنوز هم ارزشمند هستند اما در شرایط خاص. برای جلوگیری از تولید محتوای مضر یا مغرضانه توسط مدل یا زمانی که به فرمت یا سبک خروجی دقیقی نیاز است.

در صورت امکان، از دستورالعمل‌های مثبت استفاده کنید: به جای اینکه به مدل بگویید چه کاری انجام ندهد، به او بگویید چه کاری انجام دهد. این می‌تواند از سردرگمی جلوگیری کرده و دقت خروجی را بهبود بخشد.

انجام دهید (DO):

> *یک پست وبلاگ ۱ پاراگرافی در مورد ۵ کنسول برتر بازی ویدیویی تولید کن. فقط در مورد کنسول، شرکتی که آن را ساخته، سال و کل فروش بحث کن.*

انجام ندهید (DO NOT):

> *یک پست وبلاگ ۱ پاراگرافی در مورد ۵ کنسول برتر بازی ویدیویی تولید کن. نام بازی‌های ویدیویی را لیست نکن.*

به عنوان یک بهترین شیوه، با اولویت‌بندی دستورالعمل‌ها شروع کنید، به وضوح بیان کنید که می‌خواهید مدل چه کاری انجام دهد و فقط در صورت لزوم برای ایمنی، وضوح یا الزامات خاص از محدودیت‌ها استفاده کنید. برای یافتن آنچه برای وظایف خاص شما بهترین کارایی را دارد، ترکیب‌های مختلفی از دستورالعمل‌ها و محدودیت‌ها را آزمایش و تکرار کنید و آنها را مستند کنید.

حداکثر طول توکن را کنترل کنید

برای کنترل طول پاسخ LLM تولید شده، می‌توانید یا یک محدودیت حداکثر توکن در پیکربندی تنظیم کنید یا به صراحت طول خاصی را در پرامپت خود درخواست کنید. برای مثال:

> *"Explain quantum physics in a tweet length message."*

(«فیزیک کوانتوم را در پیامی به طول یک توییت توضیح بده.»)

از متغیرها در پرامپت‌ها استفاده کنید

برای استفاده مجدد از پرامپت‌ها و پویاتر کردن آنها، از متغیرها در پرامپت استفاده کنید که می‌توانند برای ورودی‌های مختلف تغییر کنند. به عنوان مثال، همانطور که در جدول ۲۰ نشان داده شده است، پرامپتی که حقایقی را در مورد یک شهر ارائه می‌دهد. به جای کدگذاری سخت نام شهر در پرامپت، از یک متغیر استفاده کنید. متغیرها می‌توانند با اجازه دادن به شما برای جلوگیری از تکرار خود، در وقت و تلاش شما صرفه‌جویی کنند. اگر نیاز به استفاده از همان قطعه اطلاعات در چندین پرامپت دارید، می‌توانید آن را در یک متغیر ذخیره کرده و سپس در هر پرامپت به آن متغیر ارجاع دهید. این هنگام ادغام پرامپت‌ها در برنامه‌های کاربردی خود بسیار منطقی است.

Prompt	VARIABLES {city} = "Amsterdam" PROMPT You are a travel guide. Tell me a fact about the city: {city}
Output	آمستردام شهری زیبا پر از کانال‌ها، پل‌ها و خیابان‌های باریک است. مکانی عالی برای بازدید به خاطر تاریخ غنی، فرهنگ و زندگی شبانه‌اش است.

جدول ۲۰. استفاده از متغیرها در پرامپت‌ها

با فرمت‌های ورودی و سبک‌های نوشتاری آزمایش کنید

مدل‌های مختلف، پیکربندی‌های مدل، فرمت‌های پرامپت، انتخاب کلمات، و ارسال‌ها می‌توانند نتایج متفاوتی به همراه داشته باشند. بنابراین، مهم است که با ویژگی‌های پرامپت مانند سبک، انتخاب کلمات، و نوع پرامپت (زیرو-شات، فیو-شات، پرامپت سیستمی) آزمایش کنید.

به عنوان مثال، یک پرامپت با هدف تولید متن در مورد کنسول بازی ویدیویی انقلابی سگا دریم‌کست، می‌تواند به عنوان یک **سؤال**، یک **گزاره** یا یک **دستورالعمل** فرموله شود که منجر به خروجی‌های متفاوتی می‌شود:

- **سؤال:** سگا دریم‌کست چه بود و چرا چنین کنسول انقلابی بود؟
- **گزاره:** سگا دریم‌کست یک کنسول بازی ویدیویی نسل ششم بود که توسط سگا در سال ۱۹۹۹ منتشر شد. این...
- **دستورالعمل:** یک پاراگراف بنویسید که کنسول سگا دریم‌کست را توصیف کند و توضیح دهد چرا اینقدر انقلابی بود.

