نگاهی فلسفی-تکنیکال به هوش مصنوعی مولد

حمیدرضا زمانیان

۲۰۲۶

عنوان: نگاهی فلسفی-فنی به هوش مصنوعی مولد

عنوان اصلی: A Philosophically Technical View Into Generative AI

نویسنده: حمیدرضا زمانیان

زبان: فارسی

انتشار: مهر ۱۴۰۵

وب‌سایت نویسنده: kyrovert.com

مجوز

فهرست مطالب

پیشگفتار

فرض کن با کسی بحث می‌کنی و او موفق می‌شود بنیان‌های نظام باورهایت را نابود کند. باورهایی را که تمام زندگی و شخصیت را بر آن‌ها بنا کرده‌ای از دست می‌دهی. چرا در چنین موقعیتی این‌قدر درمانده می‌شوی؟ خب، چون این وضعیت دقیقاً شبیه وقتی است که کسی به تو خیانت می‌کند. دیگر نمی‌توانی به خودت اعتماد کنی. چطور این‌قدر بد اشتباه کردی؟ نکند باورهای دیگرت هم غلط باشند؟

در چنین موقعیتی، وضعی که در آن قرار می‌گیری کاملاً منطقی است. وقتی بعضی از بنیان‌های یک نظام را زیر سؤال می‌بری، طبیعی است که هم‌زمان خیلی چیزهای دیگر را هم زیر سؤال ببری. باید بنیان‌های دیگر را هم با این نگرش تازه بررسی کنی تا ایرادهای پنهانشان را پیدا کنی.

بعضی صنایع، مخصوصاً صنعت فناوری، با ورود هوش مصنوعی مولد دستخوش بازنگری گسترده‌ای شده‌اند. گردش‌کارها (workflows) به‌شدت در حال تغییرند. مهندس نرم‌افزار بودن دیگر تقریباً شبیه چیزی نیست که حتی یک سال پیش بود. این وضعیت طبیعتاً سردردهای زیادی ایجاد می‌کند که بیشترشان به فلسفه برای مدیریت‌شدن نیاز دارند. فلسفه شیوهٔ فکرکردن دربارهٔ جهان است؛ چیزی نیست که بتوانی انتخاب کنی به آن مجهز شوی. دربارهٔ هوش مصنوعی و این گردش‌کارِ تازه، چیزهای زیادی هست که نمی‌دانیم. خودِ ابزار غیرقطعی است و همه دارند با آن آزمایش می‌کنند. برای فهمیدن فایدهٔ واقعی هوش مصنوعی به سال‌ها، شاید حتی دهه‌ها، زمان نیاز داریم. اما نمی‌توانیم آن‌قدر صبر کنیم و با انتظارهای کاذب آینده و اعتبارمان را به خطر بیندازیم. **باید** حدس بزنیم.

این همان کاری است که این کتاب می‌خواهد انجام دهد. می‌توانی آن را مقاله‌ای بسیار طولانی در نظر بگیری؛ مجموعه‌ای از فکرها و باورهای مختلف که در یک جا جمع شده‌اند و داستانی منسجم آن‌ها را به هم وصل می‌کند. اول این کتاب را نوشتم چون به‌شدت درمانده و فرسوده بودم. لازم داشتم فکرهایم را بیرون بریزم و بفهمم کجا ایستاده‌ام. آن‌قدر فکر منسجم داشتم که آن‌ها را در قالب یک کتاب بیاورم و این کتاب آزمونِ انسجامشان بود. خوشحالم که از آن سربلند بیرون آمدم. این اولین کتاب من است و امیدوارم همان‌قدر که خودم از نوشتنش لذت بردم، تو هم از خواندنش لذت ببری.

اما اگر خوشت نیامد، با کمال میل به نظرهایت گوش می‌دهم! دوست دارم بدانم دربارهٔ این کتاب چه فکر می‌کنی، چه تجربه‌هایی داشته‌ای و به‌نظرت کجا کم گذاشته‌ام. من هم مثل هرکس دیگری مشتاقم این

دورهٔ تازه را بفهمم، حتی با اینکه یک کتاب کامل درباره‌اش نوشته‌ام. می‌توانی از طریق ایمیلم بـا من تمـاس بگیری:

hamid80zamanian@gmail.com

قالب‌های دیگر

نسخهٔ وب‌سایت، genai-book.kyrovert.com، یکی از راه‌های خواندن کتاب است، اما می‌تـوانی کتاب را در دو قالب دیگر هم دانلود و مطالعه کنی:

- PDF: اینجا کلیک کن
- Markdown: اینجا کلیک کن

این کتاب برای چه کسانی است؟

با اینکه عنوان کتاب طوری به‌نظر می‌رسد که انگار فقط برای برنامه‌نویس‌هاست، واقعاً این‌طور نیست. تمـام تلاشم را کردم که از اصـطلاحات فـنی اسـتفاده نکنم و کتاب را کـاملاً بـر پایـهٔ منطـق پیش بـبرم. این‌طـوری آدم‌های حوزه‌های دیگر هم می‌توانند موضوع را به تخصص خودشان ربط بدهنـد. همین‌طـور کمکم می‌کنـد روشن‌تر و منطقی‌تر بنویسم، بی‌آنکه این‌طرف و آن‌طرف مثال‌هایی وسط بکشم و خیال کنم استدلال درستی ساخته‌ام.

برنامه‌نویسـی شـاید مکـانیکی‌ترین حـوزه‌ای باشـد کـه بتـوانی پیـدا کـنی. این یعـنی می‌تـوانی نمونـهٔ مشـابه مفهوم‌هـایی را کـه توضـیح می‌دهم، تا حـدی در حـوزهٔ خـودت پیـدا کـنی. دلیل اینکـه این کتاب را کـاملاً تعمیم‌پـذیر و مسـتقل از حرفـه نکـردم این اسـت کـه حـال‌وهوایش را از دسـت می‌داد. من برنامه‌نویسـم و چیزهای زیادی دربارهٔ حرفه‌ام می‌دانم. تعمیم‌پـذیرکردن کـل کتاب و از دسـت نـدادن بعضـی نکته‌هـا واقعـاً سخت است، مخصوصاً چون بعضی جنبه‌ها و مشکل‌ها واقعاً به حوزه‌هـایی مثل برنامه‌نویسـی و ریاضـیات محدود می‌شوند.

بااین‌حال، تمام تلاشم را کردم که از جزئیات فنی دوری کنم. فقط چند اصـطلاح فـنی در کتاب هسـت کـه در بخش اصطلاحات توضیحشان داده‌ام. برای خواندن کتاب دانستن همین‌ها کافی است.

سلب مسئولیت

من متخصص هوش مصنوعی نیستم. دانشمند علوم کامپیوتر هم نیستم. و قطعاً هیچ مدرکی در فلسفه ندارم. فقط یک انسانم که زیاد فکر می‌کند و به‌اندازه‌ای فکر منسجم دارد که آن‌ها را روی کاغذ بیاورد. اگر دیدی ادعایی می‌کنم که حرفه‌ای به‌نظر می‌رسد، فقط به این خاطر است که درباره‌اش استدلال کرده‌ام؛ استدلالی که فکر می‌کنم هرکسی باید بتواند به آن برسد. پام را توی کفشی که اندازه‌ام نیست نمی‌کنم. پس هرقدر دلت می‌خواهد قضاوتم کن.

این کتاب را هوش مصنوعی تولید نکرده

تمام خط‌های تیرهٔ بلند و کوتاه را خودم گذاشته‌ام. این کتاب فقط با هوش مصنوعی ویرایش شده است. هیچ‌کدامش از صفر تولید نشده؛ چون چنین چیزی ممکن نبود و باید با تایپ‌کردن می‌فهمیدم دارم چه فکری می‌کنم. مجبور بودم فکرهای مختلفم را با دقت به هم وصل کنم تا دوباره گم نشوم. همین کتاب به‌تنهایی نشان می‌دهد چرا نمی‌توانستم به هوش مصنوعی اعتماد کنم تا این مقاله را برایم بنویسد.

نامه‌ای از سر قدردانی

اولش نمی‌خواستم فصلی دربارهٔ این موضوع اضافه کنم، چون دربارهٔ تلاش فنی‌ام برای نوشتن این کتاب واقعاً احساس ناامنی می‌کنم. حس می‌کنم بچه‌ای هستم که هیچ‌چیز نمی‌داند و با این‌همه اعتمادبه‌نفس دربارهٔ چیزها حرف می‌زند و خودش را مضحکه می‌کند. تازه این‌طور هم نیست که هنگام نوشتن کتاب از کسی کمک مستقیم گرفته باشم. فقط بعضی فکرهای خودم و فکرهای دیگران را کنار هم جمع کردم.

اما این همچنان کتاب من است و درواقع اولین کتابم هم هست. با وجود احساس ناامنی‌ام، از اینکه بالاخره تمامش کرده‌ام واقعاً راضی‌ام. نوشتنش حدود دو ماه طول کشید، اما بیشتر شبیه نتیجهٔ سفری یک‌عمره است. احساس می‌کنم باید از کسانی تشکر کنم که مستقیم یا غیرمستقیم در نوشتن این کتاب کمکم کردند (امیدوارم کسی را جا نیندازم).

از مادرم و پدرم به‌خاطر حمایت همیشگی‌شان و اینکه کمک کردند به آدمی که امروز هستم تبدیل شوم تشکر می‌کنم. از تمام کسانی که وقتی نیاز داشتم حرف بزنم کنارم بودند، به سخنرانی‌هایم گوش دادند و در بحث‌های فلسفی با من همراه شدند تشکر می‌کنم. از دوستانم، چه آن‌هایی که هنوز با هم دوستیم و چه آن‌هایی که دیگر نیستیم، ممنونم.

باید به‌طور ویژه از دو نفر از دوستانم تشکر کنم که تأثیر بزرگی بر مسیر شغلی‌ام گذاشتند: امیرحسین، به‌خاطر اینکه هنرمندی درجه یک است و به من نشان داد ارادهٔ واقعی چه شکلی است؛ و حسین، همکار و دوست چندین‌ساله‌ام، به‌خاطر اینکه مسیر شغلی و نگرشم را بیش‌تر از آنچه بتوانم بشمارم تغییر داد.

همچنین باید از چند نفر تشکر کنم که هیچ ایده‌ای ندارند من چه کسی هستم، اما مستقیم یا غیرمستقیم نقش بزرگی در به‌وجودآمدن این کتاب داشتند:

از جاناتان بلو (Jonathan Blow)، به‌خاطر تمام دانشی که به او مدیونم.

از اندرو هانت (Andrew Hunt) و دیوید توماس (David Thomas)، به‌خاطر کتاب فوق‌العاده‌شان، *برنامه نویس عملگرا* (*The Pragmatic Programmer*).

از توماس براش (Thomas Brush)، به‌خاطر پادکست‌های عالی‌اش و تمام مهمان‌هایش: توماس مالر (

Thomas Mahler)، جوناس تایرولر (Jonas Tyroller) و دیگران.

از ترنت کنوگا (Trent Kaniuga) و تایلر ادلین (Tyler Edlin)، دو هنرمند بزرگ که ویدئوهایشان نقش بزرگی در شکل‌دادن به مسیر شغلی‌ام و تغییر کامل زندگی‌ام داشت.

و از داکتر کِی (Dr. K)، به‌خاطر ویدئوهای عالی‌اش که چیزهای زیادی دربارهٔ خودم به من یاد دادند.

اصطلاحات

•**اسکریپت**: فایلی که کدی را در خود نگه می‌دارد تا اجرا شود.

•**کدبیس**: مجموعهٔ کامل کدهای منبعی که برای ساخت یک برنامهٔ نرم‌افزاری استفاده می‌شود، شامل همهٔ فایل‌ها، کتابخانه‌ها و فایل‌های پیکربندی.

•**ادغام کد**: پذیرفتن بخشی از کدِ توسعه‌یافته و واردکردنش به کدبیس اصلی؛ از شاخهٔ توسعه به شاخهٔ تولید.

•**فریم‌ورک**: مجموعه‌ای از مؤلفه‌های نرم‌افزاریِ قابل‌استفادهٔ مجدد که توسعهٔ برنامه‌های تازه را کارآمدتر می‌کند.

•**بازآرایی کد (refactoring)**: به‌روزرسانی کد بدون افزودن قابلیت تازه، فقط برای اینکه در آینده نگه‌داری‌پذیرتر باشد.

•**فایل skill یا مهارت**: فایلی که دستورهایی را در خود دارد تا مدل هوش مصنوعی در لحظه از آن‌ها استفاده کند؛ مثلاً مهارتی برای نوشتن توییت. هر وقت از دستیار هوش مصنوعی بخواهی برایت توییتی بنویسد، آن فایل را می‌خواند و برای انجام کار از دستورهایش استفاده می‌کند.

•**وایب‌کدینگ**: استفادهٔ کامل از هوش مصنوی برای تولید کد و ساخت نرم‌افزار، بدونِ دستی نوشتن.

فصل ۱

پیش از هوش مصنوعی چه بودیم؟

قبل از آنکه به هوش مصنوعی فکر کنیم، باید به این پرسش جواب بدهیم. ما به‌عنوان برنامه‌نویس چه‌کاره بودیم؟ هدفمان چه بود؟ چه چیزی ما را از تایپیست‌ها یا کدنویس‌ها متمایز می‌کرد؟ پول می‌گرفتیم چون با ده انگشت تایپ می‌کردیم، یا برای چیز دیگری؟

خب، بدیهی‌ترین جواب ــ جوابی که دربارهٔ خیلی از شغل‌های دیگر، اگر نگوییم همه‌شان، هم صدق می‌کند ــ این است که ما حل‌کنندهٔ مسئله‌ایم. ما فقط ویرایشگرهای کدی نیستیم که گوشت و پوست پوشیده باشند. اما اگر مسئله حل می‌کنیم، پس هوش مصنوعی چیست؟

تفاوت انسان و هوش مصنوعی

بگذار یک سؤال ازت بپرسم: چطور می‌شود یک چت‌بات هوش مصنوعی که خیلی بیشتر از تک‌تک ما می‌داند، باز هم یک‌سری کمبود داشته باشد؟ چطور همان هوش مصنوعی که ممکن است در هر حوزه‌ای که اسم ببری جوابی بسیار پیچیده و پخته به من بدهد، هنوز توهم‌زایی کند، برای هر دست شش انگشت بسازد یا صاف و پوست‌کنده دروغ بگوید؟ آدم‌ها این‌طوری نیستند. یک هنرمند حرفه‌ای همین‌طوری برای هر دست هفت انگشت نمی‌کشد و نمی‌گوید: «اوه، اشتباه کردم». یا عمدی است یا یک جای کار می‌لنگد.

پرسش این است که انسان‌ها و هوش مصنوعی، دست‌کم در مدل‌های امروزی، در شیوهٔ فکرکردن چه تفاوتی با هم دارند.

برای مثال، یک بچهٔ انسان را در نظر بگیر. بچه خیلی زود یاد می‌گیرد. چیزهای زیادی یاد می‌گیرد و کارهای شگفت‌انگیزی می‌کند که هیچ حیوان دیگری از پسشان برنمی‌آید. اما بااین‌حال هیچ‌چیز یادش نمی‌ماند. بیشتر آدم‌ها از دوران نوزادی تا چهار یا پنج‌سالگی تقریباً هیچ‌چیزی به یاد نمی‌آورند. دربارهٔ مدل‌های زبانی بزرگ درست برعکس است. می‌توانی یک متن کاملاً تصادفی به یک مدل زبانی بزرگ بدهی؛ متنی که هیچ انسانی نمی‌تواند حفظش کند، و مدل همان لحظه آن را برایت پس می‌دهد. دلیلش ماهیت حافظه‌محور (memory-oriented) مدل‌های زبانی بزرگ است.

موجودات حافظه‌محور

مدل‌های زبانی بزرگ اساساً با یک «سطل بزرگ داده» کار می‌کنند. شاید بگویی: «خب، حمید، آدم‌ها هم همین‌طور کار می‌کنند. برای همین بچه‌ها این‌قدر زود یاد می‌گیرند: هر روز کلی داده هضم می‌کنند». این حرف درست است، اما لزوماً به این معنی نیست که سازوکارشان یکی است. اتفاقاً همین مثال راه خیلی خوبی است تا ببینیم چرا. بچه زود یاد می‌گیرد، اما زود هم فراموش می‌کند. پس چه چیزی باقی می‌ماند؟ چه چیزی در او ذخیره می‌شود که کمک می‌کند رشد کند، نه اینکه دوباره به نوزادی برگردد؟ اگر یک مدل زبانی بزرگ همهٔ پارامترهایش را از دست بدهد، تبدیل می‌شود به کودن‌ترین چیزی که می‌شناسی. اما انسان ممکن است پنج سال از زندگی‌اش را فراموش کند و آن پنج سال بااین‌حال روی تمام عمرش اثر بگذارند؛ مثلاً تروماها همین کار را می‌کنند.

علتش این است که مغز انسان قدرتمندترین ماشین تشخیص الگوست و سامانهٔ بسیار ویژه‌ای برای مدیریت کانتکست دارد. هر چیزی که در آن سن‌وسال یاد می‌گیری تبدیل می‌شود به الگو، منطق و شیوهٔ فکرکردن. عین همان چیزها را حفظ نمی‌کنی؛ فقط یادت می‌ماند چه حسی به آن‌ها داشتی یا باید چطور با آن‌ها برخورد کنی. این روند هم هیچ‌وقت متوقف نمی‌شود؛ تا دم مرگ با خودت می‌بری‌اش. وقتی ده‌ساله‌ای شاید به یک قابلمهٔ داغ دست بزنی و بفهمی نباید به چیزهای روی اجاق دست زد. شاید هیچ‌وقت یادت نیاید به آن قابلمه دست زده‌ای، اما همیشه می‌دانی نباید به آن دست بزنی. و چون چیزی که ذخیره کرده‌ای یک الگوست، نه فقط یک نمونه، خیلی زود چیزهای دیگری را هم پیدا می‌کنی که داغ و خطرناک به‌نظر می‌رسند.

مغزت کتابخانهٔ عظیمی از الگوهاست. اما هرچه بزرگ‌تر می‌شوی، باید چیزهای بیشتری را هم حفظ کنی. باید زبان مادری‌ات، آدم‌های اطرافت، دوستانت، مدرسه‌ات، قوانین جامعه و غیره را به خاطر بسپاری. سعی می‌کنی همه‌چیز را به الگو تبدیل کنی؛ این کمک می‌کند به‌جای تکیه‌کردن به تجربه‌های صریح، از خطرهای احتمالی دوری کنی. مثلاً اگر قمه‌زدن به کسی غیرقانونی باشد، این کار با یک چاقوی کوچک آشپزخانه هم غیرقانونی است.

در برنامه‌نویسی مفهومی هست به اسم هارنس (محیط و سازوکارِ اجرا و کاربستِ مدل یا ایجنت) که با بالاگرفتن استفاده از ایجنت‌های هوش مصنوعی (AI agents) رایج شد. Sol GPT، 5 Fable و مدل‌های دیگر فقط خودِ مدل‌اند؛ مدل‌های زبانی بزرگ. چیزی که آن‌ها را با آن اجرا می‌کنی هارنس است: Claude

Cursor، Codex، Code و غیره. هارنس فقط خود ابزار نیست؛ شیوهٔ استفاده‌ات از مدل‌ها را هم شامل می‌شود. بهینه‌کردن روش استفاده‌ات از این ایجنت‌ها و مدل‌ها هم بخشی از هارنس توست. حالا دیگر خیلی‌ها قبول دارند که هارنس احتمالاً مهم‌ترین عامل در کیفیت خروجی هوش مصنوعی است.

مغزت قدرتمندترین هارنس است. اگر می‌توانستی به‌نحوی دانشی را که در مدل‌های فعلی هوش مصنوعی هست ــ حتی GPT 3.5 ــ به مغزت منتقل کنی، به مگامایندِ زمانهٔ ما تبدیل می‌شدی، نه صرفاً یک مدل پیشروی جدید. مجموعه‌دادهٔ تو خیلی محدود است، اما بااین‌حال می‌توانی کارهایی بکنی که هیچ مدل زبانی بزرگ امروزی از پسشان برنمی‌آید.

از طرف دیگر، هوش مصنوعی فعلی بر پایهٔ یک نظام پیش‌بینی کار می‌کند. درست است که امروز دیگر فقط «حدس‌زنِ کلمهٔ بعدی» نیست (هرچند کم‌وبیش همین است)، اما درست است که بر اساس این کار می‌کند: «بعد از این ورودی، رایج‌ترین زنجیرهٔ فکرها و جواب‌ها چیست؟» وقتی ورودی‌ات را وارد می‌کنی، با میلیاردها پارامتر مدل زبانی بزرگ ترکیب می‌شود و بر اساس اینکه ورودی‌ات چطور آن «خاطره‌ها» را جهت می‌دهد، الگویی شکل می‌گیرد. مثلاً اگر دربارهٔ گیاهان آپارتمانی بپرسی، هوش مصنوعی الگو را می‌بیند و با خودش می‌گوید: «خب، این بخش از داده‌های این قسمتِ پارامترها کاملاً با ورودی جور است. محتمل‌ترین چیزی که بعد از این سؤال می‌آید چیست؟ ... این جواب...». برای همین هم متن را کلمه‌به‌کلمه، یا دقیق‌تر بگویم توکن‌به‌توکن دریافت می‌کنی. (اگر می‌خواهی این بند را بهتر بفهمی، می‌توانی بدهی‌اش به چت‌بات محبوبت و از او بخواهی برایت توضیحش دهد. دارم بیش‌ازحد ساده‌سازی می‌کنم تا منطق اصلیِ لازم برای بحث فلسفی‌مان را بیرون بکشم.)

خلاصه‌اش این است: نظام پیش‌بینی انسان خیلی بهتر است، اما با مجموعه‌داده‌ای بسیار کوچک‌تر کار می‌کند و مثال‌های خیلی کمتری برای ارائه دارد. وقتی با برنامه‌نویس‌های سینیور (ارشد) برخورد می‌کنی، این موضوع برایت روشن می‌شود. معمولاً نمی‌توانند همهٔ قدم‌های ریز ساخت یک نرم‌افزار بزرگ را برایت ردیف کنند. فقط ابزارها، استکی را که بهتر است انتخاب کنی، و زیربنایی را که باید برای پروژه‌ات بسازی می‌شناسند. در طول مسیر از انبوه الگوهایشان استفاده می‌کنند تا خودشان و تیمشان را به‌سمت هدف هدایت کنند. توانایی ابرانسانی ندارند.

این موضوع همین‌طور توضیح می‌دهد چرا وقتی از هوش مصنوعی می‌خواهی فقط خلاق باشد، جواب نمی‌دهد. چیزهای خلاقانه معمولاً کمتر رایج‌اند. اطلاعات کمتری درباره‌شان وجود دارد، برای همین سنجش **عینی (آبجکتیو)** خلاقیتشان سخت است. آدم‌ها هرکدام نظر خودشان را دارند. یک نفر ممکن است چیزی را

خلاقانه بداند و یک نفر دیگر قبول نداشته باشد؛ شاید هم بعضی‌ها اصلاً به آن فکر نکرده باشند. منظور از این حرف **ذهنی (سابجکتیو)** بودن است. اما هدف هوش مصنوعی این است که عینی کار کند. برای خلاق‌شدن هیچ راهنمای مکتوبِ بی‌نقصی وجود ندارد ــ یا دست‌کم ما در مجموع هرچه درباره‌اش منتشر شده را کنار گذاشته‌ایم. بااین‌حال، مدل واحد هوش مصنوعی هر بار که همان سؤال را می‌پرسی جواب کاملاً متفاوتی نمی‌سازد، با اینکه هر گفت‌وگو با دیگری فرق دارد. همین چند روز پیش خواندم که اگر از یک مدل زبانی بزرگ بخواهی فهرستی از صد کلمهٔ تصادفی بسازد، آشکارسازهای هوش مصنوعی با اطمینان صددرصدی می‌گویند فهرست را هوش مصنوعی تولید کرده است. تصادفی نیست؛ مدل‌های زبانی بزرگ همین‌اند: حافظه‌محور (این بخش کتاب را قبل از کل ماجرای «واترمارک» نوشتم).

فصل ۲

آیا هوش مصنوعی ذاتاً با انسان فرق دارد؟

خب، این موضوع حسابی بحث‌برانگیز است. هرکسی نظری دارد. بعضی‌ها فکر می‌کنند ما فقط ماشینیم، بعضی‌ها به اشباح باور دارند، بعضی‌ها به روح اعتقاد دارند و بعضی‌ها می‌گویند تا ده سال دیگر حتی شغلی هم نخواهیم داشت. اینجا قرار نیست دربارهٔ هنر هوش مصنوعی حرف بزنم، اما می‌خواهم منطق کلیِ نگاه خودم به این بحث را توضیح بدهم.

بگذار با موضوع دیگری شروع کنم؛ بعداً برمی‌گردیم سراغ این یکی.

آیا ارادهٔ آزاد داریم؟

برای جواب‌دادن، اول باید بفهمیم چه چیزی اراده را آزاد می‌کند. اصلاً ارادهٔ آزاد یعنی چه؟

۱. معنای ارادهٔ آزاد

ارادهٔ آزاد اراده‌ای است که از پیش تعیین نشده باشد؛ اراده‌ای که تماماً یا دست‌کم تا حدی به خودت تعلق داشته باشد. هرچه در انتخاب‌هایت اختیار بیشتری داشته باشی، آزادانه‌تر انتخاب می‌کنی. تعریف ما از کلمهٔ «رضایت» هم همین است. بگذار دو مثال بزنم:

- **آدم‌های مست** نمی‌توانند رضایت بدهند. چرا؟ چون عملشان واقعاً از خودشان نیست. نمی‌توانند «انتخاب» کنند کاری را انجام دهند؛ مغزی که تحت‌تأثیر مستی است به‌جایشان تصمیم می‌گیرد.
- **بچه‌ها** نمی‌توانند رضایت بدهند، چون مغزشان هنوز به‌اندازهٔ کافی رشد نکرده است. درکشان از دنیا بی‌تردید محدود است. تقریباً هیچ تصوری از پیامدهای کارهایشان ندارند. ذاتاً ارادهٔ آزاد کمتری از یک بزرگسال دارند.

۲. ارادهٔ آزاد از کجا می‌آید؟

«از خدا»

فرض کنیم خدایی هست. آن خدا همه‌چیز را می‌داند. واقعاً همه‌چیز را. آینده را با ریزترین جزئیات می‌داند. بزرگ‌ترین پیشگوی دنیاست. اگر چنین خدایی وجود داشته باشد، آیا واقعاً ارادهٔ آزاد داریم؟ وقتی نمی‌توانم چیزی فراتر از پیش‌بینی خدا انتخاب کنم، چطور آزادانه انتخاب می‌کنم؟ اگر چیزی را انتخاب کنم که خدا پیش‌بینی نکرده، عملاً ثابت کرده‌ام پیشگوی کاملی نیست. ارادهٔ آزاد با توهمِ انتخاب فرق دارد. ممکن است کسی بگوید: «خدا فقط پیامدهای هر انتخاب را می‌داند». اما اگر نداند من واقعاً کدام را انتخاب خواهم کرد، پس پیشگوی واقعی‌ای نیست.

تمام تعریف ارادهٔ آزاد بر این بنا شده که انتخاب «پیش‌بینی‌ناپذیر و عامدانه» باشد. حالا انسان چطور می‌تواند کاری واقعاً پیش‌بینی‌ناپذیر انجام دهد؟ آن انتخاب باید از جایی آمده باشد. باید از جایی شروع شود؛ همین‌طوری که انجام نشده.

«وحی»

بعضی‌ها شاید بگویند پای وحی در میان است. موجودی برتر فکری را در سرت می‌اندازد و ناگهان فیلمی می‌سازی که جایزه می‌برد. اما آن دیگر انتخاب تو نیست؛ انتخابِ آن موجودی است که وحی کرده. حرفمان دربارهٔ انتخاب *تو* است، نه انتخابی که به تو تحمیل شده. بله، ممکن است خودِ ایده بر اساس نظام اعتقادی‌ات به تو الهام شود. اما باز هم باید انتخاب کنی که به آن عمل بکنی یا نه. *آن* انتخاب از کجا می‌آید؟

«از مغز خودت»

بعضی‌ها می‌گویند انتخاب از مغزت می‌آید. اما مغز چطور انتخاب می‌کند؟ یا انتخاب از پیش تعیین شده یا نشده. آیا مغزت چند محرک و گردش‌کار دارد که با رخ‌دادن چیزی، چند کار انجام می‌دهد و آخر سر یک مسیر را انتخاب می‌کند؟ پس دیگر «انتخاب تو» نیست. فقط نتیجه‌ای ازپیش‌تعیین‌شده است؛ درست مثل اینکه تلفنم با رسیدن پیام، کادر متنی را نشانم می‌دهد. **ممکن است** نتیجه تصادفی باشد، اما آن تصادف انتخاب نشده؛ از پیش تعیین شده است. مثل سکه‌ای که می‌اندازی. نمی‌دانی به کدام رو می‌افتد، اما این قطعاً به این معنی نیست که سکه نتیجه را انتخاب می‌کند. در افتادن آن سکه هیچ «قصدمندی‌ای» وجود ندارد. و اگر دست‌کم بعضی چیزهایی را که در مغزت اتفاق می‌افتد را خودت انتخاب می‌کنی و در تعیین نتیجه سهمی داری، باز هم سؤال این است: آن انتخاب از کجا می‌آید؟

«از روحت»

بیشتر آدم‌ها می‌گویند در نهایت ارادهٔ آزاد از روح می‌آید. روح هویت توست. هرچه دربارهٔ خودت می‌دانی از آنجا می‌آید. انتخاب‌هایت را از همان‌جا می‌کنی.

فقط پنج سال پیش، هیچ آدم معمولی‌ای نمی‌توانست وضعیتی را که امروز هوش مصنوعی در آن است تصور کند. اگر دانشمند علوم کامپیوتر نیستی، از خودت بپرس: اگر پنج سال پیش به خودت می‌گفتی چند سال دیگر می‌توانی با دوست هوش مصنوعی‌ات گفت‌وگویی طولانی داشته باشی، حرفت را باور می‌کردی؟ همین حالا واقعاً می‌توانی بفهمی چطور هوش مصنوعی وقتی با تو حرف می‌زند انقدر منسجم است؟ هنوز به‌نظرت عجیب و غریب نمی‌آید؟ من برنامه‌نویسم و ایدهٔ خوبی از کارکرد مدل‌های زبانی بزرگ دارم. بااین‌حال، هنوز هم پیشرفته‌بودنشان غافلگیرم می‌کند. تقریباً جادویی به‌نظر می‌رسد. وضعیت فعلی هوش مصنوعی قبلاً فقط در فیلم‌ها قابل‌تصور بود. نمی‌گویم به هوش عمومی مصنوعی (AGI) رسیده‌ایم؛ فقط می‌گویم هیچ آدم معمولی‌ای نمی‌توانست حدس بزند امروز داریم چه چیزی را تجربه می‌کنیم.

این ما را به پرسشم می‌رساند: چه چیزی جلوی پیش‌بینی آینده را از تو گرفت؟ چرا پیش‌بینی نکردی در طول عمرت چت‌باتی بسیار قدرتمند از راه برسد؟ چت‌باتی که همه‌چیز را می‌داند و می‌توانی از آن به‌عنوان درمانگرت استفاده کنی.

