توسعه هوش مصنوعی مبتنی بر مشخصات

از قصد تا کد قابل اعتماد

راهنمایی برای انتقال قصد به عامل های برنامه نویسی هوش مصنوعی، ثبت آن در قالب مشخصات ماندگار، طراحی آزمون بر پایه خطرهای واقعی و ارزیابی مطمئن کدی که عامل ها تولید می کنند

گزیده ای از آثار هری کریشنان، لی گریفین، توماش لِلِک و آرتور اسکوورونسکی

فصل نمونه رایگان

مشخصات نشر

عنوان اصلی: Spec-Driven AI Development: From Intent to Trusted Code عنوان فارسی: توسعه هوش مصنوعی مبتنی بر مشخصات: از قصد تا کد قابل اعتماد ناشر: Manning Publications Co. حق نشر: © ۲۰۲۷ انتشارات منینگ. تمامی حقوق محفوظ است.

فهرست مطالب

- مقدمه
- فصل اول: مقدمه ای بر توسعه مبتنی بر قصد
- فصل دوم: نوشتن نخستین مشخصات
- فصل پنجم: آزمون به مثابه پیشگیری از اتلاف
- فصل هشتم: ارزیابی حرف اول را می زند؛ چگونه به کدی اعتماد کنیم که خودمان ننوشته ایم

مقدمه

عامل های برنامه نویسی هوش مصنوعی می توانند با سرعتی چشمگیر کد قابل اجرا تولید کنند. اما سریع بودن با درست بودن یکی نیست و پرداخت شده به نظر رسیدن نیز به معنای قابل اعتماد بودن نیست. چالش اصلی این است که مطمئن شویم عامل، قصد ما را درک می کند، تصمیم های مهم را حفظ می کند و کاری تحویل می دهد که بتوانیم با اطمینان آن را ارزیابی کنیم. چهار فصل این کتاب الکترونیکی بنیادی، شما را در مسیری از قصد و مشخصات تا آزمون و ارزیابی راهنمایی می کنند.

در فصل «مقدمه ای بر توسعه مبتنی بر قصد»، فصل اول کتاب «توسعه مبتنی بر مشخصات» نوشته هری کریشنان، خواهید دید که گفتن اینکه عامل چه چیزی بسازد، به ندرت کافی است. ممکن است یک درخواست از نظر فنی روشن باشد، اما هدف آن روشن نباشد. برای مثال، درخواست «قابلیت بازیابی گذرواژه را اضافه کن» چیزی درباره باطل کردن نشست های سرقت شده یا اطمینان دادن به کاربران درباره امن بودن حسابشان نمی گوید. بیان هدف، محدودیت ها و نتایج مطلوب، زمینه لازم را در اختیار عامل ها قرار می دهد تا تصمیم های بهتری بگیرند، سوءبرداشت ها را کاهش دهند و راه حل هایی ارائه کنند که به مسئله واقعی می پردازند.

قصد روشن باید فراتر از یک گفت وگوی واحد نیز دوام بیاورد. فصل «نوشتن نخستین مشخصات»، فصل دوم کتاب «توسعه مبتنی بر مشخصات»، نشان می دهد چگونه قصد را به منبعی ماندگار و مرجع حقیقت تبدیل کنیم. این فصل با استفاده از یک کوتاه کننده نشانی اینترنتی، از مرحله کشف نیاز تا طراحی و برنامه ریزی پیاده سازی پیش می رود و نشان می دهد مشخصات چگونه مانع ناپدید شدن تصمیم های مهم در تاریخچه گفت وگو می شوند. در این فصل می آموزید چگونه زمینه کسب وکار و معیارهای پذیرش، تصمیم های فنی و محدودیت های معماری و نیز برنامه اجرا را ثبت کنید تا هم افراد و هم عامل ها مسیر روشن تری داشته باشند و از حذف ناخواسته جزئیات و دوباره کاری جلوگیری شود.

وقتی رفتار مورد انتظار روشن شد، پرسش بعدی این است که چگونه از آن محافظت کنیم. در فصل «آزمون به مثابه پیشگیری از اتلاف»، فصل پنجم کتاب «ساخت سامانه های نرم افزاری با ارزش بالا» نوشته لی گریفین، آزمون ها نه بر اساس تعدادشان، بلکه بر پایه خطرهایی که پوشش می دهند و دوباره کاری ای که مانع آن می شوند ارزیابی می شوند. با شیوه انتخاب میان آزمون های واحد، یکپارچه سازی و سرتاسری آشنا می شوید، می بینید چرا آزمون زودهنگام هزینه رفع نقص را کاهش می دهد و چگونه آزمون های شکننده یا ناپایدار خود به

منبع اتلاف تبدیل می شوند. این دیدگاه به ویژه زمانی سودمند است که عامل ها بتوانند در چند دقیقه مجموعه های بزرگی از آزمون ها تولید کنند. خواهید آموخت چگونه محافظت واقعی را از اعداد ظاهرا چشمگیر تشخیص دهید و آزمون هایی کم حجم تر و هدفمندتر بسازید که تغییر ایمن را پشتیبانی کنند.

با این حال، صرف قبول شدن آزمون ها لزوما دلیل کافی برای اعتماد به کد تولیدشده نیست. ممکن است عامل پیاده سازی ای تمیز و قانع کننده تولید کند که همچنان یک حالت مرزی را نادرست مدیریت کند یا یکی از ناورداهای مهم را نقض کند. فصل «ارزیابی حرف اول را می زند: چگونه به کدی اعتماد کنیم که خودمان ننوشته ایم»، فصل هشتم کتاب «مهندسی وایب» نوشته توماش للک و آرتور اسکوورونسکی، رویکردی دقیق تر ارائه می کند. با کمک قراردادها، آزمون مبتنی بر ویژگی، آزمون جهش و فنون دیگر، یاد می گیرید مشخص کنید چه چیزی راستی آزمایی شده، چه چیزی هنوز نامطمئن است و آیا شواهد موجود برای کاربرد مورد نظر کد به اندازه کافی قوی هستند یا نه.

این فصل ها در کنار یکدیگر، روشی عملی برای کار با عامل های برنامه نویسی ارائه می کنند: هدف را روشن کنید، آن را در قالب مشخصات حفظ کنید، خطرهای مرتبط را بیازمایید و پیش از پذیرفتن نتیجه، شواهد کافی مطالبه کنید. اگر این ایده ها شما را به کاوش بیشتر ترغیب کردند، قویا پیشنهاد می کنیم کتاب های کاملی را که این فصل ها از آن ها برگزیده شده اند نیز مطالعه کنید. هر یک از آن ها راهنمایی عمیق تری برای تبدیل تولید سریع کد به نرم افزاری ارائه می دهند که بتوانید با اطمینان نگهداری و منتشر کنید.

با سپاس، تیم منینگ

فصل اول: مقدمه ای بر توسعه مبتنی بر قصد

این فصل از کتاب «توسعه مبتنی بر مشخصات» نوشته هری کریشنان برگرفته شده است.

عامل های برنامه نویسی با سرعت زیادی کار می کنند؛ اما فقط در مسیری که شما برایشان تعیین کرده اید. این فصل نشان می دهد توضیح دادن اینکه چرا یک قابلیت اهمیت دارد، چگونه به عامل ها کمک می کند مسئله واقعی را حل کنند، نه اینکه صرفا از درخواست پیروی کنند.

در این فصل می خوانید

- قصد در برنامه نویسی با هوش مصنوعی چه معنایی دارد؛
- چرا بیان روشن قصد به گلوگاه تازه توسعه نرم افزار تبدیل شده است؛
- توسعه مبتنی بر قصد چیست و چگونه کار می کند؛
- آشنایی با توسعه مبتنی بر مشخصات و حلقه ها.

افزایش بهره وری حاصل از عامل های برنامه نویسی امروزی واقعی است، اما در عین حال مسئله ای عمیق تر را آشکار کرده است. هرچه مدل ها توانمندتر می شوند و بخش های طولانی تری از کار را به صورت خودمختار انجام می دهند، گردش کار انسان محور ما که بر اساس سرعت ساخت نرم افزار شکل گرفته است، به مرزهای توان خود نزدیک می شود. گلوگاه در حال جابه جایی است: دیگر نوشتن کد مسئله اصلی نیست، بلکه بیان قصدی است که مسیر تولید کد را هدایت می کند.

ساخت سریع یک قابلیت با کمک عامل برنامه نویسی یک موضوع است؛ رفع نقص های همان قابلیت، افزودن توانایی های تازه به آن و حفظ مزیت سرعت هوش مصنوعی در هفته ها و ماه های توسعه مداوم، موضوعی کاملا متفاوت است. برای این کار به سامانه هایی نیاز داریم که اطلاعات هدایت کننده عامل های برنامه نویسی را ساختارمند کنند تا خروجی هایی بسیار همسو با خواسته ما به دست آید. روش های ابتدایی مانند کدنویسی وایبی و پرامپت نویسی موردی، زمینه لازم را برای آن دسته از عامل های روزبه روز توانمندتر که باید وظایف پیچیده را مستقلا اجرا کنند، فراهم نمی کنند.

۱.۱. درک قصد

توسعه مبتنی بر قصد (Intent-Driven Development یا IDD) رویکردی مهندسی و بومی هوش مصنوعی برای توسعه نرم افزار است که بر تبدیل قصد ضمنی انسان به زبانی کاربردی و سازگار و نیز سامانه هایی برای عامل ها تمرکز دارد؛ به گونه ای که عامل ها بتوانند مستقلا پیاده سازی هایی تولید کنند که با آن قصد همسویی زیادی داشته باشند.

توسعه مبتنی بر قصد با قرار دادن قصد در مرکز شکل گیری زمینه موثر، به مشکلات ذاتی کدنویسی وایبی می پردازد. این رویکرد از آن جهت ضروری است که هرچه بیشتر می پذیریم عامل های برنامه نویسی تقریبا تمام کد ما را خواهند نوشت و نقش انسان ها بیشتر به بازبینی و نظارت محدود خواهد شد.

هدف ما در اینجا کمک به شما برای به کارگیری توسعه مبتنی بر قصد در سطح فردی، تیمی و سازمانی است. برای آغاز این مسیر، ابتدا مفاهیم بنیادین آن را معرفی می کنیم.

یکی از جنبه های مهم توسعه مبتنی بر قصد این است که درک مشترکی که از همکاری انسان و هوش مصنوعی حاصل می شود، در مصنوعات ماندگاری مانند مشخصات و آزمون ها ثبت شود. این مصنوعات، در کنار پیاده سازی ها و کدی که از آن ها تولید می شود، جایگاهی بنیادین دارند.

از یک منظر، مصنوعات قصد حتی می توانند از خود پیاده سازی مهم تر و ماندگارتر باشند. برای نمونه، حتی در پرامپت نویسی موردی نیز چنین وضعیتی را در نظر بگیرید: از عاملی می خواهیم اسکریپتی تولید کند که فایل های گزارش قدیمی تر از یک تاریخ مشخص را حذف کند، اما فقط در صورتی که بایگانی متناظر با آن فایل ها وجود داشته باشد.

