عامل های هوش مصنوعی

نویسندگان: Julia Wiesinger, Patrick Marlow and Vladimir Vuskovic

مترجم: علی مرشدسلوک

Google

Acknowledgements

Reviewers and Contributors

Evan Huang Emily

Xue

Olcan Sercinoglu

Sebastian Riedel

Satinder Baveja

Antonio Gulli Anant

Nawalgaria

Curators and Editors

Antonio Gulli Anant

Nawalgaria Grace

Mollison

Technical Writer

Joey Haymaker

Designer

Michael Lanning

فهرست مطالب

> یک ترکیب از استدلال، منطق و دسترسی به اطلاعات خارجی که همگی به یک مدل هوش مصنوعی مولد متصل هستند، مفهومی از عامل را ایجاد می‌کند.

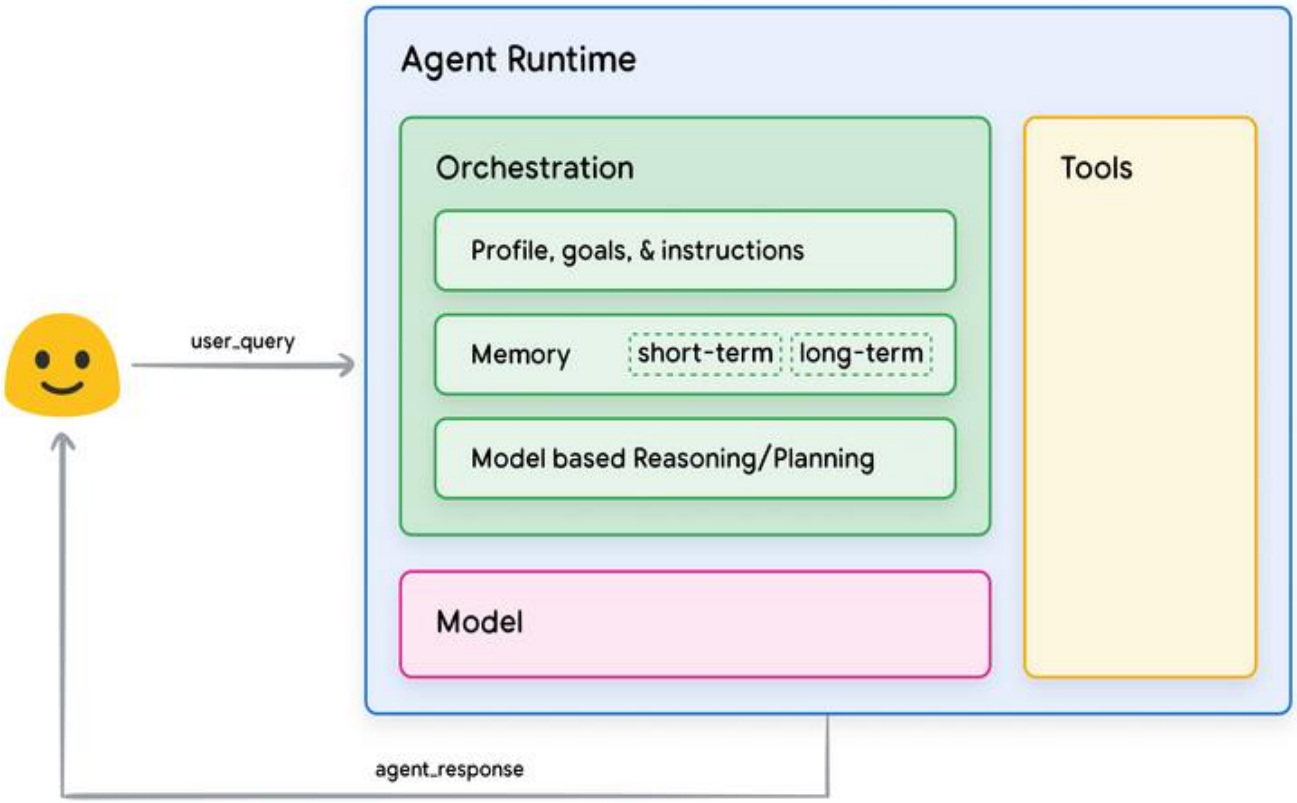
مقدمه

انسان‌ها در تشخیص الگوهای پیچیده و کثیف بسیار ماهرند. با این حال، آنها اغلب به ابزارهایی مانند کتاب‌ها، جستجوی گوگل یا ماشین حساب تکیه می‌کنند تا دانش قبلی خود را قبل از رسیدن به نتیجه تکمیل کنند. همانند انسان‌ها، مدل‌های هوش مصنوعی تولیدی می‌توانند برای استفاده از ابزارها آموزش ببینند تا به اطلاعات زمان واقعی دسترسی پیدا کنند یا عملی در دنیای واقعی پیشنهاد دهند. به عنوان مثال، یک مدل می‌تواند از یک ابزار بازیابی پایگاه داده برای دسترسی به اطلاعات خاصی مانند تاریخچه خرید مشتری استفاده کند تا پیشنهادات خرید شخصی‌سازی شده ایجاد کند. یا بر اساس پرس‌وجوی کاربر، یک مدل می‌تواند فراخوانهای مختلف API داشته تا پاسخ ایمیل به یک همکار ارسال کند یا یک تراکنش مالی از طرف شما انجام دهد. برای انجام این کار، مدل نه تنها باید به مجموعه‌ای از ابزارهای خارجی دسترسی داشته باشد، بلکه باید توانایی برنامه‌ریزی و اجرای هر وظیفه‌ای را به صورت مستقل داشته باشد. این ترکیب از استدلال، منطق و دسترسی به اطلاعات خارجی که همه به یک مدل هوش مصنوعی تولیدی متصل هستند، مفهومی ایجاد می کند به نام عامل، یا برنامه ای که فراتر از قابلیتهای یک مدل هوش مسنوعی مولد مستقل عمل می کند. در این مقاله به این مورد و جزییات مرتبط با آن پرداخته ایم.

یک عامل چیست

در ساده‌ترین شکل خود، یک عامل هوش مصنوعی مولد را می‌توان به عنوان یک برنامه تعریف کرد که سعی می‌کند با مشاهده جهان و عمل کردن در آن با استفاده از ابزارهایی که در اختیار دارد، به یک هدف دست یابد. عامل‌ها مستقل هستند و می‌توانند بدون دخالت انسان عمل کنند، به ویژه وقتی که اهداف یا اهدافی که قرار است به آنها برسند به درستی تعیین شده باشد. عامل‌ها می‌توانند در رویکرد خود به دستیابی به اهداف پیشگیر (Proactive) باشند. حتی در غیاب دستورالعمل‌های صریح از انسان، یک عامل می‌تواند در مورد اینکه چه باید بعداً انجام دهد تا به هدف نهایی خود برسد، استدلال کند. در حالی که مفهوم عامل‌ها در هوش مصنوعی بسیار کلی و قدرتمند است، این گزارش بر انواع خاصی از عامل‌هایی که مدل‌های هوش مصنوعی تولیدی قادر به ساخت آنها هستند در زمان انتشار تمرکز می‌کند.

برای درک بهتر نحوه عملکرد داخلی یک عامل، ابتدا باید اجزای بنیادی را که رفتار، اقدامات و تصمیم‌گیری عامل را هدایت می‌کنند معرفی کنیم. ترکیب این اجزا را می‌توان به عنوان یک معماری شناختی توصیف کرد و معماری‌های شناختی مختلفی وجود دارد که می‌تواند با ترکیب و تطبیق این اجزا به دست آید. با تمرکز بر عملکردهای اصلی، سه جزء ضروری در معماری شناختی یک عامل وجود دارد که در شکل ۱ نشان داده شده است.



شکل ۱. معماری کلی عامل و اجزای آن

مدل

در حوزه یک عامل، مدل به مدل زبانی **(LM)** اشاره دارد که به عنوان تصمیم‌گیرنده مرکزی در فرآیندهای عامل استفاده می‌شود. مدلی که توسط یک عامل استفاده می‌شود می‌تواند یک یا چند **LM** از هر اندازه‌ای (کوچک یا بزرگ) باشد که قادر به استدلال **(reasoning)** و منطق **(logic)** مبتنی بر دستورالعمل هستند، مانند **ReAct** ، **Chain-of-Thought** یا **Tree-of-Thoughts** . مدل‌ها می‌توانند عمومی، چندوجهی **(multimodal)** یا بر اساس نیازهای معماری عامل خاص شما تنظیم شوند. برای بهترین نتایج تولید، باید از مدلی استفاده کنید که بهترین شکل را برای کاربرد نهایی مورد نظر شما دارد و به شکل ایده‌آل، بر اساس داده‌های مرتبط با ابزارهایی که قصد استفاده از آنها در معماری شناختی **(cognitive)** را دارید، آموزش دیده باشد. مهم است بدانید که مدل معمولاً با تنظیمات پیکربندی خاص عامل (یعنی انتخاب ابزارها، تنظیمات هماهنگی/استدلال) آموزش نمی‌بیند. با این حال، می‌توان مدل را برای وظایف عامل با ارائه نمونه‌هایی که قابلیت‌های عامل را نشان می‌دهند، از جمله مواردی که عامل از ابزارهای خاص یا مراحل استدلالی در زمینه‌های مختلف **(context)** استفاده می‌کند، بیشتر تقویت کرد.

ابزارها

مدل‌های بنیادی، با وجود تولید متن و تصویر فوق‌العاده‌شان، به دلیل عدم توانایی در تعامل با دنیای خارج محدود می‌شوند. ابزارها این شکاف را پر می‌کنند و به عامل‌ها اجازه می‌دهند با داده‌ها و خدمات خارجی تعامل کنند و طیف گسترده‌تری از اقدامات را فراتر از آنچه که مدل اساسی به تنهایی می‌تواند انجام دهد، فعال کنند. ابزارها می‌توانند اشکال مختلفی داشته باشند و پیچیدگی‌های مختلفی داشته باشند، اما معمولاً با روش‌های API وب معمول مانندGET ، POST، PATCH و DELETE هم‌راستا هستند. به عنوان مثال، یک ابزار می‌تواند اطلاعات مشتری را در یک پایگاه داده به‌روز کند یا داده‌های آب‌وهوا را برای تأثیرگذاری بر توصیه سفری که عامل به کاربر ارائه می‌دهد، جستجو کند. با ابزارها، عامل‌ها می‌توانند به اطلاعات دنیای واقعی دسترسی پیدا کنند و آنها را پردازش کنند. این به آنها امکان می‌دهد سیستم‌های تخصصی‌تری مانند تولید تقویت‌شده با بازیابی (RAG) را پشتیبانی کنند که به طور قابل توجهی قابلیت‌های یک عامل را فراتر از آنچه که مدل بنیادی به تنهایی می‌تواند انجام دهد، گسترش می‌دهد. ما در مورد ابزارها به تفصیل بحث خواهیم کرد، اما مهم‌ترین چیزی که باید بفهمید این است که ابزارها شکاف بین توانایی‌های داخلی عامل و دنیای خارجی را پر می‌کنند و طیف گسترده‌تری از امکانات را فعال می‌کنند.

لایه هماهنگی

لایه هماهنگی یک فرآیند چرخشی را توصیف می‌کند که نحوه دریافت اطلاعات توسط عامل، انجام استدلال داخلی و استفاده از این استدلال برای اطلاع‌رسانی اقدام یا تصمیم بعدی را کنترل می‌کند. به طور کلی، این حلقه تا زمانی ادامه خواهد یافت که عامل به هدف خود برسد یا به نقطه توقفی برسد. پیچیدگی لایه هماهنگی می‌تواند بسته به عامل و وظیفه‌ای که انجام می‌دهد بسیار متفاوت باشد. برخی حلقه‌ها می‌توانند محاسبات ساده با قوانین تصمیم‌گیری باشند، در حالی که برخی دیگر ممکن است شامل منطق زنجیره‌ای، الگوریتم‌های یادگیری ماشین اضافی یا تکنیک‌های استدلال احتمالی دیگر باشند. ما در مورد پیاده‌سازی دقیق لایه‌های هماهنگی عامل در بخش معماری شناختی بیشتر بحث خواهیم کرد

عامل‌ها در مقابل مدل‌ها

برای درک بهتر تفاوت بین عامل‌ها و مدل‌ها، نمودار زیر را در نظر بگیرید:

مدل‌ها	عاملها
دانش آنها محدود به آن چیزی است که در داده‌های آموزشی آنها موجود است.	دانش از طریق اتصال با سیستم‌های خارجی از طریق ابزارها گسترش می‌یابد.
یک استنتاج / پیش‌بینی یکباره بر اساس پرس‌وجوی کاربر. مگر در حاتی که به طور خاص برای مدل پیاده‌سازی شود، هیچ مدیریتی از تاریخچه جلسه یا پیوستگی زمینه (کاتکست) وجود ندارد. (مانند تاریخچه چت)	مدیریت تاریخچه جلسه (مانند تاریخچه چت) برای استنتاج/پیش‌بینی چندمرحله‌ای بر اساس پرس‌وجوهای کاربر و تصمیمات گرفته شده در لایه ارکستراسیون را امکانپذیر می کند. در این زمینه، یک 'مرحله' (turn) به عنوان یک تعامل بین سیستم تعاملی و عامل تعریف می‌شود. (یعنی یک رویداد/پرس‌وجوی ورودی و یک پاسخ عامل)
پیاده‌سازی ابزار بومی وجود ندارد.	ابزارها به طور بومی در معماری عامل پیاده‌سازی شده‌اند.
لایه منطق بومی پیاده‌سازی نشده است. کاربران می‌توانند پرامپت‌ها را به صورت سوالات ساده یا از چارچوب‌های استدلالی (CoT، ReActو غیره) برای ایجاد پرامپت‌های پیچیده به منظور راهنمایی مدل در پیش‌بینی استفاده کنند.	معماری شناختی بومی که از چارچوب‌های استدلالی مانندCoT ، ReActیا چارچوب‌های عامل پیش‌ساخته مانند LangChain استفاده می‌کند.