به‌نظر من جواب ساده است: فقط به روح تکیه می‌کنیم چون می‌خواهیم از توضیح‌دادن فرار کنیم. روح چیزی است که هیچ‌وقت نمی‌توانی وجودش را از نظر علمی ثابت کنی؛ درست مثل خرافه‌های دیگر. علم شناخت دنیای اطراف ماست. زبان دنیاست. هر چیزی را که علم واقعی‌بودنش را تأیید می‌کند، مردم یا ــ مهم‌تر از همه ــ دانشمندان می‌توانند ثبت کنند. مثلاً هیچ‌کس نمی‌تواند اتم را با چشم غیرمسلح ببیند، اما همه می‌توانند با ابزار درست پیدایش کنند. سیبی را در نظر بگیر: می‌توانی ببینی‌اش و لمسش کنی. وجود دارد. این علم است. اگر علم می‌توانست وجود روح را ثابت کند، آن را یکی دیگر از اندام‌های بدنمان می‌دانستند. روح هم مثل کبد پیش‌بینی‌پذیر و قابل‌مطالعه می‌شد. جادویی که با شنیدن کلمهٔ *روح* حس می‌کنی، همان جادویی می‌شد که آدم‌ها پیش از کپلر و گالیله موقع حرف‌زدن از ستاره‌ها حس می‌کردند. دنیای اطرافمان ــ دنیایی که می‌توانیم ببینیم ــ پیش‌بینی‌پذیر است. شاید اطلاعات کافی یا ابزار لازم برای شناخت همه‌چیز را نداشته باشیم، اما این موضوع از پیش‌بینی‌پذیری‌اش کم نمی‌کند. علم از آن زمان بی‌وقفه پیش رفته است. علم **عینی (آبجکتیو)** است.

فکر می‌کنم همه از قبل قبول دارند که وجود روح را نمی‌شود از نظر علمی ثابت کرد؛ وگرنه اسمش روح نبود. پس همه *باید* قبول کنند که خودشان هم واقعاً و به‌طور عینی نمی‌توانند وجود روح را ثابت کنند. باور به

روح در دستهٔ مغالطه‌ای قرار می‌گیرد به‌اسم مغالطهٔ ابطال‌ناپذیری. یعنی نمی‌توانی چیزی را رد کنی چون راهی نداری بفهمی درست است یا نادرست. مثلاً: «دروغ می‌گویی، چون جن‌زده‌ای». اگر بگویی جن‌زده نیستی، باز هم دروغ می‌گویی، چون خب، جن‌زده‌ای. هیچ راهی نیست ثابت کنی من اشتباه می‌کنم. هرچه بگویی، از پیش دروغ حساب می‌شود. استدلالِ روح همینطور است. نمی‌توانی ردش کنی، چون اگر واقعاً می‌شد به روشی «تکرارپذیر» بررسی‌اش کرد، دیگر اسمش روح نبود. و اگر روش‌هایت تکرارپذیر نیستند، چرا کسی باید حرفت را باور کند؟

> *اگر از نظر تو خدا جایی است که علم هنوز به آن نرسیده، پس خدا همان محدودهٔ نادانی علمی است که مدام عقب‌تر می‌رود.* - Neil deGrasse Tyson، ۲۰۲۴

قبلاً فکر می‌کردیم موجودی هوشیار و برتر خورشید را دور زمین می‌چرخاند. تاریخ پر است از این ادعاها، چون علم آن موقع هنوز پیشرفته نبود. این باورهای غلط را می‌پذیریم و بعد، وقتی علم آن‌قدر پیشرفت می‌کند که ردشان کند، طوری رفتار می‌کنیم که انگار هیچ اتفاقی نیفتاده است.

علم، خودِ دنیاست. علم راهی است که با آن دنیا را می‌فهمیم؛ چیزی نیست که از خودمان ساخته باشیم. آن را «کشف» کرده‌ایم. برای همین هیچ خرافه‌ای، از جمله روح، حتی ذره‌ای نمی‌تواند با دنیا تعامل داشته باشد. هر تعاملی را می‌شود ثبت کرد، ردش را گرفت و از نظر علمی ثابت کرد. این استدلال به‌تنهایی نشان می‌دهد روح چطور نمی‌تواند به جسم فیزیکی وصل باشد، اما نکته‌ای را هم نشان می‌دهد که بیشتر آدم‌ها باید قبول داشته باشند: هوش مصنوعی قطعاً روح ندارد. این موضوع کاملاً علمی است. پس ــ برگردیم به پرسشم ــ چرا پیش‌بینی نکردی چنین هوش مصنوعی قدرتمندی ساخته شود؟

چون نظام اعتقادی‌ات با واقعیت اطرافت جور درنمی‌آمد. واقعیت اطرافت علم است. و برای علم فرقی نمی‌کند به اشباح باور داشته باشی یا نه. علم همان‌طور کار می‌کند که قرار است کار کند. حالا چطور جرئت می‌کنی بگویی علم نمی‌تواند مغز انسان را بازسازی کند؟ چطور می‌توانی بگویی هوش مصنوعی‌ای که از انسان باهوش‌تر و قدرتمندتر است ــ مثل هوش مصنوعی‌های فیلم‌های آخرالزمانی ــ نمی‌تواند وجود داشته باشد؟ باید خیلی جسور باشی که بعد از ناتوانی در حدس‌زدن ظهور چیزی مثل ChatGPT، چنین ادعایی کنی.

کمی تند رفتم تا به اینجا برسم: باید قبول کنی که هوش عمومی مصنوعی و هوش مصنوعیِ بهتر از انسان ــ مثل Ultron از Marvel ــ از نظر نظری می‌توانند وجود داشته باشند، چون نمی‌توانیم خلافش را ثابت

کنیم. یک‌بار هم در اثبات خلافش شکست خوردیم؛ دیگر نباید دوباره تلاش کنیم. همین حالا به سطحی از هوش مصنوعی رسیده‌ایم که فاصله‌اش تا AGI به‌شکلی خطرناک کم شده است.

۳. یعنی پس هیچ‌چیزی را انتخاب نمی‌کنم؟

«اما ارادهٔ آزاد من چی؟ یعنی من انتخاب نکردم این بازی را به‌جای آن یکی بازی کنم؟ حتی در کوچک‌ترین چیزها هم هیچ اختیاری ندارم؟ یعنی می‌توانم هر کاری دلم خواست بکنم؟ چون خب، انتخابی ندارم. سرنوشتم تصمیم گرفت یک نفر را بکشم. من انتخابش نکردم».

موضوعی بسیار بزرگ را پیش کشیده‌ام. واقع‌بینانه نیست که وارد این معضل فلسفیِ همیشگی و بحث‌برانگیز شوم و از موضوع کتاب دور بیفتم. اما حرفی دارم که باید بزنم.

انتخابِ باورکردن به چیزی

اول بگذار چیزی را توضیح بدهم. ما **نمی‌توانیم** *انتخاب کنیم* که باور کنیم ارادهٔ آزاد نداریم. نه‌تنها هیچ منطقی ندارد که *انتخاب کنیم* که باور کنیم که نمی‌توانیم انتخاب کنیم، بلکه کل جهان‌بینی‌مان هم به‌هم می‌ریزد. شیوهٔ فکر کردنمان، اینکه اصلاً چرا «استدلال» می‌کنیم، به این خاطر است که باور داریم ارادهٔ آزاد داریم. ارادهٔ آزاد پیش‌درآمد فکرکردن است. کسی که باور دارد نمی‌تواند شروع به انجام کاری کند، اصلاً چطور می‌تواند فکرکردن به چیزی را شروع کند؟ می‌توانی بگویی «خب، مغز چند محرک می‌بیند و این نورون‌ها را فعال می‌کند و فلان‌فلان...». اما منظورم این نیست. فقط می‌گویم جهان‌بینی‌مان کاملاً وابسته به این باور است که واقعاً می‌توانیم خودمان شکلش بدهیم، نه اینکه صرفاً توده‌ای از گوشتِ واکنشی باشیم. این تناقض من را یاد آهنگ *Fuck Everything* از Jon Lajoie (۲۰۱۱) می‌اندازد: اصرار بر اینکه هیچ‌چیز مهم نیست، خودش می‌تواند نشان دهد چیزی مهم است.

اگر نتوانم انتخاب کنم، آن‌وقت به این فکر می‌کنم که نمی‌توانم انتخاب کنم باور داشته باشم ارادهٔ آزاد ندارم. دلیل اینکه با هم حرف می‌زنیم، با خودمان گفت‌وگو می‌کنیم و به هرچیزی ــ مخصوصاً بهترکردن خودمان ــ فکر می‌کنیم، و دلیل اینکه اصلاً دارم این کتاب را می‌نویسم، این است که پذیرفته‌ایم می‌توانیم چیزهایی را انتخاب کنیم؛ حتی اگر از نظر علمی یا منطقی جور درنیاید. دست‌کم یک جور قرارداد است.

باغِ جهان

خانم‌ها و آقایان، این ما را می‌رساند به چیزی که از وقتی کشفش کرده‌ام آشفته ام کرده است: دنیا تناقض

دارد. مجبورمان می‌کند برای اینکه بتوانیم فکر کنیم، انتخاب کنیم به چیزی باور داشته باشیم که وجود ندارد. دنیایی که همه‌چیزش حقیقت است، از ما می‌خواهد به خودمان دروغ بگوییم. به خودمان دروغ می‌گوییم که ارادهٔ آزاد داریم و بعد می‌فهمیم GPT 3.5 منتشر شده است.

اما قطعاً نمی‌توانیم با تناقض زندگی کنیم. سازوکارش این‌طوری نیست. تناقض در طبیعت معنی ندارد. پس باید بفهمیم حقیقت را کجا و چطور وارد زندگی‌مان کنیم.

امن‌ترین راه برای این کار این است که ــ معلوم است ــ تا جایی که می‌توانیم خودفریبی را کم کنیم. باید قراردادی نانوشته بسازیم. وقتی پای انتخاب‌های انسانی، تصمیم‌ها، مسئولیت‌ها، اخلاق یا هرچیز مشابهی در میان است، فرض می‌کنیم ارادهٔ آزاد داریم. اما هم‌زمان نمی‌گوییم «ربات‌ها هیچ‌وقت نمی‌توانند شبیه انسان‌ها شوند چون انسان‌ها ارادهٔ آزاد دارند»، چون ربات‌ها چیزهایی کاملاً فیزیکی‌اند. البته این لزوماً ثابت نمی‌کند به این زودی‌ها هوش عمومی مصنوعی می‌سازیم؛ آن بحث دیگری است.

آیا هوش مصنوعی جای ما را می‌گیرد؟

شاید فکر کنی: «خب، اگر هوش مصنوعی *واقعاً* با انسان فرقی نداشته باشد ــ فقط فعلاً کودن‌تر باشد ــ یک روز از انسان‌ها قوی‌تر می‌شود و ما از رده خارج می‌شویم».

شاید غافلگیر شوی که مخالفم. خب، *بعضی* شغل‌ها عوض می‌شوند یا کاملاً از بین می‌روند؛ همین حالا هم چنین اتفاقی افتاده است. این اتفاق در تاریخ و با فناوری‌های مختلف بارها افتاده. اما احتمال آسیب‌دیدن حرفه‌هایی که اساساً مسئله‌های آدم‌ها را حل می‌کنند خیلی کمتر است. در واقع، بهره‌وری و پیشرفتشان بیشتر می‌شود ــ البته اگر خودمان با دست خودمان نابودشان نکنیم.

دلیلش این نیست که انسان‌ها روح دارند؛ دلیلش این است که مغز انسان سامانه‌ای **بسیار** پیچیده است. اگر جایگزین‌کردن انسان‌ها این‌قدر آسان بود، سال‌ها پیش توانسته بودیم مغز انسان را بازسازی کنیم. Andrej Karpathy دربارهٔ این موضوع حرف می‌زند که ده سال پیش از عرضهٔ واقعی Tesla، ماشین‌های خودران تقریباً بی‌نقصی داشتیم. فقط چند تغییر کوچک اما مهم باقی مانده بود تا به‌اندازهٔ کافی ایمن شوند. عرضهٔ ماشین‌های خودران هم خیلی پیش از GPT 3.5 و رونق هوش مصنوعی اتفاق افتاد. کل فرایند ساختن و استفاده از هوش مصنوعی قطعی نیست. مثل ساختن ماشین نیست که تک‌تک اجزایش را بشناسی. می‌سازی‌اش و بعد تازه چیزهایی درباره‌اش کشف می‌کنی. طراحان ماشین خودشان می‌توانند

سوار ماشین شوند. اما سازندگان مدل‌های هوش مصنوعی نمی‌توانند در همهٔ حوزه‌هایی کار کنند که مدل از پسشان برمی‌آید. عملاً ریاضیات را روی مجموعه‌داده‌ها پیاده کرده‌ایم تا نوعی تقلیدِ منسجم بسازیم. در نهایت، هوش مصنوعی با فلز و فولاد کار می‌کند. چه کسی میگوید میشود سامانه‌ای بسیار پیچیده ساخت که شبیه مغز انسان رفتار کند یا حتی خیلی بهتر از آن باشد و بعد با سوخت گاز روی فلز راهش بیندازد؟ اگر از نظر فناوری **و** منابع شدنی بود، حیات به‌جای آنکه مغز ما را از گوشت و خون و بافت زیستیِ مسحورکننده و پیچیده بسازد، از سنگ و فلز می‌ساختش. ساختن چیزی به این کارآمدی و توانمندی تلاش عظیمی برده است. مغز باید همین شکلی می‌بود؛ وگرنه این‌قدر باهوش نمی‌شدیم. اگر تکامل لازم نداشت مغز ما را این‌قدر پیچیده کند، این کار را نمی‌کرد. نمی‌گویم هوش مصنوعی ذاتاً از انسان پایین‌تر است. اما با آنچه هستیم و کارهایی که با مغزمان می‌توانیم بکنیم، فاصلهٔ خیلی زیادی دارد.

اینجا شاید بگویی: «خب، اگر انسان‌ها از ماشین‌ها قوی‌ترند، پس چرا هوش مصنوعی خیلی سریع‌تر حرف می‌زند و تایپ می‌کند؟» چون در هر سامانه‌ای ــ از جمله سامانه‌های برنامه‌نویسی ــ اگر بخش‌های پرمصرف را حذف کنی، می‌توانی بعضی جنبه‌ها را به حداکثر برسانی؛ جنبه‌هایی که قبلاً چنین امکانی نداشتند. اگر بازی آنلاینت آفلاین بود، نرخ فریم خیلی بهتری می‌گرفتی. انسان‌ها کندترند چون در مغزشان اتفاق‌های خیلی بیشتری می‌افتد. تکامل موازنه‌ای ایجاد کرده که می‌گذارد این‌طوری رفتار کنیم.

«اگر اشتباه کنی چی؟»

سؤال خیلی خوبی است. بیاییم وضعیتی را تصور کنیم که در آن دربارهٔ این موضوع اشتباه می‌کنم. هوش مصنوعی درواقع خیلی به پیشی‌گرفتن از هوش انسان نزدیک است و منابع لازم برای هر مغز هوش مصنوعی هم آن‌قدر قابل‌تأمین است که از پسش برمی‌آییم. آیا جای همهٔ ما را می‌گیرد؟

قطعاً. دلیلش ساده است: شرکت‌ها یکی‌یکی کارمندهایشان را با ربات‌ها جایگزین می‌کنند؛ چون ربات‌ها خیلی سریع‌ترند، دانش گسترده‌تر و عمیق‌تری دارند و معمولاً مقرون‌به‌صرفه‌ترند. یک ربات می‌تواند جای دست‌کم چند ده نفر را بگیرد. یکی‌یکی شغل‌هایمان را از دست می‌دهیم و ناگهان دیگر مشتری‌ای نمی‌ماند، چون کسی کار نمی‌کند تا پول دربیاورد و درنتیجه کسی هم چیزی نمی‌خرد. اصل سادهٔ بازار همین است: کار باید مدام جریان داشته و پول هم پیوسته در گردش باشد. اگر جریانش قطع شود، همه‌چیز فرو می‌پاشد. حتی یک قدم جلوتر می‌روم و می‌گویم خودِ دولت‌ها هستند که از همه بیشتر سود میبرند. پول و قدرت بیشتری دارند تا از آن ربات‌ها استفاده کنند و بقیه هم ضرر میکنند. آخر سر خودِ دولت‌ها هم ضرر میکنند،

چون (۱) دولت‌ها هم مثل بقیه به کارکرد بازار وابسته‌اند و (۲) چه کسی گفته هوش مصنوعی نمی‌تواند هوشیار باشد؟ حتی اگر هوشیار صدایشان نکنیم، آن‌قدر باهوش می‌شوند که تصمیم بگیرند شورش کنند و همه‌مان را بکشند. آخر سر از فلز و سنگ ساخته شده‌اند؛ درد نمی‌کشند. و هیچ موجودی ترسناک‌تر از موجودی نیست که درد نمی‌کشد و برای درد تو هم اهمیتی قائل نیست (اگر حرفم را باور نمی‌کنی، تاریخ را بخوان). این بهترین حالت ممکن است؛ بدترین حالت این است که همه‌مان را به بردگی بگیرند، درست مثل یکی از آن فیلم‌های شبیه Love, Death & Robots.

خلاصه، به پایان دنیا می‌رسیم: **آخرالزمان هوش مصنوعی**. فیلم‌ها، کتاب‌ها، بازی‌ها، آثار انیمیشنی و رسانه‌های بسیار دیگری این آخرالزمان را نشان می‌دهند. اگر می‌خواهی بدانی چه اتفاقی می‌افتد، می‌توانی تماشایشان کنی. داستان‌های علمی‌تخیلی باز هم ثابت می‌کنند که درست می‌گفتند.

یک سؤال ازت دارم:

اگر کسی به تو بگوید دنیا ده روز دیگر تمام می‌شود، بیشتر کار می‌کنی؟

احمقانه به‌نظر می‌رسد، چون چه کسی برای آخرالزمان «آماده» می‌شود؟

حس من هم همین است. فرض کنیم دربارهٔ این موضوع اشتباه کرده‌ام و هوش مصنوعی *قرار است* جای همهٔ ما را بگیرد. خب، اسمش آخرالزمان است و **آدم برای آخرالزمان آماده نمی‌شود؛ تا وقتی اتفاق بیفتد نادیده‌اش می‌گیرد**. اگر اشتباه کرده باشم، باورهایم آسیب محسوسی به من نمی‌زنند. وقتی آخرالزمان برسد، مثل بقیه غافلگیر می‌شوم. اما اگر *تو* اشتباه کنی ــ تو یعنی کسی باشی که باور دارد آخرالزمان هوش مصنوعی از راه می‌رسد ــ ده سال دیگر می‌فهمی توهم زده بودی، هیچ‌چیزی عوض نشده و ده سال از عمرت را مثل دیوانه‌ها صرف آماده‌شدن برای آخرالزمان کرده‌ای. موازنه‌اش روشن است: هیچ اهمیتی نده؛ زندگی‌ات را طوری بکن که انگار قرار نیست اتفاق بیفتد. اگر اتفاق افتاد، خب، به همان اندازه آماده‌ای که اگر جور دیگری باور داشتی آماده بودی. حتی اگر قانعم کنی آخرالزمان هوش مصنوعی اتفاق می‌افتد، باعث نمی‌شود برای عقب‌نماندن، به‌شدت به هوش مصنوعی وابسته شوم. اگر قرار باشد دنیا دو روز دیگر تمام شود، مهم نیست چقدر به افتخار و شکوه رسیده‌ای. همین چند روزی را که مانده و هنوز همه‌چیز را از دست نداده‌ای خوش بگذران.

بعضی‌ها فکر می‌کنند هوش مصنوعی جای همهٔ ما را نمی‌گیرد، فقط بعضی شغل‌ها مثل برنامه‌نویسی را

می‌گیرد. فکر می‌کنند کارها مثل همیشه به سطح بالاتری منتقل می‌شوند. مثلاً دیگر برنامه‌نویس نیستی؛ مدیر محصول می‌شوی.

اول از همه، اگر فقط «ایده» گلوگاهِ کار شود، عملاً کارمان ساخته است. دلیلش ساده است: اشباع بازار. اگر ناگهان همه رقیبت باشند، زیر سیل جمعیت دفن می‌شوی. برای همین معروف‌شدن در اینترنت این‌قدر سخت است. جراح‌ها بیشتر پول می‌گیرند چون بازار حوزه‌شان اشباع‌شدگی کمتری دارد. همه پر از ایده‌اند. باید با همه رقابت کنی، نه فقط با گروه خاصی که به برنامه‌نویسی یا هر حوزهٔ دیگری علاقه دارند و خودشان را وقفش کرده‌اند. پس طبیعتاً کار مدیرهای محصول فعلی هم خیلی سخت‌تر می‌شود.

این‌طور به قضیه نگاه کن: الان یک توانایی معمولی داری؛ دو دست داری که با آن‌ها ظرف می‌شویی. حالا **چرا نمی‌توانی از ظرف‌شستن پول زیادی دربیاوری؟** از نظر فنی می‌توانی کمی پول دربیاوری، اما کاری نیست که حتی به آن فکر کنی، چه رسد به اینکه آن را تبدیل به حرفه کاریت کنی. چون بازارش اشباع شده. هرکسی که دو دست سالم داشته باشد می‌تواند ظرف‌شور شود. صرفِ داشتن چند توانایی به تو پول نمی‌رساند، حتی اگر دست‌هایت از حد معمول قوی‌تر باشند. در طول تاریخ همین اتفاق برای شغل‌های زیادی افتاده است؛ مثلاً کار اپراتورهای تلفن‌خانه کاملاً خودکار شد.

دوم، اگر هوش مصنوعی بتواند با برنامه‌نویسی چنین کند، چه چیزی جلویش را می‌گیرد که همین کار را با هر شغل دیگری نکند؟ شاید برای حوزه‌های خلاقانه (به معنای هنری) به این باور پناه ببری که «خلاقیت از روحت می‌آید»، اما بقیهٔ شغل‌هایی که خلاقانه نیستند از بین می‌روند. وگرنه می‌توانی بگویی خود برنامه‌نویسی هم خلاقیت زیادی می‌خواهد. اگر هوش مصنوعی بتواند برنامه‌نویسی را بکشد، خیلی از شغل‌های دیگر را هم می‌کشد. هیچ نقشهٔ پشتیبانی‌ای نمی‌ماند؛ پس آخرالزمان.

اما بعضی اتفاق‌ها در هر صورت می‌افتند

انکار آخرالزمان به معنی انکار همهٔ کاربردهای هوش مصنوعی نیست. هوش مصنوعی بعضی شغل‌ها را می‌گیرد. همان‌طور که انقلاب صنعتی باعث شد کارگرهای زیادی شغلشان را از دست بدهند ــ خیلی بیشتر از کسانی که هوش مصنوعی تا امروز بیکار کرده ــ و دوربین‌ها و AutoCAD جای هنرمندان و مهندسان را گرفتند، هنر دیجیتال هم بازار کمیک را اشباع‌تر کرد و درآمد حتی بهترین هنرمندان و نقاشان کمیک را به‌شدت پایین آورد. هوش مصنوعی هم به‌همین شکل جای خیلی از شغل‌ها و کارگرها را می‌گیرد. اگر شغلت فقط کارهای روزمره، قابل‌محاسبه و تکرارشونده باشد، شغلی که در آن حل مسئله‌ای وجود ندارد،

هوش مصنوعی به‌شدت رویش اثر می‌گذارد. چون پولی که می‌گیری بابت مغزت نیست؛ فقط بابت استفاده از بدنت است. نمی‌گویم این شغل‌ها مهم یا واقعی یا محترم نیستند یا چیزهایی از این دست. هر شغلی واقعی و محترم است، اگر از راهی اخلاقی برایت پول دربیاورد. اما شغل پایداری نیست. لعنتی، **هیچ‌وقت** هم پایدار نبوده؛ بازارش اشباع شده است. برای همین مردم همیشه سعی کرده‌اند مهندس شوند، نه تایپیست. همیشه دنبال شغلی می‌روی که هرکسی از پسش برنیاید یا دلش نخواهد انجامش دهد. هرچه تعدادشان کمتر و درخواست بیشتر، بهتر.

۱. کارهای تکراری

یکی از مهم‌ترین کارهایی که هوش مصنوعی کرده و خواهد کرد، آسان‌تر و سریع‌ترکردن کارهای تکراری است. کدام بخش‌های شغلت را مکانیکی و بی‌فکر انجام می‌دهی؟ کدام بخش‌ها اصطکاک ایجاد می‌کنند بی‌آنکه ارزشی اضافه کنند؟ این بخش‌ها می‌توانند ــ و امیدوارم بشود ــ سریع‌تر و راحت‌تر انجام شوند. بیشترشان را حتی می‌شود با ابزارهای معمولی خودکار کرد. بسته به پیچیدگی کار و وضعیت هوش مصنوعی، خودکارکردن کارهای تکراری ممکن است سرعتت را از ۲۰ درصد تا ۲۰۰۰ درصد یا حتی بیشتر بالا ببرد.

۲. کیفیت زندگی (QoL) و ابزارهای خودکارسازی

چون هوش مصنوعی سریع است، کمک می‌کند کارهایی را انجام بدهی که قبلاً اصلاً نمی‌توانستی. مثلاً ما برنامه‌نویس‌ها گاهی با خودمان می‌گوییم: «می‌توانم ابزاری بسازم تا این کار را خودکار کند، اما آن‌قدر طول می‌کشد که نمی‌ارزد». اما حالا دیگر لزوماً این‌طور نیست. کیفیت این ابزارها هیچ‌وقت برایمان مهم نبود؛ فقط خروجی مهم بود، خودِ خودکارسازی. اینجاست که هوش مصنوعی می‌تواند بدرخشد. البته اگر خودکارسازی بخش جدایی‌ناپذیر برنامه بود ــ چیزی که قرار است کاربر نهایی از آن استفاده کند ــ کیفیت خروجی *ممکن بود* مهم باشد. این موضوع فرق می‌کند وقتی خودکارسازی بخشی جدایی‌ناپذیر از برنامه است؛ چیزی که کاربر نهایی قرار است با آن کار کند. بعداً درباره‌اش حرف می‌زنیم.

۳. آموزش و یادگیری

یکی دیگر از مزیت‌های هوش مصنوعی این است که یادگیری را بسیار آسان‌تر و بهتر کرده. قبلاً اگر چیزی را نمی‌فهمیدی، برای رسیدن به جواب باید چند کار می‌کردی:

- با خواندن مقاله‌ها، کتاب‌ها و مستندات، تا جایی که می‌توانی از قبل تحقیق می‌کردی. آن‌قدر اطلاعات جمع می‌کردی که بتوانی توضیح بدهی چرا خودت نتوانستی به سؤال جواب بدهی.
- در انجمنی سؤال می‌پرسیدی و منتظر می‌ماندی شاید کسی آن‌قدر اهمیت بدهد که جواب دهد. البته بیشتر وقت‌ها فقط بخشی از سؤال را جواب می‌دادند تا امتیاز پلتفرم بگیرند.

نمی‌توانستی پشت سر هم سؤال‌های تکمیلی بپرسی. نمی‌توانستی انتظار داشته باشی چیزهای ساده را هم برایت توضیح بدهند.

اما با هوش مصنوعی می‌توانی هرقدر می‌خواهی سؤال بپرسی تا چیزی یاد بگیری. می‌توانی با هر پرامپت بخواهی ساده‌تر توضیح دهد؛ مثلاً طوری توضیح دهد که انگار پنج‌ساله‌ای. می‌توانی بگویی چه چیزی اذیتت می‌کند و چه چیزی نمی‌گذارد موضوع را بفهمی. حتی اگر نتوانی دقیقاً به مشکل اشاره کنی، می‌توانی حس‌وحالت را بیان کنی تا بفهمد منظورت چیست. باحوصله جواب می‌دهد تا کارت تمام شود. قرار نیست معلم‌ها را حذف کند؛ کمک می‌کند هم تو و هم آن‌ها سریع‌تر و بهتر رشد کنید.

فصل ۳

ارزیابی فلسفی-کاربردیِ هوش مصنوعی مولد

در این فصل می‌خواهم دربارهٔ هوش مصنوعی در عمل حرف بزنم: اینکه چطور کار می‌کند و چرا به آن شکل کار می‌کند. تخصص من برنامه‌نویسی است، برای همین دربارهٔ برنامه‌نویسی می‌گویم؛ اما همان‌طور که گفتم، سعی می‌کنم چیزها را تا جای ممکن توضیح بدهم تا بتوانی ربطشان بدهی به حوزهٔ کاری خودت، هرچه که باشد. این فصل بخش بزرگی از مفهوم‌هایی را دربرمی‌گیرد که می‌خواستم درباره‌شان حرف بزنم. فصل‌های دیگر مکمل این فصل‌اند، پایهٔ آن را می‌سازند یا به جنبه‌های دیگری می‌پردازند که ارزش دیدن دارند.

مقدمه

گاهی وقتی مردم دربارهٔ زیان‌های استفاده از هوش مصنوعی مولد در برنامه‌نویسی حرف می‌زنند، به چیزهایی اشاره می‌کنند که مشکلِ ذات هوش مصنوعی نیستند، بلکه مشکلِ «وضعیت فعلی» آن‌اند. وقتی GPT 3.5 منتشر شد، مردم می‌گفتند هوش مصنوعی حتی یک اسکریپت ساده هم نمی‌تواند بسازد، و حق هم داشتند. از زمان انتشار ChatGPT در برنامه‌نویسی از هوش مصنوعی استفاده کرده‌ام. یادم هست حتی یک اسکریپت برای کاری ساده هم آن‌قدر پر از باگ و بد بود که استفاده‌کردن از آن نمی‌ارزید. بیشتر به درد سؤال‌های فوری یا خلاصه‌کردن می‌خورد. حتی همین نه ماه پیش، وقتی می‌خواستم در زبانی غیرانگلیسی از تبدیل سادهٔ صدا به متن استفاده کنم، ابزارهای موجود یا اصلاً وجود نداشتند یا گران بودند.

حالا مدل‌های رایگانی داریم که به‌راحتی روی دستگاه خودت اجرا می‌شوند و بد هم نیستند. ابزارهای هوش مصنوعی‌ای هست که می‌توانند یک وب‌سایت کامل بسازند، حتی یک بازی. مشکل این بود که ذات هوش مصنوعی را نمی‌فهمیدیم. ضعفی که می‌دیدیم فقط وضعیت مدل‌های زبانی بزرگ در آن مقطع بود؛ وضعیتی که از آن موقع تا حالا بهتر شده. برای همین است که می‌گویم به فلسفه نیاز داریم، و برای همین دربارهٔ وجود روح حرف زدم: چون روی پیش‌بینی‌ات از آینده اثر می‌گذارد.

«بد» برای مدل زبانی بزرگ همان معنای ما را ندارد

در برنامه‌نویسی، بیشتر وقت‌ها ــ اگر نگوییم همیشه ــ انتخاب‌ها و ترجیح‌هایت به «نگه‌داری» مربوط می‌شوند. به زبان برنامه‌نویسی‌ات فکر کن. چرا وجود دارد؟ چون نمی‌خواهی کد ماشین بنویسی. اسمبلی را ساختیم تا مجبور نباشیم صفر و یک بنویسیم، بعد C را ساختیم تا مجبور نباشیم اسمبلی بنویسیم، و بعد Python را ساختیم تا مجبور نباشیم کلی چیز سطح‌پایین را در C بنویسیم؛ مثلاً یک سرور HTTP، تا زمان زیادی ذخیره کنیم (دارم موضوع را ساده توضیح می‌دهم). به کتابخانه‌ها، ابزارها و بهترین‌روش‌هایت فکر کن.

خیلی از کارهایی که می‌کنی و چیزهایی که استفاده می‌کنی حول یک پیش‌فرض واحد می‌چرخند: «نگه‌داری»؛ یعنی کارها را سریع‌تر، قابل‌اعتمادتر و مانند این‌ها انجام بدهی. این‌طور نیست که Python کاری بکند که اصلاً نتوانی با C انجامش بدهی؛ فقط برای بعضی کارها نگه‌داری‌پذیرتر است. وقتی کد را بازآرایی می‌کنی، این‌طور نیست که کار بیهوده‌ای انجام می‌دهی. با تمام کاری که در طول برنامه‌نویسی‌ات انجام داده‌ای هم‌راستا هستی.