در ابتدا ممکن است از عامل بخواهیم اسکریپتی ویژه ویندوز تولید کند. اگر بعدا به سامانه ای مبتنی بر لینوکس مهاجرت کنیم، استفاده از اسکریپت ویندوز به عنوان مبنای پیاده سازی جدید، مسیری غیرمستقیم ایجاد می کند. عامل باید ابتدا قصد زیربنایی را از روی کد ویندوز استنباط کند و سپس آن برداشت را به پیاده سازی لینوکس تبدیل کند؛ فرایندی که ممکن است به از دست رفتن بخشی از معنا بینجامد. راه مستقیم تر این است که پیاده سازی ویندوز را کنار بگذاریم و به پرامپت اصلی بازگردیم تا قصد تولید اسکریپتی متناسب با لینوکس حفظ شود. بنابراین، پرامپت از هر اسکریپت مشخصی که تولید می کند ماندگارتر شده است. توسعه مبتنی بر قصد این اصل را به شکلی نظام مندتر به کار می گیرد. مشخصات و آزمون ها قصد مشترک را حفظ می کنند و به منبع حقیقتی

تبدیل می شوند که عامل ها می توانند بر اساس آن، کل برنامه ها و محصولات را تولید یا دوباره تولید کنند. البته این کار مستلزم بیان دقیق و کامل قصد است.

قصد، رشته پیونددهنده واحدی است که یک ایده یا شهود مبهم را تا مرحله پیاده سازی ای همراهی می کند که بازتاب دهنده خواسته واقعی ماست. برای آغاز، ببینیم وقتی این رشته پیونددهنده وجود ندارد چه اتفاقی می افتد.

۱.۱.۱. بیان نیازمندی ها بدون روشن کردن قصد

بیشتر اوقات وقتی از یک عامل برنامه نویسی می خواهیم چیزی بسازد، نیازمندی ها یا شرحی از کارکرد مورد انتظار را بیان می کنیم. برای مثال:

> قابلیتی برای بازنشانی گذرواژه اضافه کن تا کاربران بتوانند از طریق ایمیل، پیوند بازنشانی درخواست کنند.

این پرامپت کاملا منطقی به نظر می رسد. مشخص و قابل اجراست و یک عامل برنامه نویسی توانمند احتمالا چیزی تولید می کند که با آن مطابقت داشته باشد.

با این حال، نتیجه همچنان می تواند به شکلی جدی شکست بخورد؛ نه به این دلیل که عامل درخواست را بد اجرا کرده، بلکه به دلیل چیزهایی که ناگفته باقی مانده اند. تصور کنید پیاده سازی با موفقیت گذرواژه را بازنشانی می کند، اما همه نشست های فعال کاربران را باز نگه می دارد. در این حالت، تمام نیازمندی های تصریح شده برآورده شده اند. با این همه، اگر هدف واقعی کمک به کاربر برای بازیابی حسابی بوده باشد که احتمال دارد به خطر افتاده باشد، همین یک فرض، هدف اصلی را بی اثر می کند.

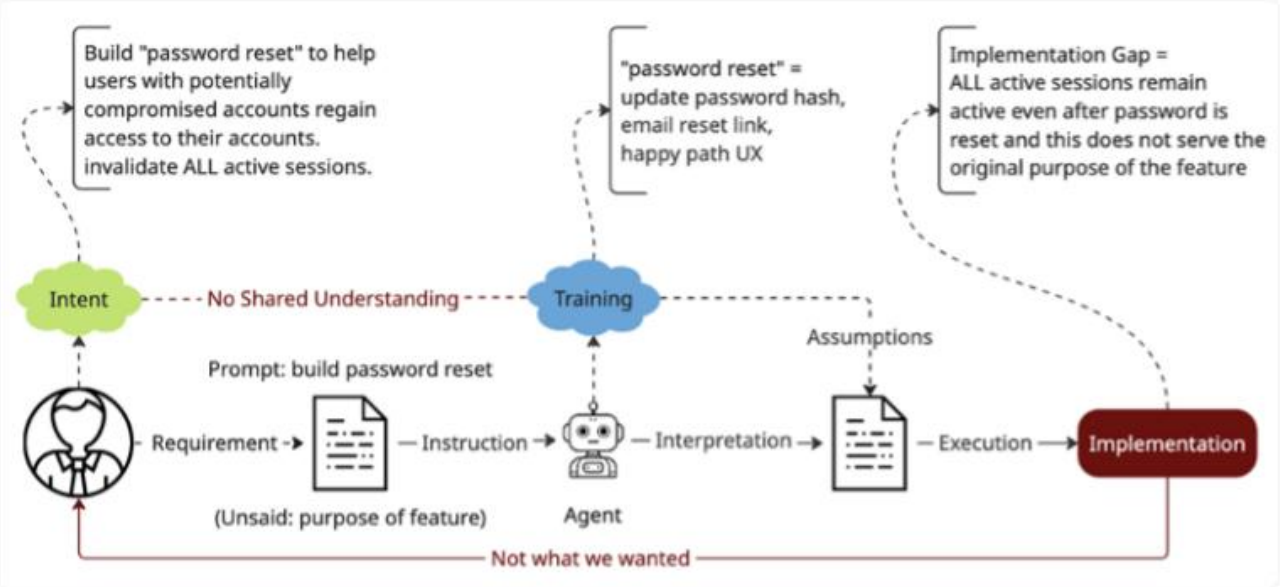
این شکاف تفسیری همان الگویی است که هر روز میان انسان ها در تیم های نرم افزاری رخ می دهد. مالک محصول درخواستی برای افزودن قابلیت بازنشانی گذرواژه ثبت می کند. او به واسطه رسیدگی به تیکت های پشتیبانی مشتریان و بررسی رخدادهای امنیتی می داند که نشست ها باید باطل شوند؛ اما این دانسته فقط در ذهن او باقی می ماند. توسعه دهنده ای که مسئول انجام تیکت است، دقیقا همان چیزی را می خواند که نوشته شده، آن را ماهرانه پیاده سازی می کند و به دلیل نبود هرگونه اشاره در تیکت، باطل کردن نشست ها را از قلم می اندازد. هیچ یک از دو طرف بی دقت نیستند. مالک محصول فرض می کند توسعه دهنده همان زمینه ای را می داند که باعث شده نیازمندی از نظر او کامل به نظر برسد. توسعه دهنده نیز فرض می کند تیکت همه اطلاعات مورد نیاز را در خود دارد.

این وضعیت در تعامل با عامل های برنامه نویسی نیز دقیقا به همین شکل رخ می دهد، با یک تفاوت مهم: هنگام نوشتن پرامپت برای عامل، جایگاه دارنده دانش ضمنی تغییر می کند. اکنون ما مهندسان در موقعیت مالک محصول قرار داریم. می دانیم هنگام بازنشانی گذرواژه باید نشست ها باطل شوند، زیرا پیش تر سامانه های احراز هویت ساخته ایم و درباره رخدادهای تصاحب حساب مطالعه کرده ایم. اما این نکته را بیان نمی کنیم؛ چون در سطحی از ذهن خود انتظار داریم عامل، آنچه را از نظر یک مهندس بدیهی است، خودش تکمیل کند. عامل نیز که فقط بر اساس متن کار می کند، دقیقا همان کاری را انجام می دهد که پرامپت از او خواسته است. سازوکار شکست در هر دو حالت یکسان است: دانشی که در ذهن یک طرف وجود دارد، به طرفی که کار را انجام می دهد منتقل نمی شود.

نکته: همه نیازمندی های ذکرشده در پرامپت بازنشانی گذرواژه برآورده شدند، اما قابلیت همچنان شکست خورد. مشکل در نحوه پیاده سازی نبود، بلکه در مواردی بود که بیان نشده بودند. افزودن مدلی پیشرفته تر یا سامانه اجرایی بهتر، این شکاف را برطرف نمی کند. تنها راه، آشکار کردن همان چیزهایی است که ناگفته مانده اند.

شکل ۱.۱ الگوی تعامل در پرامپت نویسی موردی را به صورت تصویری نشان می دهد؛ الگویی که در آن انسان فقط آنچه را باید ساخته شود به عامل می گوید و قصد زیربنایی قابلیت را بیان نمی کند. عامل نیز بر پایه آموزش های مدل خود، نیازمندی ناقص را چنین تفسیر می کند که باید فقط هش گذرواژه را به روزرسانی کند و نیازی به باطل کردن همه نشست های فعال نیست. در نتیجه، پیاده سازی با خواسته واقعی انسان همسو نخواهد بود.

یک واکنش به این وضعیت آن است که تقصیر را متوجه آموزش مدل بدانیم. شاید اگر مدل توانمندتر بود، می توانست بخش های ناگفته نیازمندی را استنباط کند. اتکا به توانایی مدل برای پر کردن این شکاف، مانند آن است که از همکاری بسیار باهوش انتظار داشته باشیم ذهن ما را بخواند. این راهبردی عملی نیست و نتیجه را به این وابسته می کند که آموزش مدل تا چه اندازه با منظور ما همخوانی داشته باشد.



شکل ۱.۱. نبود زمینه مشترک میان نیت ما و آموزش های مدل عامل می تواند به پیاده سازی هایی منجر شود که با خواسته واقعی ما همسو نیستند.

راه ساده تر این است که میزان تفسیر آزادانه ای را که به عامل واگذار می کنیم کاهش دهیم و قصد خود را روشن بیان کنیم. برای بهبود بیان قصد، ابتدا باید خود مفهوم قصد را بشناسیم.

۱.۱.۲. «قصد» چیست؟

پرامپت زیر را در نظر بگیرید که از یک عامل می خواهد گردش کار بازنشانی گذرواژه بسازد:

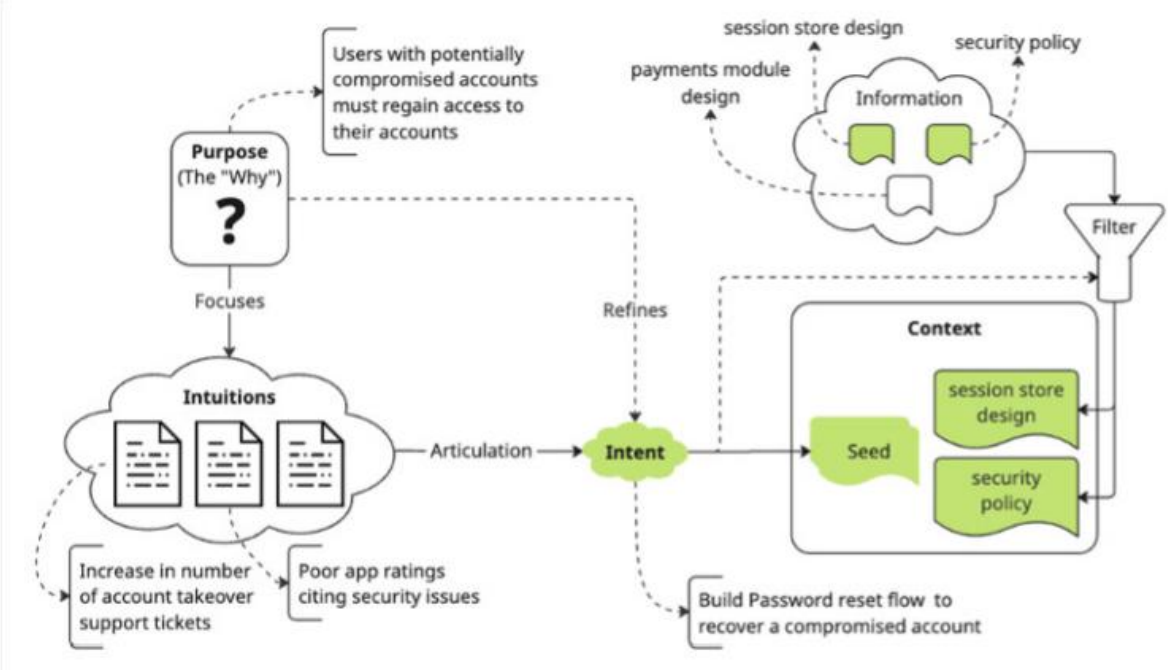
> گردش کاری برای بازنشانی گذرواژه بساز که هدف آن بازیابی حسابی باشد که احتمال دارد به خطر افتاده باشد، نه صرفا تعویض یک اعتبارنامه.