معماری‌های شناختی: نحوه عملکرد عامل‌ها

یک آشپز در یک آشپزخانه شلوغ را تصور کنید. هدف آنها تهیه غذاهای خوشمزه برای مشتریان رستوران است که شامل چرخه‌ای از برنامه‌ریزی، اجرا و تنظیم می‌شود.

- آنها اطلاعات جمع‌آوری می‌کنند، مانند سفارش مشتری و مواد موجود در انبار و یخچال.
- آنها استدلال داخلی انجام می‌دهند در مورد اینکه چه غذاها و پروفایل‌های طعمی می‌توانند بر اساس اطلاعات جمع‌آوری شده ایجاد کنند.
- آنها اقدام به تهیه غذا می‌کنند: خرد کردن سبزیجات، ترکیب ادویه‌ها، سرخ کردن گوشت

در هر مرحله از فرآیند، آشپز به همان اندازه که مواد اولیه تمام می‌شوند یا بازخورد مشتری دریافت می‌شود، تنظیماتی انجام می‌دهد و از مجموعه نتایج قبلی برای تعیین برنامه عمل بعدی استفاده می‌کند. این چرخه دریافت اطلاعات، برنامه‌ریزی، اجرا و تنظیم، یک معماری شناختی منحصر به فرد را توصیف می‌کند که آشپز برای رسیدن به هدفش از آن استفاده می‌کند.
همانند آشپز، عامل‌ها می‌توانند از معماری‌های شناختی برای رسیدن به اهداف نهایی خود با پردازش متناوب اطلاعات، گرفتن تصمیمات آگاهانه و تصحیح اقدامات بعدی بر اساس خروجی‌های قبلی استفاده کنند. در مرکز معماری شناختی عامل‌ها، لایه ارکستراسیون (هماهنگی) قرار دارد که مسئول حفظ حافظه، وضعیت، استدلال و برنامه‌ریزی است. این لایه از حوزه در حال تحول سریع مهندسی پرامپت و چارچوب‌های مرتبط برای راهنمایی استدلال و برنامه‌ریزی استفاده می‌کند و به عامل اجازه می‌دهد به طور مؤثرتری با محیط خود تعامل کند و وظایف را تکمیل کند. تحقیقات در زمینه چارچوب‌های مهندسی پرامپت و برنامه‌ریزی وظیفه برای مدل‌های زبانی به سرعت در حال تحول است و به تعدادی رویکرد امیدوارکننده منجر شده است. اگرچه این لیست جامع نیست، اما این موارد زیر چندتا از محبوب‌ترین چارچوب‌ها و تکنیک‌های استدلالی در زمان انتشار این مقاله هستند:

- **ReAct**، یک چارچوب مهندسی پرامپت که یک استراتژی فرآیند تفکر برای مدل‌های زبانی فراهم می‌کند تا در مورد پرس‌وجوی کاربر استدلال کنند و اقدام کنند، با یا بدون نمونه‌های درون متنی **(in-context)**. پرامپت‌دهی **ReAct** نشان داده است که عملکرد بهتری نسبت به چندین پایه **SOTA** دارد و اعتمادپذیری و انسان‌پسند بودن **LLM** ها را بهبود می‌بخشد.

- **Chain-of-Thought (CoT)**، یک چارچوب مهندسی پرامپت که از طریق مراحل میانی قابلیت استدلال را فعال می‌کند. روش‌های مختلفی از **CoT** شامل خودهماهنگی **(self-consistency)**، **active-prompt** و **CoT** چندوجهی وجود دارد که هر کدام نقاط قوت و ضعف خاص خود را بسته به کاربرد خاص دارند
- **Tree-of-thoughts (ToT)**، یک چارچوب مهندسی پرامپت که برای وظایف اکتشافی یا پیش‌بینی استراتژیک مناسب است. این چارچوب روی **Chain-of-Thought** گسترش می‌یابد و به مدل اجازه می‌دهد خطوط فکری مختلفی را که به عنوان مراحل میانی برای حل مسئله عمومی با مدل‌های زبانی عمل می‌کنند، اکتشاف کند.

عامل‌ها می‌توانند از یکی از تکنیک‌های استدلالی فوق یا بسیاری از تکنیک‌های دیگر برای انتخاب بهترین اقدام بعدی برای درخواست کاربر داده شده استفاده کنند. به عنوان مثال، عاملی را در نظر بگیرید که برای استفاده از چارچوب ReAct برای انتخاب اقدامات و ابزارهای صحیح برای پرس‌وجوی کاربر برنامه‌ریزی شده است. دنباله رویدادها ممکن است به این شکل باشد:

۱. کاربر پرس‌وجویی به عامل می‌فرستد

۲. عامل دنباله ReAct را شروع می‌کند

۳. عامل پرامپتی به مدل ارائه می‌دهد و از آن می‌خواهد یکی از مراحل بعدی ReAct و خروجی مربوطه را تولید کند:

a. **Question** سوال: سوال ورودی پرس‌وجوی کاربر، که با پرامپت ارائه شده است

b. **Thought** فکر: افکار مدل در مورد اینکه چه کاری باید بعداً انجام دهد

c. **Action** اقدام: تصمیم مدل در مورد اینکه چه اقدامی باید بعداً انجام شود

i. اینجا جایی است که انتخاب ابزار اتفاق می‌افتد

ii. برای مثال، یک اقدام می‌تواند یکی از [پروازها، جستجو، کد، هیچ] باشد، که سه مورد اول نشان‌دهنده ابزار شناخته‌شده‌ای است که مدل می‌تواند انتخاب کند و آخرین مورد نشان‌دهنده " انتخاب هیچ ابزاری" است.

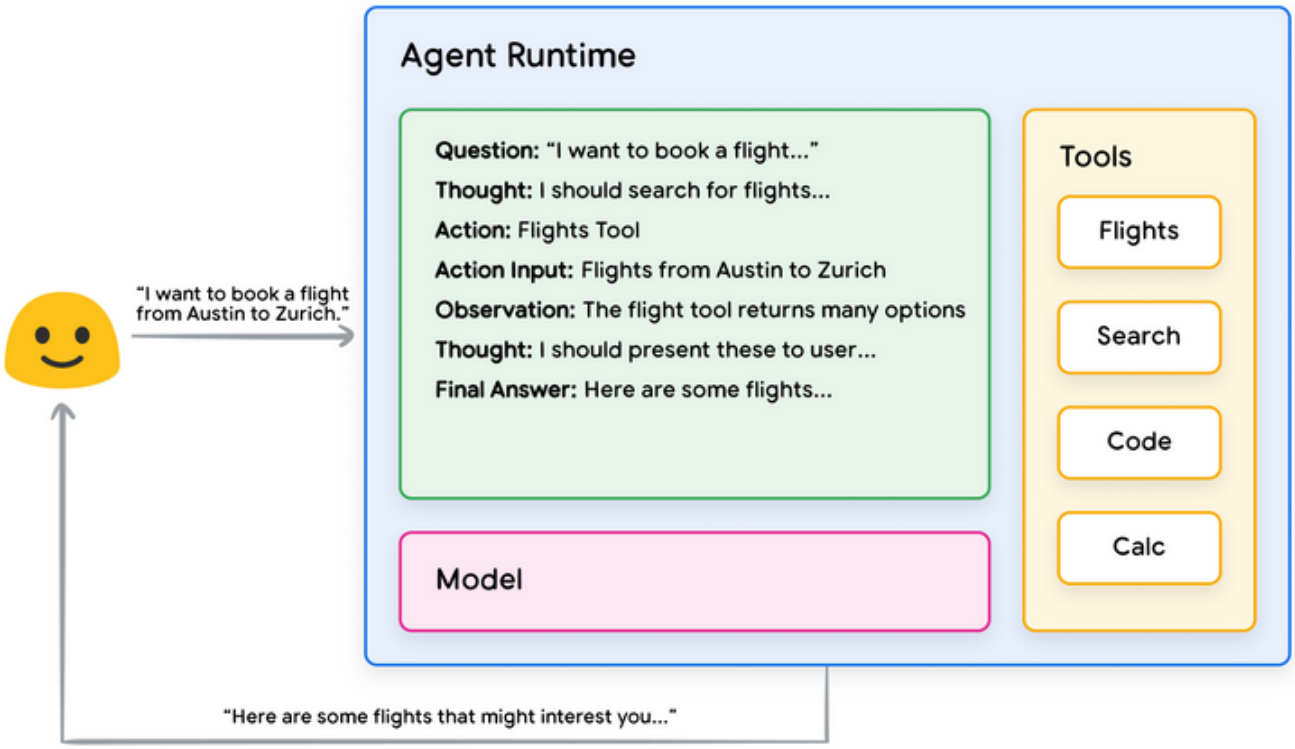
d. **Action input**ورودی اقدام: تصمیم مدل در مورد اینکه چه ورودی‌هایی باید به ابزار داده شود (در صورت وجود)

e. **Observation**مشاهده: نتیجه آن اقدام/زنجیره ورودی آن اقدام

i. این فکر/اقدام/ورودی اقدام/مشاهده می‌تواند **N بار** تکرار شود

f. **Final answer**پاسخ نهایی: پاسخ نهایی مدل برای ارائه به پرس‌وجوی اصلی کاربر

٤. حلقه ReAct به پایان می‌رسد و پاسخ نهایی به کاربر ارائه می‌شود



شکل ۲. مثالی از یک عامل با استدلال ReAct در لایه هماهنگی

همانطور که در شکل ۲ نشان داده شده است، مدل، ابزارها و پیکربندی عامل با هم کار می‌کنند تا پاسخی مستدل و مختصر بر اساس پرس‌وجوی اصلی کاربر ارائه دهند. در حالی که مدل ممکن بود بر اساس دانش قبلی خود پاسخی را حدس بزند (توهمی) ، اینجا از یک ابزار (پروازها) برای جستجوی اطلاعات خارجی بلادرنگ استفاده کرده است. این اطلاعات اضافی به مدل ارائه شد، به آن اجازه داد تا تصمیم آگاهانه‌تری بر اساس داده‌های بروز واقعی بگیرد و این اطلاعات را به کاربر خلاصه کند.

به طور خلاصه، کیفیت پاسخ‌های عامل را می‌توان مستقیماً به توانایی مدل در استدلال و عمل در مورد این وظایف مختلف، از جمله توانایی انتخاب ابزارهای مناسب و اینکه ابزار به چه خوبی تعریف شده است، مرتبط دانست. مانند یک آشپز که غذایی با مواد تازه می‌سازد و به بازخورد مشتری توجه دارد، عامل‌ها به استدلال صحیح و اطلاعات قابل اعتماد متکی هستند تا نتایج بهینه ارائه دهند. در بخش بعدی، ما به بررسی روش‌های مختلفی که عامل‌ها از طریق آنها به داده‌های تازه متصل می‌شوند، خواهیم پرداخت.

ابزارها: کلیدهای ما به دنیای بیرون

اگرچه مدل‌های زبانی در پردازش اطلاعات بسیار ماهرند، اما توانایی مستقیم درک و تأثیرگذاری بر دنیای واقعی را ندارند. این موضوع مفید بودن آنها را در موقعیت‌هایی که نیاز به تعامل با سیستم‌ها یا داده‌های خارجی دارند، محدود می‌کند. این بدان معناست که، به نوعی، یک مدل زبانی فقط به اندازه‌ای خوب است که از داده‌های آموزشی خود یاد گرفته است. اما صرف‌نظر از اینکه چقدر داده به یک مدل وارد کنیم، آنها همچنان از توانایی بنیادی برای تعامل با دنیای خارجی محروم هستند.

پس چگونه می‌توانیم مدل‌های خود را برای تعامل بلادرنگ و آگاه به زمینه با سیستم‌های خارجی توانمند کنیم؟ توابع، افزونه‌ها (Extensions)، ذخیره‌سازهای داده (Data Stores) و افزون‌ها (Plugins) همه روش‌هایی برای فراهم کردن این قابلیت حیاتی به مدل هستند .اگرچه آنها با نام‌های مختلفی شناخته می‌شوند، ابزارها چیزی هستند که پیوندی بین مدل‌های بنیادی ما و دنیای بیرون ایجاد می‌کنند. این پیوند به سیستم‌ها و داده‌های خارجی اجازه می‌دهد تا عامل ما بتواند طیف گسترده‌تری از وظایف را انجام دهد و این کار را با دقت و قابلیت اطمینان بیشتری انجام دهد. به عنوان مثال، ابزارها می‌توانند به عامل‌ها اجازه دهند تنظیمات خانه هوشمند را انجام دهند، تقویم‌ها را به‌روز کنند، اطلاعات کاربر را از یک پایگاه داده دریافت کنند یا بر اساس مجموعه‌ای از دستورالعمل‌های مشخص ایمیل ارسال کنند.