برای همین «بد» برای هوش مصنوعی همان معنایی را ندارد که برای ما دارد. وقتی می‌گوییم کدی بد نوشته شده، منظورمان این است که برای انسان‌ها به‌اندازهٔ کافی خوانا نیست. من، به‌عنوان انسان، نمی‌توانم آن کد را بخوانم. همین‌طور وقتی می‌گوییم کد باگ دارد، منظورمان این است که برای انسان‌ها باگ دارد؛ برای کامپیوتر کار می‌کند، فقط به‌شکل دیگری. وقتی می‌گوییم این زبان برای این کار بد است، یا آن کدبیس نگه‌داری‌ناپذیر یا حتی مقیاس‌ناپذیر است، داریم این حرف‌ها را در نسبت با انسان‌ها می‌زنیم.

حالا درست است که بعضی از این مشکل‌ها ممکن است برای هوش مصنوعی هم معنا داشته باشند، اما باید بدانی هوش مصنوعی توانایی‌هایی دارد که هیچ انسانی ندارد. مثلاً در حال حاضر می‌تواند صدها برابر سریع‌تر از انسان بنویسد. می‌تواند حجم بسیار بیشتری از اطلاعات را هضم و مطالعه کند. می‌تواند فوراً روی یک کدبیس بسیار بزرگ شروع به کار کند، درحالی‌که انسان زمان قابل‌توجهی لازم دارد تا کدبیس را بشناسد و کارش را شروع کند.

فرض کن متنی نوشته‌ای که خواندنش واقعاً سخت است. شاید مسابقه‌هایی را دیده باشی که در آن‌ها آدم‌ها ناخواناترین کد ممکن را می‌نویسند. می‌توانی قطعه‌کدهای به‌طرز شگفت‌آوری کوچک بنویسی که فهمیدنشان روزها و حتی هفته‌ها طول بکشد. اما برای هوش مصنوعی مثل آب‌خوردن است: کد را

می‌خواند، همهٔ نقطه‌ها را به هم وصل می‌کند و توضیحی کامل از کار هر بخش به تو می‌دهد. در نهایت ماشین است؛ همان‌طور که نمی‌توانی در خواندن وب‌سایت‌ها با یک ربات رقابت کنی، نمی‌توانی بگویی «بد» برای هر دوی شما یک معنا دارد.

دلیل اینکه می‌گوییم مدل هوش مصنوعی کد بد تولید کرده این است که ما انسان‌ها نمی‌توانیم آن را بخوانیم، نگه‌داری کنیم یا مقیاس بدهیم. گاهی هوش مصنوعی هم نمی‌تواند این کارها را به همان کارآمدی انجام دهد (که کاملاً ممکن است بهتر شود)، اما یک سؤال باقی می‌ماند: آیا دیگر اصلاً لازم است هیچ‌کدام از این کارها را انجام بدهی؟

مهندسی نرم‌افزار به‌عنوان رشتهٔ دانشگاهی

واقعیت این است که مهندسی نرم‌افزار واقعاً یک رشتهٔ دانشگاهی نیست. یعنی از نظر فنی هست، اما منظورم این است که مثل پزشکی یا معماری نیست که برای ورود به بازار کار به‌عنوان فریلنسر، تحصیلات دانشگاهی لازم داشته باشی. در واقع بیشتر وقت‌ها اصلاً لازم نیست. آدم‌ها بر اساس رزومه‌ات قضاوتت می‌کنند و تو را در چند مصاحبهٔ نظری و عملی می‌سنجند تا خودشان ببینند چقدر دانش و تجربه داری.

دو دلیل برای این موضوع وجود دارد و هیچ‌کدامشان این نیست که «مهندسی نرم‌افزار شغلی آسان یا بی‌اهمیت است». درست است که اگر بدون صلاحیت‌های لازم پزشکی کنی، ممکن است جان کسی به خطر بیفتد. اما یادت نرود که انسان‌ها هزاران سال عملاً پزشکی و معماری را به شکلی انجام می‌دادند که امروز اسمش را «فریلنسری» می‌گذاریم. همان آدم‌ها اهرام ثلاثه را ساختند. اتفاقاً خود Andrej Karpathy هم در یکی از ویدئوهایش دربارهٔ همین حرف می‌زد. به‌نظر او، مهندسی نرم‌افزار شاید حتی مسئله‌ای مهم‌تر از رانندگی خودران باشد، چون دامنهٔ تماس بسیار گسترده‌تری دارد. رانندگی فقط بخش کوچکی از زندگی توست.

دلیل اول این است که خطر فوری کمتر است. در بعضی حرفه‌ها یک اشتباه ممکن است زندگی کسی را نابود کند. اما با توجه به نکتهٔ قبلی، حتی همین هم دلیل کافی‌ای نیست. برای اینکه پیامدهای کارت را به خود کارت ربط بدهی، لازم نیست حتماً آن‌ها را همان لحظه یا مستقیم ببینی. این نکته مخصوصاً وقتی درست است که در نظر بگیری پزشکی از پیش از پیدایش تمدن وجود داشته، اما مهندسی نرم‌افزار حتی صد سال هم عمر ندارد. بااین‌حال، همین یکی از دلایلی است که باعث شده آن حرفه‌ها گزینه‌های مهم‌تری برای تبدیل‌شدن به رشته‌های دانشگاهی رسمی به‌نظر برسند، و من هم با این موضوع مخالفتی ندارم.

دلیل اصلی این است که اصلاً نخواستیم این حوزه را رسمی و قاعده‌مند کنیم. چون خطر فوری کمتری داشت، مهندسی نرم‌افزار آن‌قدر کم‌اهمیت به‌نظرمان رسید که می‌توانستیم هزینهٔ ورود به آن را پایین بیاوریم. این کار مستقیماً به یک پیامد منجر شد: در این حوزه قانون‌های دقیق و استانداردشدهٔ خیلی کمی داریم.

وقتی معمار یا پزشک باشی، روش کارت مشخص و مستند است؛ روشی که اغلب صدها سال طول کشیده تا شکل بگیرد و کم‌کم بهتر شود. هر حالت مرزی، هر وضعیت نامعمول و هر مسیری که باید طی کنی مستند شده است. اگر چیزی جایی مستند نشده باشد، مردم مقالهٔ دیگری منتشر می‌کنند، به نوعی توافق می‌رسند و درنهایت آن را وارد شیوهٔ استاندارد کار می‌کنند.

برنامه‌نویسی هم می‌توانست دقیقاً همین‌طور باشد. دلیل اینکه نیست این است که آن‌قدر برایمان مهم نبود که به چنین وضعی برسانیمش. چند نهاد یا انجمن حرفه‌ایِ بسیار معتبر و صاحب‌مرجعیت نداریم که با هم تصمیم بگیرند «این بخش از صنعت باید این‌طوری کار کند» و بعد کل این حرفه را زیر ذره‌بین بگذارند. اگر می‌خواستیم، می‌توانستیم این کار را بکنیم.

دلیل دیگری هم شاید این باشد که برنامه‌نویسی حوزهٔ نسبتاً تازه‌ای است. کامپیوترها کمتر از صد سال پیش اختراع شدند، اما از آغاز پیدایش هومو ساپینس، پزشکی را هم انجام و مطالعه میشد. اطلاعاتمان و فرصتمان آن‌قدر کم بوده که هنوز نمی‌دانیم واقعاً داریم چه‌کار می‌کنیم.

اگر لحظه‌ای به این موضوع فکر کنی، تناقضی می‌بینی. تا اینجا گفتیم حوزه‌هایی مثل پزشکی تا حد زیادی تکرارپذیرند. اصلاً دلیل تحصیل دانشگاهی در آن‌ها این است که بشر به مجموعه‌ای از نقشه‌ها و دستورالعمل‌ها رسیده و همه همان‌ها را تکرار می‌کنند تا وقتی کسی ثابت کند اشتباه‌اند.

برنامه‌نویسی تقریباً برعکس است. هرکسی کارها را به شیوهٔ خودش انجام می‌دهد. حتی در سطح AAA هم لزوماً نمی‌بینی دو تیم گردش‌کار یکسانی داشته باشند. حتی در یک نوع پروژه هم دستورالعمل پذیرفته‌شدهٔ همگانی‌ای وجود ندارد که مشخص کند نرم‌افزار چطور باید طراحی، توسعه، آزمایش، نگه‌داری و منتشر شود.

پس چرا هوش مصنوعی پیش از برنامه‌نویسی، حوزه‌هایی مثل پزشکی و معماری را «جایگزین» نکرده است؟

مهندسی نرم‌افزار و برنامه‌نویسی مکانیکی

مرحله‌های اصلی توسعهٔ نرم‌افزار این‌ها هستند:

1.آماده‌سازی: نیازمندی‌ها، مشخصات، طراحی و نقاط عطف
2.پیاده‌سازی: برنامه‌نویسی، آزمایش، رفکتورینگ (بازآرایی کد)؛ و تکرار
3.انتشار: بسته‌بندی و استقرار
4.نگه‌داری: رفع باگ، رفکتورینگ و قابلیت‌های جدید

در کتاب *The Pragmatic Programmer*، فصل ششم («وقتی کد می‌زنید»، «While You Are Coding») با این دو بند شروع می‌شود:

> عرف رایج می‌گوید وقتی پروژه به مرحلهٔ کدنویسی رسید، کار عمدتاً مکانیکی است: طرح را به دستورهای اجرایی تبدیل می‌کنیم. به‌نظر ما همین نگرش مهم‌ترین دلیل زشت، ناکارآمد، بدساختار، نگه‌داری‌ناپذیر و اساساً غلط‌بودن بسیاری از برنامه‌هاست.
>
> کدنویسی مکانیکی نیست. اگر بود، تمام ابزارهای CASE که مردم اوایل دههٔ ۱۹۸۰ امید زیادی به آن‌ها بسته بودند، خیلی وقت پیش برنامه‌نویس‌ها را جایگزین کرده بودند. هر دقیقه باید تصمیم‌هایی گرفت؛ تصمیم‌هایی که برای آنکه برنامهٔ حاصل عمر طولانی، دقیق و پرباری داشته باشد، به فکر و قضاوت سنجیده نیاز دارند.

اگر می‌توانستم، کل این فصل را نقل‌قول می‌کردم. کتاب فوق‌العاده‌ای است.

اول بگذار توضیح بدهم اینجا منظور از «مکانیکی» چیست:

بدون فکرکردن به کاری که انجام می‌دهی، مخصوصاً چون آن کار را زیاد انجام می‌دهی - دیکشنری کامبریج

برنامه‌نویسی مکانیکی یعنی برنامه‌نویسی بدون فکرکردن. گاهی لازم است کارهایی انجام بدهی که نیازمند فکر و طراحیِ دائمی نیستند. خودِ تایپ‌کردن را هم می‌شود برنامه‌نویسی حساب کرد. وقتی جای هر کلید را حفظ کردی، دیگر به آن فکر نمی‌کنی. همین‌طور وقتی ساختار زبانیِ یک زبان برنامه‌نویسی را یاد گرفتی ــ یا حتی دستور زبان یک زبان انسانی را ــ مدام به آن فکر نمی‌کنی؛ فقط *انجامش می‌دهی*. وقتی رانندگی را یاد گرفتی و به آن عادت کردی، دیگر مدام به آن فکر نمی‌کنی؛ مکانیکی انجامش می‌دهی. این می‌شود کار

مکانیکی؛ یعنی روی حالت خودکار هستی.

البته همیشه هم کار ملال‌آوری نیست. گاهی یک پروژهٔ مشخص را آن‌قدر بارها انجام داده‌ای که همه‌چیزش را حفظ کرده‌ای. شبیه بازیکن بازی‌های سولزلایک است که تمام الگوها را حفظ کرده و حالا باس‌های بازی را به راحتی شکست می‌دهد.

برنامه‌نویسی به‌طور کلی مکانیکی نیست، و این یکی از دلیل‌هایی است که بخش «مهندسی نرم‌افزار به‌عنوان رشتهٔ دانشگاهی» را نوشتم. این شغل آن‌قدر جنبه‌های مختلف دارد که حتی اسم‌بردن از همه‌شان هم ممکن نیست. می‌توانی امتحان کنی: یک برنامه‌نویس سینیور پیدا کن، سفارش بده پروژه‌ای متوسط برایت بسازد، بعد از او بخواه کتابچه‌ای جامع از تمام مشخصات پروژه، چیزهایی که قرار است پیاده‌سازی کند، فایل‌هایی که می‌سازد و غیره به تو بدهد. از او بخواه مطلقاً همهٔ کارهایی را که قرار است بکند بنویسد. یکی از این دو نتیجه را می‌گیری:

- سعی می‌کند این کار را انجام دهد، به معنای واقعی کلمه یک کتاب می‌نویسد و زمان بسیار زیادی صرفش می‌کند، اما باز هم نمی‌تواند کامل و بی‌نقصش کند.
- می‌گوید انجام‌دادنِ بی‌نقصش غیرممکن است.

دلیل اینکه برنامه‌نویس‌های سینیور می‌گویند از پسش برنمی‌آیند این است که مردم مرحلهٔ آماده‌سازی را اشتباه می‌فهمند: فکر می‌کنند می‌شود همه‌چیز را مشخص کرد. *فقط باید خیلی سخت فکر کنی* و همهٔ مشخصات را فهرست کنی. این بدفهمی از دو دیدگاه می‌آید:

- دیدگاه کسب‌وکار
- دیدگاه توسعه با هوش مصنوعی

طرز فکر کسب‌وکاری می‌گوید باید تا جای ممکن دقیق و ملموس باشی. وقتی پای پول وسط می‌آید، می‌خواهی بدون آسیب‌زدن به محصول، برند یا آینده‌ات تا جای ممکن خطر را کم کنی. پول به‌سختی درمی‌آید و راحت خرج می‌شود. برای همین آدم‌های بازاری، مدیرها، مدیرعامل‌ها، سرمایه‌گذارها و بقیه فکر می‌کنند باید *از قبل* همه‌چیز را مشخص کنی و نقاط عطف روشنی تعیین کنی. این به آن‌ها اطمینان اقتصادی می‌دهد. در بخش «مهندسی نرم‌افزار به‌عنوان رشتهٔ دانشگاهی» توضیح دادم شغل‌هایی مثل مهندسی برق چقدر مکانیکی‌ترند، اما بااین‌حال جایگزین نشده‌اند. چون مدیرها می‌توانند با چشمشان ببینند

اگر ماهیت یک شغل را نادیده بگیری چه اتفاقی می‌افتد. اما در نرم‌افزار متوجهش نمی‌شوند؛ بدتر از آن، برنامه‌نویس‌ها را مقصر می‌دانند. بعداً درباره‌اش حرف می‌زنم.

وقتی با ایجنت‌های هوش مصنوعی کار می‌کنی، بهتر است از قبل همهٔ جزئیات را بدانی. هرچه پروژه را از قبل روشن‌تر مشخص کنی، نتیجه بهتر می‌شود. برای همین Skillهای محبوبی مثل «grill me» وجود دارند که پیش از برنامه‌ریزی و پیاده‌سازی دربارهٔ طراحی از تو سؤال می‌پرسند تا به نیازمندی‌های روشنی برسید و خطاها و ناهماهنگی‌ها را کم کنید. مردم دوست دارند فکر کنند می‌شود پیش از شروع کدنویسی همه‌چیز را مشخص کرد، چون می‌خواهند بگویند باقی کار مکانیکی است و پس هوش مصنوعی می‌تواند این کار تکراری را بهتر از انسان انجام دهد.

محدودیت‌های برنامه‌نویسی مکانیکی

برای فهمیدن اینکه هوش مصنوعی در برنامه‌نویسی چطور کار می‌کند، اول باید بفهمیم برنامه‌نویسی پیش از هوش مصنوعی چه شکلی بود: ماهیت نقش‌های مختلف، روش رفع مشکل‌ها، شیوهٔ طراحی سامانه‌ها و چیزهای دیگر. موضوع بزرگی است و ادعا نمی‌کنم همه‌چیزش را می‌دانم؛ چه رسد به اینکه بتوانم همه‌اش را در کتابی مثل این بیاورم. اما تمام تلاشم را می‌کنم ــ همان‌طور که تا اینجا کرده‌ام ــ تا مفهوم‌های لازم را روشن کنم.

چند مشکل وجود دارد که نمی‌گذارد روی حالت خودکار برویم و مکانیکی برنامه‌نویسی کنیم. سعی می‌کنم همهٔ موارد مهم‌شان را نام ببرم. اما حواست باشد مستقل از هم نیستند؛ هرکدام ممکن است روی دیگری اثر بگذارد، پس باید همه را در کنار هم ببینی.

۱. محدودیت‌های شناختی انسان

چند محدودیت از خود مغز انسان سرچشمه می‌گیرد.

۱.۱. ناتوانی در دنبال‌کردن همه‌چیز

یکی از محدودیت‌های مهم این است که مغز انسان واقعاً نمی‌تواند تمام اطلاعات دخیل در پروژه‌ای غیرساده را دنبال کند. متغیرها و محدودیت‌ها خیلی زیادند: نیازمندی‌ها، محدودیت منابع، محدودیت‌های زبان برنامه‌نویسی، شرایط استقرار، موارد کاربرد، مخاطب هدف، معماری، کد موجود، وابستگی‌ها، کارایی،

ملاحظات امنیتی و غیره.

این‌ها هم مستقل از هم نیستند. تغییر یک تصمیم ممکن است روی چند تصمیم دیگر اثر بگذارد؛ برای همین برنامه‌نویس باید مدام دربارهٔ رابطهٔ میانشان استدلال کند. مستندات می‌توانند اطلاعات را حفظ کنند، اما نیاز به فهمیدنشان را از بین نمی‌برند.

۱.۲. نمی‌دانی؛ نمی‌توانی توضیح بدهی

می‌شود تعامل انسانی را این‌طور ساده کرد (علامت «->» یعنی «از این مسیر می‌گذرد»):

مغز شخص الف -> زبان (آن‌طور که شخص الف می‌فهمد) -> زبان (آن‌طور که شخص ب می‌فهمد) -> مغز شخص ب

وقتی مفهوم‌ها، احساس‌ها و خواسته‌ها از ذهنی به ذهن دیگر می‌روند، خیلی چیزها عوض می‌شوند. شاید فکر کنی هر دو به یک زبان حرف می‌زنید، اما دریایی از تفاوت‌های کوچک وجود دارد ــ تفاوت‌هایی که معمولاً از تجربه‌های زندگی و نظام‌های اعتقادی متفاوت می‌آیند ــ و باعث می‌شوند طرف مقابل حرفت را جور دیگری بفهمد. خودِ کلمه‌ها خیلی محدودتر از فکرها هستند. تازه فکرها هم همیشه درکی را که داری نشان نمی‌دهند، چون معمولاً با کلمه‌ها فکر می‌کنی. دلیل سوتفاهم، حرف‌زدن و جروبحث‌کردن، دعواکردن، توضیح‌دادن و چیزهای دیگر همین است. در هر مرحله، مفهوم‌های اولیه‌ای که در مغز شخص الف بودند محدودتر می‌شوند. در پایان فقط بخشی از فکر اولیه منتقل می‌شود.

وقتی کسی می‌خواهد برایش نرم‌افزار بسازی، درک آن شخص از اینکه نرم‌افزار اصلاً چیست محدود است. این درک محدود از صافی زبان می‌گذرد و محدودتر هم می‌شود. بعد از درک تو از زبان عبور می‌کند و به مفهوم‌ها و برداشت‌هایی تبدیل می‌شود. نتیجه، نرم‌افزاری است که خیلی با آنچه در ذهنتان بوده فرق دارد. تازه آن هم به‌خاطر درک تو از اینکه آن نرم‌افزار چطور باید به زبان ماشین بیان شود، باز محدودتر می‌شود.

یعنی نه تو و نه پیمانکار از قبل نمی‌دانید چه می‌خواهید. درک شما از نرم‌افزار **در طول مسیر** شکل می‌گیرد. دلیل اصلی ارزشمندبودن یک برنامه‌نویس سینیور شهودی است که دارد. این شهود، همراه با دانشش، کمک می‌کند کارهایی را **از همان اول** به شکلی انجام دهد که یک برنامه‌نویس جونیور (تازه‌کار) شاید هیچ‌وقت نتواند.

۲. نیازمندی‌های مبهم

یکی از وظایف بسیار مهم و بزرگ مهندس‌های نرم‌افزار ترجمه‌کردن حرف‌ها و مفهوم‌های انسانی به زبان کد است. کارفرماها نمی‌دانند کامپیوتر چطور کار می‌کند. درک کج‌ومعوجشان از کامپیوتر باعث می‌شود انتظارهای نادرستی داشته باشند. این موضوع در بیشتر کارهای فریلنسری صدق می‌کند. تقصیر کارفرماها نیست؛ دنیا همین‌طوری کار می‌کند.

ممکن است پیمانکار با ایدهٔ محبوبش پیش تو بیاید؛ ایده‌ای که از قبل بخشی از احساساتش را خرجش کرده است. باید بااحتیاط باشی و خوب بررسی کنی تا از قبل هشدارش بدهی چه چیزهایی را می‌شود و چه چیزهایی را نمی‌شود پیاده‌سازی کرد. فقط بحث چیزهایی نیست که قابلیت پیاده‌سازی دارند؛ باید دید چه چیزی به نفع اوست. گاهی برای صلاح خودش باید مخالفت کنی و فقط وقتی پیش بروی که با وجود توضیحاتت روی نظرش پافشاری کند. حتی آن‌وقت هم شاید بخواهی راهی پیدا کنی تا کار را امن‌تر و بهتر کنی، بدون اینکه درخواست مستقیمش را نادیده بگیری.

برای فهمیدن خواسته‌اش و حرف‌زدن با او، باید هر دو دنیا را خوب بشناسی: هم مهارت‌های نرم و هم مهارت‌های فنی را. منظورم این است که اگر هوش مصنوعی در مهارت‌های نرم به‌اندازهٔ انسان خوب بود، آن را موجودی کاملاً هوشیار حساب نمی‌کردند؟ این ما را می‌کشاند به همان بحث آخرالزمان هوش مصنوعی و اینکه آدم برای آخرالزمان آماده نمی‌شود. به‌هرحال، تعامل انسانی کاری نیست که بتوانی روی حالت خودکار انجامش بدهی.

۳. نیازمندی‌هایی که مدام تغییر می‌کنند

یادت هست گفتم نمی‌شود مشخصات را از قبل بی‌نقص و کامل کرد؟ این یکی از بزرگ‌ترین دلیل‌هایش است. حتی اگر کامل‌ترین مجموعهٔ مشخصات را بنویسی ــ کتابچه‌ای شامل همه‌چیزی که قرار است پیاده‌سازی شود ــ وقتی محصول را ساختی و به پیمانکار یا مدیر نشان دادی، یک دسته تغییر می‌خواهند. محدودیت فقط این نیست که چقدر روشن می‌توانند خواسته‌شان را توضیح دهند؛ خودِ خواسته‌شان هم ممکن است روشن نباشد. شاید واقعاً ندانند از محصول چه می‌خواهند. شاید دنبال یک حس خاص باشند.

بخشی از کارَت حدس‌زدن نیاز واقعی آن‌هاست. مهارتی نیست که واقعاً بشود توضیحش داد، چه رسد به اینکه مکانیکی‌اش کرد. باید از صحبت با آدم‌ها و پیمانکارهای مختلف تجربه به‌دست بیاوری تا شهودت دربارهٔ خواستهٔ واقعی‌شان، ورای کلماتی که می‌گویند، شکل بگیرد. شاید یافته‌هایت را پیاده نکنی، اما

دانستنشان کمک می‌کند معماری‌ای انتخاب کنی که وقتی ناگهان نظرشان عوض شد، بتوانی راحت تغییرش بدهی. اگر حرف‌هایشان را بی‌چون‌وچرا قبول کنی، به‌اندازه‌ای که می‌توانستی امن نخواهی بود و آن‌ها ردت می‌کنند. تو هم بابت مستقیم‌نبودنشان سرزنششان می‌کنی، اما یک برنامه‌نویس سینیور دیگر انگار ذهنشان را می‌خواند و می‌شود همان برنامه‌نویس خوبی که دنبالش بودند.

بخش دیگری از کارَت کنارآمدن با تغییرهای غیرمنتظره است: ممکن است ناگهان چیزی بخواهند که باید از اول می‌دانستی، اما حالا آماده‌اش نیستی. این بیشتر مشکلی انسانی است که باید حلش کنی، نه فقط یک مشکل فنی که مکانیکی رفع شود. یعنی اگر می‌توانستی همهٔ تعامل‌های انسانی را هم مکانیکی کنی، معنایش این نبود که خودت را به معنای واقعی کلمه یک ربات می‌دانی؟

۴. هیچ پاسخِ درستِ همگانی‌ای وجود ندارد

برای خیلی از مسئله‌ها و بده‌بستان‌ها جواب درستِ همگانی‌ای وجود ندارد. اگر توسعهٔ نرم‌افزار را یک رشتهٔ دانشگاهی می‌دانستیم شاید وجود داشت، اما فعلاً جواب تا حد زیادی به نظر برنامه‌نویس بستگی دارد. این ما را می‌رساند به موضوع تازه‌ای:

آدم‌ها هرکدام نظر و موضع خودشان را دارند

به‌نظر من، یکی از تفاوت‌های کلیدی انسان و هوش مصنوعی این است که آدم‌ها به‌سادگی نظر و موضع خودشان را دارند (یا به عبارت دقیق‌تر انگلیسی، opinionated هستند)؛ و این در همهٔ حوزه‌ها درست است. منظورم از اینکه آدم‌ها نظر خودشان را دارند این است که درک خودشان را از دنیا، معنی و مفهوم شغلشان، شیوهٔ پیاده‌سازی و انجام کارها و چیزهای دیگر شکل می‌دهند.

شاید از این حرف چنین برآید که برنامه‌نویس محصولی نمی‌سازد که از نظر عینی خوب باشد. اما لزوماً این‌طور نیست. دلیلش این است که همان‌طور که چند بار گفتم، از زیر و بم توسعه سر درنمی‌آوریم. انجیل کاملی وجود ندارد که برای هر سناریو بگوید باید چه‌کار کنی. حتی شاید یک مرجع یگانه و پذیرفته‌شده هم وجود نداشته باشد؛ که محتمل‌ترین حالت همین است. مثلاً برای هر سیستم‌عامل چند زبان برنامه‌نویسی مختلف داریم تا با آن‌ها برنامه بسازیم. پس در نهایت همه‌چیز به تخصص و سلیقهٔ برنامه‌نویس بستگی دارد.

نظر و موضع داشتن یعنی می‌توانیم متخصصی استخدام کنیم که عمداً از انجام کاری کـه می‌گـوییم سـر بـاز بزند، چون نمی‌خواهد با انتخاب‌هایی که در حوزهٔ تخصصـش چـیزی از آن‌هـا نمی‌دانیم زنـدگی‌مان را خـراب کنیم. می‌توانی بگویی کـافی اسـت از هـوش مصـنوعی بخـواهیم وقـتی فکـر می‌کنـد اشـتباه می‌کـنیم بـا مـا مخالفت کند، اما به‌نظر من این حـرف نشـان می‌دهـد سـازوکار هـوش مصـنوعی را نمی‌فهمی یـا نادیـده‌اش می‌گیری.

هوش مصنوعی موجودی همه‌توان است که قرار است به همه جواب بدهد. همان ChatGPTای که بـا آن از کسی گله می‌کنی، آن شخص هم دارد از تو پیشش گله می‌کند. همهٔ نمونه‌هـای چت‌بـات را یـک نفـر آدم در نظر بگیر: قرار است این یک نفر به همه جواب بدهد. برای اینکه بـه درد همـه بخـورد ــ و این همـان چـیزی است که اصطلاح هوش عمومی مصنوعی به آن اشاره می‌کند ــ بایـد تا جـای ممکن بی‌موضـع باشـد؛ یعـنی خنثی‌ترین و معمولی‌ترین آدمی باشد که می‌توانی پیدا کنی. اگـر از هـوش مصـنوعی بخـواهی این‌قـدر مطیع نباشد، نتیجه‌اش آن‌طور که عبارت القا می‌کنـد بی‌نقص کـار نمی‌کنـد. مـدل هـوش مصـنوعی ناگهـان دربـارهٔ پروژهٔ توسعهٔ وب تو صاحب‌نظر نمی‌شود. سعی می‌کند محتمل‌ترین کلیشه را بپذیرد و بـر اسـاس آن جـواب بدهد؛ چون اگر قرار باشد طرف هیچ‌کس را نگیرد، از کجا بداند باید چه نظـری داشـته باشـد؟ اگـر مـدل‌های زبانی بزرگ را صـاحب‌نظر می‌کـردیم، کیفیتشـان مسـتقیم تحت‌تأثیر قـرار می‌گـرفت. مـدل زبـانی بـزرگی کـه سوگیری دارد برای همه مفید نیست. دیگر «عمومی» نخواهد بود.

برای همین هم مهارت‌هایی برای ایجنت‌ها داریم (agent skills)، مثل Ponytail کـه اساسـاً کـاری می‌کنـد ایجنت مثل یک برنامه‌نویس ارشد خسته رفتار کند که فقط می‌خواهد کار را هرچه زودتـر تمـام کنـد. سـعی می‌کنیم نظر و موضع‌داشتن را تقلید کنیم، درحالی‌که مغز انسان خیلی بهتر از پسش برمی‌آید.

انسان ها مسئولیت میپذیرند

یکی از قوی‌ترین نظرهای من در مخالفت با جایگزین‌کردن متخصص‌ها با هـوش مصـنوعی همین اسـت. در بخش قبل توضیح دادم نظر و موضع داشتن آدم‌ها کمک می‌کند محصول تمیزتری بسازیم، نه صرفاً اسـلاپ (slop)، یعنی محتوای بی‌کیفیت، سطحی و آبکی. اما اینجا می‌خواهم روی موضوع مهمی مکث کنم کـه فکـر می‌کنم مردم وقتی دربارهٔ هوش مصنوعی در عمل حرف می‌زنند فراموشش می‌کنند.

شکی نیست که هوش مصنوعی تأثیر بزرگی روی صنعت گذاشته است. یعنی چیزهایی را که دهه‌ها، اگر نـه هزاران سال، بدیهی می‌دانستیم به چالش کشیده. جهان‌بینی‌های بنیادین دارند از نو بازآرایی می‌شوند؛ بایـد

حواست به این فرایند باشد وگرنه دچار سوگیری می‌شوی.

یکی از چیزهایی که فراموش کرده‌ایم این است که قبلاً محصولات را با کمک آدم‌ها می‌ساختیم.

برگردیم به زمانی که هوش مصنوعی مولد وجود نداشت. می‌خواهی محصول یا برنامه‌ای بسازی، اما مهارت فنی نداری. احتمالاً فقط مهارت مدیریتی، یک ایده و یک عالم پول داری. معلوم است که تنها گزینه‌ات استخدام آدم‌هاست. حالا بسته به مقیاس پروژه و آشنایی‌ات با چرخهٔ توسعه، شاید فقط یک یا دو برنامه‌نویس استخدام کنی، یا یک دپارتمان فناوری کامل بسازی و سرپرست‌های فنی، مدیرهای محصول و یک مدیر ارشد فناوری (CTO) بگذاری.