هدف گسترده تر، یا دلیل ساخت این قابلیت، کمک به کاربران برای بازیابی دسترسی به حساب هایی است که ممکن است در معرض نفوذ قرار گرفته باشند. این هدف را می توان به روش های گوناگونی دنبال کرد؛ از جمله اینکه از کاربران آسیب دیده بخواهیم برای دریافت راهنمایی با پشتیبانی مشتریان تماس بگیرند. با این حال، هدف به تنهایی عملیاتی نیست. باید آن را به رویکردی مشخص، مانند ساخت گردش کار بازنشانی گذرواژه، تبدیل کرد.

قصد، بیان عملیاتی و اقدام پذیر همین هدف است.

هنگام بیان نیازمندی ها، تمرکز معمولا بر رویکرد مشخص یا همان چیزی است که باید ساخته شود و چرایی زیربنایی آن ناگفته می ماند. بیان قصد، همان طور که در پرامپت نمونه دیدیم، این چرایی را آشکار می کند و آن

را به چیستی کار پیوند می دهد. این چرایی سپس برای تمام تصمیم های بعدی زمینه فراهم می کند؛ از صورت بندی محصول و انتخاب فناوری گرفته تا شیوه راستی آزمایی نتیجه.



شکل ۱.۲ نشان می دهد چگونه شهودهای سطح بالا، از طریق بیان روشن، به قصد تبدیل می شوند. هدف، همان «چرا»یی است که بیان قصد را هدایت می کند. سپس قصد به هسته اولیه زمینه تبدیل می شود و اطلاعات دیگری را که باید در این زمینه قرار بگیرند، پالایش و انتخاب می کند.

در مثال بازنشانی گذرواژه، وقتی هدف را بازیابی حسابی معرفی می کنیم که احتمال دارد به خطر افتاده باشد، فرد یا عاملی که درخواست را تفسیر می کند می تواند این هدف را در تمام تصمیم های بعدی لحاظ کند. طراحی فنی می تواند سازوکاری برای باطل کردن همه نشست های فعال داشته باشد، نه اینکه فقط هش گذرواژه را تغییر دهد. معیارهای راستی آزمایی می توانند شامل آزمون هایی باشند که تایید می کنند پس از بازنشانی، همه نشست های موجود خاتمه یافته اند؛ نه اینکه فقط کارکرد گذرواژه جدید را بررسی کنند. حتی تجربه کاربری نیز می تواند فوریت موضوع را منتقل کند و به کاربر اطمینان دهد که حسابش ایمن شده است، نه اینکه صرفا تغییر گذرواژه را تایید کند. اگر قصد در هر یک از این سطوح آشکار نشود، عامل ممکن است در هر مرحله فرض های خودش را وارد کند و نتیجه به تدریج از هدف اولیه فاصله بگیرد.

۱.۱.۳. «قصد» چه چیزی نیست؟

قصد را نباید با شهود، ایده یا هدف های کلی اشتباه گرفت. شهود، احساسی مبتنی بر تجربه است که به ما می گوید چیزی اهمیت دارد. ممکن است یکی از اعضای تیم که در گذشته با رخدادهای مشکوک تصاحب حساب روبه رو بوده، احساس مبهمی داشته باشد که امنیت حساب های برنامه باید بهتر شود. این احساس ممکن است کاملا درست باشد، اما برای اقدام بیش از اندازه مبهم است. چنین شهودی نمی تواند به عامل برنامه نویسی یا حتی هم تیمی ما بگوید دقیقا چه چیزی باید تغییر کند. ایده ها جهت های احتمالی را پیشنهاد می کنند؛ مثلا اینکه کاربران بتوانند گذرواژه خود را بازنشانی کنند. یک هدف کلی نیز ممکن است ارتقای امنیت حساب را تعیین کند. هیچ یک از این موارد به تنهایی آن قدر دقیق نیست که بتواند راهنمای یک تغییر مشخص باشد.

قصد زمانی شکل می گیرد که هدف به اندازه ای ملموس شود که دامنه اقدام را محدود و هدایت کند. عبارت «این قابلیت برای بازیابی حسابی طراحی شده است که احتمال دارد به خطر افتاده باشد» بیان قصد است. در این نقطه است که هدف می تواند بر آنچه باید ساخته شود، شیوه ساخت آن و معیار قضاوت درباره نتیجه اثر بگذارد. برای مثال، قصد، راستی آزمایی درستی قابلیت بازنشانی گذرواژه را به این پرسش پیوند می دهد که آیا قابلیت به هدف خود، یعنی کمک به کاربران برای بازیابی حساب های در معرض خطر، دست یافته است یا نه؛ به جای آنکه صرف انجام شدن بازنشانی گذرواژه را معیار موفقیت بدانیم. این توانایی در راستی آزمایی نتایج، برای اجرای مستقل عامل ها بدون دخالت انسان ضروری است.

باید توجه داشت که قصد فقط یکی از اجزای زمینه است. قصد خودِ زمینه نیست، اما کمک می کند تشخیص دهیم چه اطلاعات دیگری برای دستیابی به نتیجه مورد نظر اهمیت دارند.

۱.۱.۴. بیان قصد برای هوش مصنوعی

بیان قصد، فرایند آشکار کردن قصد به اندازه ای است که افراد و عامل ها بتوانند بر اساس آن عمل کنند. این فرایند، شهود خام، ایده های مبهم، هدف های کلی و فرض های ناگفته را به چیزی ملموس تبدیل می کند که بتواند مسیر اجرا را هدایت کند.

مشکل رویکردهایی مانند به اشتراک گذاشتن یک سند نیازمندی این است که تمام بار بیان قصد بر دوش نویسنده سند قرار می گیرد؛ در حالی که نویسنده، به نوبه خود، نمی داند طرف دریافت کننده از چه چیزهایی بی خبر است.

مسئله به ماهیت دستوری این الگوی تعامل برمی گردد؛ الگویی که گفت وگو میان نویسنده نیازمندی و دریافت کننده را برای ساخت درکی مشترک تشویق نمی کند. همین مسئله به تعامل انسان و عامل نیز منتقل شده است.

در تعامل انسان و عامل، بیان صرف آنچه می خواهیم تنها زمانی جواب می دهد که آموزش مدل با منظور ما همسو باشد. این همخوانی میان خواسته ما و آموزش مدل سازوکاری قابل اتکا نیست و همسویی نتیجه فقط احتمالی خواهد بود. این همان کدنویسی وایبی کلاسیک است: وقتی حال و هوای کار درست از آب دربیاید، خوش شانس بوده ایم؛ اما وقتی چنین نشود، هیچ دیدی نسبت به دلیل آن نداریم. از آنجا که عامل در ابتدا با ما تعامل نمی کند و صرفا دستورها را اجرا می کند، فرصتی هم برای آشکار کردن فرض هایی که در نظر گرفته است وجود ندارد.

حالت برنامه ریزی (Plan Mode)، که در آن عامل پیش از نوشتن هرگونه کد، رویکرد مورد نظر خود را ترسیم می کند، این وضعیت را بهتر می کند؛ زیرا تفسیر عامل را پیش از آغاز پیاده سازی آشکار می سازد. با بازبینی برنامه پیشنهادی عامل، وابستگی خود را به همان همخوانی احتمالی در کدنویسی وایبی کاهش می دهیم. تعامل میان ما و عامل حالتی گفت وگومحورتر پیدا می کند و لازم نیست تا پایان اجرا برای ارائه نظر بازبینی صبر کنیم. با وجود این، این گام رو به جلو نیز محدودیت دارد: کیفیت برنامه اولیه همچنان احتمالی است و به کیفیت پرامپت نخستین ما بستگی دارد. برای رفع این مشکل می توانیم از عامل بخواهیم پیش از برنامه ریزی، با ما مصاحبه کند. این کار گفت وگومحورتر به نظر می رسد و ممکن است ملاحظاتی را آشکار کند که خودمان به آن ها فکر نکرده بودیم. اما حتی برای پرسیدن پرسش های مرتبط نیز عامل به نوعی قصد اولیه نیاز دارد تا گفت وگو را بر اساس آن جهت دهی کند؛ همان قصدی که هنوز بیان نکرده ایم.

برای مثال، اگر بدون توضیح هدف، از عامل بخواهیم قابلیت بازنشانی گذرواژه بسازد، ممکن است درباره ارائه دهندگان ایمیل، قواعد گذرواژه و محدودیت تعداد درخواست ها از ما سوال کند. اینها پرسش هایی منطقی هستند، اما گفت وگو را به سمت جزئیات پیاده سازی قابلیت محدود می کنند و ممکن است نیاز به ایمن سازی حسابی را که در معرض نفوذ بوده است نادیده بگیرند. از سوی دیگر، اگر فقط هدف جلوگیری از سوءاستفاده بیشتر را بیان کنیم، عامل ممکن است گزینه هایی مانند احراز هویت دومرحله ای یا مسدودسازی حساب از طریق پشتیبانی را بررسی کند. این موضوع در مرحله کشف نیاز می تواند مفید باشد، اما دامنه مصاحبه را فراتر از تغییری می برد که قصد انجامش را داریم. اگر مصاحبه را با قصد اولیه ساخت گردش کار بازنشانی گذرواژه برای کمک به کاربران در بازیابی حساب های در معرض خطر آغاز کنیم، عامل می تواند پرسش های متمرکزی درباره پیامدهای مرتبط

مطرح کند؛ از جمله باطل کردن نشست های فعال، ثبت رویدادهای مشکوک و تایید ایمن شدن حساب. مصاحبه می تواند قصد را به تدریج دقیق تر کند، اما همچنان به یک نقطه آغاز نیاز دارد.

این تجربه بیان قصد با سازوکارهای موردی، در بهترین حالت برای ساخت نمونه های اولیه و برنامه های کوچک تر یا کم پیچیدگی قابل مدیریت است. اما تکرار آن طی ماه ها و سال ها برای برنامه های پیشرفته با معماری پیچیده، امکان پذیر نیست. برای یافتن رویکردی بهتر، می توانیم به شیوه هایی نگاه کنیم که پیش از ظهور عامل های برنامه نویسی، انسان ها با یکدیگر قصد خود را بیان می کردند.