تا تاریخ انتشار این مقاله (سپتامبر ۲۰۲٤)، سه نوع ابزار اصلی وجود دارد که مدل‌های گوگل می‌توانند با آنها تعامل داشته باشند: افزونه‌ها، توابع و ذخیره‌سازهای داده. با مجهز کردن عامل‌ها به ابزارها، ما پتانسیل بزرگی را برای آنها فعال می‌کنیم تا نه تنها دنیا را درک کنند، بلکه بر آن تأثیر بگذارند، درها را به مجموعه‌ای از کاربردها و امکانات جدید باز می‌کنیم.

افزونه ها

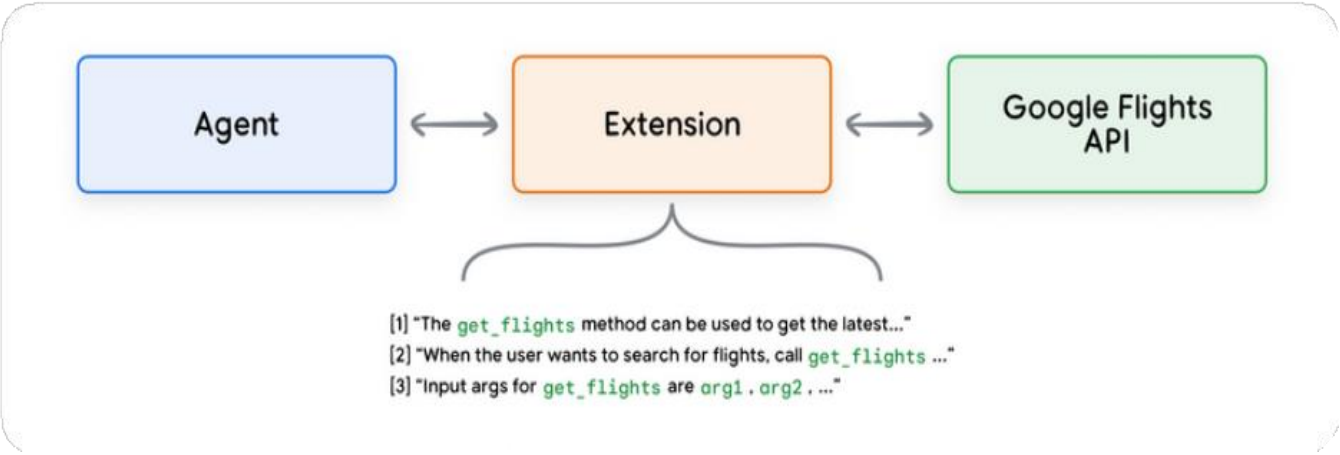
ساده‌ترین راه برای درک افزونه‌ها این است که آنها را به عنوان پلی در نظر بگیرید که شکاف بین یک API و یک عامل را به روشی استاندارد پر می‌کند و به عامل‌ها اجازه می‌دهد تاAPI ها را بدون توجه به پیاده‌سازی زیربنایی آنها به راحتی اجرا کنند. بیایید فرض کنیم که شما یک عامل ساخته‌اید که هدفش کمک به کاربران برای رزرو پروازها است. شما می‌دانید که می‌خواهید از API Google Flights برای دریافت اطلاعات پرواز استفاده کنید، اما مطمئن نیستید که چگونه عامل خود را برای تماس با این نقطه پایانی API تنظیم کنید .

شکل ۳. عاملها چگونه با API ها بیرونی تعامل می کنند؟

یک رویکرد ممکن است این باشد که کد سفارشی پیاده‌سازی کنید که پرس‌وجوی ورودی کاربر را بگیرد، پرس‌وجو را برای اطلاعات مربوطه تجزیه کند و سپس تماس API را انجام دهد. به عنوان مثال، در یک مورد استفاده از رزرو پرواز، یک کاربر ممکن است بگوید "می‌خواهم یک پرواز از آستین به زوریخ رزرو کنم." در این سناریو، راه‌حل کد سفارشی ما باید "آستین" و "زوریخ" را به عنوان موجودیت‌های مرتبط از پرس‌وجوی کاربر استخراج کند قبل از اینکه سعی کند تماس API را انجام دهد. اما اگر کاربر بگوید "می‌خواهم یک پرواز به زوریخ رزرو کنم" و هیچ شهر مبدا ارائه ندهد چطور؟ تماس API بدون داده‌های لازم شکست خواهد خورد و کد بیشتری باید پیاده‌سازی شود تا موارد لبه و گوشه را مدیریت کند. این رویکرد مقیاس‌پذیر نیست و می‌تواند در هر سناریویی که خارج از کد سفارشی پیاده‌سازی شده باشد به راحتی خراب شود.

یک رویکرد مقاومتر استفاده از یک افزونه است. یک افزونه شکاف بین یک عامل و یک API را با این روش پر می‌کند:

۱. آموزش دادن به عامل چگونگی استفاده از نقطه پایانی API با استفاده از نمونه‌ها.
۲. آموزش دادن به عامل اینکه چه آرگومان‌ها یا پارامترهایی برای تماس موفق با نقطه پایانی API نیاز است.



شکل ٤. افزونه ها عاملها را به API های بیرونی متصل می‌کنند

افزونه‌ها می‌توانند به‌صورت مستقل از عامل ساخته شوند، اما باید بخشی از پیکربندی عامل ارائه شوند. عامل در زمان اجرا از مدل و نمونه‌ها استفاده می‌کند تا تصمیم بگیرد کدام افزونه، در صورت وجود، برای حل پرس‌وجوی کاربر مناسب است. این موضوع یکی از نقاط قوت کلیدی افزونه‌ها، یعنی نمونه‌های نوع‌بندی شده داخلی آنها، را برجسته می‌کند که به عامل اجازه می‌دهد افزونه مناسب را برای وظیفه به‌صورت پویا انتخاب کند .

شکل ٥. رابطه یک‌به‌چند بین عامل‌ها، افزونه‌ها وAPI ها

این موضوع را به همان روشی که یک توسعه‌دهنده نرم‌افزار تصمیم می‌گیرد کدام نقاط پایانی API را هنگام حل و ساختن راه‌حل برای مشکل کاربر استفاده کند، در نظر بگیرید. اگر کاربر بخواهد یک پرواز رزرو کند، توسعه‌دهنده ممکن است از API Google Flights استفاده کند. اگر کاربر بخواهد بداند نزدیکترین قهوه‌خانه نسبت به موقعیت فعلی‌اش کجاست، توسعه‌دهنده ممکن است از API Google Maps استفاده کند. به همین ترتیب، پشته عامل/مدل از مجموعه‌ای از افزونه‌های شناخته‌شده استفاده می‌کند تا تصمیم بگیرد کدام یک بهترین تناسب را برای پرس‌وجوی کاربر دارد. اگر می‌خواهید افزونه‌ها را در عمل ببینید، می‌توانید آنها را در برنامه Gemini تست کنید. برای این کار به تنظیمات افزونه‌ها بروید و سپس هر کدام که می‌خواهید تست کنید فعال کنید. به عنوان مثال، می‌توانید افزونه Google Flights را فعال کنید و سپس از Gemini بپرسید "پروازهایی از آستین به زوریخ که هفته آینده حرکت می‌کنند را به من نشان بده".

نمونه‌های افزونه

برای ساده‌سازی استفاده از افزونه‌ها، گوگل برخی افزونه‌های آماده ارائه می‌دهد که می‌توانند به سرعت به پروژه شما وارد شوند و با حداقل پیکربندی استفاده شوند. به عنوان مثال، افزونه تفسیر کد در قطعه ۱ به شما امکان می‌دهد کد پایتون را از یک توصیف زبان طبیعی تولید و اجرا کنید.

Python

```
import vertexai
import pprint



PROJECT_ID = "YOUR_PROJECT_ID"
REGION = "us-central1"

vertexai.init(project=PROJECT_ID, location=REGION)



from vertexai.preview.extensions import Extension


extension_code_interpreter = Extension.from_hub("code_interpreter")
CODE_QUERY = """Write a python method to invert a binary tree in O(n) time."""



response = extension_code_interpreter.execute(
operation_id = "generate_and_execute",
operation_params = {"query": CODE_QUERY}
)



print("Generated Code:")
pprint.pprint({response['generated_code']})



# The above snippet will generate the following code.
``` Generated Code: class TreeNode:



  def __init__(self, val=0, left=None, right=None):
    self.val = val
    self.left = left
    self.right = right
```

ادامه در صفحه بعد...


```

Python

```
def invert_binary_tree(root):
    """
    Inverts a binary tree.
    Args:
        root: The root of the binary tree.
    Returns:
        The root of the inverted binary tree.
    """



    if not root:
        return None



    # Swap the left and right children recursively
    root.left, root.right =
invert_binary_tree(root.right), invert_binary_tree(root.left)



    return root



# Example usage: # Construct a
sample binary tree root =
TreeNode(4)      root.left      =
TreeNode(2)     root.right      =
TreeNode(7)    root.left.left    =
TreeNode(1)   root.left.right   =
TreeNode(3)   root.right.left   =
TreeNode(6)  root.right.right   =
TreeNode(9)



# Invert the binary tree
inverted_root = invert_binary_tree(root)
```

قطعه ۱. افزونه تفسیر کد (Code Interpretter) می‌تواند کد پایتونی را تولید و اجرا کند

به طور خلاصه، افزونه‌ها راهی برای عامل‌ها فراهم می‌کنند تا به روش‌های مختلفی دنیای بیرون را درک نموده، تعامل کنند و بر آن تأثیر بگذارند. انتخاب و فراخوانی این افزونه‌ها با استفاده از مثالهای نمونه هدایت می‌شود، که همه آنها بخشی از پیکربندی افزونه (Extension) تعریف شده‌اند.

توابع

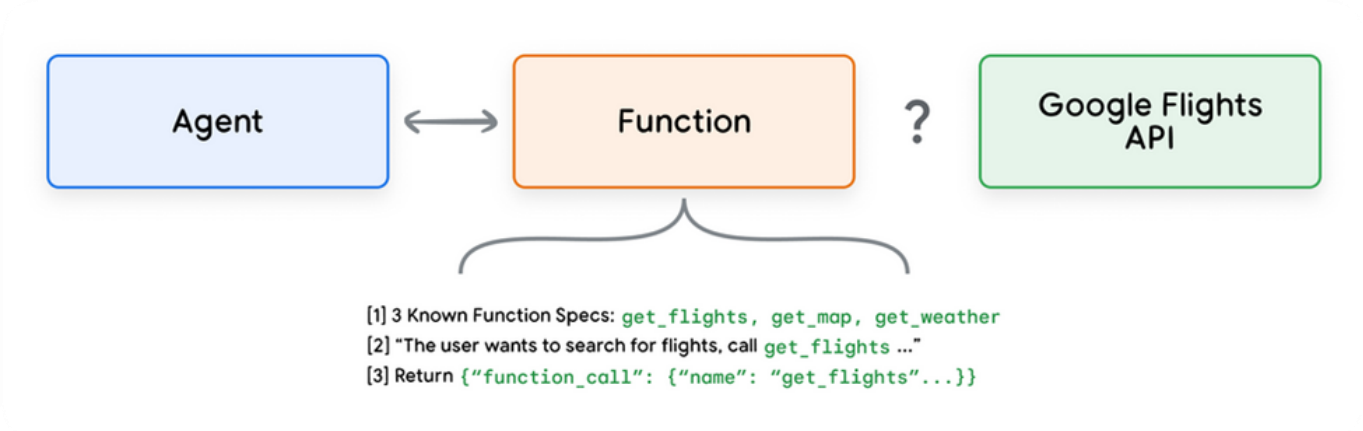
در دنیای مهندسی نرم‌افزار، توابع به عنوان ماژول‌های خودکفا تعریف می‌شوند که وظیفه خاصی را انجام می‌دهند و در صورت نیاز می‌توانند مجدداً استفاده شوند. وقتی یک توسعه‌دهنده نرم‌افزار (software developer) یک برنامه می‌نویسد، اغلب توابع متعددی برای انجام وظایف مختلف ایجاد می‌کند. آنها همچنین منطق را برای زمان فراخوانی function_a در مقابل function_b ، و همچنین ورودی‌ها و خروجی‌های مورد انتظار تعریف می‌کنند .

در اینجا نیز توابع به طور مشابه در دنیای عامل‌ها کار می‌کنند، اما می‌توانیم توسعه‌دهنده نرم‌افزار را با یک مدل جایگزین کنیم. یک مدل می‌تواند مجموعه‌ای از توابع شناخته‌شده بگیرد و تصمیم بگیرد که چه زمانی از هر تابع استفاده کند و چه آرگومان‌هایی تابع بر اساس مشخصات آن نیاز دارد. توابع از افزونه‌ها در چندین جنبه متفاوت هستند، به‌ویژه:

۱. یک مدل در خروجی یک تابع و آرگومان‌های آن را تولید می‌کند، اما تماس API زنده‌ای انجام نمی‌دهد.

۲. توابع در سمت کلاینت اجرا می‌شوند، در حالی که افزونه‌ها در سمت عامل اجرا می‌شوند .