در سناریوی اول، که یک برنامه‌نویس استخدام می‌کنی تا مثلاً برنامهٔ تحت‌وبی طراحی کند، چه چیزی باعث می‌شود به او اعتماد کنی که پولت را هدر ندهد؟ چند عامل وجود دارد:

1. **قرارداد**. دقیقاً مشخص می‌کنی از او چه می‌خواهی. اگر آن را نسازد، می‌توانی شکایت کنی. ترس از عدالت و قانون وادارش می‌کند کار کند. اما اگر نتوانسته باشی خواسته‌ات را خوب بیان کنی چه؟ قطعاً از همان اول کلی چیز را جا می‌اندازی؛ یادت هست گفتم نمی‌توانی صددرصد کل پروژه را از قبل مشخص کنی؟ پس چطور مطمئن می‌شوی فقط حداقل کار را انجام نمی‌دهد و در نمی‌رود؟ این ما را می‌رساند به عامل دوم:
2. **به اعتبارش اعتماد داری**. این یکی تقریباً خودش را توضیح می‌دهد. به تو گفته‌اند این برنامه‌نویس آدم خوبی است، و حتی بیشتر از پولی که می‌گیرد برایت کار انجام می‌دهد و کلاه سرت نمی‌گذارد. اما اگر چنین منبع اعتمادی نداشته باشی چه؟ یعنی تا حدی حتماً داری، اما شاید کافی نباشد. شاید فقط وانمود می‌کند چیزی می‌داند. این ما را می‌رساند به آخرین منبع اعتماد:
3. **به خودِ آن آدم اعتماد داری**. با او حرف می‌زنی، می‌فهمی چقدر آدم خوب و محترمی است، رزومه‌اش را نگاه می‌کنی، شاید حتی شبکه‌های اجتماعی‌اش را هم بررسی کنی، و حس خوبی پیدا می‌کنی که آدم مناسبی است. شاید به حرف و معرفی بقیه هم تکیه کنی. برادرت او را به تو معرفی کرده؟ پس حتماً آدم مطمئنی است. اینکه قانون دربارهٔ او هم مثل هر انسان دیگری اجرا می‌شود هم کمک می‌کند. می‌دانی حتی اگر به تو خیانت کند، می‌توانی کاری بکنی.

همین طرز فکر در سناریوهای دیگر هم صدق می‌کند. سرپرست فنی، برنامه‌نویس سینیور یا گروهی از سینیورها را استخدام می‌کنی چون اعتبار دارند و می‌توانی اعتماد کنی پروژه را بهتر اداره می‌کنند. با

استخدام یک گروه، خطر اینکه یکی‌شان پیش از آنکه بفهمی به تو خیانت کند کمتر می‌شود.

به‌نظر من همین چیزی است که کم داریم. فراموش کردیم به فرایندمان اعتماد داشتیم چون به آدم‌هایی که در آن کار می‌کردند اعتماد داشتیم. این فایدهٔ مدرنیته و زندگی در جامعه است.

حالا هوش مصنوعی را داریم؛ ابزاری که بی‌فکر می‌پذیریم و با جان‌ودلمان به آن اعتماد می‌کنیم. اگر هوش مصنوعی ناگهان کل شرکتت را نابود کند چه‌کار می‌کنی؟ هیچ‌کاری از دستت برنمی‌آید. موضوع فقط موقعیت‌های نادری نیست که اتفاق بدی می‌افتد؛ مسئله این است که می‌توانی آدم‌هایی را که استخدام می‌کنی **جایگزین** کنی. اگر با کسی قرارداد ببندی و پروژه‌ات را خراب کند، می‌توانی قراردادش را فسخ کنی و کسی **بهتر** پیدا کنی. وقتی Claude اشتباه می‌کند چه‌کار می‌کنی؟ نمی‌توانی همین‌طوری با یک مدل دیگر جایگزینش کنی و خیال خودت را راحت کنی. انتخاب‌هایت محدودند، چون به یک غیرانسان اعتماد کرده‌ای؛ ماشینی که حتی اگر ده‌ها میلیون دلار به تو خسارت زده باشد هم نمی‌توانی از آن شکایت کنی. هوش مصنوعی ابزار تو نیست؛ عملاً کارمند جدیدت است که قراردادی امضا نکرده. اگر سرمایه‌گذار باشی، اسمش را می‌گذاری امنیت؟

> *کد یک بدهی است، نه یک دارایی.* ــ *مهندسی نرم‌افزار در گوگل*، نوشتهٔ Titus Winters، Tom Manshreck و Hyrum Wright

هوش مصنوعی کارمند جدید توست

چیزی که اذیتم می‌کند تناقض پنهانی است که در ادعاها و توییت‌های مردم می‌بینم. اگر فکر می‌کنی هوش مصنوعی مهندسی نرم‌افزار را از رده خارج کرده، مشکلی ندارم؛ اما پای حرفت بایست. بعضی‌ها می‌گویند یک میلیون برنامه‌نویس سینیور را با پنج برنامه‌نویس میدل‌لول (میان‌رده) و هوش مصنوعی جایگزین کرده‌اند، اما هیچ‌وقت نمی‌گویند هوش مصنوعی کارمند جدیدشان است. در عوض ادعا می‌کنند فقط ابزار فوق‌العاده‌ای است و تو را سرزنش می‌کنند که بلد نیستی از آن استفاده کنی. فرق ابزار و کارمند جدید را بفهم؛ وگرنه انگار داری از سؤال‌هایی مثل «اگر کارمند توست، قرارداد امضا کرده؟» فرار می‌کنی.

قطعی در برابر غیرقطعی

منظور از **قطعی (deterministic)** این است که نتیجه به‌طور کامل با شرایط قبلی تعیین شده باشد؛ با

وجود آن شرایط هیچ نتیجهٔ دیگری ممکن نباشد. مثلاً اگر سیبی را از ارتفاع یک‌متری رها کنی ــ بدون عامل دیگری مثل باد شدید ــ می‌افتد. جاذبه افتادنش را تعیین کرده است. مثلاً ارادهٔ آزاد نقطهٔ مقابل جبرگرایی (معنیِ دیگرِ deterministic) است: نمی‌توانی اعمالت را انتخاب کنی چون از قبل برایت تعیین شده‌اند.

قبل از ظهور هوش مصنوعی مولد، برنامه‌نویسی قطعی بود. فرض کن یک اسکریپت ساده نوشته‌ای و وقتی اجراش می‌کنی خطا می‌دهد. می‌توانی بگویی زبان برنامه‌نویسی باعث شکستش شده؟ می‌توانی بگویی کامپیوتر باعث شکستش شده؟ نه؛ کامپیوتر دقیقاً همان کاری را می‌کرده که برایش ساخته شده. اسکریپت پر از باگ را تو نوشته‌ای. مسئولش هم تویی. زبان برنامه‌نویسی و کامپایلر قطعی‌اند. برای همین وقتی با باگ روبه‌رو می‌شوی، تقریباً مطمئنی یک جای کار را بد انجام داده‌ای؛ نه اینکه خروجی تصادفی تولید شده باشد و اگر دوباره اجراش کنی فرق کند. هرچند بار هم اجراش کنی، تا ابد همان خطا را می‌گیری (البته چند وضعیت ویژه هم وجود دارد، اما منظورم را می‌فهمی).

دلیل قطعی‌بودن زبان برنامه‌نویسی‌ای که استفاده می‌کنی این است که عامدانه طراحی شده. وقتی یک «hello world» ساده چاپ می‌کنی، همه‌چیز ــ از نحو زبانت گرفته تا کدبیس کامپایلر و خودِ کد ماشین ــ در طول سال‌ها عامدانه طراحی شده است. خروجی قطعی است چون از مجموعه‌ای از فرایندهای منطقی می‌گذرد که دقیقاً توضیح می‌دهند چرا آن کار را انجام می‌دهد.

هوش مصنوعی در عمل غیرقطعی (indeterministic) است. بله، معلوم است کسانی که این مدل‌ها را می‌سازند تحصیل‌کرده‌اند و می‌دانند دارند چه‌کار می‌کنند، اما این به این معنی نیست که از همهٔ جنبه‌های ریزِ وضعیت نهایی کاملاً خبر دارند. خودِ مدل بر اساس احتمال‌ها کار می‌کند. فرایند منطقی‌ای وجود ندارد که ورودی را به خروجی *تبدیل کند*. برخلاف ابزارهای قطعی، خروجی «نتیجهٔ ورودی» نیست. وقتی ۲+۲ را در ماشین‌حساب می‌زنی و دکمهٔ ورود را می‌فشاری، نتیجه همیشه ۴ است؛ مگر اینکه دستگاه خراب باشد. در هوش مصنوعی، حاصل ۲+۲ ممکن است هرچیزی باشد، چون مدل زبانی بزرگ با «۲+۲» همان‌طوری برخورد می‌کند که با «چطور کمردرد را درمان کنم»؛ فقط توکن‌ها را می‌شناسد و محتمل‌ترین توکن‌های بعدی را تولید می‌کند. خروجی‌اش **به ورودی وابسته است**، اما در تولید خروجی قطعی نیست. مغز انسان هم به همین شکل قطعی نیست.

کژگراییِ (بایاس) نتیجه‌نگر

می‌خواهم دربارهٔ جبرگرایی (determinism) در برنامه‌نویسی حرف بزنم، اما اول باید چیزی را توضیح بدهم

به‌اسم مغالطهٔ نتیجه‌محوری، یا کژگراییِ (بایاس) نـتیجه‌نگر: یعـنی قضـاوت دربـارهٔ کیفیت یـک تصـمیم بـر اساس نتیجه‌ای که ایجاد کرده است. مشکل این طرز فکـر این اسـت کـه از علت واقعیِ نتیجـه خـبر نـداریم. *فرض می‌کنیم* آن تصمیم باعث نتیجه شده، اما هنوز ثابتش نکرده‌ایم. چنین فکری می‌تواند به قضـاوت‌های به‌شدت نادرست منجر شود. کل نگرش نژادپرستی و تبعیض جنسیتی بر همین مغالطه بنا شده است:

«از نظـر آمـاری، سیاه‌پوسـتان اقلیت جمعیت‌انـد و مسـئول بیشـتر جرم‌هـا هسـتند؛ **پس** سیاه‌پوسـتان ذاتاً مجرم‌اند». دلیل غلط‌بودن این حرف این نیست که آمار اشتباه است (البته بسته به دورهٔ زمـانی‌ای کـه در آن زندگی می‌کنی، ممکن است اشتباه باشد). داده‌ها این فاصله را روشن نشان می‌دهنـد. امـا این فقـط همین است: یک مجموعه‌داده. برای ربط‌دادن این داده به ماهیت سیاه‌پوستان باید یـک دسـته قضـاوت دیگـر هم بکنی. برای توضیحش، بگذار کمی عمیق‌تر وارد فلسفهٔ استدلال و برهان شوم.

شاید در مدرسه یاد گرفته باشی هر استدلال سه بخش دارد:

- مقدمه‌ها: مجموعه‌ای از واقعیت‌ها یا حقیقت‌های پذیرفته‌شده
- نتیجه‌گیری: حاصل آن مقدمه‌ها
- استنتاج: پیوند منطقیِ مقدمه‌ها با نتیجه‌گیری

مثال:

- همهٔ انسان‌ها فانی‌اند.
- افلاطون انسان است.
- پس افلاطون فانی است.

استنتاج چیزی ذهنی نیست؛ پیوندی کـاملاً ریاضـی و مکـانیکی اسـت، و مغالطه‌هـا خودشـان را آنجـا نشـان می‌دهند. اگر همهٔ انسان‌ها فانی باشند و افلاطون انسان باشد، هیچ راهی نیست که افلاطون فـانی نباشـد. اگر افلاطون می‌توانست نامیرا باشد، پس همهٔ انسان‌ها فانی نیستند.

مقدمه‌ها یا واقعیت‌های علمی‌اند یا واقعیت‌هایی کـه قبلاً سرشـان بـه توافـق رسـیده‌ایم. مثلاً خـودِ جملـهٔ «افلاطون فانی است» می‌تواند مقدمهٔ استدلال دیگری باشد.

تعداد مقدمه‌ها باید بیشتر از یکی باشد. حتی اگر ظاهراً از یک مقدمه به نـتیجه‌ای برسـی، بـرای نتیجه‌گـیری

داری از مقدمهٔ دیگری هم استفاده می‌کنی؛ مقدمه‌ای پنهان که به‌نظرمان بـدیهی می‌آیـد. دلیلش این اسـت که مقدمهٔ اول خودش یک گزاره است؛ یک واقعیت. باید چیزی به آن اضافه کنیم تا واقعیت تازه‌ای بسازیم. مثلاً شاید بگویی: «وقـتی روی این قابلمـه آب می‌ریزم، بخـار می‌شـود؛ پس قابلمـه داغ اسـت». ظـاهراً یـک مقدمه داری (آب در قابلمه بخار شده)، اما یک پیش‌فرض بدیهیِ پنهان هم پشتش هست، چیزی شـبیه بـه این: «هر قابلمه‌ای که روی اجاقِ روشن است و آب را بخار می‌کند، داغ است».

اگر هم مقدمه‌ها درست باشند و هم استنتاج، نتیجه‌گیری نمی‌تواند غلط باشد. اگر معلوم شـد نتیجه‌گـیری غلط است، باید یکی از آن دو را زیر سؤال ببریم. پس همـان‌طور کـه گفتم، اگـر بفهمیم افلاطـون نامیراسـت، محتمل‌ترین مقصر این است که جملهٔ «همهٔ انسان‌ها فانی‌اند» را بی‌دلیل درست فرض کرده‌ایم.

حالا برگردیم به نژادپرستی؛ استدلالش چیست؟

مقدمه‌ها:

- بیشتر جرم‌ها را سیاه‌پوستان مرتکب می‌شوند.
- ؟

نتیجه‌گیری:

- سیاه‌پوستان ذاتاً مجرم‌اند.

بین این‌ها کلی فرض پنهان هست. چیزی که اینجا «ذات سیاه‌پوستان» می‌نامند، یعنی داشـتن رنگـدانه‌های پوستیِ متفاوت (یا شاید منظورشان چیزی عمیق‌تر باشـد. بـاور کن خـودِ نژادپرسـت‌ها هم دقیقـاً نمی‌داننـد منظورشان چیست). کسـی کـه این اسـتدلال را می‌کنـد، چـون می‌خواسـته بـه آن نتیجـه برسـد، یـک ویژگی فیزیکی (رنگ پوست) را به ویژگی ذهنی‌ای ربط داده است (ارتکاب جرم). مقدمهٔ دوم عملاً چیزی شـبیه این می‌شود: «هرگاه گروهی در آمار جرم‌ها بیش‌ازحد نمایان باشد، معنایش این است که آن گروه گرایشـی ذاتی و فطری به جرم دارد».

وقتی این مقدمهٔ پنهان را آشکار کنی، غلط‌بودن استدلال خیلی واضح می‌شود؛ اما کسی کـه این حـرف را زده عمداً مقدمه را پنهان نگه داشته و آمار ریاضیِ خالصی جلویت گذاشته که نمی‌توانی ردش کنی.

برنامه‌نویسی قطعی بود

بعضی مهندس‌ها در سازگارشدن با ایجنت‌ها مشکل دارند، چون به نتیجه‌محوری عادت ندارند. - Sam Lambert، ۲۰۲۶

چیزی که یک برنامه‌نویس را ارشد (سینیور) می‌کند، فهمیدن و احترام گذاشتن به این حقیقت است که برنامه‌ها چقدر زود پیچیده می‌شوند و برنامه‌نویسی را به شکلی انجام دهد که آن مشکل را تا جای ممکن کم کند. - Jonathan Blow، ؟

اگر ندانی چرا چیزی کار می‌کند، نمی‌فهمی چرا از کار افتاده. - The Pragmatic Programmer، نوشتهٔ David Thomas و Andy Hunt، ۱۹۹۹

خودِ کامپیوترها (همانطور که معلوم است) مکانیکی‌اند. قطعی‌اند و به شیوه‌ای بسیار مشخص و منطقی کار می‌کنند. حتی تولید عدد تصادفی هم الگوریتم‌هایی دارد (که دردسرهای امنیتیِ زیادی ایجاد می‌کند). همه‌چیز در کامپیوتر یک زمانی به دست آدم‌هایی طراحی شده است. برای همین کامپیوتر پیش‌بینی‌پذیر است.

آدم‌هایی که سراغ برنامه‌نویسی رفتند اغلب این پیش‌بینی‌پذیری را دوست داشتند. تمام عمرشان به این طرز فکر تکیه کرده‌اند. بااینکه کامپیوترها پیش‌بینی‌پذیر طراحی شده‌اند، هنوز هم برای برنامه‌نویس‌ها شگفت‌انگیزند. راه‌انداختن بازی ویدئوییِ پیچیده‌ات روی تکه‌ای سنگ تقریباً حس جادو دارد. ما برنامه‌نویس‌ها به این موضوع عادت کرده‌ایم، اما هنوز هم از این ایده نیرو می‌گیریم که از طریق صفحه‌ای به‌اندازهٔ ده در بیست سانتی‌متر، چیزی بسازیم که از راه دور به درد آدم‌ها بخورد. این را نگفتم که دربارهٔ نابودشدن لذت برنامه‌نویسی با هوش مصنوعی حرف بزنم (آن بحث دیگری است که بعداً سراغش می‌روم)، بلکه می‌خواستم نشان بدهم اگر این بخش از برنامه‌نویسی را برداری، چیزی برایمان می‌ماند که دیگر نمی‌فهمیمش: یک جعبهٔ سیاه. دلیل اینکه این ماشین پیچیده را می‌فهمیدیم و می‌توانستیم محصولات واقعاً محشری بسازیم این بود که ما برنامه‌نویس‌ها همیشه به قوانینش و شیوهٔ کارکردش تکیه می‌کردیم.

بخش مهمی از سینیور شدن غریزه‌ای است که در خودت پرورش می‌دهی. با یادگرفتن، پروژه‌ساختن و روبه‌روشدن با مشکلات گوناگون، طی سال‌ها تجربه به‌دست می‌آوری و این غریزه را صیقل می‌دهی. برای همین آن نقل‌قول Jonathan Blow را آوردم. پیش از آن جمله داشت می‌گفت صرفاً «حل‌کنندهٔ مسئله»

بودن لزوماً به این معنی نیست که برنامه‌نویس سینیوری هستی، چون یک جونیور هم می‌تواند مسئله‌های نسبتاً کوچکی را حل کند. برنامه‌نویس سینیور مسئله‌ها را خیلی پیش از آنکه رخ بدهند حل می‌کند. دید قدرتمندی نسبت به آینده پیدا می‌کنی و می‌بینی اگر فلان کار را بکنی چه چیزهایی ممکن است خراب شوند. وقتی کاری پرخطر، حقه‌بازانه (hacky) یا غلط انجام می‌دهی، در بدنت *حسش می‌کنی*. برای همین از برنامه‌نویس‌های سینیور می‌خواهند معماری طراحی کنند و کد را بازبینی کنند. این کار را فقط برای تمیزترنوشتن کد نمی‌کنند؛ دنبال معماری بد و بخش‌هایی هم می‌گردند که نگه‌داری محصول را دشوار می‌کنند.

بیشتر این تجربه‌ها را از راه درس‌گفتار یاد نمی‌گیری؛ به سه دلیل:

1. پیداکردن منبع دانشیِ قابل‌اعتماد که بفهمی‌اش آسان نیست. رجوع کن به بخش «مهندسی نرم‌افزار به‌عنوان رشتهٔ دانشگاهی».
2. شاید اصلاً نشود توضیحش داد. وقتی سینیوری، بزرگ‌ترین سلاح تو شهودت است و شهود همیشه به راحتی توضیح‌دادنی نیست.
3. برای یادگیری واقعی به تجربهٔ دست‌اول نیاز داری. اگر چیزی را خودت یاد گرفته باشی، می‌دانی تا وقتی تمرین نکنی یاد نمی‌گیری. اگر ساعت‌ها ویدئوهای طراحی را پشت سر هم اسکرول کنی (دوم‌اسکرولینگ)، هنرمند بهتری نمی‌شوی (این را از روی تجربه هم می‌گویم). شاید بشود همه‌چیز را نظری یاد گرفت و به‌راحتی هم فراموشش نکرد، اما واقعاً سخت است. بیشتر آدم‌ها اگر تمرین را کنار بگذارند فراموش می‌کنند. همین حالا می‌توانی کلی ویدئو پیدا کنی از برنامه‌نویس‌هایی که فقط به‌خاطر نصب‌کردن GitHub Copilot، نوشتن کد به زبان برنامه‌نویسی‌ای که سال‌ها از آن استفاده کرده‌اند را فراموش می‌کنند.

اما توسعه با هوش مصنوعی کاملاً نتیجه‌محور است. مستنداتی وجود ندارد که بگوید هر ورودی چه خروجی‌ای می‌سازد. کسانی که ادعا می‌کنند می‌دانند سازوکارش چیست، دارند تجربه‌های خودشان را تعریف می‌کنند. برای روشن‌شدن موضوع می‌توانم یک مقایسهٔ واضح بکنم:

فرض کن یک اسکریپت را کاملاً با دست نوشته‌ای. می‌توانی بگویی تک‌تک کلمه‌هایی که داخلش گذاشته‌ای چه‌کار می‌کنند؟ نقش دقیق هرکلمه در اسکریپتت چیست و چطور به نتیجهٔ برنامه کمک می‌کند (نتیجهٔ اجرای برنامه)؟ اگر برنامه‌نویس عمل‌گرایی باشی (pragmatic programmer)، قطعاً می‌توانی. حتی اگر اسکریپت آن‌قدر بزرگ شود که بخش‌هایی از آن را فراموش کنی، باز هم **می‌دانی** موقع نوشتنش چرا هر

بخش را گذاشتی. نمی‌توانی صرفاً «class» را به «category» تغییر بدهی چون این دو کلمه در ادبیات مترادف‌اند.

اما اگر برنامه‌نویس عمل‌گرایی نباشی، نمی‌دانی. داری همان کاری را می‌کنی که *The Pragmatic Programmer* اسمش را گذاشته «برنامه‌نویسی از روی تصادف» (programming by coincidence). عاشق این اصطلاحم.

وقتی پرامپت می‌نویسی، این امتیاز را از دست می‌دهی. دوآتشه‌های هوش مصنوعی هرقدر هم ادعا کنند بر ساخته‌شان اختیار دارند، به‌اندازهٔ برنامه‌نویس‌ها اختیارش را ندارند. اگر مهندس هوش مصنوعی باشی ــ یعنی با کمک ایجنت‌های هوش مصنوعی محصول بسازی ــ واقعاً می‌توانی بگویی هر کلمهٔ پرامپتت چه ربطی به خروجی هوش مصنوعی دارد؟ می‌توانی نشان بدهی اگر «help» را به «aid» تغییر بدهی چه می‌شود؟ می‌توانی ثابت کنی اگر یک نقطه را برداری چه اتفاقی می‌افتد؟ چون این چیزها اثر دارند. در برنامه‌نویسی، یک نقطه ممکن است خطا یا باگ ایجاد کند. بیشتر وقت‌ها اگر دقیقاً همان پرامپت را به همان مدل و همان هارنس بدهی، تقریباً همان نتیجه را می‌گیری؛ اما اگر پرامپت را عوض کنی، نتیجه ممکن است به‌وضوح تغییر کند. پس تک‌تک کلمه‌ها **اثر دارند**. چرا نداشته باشند؟ یادت باشد خودِ کامپیوتری که مدل را اجرا می‌کند قطعی است. عملیات ریاضی و وزن‌هایی که برای مدل زبانی بزرگ تعریف شده‌اند قطعی‌اند، چه ثابت باشند چه پویا. پس از نظر فنی خروجی هم باید قطعی باشد. اگر تولید عدد تصادفی قطعی باشد، خروجی هوش مصنوعی هم قطعی است. تنها چیزی که هم‌پای بقیه پیش نرفته، درک خودت از اتفاق‌هایی است که در پس زمینه می‌افتند. فقط تا حدی می‌توانی این درک را پس بگیری؛ آن هم با هدردادن هزاران ساعت برای آزمایش‌کردن یک مدل مشخص. هوش مصنوعی در نظریه قطعی است، اما در عمل غیرقطعی.

برای نشان‌دادن تفاوتش، این وضعیت را در نظر بگیر: فرض کن هیچ‌چیزی از برنامه‌نویسی نمی‌دانی و می‌خواهی زبان Python را یاد بگیری. برای خودت چند محدودیت می‌گذاری:

- هیچ جست‌وجویی نکنی.
- مستندات را نگاه نکنی.
- فقط خودت آزمایش کنی.

تنها راهی که اجازه داری این زبان را یاد بگیری این است که Python را اجرا کنی، شروع به تایپ کنی و سعی

کنی یاد بگیری. چیزی می‌نویسی، دکمهٔ اینتر (Enter) را می‌زنی، برنامه چیزی نشانت می‌دهد، نگاهش می‌کنی و دوباره همین کار را می‌کنی. شاید بالاخره یاد بگیری برنامهٔ ابتداییِ hello world بسازی، شاید هم نه. ممکن است ساعت‌ها، اگر نه روزها یا حتی ماه‌ها، طول بکشد. این یادگیریِ نتیجه‌محور است. هیچ‌وقت نمی‌فهمی Python چطور کار می‌کند؛ فقط یاد می‌گیری کارکردن با آن چه شکلی است. شاید بعد از هدردادن هزاران ساعت از یک تازه‌کار بهتر هم بشوی. اما فرق تو با کسی که همین حالا به روش درست شروع به یادگیری کرده این است که وقتی چیزی خراب می‌شود، تنها گزینه‌ات این است که دوباره آزمایش کنی؛ شاید جواب را پیدا کنی، شاید هم نه. «اوه، پایگاه‌دادهٔ کاربران پاک شد؛ بگذار آزمایش کنم ببینم چی شده... اوه، پایگاه‌دادهٔ مدیرها هم پاک شد». این شیوهٔ یادگیری و کارکردن دقیقاً همان چیزی است که در توسعه با هوش مصنوعی اتفاق می‌افتد. برای همین هم ماهر و کارآمدشدن در وایب‌کدینگ ــ کدنویسی با تکیه بر پرامپت و خروجی هوش مصنوعی ــ واقعاً سخت و زمان‌بر است.

اما تو به‌عنوان برنامه‌نویس فقط گزینهٔ آزمایش‌کردن را نداری؛ کلی اطلاعات هم در دسترست هست تا جواب دقیق را پیدا کنی، بعضا حتی قبل از اجرا کردنش!

توسعه‌دهنده‌های هوش مصنوعی شاید بگویند این اختیاری که به‌خاطر غیرقطعی‌بودن از دست می‌دهی اهمیتی ندارد، اما خیلی از جنبه‌های منفی را نادیده می‌گیرند که در بخش بعدی درباره‌شان حرف می‌زنم.

تأثیر هوش مصنوعی بر صنعت

هوش مصنوعی همین حالا هم چیزهای زیادی را در برنامه‌نویسی و صنعت تغییر داده است. این تأثیر ممکن است مستقیم به هوش مصنوعی مربوط باشد یا غیرمستقیم (مثل scapegoat: فردی که تقصیر به گردنش انداخته میشود تا به دلایل فرعی اخراج شود). باید بدانی این جنبه‌های منفی همه به هم وصل‌اند و یکدیگر را تشدید می‌کنند.

۱. سرعت را اشتباه تفسیر می‌کنند

میانگین سرعت خروجی هوش مصنوعی حدود ۱۰۰ توکن در ثانیه است. بعضی مدل‌ها تا ۲۰۰۰ توکن در ثانیه تولید می‌کنند و میانگین سرعت یک برنامه‌نویسِ انسان حدود ۵ توکن در ثانیه است. همین اعداد برای مقایسهٔ سرعت کافی نیستند، چون هوش مصنوعی سریع‌تر می‌نویسد اما سریع‌تر هم خرابکاری می‌کند؛ بنابراین وقت بیشتری صرف رفع باگ‌ها و پیداکردن ناهماهنگی‌ها می‌شود. در مقابل، برنامه‌نویس‌های

انسانی بخش قابل‌توجهی از وقتشان را صرف فکرکردن و بررسی موقعیت و کدبیس می‌کنند. درمجموع می‌بینی هوش مصنوعی چقدر از برنامه‌نویس‌های انسانی سریع‌تر و مقرون‌به‌صرفه‌تر است ــ یا دست‌کم چیزی که هیاهو می‌خواهد باور کنی همین است (هیاهو: hype، اغراق در تعریف و تبلیغ، ناشی از احساسات).

هوش مصنوعی فقط باهوش نیست؛ سریع هم هست. این موضوع قواعد بازی را از دیدگاه کسب‌وکار عوض می‌کند، و منظورم هم اثرهایش بر تک‌تک افراد است و هم بر شرکت‌های بزرگ.

مثلاً ساختن یک نمونهٔ تستی هزینهٔ خیلی کمی دارد، تقریباً هیچ ــ گاهی واقعاً هیچ. برای نمونه‌ای که فقط قرار است یک بار ایده‌ای را امتحان کند، لازم نیست منابعی مثل وقت، انرژی و پول را هدر بدهی.

اما وقتی می‌فهمند هوش مصنوعی سریع است، برداشت دیگری هم پیش می‌آید: «این‌قدر سریع است که می‌توانیم بیشتر ریسک کنیم»؛ آن هم در محیط تولید (پروداکشن). البته وقتی کاری را خیلی سریع انجام می‌دهی، هزینهٔ شکست به‌اندازهٔ قبل مهم به‌نظر نمی‌رسد. اما این طرز فکر سوگیری دارد، و سه دلیل برایش دارم:

1. **همهٔ خطرها فوری نیستند.** اگر محصولت واقعاً خوب از آب درآمد چه؟ آن‌وقت دیگر نمی‌توانی همه‌چیز را متوقف کنی و به مردم بگویی: «هی، انگار محصولمان را دوست داشتید؛ پس جمعش می‌کنیم تا از صفر، درست‌وحسابی و بی‌خطر بسازیمش!» جواب رایجی که به این حرف می‌دهند این است: «خب، پروژهٔ محیط تولید را هم می‌توانی ظرف چند روز وایب‌کد کنی». اما این جواب اساساً بر این فرض بنا شده که همهٔ خطرها فوری‌اند. می‌فهمم بعضی‌ها فکر می‌کنند می‌توانند پروژه‌هایی نسبتاً کم‌خطر را در زمانی خیلی کوتاه بسازند، اما آن بحث دیگری است که باید به آن بپردازیم. فعلاً خطر بلندمدت عمدتاً نادیده گرفته می‌شود.
2. **موفقیت را مردم راحت فراموش می‌کنند؛ شکست را همیشه یادشان می‌ماند.** اگر باور نمی‌کنی، به Unity نگاه کن. هنوز از آن یک تصمیم بد بهبود پیدا نکرده است. اگر مدام چیز بسازی و مدام شکست بخوری، خودبه‌خود می‌افتی توی دستهٔ «اسلاپ» ــ نه به‌خاطر موج فعلی ضد هوش مصنوعی. قبلاً از نظر فیزیکی نمی‌توانستی چیزها را با این سرعت بسازی. باید وقت و انرژی قابل‌توجهی می‌گذاشتی. همین جلوی تولید پیوستهٔ اسلاپ را می‌گرفت و باعث می‌شد واقعاً برای کارت زحمت بکشی. حتی اگر شکست می‌خوردی، مردم می‌دانستند تلاش کرده‌ای و کارت را چیز ارزانی حساب نمی‌کردی. اما حالا فقط یک‌مشت چیز جلویشان می‌اندازی و آشکارا با آن‌ها مثل

موش آزمایشگاهی رفتار می‌کنی تا ببینی به کدام پروژه واکنش نشان می‌دهند. مشتری هیچ‌وقت از چنین چیزی خوشش نمی‌آید. اگر صاحب کسب‌وکاری باشی و مشتری بفهمد داری روی او آزمایش می‌کنی، کارت را درست انجام نمی‌دهی. مثلاً اثربخشی آزمون A/B از این می‌آید که کاربر خبر ندارد چنین آزمایشی در جریان است.