۱.۲. بیان قصد پیش از ظهور هوش مصنوعی در برنامه نویسی

پیش از ورود هوش مصنوعی، بیان قصد میان انسان ها بر الگوهای همکاری متکی بود؛ الگوهایی که به گفت وگو ساختار می دادند. ما به گفت وگویی میان انسان و هوش مصنوعی نیاز داریم که به قالبی متکی باشد و امکان گفت وگوی معنادار و دستیابی به درک مشترک را فراهم کند. برخلاف دستورالعمل ها و مصاحبه ها که هر کدام به نفع یکی از طرفین تعامل سوگیری دارند، گفت وگویی که بر اساس چارچوبی مشترک و با مشارکت هر دو طرف، یعنی عامل برنامه نویسی و ما، ساختار یافته باشد و برای هر یک نقشی متناسب در نظر بگیرد، قصد را به موثرترین شکل آشکار می کند.

چنین گفت وگوی ساختارمندی در عمل چه شکلی دارد؟ این ایده تازه نیست. مدت ها پیش از ظهور عامل های برنامه نویسی هوش مصنوعی، مهندسی نرم افزار روش هایی را توسعه داده بود که دقیقا بر همین اصل بنا شده بودند: داستان های کاربر، توسعه مبتنی بر رفتار و توسعه مبتنی بر آزمون.

داستان های کاربر و معیارهای پذیرش

داستان های کاربر یکی از تلاش ها در همین راستا بودند. شکست تعامل انسانی پیشین را به یاد بیاورید؛ جایی که دانش مالک محصول درباره لزوم باطل کردن نشست ها هرگز به توسعه دهنده منتقل نشد. داستان های کاربر پاسخی ساختارمند به همین نوع شکست هستند. ساختار آن ها عمدا گفت وگومحور طراحی شده است. به جای بیان مستقیم یک نیازمندی، داستان کاربر درخواست را از دیدگاه فردی صورت بندی می کند که به آن قابلیت نیاز دارد و هدف را آشکار می سازد:

فهرست ۱.۱. داستان کاربر برای بازنشانی گذرواژه:

- به عنوان دارنده حساب،
- می خواهم گذرواژه خود را بازنشانی کنم،
- تا اگر به نفوذ به حسابم مشکوک شدم،
- بتوانم دوباره کنترل آن را در اختیار بگیرم.

در فهرست ۱.۱، عبارت «تا اینکه» مهم ترین نقش را ایفا می کند؛ زیرا چرایی کار را مشخص می کند. مهم تر از آن، اطلاعات به صورت تدریجی آشکار می شوند. ابتدا شخصیت یا گروه کاربری ای را که عمدتا از قابلیت استفاده خواهد کرد مشخص می کنیم، زمینه او را می شناسیم و می فهمیم این زمینه چگونه بر طراحی اثر می گذارد. سپس به نیاز او برای این قابلیت می رسیم؛ موضوعی که امکان بررسی اطلاعات و درونی کردن تدریجی جزئیات را فراهم می کند.

وقتی تعریف معیارهای پذیرش را آغاز می کنیم، جنبه های عینی و قابل راستی آزمایی قصد را بیش از پیش آشکار می سازیم. فهرست ۱.۲ معیارهای پذیرش داستان کاربری را نشان می دهد که نخستین بار در فهرست ۱.۱ معرفی شد.

فهرست ۱.۲. معیارهای پذیرش داستان کاربر برای بازنشانی گذرواژه:

سناریو: بازنشانی گذرواژه، دسترسی های قبلی را لغو می کند

- با فرض اینکه دارنده حساب هستم و یک یا چند نشست فعال دارم؛
- و درخواست بازنشانی گذرواژه داده ام؛
- وقتی با استفاده از پیوند بازنشانی، گذرواژه جدیدی را با موفقیت ثبت می کنم؛
- آنگاه همه نشست های موجود حساب من باطل می شوند؛
- و باید در هر دستگاه دوباره وارد حساب شوم.

سناریو: پیوند بازنشانی فقط برای مدت محدودی معتبر است

- با فرض اینکه درخواست بازنشانی گذرواژه داده ام؛
- وقتی پیوند بازنشانی را ظرف ۳۰ دقیقه باز می کنم؛
- آنگاه اجازه دارم گذرواژه جدیدی انتخاب کنم.

سناریو: از پیوند بازنشانی منقضی شده نمی توان استفاده کرد

- با فرض اینکه درخواست بازنشانی گذرواژه داده ام؛
- وقتی پیوند بازنشانی را پس از ۳۰ دقیقه باز می کنم؛
- آنگاه پیوند رد می شود؛
- و از من خواسته می شود پیوند تازه ای درخواست کنم.

توسعه مبتنی بر رفتار

این ساخت زمینه مشترک از طریق معیارهای پذیرش، یکی از پایه های توسعه مبتنی بر رفتار (Behavior-Driven Development یا BDD) بود. زبان متنی ساده گرکین (Gherkin) این نحو را رسمی کرد تا درکی مشترک میان بخش کسب وکار و بخش فناوری ایجاد شود. دلیل اهمیت آن، خود نحو نبود؛ بلکه گفت وگویی بود که این ساختار شکل می داد. سناریوهایی که از این فرایند بیرون می آمدند، مصنوعات مشترکی بودند که هر دو طرف در تولیدشان مشارکت داشتند، نه اسنادی که دست به دست می شدند.

اگر BDD را با ابزار توسعه دهنده اشتباه بگیریم و تصور کنیم تیم محصول باید اسناد نیازمندی را به توسعه دهندگان تحویل دهد تا آن ها بعدا آن اسناد را به نحو گرکین تبدیل کنند، دوباره به نقطه اول بازگشته ایم: هیچ درک مشترکی شکل نگرفته و زمان بیشتری نیز در فرایند ترجمه و تبدیل هدر رفته است.

توسعه مبتنی بر آزمون

توسعه مبتنی بر آزمون (Test-Driven Development یا TDD) گفت وگوی ساختارمند مشابهی را در سطحی دیگر ممکن می کند. نوشتن آزمون واحد پیش از نوشتن کد ما را وادار می کند پیش از پیاده سازی، قاعده مشخصی از سامانه را آشکار و صریح کنیم. در مثال بازنشانی گذرواژه، این قاعده می تواند چنین باشد:

فهرست ۱.۳. آزمون راستی آزمایی باطل شدن نشست ها پس از بازنشانی گذرواژه

کد

```
test "password reset invalidates all existing sessions for
the user"
    # آماده سازی
    user = create_user_with_password("old-password")
    session_1 = login(user, device="laptop")
```

```
    session_2 = login(user, device="phone")
    session_3 = login(user, device="tablet")
    reset_token = request_password_reset(user.email)

    # اجرا
    complete_password_reset(reset_token,
new_password="new-password")

    # بررسی نتیجه
    assert session_store.active_sessions_for(user.id) ==
[]
```

نوشتن آزمونی مانند نمونه فهرست ۱.۳، پیش از آنکه حتی یک خط از پیاده سازی وجود داشته باشد، قصد را در سطح قواعد سامانه آشکار می کند. هنگام نوشتن آزمون، صورت مسئله را به گونه ای تنظیم می کنیم که قصد قابل راستی آزمایی شود. سپس وقتی سراغ نوشتن کد می رویم، نقش خود را تغییر می دهیم و تا زمان قبول شدن آزمون، راه حل آن صورت بندی را پیاده می کنیم.

وقتی یک نفر به تنهایی TDD را به کار می گیرد، این روش به نوعی گفت وگو با خودمان است. پیوسته میان صورت بندی مسئله (چیستی) و حل آن (چگونگی) جابه جا می شویم و در تمام این مدت مطمئن می شویم نتیجه با دلیلی که کار را به خاطر آن آغاز کرده ایم همسو باشد. همانند داستان های کاربر، اطلاعات به صورت تدریجی آشکار می شوند.

نوشتن آزمون پیش از پیاده سازی برای بیان قصد قابل راستی آزمایی، همان چیزی است که TDD را هنگام استفاده از عامل های برنامه نویسی نیز موثر می کند. به جای آنکه کل پیاده سازی را یکباره تولید کنیم، TDD عامل را محدود می کند تا روی قصد موجود در فضایی کوچک تمرکز کند و پیش از رفتن به مسئله بعدی، همان بخش را کامل کند. این رویکرد اساسا با درخواست از عامل برای نوشتن آزمون پس از پایان پیاده سازی تفاوت دارد. در حالت دوم، با پیاده سازی ای روبه رو هستیم که بدون هدایت قصدی قابل راستی آزمایی ساخته شده و ممکن است از همان ابتدا از هدف اصلی ما فاصله گرفته باشد. آزمون هایی که برای یک پیاده سازی ناهمسو نوشته می شوند، فقط رفتار نادرست را ثبت می کنند و به این ترتیب هدف اصلی آزمون نویسی از بین می رود.

با گذشت زمان، مجموعه آزمون هایی که از طریق TDD ساخته شده اند به مستنداتی زنده برای کد تبدیل می شوند. عامل های برنامه نویسی به جای آنکه قصد ضمنی موجود در کد فعلی را استنباط کنند، می توانند برای

تغییرات بعدی مستقیما به قصد صریح و قابل راستی آزمایی ای دسترسی داشته باشند که در مجموعه آزمون ها ثبت شده است.

روش هایی مانند TDD ،BDD، داستان های کاربر و معیارهای پذیرش در یک ویژگی مشترک اند: همه آن ها شکل هایی از مهندسی زمینه هستند که مدت ها پیش از ورود هوش مصنوعی، برای همکاری انسان ها به کار می رفتند. این روش ها تصویر کامل را در دسترس قرار می دهند و اطلاعات را چنان سازمان می دهند که خواننده بتواند بخش مورد نیازش را پیدا کند.

۱.۳. توسعه مبتنی بر قصد چگونه کار می کند؟

توضیح دادیم که توسعه مبتنی بر قصد شامل ساخت درک مشترک از طریق همکاری انسان و هوش مصنوعی است؛ درکی که سپس در مصنوعات ماندگاری مانند مشخصات و آزمون ها ثبت می شود تا عامل بتواند مستقلا اجرا را بر عهده بگیرد. اکنون می توانیم معنای عملی هر یک از این اجزا را بررسی کنیم.

توانایی یک عامل برنامه نویسی برای اجرا، فقط به اندازه اطلاعاتی است که می توانیم در اختیارش بگذاریم. اگر کل پایگاه کد را همراه با سیاست های امنیتی در اختیار عامل بگذاریم و از او بخواهیم قابلیت بازنشانی گذرواژه را بسازد، ممکن است بخش قابل توجهی از پنجره زمینه عامل پیش از آنکه حتی یک پرسش روشن کننده از ما بپرسد یا نخستین خط کد را بنویسد، مصرف شود. اگر بدون توجه به این نشانه مصرف پنجره زمینه، گفت وگو را ادامه دهیم، ممکن است با فشرده سازی زمینه روبه رو شویم؛ وضعیتی که در آن عامل هنگام خلاصه کردن گفت وگو برای آغاز نشست تازه، تصمیم های مهم را فراموش می کند.