با استفاده مجدد از مثالGoogle Flights ، یک تنظیم ساده برای توابع ممکن است شبیه مثال در شکل V باشد.



شکل ۷. توابع چگونه باAPI های خارجی تعامل می‌کنند؟

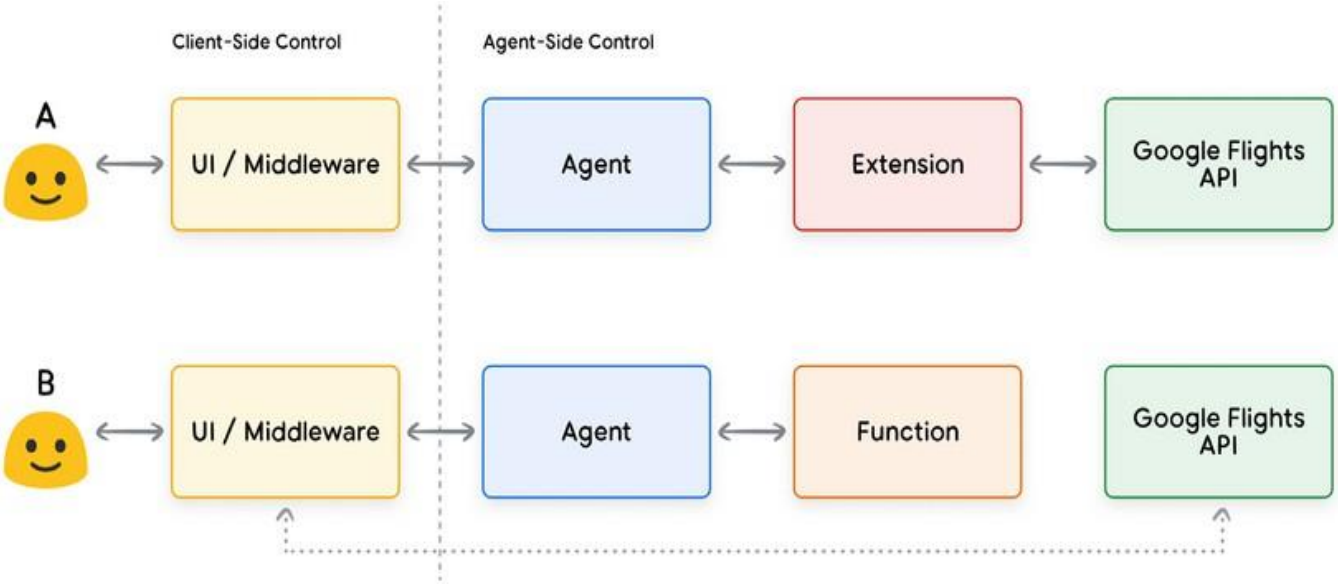
توجه داشته باشید که تفاوت اصلی اینجا این است که نه تابع و نه عامل مستقیماً با API Google Flights تعامل نمی‌کنند. پس تماس API واقعاً چگونه اتفاق می‌افتد؟

با توابع، منطق و اجرای تماس نقطه پایانی API واقعی از عامل دور انداخته شده و به برنامه کلاینت‌ساید بازمی‌گردد، همان‌طور که در شکل ۸ و شکل ۹ زیر نشان داده شده است. این موضوع به توسعه‌دهنده کنترل دقیق‌تری بر جریان داده‌ها در برنامه می‌دهد. دلایل زیادی وجود دارد که یک توسعه‌دهنده ممکن است ترجیح دهد از توابع به جای افزونه‌ها استفاده کند، اما چند مورد استفاده رایج عبارتند از:

- تماس‌های API باید در لایه دیگری از پشته برنامه، خارج از جریان مستقیم معماری عامل انجام شوند (مثلاً یک سیستم میان‌افزار، یک چارچوب فرانت‌اند و غیره)
- محدودیت‌های امنیتی یا احراز هویت که از عامل جلوگیری می‌کنند تماس مستقیم با API داشته باشد (مثلاً API به سمت اینترنت در معرض دید نیست یا توسط زیرساخت عامل قابل دسترسی نیست)
- محدودیت‌های زمان‌بندی یا ترتیب عملیات که از عامل جلوگیری می‌کنند تماس‌های API را به صورت بلادرنگ انجام دهد (یعنی عملیات دسته‌ای، نیاز به بررسی عامل انسانی در حلقه و غیره)

- نیاز به منطق تبدیل داده‌های اضافی برای پاسخ API که عامل نمی‌تواند انجام دهد. به عنوان مثال، یک نقطه پایانی API را در نظر بگیرید که مکانیزم فیلتر کردن برای محدود کردن تعداد نتایج بازگشتی ارائه نمی‌دهد. استفاده از توابع در سمت کلاینت به توسعه‌دهنده فرصت‌های اضافی برای انجام این تبدیل‌ها می‌دهد
- توسعه‌دهنده می‌خواهد بر توسعه عامل بدون مستقر کردن زیرساخت‌های اضافی برای نقاط پایانی API تکرار کند (یعنی فراخوانی تابع می‌تواند مثلAPI "stubbing" ها عمل کند)

در حالی که تفاوت در معماری داخلی بین دو رویکرد ظریف است همان‌طور که در شکل ۸ دیده می‌شود، کنترل اضافی و وابستگی جدا شده به زیرساخت خارجی فراخوانی تابع را به گزینه‌ای جذاب برای توسعه‌دهنده تبدیل می‌کند



شکل ۸. تفکیک کنترل سمت کلاینت در مقابل سمت عامل برای افزونه‌ها و فراخوانی تابع

موارد استفاده

یک مدل می‌تواند برای فراخوانی توابع استفاده شود تا جریان‌های اجرایی پیچیده سمت کلاینت را برای کاربر نهایی مدیریت کند، جایی که توسعه‌دهنده عامل ممکن است نخواهد که مدل زبانی مدیریت اجرای API را بر عهده داشته باشد (مانند حالت افزونه‌ها). بیایید مثال زیر را در نظر بگیریم که در آن یک عامل به عنوان یک کنسیرژ سفر آموزش داده می‌شود تا با کاربرانی که می‌خواهند سفرهای تعطیلات را رزرو کنند تعامل کند. هدف این است که عامل لیستی از شهرها تولید کند که می‌توانیم در برنامه میان‌افزار خود برای دانلود تصاویر، داده‌ها و غیره برای برنامه‌ریزی سفر کاربر استفاده کنیم. یک کاربر ممکن است چیزی شبیه به این بگوید:

می‌خواهم با خانواده‌ام یک سفر اسکی داشته باشم اما مطمئن نیستم کجا بروم.
I'd like to take a ski trip with my family but I'm not sure where to go.

در یک پرس‌وجوی معمولی به مدل، خروجی ممکن است شبیه به موارد زیر باشد:

حتماً، اینجا لیستی از شهرهایی است که می‌توانید برای سفرهای اسکی خانوادگی در نظر بگیرید:

- Crested Butte, Colorado, USA
- Whistler, BC, Canada
- Zermatt, Switzerland

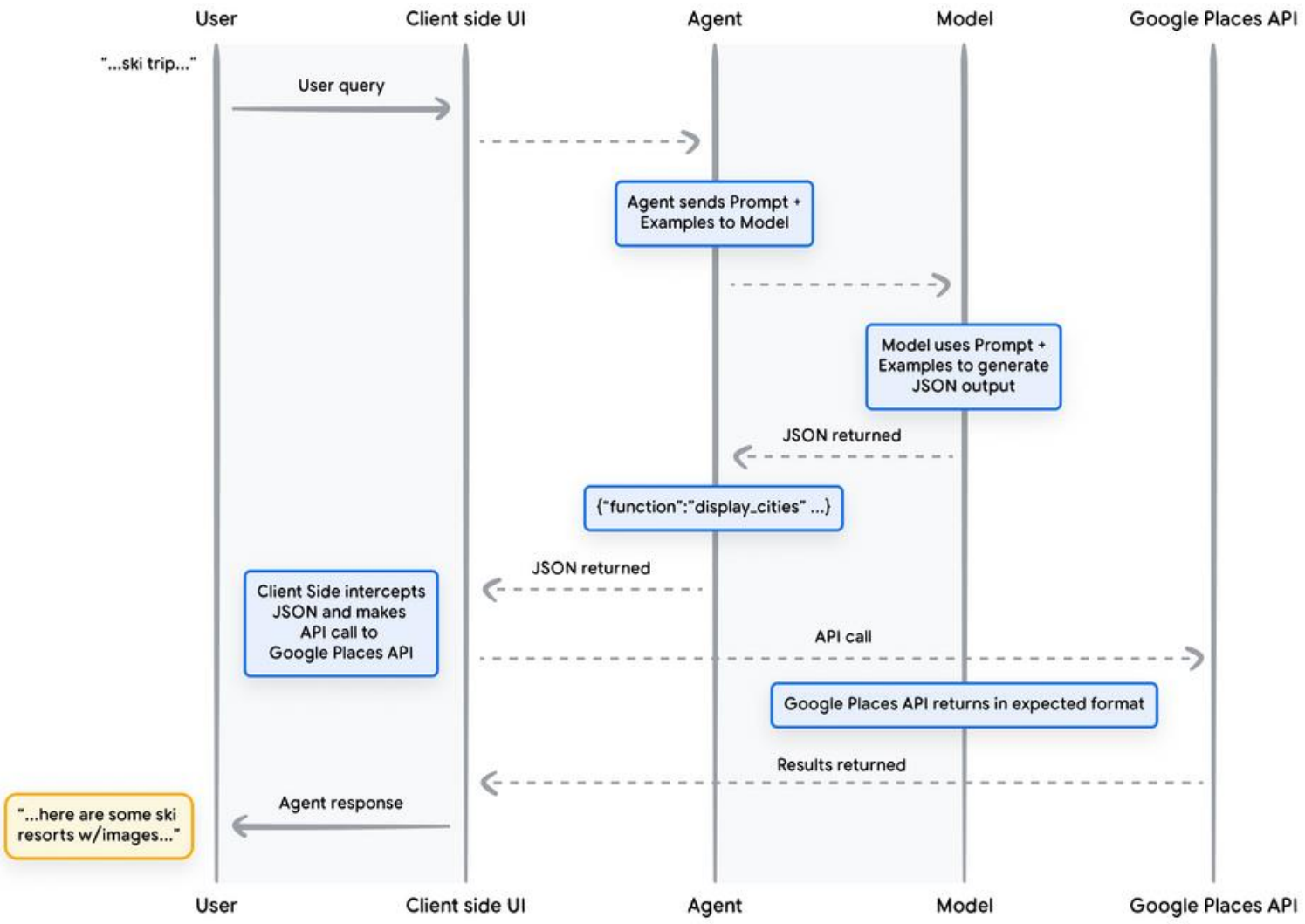
اگرچه خروجی بالا شامل داده‌هایی است که نیاز داریم (نام شهرها)، اما قالب آن برای پارس کردن (تحلیل گرامری) ایده‌آل نیست .با استفاده از فراخوانی تابع، می‌توانیم به مدل آموزش دهیم که این خروجی را در یک سبک ساختارمند (مانند JSON) قالب‌بندی کند که برای سیستم دیگری برای پارس کردن راحت‌تر باشد. با توجه به همان پرس‌وجوی ورودی از کاربر، یک خروجی JSON نمونه از یک تابع ممکن است شبیه قطعه ۵ باشد.

Unset

```
function_call {
name: "display_cities"
args: {
    "cities": ["Crested Butte", "Whistler", "Zermatt"],
    "preferences": "skiing"
    }
}
```

قطعه ۵. نمونه payload فراخوانی تابع برای نمایش لیستی از شهرها و ترجیحات کاربر

این payload JSON توسط مدل تولید می‌شود و سپس به سرور سمت کلاینت ما ارسال می‌شود تا هر کاری که می‌خواهیم با آن انجام دهیم. در این مورد خاص، ما از API Google Places استفاده خواهیم کرد تا شهرهای ارائه‌شده توسط مدل را بگیریم و تصاویر را جستجو کنیم و سپس آنها را به صورت محتوای غنی فرمت‌شده به کاربر ارائه دهیم. این دیاگرام توالی در شکل ۹ را در نظر بگیرید که تعامل فوق را به صورت مرحله به مرحله نشان می‌دهد.



شکل ۹. دیاگرام توالی که چرخه عمر یک فراخوانی تابع را نشان می‌دهد

نتیجه مثال شکل ۹ این است که مدل برای "پر کردن جاهای خالی" با پارامترهای مورد نیاز برای UI سمت کلاینت استفاده می‌شود تا تماس با API Google Places انجام شود. UI سمت کلاینت با استفاده از پارامترهای ارائه‌شده توسط مدل در تابع بازگشتی، تماس API واقعی را مدیریت می‌کند. این فقط یک مورد استفاده برای فراخوانی تابع است، اما سناریوهای دیگری نیز وجود دارد که می‌توان در نظر گرفت:

- می‌خواهید یک مدل زبانی پیشنهادی برای تابعی که می‌توانید در کد خود استفاده کنید ارائه دهد، اما نمی‌خواهید اعتبارنامه‌ها (credentials) را در کد خود شامل کنید. چون فراخوانی تابع، تابع را اجرا نمی‌کند، نیازی نیست که اعتبارنامه‌ها را به همراه اطلاعات تابع در کد خود قرار دهید.
- شما عملیات غیرهمزمانی اجرا می‌کنید که می‌تواند بیش از چند ثانیه طول بکشد. این سناریوها با فراخوانی تابع به خوبی کار می‌کنند زیرا عملیاتی غیرهمزمان است.
- می‌خواهید توابع را روی دستگاهی اجرا کنید که متفاوت از سیستم تولیدکننده تماس‌های تابع و آرگومان‌های آن است.