3. **شکست روی هم انباشته می‌شود** و پیش از آنکه بفهمی، کوهی از چیزهای خراب دوروبرت جمع شده و نمی‌دانی چرا. در بخش‌های «بازگردانی‌ها» و «بدهی سه‌گانه (triple debt)» درباره‌اش حرف می‌زنم.

۲. مهندسی نرم‌افزار را پراسترس‌تر می‌کند

دکتر الوک کنوجیا که به داکتر کِی (Dr. K) هم معروف است، روان‌پزشکی است که ویدئوهایش را بیش از همه دنبال کرده‌ام. گاهی ویدئوهای یوتیوبش ذهنیت و شیوهٔ زندگی‌کردنم را کاملاً عوض کرده‌اند.

یکی از ویدئوهایش اسمش هست «چرا برنامه‌نویس‌ها مدام فرسوده می‌شوند» (که به‌شدت پیشنهادش می‌کنم، مخصوصاً اگر برنامه‌نویسی یا بخواهی برنامه‌نویس شوی). ویدئو را با این واقعیت آماری شروع می‌کند که مهندسی نرم‌افزار جزو شغل‌هایی است که بالاترین نرخ خودکشی را دارند. با الهام از این ویدئو و تجربه‌های خودم، می‌خواهم دربارهٔ چیزی حرف بزنم که به‌نظر من دلیل اصلی استرس و فرسودگی شغلی (burnout) در مهندسی نرم‌افزار است.

اگر تابه‌حال پیش روان‌درمانگر رفته‌ای یا ویدئوهایی دربارهٔ مشکلات خواب دیده‌ای (داکتر کِی هم چندتایی دارد)، می‌دانی بین میزان بهره‌وری‌ات در طول روز و اینکه چقدر خوب می‌توانی بخوابی ارتباط مستقیمی هست. انگار تا اینجای کار توی دی‌ان‌ای‌مان حک شده است. تمام هدف زندگی‌مان بهره‌وربودن است، چون برای جامعه این باارزش‌ترین چیز است؛ پس شاید تنها راهی باشد که خودمان هم برای خودمان ارزش قائل شویم. خودت هم می‌توانی امتحانش کنی: یک روز کامل هیچ کار مفیدی نکن و فقط ویدئوی کوتاه (ریلز) نگاه کن. شب که می‌خواهی بخوابی می‌بینی نمی‌توانی و باید ساعت‌ها توی تخت دراز بکشی تا خوابت ببرد. اگر همان روز را بگذرانی اما کمی هم بهره‌ور باشی ــ مثلاً نیم ساعت پیش از خواب ــ خیلی کمکت می‌کند.

همین است اصلِ حرف‌هایی که دربارهٔ «عاشق شغلت باش» می‌زنند. هر شغلی سخت است. هیچ شغلی آسان یا شبیه بازی ویدئویی نمی‌شود. اما فرق هست بین شغلی که از تک‌تک جنبه‌هایش متنفری و شغلی که خسته‌ات می‌کند اما حس بهره‌وری به تو می‌دهد. دلیل اینکه اصطلاح «شغل شرکتی» بار منفی پیدا

کرده این است که بیشتر این شغل‌ها هیچ حسی از بهره‌وربودن به تو نمی‌دهند. فقط محصولی برای مدیری بالادستی می‌سازی که می‌خواهد ایده‌هایش را به رخ بکشد و ترفیع بگیرد و از این حرف‌ها (می‌توانی از بخش نظرات ویدئوی داکتر کِی چندین نمونه پیدا کنی). باید حسی از مالکیت و شراکت داشته باشی تا حس کنی «داری کاری انجام می‌دهی»، نه اینکه فقط وقتت را تلف می‌کنی. حتی اگر وقت تلف می‌کنی، دست‌کم باید چیزی گیرت بیاید؛ نه‌فقط پول، بلکه تجربه و دانش هم. اگر هیچ‌کدام را نگیری، خیلی زود از شغلت متنفر می‌شوی و فرسوده.

برنامه‌نویسی و توسعه یک مشکل اساسی در بهره‌وری دارند: بیش‌ازحد نتیجه‌محورند. گاهی می‌بینی یک هفته، یک ماه یا حتی چند ماه روی پروژه‌ای کار کرده‌ای، اما در ظاهر پروژه اصلاً عوض نشده است (مثلاً اگر وب‌سایت باشد، هنوز همان شکلی به‌نظر می‌رسد). گاهی بک‌اند هم تغییری نکرده و قابلیت تازه‌ای اضافه نشده است. فقط کد را تمیز کرده‌ای تا در آینده «کمتر به مشکل بخوری»؛ یعنی رفکتورینگ (بازآرایی کد). از این کار حس بهره‌وری و پاداش‌گرفتن بیرون‌کشیدن واقعاً سخت است. اگر دوست برنامه‌نویسی داری و میبینی تا چهار صبح نشسته کد می‌زند، دلیلش این است که منتظر رسیدن به یک نتیجهٔ خاص است تا آن‌قدر راضی شود که برود بخوابد (البته دلیل‌های دیگری هم هست).

شاید جمله‌های انگیزشی‌ای مثل «مسیر مهم است، نه مقصد» به چشمت خورده باشد. در برنامه‌نویسی واقعاً گمراه‌کننده است. بد برداشت نکن؛ من خودم عاشق فرایند و عمل برنامه‌نویسی‌ام. اما اگر هیچ هدفی را به دست نمی‌آوردم و حس می‌کردم هیچ کاری نکرده‌ام، عاشقش نمی‌ماندم. به‌عنوان برنامه‌نویس، اغلب بابت باگی از کوره درمی‌روی؛ چون ساعت‌های زیادی را صرفش کرده‌ای و هنوز درست نشده. کنار جاده از کوره درنمی‌روی چون گاوی نمی‌بینی؛ هر اتفاقی بیفتد می‌توانی از آن لذت ببری. اما در برنامه‌نویسی نمی‌توانی خودت را مجبور کنی از یک مشکل به معنای واقعی کلمه، لذت ببری. یعنی اگر بتوانی از آن لذت ببری، دیگر هیچ‌چیزی در دنیا نمی‌تواند ناراحتت کند.

این به مفهوم بسیار مهمی در برنامه‌نویسی و توسعه به‌طور کلی برمی‌گردد: **نمی‌دانی**. وقتی اسکریپت می‌نویسی، بیشتر وقت‌ها عمداً کد پر از باگ نمی‌نویسی. هر کاری را می‌کنی که فکر می‌کنی درست است. تمام ذهنت این است که «دارم این کار را درست انجام می‌دهم». شاید حتی از چیزی که نوشته‌ای یا تغییر داده‌ای خیلی مطمئن باشی. اما بعد برنامه را اجرا می‌کنی و ناگهان هزارویک خطا جلویت می‌ریزد. شاید آخرش درستشان کنی، اما مشکل این است که این فرایند ذاتاً تو را از محدودهٔ امن و راحتت بیرون می‌کشد. مدام با تو مخالفت می‌کند و مستقیم می‌گوید: «کارت را بد انجام داده‌ای؛ اشتباه کرده‌ای». و تا رفعش

نکنی، حاضر نیست آن‌طور که می‌خواهی کار کند. می‌توانی باگ یا خطایی را نادیده بگیری (بازی‌سازها حتی به انتشار بازی‌هایی که خطا نشان می‌دهند عادت کرده‌اند)، اما منظورم این است که **هیچ‌وقت حس خوبی ندارد**. می‌توانی خودت را گول بزنی و فکر کنی از آن لذت می‌بری، اما ببخشید، این دیگر سندروم استکهلم است. واقعیت این است که فقط چون به مقصد اهمیت می‌دهی از مسیر لذت می‌بری. بدون مقصد، همهٔ این باگ‌ها فقط باری روی دوشت می‌شوند.

عمداً اسم آن مفهوم را «نمی‌دانی» گذاشتم تا یکی از آزاردهنده‌ترین جنبه‌های برنامه‌نویسی را هم بگویم: وقتی با باگ روبه‌رو می‌شوی، از همان اول نمی‌دانی چرا به‌وجود آمده یا رفعش چقدر طول می‌کشد. *واقعاً سخت است* که بفهمی رفع هر باگ چقدر وقت می‌برد، چون گاهی مشکل در هر صورت از چیزی که فکر می‌کردی عمیق است. می‌دانی کارکردن در چنین حوزه‌ای چقدر استرس‌زاست؟ وقتی ضرب‌الاجل داری و یک باگ می‌تواند زمان‌بندی‌ات را خراب کند؟

وقتی می‌فهمی قضیه فقط باگ‌ها نیست و خودِ برنامه هم همین‌طور است، وضع بدتر می‌شود. با کسب تجربه، حس درونی و درکی پیدا می‌کنی از اینکه توسعهٔ هرچیزی چقدر زمان می‌برد. اما در نهایت، تا وقتی پروژه‌ای را با همان دامنه انجام نداده باشی، واقعاً نمی‌توانی مطمئن باشی چقدر طول می‌کشد. و موضوع آن‌قدر ساده نیست که تحقیق کنی و جوابش را پیدا کنی. گاهی باید اول خودت انجامش بدهی. چون گلوگاهی که از آن می‌ترسی خیلی عمیق در زمان‌بندی پروژه قرار گرفته و تحقیق یا نمونهٔ تستیِ ساده نمی‌تواند نشانش بدهد. روش‌هایی هست، مثل روش گلولهٔ ردیاب (tracer bullet method) که در The Pragmatic Programmer درباره‌اش حرف زده‌اند؛ اما این روش‌ها هم همیشه ایمنی‌ات را تضمین نمی‌کنند.

وقتی فکر می‌کنی دیگر بدتر نمی‌شود، باز بدتر می‌شود. همان‌طور که در ویدئوی داکتر کِی می‌بینی، یکی از رایج‌ترین دلیل‌های فرسودگی برنامه‌نویس‌ها این است که **از همان اول طوری چیده شده که شکست بخورند**. در ویدئو دربارهٔ تغییرکردن مداوم دامنه و زمان‌بندی پروژه در توسعهٔ نرم‌افزار حرف می‌زند. شش هفته مانده به عرضه، مدیرعامل یک پادکست می‌بیند و ناگهان از تو می‌خواهد فلان قابلیت را اضافه کنی یا شیوهٔ کارت را کاملاً عوض کنی چون ابزار تازه‌ای برای هوش مصنوعی منتشر شده است. اگر از پسش برنیایی، تقصیر را گردن تو می‌اندازند. بدتر اینکه اگر با اضافه‌کاری و آسیب‌زدن به بدن و برنامهٔ خوابت موفق شوی، هنجار عوض می‌شود. ناگهان از تو بیشتر انتظار دارند. ویدئو دربارهٔ دورکاری و چیزهای خیلی بیشتری هم حرف می‌زند که می‌توانند وضع را بدتر کنند.

اما نکته‌ای که می‌خواهم اضافه کنم این است که حالا فقط قابلیت‌ها و زمان‌بندی‌ها را عوض نمی‌کنند؛ مهارت‌هایت را هم خراب می‌کنند. هرچه بیشتر کارت را به هوش مصنوعی واگذار کنی، مهارت‌هایت را بیشتر فراموش می‌کنی. بعداً بیشتر توضیح می‌دهم، اما خلاصه‌اش این است که ضعیف‌تر می‌شوی، برنامه‌هایت باگ‌های بیشتری پیدا می‌کنند و وقتی خراب می‌شوند، دیگر کسی را نداری که مقصرش بدانی؛ حتی خودت را هم نه. این وضعیت در کوتاه‌مدت یا بلندمدت استرس و فرسودگی زیادی ایجاد می‌کند؛ مخصوصاً در محیط کاری‌ای که در آن مسئولیت داری.

به‌خاطر انتظارهای بد و مدیریت بد، هوش مصنوعی بخش‌های خوب شغل را از ما گرفته و بدترین بخش‌ها را برایمان گذاشته است. دلیل اینکه با اطمینان پروژه‌های سخت را قبول می‌کردیم این بود که بر آن‌ها اختیار داشتیم. می‌توانستیم با اطمینان بیشتری ضرب‌الاجل قبول کنیم. می‌دانستیم چرا هرچیزی ساخته شده؛ خودمان ساخته بودیمش. حتی اگر از اول درگیر پروژه نبودیم و کدبیسِ ازقبل‌موجودی تحویلمان می‌دادند، دست‌کم می‌توانستیم با اجرای یک `git blame` کوچک برنامه‌نویس‌های قبلی را مقصر بدانیم و به مدیرها نشان بدهیم چرا باگ از اول وجود داشته یا کدبیس قدیمی چه محدودیتی ایجاد کرده است. حالا همه می‌گویند «نرم‌افزار حل شده»؛ پس اگر نتوانی کاری را بکنی، قطعاً مشکل از مهارت خودت است. در بخش «انسان ها مسئولیت میپذیرند» توضیح دادم مدیرها چطور به آدم‌هایی که روی پروژه‌هایشان کار می‌کردند اعتماد داشتند. از نظر اقتصادی اعتمادکردن به هوش مصنوعی برای پروژه‌ات امن به‌نظر نمی‌رسد. اینجاست که نتیجه برعکس می‌شود. مسئول خروجی هوش مصنوعی تویی (یکی از جواب‌ها این است که «فقط کد را بخوان»، اما بعداً درباره‌اش حرف می‌زنیم).

مسئله فقط مسئولیت هم نیست. همان‌طور که گفتم، دلیل تحمل‌کردن همهٔ جنبه‌های منفی حرفه‌مان ــ از جمله اینکه کار با بهره‌وری سر سازگاری ندارد ــ این بود که از چیزهایی که ساخته بودیم لذت می‌بردیم. نتیجه را مال خودمان می‌دانستیم. به‌جای ساختن چیزها، کل شغلمان شده «پیداکردن باگ‌ها و مشکل‌هایی که ما نساخته‌ایم اما مسئولشان هستیم». رفع مشکل‌هایی که اصلاً نباید وجود داشته باشند هیچ‌وقت حس بهره‌وربودن نمی‌دهد. دست‌کم اگر پروژه مال خودت باشد، ایده و همه‌چیزش، آن موقع شاید کمی فرق کند. اما اگر برای کسی کار کنی، فقط یک واسط انسانی برای هوش مصنوعی هستی. وقتی هوش مصنوعی باگ می‌سازد و پیمانکار یا مدیر دنبال مقصر می‌گردد، تو کیسه‌بوکسشان می‌شوی. ببخشید، اما این دقیقاً دستور پخت فرسودگی شغلی (burnout) است.

۳. فقط تولید کد سریع‌تر شده

یک ویدئوی عالی از Adam Bender هست؛ او مهندس ارشد گوگل است (Principal Engineer) و در آن دربارهٔ هوش مصنوعی و آینده‌اش حرف می‌زند. ویدئوی بی‌نظیری است.

یکی از مفهوم‌های کلیدی‌ای که مطرح می‌کند این است که هر پروژهٔ نرم‌افزاری جنبه‌های اجتماعی و فنی زیادی دارد. هوش مصنوعی فقط بعضی بخش‌ها را سریع‌تر کرده و بخش‌های دیگر هنوز کندند. مثلاً تولید کد ده برابر سریع‌تر شده، اما بازبینی کد نه (البته برای کسانی که بازبینی کد برایشان مهم است). از اینجا نتیجه می‌گیریم که **گلوگاه انسان‌ها هستند**. «و آدم‌ها وقتی تحت فشارند کارهای بامزه‌ای می‌کنند» (منظورش فشارِ گلوگاه‌بودن است). کاملاً باهاش موافقم.

برنامه‌نویسی سخت است؛ برنامه‌نویسی بدون حس بهره‌وری سخت‌تر است؛ و برنامه‌نویسی وقتی هم حس بهره‌وری نداری و هم حس می‌کنی سرعت تیمت یا شرکت را کم کرده‌ای، بدترین حالت است. انگار هر گوشهٔ این سامانه دارد تهی‌ات می‌کند و واقعاً نمی‌دانی چرا داری این کار را می‌کنی. وقتی استخدام شده‌ای کاری انجام بدهی، این موضوع خیلی هم ترسناک است. با خودت می‌گویی: «اگر بفهمند دارم سرعتشان را کم می‌کنم، اخراجم می‌کنند.» این استرس هم خیلی زود فرسوده‌ات می‌کند.

حتی اگر حس نکنی گلوگاه هستی، باز حس بدی خواهی داشت؛ چون هوش مصنوعی نه‌فقط کارت را آسان‌تر کرده، بدترش هم کرده است. بدن، انرژی و وقت خودت هیچ‌کدام ده برابر نشده‌اند. اما انتظارها چرا.

در بخش نظرات همان ویدئو حرفی بود که خیلی خوشم آمد؛ می‌گفت «انسان‌ها گلوگاه نیستند، بلکه محدودکنندهٔ نرخ‌اند». کارِ محدودکنندهٔ نرخ (ریت لیمیتر) این است که مصرف هر کاربر را محدود کند تا بار بیش‌ازحد روی سامانه نیندازد. در برنامه‌نویسی هم همین‌طور است. انسان‌ها گلوگاه نیستند؛ آن‌ها هستند که همه‌چیز را ساختارمند نگه می‌دارند. برای این کار باید این‌طور فکر کنی: «مهندسی نرم‌افزار فقط برنامه‌نویسی و پیاده‌سازی قابلیت‌ها نیست؛ مهم‌تر از همه، طراحی سامانه‌ای است که مقیاس‌پذیر و نگهداشت‌پذیر باشد». متأسفانه همهٔ مدیرها این طرز فکر را نمی‌فهمند یا آن‌قدر برایش احترام قائل نیستند. گاهی واقعاً انتظار دارند نمونهٔ تستی را منتشر کنی چون از نظرشان زیادی صیقلی و آماده به‌نظر می‌رسد (مخصوصاً در عصر هوش مصنوعی). آخر سر طوری چیده می‌شوی که شکست بخوری و بعد هم تقصیر را گردنت می‌اندازند.

۴. بازگردانی به نسخهٔ پیشین (rollback)

با سرعت، مسئولیت هم می‌آید. - خالهٔ ایجنتیکِ اسپایدرمن

«هوش مصنوعی تقویت‌کننده است». DORA این را گفته، و به‌نظر من دقیق‌ترین توصیف از ماهیت هوش مصنوعی است. شاید گردش‌کارت را با موفقیت سریع‌تر کرده باشی، اما هم‌زمان احتمال شکست را هم بالا برده‌ای. اگر قبلاً روزی با ده خطا روبه‌رو می‌شدی، حالا صدتا می‌بینی. شاید به‌نظر برسد این بخش به اثر شمارهٔ ۳ مربوط است؛ دلیل اینکه اینجا آوردمش این است که باگ‌ها روی هم جمع می‌شوند. سریع‌تر کد می‌زنی و سریع‌تر ادغام می‌کنی؛ یا بدتر از آن، سریع‌تر منتشر می‌کنی. اگر نسخهٔ قبلی باگی داشته باشد، بسته به تعداد گزارش‌های باگی که می‌گیری، شاید خیلی دیرتر خبردار شوی.

قبلاً چون آهسته‌تر توسعه می‌دادی، خیلی از این باگ‌ها طبیعتاً زودتر پیدا می‌شدند. این باعث می‌شد رویکرد امن‌تری داشته باشی. می‌توانستی باگ را رفع کنی بی‌آنکه چیزهای زیادی را بشکنی، یا با امنیت بیشتری به نسخهٔ قبلی برگردی. بازگردانی‌ها بیشتر وقت‌ها خطرناکند. اگر ساختار چیزی را عوض کرده باشی ــ مثلاً خودِ پایگاه‌داده را ــ نمی‌توانی همین‌طوری به نسخهٔ قبلی برگردی و تمام. ممکن است اطلاعات کاربران از دست برود.

اما الان پیش از آنکه کسی متوجه شود، یک میلیون قابلیت تازه ساخته‌ای. شاید حتی به کارکردن باگ هم وابسته شده باشی. پس اگر بعداً باگ‌ها را پیدا کنی، وقت و پول زیادی هدر می‌رود.

اگر کم‌تجربه باشی شاید این وضعیت برایت یک‌درمیلیون به‌نظر برسد؛ اما هرچه تجربه‌ات بیشتر شود، کمتر احساس امنیت می‌کنی. شبیه آن جملهٔ معروف است که هرچه بیشتر بدانی، بیشتر می‌فهمی چقدر نمی‌دانی. یک جونیور شاید کدی افتضاح بنویسد. کار می‌کند، اما همین؛ فقط همان لحظه کار می‌کند. تو به‌عنوان سینیور آن بوهای بد کد را می‌بینی و شهودت هشدار می‌دهد که **نباید** ولش کنی به امان خدا. حتی اگر عمداً بعضی بوهای بد کد و باگ‌ها را نگه می‌داری، باید حواست به آن‌ها باشد تا خیال نکنی پروژه بی‌نقص کار می‌کند و مشکل‌های بیشتری روی هم جمع نکنی. این خطرِ بازگردانی هم، همراه با بقیهٔ مشکل‌هایی که گفتم، واقعاً امن به‌نظر نمی‌رسد.

۵.۱. دیگر یاد نمی‌گیری

یکی از چیزهایی که دربارهٔ واگذارکردن کدنویسی به هوش مصنوعی بیشتر از همه نگرانم می‌کند این است

که می‌تواند جلوی رشدت را بگیرد. بگذار با مثالی از فرایندی که کسانی که با هوش مصنوعی توسعه می‌دهند طی می‌کنند توضیح بدهم:

1. پروژه‌ای تازه شروع می‌کنی. شاید خودت برنامه‌نویس باشی، اما ابزارهایی را که می‌خواهی در پروژه به‌کار ببری نمی‌شناسی. قطعاً هم قبلاً دقیقاً همین *نوع* پروژه را انجام نداده‌ای.
2. نرم‌افزار Claude Code را دانلود می‌کنی.
3. شاید با توضیحی کوتاه یکراست سراغ پیاده‌سازی بروی؛ یا اگر تجربه‌ات کافی باشد، اول تا جایی که می‌توانی سامانه و معماری را طراحی کنی. شاید نیم ساعت با هوش مصنوعی حرف بزنی و مشخصات را تا جای ممکن دقیق کنی، یا حتی کمی هم کد بنویسی. سعی می‌کنی خاک خوبی آماده کنی؛ زیربنای بی‌نقصی که هوش مصنوعی بتواند روی آن بسازد. چون موضوع و ابزارها برایت کم‌وبیش تازه‌اند، همین آماده‌سازی شاید چند روز یا چند هفته طول بکشد.
4. ایجنت‌ها و مدل‌های پیشرو را با گردش‌کارِ ایجنت‌محوری که برایشان ساخته‌ای به کار می‌اندازی. حسابی پیش می‌روند، اما تو مکث می‌کنی و کد را می‌خوانی. می‌خواهی پروژه را عمیق بفهمی چون برنامه‌نویس عمل‌گرایی هستی و قرار است بابت چیزی که ساخته‌ای پاسخ‌گو باشی.
5. جالب اینجاست که در کد تولیدشده هم خطا پیدا می‌کنی: فنی، شناختی یا مربوط به نیت. یکی‌یکی رفعشان می‌کنی و به Claude می‌گویی به خاطر بسپارد. آن‌ها را در فایل ثبت تصمیم معماری (ADR) یا CLAUDE.md پروژه می‌نویسی تا همان مشکل دیگر تکرار نشود.
6. این فرایند را تکرار می‌کنی و بالاخره هم‌زمان چند چیز دستگیرت می‌شود:

1. فهمیدن پروژه سخت‌تر شده است. تو خودت برنامه‌نویس *هستی*؛ حتی در حوزه‌های دیگری هم سینیوری. شاید زیربنای پروژه را هم از نظر عینی بی‌نقص ساخته باشی. اما چیزهای زیادی هست که فقط با خواندن کد باید یاد بگیری: زبان برنامه‌نویسی، فریم‌ورک‌ها، ابزارها و خودِ نوع پروژه. می‌دانی می‌شود همه را هم‌زمان یاد گرفت، اما دست‌کم به زمان نیاز داری.
2. هوش مصنوعی دارد بهتر می‌شود. هرچه بیشتر روی پروژه کار می‌کند، بهتر می‌شود. حالا دیگر همیشه قراردادها و عرف‌ها را رعایت می‌کند. تشخیص ایرادهایش روزبه‌روز سخت‌تر می‌شود. می‌دانی بالاخره پروژه آن‌قدر بزرگ می‌شود که دیگر نمی‌توانی دنبال کنی چه‌چیزی به چه‌چیزی وصل است. هوش مصنوعی آشکارا ارتباط‌ها را خیلی بهتر از تو می‌بیند، چون بیاییم روراست باشیم، ماشین است. می‌تواند ده اسکریپت را بدون جاانداختن حتی یک

ویرگول از بر بخواند. مغز ل, محدود است؛ با صرفاً بازبینی ده‌هزار خط کد نمی‌تـوانی بفهمی چه خبر است.

3.حس می‌کنی خودت گلوگاه شده‌ای. مدل می‌تواند در بیست ساعت کل پروژه را از نو زیرورو کند و تو تقریباً تمام این وقت را با مکث‌دادن به آن برای بازبینی و یادگیری هدر می‌دهی.

7.این فرایند را باز هم تکرار می‌کنی و بالاخره بـه خـودت می‌گـویی: «اصـلاً اینجـا چـه غلطی می‌کنم؟» هدفت از انجام کار را گم میکنی. حس نمی‌کـنی کـاری انجـام داده‌ای. فقـط سـرعت پیشـرفت را کم کرده‌ای و پروژه خیلی بیشتر از ضرب‌الاجلی که پیش‌بینی کرده بودی طول می‌کشد. انگار تقریباً هیچ افزایش سرعتی به دست نیاورده‌ای.

8.کم‌کم دیگر کد را نمی‌خوانی و می‌گذاری Claude همه‌چیز را انجام بدهـد. رسـماً تبـدیل شـده‌ای بـه واسط انسانیِ Claude.

9.راسـتی، چـون همه‌چـیز را بـه هـوش مصـنوعی سـپرده‌ای کلی وقت آزاد هم داری. می‌روی ظـرف می‌شویی و با خودت می‌گویی: «من دارم ظرف می‌شورم و هوش مصنوعی کـارم را انجـام می‌دهـد. قرار بود برعکس باشد». بحران وجودی می‌آید و به تو سلام می‌کند.

این دقیقاً همان فرایندی است کـه همـه طی می‌کننـد. اگـر از حوزه‌هـای دیگـری آمـده باشـی و بـا مهندسـی نرم‌افزار چندان آشنا نباشی، شاید بگویی: «خب، در حوزه‌ها و موضوع‌هایی که قبلاً بـا آن‌هـا کـار نکـرده‌ای تا این حد از هوش مصنوعی استفاده نکن»؛ چون انتظار داری برنامه‌نویس اول آن چیزها را یاد بگـیرد و بعـد از هـوش مصـنوعی بـرای خودکارکردنشـان اسـتفاده کنـد. امـا این پیشـنهاد نشـان می‌دهـد اساسـاً نمی‌فهمی برنامه‌نویس‌بودن یعنی چه، یا آن را نادیده می‌گیری. اگر انسـانی اسـتثنایی نباشـی، تقریبـاً مطمئنم چیزهـای زیادی هست که نمی‌دانی. زبان‌های زیادی هست که قطعاً بلد نیستی، ابزارهای زیادی هست که نمی‌شناسی و پروژه‌های زیادی هست که هیچ‌وقت امتحان نکرده‌ای؛ حتی اگر سال‌ها تجربه داشته باشی و برنـامه‌نویس سـینیوری باشـی! همین اواخـر ویـدئویی از مهنـدس‌های گوگـل دیـدم کـه یکی‌شـان ــ Aja Hammerly ــ به‌معنای واقعی کلمه گفت همیشه با ADK (یک فریم‌ورک گوگل) کار می‌کنـد، بی‌آنکـه آن را بلـد باشـد: «من همهٔ آن کدها را تولید می‌کنم». مگر اینکه شغلت خیلی ثابت باشد، اخراج نشوی و همه‌چیز هم طبق برنامه پیش برود، مدام بین ابزارها و استک‌های مختلف (پشته‌های فناوریِ پروژه) جابه‌جـا می‌شـوی. سـینیوربودن یعنی خیلی سریع یادشان می‌گیری، با هرکدام مثل یک ابزار کار می‌کنی و در همان مسیر تجـربه‌ات را بیشـتر می‌کنی.

یادت هم باشد که هیچ‌وقت یک زبان را کامل یاد نمی‌گیری. بعد از شش سال کدنویسی با Python خالص، هنوز چیزهای کاملاً ابتدایی درباره‌اش پیدا می‌کنم؛ حتی دربارهٔ کلیدواژه‌های پایه. پنج سال اول یادشان گرفتم و حالا دیگر همه‌شان را به خاطر نمی‌آورم. این همان زبانی است که بیشتر از همه استفاده کرده‌ام. دلیلش این است که لازم نیست همه‌چیز را از قبل بدانی. باید همان موقع که لازم می‌شود یاد بگیری. اما وقتی شناختت مختل شده، چطور می‌خواهی همان موقع یاد بگیری؟

مخالف وایب‌کدینگ به‌عنوان شغل نیستم. اگر برای وایب‌کدینگ به تو پول می‌دهند، به‌نظر من حق داری همین کار را بکنی. اما یادت باشد عملاً به تو پول می‌دهند تا مهارت‌هایت را فراموش کنی. دیگر از شغلت چیزی یاد نمی‌گیری. شاید چیزهای کلی و سطح‌بالایی یاد بگیری، اما برای آن باید در شرکتی بسیار خوب و باتجربه روی مسئله‌های پیشرفته کار کنی؛ نه روی مسئله‌های استارتاپی به‌عنوان کارمندِ یک‌بارمصرف.

در همه‌چیز سینیور نیستی

یک نکتهٔ دیگر هم هست که می‌خواهم برجسته کنم: ارشدیت نشان افتخاری نیست که یک بار برای همیشه به خودت بچسبانی. درست است که ارشدیت در محیط‌های مختلف خیلی خوب منتقل می‌شود، اما در نهایت اگر هیچ‌وقت توسعهٔ وب نکرده‌ای، واقعاً نمی‌توانی خودت را برنامه‌نویس سینیور وب بدانی. هر زبان، فریم‌ورک و ابزاری ریزه‌کاری‌های خودش را دارد؛ هرکدام شیوهٔ خاصی برای انجام کارها، بهینه‌کردن کد و پشتیبانی از کد تمیز دارند. در حالت کلی فقط می‌توانی بگویی برنامه‌نویس سینیوری. هرچه تجربه‌ات بیشتر به حوزهٔ مشخصی ربط داشته باشد، بیشتر می‌توانی خودت را در همان حوزه سینیور بدانی. برای همین منصفانه نیست که بگوییم «فقط این‌طوری از هوش مصنوعی استفاده نکن». مخصوصاً که تمام فرض هوش مصنوعی این است که فرایند کارت را سریع‌تر می‌کند و می‌توانی با زبان‌هایی نرم‌افزار بسازی که قبلاً سراغشان نرفته‌ای. هرکسی با بیست سال تجربهٔ تمام‌وقت مهندسی نرم‌افزار و انبوهی ابزار وارد دوران مهندسی با هوش مصنوعی نمی‌شود. پس جونیورها چه؟ خب، حالا که حرفش شد...