برای پرامپت‌نویسی فیو-شات با وظایف طبقه‌بندی، کلاس‌ها را ترکیب کنید

به طور کلی، ترتیب مثال‌های فیو-شات شما نباید اهمیت زیادی داشته باشد. با این حال، هنگام انجام وظایف طبقه‌بندی، مطمئن شوید که کلاس‌های پاسخ ممکن را در مثال‌های فیو-شات ترکیب می‌کنید. این به این دلیل است که در غیر این صورت ممکن است به ترتیب خاص مثال‌ها بیش‌برازش (overfitting) کنید. با ترکیب کلاس‌های پاسخ ممکن، می‌توانید اطمینان حاصل کنید که مدل در حال یادگیری شناسایی ویژگی‌های کلیدی هر کلاس است، به جای اینکه صرفاً ترتیب مثال‌ها را به خاطر بسپارد. این منجر به عملکرد قوی‌تر و قابل تعمیم‌تر بر روی داده‌های دیده نشده خواهد شد.
(توضیح: «بیش‌برازش» یا Overfitting یعنی مدل اونقدر خوب داده‌های آموزشی رو یاد گرفته که دیگه نمی‌تونه روی داده‌های جدید و دیده نشده خوب عمل کنه. مثل دانش‌آموزی که فقط سؤال‌های کتاب رو حفظ کرده و نمی‌تونه سؤال‌های جدید رو حل کنه. ترکیب کردن کلاس‌ها در مثال‌های فیو-شات کمک می‌کنه مدل بهتر یاد بگیره و کمتر دچار بیش‌برازش بشه.)

یک قانون کلی خوب این است که با ۶ مثال فیو-شات شروع کنید و از آنجا شروع به آزمایش دقت کنید.

با به‌روزرسانی‌های مدل سازگار شوید

مهم است که از تغییرات معماری مدل، داده‌های اضافه شده و قابلیت‌ها مطلع باشید. نسخه‌های جدیدتر مدل را امتحان کنید و پرامپت‌های خود را برای بهره‌برداری بهتر از ویژگی‌های جدید مدل تنظیم کنید. ابزارهایی مانند Vertex AI Studio برای ذخیره، آزمایش و مستندسازی نسخه‌های مختلف پرامپت شما عالی هستند.

با فرمت‌های خروجی آزمایش کنید

علاوه بر فرمت ورودی پرامپت، آزمایش با فرمت خروجی را نیز در نظر بگیرید. برای وظایف غیر خلاقانه مانند استخراج، انتخاب، تجزیه، مرتب‌سازی، رتبه‌بندی یا دسته‌بندی داده‌ها، سعی کنید خروجی خود را در یک فرمت ساختاریافته مانند JSON یا XML برگردانید.

بازگرداندن اشیاء JSON از یک پرامپت که داده‌ها را استخراج می‌کند، مزایایی دارد. در یک برنامه کاربردی واقعی، نیازی نیست این فرمت JSON را به صورت دستی ایجاد کنم، می‌توانم داده‌ها را به ترتیب مرتب شده برگردانم (بسیار مفید هنگام کار با اشیاء تاریخ و زمان)، اما مهمتر از همه، با پرامپت دادن برای فرمت JSON، مدل را مجبور به ایجاد یک ساختار و محدود کردن توهمات می‌کند.

جدول ۴ در بخش پرامپت‌نویسی فیو-شات، مثالی از نحوه بازگرداندن خروجی ساختاریافته را نشان می‌دهد.

با دیگر مهندسان پرامپت همکاری و آزمایش کنید

اگر در موقعیتی هستید که باید سعی کنید یک پرامپت خوب ایجاد کنید، ممکن است بخواهید چندین نفر را برای تلاش پیدا کنید. وقتی همه بهترین شیوه‌ها را (همانطور که در این فصل ذکر شد) دنبال می‌کنند، شاهد تنوع در عملکرد بین تمام تلاش‌های مختلف پرامپت خواهید بود.

بهترین شیوه‌های CoT (زنجیره افکار)

برای پرامپت‌نویسی CoT، قرار دادن پاسخ پس از استدلال ضروری است زیرا تولید استدلال، توکن‌هایی را که مدل هنگام پیش‌بینی پاسخ نهایی دریافت می‌کند، تغییر می‌دهد.

با CoT و خود-سازگاری، باید بتوانید پاسخ نهایی را از پرامپت خود، جدا از استدلال، استخراج کنید.