مهندسی زمینه، عمل فراهم کردن اطلاعات درست و سازمان دهی آن به گونه ای است که خواننده یا عامل بتواند برای اجرای پرامپت های ما به اطلاعات مورد نیازش دسترسی پیدا کند. توسعه مبتنی بر قصد، در درجه نخست با ساخت این زمینه موثر از طریق گفت وگوی انسان و عامل سروکار دارد تا عامل پس از آن بتواند پیاده سازی را مستقلا تکمیل کند.

پنجره زمینه و فشرده سازی

پنجره زمینه، مقدار محدودی از اطلاعات است که مدل می تواند در یک تعامل واحد در نظر بگیرد. هر پیامی که می فرستیم، هر فایلی که عامل می خواند و هر خروجی ابزاری که دریافت می کند، بخشی از فضای این پنجره را اشغال می کند. وقتی پنجره پر شود، ناگزیر باید بخشی از اطلاعات کنار گذاشته شود.

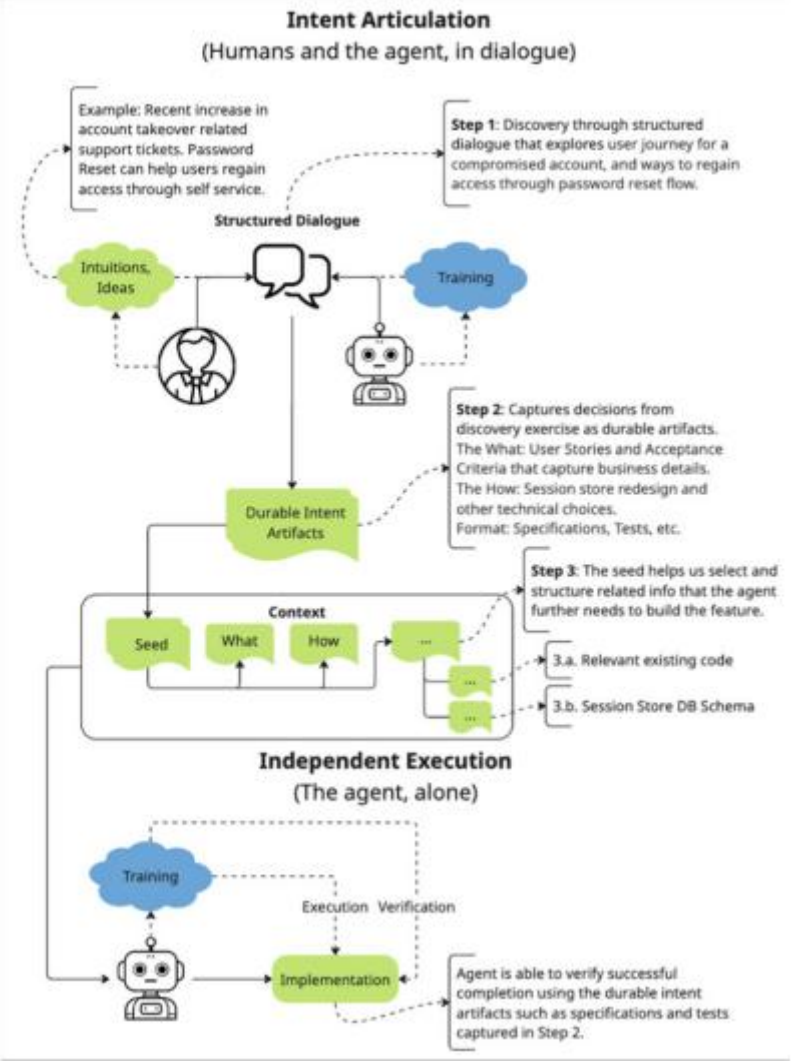
بیشتر عامل های برنامه نویسی این وضعیت را با فشرده سازی مدیریت می کنند. وقتی زمینه به حد نهایی خود نزدیک می شود، عامل بخش های پیشین گفت وگو را در قالب بازنمایی کوتاه تری خلاصه می کند و آن خلاصه را جایگزین متن اصلی می سازد. این مبادله آشناست: به بهای کاهش وفاداری، فضا آزاد می کنیم. جزئیاتی که عامل آن ها را بی اهمیت تشخیص داده است حذف می شوند و ظرافت هایی که بر تصمیم های قبلی اثر گذاشته اند ممکن است در خلاصه باقی نمانند.

دشواری کار این است که به ندرت از قبل می دانیم کدام جزئیات در آینده اهمیت پیدا خواهند کرد. ممکن است محدودیتی که در پیام سوم یک نشست به طور گذرا مطرح شده، برای نحوه ساخت قابلیت کاملا اساسی باشد؛ اما اگر آن محدودیت در فرایند فشرده سازی حفظ نشود، عامل طوری به کار ادامه می دهد که گویی هرگز چنین محدودیتی وجود نداشته است. نشانه های این مشکل معمولا بسیار دیرتر ظاهر می شوند؛ اغلب به شکل پیاده سازی ای که به طرز ظریفی نادرست است و ردیابی ارتباط آن با بخشی از زمینه که از دست رفته، دشوار خواهد بود.

بنابراین نباید برای حفظ گفت وگوی خود به نشست عامل وابسته باشیم. فشرده سازی آنچه را عامل مهم تشخیص می دهد نگه می دارد، اما روش قابل اتکایی برای تشخیص محدودیتی ندارد که شاید در گفت وگویی گذرا مطرح شده باشد و بعدا اهمیت پیدا کند. در عوض، باید مصنوعات ماندگاری را استخراج و بازبینی کنیم که تصمیم های گرفته شده، محدودیت های مورد توافق و موضوعات بررسی شده را ثبت می کنند. این مصنوعات، از جمله مشخصات و اسناد خلاصه، بیرون از پنجره زمینه نگهداری می شوند و می توان آن ها را در نشست های آینده بارگذاری کرد. توسعه مبتنی بر قصد رویکردی نظام مند برای ایجاد و سازمان دهی این مصنوعات فراهم می کند و همکاری انسان و عامل را در برابر محدودیت های پنجره زمینه مقاوم می سازد.

شکل ۱.۳ نشان می دهد که توسعه مبتنی بر قصد، در بخش بالایی فرایند، بیان قصد را از ساخت زمینه جدا می کند؛ زمینه ای که در ادامه امکان اجرای مستقل عامل را در بخش پایینی فراهم می آورد. بیان قصد از طریق

گفت وگویی ساختارمند انجام می شود و به تولید مصنوعات ماندگار قصد می انجامد. این مصنوعات بذر اولیه زمینه هستند و می توانند برای ساخت سایر بخش های زمینه نیز به کار روند. به این ترتیب، اطلاعاتی کامل و اقدام پذیر ایجاد می شود که عامل می تواند بر اساس آموزش های مدل خود از آن برای اجرا و در نهایت راستی آزمایی پیاده سازی استفاده کند.



شکل ۱.۳. الگوی ذهنی توسعه مبتنی بر قصد: تمرکز بر بیان موثر قصد برای مهندسی زمینه، به گونه ای که عامل بتواند مستقلا اجرا کند و نتایجی همسو تحویل دهد.

۱.۴. مهار و هدایت قصد

در شکل ۱.۳ دیدیم که توسعه مبتنی بر قصد چگونه قصد ما را از طریق گفت وگوی ساختارمند آشکار می کند، آن را در مصنوعات ماندگار ثبت می کند و از زمینه حاصل برای اجرای مستقل عامل بهره می برد. بخشی از این

جریان که در آن قصد قابل استفاده را از دل ایده های مبهم بیرون می کشیم، «مهار و هدایت قصد» نام دارد. برای درک این مفهوم، ابتدا باید اصطلاح «مهار» را بشناسیم.

اصطلاح مهار (Harness) در گفت وگوهای هوش مصنوعی زیاد به کار می رود: مهار عامل، مهندسی مهار، مهار زمینه و مانند آن. برای روشن شدن معنای این اصطلاح، بهتر است به یکی از تعریف های رایج عامل نگاه کنیم.

بر اساس تعریفی که در مقاله ای با عنوان «کالبدشناسی مهار عامل» در وبلاگ رسمی لنگ چین در سال ۲۰۲۶ آمده است، عامل ترکیبی از یک مدل و یک مهار است؛ مهار نیز ابزارها، مهارت ها، پرامپت های سیستمی و مواردی از این دست را در بر می گیرد.

به بیان ساده، مهار در زمینه عامل به هر چیزی غیر از خود مدل گفته می شود که به ما امکان می دهد از ظرفیت خام آن مدل، کاری مفید استخراج کنیم. این مفهوم شباهت زیادی دارد به بادبان ها و سازوکارهای وابسته به آن ها در یک قایق بادبانی که به ما کمک می کنند از انرژی بالقوه باد برای پیمودن اقیانوس ها بهره ببریم.

مهار اجرا

مهارها را می توان برای کاربردهای خاص تخصصی کرد. برای مثال، Claude Cowork یک عامل دسکتاپ برای کارهای دانشی است و مهار همراه آن برای اجرای فعالیت های دانشی مانند کارهای حقوقی، مالی و بازاریابی طراحی شده است. در مقابل، Claude Code به طور ویژه برای برنامه نویسان تخصصی شده و هدف آن ساخت سامانه های نرم افزاری است. هر دو ممکن است از یک مدل یکسان استفاده کنند، اما کارهای متفاوتی را از آن استخراج می کنند. اگر تشبیه پیشین را ادامه دهیم، همان طور که بادبان های یک قایق بادبانی انرژی باد را برای جابه جایی مهار می کنند، توربین های بادی نیز می توانند از همان انرژی برای تولید برق بهره بگیرند. مقصود از این تشبیه آن است که مهارهای متفاوت می توانند یک ظرفیت خام یکسان را به دسته های متفاوتی از کار مفید تبدیل کنند.

در مجموع، این چیدمان عامل محور به ما کمک می کند بدون انجام مستقیم کارها، آن ها را اجرا کنیم. از آنجا که تمرکز اصلی این چیدمان بر خودِ اجراست، این نوع مهار را «مهار اجرا» می نامیم.

برای بهره برداری حداکثری از قابلیت های مدل و هماهنگ سازی های پیچیده عامل ها در لایه مهار اجرا، به یک انتزاع در سطحی بالاتر نیاز داریم؛ انتزاعی که مسئولیت های کامل را مدیریت کند، برای مثال ساخت یک

قابلیت کامل، نه صرفا نوشتن یک تابع. چنین لایه ای باید به ما کمک کند قصد خود را بیان کنیم و زمینه ای را شکل دهیم که مهار اجرا را برای انجام مستقل وظایف پیچیده و تحویل نتایج همسو هدایت می کند.