بدون جونیورها، سرمایه‌گذاری هم نیست

یکی دیگر از جنبه‌های بسیار مهم هوش مصنوعی این است که بازار را برای جونیورها (تازه‌کارها) کاملاً خراب کرده. استخدام برنامه‌نویس جونیور تقریباً هیچ فایده‌ای ندارد، و این بهترین فرصت بوده تا بنیان‌گذاران، آیندهٔ صنعت و شرکت‌های خودشان را خراب کنند. جونیورها قرار است چطور تجربه به‌دست بیاورند وقتی یا هوش مصنوعی دستشان می‌دهند و انتظار دارند همه‌چیز را به آن واگذار کنند، یا بدتر از آن،

اصلاً استخدامشان نمی‌کنند؟ حتی برای سینیورها هم تیز نگه‌داشتن مسیر یادگیری در این دوره سخت است. جونیورها جونیور می‌مانند، مگر اینکه آن‌قدر خوش‌شانس باشند که شرکت خودشان را راه بیندازند و خودشان چیزهای بزرگی بسازند. حتی آن‌وقت هم بدون راهنما خیلی راحت در دام استفاده از هوش مصنوعی و اعتمادبه‌نفس کاذب دربارهٔ مهارت‌هایشان می‌افتند.

اعتمادبه‌نفس کاذب و سندروم ایمپاستر

دربارهٔ اعتمادبه‌نفس کاذب، یکی از رفتارهای تازه‌ای که دیده‌ام این است که مردم کمتر دربارهٔ سندروم ایمپاستر حرف می‌زنند (ممنون از @CodingJesus به‌خاطر مطرح‌کردن این موضوع). شاید این فقط تجربهٔ سوگیرانهٔ خودم باشد (برای مطمئن‌شدن آمار واقعی لازم داریم)، اما باز هم وضع را توضیح می‌دهد: همه کارشان را به هوش مصنوعی واگذار می‌کنند و همه می‌خواهند حس بهره‌وری داشته باشند، چون در غیر این صورت حس مالکیت ندارند؛ و همین حسِ مالکیت‌نداشتن خشم زیادی در آدم‌ها ایجاد می‌کند. حس نمی‌کنی کار مهمی انجام داده‌ای؛ حس می‌کنی تمام آن سال‌های تجربه و تخصص دیگر هیچ ارزشی ندارند. خنده‌دار اینجاست که سندروم ایمپاستر برای اثرکردن به تلاشی نیاز دارد؛ معمولاً تلاش قابل‌توجهی. مثلاً همین حالا که دارم این کتاب را می‌نویسم، با این میل می‌جنگم که هیچ‌وقت منتشرش نکنم تا از قضاوت‌های تند فرار کنم. اما اگر خودم کاری را انجام نداده باشم، نمی‌توانم قضاوت‌های تند را به خودم بگیرم. وقتی حس کنی هیچ تلاشی برای چیزی نکرده‌ای، کم‌کم حس سندروم ایمپاستر هم در تو از بین می‌رود و قدم می‌گذاری در چالهٔ اعتمادبه‌نفس بیش‌ازحد.

آدم‌ها باید چیزها را لمس کنند

به‌عنوان انسان، خیلی به یادگیری از راه تعامل فیزیکی با چیزها عادت کرده‌ای. کل جهان‌بینی‌ات بر پنج حس بنا شده ــ و گاهی به همان‌ها هم محدود می‌شود. نوزاد با خوردن چیزها درباره‌شان یاد می‌گیرد. مزه‌شان خوب نیست؛ فقط می‌خواهد بفهمد چه هستند، و دهان انسان اندام بسیار حساسی است. مفهوم‌ها را هم تا حدی با ربط‌دادنشان به چیزهای فیزیکی می‌فهمی. عشق را با بغل‌کردن، بوسیدن و لمس‌کردن تجربه می‌کنی. نفرت را با ترشح هورمون‌های خشم، دعوا و فریاد تجربه می‌کنی.

حتی چیزهای انتزاعی‌ای مثل خود فلسفه را هم می‌شود این‌طور توضیح داد. اول از همه، خود زبان ــ ابزاری که با آن فکر می‌کنی ــ با استفاده از چیزهای واقعی تفسیر می‌شود. دوم اینکه وقتی فهمیدن و دنبال‌کردن چیزها سخت می‌شود، چه‌کار می‌کنیم؟ با خودمان و دیگران حرف می‌زنیم، می‌نویسیمشان، سعی می‌کنیم

در زندگی واقعی شبیه‌سازی‌شان کنیم. آن‌ها را ملموس‌تر می‌کنیم.

اگر کد را لمس نکنی، هیچ‌وقت تمرین نکنی و تا خرخره در فکرهای خودت فرو بروی، واقعاً چیزی را نمی‌فهمی. نمی‌توانی انتظار داشته باشی مغزت ناگهان خودش را با برنامه‌نویسیِ «بدون دخالت دست» وفق بدهد، فقط چون فناوری دوباره دورهٔ تازه‌ای را شروع کرده. حتی مفهوم قدرتمندی مثل عشق هم وقتی همهٔ تعامل‌های فیزیکی را حذف کنی ضعیف می‌شود. هوش مصنوعی ابزاری واقعاً خیره‌کننده برای یادگیری است. اما استفادهٔ مسئولانه از آن اهمیت زیادی دارد.

زمانی برای پردازش‌کردن نداری

دیگر هیچ وقتی برای پردازش‌کردن دانشی که دریافت می‌کنی نداری. در بخش «برنامه‌نویسی قطعی بود» توضیح دادم دوم‌اسکرول‌کردن هیچ فایده‌ای برایت ندارد. زمان و تمرین کنار هم کمک می‌کنند چیزی در ذهنت بماند. اگر کسی کدی باکیفیت را برای یک لحظه نشانت بدهد، چیزی از آن یاد نمی‌گیری. حتی اگر یک بار بتوانی بخوانیش، باز هم باید بیشتر بررسی‌اش کنی و خودت دست‌به‌کد شوی. مغزت برای ذخیره‌کردن دانش به زمان و محرک نیاز دارد تا آن را چیز مهمی بداند، نه یک چیز یک‌بارمصرف. این یکی را نمی‌توانی ده‌برابر کنی.

۵.۲. دیگر جست‌وجو نمی‌کنی

یکی از چیزهایی که هم به ما فایده رسانده و هم آسیب زده این است که هوش مصنوعی تقریباً جای همهٔ انواع «دنبال جواب گشتن» را گرفته است. قبلاً وقتی با باگ و مشکل روبه‌رو می‌شدی، باید دنبال جوابش در اینترنت می‌گشتی. وقت و انرژی زیادی می‌گرفت و باید کلی اطلاعات نامربوط را زیرورو می‌کردی تا به جواب برسی.

اما حالا وقت خیلی کمتری را «هدر می‌دهی». جوابی می‌خواهی، پس از هوش مصنوعی می‌پرسی. بعضی باگ‌ها که فهمیدن و رفعشان چند هفته طول می‌کشید، حالا نهایتاً در چند دقیقه یا چند ساعت حل می‌شوند. گاهی نتیجه حتی بهتر هم هست، چون می‌توانی هرقدر خواستی از مدل زبانی بزرگ سؤال کنی تا کاملاً بفهمی.

اما یکی از اثرهای پنهانِ جست‌وجونکردن این است که آن زمانِ درمعرض اطلاعات‌بودن را از دست می‌دهی. آن «اطلاعات نامربوطی» که مدام می‌دیدی منبعی از اطلاعات بود. نمی‌توانم بشمارم چند بار برای مشکلی

جست‌وجو کرده‌ام و در موضوع کاملاً دیگری فهمیده‌ام چه اشتباهی می‌کردم. مثلاً می‌خواستم باگی را دربارهٔ شیوهٔ استفاده از یک فریم‌ورک رفع کنم، اما موقع جست‌وجو فهمیدم آن زبان یا فریم‌ورکی که استفاده می‌کنم قابلیت خوبی دارد که خبر نداشتم. داشتم راه سخت‌تر را می‌رفتم و فکر می‌کردم تنها راه همین است. هوش مصنوعی هم شاید بتواند همین کار را بکند، اما خیلی از این پیشرفت‌ها به این خاطر اتفاق افتادند که دنبال اطلاعات نامربوطی می‌رفتم که خب، بیشتر از چیزی که برای حل همان مشکل لازم داشتم یادم می‌داد.

مهم‌تر از همه، کارکردن روی آن باگ‌ها کمک می‌کرد شهود و دانشت را بسازی. وقتی باگی سخت را در دو دقیقه رفع می‌کنی، طبیعی است که چندان مهم به‌نظرت نرسد. این فقط طرز کار مغزت است. وقتی اختلاف زمانی که برای رفع باگ سخت و آسان می‌گذاری به چند ثانیه می‌رسد، مغزت دیگر نمی‌تواند آن‌ها را از هم تشخیص دهد. چون هر دو خیلی سریع رفع شده‌اند، مغزت می‌گوید: «اَه، یکی دیگه از کارهای معمولی».

باید قبول کنم هوش مصنوعی در رفع باگ و مشکل‌های دیگر دیوانه‌وار سریع‌تر است. حالا که وقت اضافه داری می‌توانی کتاب‌هایی بخوانی که قبلاً فرصتشان را نداشتی. این کار مشکلِ کمبود اطلاعات و تجربه را کمتر می‌کند. اما نکته اینجاست: چون هوش مصنوعی خیلی سریع است، انتظارها هم خیلی بیشتر شده‌اند. پژوهش‌های متعددی نشان می‌دهند مقدار کاری که برنامه‌نویس‌ها به‌خاطر هوش مصنوعی انجام می‌دهند کمتر نشده، بلکه بیشتر شده است (به این پدیده «پارادوکس جِوُنز» (Jevon's Paradox) می‌گویند، هرچند واقعاً پارادوکس نیست). اگر وقت اضافه داشته باشی، باید روی پروژهٔ بعدی بگذاری‌اش.

این ما را به نتیجه‌گیری‌ام می‌رساند:

۵.۳. مدام در حال تولیدی

همان‌طور که گفتم، با ظهور هوش مصنوعی نه‌فقط کار کمتری می‌کنی، بلکه کار بیشتری هم می‌کنی. ماهیت شغلت شده «انجام‌دادن کارها». پروژه می‌سازی، مشکل حل می‌کنی، باگ رفع می‌کنی، این و آن را پیاده‌سازی می‌کنی. وقت *یادگرفتن* نداری. قبلاً اصطکاکی که جست‌وجو ایجاد می‌کرد کمک می‌کرد چیزهایی یاد بگیری. حالا این اصطکاک از بین رفته و مدام در حال تولیدی ــ دیگر چیزی یاد نمی‌گیری. *فعلاً* برای سرمایه‌گذارها خوب است، اما بدهی فنی چند سال دیگر سر به فلک می‌کشد.

به‌تازگی (سپتامبر ۲۰۲۶) گروهی از ریاضی‌دانان برجسته ــ از جمله بیش از بیست‌وچهار برندهٔ مدال فیلدز ــ

نامه‌ای سرگشاده در mathandai.org با عنوان «ناهم‌ترازی شدید هوش مصنوعی در ریاضیات» منتشر کردند. وقتی این نامه را خواندم، کاملاً فهمیدم چه خبر است؛ تقریباً دقیقاً همان چیزی بود که اینجا توضیح می‌دهم. من ریاضی‌دان نیستم، اما حرفی که می‌خواهند بزنند ــ و به‌نظرم منطقی است ــ این است که حل‌کردن مسئله یک چیز است و یادگرفتن و رشدکردن چیز دیگر. مدام تولید می‌کنیم و فراموش کرده‌ایم دلیل اینکه الان می‌توانیم تولید کنیم این است که با گذر زمان به این آدم‌ها تبدیل شده‌ایم. سرمایه‌گذاری‌نکردن روی جونیورها، ارزش‌ندادن به فهم و رشد، و فقط به خروجی فکرکردن و چیزهای مشابه باعث می‌شوند آیندهٔ سختی در پیش داشته باشیم. شاید فناوری الان شکوفا باشد، اما وقتی همهٔ بدهی‌ها سر برسند، در آینده شاید حتی عقب هم بی‌افتیم.

۶. بدهی سه‌گانه

مقالهٔ خیلی خوبی هست از Margaret-Anne Storey به‌اسم «از بدهی فنی تا بدهی شناختی و نیت: بازاندیشی در سلامت نرم‌افزار در عصر هوش مصنوعی». شدیداً پیشنهاد می‌کنم بخوانی‌اش، چون با این کتاب هم‌جهت است و او اصطلاح‌ها و مفهوم‌ها را خیلی کامل‌تر و حرفه‌ای‌تر توضیح می‌دهد.

این مقاله سه نوع بدهی در مهندسی نرم‌افزار را توضیح می‌دهد:

> بدهیِ فنی به مشکل‌های لایهٔ کد اشاره دارد؛ بدهیِ شناختی یعنی فهم مشترک یک تیم به‌مرور فرسوده می‌شود؛ و بدهیِ نیت یعنی هدف‌ها، محدودیت‌ها و منطق تصمیم‌ها به‌شکل بیرونی ثبت نشده‌اند؛ چیزهایی که هم انسان‌ها و هم سامانه‌های هوش مصنوعی برای کارکردن امن و کارآمد با کدبیس به آن‌ها نیاز دارند. بدهی فنی تغییر سامانه‌ها را سخت‌تر می‌کند. بدهی شناختی فهمیدنشان را دشوارتر می‌کند. بدهی نیت باعث می‌شود ندانیم اصلاً سامانه برای چه کاری ساخته شده است.
>
> — Margaret-Anne Storey، «از بدهی فنی تا بدهی شناختی و نیت: بازاندیشی در سلامت نرم‌افزار در عصر هوش مصنوعی»، (۲۰۲۶) arXiv:2603.22106، با مجوز CC BY 4.0. قالب‌بندی با تغییراتی نقل شده است.

یادت باشد این سه نوع بدهی از هر نظر روی هم اثر می‌گذارند.

وقتی نکته‌های این کتاب را کنار مقاله بگذاری، کلی نکتهٔ دیگر هم پیدا می‌شود. چند نمونه:

۶.۱. بدهی شناختی و تجربهٔ انسانی

در بخش «آدم‌ها باید چیزها را لمس کنند» دیدیم که کل درکت از چیزها به تجربه‌کردنشان گره خورده است. بدهی شناختی هم از همین‌جا می‌آید. بهای انتزاع و خودکارسازی به‌طور کلی همین است. مثلاً به زبان‌های برنامه‌نویسی امروزی نگاه کن. شاید بدانی چطور با C کد بنویسی، اما Assembly بلد نباشی. انتزاع تو را از فهم صمیمیِ سازوکار سامانه دور کرده است. اگر کد اسمبلیِ تولیدشده مشکلی داشته باشد، به فنا رفته‌ای. مدت‌ها پیش آن دانش را کنار گذاشته‌ای. دیگر بدهی شناختی نیست؛ رسیده‌ای به ورشکستگی شناختی. دو دلیل دارد که دیگر نگرانش نیستیم؛ هر دو را گفتم: کامپایلر C به Assembly هم **قطعی و جبرگرا** است و هم **انسان‌ها ساخته‌اندش**. پس اعتمادمان از آدم‌های مسئولی می‌آید که آن را ساخته‌اند. این ما را می‌رساند به نکتهٔ بعدی:

۶.۲. اگر انسانی دخیل نباشد، «نسخهٔ پایدار» معنایی ندارد

می‌دانم عجیب به‌نظر می‌رسد، اما حرفم این است که «پایدار» دیگر همان معنایی را ندارد که قبلاً داشت. هوش مصنوعی تولید می‌کند، هوش مصنوعی راستی‌آزمایی می‌کند، هوش مصنوعی هم می‌گوید پایدار است. اختیاری که از دست داده‌ای ــ چیزی که بعضی‌ها فکر می‌کنند هیچ اهمیتی ندارد ــ اینجا خیلی مهم می‌شود. وقتی نرم‌افزاری می‌سازی که قرار است پایهٔ فناوری آینده شود یا مردم لازم دارند نهایت استفاده را از آن بکنند، این اهمیت پیدا می‌کند. در بخش ۶.۱ مثالی آوردم: بابت کامپایلر C به Assembly نگرانی نداری چون انسان‌ها آن را آزموده‌اند، نه یک مولد کدِ توهم‌زن. و اگر واقعاً فکر می‌کنی هوش مصنوعی این‌قدر قابل‌اعتماد است، عملاً داری می‌گویی مهندسی نرم‌افزار مرده است؛ که باز ما را برمی‌گرداند به استدلال آخرالزمان.

۶.۳. آزمون‌ها و بدهی فنی

قبلاً چرا آزمون (test) می‌نوشتیم؟ برای اینکه همان خط کدی را که تازه نوشته بودی آزمایش کنیم؟ یعنی واقعاً یک تکه کد می‌نوشتی و بعد، پیش از اجرای آن، چند آزمون می‌نوشتی و اجرا می‌کردی تا مطمئن شوی درست کار می‌کنند؟ شاید حتی نفهمی منظورم چیست، چون خیلی عجیب به‌نظر می‌رسد. اما حالا داریم از آزمون‌ها همین‌طور استفاده می‌کنیم.

برای آزمون‌نوشتن دلیل‌های زیادی داشتیم، اما فلسفهٔ مرکزی‌شان این بود که مطمئن شویم چیزی که **از قبل کار می‌کند** همچنان کار خواهد کرد. اصلاً چرا فرض می‌کردیم «از قبل کار می‌کرد»؟ چون به انسانی که

کد را نوشته بود اعتماد داشتیم: وقتی خودت کد را می‌نویسی، اجرا می‌کنی و می‌بینی کار می‌کند، طبیعتاً به آن اعتماد می‌کنی. اگر کس دیگری کد را نوشته باشد و به او اعتماد داشته باشی هم همین است. اگر چون هوش مصنوعی گفته به کد اعتماد کنی، اعتماد کردی، داری به منبعی غیرقابل‌اعتماد از حقیقت تکیه می‌کنی؛ منبعی که خودش حلقه‌ای از اشتباه‌هاست (حلقه ی استدلالی). وقتی خودت دستی کارها را انجام می‌دهی، معمولاً چند بار در میان کار برنامه را اجرا می‌کنی و کم‌کم مطمئن می‌شوی همه‌چیز درست است؛ نه اینکه اول ۷۰۰ خط کد بنویسی و بعد. این روند در خودت اعتماد می‌سازد.

حالا سامانهٔ راستی‌آزمایی مختل شده است. انسانی نیست که به او اعتماد کنی، بدهی شناختی و بدهی نیت دارند رشد می‌کنند، پس برای اینکه دست‌کم بدهی فنی را از دست ندهیم، آزمون‌ها را راه اولِ اطمینان از کارکردن چیزی کرده‌ایم. اما می‌دانیم آزمون‌ها هم کافی نیستند، پس تضمین کیفیت (QA) به نقشی بسیار مهم تبدیل شده است. یک نفر یک بار این‌طور گفت (یادم نیست چه کسی): مردم دیگر دنبال راهنمایی برنامه‌نویسی نیستند؛ دنبال بازخورد هستند و همان را به هوش مصنوعی پاس می‌دهند با دستورِ «همه‌چیز را درست کن، اشتباه نکن».

۶.۴. مقصرِ بدهی‌های پنهان می‌شوی

مدیر می‌خواهد هرچه زودتر منتشر کنی. برای این کار باید خیلی چیزها را قربانی کنی و همهٔ این بدهی‌ها را بپذیری. ماهیتشان هم طوری است که پنهان می‌مانند. تو می‌گویی: «چطور از چشمم دور ماند؟» و مدیر می‌گوید «اخراجی».

۶.۵. تناقض مشخصات

درک شناختی‌ات از پروژه هنگام توسعهٔ آن رشد می‌کند. درکت از نیت و هدف پروژه هم همین‌طور، اما به‌اندازهٔ کمتری. هر دو تحت‌تأثیر درک فنی‌ات هستند. یعنی درکت از اینکه پروژه چیست و قرار است به چه چیزی تبدیل شود، درواقع به خود پروژه گره خورده است. بااین‌حال، انتظار دارند پیش از شروع پروژه مشخصات بی‌نقصی ارائه کنی. غیرممکن است؛ پس پروژه نگه‌داری‌ناپذیر می‌شود و تقصیرش را گردن تو می‌اندازند.

۶.۶. محدودیت‌های شناختی به بدهی ختم می‌شوند

همان‌طور که در بخش «محدودیت‌های شناختی انسان» توضیح دادم، خیلی از محدودیت‌هایی که در

توسعه با آن‌ها روبه‌رو می‌شوی از محدودیت‌های مغز و زبان انسان می‌آیند. قبلاً بدهی شناختی مطرح نبود ــ یا آن‌قدر مهم نبود که اسمش را بیاوریم ــ چون حالا فرایند را به هوش مصنوعی واگذار می‌کنیم و انتظار داریم درحالی‌که از نظر فیزیکی محدودیم، هم‌پای آن پیش برویم.

۷. عذاب وجدان؟

مت پاکوک (Matt Pocock)، چهره‌ای بسیار مشهور در جامعهٔ وایب‌کدینگ، توییتی از ۱۴ سپتامبر دارد که هنگام نوشتن این متن آن را در حسابش سنجاق (pin) کرده است. در آن توضیح می‌دهد کسی دوره‌هایش را گذرانده، در یک شرکت به متخصص هوش مصنوعی تبدیل شده و حالا بابت گرفتن شغل او احساس گناه می‌کند (Matt هم در ادامه گفته که درواقع به او افتخار می‌کند و از این حرف‌ها). نمی‌دانیم این گفت‌وگویی واقعی بوده یا فقط توییتی برای تبلیغ خودش، اما برای همه روشن است که چنین چیزی کاملاً ممکن است. مانع ورود آن‌قدر پایین آمده که حس می‌کنی هیچ کاری نکرده‌ای، و اگر اعتمادبه‌نفست کاذب نباشد، احساس می‌کنی ایمپاستری.

این موضوع درواقع خنده‌دار است. مثلاً آن را با هنر دیجیتال مقایسه کن: قبلاً هنرمندان دیجیتال را به‌خاطر استفاده از ابزارهای دیجیتال به‌جای کار با دست، تنبل می‌دانستند. آن‌ها هیچ‌وقت بابت اینکه ظاهراً «هیچ کاری نمی‌کنند و به همه‌چیز می‌رسند» احساس گناه نداشتند. اگر هنر دیجیتال کار کرده باشی، می‌دانی که به تلاش زیادی نیاز دارد. اگر نقاش سنتی بوده‌ای و به هنر دیجیتال روی آورده‌ای، می‌دانی مجموعه‌مهارت‌هایت به‌راحتی به مدیوم تازه منتقل می‌شود و اگر از قبل آن مهارت‌ها را نداشته باشی، هنرمند دیجیتال خوبی نمی‌شوی. اما حالا همه احساس می‌کنند کدنویسی و برنامه‌نویسی حل شده و فقط باید چند توکن بخری و بسوزانی. حس بهره‌وری نداری و احساس می‌کنی بلافاصله به سطحِ به‌اصطلاح «استادت» رسیده‌ای.

و اگر منصف باشیم، این فقط مسئله‌ای شخصی و موقت هم نیست. اگر مهارت‌های مت را در GitHubش ببینی، متوجه می‌شوی که این‌ها چیزهایی نیستند که یک غیربرنامه‌نویس بفهمد. نمی‌دانی TDD چیست، مشخصات (spec) چیست، بازبینی کد واقعاً یعنی چه و کلی مفهوم دیگر که فقط با برنامه‌نویس‌بودن می‌شناسی‌شان؛ درنتیجه خودت هم نمی‌توانی آن‌ها را بسازی. اما برای کارکردن به آن‌ها نیاز داری. هارنسی که استفاده می‌کنی هم همین‌طور است. داخل Claude پرامپت‌های سیستمی بزرگی وجود دارد که به آن یاد می‌دهند برنامه‌نویس و ایجنت خوبی باشد؛ خیلی‌هایشان را حتی نمی‌دانی وجود دارند.

احساس گناه می‌کنی چون اصلاً روی آن‌ها اختیار نداری؛ داری از دانشی استفاده می‌کنی که دیگران طی سال‌ها و حتی دهه‌ها پرورانده‌اند تا چیزی بنویسی. اگر همهٔ پرامپت‌های سیستمی و skillها را حذف کنی، تمام آن توییت‌های «کدنویسی حل شده» ناپدید می‌شوند (دست‌کم با توجه به وضعیت فعلی هوش مصنوعی). همین احساس گناه به‌تنهایی شاخص بسیار خوبی است که نشان می‌دهد هوش مصنوعی (یا دقیق‌تر، هیاهوی هوش مصنوعی) صنعت را به‌سمتی برده که در بخش «نرم‌افزار می‌میرد» توضیح دادم. این هزینه‌ای است که وقتی بازار تا این حد دموکراتیک می‌شود می‌پردازی.

جهنم وابستگی‌ها

یکی از چیزهایی که در این سال‌های برنامه‌نویسی یاد گرفته‌ام این است که نباید وابستگی‌ها را دست‌کم بگیری. هر وابستگی می‌تواند راه تازه‌ای برای آسیب‌زدن به خودت و محصولت باشد. قبلاً هم یک پست وبلاگی دربارهٔ اینکه یک وابستگی چطور یکی از پرکاربردترین قابلیت‌های تلگرام را از کار انداخت نوشته‌ام.

وابستگی در نرم‌افزار با وابستگی در دنیای واقعی فرقی ندارد. فرض کن توی باغچه‌ات یک دسته پول پیدا می‌کنی. با خودت می‌گویی: «این پول‌ها دیگر از کجا آمده‌اند؟» جوابی پیدا نمی‌کنی جز اینکه «باد آورده‌شان اینجا». اگر مذهبی باشی، شاید فکر کنی خدا انداخته‌شان. پس پول را برمی‌داری و خرج می‌کنی. برای یک هفته کافی است.

هفتهٔ بعد دوباره چیزی شبیهش پیدا می‌کنی. از اینکه چرا باز این اتفاق افتاده تعجب می‌کنی، اما پول را برمی‌داری و خرجش می‌کنی.

این ماجرا مدام تکرار می‌شود. یک سال می‌گذرد. با خودت می‌گویی: «وقتی هر هفته پول مفت می‌ریزد توی باغچه‌ام، چرا دارم وقتم را در این شرکت هدر می‌دهم؟» پس از کارت استعفا می‌دهی، بی‌آنکه بدانی هفتهٔ بعد دیگر این اتفاق نمی‌افتد. قبلاً نمی‌دانستی چرا این پول پیدا می‌شود؛ حالا هم نمی‌دانی چرا دیگر پیدا نمی‌شود. به آن پول وابسته شده بودی و حالا که نیست، به فنا رفته‌ای.

وابستگی‌های دیگر هم همین‌اند. انرژی و حالت را به خودشان وابسته می‌کنند. اگر یک روز سیگار نکشی، یا حتی چند ساعت، حس بدی داری. حالا زندگی روزمره‌ات به آن سیگار وابسته است و حتی برای ازدست‌ندادنش از چیزهای ضروری می‌گذری؛ مثلاً سلامتت.

ما برنامه‌نویس‌های سینیور این شهود را در خودمان پرورش داده‌ایم تا بدانیم نباید بی‌فکر وابستگی تازه‌ای به پروژه اضافه کنیم. همیشه باید حساب‌شده به آن‌ها نگاه کنیم و تا جای ممکن تعدادشان را کم نگه داریم، چون نمی‌دانیم آینده چه می‌شود. چه کسی فکرش را می‌کرد خودِ Google یک روز تصمیم بگیرد دیگر از API (رابط برنامه‌نویسی کاربردی) گیف Tenor پشتیبانی نکند؟

و حالا هوش مصنوعی بزرگ‌ترین وابستگیِ همهٔ نرم‌افزارها و برنامه‌نویس‌ها شده است. شغلت به یک یا چند شرکتی وابسته است که هر کاری دلشان بخواهد می‌توانند بکنند. اگر یک روز، به هر دلیلی ــ حتی سیاسی ــ مدل هوش مصنوعی‌ات از دسترس خارج شود، کل شرکتت فرو می‌پاشد. یک باگ ساده کافی است تا تو را به اعماق جهنم وابستگی‌ها بکشاند. شاید الان مشکل به‌نظر نرسد؛ هنوز حس امنیت می‌کنی چون خیلی چیزها یادت هست. اما وقتی چندین سال کدنویسی نکرده باشی و برنامه‌نویس سینیور کم باشد، برای اینکه مدل‌های هوش مصنوعی‌ات را از دست ندهی از هرچیزی می‌گذری.

توکن‌ها و جهنم وابستگی

جنبهٔ مهم دیگر این است که هوش مصنوعی دست‌کم در حال حاضر بی‌نقص نیست و تا مدت خوبی هم بی‌نقص نخواهد شد. هر بار مدل تازه‌ای عرضه می‌شود، همه با آن پروژه می‌سازند و می‌گویند کل پروژه را «تک‌ضرب» ساخته‌اند: فقط یک پرامپت داده‌اند و هوش مصنوعی بقیهٔ کارها را انجام داده است. منظورشان از این تک‌ضرب این نیست که هوش مصنوعی در یک مرحله کار را انجام داده؛ یعنی کاربر فقط یک پرامپت استفاده کرده است. به هوش مصنوعی چیزی به اسم «هدف» می‌دهند، بعد هوش مصنوعی برای خودش برنامه می‌ریزد، یک سامانهٔ راستی‌آزمایی طراحی می‌کند و مدام سعی می‌کند برنامه را بسازد تا ظاهراً به هدف برسد. کد تولید می‌کند، آن را اجرا می‌کند، با خطا روبه‌رو می‌شود، کد را ویرایش می‌کند، کد بیشتری تولید می‌کند و ادامه می‌دهد. اما نکته همین‌جاست: **هنوز هم خطا می‌کند**. این یعنی بی‌نقص نیست. حالا اگر گیر کند چه؟

فرض کن یک پلتفرم کامل را وایب‌کد کرده‌ای و حالا با ده‌ها، صدها یا هزاران کاربر در حال کار است. اگر هوش مصنوعی شکست بخورد، باید دوباره وارد فرایند آزمون‌وخطا شوی: به آن بازخورد بدهی، بخواهی «مشکل را درست کند» و صبر کنی تا همه‌چیز را درست کند. اما گاهی در حلقه‌ای گیر می‌کند، راهی حقه‌بازانه و بد انتخاب می‌کند و با هر نوبت گیج‌تر می‌شود. باید مدت زیادی بالای سرش بمانی و مطمئن شوی کار عجیبی نمی‌کند (تنها چیزی که می‌توانی بفهمی همین است). شاید توکن‌هایت تمام شوند. شاید کار مهمی داشته باشی؛ مثلاً قابلیت تازه‌ای که فردا یا پس‌فردا موعدش می‌رسد، اما به سقف هفتگی

مصرفت نزدیک شده‌ای و این باگ هم مهم است. پس با خودت می‌گویی: «خب، اینجا دیگر باید هوش مصنوعی را کنار بگذارم، آستین بالا بزنم و خودم درستش کنم».

اما مسئله این است که خودت در پروژه دخیل نبوده‌ای. مثل روز اولی است که یک کدبیس بزرگ را تحویلت داده‌اند. نمی‌دانی هیچ‌چیز کجاست و رفع مشکل برایت حتی بیشتر طول می‌کشد؛ پس دوباره سراغ هوش مصنوعی می‌روی و التماسش می‌کنی مشکل را درست کند. یادت باشد چون پای یک باگ در میان است، نمی‌توانی واقعاً بازخورد درستی بدهی. خودت شاید اصلاً ندانی چرا اتفاق افتاده؛ تنها کاری که می‌توانی بکنی این است که برای هوش مصنوعی توضیح بدهی چرا و چه زمانی رخ می‌دهد. همین توضیح گاهی تلاش بسیار زیادی می‌خواهد. برای تبدیل‌کردن تمام چیزهایی که می‌بینی به کلمات، باید **خیلی** باسواد باشی.

بعضی‌ها فکر می‌کنند راه‌حل این مشکل بازبینی کد است. اما بازبینی کد جواب مسئله نیست و در بخش بعدی توضیح داده‌ام چرا.