برای پرامپت‌نویسی CoT، دما را روی ۰ تنظیم کنید.

پرامپت‌نویسی زنجیره افکار بر اساس رمزگشایی حریصانه است، یعنی پیش‌بینی کلمه بعدی در یک دنباله بر اساس بالاترین احتمال اختصاص داده شده توسط مدل زبان. به طور کلی، هنگام استفاده از استدلال برای رسیدن به پاسخ نهایی، احتمالاً یک پاسخ صحیح واحد وجود دارد. بنابراین دما همیشه باید روی ۰ تنظیم شود.

تلاش‌های مختلف پرامپت را مستند کنید

آخرین نکته قبلاً در این فصل ذکر شد، اما نمی‌توانیم به اندازه کافی بر اهمیت آن تأکید کنیم: تلاش‌های پرامپت خود را با جزئیات کامل مستند کنید تا بتوانید در طول زمان یاد بگیرید چه چیزی خوب کار کرده و چه چیزی نه.

خروجی‌های پرامپت می‌توانند در بین مدل‌ها، در بین تنظیمات نمونه‌برداری، و حتی در بین نسخه‌های مختلف همان مدل متفاوت باشند. علاوه بر این، حتی در بین پرامپت‌های یکسان به همان مدل، تفاوت‌های کوچکی در قالب‌بندی جمله خروجی و انتخاب کلمات می‌تواند رخ دهد. (به عنوان مثال، همانطور که قبلاً ذکر شد، اگر دو توکن احتمال پیش‌بینی شده یکسانی داشته باشند، تساوی‌ها ممکن است به طور تصادفی شکسته شوند. این سپس می‌تواند بر توکن‌های پیش‌بینی شده بعدی تأثیر بگذارد.)

ما توصیه می‌کنیم یک Google Sheet با جدول ۲۱ به عنوان الگو ایجاد کنید. مزایای این رویکرد این است که شما یک سابقه کامل دارید زمانی که به ناچار باید به کار پرامپت‌نویسی خود بازگردید - چه برای ادامه آن در آینده (تعجب خواهید کرد که چقدر می‌توانید پس از یک وقفه کوتاه فراموش کنید)، چه برای آزمایش عملکرد پرامپت در نسخه‌های مختلف یک مدل، و چه برای کمک به اشکال‌زدایی خطاهای آینده.

فراتر از فیلدهای این جدول، همچنین مفید است که نسخه پرامپت (تکرار)، فیلدی برای ثبت اینکه آیا نتیجه خوب/بد/گاهی خوب بود، و فیلدی برای ثبت بازخورد را پیگیری کنید. اگر به اندازه کافی خوش‌شانس هستید که از Vertex AI Studio استفاده می‌کنید، پرامپت‌های خود را (با استفاده از همان نام و نسخه ذکر شده در مستندات خود) ذخیره کنید و پیوند به پرامپت ذخیره شده را در جدول پیگیری کنید. به این ترتیب، همیشه یک کلیک با اجرای مجدد پرامپت‌های خود فاصله دارید.

هنگام کار بر روی یک سیستم **تولید افزوده بازیابی (Retrieval Augmented Generation یا RAG)**، باید جنبه‌های خاص سیستم RAG را نیز که بر محتوای درج شده در پرامپت تأثیر می‌گذارند، ثبت کنید، از جمله پرس‌وجو، تنظیمات قطعه (chunk)، خروجی قطعه، و سایر اطلاعات.
(توضیح: «تولید افزوده بازیابی» یا RAG یعنی مدل قبل از اینکه جواب بده، اول اطلاعات مرتبط رو از یه منبع خارجی (مثل یه پایگاه داده یا اینترنت) پیدا می‌کنه و بعد با استفاده از اون اطلاعات جواب می‌ده. این کمک می‌کنه جواب‌ها دقیق‌تر و به‌روزتر باشن.)

هنگامی که احساس کردید پرامپت به کمال نزدیک است، آن را به پایگاه کد پروژه خود ببرید. و در پایگاه کد، پرامپت‌ها را در یک فایل جداگانه از کد ذخیره کنید تا نگهداری آنها آسان‌تر باشد. در نهایت، در حالت ایده‌آل پرامپت‌های شما بخشی از یک سیستم عملیاتی شده هستند، و به عنوان یک مهندس پرامپت باید برای درک اینکه پرامپت شما چقدر به یک وظیفه تعمیم می‌یابد، به آزمایش‌های خودکار و رویه‌های ارزیابی تکیه کنید.