مهار قصد

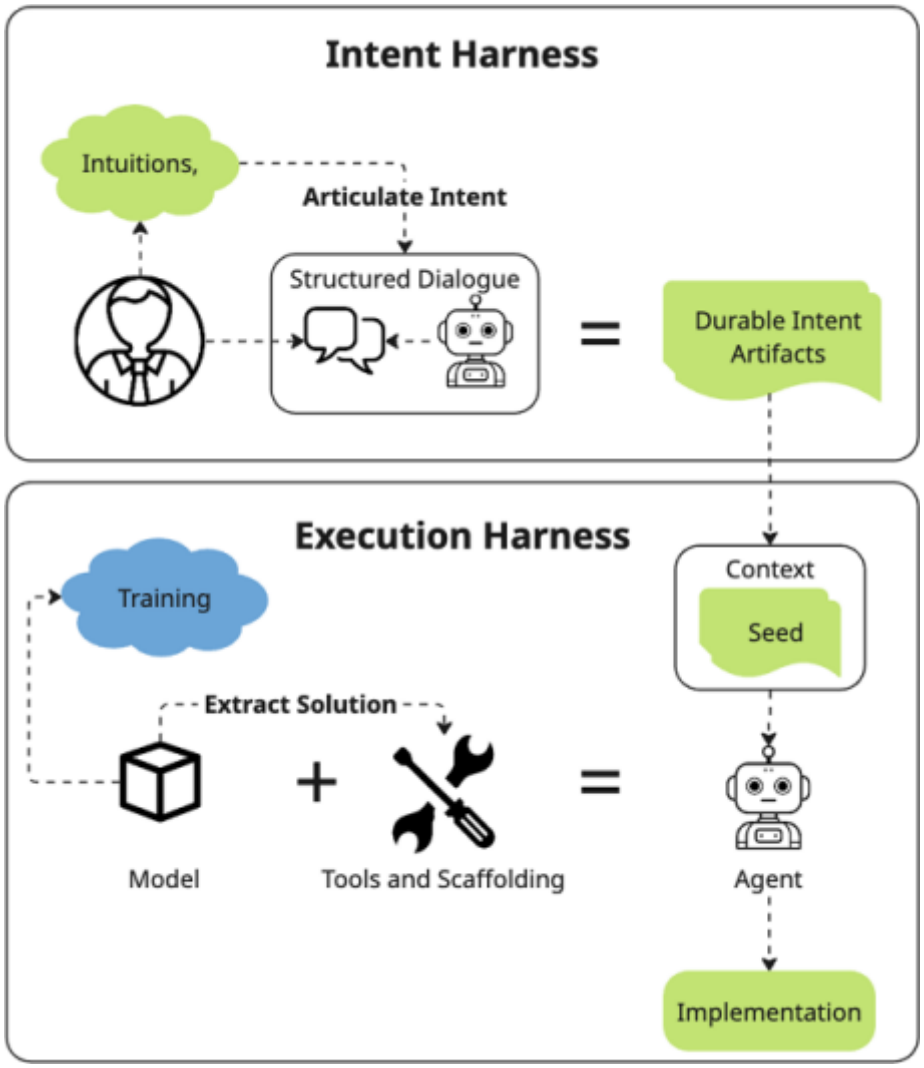
مهار قصد به ما امکان می دهد شهودها و ایده های خود را پالایش کنیم و قصد را به شکلی روشن بیان کنیم؛ در نتیجه، زمینه مهار اجرا را برای دستیابی به نتایج همسو شکل می دهیم.

شکل ۱.۴ نشان می دهد که مهار قصد در لایه ای بالاتر از مهار اجرا قرار می گیرد. وظیفه مهار قصد، ممکن کردن بیان قصد از طریق گفت وگوی ساختارمند میان انسان ها و عامل هاست؛ گفت وگویی که مصنوعات ماندگار قصد را تولید می کند و بذر اولیه زمینه را فراهم می سازد. مهار اجرا بر اساس این زمینه می تواند پیاده سازی را تولید کند.

با استفاده از تشبیه قایق بادبانی و توربین بادی، انرژی بالقوه خام در اینجا ایده ها و شهودهای ما هستند. گفت وگوی ساختارمند کمک می کند این مواد خام را استخراج و پالایش کنیم و به مصنوعات کاربردی قصد تبدیل کنیم؛ مصنوعاتی که سپس زمینه مهار اجرا را شکل می دهند.

یکی از ویژگی های مهم مهار قصد، ماندگاری مصنوعاتی است که تولید می کند؛ برای مثال، مشخصات و آزمون هایی که در شکل ۱.۳ دیدیم. این مصنوعات از یک نشست فراتر می روند و حتی ممکن است بیشتر از خود پیاده سازی عمر کنند. از نظر نظری، اگر توانسته باشیم از طریق گفت وگوی ساختارمند زمینه را به درستی بسازیم، خود پیاده سازی را نیز می توان دوباره از روی همین زمینه تولید کرد. مصنوع قصد، سطح اصلی کنترل است و پیاده سازی یا کد، نمود آن به شمار می رود.

هر ابزار و فروشنده ممکن است این قابلیت ها را به شکلی متفاوت بسته بندی کند. برخی ابزارها ممکن است هم مهار قصد و هم مهار اجرا را پوشش دهند و برخی دیگر فقط بخش هایی از آن را ارائه کنند. داشتن الگوی ذهنی شکل ۱.۴ مهم است تا بتوانیم قابلیت های عرضه شده را دسته بندی کنیم و تشخیص دهیم آیا پشته کامل مورد نیاز برای مهندسی نرم افزار بومی هوش مصنوعی را در اختیار داریم یا خیر.



شکل ۱.۴. مهار قصد یک سطح بالاتر از مهار اجرا قرار دارد و به ما کمک می کند قصد را بیان کنیم و زمینه لازم برای مهار اجرا را مهندسی کنیم.

۱.۴.۱. سازوکارهای بیان قصد

پس مهار قصد از چه اجزایی تشکیل شده است؟ در هسته خود، مجموعه ای از فنون و رویکردها برای انجام و ثبت گفت وگوی ساختارمند میان انسان و هوش مصنوعی است. این فنون از نظر زمان و شیوه انجام گفت وگوی ساختارمند و نیز از نظر شکلی که قصد هدایت کننده پس از گفت وگو به خود می گیرد، با یکدیگر تفاوت دارند.

دو رویکرد اصلی عبارت اند از توسعه مبتنی بر مشخصات و گردش کارهای تکرار خودمختار؛ این دو رویکرد رابطه های متفاوتی میان قصد و اجرا ایجاد می کنند.

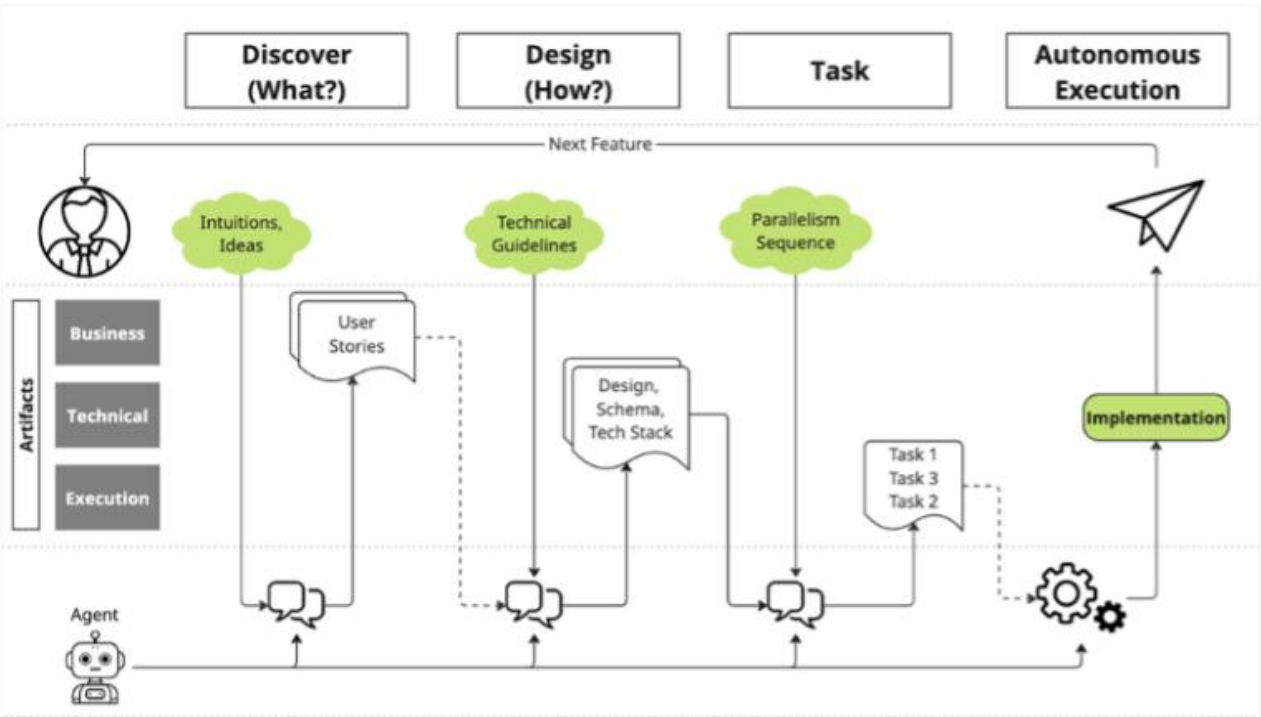
قصد به مثابه هدف: مشخصات در توسعه مبتنی بر مشخصات

در توسعه مبتنی بر مشخصات، گفت وگوی انسان و عامل حول سه لایه سازمان می یابد:

- چیستی (کشف): زمینه کسب وکار و معیارهای پذیرش؛
- چگونگی (طراحی): رویکرد فنی و محدودیت های معماری؛
- وظایف: برنامه اجرایی ای که عامل می تواند مستقیما بر اساس آن عمل کند.

این ساختار گفت وگوی ما را در هر مرحله روی یک جنبه متمرکز نگه می دارد. برای مثال، در مرحله نخست که موارد کاربرد کسب وکار را کشف می کنیم، گفت وگو وارد جزئیات پیاده سازی نمی شود. در مرحله طراحی نیز فقط بر تصمیم های معماری مبتنی بر مورد کاربرد کسب وکار تمرکز می کنیم و در نهایت، تعریف وظایف بر اساس مراحل پیشین انجام می شود. هر یک از این مراحل شامل گفت وگوی ما با هوش مصنوعی است؛ گفت وگویی که در آن چارچوب بحث را تعیین می کنیم و آن را متمرکز، مرتبط و منسجم نگه می داریم تا زمینه ای برای اجرای مستقل عامل مهندسی شود.

شکل ۱.۵ به صورت تصویری نشان می دهد که توسعه مبتنی بر مشخصات چگونه در هر مرحله، یعنی کشف، طراحی و تعریف وظایف، گفت وگوهای متمرکزی میان انسان و عامل برقرار می کند و مصنوعات ماندگار قصد را به وجود می آورد؛ در اینجا این مصنوعات همان فایل های مشخصات هستند. پیاده سازی فقط زمانی آغاز می شود که از طریق بیان قصد، میان انسان و عامل همسویی حاصل شده باشد. نتیجه این فرایند، پیاده سازی ای است که با خواسته ما همسویی زیادی دارد. این مشخصات ماندگار همچنین اطلاعات بنیادینی برای قابلیت بعدی فراهم می کنند.



شکل ۱.۵. نمایی کلی از اینکه توسعه مبتنی بر مشخصات چگونه از طریق مراحل کشف، طراحی و تعریف وظایف به بیان قصد کمک می کند و زمینه اجرای خودمختار را فراهم می سازد.