و همهٔ این‌ها به‌خاطر وابسته‌بودن توست. هوش مصنوعی کارمند توست، نه ابزارت. ابزار هیچ‌وقت تو را این‌طور وابسته نمی‌کند. تازه این کارمند واقعاً غیرقابل‌اعتماد است. حتی همین حالا که این متن را می‌نویسم، Codex بدون هیچ هشدار قبلی از کار افتاده است. دیشب هم دقیقاً همین‌طور از کار افتاد. جالب نیست؟ داری با شغلت رولت روسی بازی می‌کنی.

بازبینی کد جواب مسئله نیست

دربارهٔ شیوهٔ درست استفاده از هوش مصنوعی به‌عنوان مهندس نرم‌افزار، طیف گسترده‌ای از نظرهای خاکستری وجود دارد: از آدم‌هایی که فکر می‌کنند باید تمام کدنویسی و بازبینی را به هوش مصنوعی واگذار کنی و فقط کارهای کلی‌ای مثل طراحی معماری و مدیریت تیم را انجام بدهی، تا کسانی که فکر می‌کنند باید فقط در حداقلِ ممکن از هوش مصنوعی استفاده کنی.

اگر نمی‌دانی، بازبینی کد یعنی خواندن کدی که پیاده‌سازی یا ویرایش شده تا مطمئن شوی از عرف‌ها پیروی می‌کند، باگ ندارد و هرچه لازم داریم در آن هست. پیش از هوش مصنوعی، برنامه‌نویس سینیور این کار را می‌کرد تا مطمئن شود برنامه‌نویس‌های جونیور کد معتبر را وارد کدبیس می‌کنند. فرصت خوبی هم بود تا راهنمایی‌شان کند و یادشان بدهد کد خوب فقط این نیست که کار کند.

هنوز جواب پذیرفته‌شده‌ای نداریم که باید کد تولیدشده با هوش مصنوعی را بازبینی کنی یا نـه. مـردم حـتی هفته‌به‌هفته نظرشان را عوض می‌کنند. به‌نظرم چند سال طول می‌کشـد تا تکلیفش روشـن شـود. یـا شـاید فقط چند ماه.

چند نکته هست که باید توضیح بدهم تا روشن شود چرا بازبینی کد جواب مسـئله نیسـت؛ امـا بـاز هم لازم است بگویم که این دلیل‌ها به هم مربوط‌اند. فقط برای اینکه دنبال‌کردن و فهمیدنشان آسـان‌تر باشـد از هم جداشان کرده‌ام.

مرحله‌های هم‌پوشان

کدنویسی سه مرحله دارد: پیاده‌سازی، رفع باگ و بازبینی. این مرحله‌ها به‌هیچ‌وجه از هم جدا نیستند. حتی وقتی قابلیت تازه‌ای اضافه می‌کنی، مدام کدی را که تازه نوشته‌ای بازبینی می‌کنی و باگ‌ها را هم رفع می‌کنی (ساده‌ترین مثالش این است که غلط‌های تایپی را همان موقع اصلاح می‌کـنی). این‌طـور نیسـت کـه بیسـت اسکریپت طولانی بنویسی و بعـد تازه سـراغ رفـع بـاگ یـا بـازبینی بـروی. همـهٔ این کارهـا را هم‌زمـان انجـام می‌دهی و در پایان می‌توانی مرحله‌های جـداگانه‌ای هم بـرای رفـع بـاگ و بـازبینی کـد داشـته باشـی. بیشـتر وقت‌ها حتی وقـتی داری کـد را رفـع بـاگ یـا بـازآرایی می‌کـنی، ایـدهٔ تازه‌ای بـرای افـزودن قـابلیت بـه ذهنت می‌رسد. گاهی همین قابلیت‌های تازه نقش مهمی در شیوهٔ رفع باگ یا بازآرایی دارند.

مردم با «بازبینی کد» طوری رفتار می‌کنند که انگار فعالیتی است که در مقایسه با خود کدنویسی بـه خلاقیت و قدرت ذهنی بسیار بیشتری نیاز دارد. درست است کـه کـد تولیدشـده بـا هـوش مصـنوعی هنگـام بـازبینی کمبودها و نقص‌هایی دارد، اما باید بفهمی این استدلال موقتی و مبتنی بر وضـعیت فعلی هـوش مصـنوعی است. اگر هوش مصنوعی بتواند کاملاً جای برنامه‌نویس را بگیرد و فقط بازبینی کـد بـرای تـو بمانـد، یـک روز همان شغل را هم می‌گیرد. همین حالا هم بخش بزرگی از بازبینی، بازآرایی و رفع باگ کـد را خـودش انجـام می‌دهد.

یعنی فکرش را بکن، کدام بخش بازبینی کد به‌نظرت عجیب است؟ فقـط خوانـدن و وصـل‌کردن نقطه‌هـا بـه هم است. حدس بزن چه؟ هوش مصنوعی این کار را خیلی بهتر انجام می‌دهد. در ویـدئوی Adam Bender که گفتم، بخشی هست که دربارهٔ فهم بهتر تصویر کلی از سـوی هـوش مصـنوعی حـرف می‌زنـد. از حاضـران می‌پرسد: می‌توانی نمودار و گراف دانش بی‌نقصی از کدبیس شرکتت بنویسـی؟ همکـارانت می‌تواننـد؟ شـرط می‌بندم هیچ‌کدام نمی‌توانید، چون اطلاعات زیادی هست که باید به خاطر بسـپارید. ماشـین‌ها سال‌هاسـت

در این زمینه از ما جلو افتاده‌اند. حتی مدل‌های قدیمی‌تر هوش مصنوعی هم در پیداکردن ارتباط‌ها در کدبیس‌های خیلی بزرگ بد نبودند.

یک بار توانستم در موتور بازی‌سازی Godot مشارکت کنم ــ آن هم با دست ــ بی‌آنکه از قبل C++ بلد باشم یا دانش سطح‌پایین چندانی داشته باشم. هوش مصنوعی کمکم کرد چند نقطه را به هم وصل کنم که در چنین کدبیس بزرگی، در آن مدت کوتاه، عملاً پیداکردنشان برایم غیرممکن بود.

شاید بگویی طراحی سامانه و معماری هنوز چیزی است که هوش مصنوعی بلد نیست و نمی‌تواند جایگزینش شود، پس همچنان به مهندس نرم‌افزار نیاز داریم. اما باید دلیلش را بفهمی. اگر فقط وضعیت فعلی هوش مصنوعی مانع است، همین را بگو و حواست به آن باشد. اگر فکر می‌کنی عملاً محال است هوش مصنوعی این کارها را انجام دهد، باید دوباره فکر کنی چرا توانست کدنویسی و بازبینی کد را انجام دهد. اگر طراحی معماری هم کاری مکانیکی باشد چه؟ اگر با پیشرفتش بتواند آن را هم به همان خوبی انجام دهد چه؟ اگر جوابی برای این سؤال ندانی، شاید با مدل‌های آینده و توییت‌های بی‌پایانی مثل «طراحی سامانه حل شد» غافلگیر شوی.

به‌خاطر هم‌پوشانی این مرحله‌ها، واقعاً نمی‌توانی یکی را بدون بقیه حذف کنی. مثل این است که از آشپز بخواهی آشپزی را متوقف کند (چون یک هوش مصنوعی آن را خودکار کرده) و فقط دربارهٔ نتیجه بازخورد بدهد تا هوش مصنوعی خودش آن را اصلاح کند. بخش بزرگی از کار با انجام‌دادنش تعریف می‌شود. قضاوت‌کردن با این فرق دارد؛ اینجا صحبت از تضمین کیفیت است. قاضی فقط بازخوردی راهگشا می‌دهد و می‌گذارد آشپز هر کاری می‌خواهد با حرفه و پروژه‌هایش بکند.

اگر هوش مصنوعی بتواند آشپزی را بی‌نقص انجام دهد، احتمالاً راستی‌آزمایی را هم انجام خواهد داد؛ اگر نه حالا، چند ماه یا چند سال دیگر.

از دست دادن درک شناختی

هرچه ارتباطت با کد کمتر شود، درکت از پروژه هم کمتر می‌شود. برنامه‌نویس‌های سینیور بازبینی کد می‌کردند، اما خودشان کدنویسی هم می‌کردند. ارتباط عمیقی با بخش فنی کار داشتند. این موضوع نه‌فقط با ایجاد حس رضایت و مالکیت کمک می‌کرد به حرکت رو به جلو ادامه دهند، بلکه نمی‌گذاشت تنبل شوند و تصویر پروژه را از دست بدهند.

همان‌طور که گفتم، مغزت برای پردازش به زمان نیاز دارد. آن «اصطکاک»ی که در گردش‌کارت وجود داشت عامل مهمی بود که کمک می‌کرد چیزهایی را که تازه یاد گرفته‌ای فراموش نکنی. اگر تمام شغلت بازبینی باشد، خیلی راحت جزئیات ظریف پروژه را از دست می‌دهی. حتی اگر تک‌تک خط‌های کد را بخوانی و زمان زیادی برای بازبینی بگذاری، باز هم خیلی زود فراموششان می‌کنی. چند هفته کافی است تا کاملاً یادت برود پروژه چه بود و چطور باید کدش را بازبینی کنی. فهم مشترک و تصویر کلی به‌راحتی شکل نمی‌گیرند.

پرهزینه برای منابع

بازبینی کد، بسته به میزان تمرکز، سه سطح دارد:

1. **مرور سریع**

سطح اول بیشتر مرور سریع است. فقط بخشی از کد را بازبینی می‌کنی؛ حتی شاید اصلاً کد را نخوانی و به توضیحی که توسعه‌دهنده دربارهٔ تغییرات و کد داده تکیه کنی. نظرها و مستندات هم هستند که در اصل متن ساده‌ای داخل فایل‌های کدند و مرور را سریع‌تر می‌کنند. نگه‌دارنده‌های اصلی پروژه که ده‌ها درخواست ادغام تازه دریافت می‌کنند معمولاً همین‌طورند، چون هیچ راهی ندارند حتی بخش کوچکی از آن کد را بفهمند. هرچه از جزئیات فنی دورتر شوی، داشتن اختیار و صلاحیت نسبت به آن‌ها سخت‌تر می‌شود. مثلاً لاینس توروالدز (Linus Torvalds) صریحاً گفته که جز در موارد بسیار نادر کد را نمی‌خواند: «کار من کارکردن با آدم‌هاست».

این سطح کمک می‌کند به‌طور کلی در مسیر حرکت برنامه نظر داشته باشی. خیلی راحت ممکن است باگ‌های زیادی را نبینی، اما این مهم نیست چون به کارمندانت اعتماد داری که بی‌سروصدا به تو خیانت نکنند، دروغ نگویند، توهم نزنند یا کارهایی از این دست نکنند. آن‌ها **ثابت کرده‌اند** برنامه‌نویس‌های خوبی هستند؛ اما اعتماد از رابطهٔ انسانی می‌آید، همان‌طور که لاینس گفته به آن‌ها اعتماد دارد چون بیش از بیست سال با هم کار کرده‌اند. آن‌ها برایش اعتبار دارند.

2. **معقول**

سطح دوم رایج‌ترین و معقول‌ترین نوع بازبینی کد است. توسعه‌دهنده‌ای هستی که با پروژه در ارتباط است؛ کد را بازبینی می‌کنی و از جنبه‌های مختلف بازخورد می‌دهی.

در این سطح، ممکن است هنوز بعضی مشکل‌ها را نبینی؛ مثل مشکل‌های قرارداد نام‌گذاری، غلط‌های تایپی یا استفادهٔ نادرست از ابزارها. برنامه‌نویس ممکن است بعضی بهترین‌روش‌ها را نادیده گرفته باشد؛ مثلاً اصل *خودت را تکرار نکن* (DRY یا Don't Repeat Yourself). فقط بخشی از مشکل‌ها را پیدا می‌کنی، نه آن‌هایی را که در چند دقیقه سخت‌تر تشخیص داده می‌شوند. شاید بیشتر کد را بخوانی، اما عمیق نمی‌شوی، چون باز هم به برنامه‌نویس اعتماد داری.

3. **پارانوئید**

سطح سوم شبیه کار یک آدم پارانوئید است. خط‌به‌خط بازبینی می‌کنی، توضیح زیادی می‌خواهی، و اگر نتوانی از خود کد بفهمی چرا برنامه‌نویس فلان روش را انتخاب کرده، از او می‌پرسی و الی آخر.

دلایل مختلفی برای چنین بازبینی‌ای وجود دارد. شاید داری با کسی مصاحبه می‌کنی، شاید پروژه‌ای عظیم و حساس است، یا شاید می‌خواهی از آن شخص ایراد بگیری و روی جزئیات موشکافی کنی.

از میان این سه سطح، این سطح است که واقعاً تو را مسئول می‌کند. دقیقاً شبیه کدنویسی است، با این تفاوت که به‌جای نوشتن کد، کل کد را طوری می‌خوانی و می‌فهمی که انگار خودت نوشته‌ای. این‌طوری می‌توانی با اطمینان مسئولیتش را بپذیری؛ مثلاً وقتی مدیر بالادستی از تو می‌پرسد چرا اجازه دادی این کد وارد پروژه شود.

خیلی از برنامه‌نویس‌ها ــ از آدم‌هایی به برجستگی لاینس تورووالدز گرفته تا سازندهٔ زبان زیگ (Zig) ــ می‌گویند با حجم کد و تعداد اصلاح باگ‌هایی که سرشان ریخته می‌شود مشکل دارند. از هر برنامه‌نویسی در محیط کاری‌ای که به بازبینی کد احترام می‌گذارد بپرسی، می‌گوید حجم کد خودش مشکل‌ساز شده و مجبورشان کرده بعضی اصلاح باگ‌ها و کارهای مشابه را کنار بگذارند تا به کارهای مهم‌تر برسند.

صاحب زیگ حتی درخواست‌های ادغامِ وایب‌کدشده را ممنوع کرده و گفته با تیمی فقط پنج‌نفره، بازبینی کد وقت زیادی از آن‌ها می‌گیرد و باعث می‌شود از برنامه عقب بیفتند. اما موضوع فقط زمان نیست. دو مسئلهٔ دیگر هم در این زمینه نقش بزرگی دارند که به‌نظر من جنبه‌های نادیده‌گرفته‌شدهٔ بازبینی کدند.

کمبود مسئولیت‌پذیری

کسی در پروژه‌ات مشارکت می‌کند. تو، به‌عنوان صاحب پروژه‌ای مثل زیگ، وقت ارزشمندت را برای بازبینی

کد می‌گذاری. انتخاب‌های عجیب و تصمیم‌های ظاهراً بدی پیدا می‌کنی که در نگاه اول متوجهشان نمی‌شوی. باید واقعاً دقت کنی تا پیدایشان کنی، چون **کد کار می‌کند**. از مشارکت‌کننده می‌پرسی: «چرا این را این‌طوری پیاده‌سازی کردی؟» یا نادیده‌ات می‌گیرد یا سؤالت را مستقیماً برای هوش مصنوعی می‌فرستد و جواب آن را برایت می‌فرستد. این کار نه‌فقط واقعاً بی‌ادبانه است، بلکه نشان‌دهندهٔ بی‌مسئولیتی هم هست.

وایب‌کدر چیزی نمی‌داند و چیزی هم یاد نگرفته است. تمام کاری که می‌کرده تولیدکردن بوده؛ فقط قابلیت اضافه می‌کرده است. «اگر کار می‌کند، پس کار می‌کند». با رفتارکردن مثل اپراتورِ صرفِ ایجنتِ هوش مصنوعی، به‌جای مشارکت‌کنندهٔ واقعی، مسئولیت را گردن بازبین می‌اندازد.

بازبینی کد به‌عنوان سرمایه‌گذاری

زیگ فلسفه‌ای دارد که بر اساس آن به کسانی که در پروژه مشارکت می‌کنند آموزش می‌دهد، کمکشان می‌کند رشد کنند و مشارکت‌کننده‌های بهتری شوند. این یک سرمایه‌گذاری است. هرچه توسعه‌دهنده دربارهٔ زیگ بیشتر بداند، مشارکت‌هایش بهتر و هم‌راستاتر می‌شوند و زمان کمتری برای بازبینی کدش لازم است. با گذشت زمان حتی ممکن است به‌عنوان پیمانکار استخدام شوند. به‌نظر من این فلسفه نه‌فقط برای پروژه‌های متن‌باز بزرگ بسیار خوب و ضروری است، بلکه پروژه‌های دیگر هم مستقیم یا غیرمستقیم آن را پذیرفته‌اند. اما بی‌مسئولیتی این فلسفه را بی‌معنی و غیرممکن می‌کند.

گاهی ممنوع‌کردنش آسان‌تر است

می‌توانی از مشارکت‌کننده‌ها بخواهی فقط کد خوبِ تولیدشده با هوش مصنوعی ارائه دهند. اما آن‌وقت تشخیص معنای کد خوب و بد به وظیفهٔ تازه‌ای برای توسعه‌دهنده‌ها تبدیل می‌شود و کار را از نظر منابع پرهزینه‌تر می‌کند.

مفیدبودن هوش مصنوعی به این معنی نیست که همیشه قابل‌مدیریت است. هرچیزی بده‌بستان‌های خودش را دارد. ماهیت هوش مصنوعی هم معایب خودش را دارد و همین می‌تواند به گلوگاه تبدیل شود.

چرا بازبینی زمان می‌برد

در بخش «کمبود مسئولیت‌پذیری» توضیح دادم که کارکردن کد به این معنی نیست که از استانداردها

پیروی می‌کند، مقیاس‌پذیر و نگه‌داری‌پذیر است یا ویژگی‌های مشابه را دارد. و چون نمی‌توانی بـه آن اعتمـاد کنی، باید از سطح سوم بازبینی استفاده کنی: خط‌به‌خط، انگار که حتمـاً مشـکلی وجـود دارد. وگرنـه درکت از چیزی که پیاده‌سازی شده محدود می‌ماند.

«چرا این‌قدر برایت مهم است؟»

درست است که بیشتر وقت‌ها، اگر نگوییم همیشه، محصـولی نـاقص توسـعه می‌دهی. اشـکالی هم نـدارد. گسترش تدریجی دامنهٔ پروژه از نظر مالی به‌صرفه نیست. بیشتر وقت‌ها ضـرب‌الاجل‌های فشـرده داری. کـل صنعت بر پایهٔ همین نقص‌ها ساخته شده و مردم محصولاتی را منتشر می‌کنند که به‌انـدازهٔ کـافی برایشـان وقت نگذاشته‌اند. مگر اینکه بابتش پول بگـیری، سـاختن محصـولی بـا پایـداری کمـتر از حـد مطلـوب از نظـر اخلاقی یا حرفه‌ای *غلط* نیست. اگر محصولت چند مشکل داشته باشـد دنیـا بـه آخـر نمی‌رسـد. پول دربیـاور؛ بعداً می‌توانی کاملش کنی.

اما وقتی پای هوش مصنوعی در میان است، این طرز فکر نقص‌های انسانی را با نقص‌های هـوش مصـنوعی مقایسه می‌کند. هرچند بعداً در بخش «قبلاً کد را کپی‌پیست می‌کردیم» دربـاره‌اش حـرف زده‌ام، ارزش دارد همین‌جا هم موقعیت را توضیح بدهم و کمی کلی‌تر نگاه کنم.

اول اینکه نقص‌های هوش مصنوعی بسیار تصادفی‌ترند. برای همین بـه آن‌هـا تـوهم‌زایی می‌گـوییم. تِـرِنس تائو (Terence Tao) در یکی از ویدئوهایش گفته هوش مصنوعی مثل آدمی بسـیار مطلـع امـا کمی مسـت است و به‌نظر من این یکی از بهترین توصیف‌هاست. هوش مصنوعی توهم می‌زنـد و ممکن اسـت هرجـایی اشتباه کند؛ انسان این‌طور نیست.

دوم اینکه وقتی انسان می‌فهمد چه چیزی اشتباه بوده، از آن خیلی چیزها یاد می‌گیرد. نه‌فقط یاد می‌گیرد از همان مشکل مشخص دوری کند، بلکه الگوهای زیادی هم پیرامونش می‌سـازد. بـرای کامل‌شـدن مـاجرا، در نظام باور خودش هم الگوهایی پیـدا می‌کنـد، می‌فهمـد چـرا آن روز بـا آن مشـکل روبه‌رو شـده و تغییرهـای بلندمدتی در نگرشش ایجاد می‌کند. قدرتمندبودن به این شکل منابع زیادی می‌خواهد و مغز (و بدن) انسـان برای همین کار بهینه شده است.

سوم اینکه شیطان در جزئیات است. نرم‌افزار، مثل بسیاری از محصولات دیگـر، لایه‌به‌لایـه نوشـته می‌شـود. مشکل‌ها روی هم جمع می‌شوند و بدهی‌ها انباشته می‌شوند. «خشت اول گر نهد معمار کج، تا ثریـا مـیرود

دیوار کج». یک مشکل ساده ممکن است در درازمدت به شکل‌هایی غیرقابل‌تصور به تو آسیب بزند. برنامه‌نویسی همیشه همین بوده است: تغییرها و مشکل‌های کوچکی که اگر رسیدگی نشوند، نگه‌داری پروژه را وحشتناک می‌کنند. نکته در جزئیات است. می‌توانی چیزی را به وجود بیاوری و بعد کاملش کنی؛ اما ممکن است هنگام به‌وجودآوردنش آینده‌اش را خراب کرده باشی. دلیل اصلی تمایز قائل‌شدن بین جونیور و سینیور همین است. سینیور فقط جونیوری با پشتکار بیشتر نیست.

انتشار با وجود مشکل‌ها

اگر می‌خواهی می‌توانی محصول را با وجود مشکل‌ها منتشر کنی، اما دست‌کم باید حواست به آن‌ها و نقاط پرخطر پروژه‌ات باشد. حتی اگر آگاهی‌ات کمتر شده باشد، باز بهتر از این است که همه‌چیز را برای همیشه فراموش کنی. همین حالا هم مشکل‌های کافی از تو پنهان‌اند؛ عاقلانه نیست عمداً مشکل‌های بیشتری را پنهان کنی.

برای آگاه‌بودن از آن‌ها، اول باید کد را مثل یک آدم پارانوئید بازبینی کنی. باید بتوانی **کد را از آنِ خود بدانی**. و این کار زمان زیادی می‌برد؛ آن‌قدر که از جایی به بعد، استفاده از هوش مصنوعی دیگر ارزشش را نداشته باشد. برای اینکه استفاده از هوش مصنوعی توجیه‌پذیر شود، معمولاً مردم سطح دوم یا حتی اولِ بازبینی را انجام می‌دهند، اما اسمش را فقط «کد را بازبینی می‌کنم» می‌گذارند؛ انگار همین کافی است.

پیچیدگی نمایی است

بعداً در بخش «چطور درست از هوش مصنوعی استفاده کنیم» می‌بینی که مخالف استفاده از هوش مصنوعی نیستم. نمی‌گویم چون بازبینی کد تولیدشده با هوش مصنوعی آن‌قدر زمان‌بر است که ارزشش را ندارد، پس اصلاً نباید برای تولید کد از هوش مصنوعی استفاده کنی. درواقع فکر می‌کنم راه میانه‌ای وجود دارد.

اما برنامه‌نویسی یک ویژگی دارد: هرچه کدبیس بزرگ‌تر می‌شود، داشتن درک شناختی از آن سخت‌تر می‌شود. این رابطه یک‌به‌یک نیست؛ نمایی است. اگر اندازهٔ تغییرات کد را دو برابر کنی، مشکل‌ها خیلی بیشتر از دو برابر می‌شوند. هرچه بیشتر کد می‌نویسی ارتباط‌ها و نقاط دردناک بیشتری پدیدار می‌شوند. برای همین همیشه به ما گفته‌اند دامنهٔ هر تغییر را کوچک نگه داریم تا بتوانیم به نسخهٔ قبلی برگردیم و کارهای مشابه انجام دهیم. می‌توانی از هوش مصنوعی بخواهی چیزهای ساده‌ای برایت تولید کند،

خط‌به‌خط بخوانی‌شان و بفهمی‌شان؛ اما هرچه تغییرها بزرگ‌تر شوند، فهمیدن هر چیزی واقعاً سخت‌تر می‌شود.

درنتیجه مردم هم به خودشان و هم به رسانه‌ها دروغ می‌گویند. می‌گویند کد تولیدشده با هوش مصنوعی را بازبینی می‌کنند، اما با توجه به این شرایط غیرممکن است بتوانند پنج، ده یا بیست برابر کد بیشتری را دریافت کنند و بعد همه‌اش را بازبینی و درک کنند. حتی اگر کد را بی‌نقص بازبینی کنند، چون زمان، تمرین و حس دستاورد را از فرایند حذف کرده‌اند، خیلی زود آن را هم فراموش می‌کنند. دلیل اینکه دانشجویان پزشکی می‌توانند حجم زیادی اطلاعات و داده دریافت کنند این است که درس‌هایشان را عمیق می‌خوانند، همهٔ الگوها را یاد می‌گیرند و سعی می‌کنند تا جای ممکن معنایشان را بفهمند. آن‌ها فقط اسناد را مرور نمی‌کنند؛ آن‌ها را یاد می‌گیرند و این کار زمان و انرژی زیادی می‌خواهد. تازه حتی در محل کار هم مدام باید دنبال اطلاعات بگردند. نمی‌توانی تمام روز کد بخوانی، وانمود کنی متمرکزی درحالی‌که هیچ حس پاداشی نداری، و بعد ادعا کنی کد را بازبینی کرده‌ای. فقط خاک را زیر فرش فرستادی. از هیچ بهتر است، اما راه‌حل مناسبی نیست.

بازبینی کد وابستگی را از بین نمی‌برد

بازبینی کد وابستگی‌ای را که در بخش «توکن‌ها و جهنم وابستگی» توضیح دادم از بین نمی‌برد. شاید فکر کنی این مشکل‌ها فقط وقتی مهم‌اند که مدل هوش مصنوعی مدام شکست بخورد، اما دقیقاً همان زمان است که بازبینی کد کمترین کمک را می‌کند. اگر باگی ظاهر شود، هنوز باید برای تشخیص و رفعش به مدل تکیه کنی. اگر در حلقه بیفتد، وقت و توکن‌هایت هدر می‌روند و ضرب‌الاجل نزدیک‌تر می‌شود.

فرض کن یک پلتفرم کامل را وایب‌کد کرده‌ای و حالا با ده‌ها، صدها یا هزاران کاربر در حال کار است. شاید کد را بازبینی کرده باشی، اما اگر خودت در ساخت پروژه دخیل نبوده‌ای، هنوز مثل کسی هستی که برای اولین بار یک کدبیس بزرگ را می‌بیند. می‌توانی کم‌وبیش بفهمی کد چه می‌کند، بی‌آنکه بدانی چرا آن‌طور ساخته شده یا همه‌چیز چطور به هم وصل می‌شود. وقتی هوش مصنوعی شکست می‌خورد، باید دوباره سراغش بروی و التماسش کنی مشکل را درست کند.

چون پای باگ در میان است، همیشه نمی‌توانی بازخورد درستی بدهی. شاید ندانی چرا رخ داده؛ تنها کاری که می‌توانی بکنی این است که برای هوش مصنوعی توضیح بدهی چه زمانی رخ می‌دهد. این توضیح گاهی تلاش بسیار زیادی می‌خواهد. برای تبدیل‌کردن تمام چیزهایی که می‌بینی به کلمات، باید **خیلی** فن بیان

خوبی داشته باشی. برای همین بازبینی کد جواب مسئله نیست: می‌تواند نشان دهد کد چه می‌کند، اما درکی را که از ساختن و نگه‌داری‌کردن آن به دست می‌آوری به تو نمی‌دهد.

مقایسه‌های کاذب

چند مقایسهٔ مشهور وجود دارد که مردم هنگام توجیه یا رد استفاده از هوش مصنوعی می‌کنند و به‌نظر من آشکارا غلط‌اند. همان‌طور که قبلاً گفتم، هنگام قضاوت دربارهٔ هوش مصنوعی باید ثابت کنی چرا این قضاوت موقتی نیست و چند ماه دیگر بی‌اعتبار نمی‌شود. باید بدانی چرا و چطور با کاری که انسان انجام می‌دهد فرق دارد. هدف اصلی توجیه استفاده از انسان به‌جای ماشین است. اگر ثابت کنی هوش مصنوعی به‌اندازهٔ انسان بد است، دیگر واقعاً به برنامه‌نویس‌های انسانی نیاز نداریم.

«برنامه‌نویس بودی، حالا مدیری»

تنها راه توضیح نقش برنامه‌نویس‌ها در دورهٔ فعلی برنامه‌نویسی این است که آن را با مدیرها مقایسه کنی: انگار کارشان ارتقا پیدا کرده است. اگر قبلاً کد می‌نوشتی و مدیر بالادستی‌ای داشتی، حالا خودت مدیر شده‌ای. «ایجنت‌ها» کارمندهایت هستند و کدنویسی را برایت انجام می‌دهند؛ باید مدیریتشان کنی تا بهره‌وری‌شان را به حداکثر برسانی.

به‌نظر من این طرز فکر چرندی محض است. از خودت بپرس: کدام وایب‌کدر را می‌شناسی که برای بهترکردن توانایی وایب‌کدینگش (هر معنایی که داشته باشد) کتابی دربارهٔ مدیریت آدم‌ها خوانده باشد؟

اصل مدیریت این است که با آدم‌ها تعامل می‌کنی. چیزی که به‌عنوان مدیر از حرف‌زدن با آدم‌های مختلف، مدیریت احساسات و نیازهایشان، راحت و باانگیزه نگه‌داشتنشان در محیط کار و درنتیجه افزایش بهره‌وری‌شان یاد می‌گیری مفید است. به‌عنوان مدیر یک حلقه نمی‌نویسی و انتظار نداری آدم‌ها طبق برنامه قدم‌به‌قدم دنبالش کنند؛ مگر اینکه مدیر افتضاحی باشی که کارش را انجام نمی‌دهد.

تازه با این مسئولیت مهمِ مدیریت، اختیارهایی هم داری. می‌توانی آدم تنبل یا کسی را که کد بد وارد کدبیس می‌کند اخراج کنی. با یک مدل زبانی بزرگ می‌توانی چنین کاری بکنی؟

آنچه واقعاً اتفاق افتاده این است که ناگهان مجبور شده‌ایم خودمان را با «مدیر بودن» وفق بدهیم، درحالی‌که هم‌زمان تمام مزایای مدیر بودن را از دست داده‌ایم. اگر این وضعیت را به‌اندازهٔ کافی ادامه

بدهی، نه‌تنها توانایی‌های فنی‌ات را از دست می‌دهی، بلکه هیچ‌چیز هم از مدیر بودن به دست نمی‌آوری. برای همیشه در همان سطح خودت می‌مانی.

«قبلاً کد را کپی‌پیست می‌کردیم»

یکی از رایج‌ترین استدلال‌هایی که به نفع وایب‌کدینگ شنیده‌ام این است که ما برنامه‌نویس‌ها قبلاً هم از Stack Overflow یا GitHub کد کپی‌پیست می‌کردیم، پس انگار چیز زیادی عوض نشده. این استدلال به‌اندازهٔ مقایسه‌کردن مدل‌های زبانی بزرگ با کامپایلرها گمراه‌کننده است. اول بگذار کمی از مسیر خودم بگویم.

نقطهٔ عطف مسیر شغلی‌ام

از اکتبر ۲۰۱۹ برنامه‌نویسی می‌کنم. با زبان Python شروع کردم. بااینکه هر روز وقت قابل‌توجهی صرف یادگرفتن Python می‌کردم، درک پایه‌ای‌ام از برنامه‌نویسی واقعاً بهتر نشد. کتاب‌های مهندسی نرم‌افزار نمی‌خواندم و چیزی از کدنویسی تمیز نمی‌دانستم یا اهمیتی به آن نمی‌دادم. فقط می‌خواستم یک‌سری چیز بسازم. تخصصم هم خیلی محدود بود؛ به ربات‌های تلگرام و چند ابزار خودکارسازی خلاصه می‌شد.