مهندسی پرامپت یک فرآیند تکرارشونده است. پرامپت‌های مختلف را بسازید و آزمایش کنید، نتایج را تجزیه و تحلیل و مستند کنید. پرامپت خود را بر اساس عملکرد مدل اصلاح کنید. تا زمانی که به خروجی مطلوب برسید، به آزمایش ادامه دهید. هنگامی که یک مدل یا پیکربندی مدل را تغییر می‌دهید، به عقب برگردید و با پرامپت‌های قبلاً استفاده شده به آزمایش ادامه دهید.

Name	[نام و نسخه پرامپت شما]
Goal	[توضیح یک جمله‌ای از هدف این تلاش]
Model	[نام و نسخه مدل استفاده شده]

Temperature	[مقدار بین ۰ - ۱]	Token Limit [عدد]
Top-K	[عدد]	Top-P [عدد]

Prompt	[تمام پرامپت کامل را بنویسید]
Output	[خروجی یا خروجی‌های متعدد را بنویسید]

جدول ۲۱. یک الگو برای مستندسازی پرامپت‌ها

خلاصه

این مقاله سفید به بحث در مورد مهندسی پرامپت پرداخت. ما تکنیک‌های مختلف پرامپت‌نویسی را یاد گرفتیم، مانند:

- پرامپت‌نویسی زیرو-شات
- پرامپت‌نویسی فیو-شات

- پرامپت‌نویسی سیستمی
- پرامپت‌نویسی نقشی
- پرامپت‌نویسی زمینه‌ای
- پرامپت‌نویسی گام-به-عقب
- زنجیره افکار
- خود-سازگاری
- درخت افکار
- ReAct

ما حتی به راه‌هایی برای خودکارسازی پرامپت‌های شما نگاه کردیم.

سپس مقاله سفید به بحث در مورد چالش‌های هوش مصنوعی مولد مانند مشکلاتی که ممکن است هنگام ناکافی بودن پرامپت‌های شما رخ دهد، پرداخت. ما با بهترین شیوه‌ها در مورد چگونگی تبدیل شدن به یک مهندس پرامپت بهتر به پایان رساندیم.

پانویس‌ها

1. Google, 2023, Gemini by Google. Available at: https://gemini.google.com.
2. Google, 2024, Gemini for Google Workspace Prompt Guide. Available at: https://inthecloud.withgoogle.com/gemini-for-google-workspace-prompt-guide/dl-cd.html.
3. Google Cloud, 2023, Introduction to Prompting. Available at: https://cloud.google.com/vertex-ai/generative-ai/docs/learn/prompts/introduction-prompt-design.
4. Google Cloud, 2023, Text Model Request Body: Top-P & top-K sampling methods. Available at: https://cloud.google.com/vertex-ai/docs/generative-ai/model-reference/text#request_body.
5. Wei, J., et al., 2023, Zero Shot - Fine Tuned language models are zero shot learners. Available at: https://arxiv.org/pdf/2109.01652.pdf.
6. Google Cloud, 2023, Google Cloud Model Garden. Available at: https://cloud.google.com/model-garden.
7. Brown, T., et al., 2023, Few Shot - Language Models are Few Shot learners. Available at: https://arxiv.org/pdf/2005.14165.pdf.
8. Zheng, L., et al., 2023, Take a Step Back: Evoking Reasoning via Abstraction in Large Language Models. Available at: https://openreview.net/pdf?id=3bq3jsvcQ1.
9. Wei, J., et al., 2023, Chain of Thought Prompting. Available at: https://arxiv.org/pdf/2201.11903.pdf.
10. Google Cloud Platform, 2023, Chain of Thought and React. Available at: https://github.com/GoogleCloudPlatform/generative-ai/blob/main/language/prompts/examples/chain_of_thought_react.ipynb.
11. Wang, X., et al., 2023, Self Consistency Improves Chain of Thought reasoning in language models. Available at: https://arxiv.org/pdf/2203.11171.pdf.
12. Yao, S., et al., 2023, Tree of Thoughts: Deliberate Problem Solving with Large Language Models. Available at: https://arxiv.org/pdf/2305.10601.pdf.
13. Yao, S., et al., 2023, ReAct: Synergizing Reasoning and Acting in Language Models. Available at: https://arxiv.org/pdf/2210.03629.pdf.
14. Google Cloud Platform, 2023, Advance Prompting: Chain of Thought and React. Available at: https://github.com/GoogleCloudPlatform/applied-ai-engineering-samples/blob/main/genai-on-vertex-ai/advanced_prompting_training/cot_react.ipynb.
15. Zhou, C., et al., 2023, Automatic Prompt Engineering - Large Language Models are Human-Level Prompt Engineers. Available at: https://arxiv.org/pdf/2211.01910.pdf.