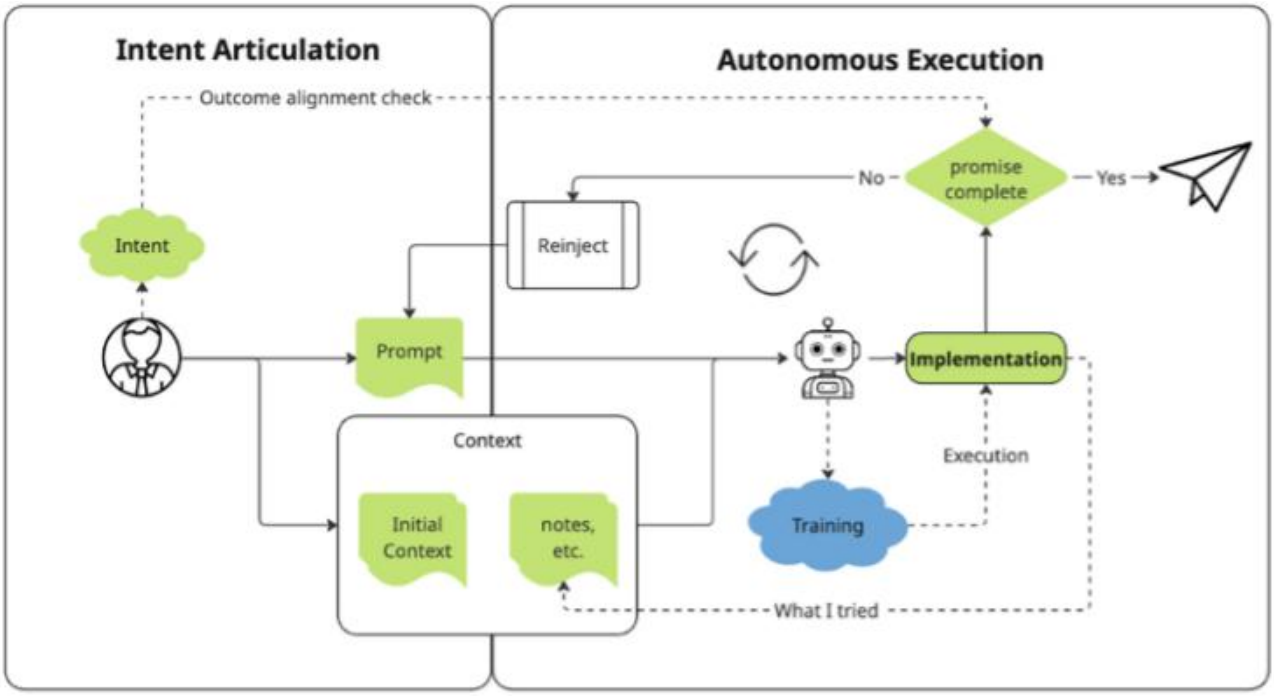
در راستای هدف های توسعه مبتنی بر قصد، مشخصات در اینجا جایگاهی بنیادین در فرایند مهندسی نرم افزار ما دارند و به همین دلیل، مصنوعات ماندگار محسوب می شوند. این مشخصات ابزار بیان قصد هستند. این رویکرد چند مزیت فنی دارد؛ برای نمونه، محدودیت های پنجره زمینه و فشرده سازی را کاهش می دهد، زیرا هر یک از گفت وگوهای مراحل فوق می توانند در نشست هایی جداگانه و با پنجره زمینه ای تازه انجام شوند. همچنین می توانیم در مراحل مختلف، مدل ها را با یکدیگر ترکیب کنیم تا از نقاط قوت هر یک بهره ببریم و حتی آن ها را در برابر یکدیگر قرار دهیم تا نتایجی متوازن تر به دست آید. این مزایا، در کنار چندین فایده دیگر، باعث شده اند این رویکرد به عنوان پادزهری برای کد هوش مصنوعی ای که مطابق انتظار رفتار نمی کند و استدلال درباره آن یا تکامل دادنش دشوار است، به طور گسترده مورد توجه قرار گیرد. با این حال، شاید مهم ترین اثر توسعه مبتنی بر مشخصات، فرهنگی باشد نه فنی. درباره این رویکرد سوءبرداشت هایی نیز وجود دارد؛ از جمله اینکه آن را احیای مدل آبشاری می دانند.

قصد به مثابه تنظیم گر: حلقه ها در گردش کارهای تکرار خودمختار

در گردش کارهای تکرار خودمختار، گفت وگو به شکل دیگری آغاز می شود. برخلاف توسعه مبتنی بر مشخصات که در آن تمام گفت وگوی میان ما و عامل پیش از آغاز اجرا انجام شده است، در اینجا ابتدا مشخص می کنیم موفقیت چه معنایی دارد و آن را به صورت یک محدودیت هدایت کننده تعیین می کنیم. سپس عامل گفت وگو را از طریق تکرار ادامه می دهد. در این حالت، قصد در محدودیت هدایت کننده تجسم می یابد؛ محدودیتی که همگرایی عامل به سوی راه حل را هدایت می کند و در عین حال، آموخته های عامل را به عنوان ورودی تکرارهای بعدی ثبت می کند.

حلقه های رالف (Ralph Loops) این مفهوم را رواج دادند. در هسته خود، حلقه رالف یک حلقه while است که همان پرامپت را بارها اجرا می کند تا هدفی حاصل شود یا حداکثر تعداد تلاش ها به پایان برسد. هر تکراری که نتواند محدودیت تعیین شده را برآورده کند، ورودی یادگیری تازه ای به زمینه اضافه می کند. زمانی که حلقه همگرا می شود، یادگیری انباشته شده عملا به مصنوعی تبدیل می شود که به نوعی نقش مشخصات را ایفا می کند. این مصنوع یا مشخصات را نمی شد از ابتدا نوشت، زیرا تنها راه آشکار شدن آن، اجرای کار بود.

شکل ۱.۶ به صورت تصویری نشان می دهد که قصد در معیارهای هدایت کننده تجسم یافته است؛ در این مثال، این معیارها همان وعده یا تعهد نهایی هستند.



شکل ۱.۶. نمایی کلی از گردش کارهای تعامل خودمختار که فقط زمینه اولیه و معیارهای حاکم بر تکمیل کار را فراهم می کنند و به عامل اجازه می دهند بدون آنکه لازم باشد دقیقا چگونگی دستیابی به نتیجه را شرح دهیم، به تدریج به سوی راه حل همگرا شود.

در حالی که هنگام بیان قصد فقط اطلاعات اولیه به زمینه افزوده می شود، عامل در جریان تکرار و گفت وگو با خودش، بقیه زمینه را به روزرسانی می کند. هر چرخه اجرا هم پیاده سازی و هم آموخته های حاصل از آن تلاش را تولید می کند.

گردش کارهای تکرار خودمختار بسیار فراتر از یک حلقه ساده هستند؛ از جمله ساختارهای هدایت کننده متفاوت، شیوه صورت بندی محدودیت ها برای امکان راستی آزمایی و زمان مناسب استفاده از تکرار به جای آنکه صرفا راه حلی موقت باشد.

مقایسه مشخصات و حلقه ها: کدام را چه زمانی به کار ببریم؟

مشخصات زمانی بهترین کارایی را دارند که بتوانیم قصد را با جزئیات کافی بیان کنیم تا پیاده سازی مستقیما بر اساس آن هدایت شود. می دانیم قابلیت باید چه کاری انجام دهد، چگونه باید ساخته شود و کدام معیارهای پذیرش تکمیل آن را تعریف می کنند. قابلیت بازنشانی گذرواژه نمونه خوبی است: می توانیم مشخص کنیم که

هدف قابلیت، بازیابی حساب به خطر افتاده است، نشست ها باید باطل شوند، پیوند بازنشانی باید در بازه زمانی معینی منقضی شود و تجربه کاربری باید فوریت موضوع را منتقل کند.

حلقه های هدایت کننده زمانی بهترین کارایی را دارند که بتوان قصد را در قالب یک محدودیت بیان کرد، اما مسیر برآورده کردن آن محدودیت فقط از طریق اجرا کشف شود. برای مثال، هنگام انتقال داده از یک سامانه قدیمی، ممکن است ناسازگاری ها فقط با تلاش برای انجام مهاجرت آشکار شوند. به محض آنکه یک دسته از مشکلات داده را برطرف کنیم، دسته بعدی خود را نشان می دهد.

نکته: هنگام انتخاب میان مشخصات و حلقه هدایت کننده، از خود بپرسید آیا می توان مسیر دستیابی به نتیجه را از ابتدا بیان کرد یا این مسیر فقط در جریان اجرا آشکار می شود. اگر مسیر از پیش قابل شناخت است، مشخصات انتخاب مناسبی هستند. اگر مسیر فقط با اجرای کار نمایان می شود، حلقه گزینه مناسب تری است. بسیاری از قابلیت های واقعی ترکیبی از هر دو رویکرد هستند: مشخصات چارچوب کلی کار را تعیین می کنند و حلقه ها بخش هایی را مدیریت می کنند که بیان جزئیاتشان از ابتدا ممکن نیست.

مشخصات و حلقه ها متضاد یا مانعۀ الجمع نیستند. این دو به شکل های متفاوتی از مسئله پاسخ می دهند و اغلب در نقش هایی مکمل کنار یکدیگر به کار می روند. در کنار هم، اجزای اصلی مهار قصد را تشکیل می دهند.

۱.۵. بیان قصد به گلوگاه تازه تبدیل شده است

در گردش کارهای انسان محور، می توانیم تا حدی ناهمسویی نتایج ناشی از بیان ضعیف قصد و دوباره کاری حاصل از آن را تحمل کنیم. سرعتی که انسان ها با آن پیاده سازی را تحویل می دهند به اندازه ای پایین است که بازبین ها فرصت تنفس داشته باشند. حتی می توانستیم با بیان غیربهینه قصد نیز کنار بیاییم. اما وقتی عامل های برنامه نویسی وارد جریان کار می شوند و گردش کار همچنان انسان محور باقی می ماند، هر وظیفه ای که قصد آن به درستی بیان نشده باشد، زنجیره ای از نتایج ناهمسو ایجاد می کند. سیل مداوم پیاده سازی های نامناسب، چرخه ای از مشکل و اصلاح به راه می اندازد که همه افراد درگیر را فرسوده می کند.

۱.۵.۱. نداشتن الگوی ذهنی از کد

وقتی بدون کمک هوش مصنوعی کد می نویسیم، به طور طبیعی درکی از محل قرار گرفتن هر بخش از قابلیت ها به دست می آوریم. می دانیم نشست ها چگونه مدیریت می شوند، چرا جدول توکن به شکل خاصی طراحی شده است و هنگام تغییر بخش احراز هویت، چه چیزهایی ممکن است خراب شوند. وقتی لازم است مسئله ای

را برطرف کنیم، از این درک برای ایجاد تغییرات مناسب استفاده می کنیم و همین کار دانش ما از سامانه را دقیق تر می کند. با گذشت زمان، این درک مشترک کمبود مستندات و حتی سازمان دهی ضعیف کد را جبران می کند. به این وضعیت «الگوی ذهنی» ما از برنامه می گوییم. همین الگوی ذهنی است که مهندس آشنا با یک پایگاه کد را از نیروی تازه وارد متمایز می کند. حتی اگر نیروی تازه وارد مهندسی بسیار باتجربه باشد، ممکن است به سرعت کسی که مدت ها با آن کد زندگی کرده است حرکت نکند.