یک نکته کلیدی که باید در مورد توابع به یاد داشته باشید این است که آنها به توسعه‌دهنده کنترل بسیار بیشتری بر روی نه تنها اجرای تماس‌هایAPI ، بلکه کل جریان داده‌ها در برنامه به عنوان یک کل می‌دهند. در مثال شکل ۹، توسعه‌دهنده انتخاب کرد که اطلاعات API را به عامل بازگرداند زیرا برای اقدامات آینده عامل مرتبط نبود. با این حال، بر اساس معماری برنامه، ممکن است بازگرداندن داده‌های تماس API خارجی به عامل معقول باشد تا استدلال، منطق و انتخاب‌های عملیاتی آینده را تحت تأثیر قرار دهد. در نهایت، انتخاب اینکه چه چیزی برای برنامه خاص مناسب است به توسعه‌دهنده برنامه بستگی دارد.

کد نمونه تابع

برای رسیدن به خروجی بالا از سناریوی تعطیلات اسکی ما، بیایید هر یک از اجزا را برای کار با مدل gemini-1.5-flash-001 خود بسازیم .

ابتدا، تابع display_cities خود را به عنوان یک روش ساده پایتون تعریف می‌کنیم.

Python

```python
def display_cities(cities: list[str], preferences: Optional[str] = None):
    """Provides a list of cities based on the user's search query and preferences.

    Args:
        preferences (str): The user's preferences for the search, like skiing,
        beach, restaurants, bbq, etc.
        cities (list[str]): The list of cities being recommended to the user.

    Returns:
        list[str]: The list of cities being recommended to the user.
    """

    return cities
```

قطعه ۶. مثال نمونه متد پایتون برای تابعی که لیستی از شهرها را نمایش می‌دهد.

بعد، ما مدل خود را مقداردهی می‌کنیم، ابزار را می‌سازیم و سپس پرس‌وجوی کاربر و ابزارها را به مدل می‌فرستیم. اجرای کد زیر منجر به خروجی می‌شود که در انتهای قطعه کد دیده می‌شود.

Python

```
from vertexai.generative_models import GenerativeModel, Tool, FunctionDeclaration

model = GenerativeModel("gemini-1.5-flash-001")

display_cities_function = FunctionDeclaration.from_func(display_cities)
tool = Tool(function_declarations=[display_cities_function])

message = "I'd like to take a ski trip with my family but I'm not sure where
to go."

res = model.generate_content(message, tools=[tool])

print(f"Function Name: {res.candidates[0].content.parts[0].function_call.name}")
print(f"Function Args: {res.candidates[0].content.parts[0].function_call.args}")

> Function Name: display_cities
> Function Args: {'preferences': 'skiing', 'cities': ['Aspen', 'Vail',
'Park City']}
```

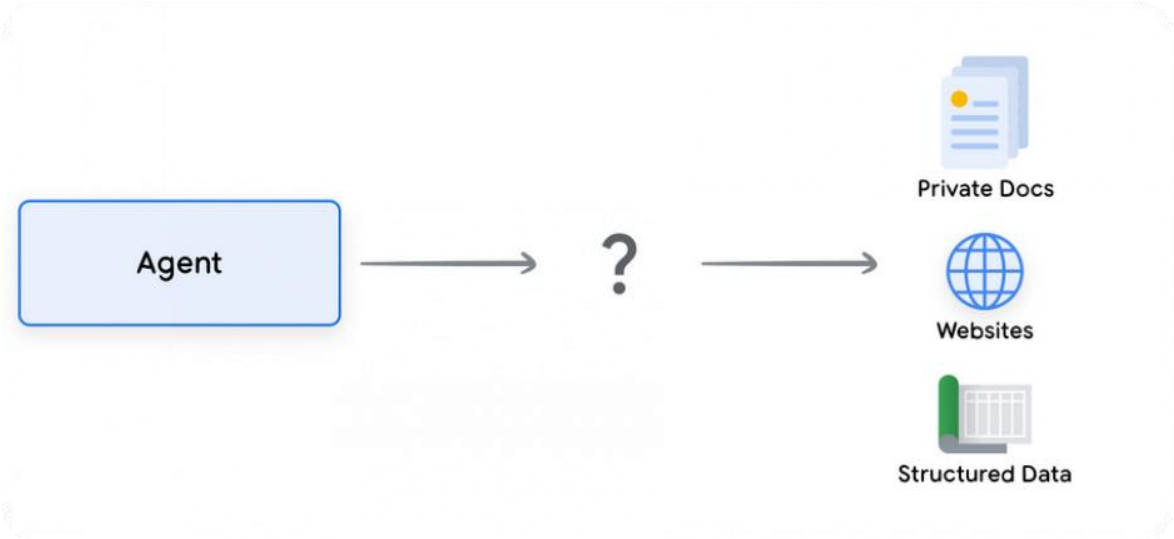
قطعه ۷. ساخت یک ابزار، ارسال به مدل با یک پرس‌وجوی کاربر و اجازه دادن به فراخوانی تابع برای انجام شدن

به طور خلاصه، توابع یک چارچوب ساده ارائه می‌دهند که به توسعه‌دهندگان برنامه، کنترل دقیق‌تری بر روی جریان داده و اجرای سیستم می‌دهند، در حالی که به طور مؤثر از عامل/مدل برای تولید ورودی‌های حیاتی استفاده می‌کنند. توسعه‌دهندگان می‌توانند به‌صورت انتخابی تصمیم بگیرند که آیا عامل را "در حلقه" نگه دارند یا بازگرداندن داده‌های خارجی، یا بر اساس نیازهای معماری برنامه خاص آن را حذف کنند.

ذخیره‌سازهای داده

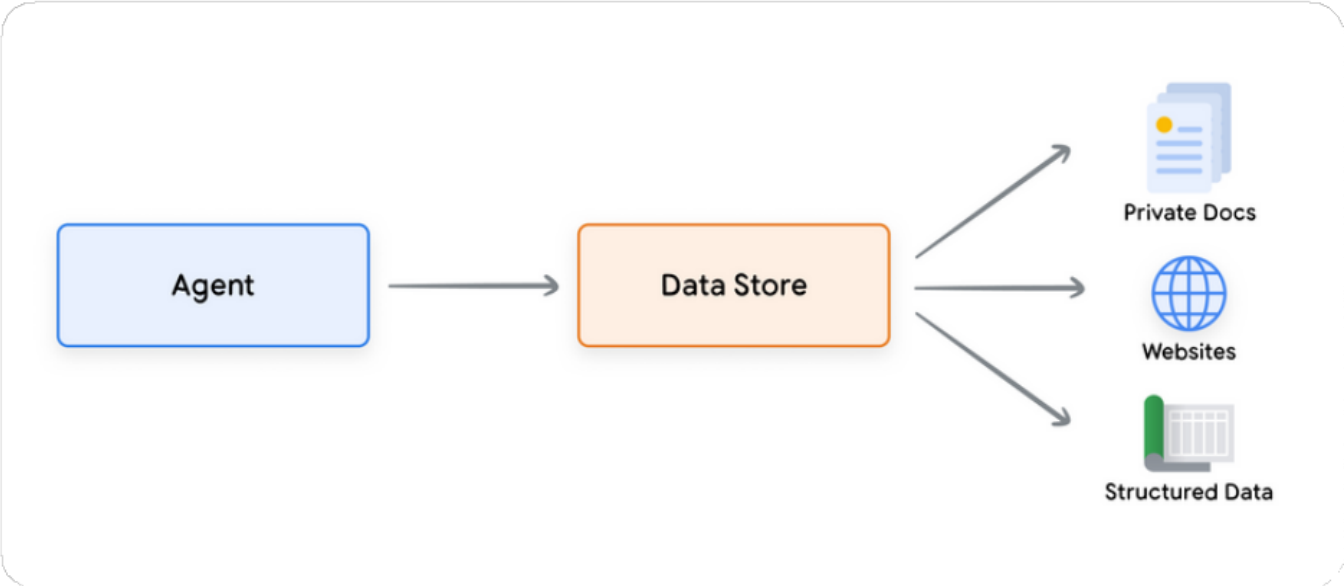
یک مدل زبانی را مانند کتابخانه‌ای عظیم از کتاب‌ها تصور کنید که شامل داده‌های آموزشی آن است. اما برخلاف یک کتابخانه که به طور مداوم جلد جدیدی به دست می‌آورد، این کتابخانه ثابت می‌ماند و فقط دانشی را که در ابتدا با آن آموزش دیده است نگه می‌دارد. این موضوع چالشی ایجاد می‌کند، زیرا دانش دنیای واقعی به طور مداوم در حال تغییر است. ذخیره‌سازهای داده (Data Stores) این محدودیت را با فراهم کردن دسترسی به اطلاعات پویاتر و به‌روزتر برطرف می‌کنند و مطمئن می‌شوند که پاسخ‌های مدل در واقعیت و مرتبط بودن متمرکز باقی می‌مانند.

یک سناریوی رایج را در نظر بگیرید که در آن یک توسعه‌دهنده ممکن است نیاز داشته باشد مقدار کمی داده اضافی به یک مدل ارائه دهد، شاید به شکل صفحه‌گسترده‌های اکسل یاPDF ها.



شکل ۱۰. عامل‌ها چگونه با داده‌های ساختاریافته و بدون ساختار تعامل می‌کنند؟

ذخیره‌سازهای داده به توسعه‌دهندگان اجازه می‌دهند داده‌های اضافی را به شکل اصلی خود در اختیار عامل قرار دهند، که نیاز به تبدیلات زمان‌بر داده، بازآموزش مدل یا تنظیم دقیق آن را از بین می‌برد. ذخیره‌ساز داده، سند ورودی را به مجموعه‌ای از تعبیه‌سازی‌های پایگاه داده برداری (vector database embeddings) تبدیل می‌کند که عامل می‌تواند از آنها برای استخراج اطلاعات مورد نیاز به منظور تکمیل عملیات بعدی یا پاسخ به کاربر استفاده کند.



.شکل ۱۱. ذخیره‌سازهای داده، عامل‌ها را به منابع داده بلادرنگ جدید از انواع مختلف متصل می‌کنند

اجرا و کاربرد

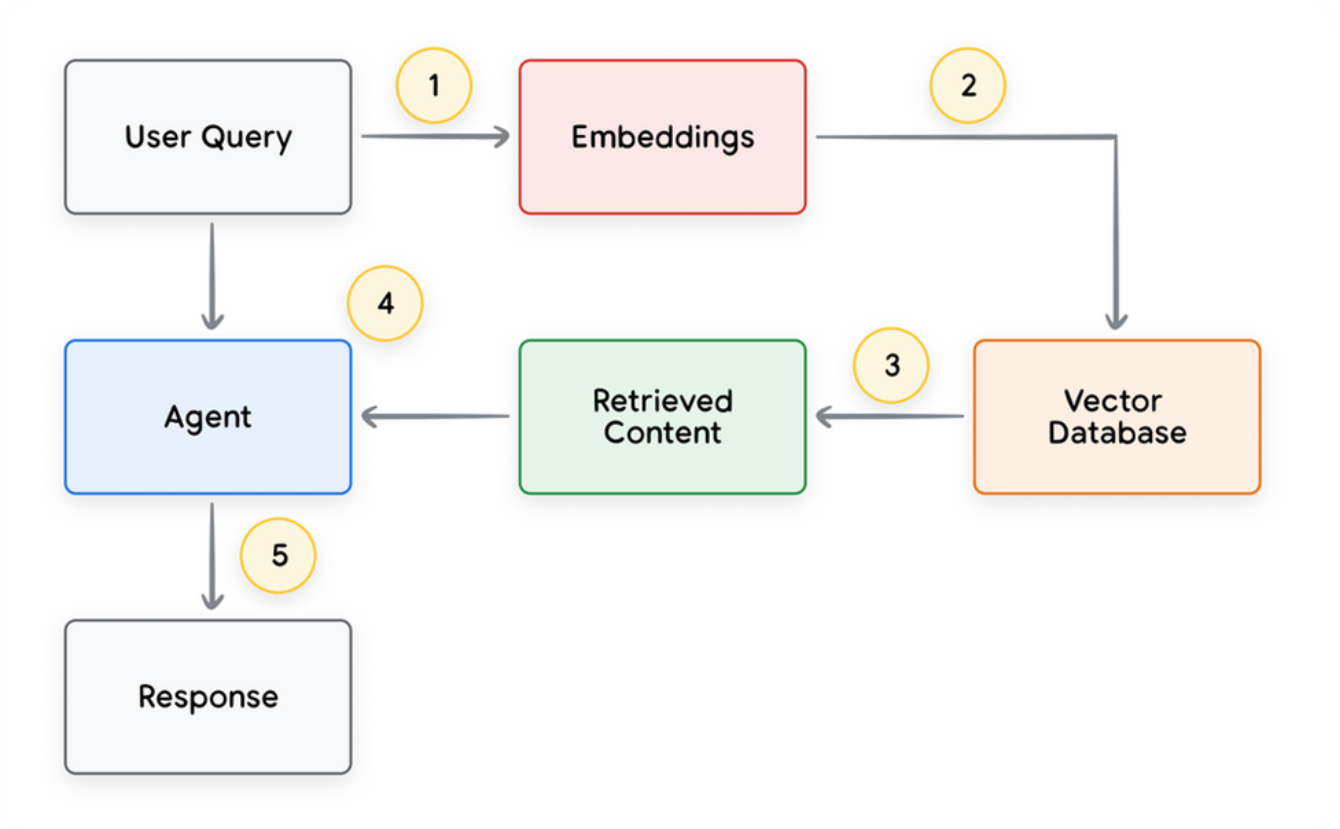
در زمینه عامل‌های هوش مصنوعی تولیدی، ذخیره‌سازهای داده معمولاً به عنوان یک *پایگاه داده برداری* پیاده‌سازی می‌شوند که توسعه‌دهنده می‌خواهد عامل در زمان اجرا به آن دسترسی داشته باشد. اگرچه ما به طور عمیق به پایگاه‌های داده برداری نمی‌پردازیم، اما نکته کلیدی که باید درک کرد این است که آنها داده‌ها را به شکل تعبیه‌های برداری یعنی نوعی بردار یا نمایش ریاضی از داده‌های ارائه‌شده با ابعاد بالا، ذخیره می‌کنند. یکی از پرکاربردترین نمونه‌های استفاده از ذخیره‌سازهای داده با مدل‌های زبانی در این روزها، پیاده‌سازی برنامه‌های مبتنی بر تولید تقویت‌شده با بازیابی **(RAG)** بوده است. این برنامه‌ها به دنبال گسترش عرض و عمق دانش مدل فراتر از داده‌های آموزشی بنیادی هستند، با این هدف که به مدل دسترسی به داده‌ها در قالب‌های مختلفی مانند:

- محتوای وب‌سایت
- داده‌های ساخت‌یافته در قالب‌هایی مانندPDF ، Word Docs، CSV، صفحه‌گسترده‌ها و غیره.
- داده‌های بدون ساختار در قالب‌هایی مانندHTML ، PDF، TXTو غیره.

شکل ۱۲. رابطه یک-به-چند بین عامل‌ها و ذخیره‌سازهای داده که می‌توانند انواع مختلفی از داده‌های پیش‌شاخص‌شده را نمایش دهند

فرآیند زیربنایی برای هر درخواست کاربر و حلقه پاسخ عامل عموماً به شکل دیده شده در شکل ۱۳ مدل‌سازی شده است.

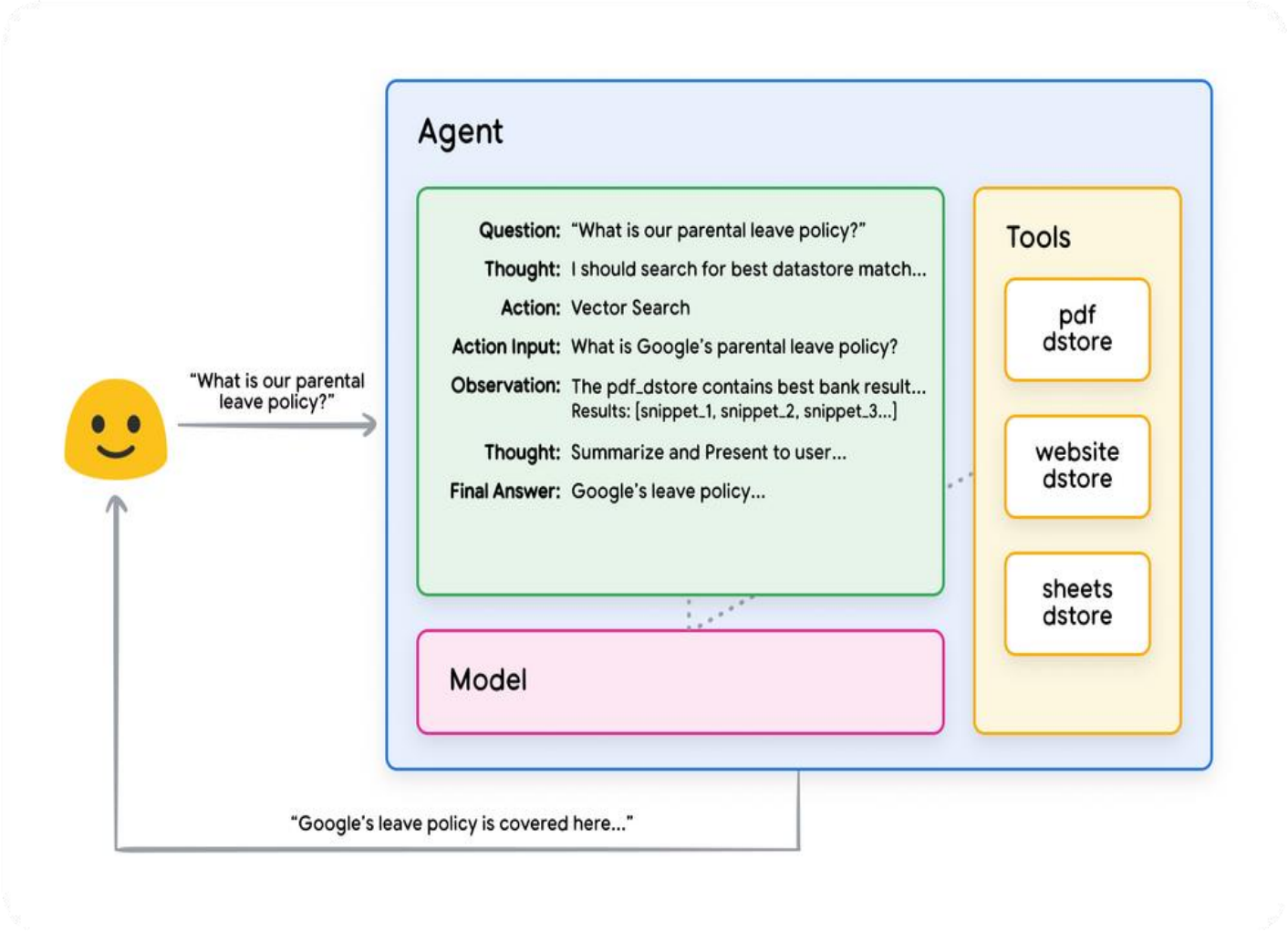
۱. یک پرس‌وجوی کاربر به یک مدل تعبیه‌سازی (embedding model) ارسال می‌شود تا تعبیه‌سازی‌ها برای پرس‌وجو تولید شوند.
۲. سپس تعبیه‌سازی‌های پرس‌وجو (query embeddings) با استفاده از یک الگوریتم منطبق مانند SCaNN با محتوای پایگاه داده برداری تطبیق داده می‌شوند.
۳. محتوای تطبیق‌یافته به شکل متنی از پایگاه داده برداری بازیابی شده و به عامل بازمی‌گردد.
٤. عامل هم پرس‌وجوی کاربر و هم محتوای بازیابی‌شده را دریافت می‌کند، سپس یک پاسخ یا عملی را شکل می‌دهد.
٥. یک پاسخ نهایی به کاربر ارسال می‌شود.



شکل ۱۳. چرخه عمر درخواست کاربر و پاسخ عامل در یک برنامه مبتنی برRAG

نتیجه نهایی یک برنامه است که به عامل اجازه می‌دهد پرس‌وجوی کاربر را با استفاده از جستجوی برداری به یک ذخیره‌ساز داده شناخته‌شده تطبیق دهد، محتوای اصلی را بازیابی کند و آن را به لایه ارکستراسیون و مدل برای پردازش بیشتر ارائه دهد. عمل بعدی ممکن است ارائه پاسخ نهایی به کاربر باشد یا انجام یک جستجوی برداری دیگر برای دقیق‌تر کردن نتایج.

یک نمونه تعامل با یک عامل که RAG را با استدلال/برنامه‌ریزی ReAct پیاده‌سازی می‌کند، می‌تواند در شکل ١٤ دیده شود.



شکل ١٤. نمونه برنامه مبتنی بر RAG با استدلال/برنامه‌ریزیReAct

خلاصه بحث ابزارها

به طور خلاصه، افزونه‌ها، توابع و ذخیره‌سازهای داده چند نوع مختلف از ابزارهای موجود برای استفاده عامل‌ها در زمان اجرا را تشکیل می‌دهند. هر یک هدف خاص خود را دارند و می‌توانند به صورت مستقل یا همراه با یکدیگر، بسته به تصمیم توسعه‌دهنده عامل، استفاده شوند.

	افزونه ها	فراخوانی توابع	ذخیره سازهای داده
اجرا	اجرای سمت-عامل	اجرای سمت-کاربر	اجرای سمت-عامل
مورد استفاده	• توسعه‌دهنده می‌خواهد عامل کنترل تعاملات با نقاط پایانی API را بر عهده داشته باشد. • مفید برای زمانی که از افزونه‌های پیش‌ساخته بومی استفاده می‌شود (مثل جستجوی Vertex ، تفسیر کد و غیره). • برنامه‌ریزی چندمرحله‌ای و فراخوانی API (یعنی عمل بعدی عامل به خروجی‌های عمل قبلی/فراخوانی API وابسته است).	• محدودیت‌های امنیتی یا احراز هویت که از عامل جلوگیری می‌کند تا به طور مستقیم API را فراخوانی کند. • محدودیت‌های زمان‌بندی یا محدودیت‌های ترتیب عملیات که از عامل جلوگیری می‌کنند تا فراخوانی API را به صورت بلادرنگ انجام دهد (یعنی عملیات دسته‌ای، بررسی انسان در حلقه و غیره). • APIکه به سمت اینترنت در معرض دید نیست یا توسط سیستم‌های گوگل قابل دسترسی نیست.	• توسعه‌دهنده می‌خواهد تولید تقویت‌شده با بازیابی (RAG)را با هر یک از انواع داده‌های زیر پیاده‌سازی کند: • محتوای وب‌سایت از دامنه‌ها وURL های پیش‌شاخص‌شده • داده‌های ساخت‌یافته در قالب‌هایی مانندPDF ، Word ، Docs، CSV، صفحه‌گسترده‌ها و غیره. • پایگاه‌های داده رابطه‌ای / غیررابطه‌ای • داده‌های بدون ساختار در قالب‌هایی مانندHTML ، PDF، TXTو غیره.

بهبود عملکرد مدل با یادگیری هدفمند

یک جنبه حیاتی در استفاده مؤثر از مدل‌ها، توانایی آنها در انتخاب ابزارهای مناسب هنگام تولید خروجی است، به ویژه هنگام استفاده از ابزارها به شکل مقیاس پذیر در محیط تولید. در حالی که آموزش عمومی به مدل‌ها کمک می‌کند تا این مهارت را توسعه دهند، سناریوهای دنیای واقعی اغلب نیاز به دانشی فراتر از داده‌های آموزشی دارند. این را می‌توان به عنوان تفاوت بین مهارت‌های پخت و پز پایه‌ای و تسلط بر یک آشپزی خاص در نظر گرفت. هر دو نیاز به دانش پایه‌ای پخت و پز دارند، اما دومی نیاز به یادگیری هدفمند برای نتایج دقیق‌تر دارد.

برای کمک به مدل برای دسترسی به این نوع دانش خاص، چندین رویکرد وجود دارد:

- **In-context learning**: یادگیری درون متنی: این روش یک مدل عمومی را با یک پرامپت، ابزارها و چند مثال کوتاه در زمان استنتاج تجهیز می‌کند که به آن اجازه می‌دهد "در پرواز" یاد بگیرد که چگونه و چه زمانی از آن ابزارها برای یک وظیفه خاص استفاده کند. چارچوب ReAct نمونه‌ای از این رویکرد در زبان طبیعی است.

- **Retrieval-based in-context learning**: یادگیری درون متنی مبتنی بر بازیابی: این تکنیک پرامپت مدل را به طور پویا با مربوطترین اطلاعات، ابزارها و مثال‌های مرتبط با بازیابی از حافظه خارجی پر می‌کند. نمونه‌ای از این موضوع "Example Store" در افزونه‌های Vertex AI یا معماری مبتنی بر RAG ذخیره‌سازهای داده که قبلاً ذکر شد، می‌باشد.

- **Fine-tuning based learning**:یادگیری مبتنی بر تنظیم دقیق: این روش شامل آموزش یک مدل با استفاده از مجموعه داده بزرگتری از نمونه‌های خاص قبل از استنتاج است. این به مدل کمک می‌کند تا قبل از دریافت هرگونه پرس‌وجوی کاربر، زمان و نحوه استفاده از ابزارهای خاص را درک کند.

برای ارائه بینش بیشتر در مورد هر یک از رویکردهای یادگیری هدفمند، بیایید به تمثیل آشپزی خود بازگردیم.

- تصور کنید یک آشپز یک دستورالعمل خاص (پرامپت)، چند ماده اولیه کلیدی (ابزارهای مرتبط) و چند نمونه غذا (مثال‌های کوتاه) از مشتری دریافت کرده است. بر اساس این اطلاعات محدود و دانش عمومی آشپزی خود، آنها باید بفهمند که چگونه "در پرواز" غذایی را آماده کنند که بیشترین تطابق را با دستور العمل و ترجیحات مشتری داشته باشد. این یادگیری درون متنی *(in-context learning)* است.