با ورود عامل های برنامه نویسی، کار ما از نوشتن کد به بازبینی آن تغییر می کند و این وضعیت دوراهی دشواری به وجود می آورد. اگر با دقت کامل بازبینی کنیم، خودمان به گلوگاه تبدیل می شویم، زیرا عامل سریع تر از آن کد تولید می کند که بتوانیم آن را جذب و درک کنیم. اگر بازبینی سطحی انجام دهیم یا اصلا آن را کنار بگذاریم، الگوی ذهنی ای شکل نمی گیرد و در عوض باید هزینه بازیابی و رفع نقص هایی را بپردازیم که عامل وارد کرده است. در هر دو حالت، برای هر تغییر و اصلاح بیشتر به عامل وابسته می شویم و در پایگاه کد خودمان احساس می کنیم تازه وارد هستیم. با این حال، مسئولیت این کد بر عهده ماست، نه عامل برنامه نویسی. پاسخگو بودن در قبال راه حل هایی که دیگر نمی توانیم توضیحشان دهیم، هم ناامیدکننده است و هم خطرناک. در نهایت فقط سرعت تولید کد افزایش یافته است. ممکن است ارزش تحویل شده اصلا بهبود پیدا نکند، زیرا همه مراحل بعدی به دلیل بازبینی کند شده اند. به این ترتیب، نبود الگوی ذهنی، بازبینی را به گلوگاه تبدیل می کند.

برای رفع این گلوگاه باید به مراحل بالادستی بازگردیم و خود مسئله بیان قصد را که علت ریشه ای است برطرف کنیم.

۱.۵.۲. بیان قصد را اصلاح کنید، نه پیاده سازی را

واکنش طبیعی به گلوگاه بازبینی که پیش تر دیدیم، استفاده از عامل های بیشتر برای حل مسئله است. اما اگر مثال بازنشانی گذرواژه را در نظر بگیریم، یک عامل بازبین که از همان پرامپت عامل برنامه نویس استفاده می کند نیز احتمالا باطل کردن نشست ها را از قلم می اندازد؛ زیرا اطلاعات بیشتری درباره قصد اصلی در اختیار ندارد. دست کم بازبین انسانی، قصد ناگفته را در ذهن خود داشت و به همین دلیل می توانست شکاف را تشخیص دهد.

به جای تلاش برای حل مسئله در مرحله بازبینی، عملی تر آن است که علت ریشه ای، یعنی بیان ناقص قصد، را برطرف کنیم. در گردش کارهای انسان محور، واکنش رایج به یک پیاده سازی ناهمسو این بود که آن را وصله

کنیم، منتشر کنیم و رفع علت اصلی را به روز دیگری موکول کنیم؛ زیرا بخش عمده زمان ما صرف نوشتن کد می شد. با ورود عامل ها، محاسبه تغییر می کند. با کمک هوش مصنوعی، زمانی را که قبلا صرف نوشتن کد می کردیم می توانیم صرف فهمیدن این کنیم که چرا عامل باطل کردن نشست ها را از قلم انداخته است. این کار به ما کمک می کند گفت وگوی میان انسان و عامل یا مصنوعاتی را که حاصل آن را ثبت کرده اند، مانند مشخصات یا آزمون های پذیرش، بهبود دهیم و سپس پیاده سازی را از روی قصد روشن تر دوباره تولید کنیم. برخلاف اصلاح موردی یک پیاده سازی مشخص، بهبود بیان قصد مانع تکرار همان دسته از شکاف ها در قابلیت های آینده می شود.

این شیوه کار، بی میلی عمیق ما به حذف کد را به چالش می کشد. بازنویسی قابلیت ها یا ساخت دوباره برنامه ها بدون کمک هوش مصنوعی پرهزینه و زمان بر بود؛ بنابراین، حفظ کد موجود و وصله کردن آن اغلب انتخابی عملی محسوب می شد. اما وقتی عامل ها می توانند کد را به سرعت دوباره تولید کنند، نگه داشتن پیاده سازی معیوب ممکن است پرهزینه تر از کنار گذاشتن آن باشد. عاملی که چنین پیاده سازی ای را وصله می کند، باید هم کد موجودی را در نظر بگیرد که بر اساس قصد ناقص تولید شده و هم پرامپت اصلاح شده ای را که برای رفع همان نقص ارائه شده است. این وضعیت بار زمینه عامل را افزایش می دهد و آن را مجبور می کند منابع راهنمای متعارض را با یکدیگر سازگار کند. اگر پیاده سازی معیوب را کنار بگذاریم و آن را بر اساس قصد اصلاح شده از نو تولید کنیم، عامل می تواند با یک منبع حقیقت روشن تر کار کند و تمام تمرکزش را بر تولید پیاده سازی ای بگذارد که آن قصد را برآورده می کند.

نکته: غریزه حفظ کد از طریق وصله کردن یک پیاده سازی ناهمسو، عادتی است که از گردش کارهای انسان محور به ارث برده ایم؛ گردش کارهایی که در آن ها تولید کد پرهزینه بود. در گردش کارهای عامل محور، وصله کردن کد بر همان فرض های نادرستی تکیه می کند که باعث ناهمسویی اولیه شده اند. تولید دوباره کد بر اساس قصد اصلاح شده معمولا ارزان تر از تعمیر پیاده سازی معیوب است.

۱.۶. چه زمانی نباید از توسعه مبتنی بر قصد استفاده کرد؟

فنون توسعه مبتنی بر قصد، مهندسی نرم افزار بومی هوش مصنوعی را پایدار می کنند. برخلاف کدنویسی وایبی، توسعه مبتنی بر قصد در گذر زمان دچار فرسودگی نمی شود. در واقع، مزایای آن با گذشت زمان آشکارتر می

شوند. با این حال، این مزیت مستلزم سرمایه گذاری زمانی برای هر قابلیت تازه، رفع هر اشکال، بهبود و مانند آن در سراسر چرخه تکامل برنامه یا محصول است.

کدنویسی وایبی ذاتا رویکرد بدی نیست، اما برای ساخت و نگهداری بلندمدت نرم افزار روش عملی ای به شمار نمی رود. با این حال، برای نیازهای فوری، مانند ساخت نمونه اولیه برای نمایش به سرمایه گذاران اولیه یا اثبات مفهوم برای ارزیابی ظاهر و عملکرد یک فناوری، ممکن است کدنویسی وایبی گزینه بهتری باشد. این رویکرد بسیار سریع تر است و به یادگیری کمک می کند. پس از اثبات ایده، می توان آموخته های حاصل را به کار گرفت و ساخت نسخه تولیدی همان نرم افزار را از ابتدا، با رویکردی سنجیده تر و مبتنی بر قصد، آغاز کرد. همچنین ممکن است برای برنامه های بسیار ساده، مانند ابزارهای یک بارمصرف که چند انحراف در پیاده سازی آن ها قابل قبول است و می توان با چند پرامپت اصلاحی سریع برطرفشان کرد، سرمایه گذاری زمانی برای گفت وگوی ساختارمند در چارچوب توسعه مبتنی بر قصد توجیه پذیر نباشد.

در هسته توسعه مبتنی بر قصد، همکاری میان ذی نفعان قرار دارد. از آنجا که این رویکرد بعد فرهنگی مهمی دارد، پذیرش آن فقط در سطح فردی بخش قابل توجهی از ارزش بالقوه را نادیده می گیرد. پذیرش توسعه مبتنی بر قصد قابلیتی سازمانی است که باید پرورش داده شود، نه صرفا یک رویه فنی که نصب و راه اندازی شود. کسانی که تجربه پذیرش اسکرام در سازمان های بزرگ را داشته اند، این الگو را می شناسند. بدون همسویی فرهنگی، اگر رویکردهایی مانند توسعه مبتنی بر مشخصات را صرفا ابتکارهایی در زمینه ابزارهای توسعه دهنده بدانیم، خطر پدیده «Specfall» وجود دارد. این پدیده معادل توسعه مبتنی بر مشخصات برای «Scrumerfall» است؛ وضعیتی که در آن تیم ها اصطلاحات و آیین های اسکرام را می پذیرند، بی آنکه ذهنیت زیربنایی آن را درونی کنند. در Specfall، مشخصات به اسنادی تبدیل می شوند که باید تولید و میان افراد توزیع شوند، نه ابزارهای کنترلی مشترکی برای بیان قصد.

خلاصه فصل اول

- قصد، بیان اقدام پذیر هدف برای یک تغییر مشخص است. قصد، چرایی گسترده تر کار را به اندازه ای ملموس می کند که بتواند پیاده سازی، راستی آزمایی و تصمیم های طراحی بعدی را هدایت کند.
- بیان قصد، شهود خام را به زمینه ای مهندسی و قابل استفاده تبدیل می کند. داستان های کاربر، معیارهای پذیرش، BDD و TDD مدت هاست به تیم ها کمک می کنند از طریق گفت وگوی ساختارمند قصد را آشکار کنند؛ توسعه مبتنی بر قصد این بنیاد را برای توسعه بومی هوش مصنوعی گسترش می دهد.
- توسعه مبتنی بر قصد از گفت وگوی ساختارمند انسان و هوش مصنوعی برای استخراج، پالایش و حفظ قصد استفاده می کند. هدف آن ثبت قصد در مصنوعات ماندگاری است که از یک گفت وگو، مدل یا نشست برنامه نویسی فراتر می روند.
- عامل های برنامه نویسی، مهارهای اجرا هستند؛ در حالی که مهارهای قصد مشخص می کنند این عامل ها بر اساس چه چیزی اجرا کنند. مهار قصد در سطحی بالاتر از مهار اجرا قرار می گیرد و زمینه ساختارمند لازم را برای اجرای مستقل تر و همسوتر عامل فراهم می کند.
- توسعه مبتنی بر مشخصات و گردش کارهای تکرار خودمختار، دو رویکرد مکمل در توسعه مبتنی بر قصد هستند. توسعه مبتنی بر مشخصات از مشخصات به عنوان ابزار بیان قصد استفاده می کند، در حالی که در گردش کارهای تکرار خودمختار، قصد در محدودیت های هدایت کننده تجسم می یابد؛ محدودیت هایی که اجرا، یادگیری و همگرایی تکرارشونده را هدایت می کنند.
- توسعه مبتنی بر قصد اهمیت دارد، زیرا تولید سریع تر کد، گلوگاه را از نوشتن کد به بیان قصد منتقل می کند. بیان ضعیف قصد به موجی از بازبینی و دوباره کاری می انجامد؛ در حالی که مصنوعات ماندگار قصد، مانند مشخصات، به انسان ها کمک می کنند زمینه را تعریف کنند، تخصص را ثبت کنند و نتایج را راستی آزمایی کنند و به این ترتیب، پایه های کارخانه های نرم افزاری را بنا می گذارند.

جهت خرید و دانلود و مطالعه کامل این کتاب و مشاهده کتاب های بیشتر روی لینک زیر کلیک نمایید:

خرید نسخه کامل کتاب