- حال تصور کنید آشپز ما در یک آشپزخانه است که انباری (حافظه خارجی) پر از انواع مواد اولیه و کتاب‌های آشپزی (مثال‌ها و ابزارها) دارد. آشپز اکنون می‌تواند به طور پویا مواد اولیه و کتاب‌های آشپزی را از انبار انتخاب کند و بهتر با دستور العمل و ترجیحات مشتری هماهنگ شود. این اجازه می‌دهد تا آشپز یک غذای آگاهانه‌تر و ریزتر را با استفاده از هم دانش موجود و هم دانش جدید ایجاد کند. این یادگیری درون متنی مبتنی بر بازیابی *(retrieval-based in-context learning)* است.

- در نهایت، تصور کنید که آشپز خود را به مدرسه فرستاده‌ایم تا یک آشپزی جدید یا مجموعه‌ای از آشپزی‌ها (آموزش قبلی بر روی مجموعه داده بزرگتری از نمونه‌های خاص) یاد بگیرد. این به آشپز اجازه می‌دهد تا با درک عمیق‌تری به دستور العمل‌های مشتریانی که قبلاً ندیده است، رویکرد پیدا کند. این رویکرد برای مواقعی که می‌خواهیم آشپز در حوزه‌های دانش خاص (مثل آشپزی‌های خاص) برتری یابد، عالی است. این یادگیری مبتنی بر تنظیم دقیق *(fine-tuning based learning)* است.

هر یک از این رویکردها مزایا و معایب منحصر به فردی از نظر سرعت، هزینه و تأخیر دارند. با این حال، با ترکیب این تکنیک‌ها در یک چارچوب عامل، می‌توانیم از نقاط قوت مختلف استفاده کرده و نقاط ضعف آنها را به حداقل برسانیم، که این امکان حل‌وسازی قوی‌تر و قابل‌انعطاف‌تر را فراهم می‌کند.

شروع سریع عامل باLangChain

برای ارائه یک نمونه اجرایی واقعی از عملکرد یک عامل، ما یک مثال نمونه اولیه سریع با کتابخانه‌های LangChain و LangGraph خواهیم ساخت. این کتابخانه‌های منبع باز محبوب به کاربران اجازه می‌دهند عامل‌های سفارشی را با "زنجیره‌ای کردن" دنباله‌ای از منطق، استدلال و فراخوانی ابزارها برای پاسخ به پرس‌وجوی کاربر بسازند.

ما از مدل gemini-1.5-flash-001 و برخی ابزارهای ساده برای پاسخ به یک پرس‌وجوی چندمرحله‌ای از کاربر استفاده خواهیم کرد که در قطعه ۸ دیده می‌شود.

ابزارهایی که استفاده می‌کنیم SerpAPI (برای جستجوی گوگل) و Google Places API هستند. پس از اجرای برنامه خود در قطعه ۸، نمونه خروجی را در قطعه ۹ خواهید دید.

Python

```python
from langgraph.prebuilt import create_react_agent
from langchain_core.tools import tool
from langchain_community.utilities import SerpAPIWrapper
from langchain_community.tools import GooglePlacesTool


os.environ["SERPAPI_API_KEY"] = "XXXXX"
os.environ["GPLACES_API_KEY"] = "XXXXX"


@tool
def search(query: str):
    """Use the SerpAPI to run a Google Search."""
    search = SerpAPIWrapper()
    return search.run(query)


@tool
def places(query: str):
    """Use the Google Places API to run a Google Places Query."""
    places = GooglePlacesTool()
    return places.run(query)


model = ChatVertexAI(model="gemini-1.5-flash-001")
tools = [search, places]


query = "Who did the Texas Longhorns play in football last week? What is the
address of the other team's stadium?"


agent = create_react_agent(model, tools)
input = {"messages": [("human", query)]}


for s in agent.stream(input, stream_mode="values"):
    message = s["messages"][-1]
    if isinstance(message, tuple):
        print(message)
    else:
        message.pretty_print()
```

قطعه کد ۸. نمونه عامل مبتنی بر LangChain و LangGraph همراه با ابزارها

Unset

```
================================ Human Message =================================
Who did the Texas Longhorns play in football last week? What is the address
of the other team's stadium?
================================== Ai Message ==================================
Tool Calls: search
Args:
   query: Texas Longhorns football schedule
================================= Tool Message =================================
Name: search
{...Results: "NCAA Division I Football, Georgia, Date..."}
================================== Ai Message ==================================
The Texas Longhorns played the Georgia Bulldogs last week.
Tool Calls: places
Args:
   query: Georgia Bulldogs stadium
================================= Tool Message =================================
Name: places

{...Sanford Stadium Address: 100 Sanford...}
================================== Ai Message ==================================
The address of the Georgia Bulldogs stadium is 100 Sanford Dr, Athens, GA
30602, USA.
```

قطعه کد ۹. خروجی برنامه ما از قطعه کد ۸

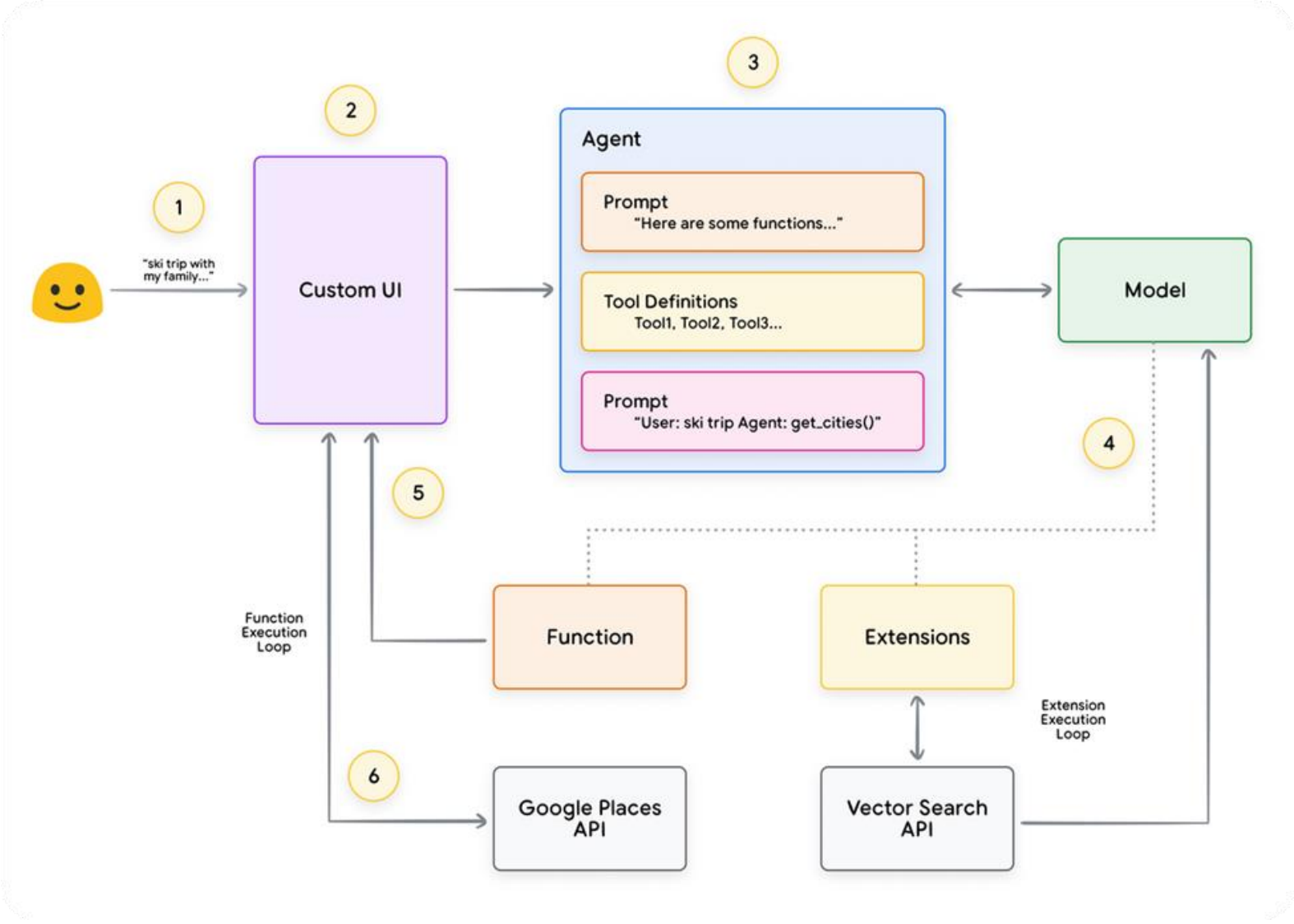
در حالی که این نمونه عامل ساده است، اما اجزای بنیادی مدل، ارکستراسیون و ابزارها را به خوبی نشان می‌دهد که همه با هم برای رسیدن به یک هدف خاص کار می‌کنند. در بخش پایانی، ما به بررسی نحوه ترکیب این اجزا در محصولات مدیریت شده در مقیاس گوگل مانند عامل‌های Vertex AI وPlaybook های تولیدی خواهیم پرداخت.

کاربردهای تولیدی با عامل‌های Vertex AI

در حالی که این مقاله به بررسی اجزای اصلی عامل‌ها پرداخت، ساخت برنامه‌های در حد درجه-تولید نیازمند ادغام آنها با ابزارهای اضافی مانند رابطهای کاربری، چارچوب‌های ارزیابی و مکانیزم‌های بهبود مستمر دارد.

پلتفرم Vertex AI گوگل این فرآیند را ساده می‌کند و یک محیط مدیریت شده کامل ارائه می‌دهد که تمام عناصر اساسی ذکر شده قبلی را پوشش می‌دهد. با استفاده از یک رابط زبان طبیعی، توسعه‌دهندگان می‌توانند به سرعت عناصر حیاتی عامل‌های خود را تعریف کنند - اهداف، دستورالعمل‌های وظایف، ابزارها، زیرعامل‌ها برای واگذاری وظایف و نمونه‌ها - تا به راحتی رفتار سیستم مورد نظر را بسازند.
علاوه بر این، این پلتفرم مجهز به مجموعه‌ای از ابزارهای توسعه است که امکان تست، ارزیابی، اندازه‌گیری عملکرد عامل، اشکال‌زدایی و بهبود کیفیت کلی عامل‌های توسعه‌یافته را فراهم می‌کند. این اجازه می‌دهد توسعه‌دهندگان بر روی ساخت و بهینه‌سازی عامل‌های خود تمرکز کنند در حالی که پیچیدگی‌های زیرساخت، استقرار و نگهداری توسط خود پلتفرم مدیریت می‌شوند.

در شکل ۱۵، ما یک نمونه معماری عامل را ارائه داده‌ایم که بر روی پلتفرم Vertex AI با استفاده از ویژگی‌های مختلفی مانندVertex Agent Builder ، Vertex Extensions، Vertex Function Callingو Vertex Example Store ساخته شده است. این معماری شامل بسیاری از اجزای مختلف لازم برای یک برنامه آماده تولید است.



شکل ۱۵. نمونه معماری عامل از ابتدا تا انتها که بر روی پلتفرم Vertex AI ساخته شده است

شما می‌توانید نمونه‌ای از این معماری عامل پیش‌ساخته را از آرشیو مستندات رسمی ما امتحان کنید.

خلاصه

در این مقاله، ما به بررسی بلوک‌های ساختمانی اساسی عامل‌های هوش مصنوعی تولیدی، ترکیب آنها و روش‌های مؤثر برای پیاده‌سازی آنها در قالب معماری‌های شناختی پرداختیم. برخی از نکات کلیدی این مقاله شامل موارد زیر است:

۱. عامل‌ها قابلیت‌های مدل‌های زبانی را با استفاده از ابزارها برای دسترسی به اطلاعات بلادرنگ، پیشنهاد اقدامات دنیای واقعی و برنامه‌ریزی و اجرای وظایف پیچیده به صورت خودگردان گسترش می‌دهند. عامل‌ها می‌توانند از یک یا چند مدل زبانی برای تصمیم‌گیری در مورد زمان و نحوه انتقال بین وضعیت‌ها و استفاده از ابزارهای خارجی برای تکمیل هر تعداد وظایف پیچیده که مدل به تنهایی قادر به انجام آن نیست، استفاده کنند.

۲. در مرکز عملکرد یک عامل، لایه ارکستراسیون قرار دارد که یک معماری شناختی است که استدلال، برنامه‌ریزی، تصمیم‌گیری و اقدامات را ساختاردهی می‌کند. تکنیک‌های استدلال مختلفی مانند ReAct ، Chain-of-Thought و Tree-of-Thoughts ، چارچوبی را برای لایه ارکستراسیون فراهم می‌کنند تا اطلاعات را دریافت کند، استدلال داخلی انجام دهد و تصمیمات یا پاسخ‌های آگاهانه تولید کند.

۳. ابزارها، مانند افزونه‌ها، توابع و ذخیره‌سازهای داده، به عنوان کلیدهای دسترسی عامل‌ها به دنیای خارج عمل می‌کنند و به آنها اجازه می‌دهند با سیستم‌های خارجی تعامل کنند و به دانشی دسترسی پیدا کنند که فراتر از داده‌های آموزشی آنهاست. افزونه‌ها پلی بین عامل‌ها وAPI های خارجی فراهم می‌کنند و اجرای تماس‌های API و بازیابی اطلاعات بلادرنگ را ممکن می‌سازند. توابع کنترل دقیق‌تری را از طریق تقسیم کار برای توسعه‌دهنده فراهم می‌کنند و به عامل‌ها اجازه می‌دهند پارامترهای تابعی را تولید کنند که می‌توانند در سمت کلاینت اجرا شوند. ذخیره‌سازهای داده به عامل‌ها دسترسی به داده‌های ساخت‌یافته یا بدون ساختار می‌دهند و امکان ایجاد برنامه‌های مبتنی بر داده را فراهم می‌کنند.

آینده عامل‌ها پیشرفت‌های هیجان‌انگیزی را در بر دارد و ما تنها به سطحی از آنچه ممکن است رسیده‌ایم. با پیشرفته‌تر شدن ابزارها و تقویت قابلیت‌های استدلال، عامل‌ها قادر خواهند بود مشکلات پیچیده‌تری را حل کنند.

علاوه بر این، رویکرد استراتژیک "زنجیره‌سازی عامل‌ها" (Agent chaining) به طور مداوم به محبوبیت خود ادامه خواهد داد. با ترکیب عامل‌های تخصصی - که هر کدام در یک حوزه یا وظیفه خاص موفق هستند - می‌توانیم رویکردی از "ترکیب عامل‌های متخصص" ایجاد کنیم که قادر به ارائه نتایج فوق‌العاده در صنایع مختلف و حوزه‌های مشکل باشد.

به یاد داشته باشید که ساخت معماری‌های پیچیده عامل‌ها نیازمند رویکردی تکراری است. آزمایش و بهینه‌سازی کلید حل مشکلات خاص کسب‌وکار و نیازهای سازمانی هستند. هیچ دو عاملی به دلیل ماهیت تولیدی مدل‌های بنیادی که زیرساخت آنها را تشکیل می‌دهند، به یک شکل ساخته نمی‌شوند. اما با بهره‌گیری از نقاط قوت هر یک از این اجزای بنیادی، می‌توانیم برنامه‌هایی مؤثر ایجاد کنیم که قابلیت‌های مدل‌های زبانی را گسترش دهند و ارزش واقعی در دنیای واقعی ایجاد کنند.

مراجع

1. Shafran, I., Cao, Y. et al., 2022, 'ReAct: Synergizing Reasoning and Acting in Language Models'. Available at: https://arxiv.org/abs/2210.03629

2. Wei, J., Wang, X. et al., 2023, 'Chain-of-Thought Prompting Elicits Reasoning in Large Language Models'. Available at: https://arxiv.org/pdf/2201.11903.pdf.

3. Wang, X. et al., 2022, 'Self-Consistency Improves Chain of Thought Reasoning in Language Models'. Available at: https://arxiv.org/abs/2203.11171.

4. Diao, S. et al., 2023, 'Active Prompting with Chain-of-Thought for Large Language Models'. Available at: https://arxiv.org/pdf/2302.12246.pdf.

5. Zhang, H. et al., 2023, 'Multimodal Chain-of-Thought Reasoning in Language Models'. Available at: https://arxiv.org/abs/2302.00923.

6. Yao, S. et al., 2023, 'Tree of Thoughts: Deliberate Problem Solving with Large Language Models'. Available at: https://arxiv.org/abs/2305.10601.

7. Long, X., 2023, 'Large Language Model Guided Tree-of-Thought'. Available at: https://arxiv.org/abs/2305.08291.

8. Google. 'Google Gemini Application'. Available at: h ttp://gemini.google.com.

9. Swagger. 'OpenAPI Specification'. Available at: https://swagger.io/specification/.

10. Xie, M., 2022, 'How does in-context learning work? A framework for understanding the differences from traditional supervised learning'. Available at: https://ai.stanford.edu/blog/understanding-incontext/.

11. Google Research. 'ScaNN (Scalable Nearest Neighbors)'. Available at: https://github.com/google-research/google-research/tree/master/scann.

12. LangChain. 'LangChain'. Available at: https://python.langchain.com/v0.2/docs/introduction/.

برای طراحی نیازهای هوش مصنوعی خود با ما تماس بگیرید

- ماموریت تیم دیجی هوش هما، آشناسازی اعضای محترم هیات مدیره، مدیران عامل، مدیران ارشد، و متخصصین سازمانها برای چگونگی پیاده‌سازی و استفاده از فناوریهای هوشمندسازی، و به ویژه هوش‌مصنوعی به عنوان فناوری شالوده شکن عصر حاضر، در سازمان یا دپارتمان مدنظر آنها می‌باشد.
- ما باور داریم که امروزه تحول دیجیتال و پیاده‌سازی هوش‌مصنوعی در سازمان‌ها نه یک انتخاب، بلکه یک ضرورت است، چرا که هزینه عدم استفاده از آن برای شرکت زیان بار خواهد بود. ما در مجموعه دیجی هوش هما (صبا) با افتخار آماده‌ایم تا شما را در این مسیر همراهی کنیم.
- مجموعه دیجی هوش هما (صبا) در نقش مشاوره، آموزش، و ارائه راه‌حلهای هوش‌مصنوعی به شما و سازمان شما متعهد بوده و پیاده‌سازی بهینه راه‌حلهای هوش‌مصنوعی را با کمک برترین متخصصین و شرکتهای ارائه دهنده خدمات هوش‌مصنوعی برای شما رقم خواهد زد.
- تکنولوژی‌های هوشمندسازی و هوش‌مصنوعی می‌توانند تغییرات اساسی در نحوه کارکرد و بهره‌وری سازمان‌ها به وجود آورند. با استفاده از راه‌حل‌های ما، می‌توانید فرایندهای پیچیده را ساده‌سازی کنید، تصمیم‌گیری‌های بهتری انجام دهید و بهره‌وری سازمان خود را به شکل چشمگیری افزایش دهید.
- چشم‌انداز ما در دیجی هوش هما (در نقش اپراتور هوش‌مصنوعی **AI Integrator**)، ایجاد تحولی واقعی و ملموس در سازمان‌های متقاضی است. ما با تیمی‌مجرب و متخصص، آماده‌ایم تا بهترین مشاوره‌ها و راه‌حل‌های هوشمند را به شما ارائه دهیم. از مشاوره، آموزش‌های درون‌سازمانی تا پیاده‌سازی کامل راه‌حل‌های هوش‌مصنوعی و اینترنت اشیا، ما در تمامی‌مراحل همراه شما هستیم.
- منتظر تماس شما متقاضی گرامی هستیم تا با همدیگر آینده‌ای هوشمندتر و پر رونق‌تر برای مملکت خود بسازیم. برای دریافت اطلاعات بیشتر و مشاوره تخصصی، با ما تماس بگیرید.

شرکت صبا (دیجی هوش هما)

راه های تماس با ما

ایمیل cmo@sabaind.com

تلگرام @DigiHooshHoma

لینکداین https://www.linkedin.com/company/maaia/

وب سایت www.sabaind.com

موبایل +98-0905 611 0895

تلفن +98-021-22146343

آدرس تهران – سعادت آباد – میدان بهرود – خیابان عابدی – ساختمان صبا

خلاصه مقاله

این فایل یک سند تخصصی درباره عامل‌های هوش مصنوعی (AI Agents) است که توسط گوگل منتشر شده است. در ادامه خلاصه‌ای جامع و مبسوط از آن به زبان فارسی ارائه می‌شود:

مقدمه

- **هدف اصلی** :عامل‌های هوش مصنوعی، برنامه‌هایی هستند که قابلیت‌های مدل‌های زبانی را گسترش داده و قادر به استفاده از ابزارها برای دسترسی به اطلاعات بلادرنگ، پیشنهاد اقدامات دنیای واقعی و اجرای وظایف پیچیده به صورت خودگردان هستند.
- **شباهت به انسان** :انسان‌ها اغلب برای تصمیم‌گیری از ابزارها (مانند کتاب، موتور جستجوی گوگل یا ماشین‌حساب) استفاده می‌کنند. به طور مشابه، مدل‌های زبانی نیز می‌توانند با استفاده از ابزارها، دانش خود را تقویت کنند.

عامل چیست؟

- **تعریف** :یک عامل هوش مصنوعی، برنامه‌ای است که با استفاده از ابزارهای موجود، به دنبال رسیدن به یک هدف است. این عامل‌ها می‌توانند بدون دخالت مستقیم انسان عمل کنند.
- **ویژگی‌ها**:
  - **مستقل** :قادر به تصمیم‌گیری و انجام اقدامات بدون نیاز به دستور مستقیم انسان.
  - **پیش‌بینی‌کننده** :حتی در صورت عدم وجود دستورالعمل‌های صریح، می‌توانند تصمیم بگیرند که چه کاری بعدی انجام دهند.

ساختمان پایه عامل‌ها

عامل‌ها از سه جزء اصلی تشکیل شده‌اند:

۱. **مدل**:
   - مدل زبانی (LM) که به عنوان مرکز تصمیم‌گیری عمل می‌کند.
   - می‌تواند از یک یا چند مدل زبانی استفاده کند.
۲. **ابزارها**:
   - ابزارها پلی بین مدل‌های زبانی و دنیای خارج هستند.
   - شامل API ها، توابع و ذخیره‌سازهای داده می‌شوند.
۳. **لایه ارکستراسیون**:
   - یک ساختار شناختی که استدلال، برنامه‌ریزی، تصمیم‌گیری و اقدامات را مدیریت می‌کند.
   - می‌تواند از تکنیک‌های مختلفی مانندReAct ، Chain-of-Thoughtو Tree-of-Thoughts استفاده کند.

تفاوت عامل‌ها و مدل‌ها

- **مدل‌ها**:
  - دانش آنها محدود به داده‌های آموزشی است.
  - فقط بر اساس پرس‌وجوی کاربر پاسخ می‌دهند.
  - فاقد ابزارهای داخلی برای تعامل با دنیای خارج.
- **عامل‌ها**:
  - دانش آنها از طریق ابزارها و دسترسی به داده‌های خارجی گسترش می‌یابد.
  - قادر به مدیریت تاریخچه گفتگوها و انجام وظایف چندمرحله‌ای هستند.
  - دارای ساختار شناختی برای استدلال و تصمیم‌گیری.

ابزارها: کلیدهای دسترسی به دنیای خارج

ابزارها به عامل‌ها اجازه می‌دهند با سیستم‌های خارجی تعامل کنند و دانشی فراتر از داده‌های آموزشی خود کسب کنند. این ابزارها شامل موارد زیر هستند:

۱. **افزونه‌ها**:
   - پلی بین عامل‌ها وAPI های خارجی.
   - به عامل‌ها کمک می‌کنندAPI ها را به صورت استاندارد فراخوانی کنند.
۲. **توابع**:
   - این ابزارها به توسعه‌دهندگان کنترل دقیق‌تری می‌دهند.
   - اجرا در سمت کلاینت انجام می‌شود.
۳. **ذخیره‌سازهای داده**:
   - دسترسی به داده‌های ساخت‌یافته و بدون ساختار.
   - امکان ایجاد برنامه‌های مبتنی بر داده.(RAG)

بهبود عملکرد مدل با یادگیری هدفمند

- **رویکردهای یادگیری**:

۱. **یادگیری درون متنی**:
   - مدل با استفاده از نمونه‌های کوتاه و پرامپت‌ها، در زمان استنتاج یاد می‌گیرد.
۲. **یادگیری مبتنی بر بازیابی**:
   - پرامپت مدل به طور پویا با اطلاعات مرتبط پر می‌شود.
۳. **یادگیری مبتنی بر تنظیم دقیق**:
   - مدل قبل از استنتاج با مجموعه داده بزرگتری آموزش می‌بیند.

نمونه‌های عملی

- **مثال با LangChain:**
  - استفاده از ابزارهایی مانند SerpAPI (برای جستجوی گوگل) و Google Places API.
  - عامل با استفاده از این ابزارها به پرس‌وجوی کاربر پاسخ می‌دهد.
- **مثال با Vertex AI:**
  - ارائه یک معماری عامل کامل با استفاده از افزونه‌ها، توابع و ذخیره‌سازهای داده.

جمع‌بندی

- **نقاط کلیدی:**

۱. عامل‌ها قابلیت‌های مدل‌های زبانی را با استفاده از ابزارها گسترش می‌دهند.

۲. لایه ارکستراسیون، قلب عملکرد عامل است که استدلال، برنامه‌ریزی و تصمیم‌گیری را مدیریت می‌کند.

۳. ابزارها (افزونه‌ها، توابع و ذخیره‌سازهای داده) به عامل‌ها اجازه می‌دهند با دنیای خارج تعامل کنند.

- **آینده عامل‌ها:**
  - با پیشرفت ابزارها و تقویت قابلیت‌های استدلال، عامل‌ها قادر خواهند بود مشکلات پیچیده‌تری را حل کنند.
  - رویکرد "زنجیره‌سازی عامل‌ها" به محبوبیت خود ادامه خواهد داد.

این خلاصه، به طور کلی تمام جنبه‌های مهم فایل را پوشش می‌دهد و به درک بهتر مفهوم عامل‌های هوش مصنوعی کمک می‌کند.

شرکت صبا (دیجی هوش هما)