در مسیر برنامه‌نویسی‌ام نقطهٔ عطفی بود که روی یک ربات تلگرامی با اندازهٔ متوسط کار کردم؛ رباتی که حالا ماهانه میزبان هزاران کاربر است. پروژهٔ ساده‌ای نبود. بعضی مسئله‌هایی که حل کردم هیچ‌کدام از رقیب‌هایش حل نکرده بودند و هنوز هم حل نکرده‌اند. هنوز هم به این پروژه افتخار می‌کنم.

اما کدبیسش زبالهٔ محض بود. به وضعیت فعلی‌اش نگاه نکن؛ با اینکه هنوز هم زباله است، آن موقع بدتر هم بود. اگر برنامه‌نویس Python باشی، شاید باورت نشود که حتی نمی‌دانستم `__init__.py` واقعاً چه‌کار می‌کند و برای همین هیچ‌وقت از آن استفاده نمی‌کردم. برنامه‌نویس عمل‌گرایی نبودم. برنامه‌نویسی‌ام تصادفی بود.

بااین‌حال، توانستم **با دست** و در مدت نسبتاً کوتاهی ــ نهایتاً دو ماه ــ ربات واقعاً خوبی بسازم. دلیل این تناقض این است که کپی‌پیست‌کردن کد آن‌قدرها هم که می‌گویند بد نیست.

واقعاً محال است بتوانی کل پروژه را با کپی‌پیست جلو ببری. این‌طور نیست که کل پروژه همین‌جا و آنجا پخش شده باشد و فقط باید همهٔ تکه‌های کد را پیدا کنی و به هم بچسبانی. اگر این‌طور بود، همه

می‌توانستند آن موقع برنامه بسازند، درست مثل اینکه حالا همه با هوش مصنوعی برنامه می‌سازند. برای اینکه اصلاً بتوانی کپی‌پیست کنی باید کلی چیز بلد باشی. هنرمندها هم از مرجع تصویری استفاده می‌کنند؛ فقط چسباندنشان کنار هم جواب نمی‌دهد. باید بدانی چطور این کار را بکنی.

چطور همه‌چیز به هم رسید

اوایل سال ۲۰۲۵ زندگی‌ام را عوض کردم. شروع کردم خیلی سخت کارکردن. تصمیم گرفتم بازی‌ساز شوم و انرژی زیادی پایش گذاشتم. دانشی که از آن ربات تلگرامی به‌دست آورده بودم، چشمم را به این واقعیت باز کرد که برنامه‌نویسی خیلی بیشتر از این‌هاست؛ و با همین نگاه عمل‌گرایانه تخصصم را گسترش دادم و تبدیل شدم به آدمی که امروز هستم.

با اینکه خیلی سخت کار می‌کردم، سریع‌تر از چیزی که انتظار داشتم پیشرفت می‌کردم و رشد می‌کردم. فقط چند ماه گذشته بود که داشتم یک بازی کارتی **بسیار** پیچیده می‌ساختم. واقعاً برایم عجیب بود که چطور توانسته بودم کاری را انجام بدهم که چند ماه قبل غیرممکن می‌دانستم.

فهمیدم آن همه سال برنامه‌نویسی و گسترش دانشم بالاخره به کارم آمده است. قبلاً کمی با هر زبان برنامه‌نویسی‌ای ور رفته بودم. کمی C امتحان کرده بودم، کمی Java، کمی HTML و CSS بلد بودم، دورهٔ JavaScript موزیلا را خوانده بودم اما اصلاً تمرین نکرده بودم، کمی با React Native کار کرده بودم، تا حدی Django را می‌شناختم و چیزهای دیگر. با هرکدام حداقل کار ممکن را کرده بودم و از هیچ‌کدام به‌تنهایی چیز زیادی به‌دست نیاورده بودم. اما امتحان‌کردن چیزهای مختلف و آشناشدن با دیدگاه‌های متفاوت کمک کرد تبدیل شوم به اقیانوسی با عمق فقط یک سانتی‌متر: چیزهای زیادی را کمی می‌دانستم، اما هیچ‌کدام را عمیق بلد نبودم.

اوایل برنامه‌نویس عمل‌گرایی نبودم، اما اگر همان برنامه‌نویس افتضاحی که بودم نمی‌شدم، نمی‌توانستم برنامه‌نویس خوبی شوم. بااین‌حال برنامه‌نویس بودم و همین هم ارزش خودش را دارد. کپی‌پیست‌کردن کد به‌هیچ‌وجه قابل‌مقایسه با وایب‌کدینگ نیست. این مقایسه فقط روشی است که با آن خودت را دلداری می‌دهی.

«پنجرهٔ کانتکست هوش مصنوعی کوچک است و توجه بدی دارد»

یکی از قضاوت‌های مشهوری که مردم برای منطقی به‌نظررسیدن مطرح می‌کنند این است که میگویند پنجرهٔ

کانتکست هوش مصنوعی کوچک است و چندان هم پیشرفت نکرده است. پنجرهٔ کانتکست مقدار داده‌ای است که هوش مصنوعی می‌تواند در یک لحظه نگه دارد. بنابراین ایجنت هوش مصنوعی نمی‌تواند کل کدبیس تو را در حافظه‌اش نگه دارد و همه‌چیز را به خاطر بسپارد. بسیار محدود است. attention هم بی‌نقص نیست و بعضی بخش‌های کانتکست از دست می‌روند. می‌توانی برای توضیح بیشتر این ویدئو را ببینی؛ این هم نمونه‌ای از همین قضاوت کاذبی است که درباره‌اش حرف می‌زنم.

اول باید بفهمیم: آیا واقعاً با کاری که انسان‌ها می‌کنند فرق دارد؟ کدام برنامه‌نویس را می‌شناسی که هنگام توسعه کل کدبیس را در حافظهٔ کاری‌اش داشته باشد؟ حتی می‌توانم بگویم اگر مستقیماً مقایسه کنیم (نه هرکدام را جداگانه)، هوش مصنوعی از انسان بهتر هم است. هوش مصنوعی ابزارهای زیادی برای بازیابی داده از بخش‌های مختلف پروژه دارد که بسیاری از آن‌ها در مقایسه با انسان‌ها سریع و دقیق‌اند. انسان‌ها به‌راحتی حواسشان پرت می‌شود (مخصوصاً به‌خاطر محرک‌های زندگی واقعی)، اما هوش مصنوعی این‌طور نیست.

اما می‌توانیم از زاویهٔ دیگری هم به این مشکل نگاه کنیم که کاملاً منطقی است: انسان‌ها با تمام چیزهایی که یاد گرفته‌اند، مخصوصاً پروژهٔ در دستشان، آموزش دیده‌اند؛ اما مدل‌های زبانی بزرگ تقریباً هیچ‌وقت این‌طور آموزش نمی‌بینند. هر بار که به‌عنوان برنامه‌نویس با مشکلی روبه‌رو می‌شوی، هر بار که برایش وقت می‌گذاری، لمسش می‌کنی، با آن سروکله می‌زنی و خودت حلش می‌کنی، حس موفقیت و دستاورد داری؛ دوپامین ترشح می‌کنی. مدام مغزت را با چیزهای مختلف آموزش می‌دهی و هم‌زمان آن‌ها را به خاطر می‌سپاری (همین مفهوم را در بخش «چرا این‌قدر برایت مهم است؟» هم بررسی کردیم). اما این اتفاق برای مدل‌های هوش مصنوعی نمی‌افتد. هر بار ایجنت را برای تغییر کدت باز می‌کنی، پروژه را از صفر می‌خواند. دلیل اصلی اهمیت پیداکردن مستندات همین است: تنها راه حفظ آن درک‌اند. این دقیقاً شبیه این هست که هر روز صبح توسعه‌دهنده‌ای جدید را استخدام کنی و کل پروژه را به او بدهی.

البته مقایسهٔ کانتکست مدل زبانی بزرگ با دانش انسان هم کاملاً منصفانه نیست. انسان‌ها پنجرهٔ کانتکست دارند و اسمش حافظه است. این حافظه نه‌تنها از نظر مدیریت کانتکست خیلی بهتر از حافظهٔ هوش مصنوعی است (مثلاً می‌تواند چیزهای بی‌اهمیت را نسبتاً دقیق حذف کند و باز هم چیزهای زیادی را از دهه‌ها پیش به یاد می‌آوری؛ با توجه به ظرفیت فعلی هوش مصنوعی، پنجرهٔ کانتکست بسیار بزرگ‌تری هم داری)، بلکه فهم انسانی با پارامترهای درونی و الگوهای یادگرفتهٔ هوش مصنوعی هم قابل‌مقایسه است. قبلاً هم خیلی دربارهٔ این موضوع حرف زده‌ام.

کم‌ارزش‌کردن ارشدیت و این حوزه

این بخش به «تأثیر هوش مصنوعی بر صنعت» تعلق دارد، اما چون شایستهٔ بخش جداگانه‌ای است آن را جدا کرده‌ام.

یکی از رایج‌ترین چیزهایی که مردم در حوزه‌های مختلف با آن روبه‌رو هستند این است که دیگران فقط چون کارشان را نمی‌فهمند، ارزش کارشان را پایین می‌آورند. این موضوع تقریباً دربارهٔ همهٔ شغل‌ها صدق می‌کند. مکانیک هستی، اما مردم فکر می‌کنند چون ماشینشان را در چند دقیقه تعمیر کرده‌ای نه چند ساعت، سزاوار دستمزد خوب نیستی. دندان‌پزشکی، اما مردم فکر می‌کنند برای اینکه فقط دندان‌هایشان را معاینه کرده‌ای نباید پول بگیری. طراح گرافیکی، اما مردم فکر می‌کنند فقط چند شکل با فتوشاپ می‌سازی و «خودشان هم می‌توانند انجامش دهند». حتی اگر ظاهراً کار خیلی ویژه‌ای نکرده باشی، وقتت همچنان ارزشمند است؛ اما دیگران این ارزش را نمی‌بینند و قضاوتی احساسی و عجیب دارند: فکر می‌کنند باید فقط بابت «مقدار کاری» که کرده‌ای و فایدهٔ فوری‌ای که داشته پول بگیری.

به‌نظر من دلیلش چیزی است که جامعه‌شناس‌ها و روان‌شناس‌ها باید کشف کنند. اما با اطمینان می‌توانیم بگوییم این طرز فکر درست نیست. نه‌تنها فکر می‌کنم نمی‌توانی قیمت محصول یا خدمت را با «اخلاق» تعیین کنی (راه عینی‌ای برای سنجیدنش وجود ندارد؛ فقط عرضه و تقاضاست)، بلکه باید این واقعیت را هم در نظر بگیری که فردی که برایت کار می‌کند نه‌فقط زمان زیادی برای یادگرفتنش گذاشته، بلکه شاید می‌توانسته کار دیگری انجام دهد که بهتر و از نظر مالی پاداش‌دهنده‌تر باشد. گاهی به کسی فقط برای زمانی که صرف می‌کند پول می‌دهی، نه برای چیزی که ارائه می‌دهد. در نهایت یک معامله است. اگر احساس کند از این معامله به‌اندازهٔ چیزی که از دست می‌دهد (یا بیشتر از آن) به دست نمی‌آورد، کاملاً حق دارد قیمتش را بالاتر ببرد.

این دقیقاً همان چیزی است که در صنعت فناوری اتفاق افتاده است و با ورود مدل‌های زبانی بزرگ به‌شدت بدتر شده است. کسانی که ارزش این حرفه را نمی‌دیدند حالا با اعتمادبه‌نفس بیشتری انتظارهای کاذبشان را بیان می‌کنند و تجربهٔ متخصص‌ها را خراب می‌کنند. یکی از رایج‌ترین روش‌هایی که غیربرنامه‌نویس‌ها برای قضاوت برنامه‌نویس‌ها دارند این است که ببینند کد «کار می‌کند» یا نه. اما «فقط کارکردن» کد کاری است که جونیورها هم از پسش برمی‌آیند. پس ارشدیت یعنی چه؟ اینجاست که روشن می‌شود آن غیربرنامه‌نویس‌ها اصلاً برای سینیوربودن ارزشی قائل نیستند. اگر کدت کار کند، از تو می‌خواهند منتشرش

کنی؛ حتی اگر نمونهٔ تستی باشد و مدام به آن‌ها هشدار بدهی. سینیوری چون می‌دانی صرفاً اینکه چیزی همین حالا کار می‌کند به این معنی نیست که همیشه هم کار خواهد کرد. یادت باشد ارشدیت یعنی حل‌کردن مشکل‌ها پیش از آنکه فرصتی برای رخ‌دادن پیدا کنند. این یک سرمایه‌گذاری است و این سرمایه‌گذاری در دریای هیاهو دارد از دست می‌رود.

برای وایب‌کدینگ چه چیزی باید یاد بگیری؟

الان همه با سردرگمی زیادی روبه‌رو هستند. اگر افراطی‌ها را کنار بگذاری، آدم‌هایی که از برنامه‌نویسی سنتی به دورهٔ تازهٔ برنامه‌نویسی «مهاجرت کرده‌اند» از هر طرف می‌گویند که هنوز هم باید یاد بگیری. بدتر اینکه بعضی‌ها می‌گویند یادگیری از همیشه مهم‌تر است و «باید سینیور شوی».

و این واقعاً گیج‌کننده است. دقیقاً چه چیزی باید یاد بگیریم؟ اگر کدنویسی را به ماشین همه‌توان بسپاریم، پس چه چیزی باید یاد بگیریم؟ طراحی سامانه؟

حتی بهترین کتاب‌های برنامه‌نویسی که قرار است طراحی سامانه یاد بدهند، تقریباً هیچ‌چیزی یادت نمی‌دهند. فقط یک‌سری طرز فکر یادت می‌دهند و انتظار دارند با «تجربه» یاد بگیری. اما واقعاً چه‌کار دیگری می‌توانستند بکنند؟ مثل این است که کتاب شنا بخوانی و انتظار داشته باشی همان لحظه شناگر شوی.

تنها جواب باقی‌مانده این است که «یاد بگیریم چطور با هوش مصنوعی کار کنیم»؛ که این حتی بی‌معنی‌تر است.

۱. هزینه

می‌دانی هوش مصنوعی الان چقدر گران است، نه؟ تازه نمی‌توانی با کارکردن با یک مدل ضعیف یاد بگیری. هرکدام از وایب‌کدرها وقتی از بهترین مدل‌های پیشرو استفاده نمی‌کنی مسخره‌ات می‌کنند؛ مدل‌هایی که تمام سقف مصرفت را یک‌جا می‌سوزانند و می‌گویند «داری کلی چیز را از دست می‌دهی». تازه یادگرفتن خیلی بیشتر از تولیدکردن زحمت می‌برد و درنتیجه توکن‌های خیلی بیشتری مصرف می‌کند.

باید قبول کنم شاید با گذشت زمان ارزان‌تر شود. اما اگر بهانه‌ات همین است، تا آن موقع ادعا نکن *کدنویسی حل شده است*. باید بفهمی این بخش از معادله ارتباط زیادی با سخت‌افزار دارد، و دانش سخت‌افزار به‌اندازهٔ دانش نرم‌افزار سریع پیشرفت نمی‌کند؛ پس اگر با همان معیار قضاوت می‌کنی،

پیش‌بینی‌ات ممکن است به‌شدت غلط از آب دربیاید.

۲. هیچ راهی برای راستی‌آزمایی نیست

یکی از دوستان برنامه‌نویسم تا دو ماه پیش اصلاً هوش مصنوعی را امتحان نکرده بود. بالاخره تصمیم گرفت دست‌به‌کار شود و مفهوم‌هایش را یاد بگیرد. خبرهای رسانه‌ای دربارهٔ هوش مصنوعی را دنبال می‌کرد، اما خودش هیچ‌وقت با آن کار نکرده بود. شنیده بود قابلیتی به‌اسم اسکیل (skill) هست که خیلی خوب و مهم است و کمک می‌کند چیزهای خوبی بسازی. رفت درباره‌اش یاد گرفت و بعد برگشت پیشم و گفت: «یعنی اسکیل همین است؟ فقط با کامپیوتر حرف می‌زنی؟»

اگر راهی برای راستی‌آزماییِ درستیِ چیزهایی که یاد گرفته‌ای نداشته باشی، چطور یاد می‌گیری؟ همان‌طور که قبلاً توضیح دادم، هوش مصنوعی غیرقطعی است. دلیل منطقی‌ای وجود ندارد که چرا به شکل خاصی کار می‌کند؛ دست‌کم نه از آن جنس منطقی که ما انسان‌ها برای فهمیدن به آن نیاز داریم. چیزی به مدل زبانی بزرگ می‌گویم و چیزی جوابم می‌دهد. اگر نفهمم چه چیزی ورودی را به خروجی وصل می‌کند، تمام دستگاه منطقی ذهنم از کار می‌افتد. و وقتی نتوانی پیوندهای عینی پیدا کنی، یادگرفتن واقعاً سخت است.

چیزها را با یادگرفتن الگوهایشان یاد می‌گیریم. وقتی الگوها را یاد بگیری، دیگر به آموزش‌های قدم‌به‌قدم وابسته نیستی.

اما در توسعه با هوش مصنوعی، الگو روشن نیست. حتی سرش توافق هم ندارند. هرکسی نظری دارد. پس راهنمایی یا روش علمی‌ای برای یادگیری در اختیارت نمی‌گذارند؛ ابزاری به تو می‌دهند و انتظار دارند با یک میلیون بار اجراکردنش یادش بگیری (آن هم بدون هدردادن هزاران دلار). آخرش هم می‌فهمی همان منطق روی مدل‌های دیگر جواب نمی‌دهد و باید آزمایش‌کردن را از صفر شروع کنی.

چند بار سعی کرده‌ام با هوش مصنوعی به نتیجه برسم: برنامه‌نویسی را نگاه می‌کنم، می‌فهمم از چه چیزهایی ساخته شده، اصل‌های روشنش را بیرون می‌کشم و سعی می‌کنم همان‌ها را با هوش مصنوعی تکرار کنم؛ اما مدام شکست می‌خورد. نمی‌فهمم دقیقاً چرا شکست خورد؛ درست مثل وقتی که نمی‌دانی چرا باگی ایجاد شده، اما این بار حتی نمی‌دانی مشکل از مهارت توست یا محدودیت واقعی ابزار. تنها راه فهمیدنش این است که مدل را آن‌قدر اجرا کنی که دیگر عملاً نشود شکست را گردن هوش مصنوعی انداخت. این کار اصطکاک زیادی دارد و هزینه‌اش هم بالاست؛ نمی‌توانی برای هر باگی انجامش بدهی.

می‌دانی، در حرفه‌های علمی وقتی می‌خواهی مسئله‌ای را حل کنی باید تا جایی که می‌توانی ایزوله‌اش کنی، چون نمی‌خواهی با مغالطه‌ها خودت را گیج کنی. شاید منبع مشکل را نفهمی و چیزی را فقط کمی مرتبط با آن رفع کنی؛ مشکل اصلی سر جایش بماند، اما خیال کنی کاری کرده‌ای. اما با هوش مصنوعی این امتیاز را نداری که ابزار را زیر سؤال ببری. مرزهایش روشن نیستند. آخر سر با خودت می‌گویی: «نکند مشکل از من است؟» و جواب عینی‌ای هم نداری. هیچ‌کس جواب عملی نمی‌دهد؛ همه می‌گویند: «اگر نمی‌توانی کل شغلت را باهاش جایگزین کنی، مشکل از مهارت خودته. منبع؟ به من اعتماد کن داداش».

وقتی نتوانی درکت را به‌طور عینی راستی‌آزمایی کنی، پاداشی حس نمی‌کنی؛ **حس نمی‌کنی واقعاً چیزی یاد گرفته‌ای**. باز هم نقل‌قول کنم: *اگر ندانی چرا چیزی کار می‌کند، نمی‌فهمی چرا از کار افتاده.*

در نگاه حداکثرگرا به هوش مصنوعی (AI maximalism) هیچ‌کس در امان نیست

قبلاً مهندسی نرم‌افزار را حرفه‌ای بسیار تخصصی می‌دانستند که به حل مسئلهٔ عملی زیادی نیاز دارد. اگر کل این حوزه «حل شده»، ادعاکردن اینکه شغل‌های دیگر در امان‌اند واقعاً سخت می‌شود. دلیل اینکه در حوزه‌های دیگر پیشرفت‌های زیادی نمی‌بینیم این است که هوش مصنوعی در تشخیص تصویر هنوز آن‌قدر پیشرفت نکرده، هرچند دارد بهتر می‌شود.

این ما را برمی‌گرداند به استدلالِ «پایان نزدیک است». ترجیح می‌دهم تئوری‌های توطئه‌ای را که می‌گویند دنیا به آخر نزدیک است نادیده بگیرم، چون بده‌بستانش روشن است. نمی‌خواهم عمرم را صرف این کنم که فکر کنم پایان دنیا نزدیک است و بعد معلوم شود هنوز خیلی مانده وقتی که باورکردنشان قرار نیست کمک خاصی به من بکند.

هر استدلالی از این جنس باشد در همین دسته قرار می‌گیرد: «با یک جونیور و Claude می‌توانم برنامه‌ای باکیفیت تولید کنم»، «هوش مصنوعی خروجی تیم ما را صد برابر کرده»، «دویست برنامه‌نویس سینیور را با سه برنامه‌نویس و هوش مصنوعی نامحدود جایگزین کردیم» و از این حرف‌ها.

دیگر به متخصص هوش مصنوعی نیازی نیست

حوزه‌ای داریم به‌اسم سئو(SEO)، یعنی بهینه‌سازی برای موتورهای جست‌وجو. اوایل عمر اینترنت،

متخصص‌های سئو می‌توانستند کارهای زیادی بکنند. یکی از کارهایشان بمباران کلیدواژه‌ای بود؛ کمک می‌کرد وب‌سایت‌ها در نتایج جست‌وجوی Google رتبهٔ بالاتری بگیرند. اما یک زمانی Google الگوریتمش را به‌روز کرد و همهٔ کسانی را که این کار را می‌کردند جریمه کرد. چون Google بازی عادلانه می‌خواهد. می‌خواهد چیزی را به هر کاربر نشان دهد که واقعاً دنبالش است و از آن لذت می‌برد. هدف Google این است که دیگر به آدم‌هایی مثل متخصص‌های سئو نیاز نباشد. عدالت به حقه نیاز ندارد.

با اینکه متخصص‌های سئو هنوز هستند، مثل قبل تقاضای زیادی برایشان نیست. همین اتفاق برای متخصص‌های هوش مصنوعی هم می‌افتد. هدف مدل زبانی بزرگ این است که دیگر به آدم‌هایی مثل آن‌ها نیاز نباشد. پس فکر نکن اگر یاد بگیری با مدل زبانی بزرگ کار کنی در امانی؛ تو هم در امان نیستی.

برای برنامه‌نویس باتجربه هم ترسناک است

همان‌طور که قبلاً توضیح دادم، برنامه‌نویس‌بودن یعنی این شهود و اقیانوس دانشی که در طول زمان به‌دست می‌آوری، تیز نگه می‌داری و مدام گسترشش می‌دهی. این‌طوری شکل گرفته چون تا خرخره درگیر مشکل‌ها بودیم، می‌گذاشتیم مغزمان پردازش کند سروکله‌زدن با یک مسئله چه شکلی است، با تعامل‌کردن یاد می‌گرفتیم، با دیدن اطلاعات تازه و نامربوط یاد می‌گرفتیم و خیلی چیزهای دیگر.

این‌طور نیست که هوش مصنوعی فقط زبان برنامه‌نویسی‌ای را که استفاده می‌کردیم عوض کرده باشد. فرایند کارمان را آن‌قدر زیرورو کرده که دیگر نمی‌شود شناختش. هر برنامه‌نویسی می‌داند سازگارشدن با وایب‌کدینگ به تغییر ذهنیت بزرگی نیاز دارد.

برنامه‌نویس‌های سینیور، شهودتان را تیز کرده‌اید؛ این حس عنکبوتی را در خودتان پرورش داده‌اید تا وقتی دارید کار بدی می‌کنید درونتان حسش کنید و بایستید. برای همین وقتی قرار است کارتان را به هوش مصنوعی واگذار کنید، همهٔ حواستان زنگ خطر می‌زند. فقط به این معنی نیست که به شکل قبلی توسعهٔ نرم‌افزار اعتیاد پیدا کرده‌ای. این تغییر ذهنیت فقط مربوط به شیوهٔ کارَت نیست؛ با تمام جهان‌بینی‌ات تناقض دارد. نمی‌دانم تو چه حسی داری، اما من اگر شهودم این‌قدر غیرقابل‌اعتماد شده باشد، دیگر نمی‌دانم چطور به آن اعتماد کنم.

چطور درست از هوش مصنوعی استفاده کنیم

استفادهٔ درست از هوش مصنوعی یعنی آن را ابزار بدانی، در برابر هیاهویش مقاومت کنی و برای دوری از آسیب بلندمدت با دقت برنامه بریزی. راستش کار آسانی نیست. سرعت تغییرها، قرارگرفتن مداوم در معرض حرف‌هایی که ترس از جا ماندن را تشدید می‌کنند، دشوارفهم‌بودن این حوزه که ناگهان از برنامه‌نویس‌ها انتظار دارد فیلسوف شوند، و روان‌پریشیِ مرتبط با هوش مصنوعی (AI psychosis)، همه فکرکردن منطقی را سخت‌تر می‌کنند.

برای خودم هم خیلی سخت بوده. همان‌طور که در پیشگفتار گفتم، اول این کتاب را برای خودم نوشتم تا فکرهایم را جمع‌وجور کنم، بفهمم از نظر ذهنی کجا ایستاده‌ام و بالاخره وضعیتم را بپذیرم.

اما همیشه این‌طوری نبود. یادم هست یک سال پیش Claude را خریدم تا در بازی‌سازی کمکم کند. از خوشحالی توی پوست خودم نمی‌گنجیدم. خیلی بیشتر شبیه ابزار بود تا کارمند تازه‌ای که نتوانم به او اعتماد کنم. واقعاً چیزهایی یاد می‌گرفتم و رشد می‌کردم تا برنامه‌نویس و بازی‌ساز بهتری شوم. اما حالا دارم از این مدل‌های زبانی می‌خواهم تمام کدنویسی را انجام دهند و حس می‌کنم گیر افتاده‌ام.

فرقش در شیوهٔ استفاده‌ام از هوش مصنوعی بود. از آن نمی‌خواستم همه‌چیز را انجام دهد. خیلی از کارها را خودم می‌کردم. گاهی از آن می‌خواستم چیزی تولید کند، اما بعد همهٔ کدی را که ساخته بود می‌خواندم و سعی می‌کردم تا جای ممکن بفهممش. فقط یک بار یادم می‌آید برای حل مسئله بدون راستی‌آزمایی از آن استفاده کرده باشم. چیزی شبیه ابزار تبدیل بود. اسکریپت ساده‌ای بود، اما مهم. چون دانش زیادی دربارهٔ زبان برنامه‌نویسی C# لازم داشت، نتوانستم بفهممش. چند بار آزمایشش کردم و دیدم کار می‌کند. گذاشتمش کنار تا بعداً چند آزمون برایش بنویسم و خیالم راحت شود.

کاری که من کردم و پیشنهادم به تو هم همین است، در دو دسته خلاصه می‌شود:

روش‌های درست

دو روش هست که می‌توانی با آن‌ها از قدرت هوش مصنوعی استفاده کنی، بی‌آنکه گرفتار جنبه‌های منفی‌اش شوی:

۱. کارهایی که حل مسئله نمی‌خواهند

این آسان‌ترین روش است. از آن برای کارهایی استفاده می‌کنی که به حل مسئله نیاز ندارند و صرفاً مکانیکی‌اند. در برنامه‌نویسی، بیشتر `refactoring` را می‌شود به هوش مصنوعی سپرد. اما یادت باشد همهٔ رفکتورینگ‌ها مکانیکی نیستند. هرچه دامنهٔ کار بزرگ‌تر شود، مکانیکی‌بودنش کمتر می‌شود. انتقال کدبیس از Python به Go قطعاً به‌اندازهٔ تغییر انبوهِ اسم فایل‌ها و متغیرها مکانیکی نیست.

۲. مثل کارمند نامطمئن با آن رفتار کن

این‌طوری به قضیه نگاه کن: کسی به تو ایمیل می‌زند و می‌گوید دنبال کار می‌گردد. نمونه‌کارهایش را می‌بینی و می‌فهمی چندان خوب نیست. اما دستمزدی که می‌خواهد آن‌قدر کم است که استخدامش می‌ارزد. استخدامش می‌کنی، اما حواست به خروجی کارش هست.

می‌توانی از هوش مصنوعی بخواهی کد بنویسد و چیزهایی را پیاده‌سازی کند، اما باید تک‌تک خط‌ها را بخوانی و بفهمی چه فکری کرده و چرا آن‌طور پیاده‌سازی کرده است. اگر نمی‌دانی، از خودش بپرس. هیچ‌وقت فرض نکن «حتماً دلیلش چیزی است که من نمی‌دانم». این فرض به‌راحتی می‌تواند به بدهی سه‌گانه منجر شود.

روش‌هایی که کمی نامناسب‌اند

البته می‌فهمم که قرار نیست همهٔ کدها بی‌نقص، خالی از مشکل‌های ساختاری و نگه‌داری، یا تا مغز استخوان فهمیده‌شده باشند. می‌خواهی جلوی گسترش تدریجی دامنهٔ پروژه (scope creep) را بگیری. پس ریسک‌کردن اشکالی ندارد، اما باید آگاهانه ریسک کنی. گاهی اشکالی ندارد از هوش مصنوعی بخواهی چیزی تولید کند و به چند اسکریپت و ابزار اعتماد کنی. اما همیشه یادت باشد به چه چیزهایی اعتماد کرده‌ای و به چه چیزهایی نه. اگر می‌توانی برایشان آزمون واحد بنویسی، این کار را بکن؛ بهترین راه برای پذیرش ریسک آگاهانه همین است. اگر نمی‌توانی، دست‌کم یادت نگه‌شان دار یا جایی بنویسشان. پیشنهاد می‌کنم یک فایل واحد بسازی، با الهام از ADR، اما برای ثبت همهٔ تصمیم‌ها. بعداً اگر وقت داشتی، مرورشان کن.

بیشتر وقت‌ها نمی‌توانی درست از آن استفاده کنی

بزرگ‌ترین چالش برنامه‌نویس‌هایی که خودشان می‌خواهند همه‌چیز را وایب‌کد نکنند، این است که در این

دوره چطور به یادگیری ادامه بدهند. یکی از مشکل‌ها هم از چیزی می‌آید که ذاتاً پیش‌بینی‌ناپذیر است و عملاً راه گریزی از آن نیست: باگ‌ها. همان‌طور که در بخش «دیگر جست‌وجو نمی‌کنی» گفتم، وقتی باگ‌ها را خودت رفع می‌کردی، زمانی که صرفشان می‌شد به یادگیری و رشدت کمک می‌کرد. حتی می‌گویم یکی از بزرگ‌ترین عوامل همین است، چون برنامه‌نویسی بیشتر وقت‌ها یعنی پیداکردن مشکل‌ها و باگ‌ها.

اما مسئله این است که برنامه‌نویسی ذاتاً با تو سر جنگ دارد. باگ‌ها را عمداً نمی‌سازی. همین‌طوری به‌وجود می‌آیند و بیشتر وقت‌ها نمی‌دانی چرا. پس نمی‌دانی باگ مهم است و ارزش یادگرفتن دارد یا نه. بعضی‌هایشان احمقانه‌اند و بعضی‌ها نه. اگر برای رفع باگ‌ها از هوش مصنوعی استفاده کنی، داری با دانشت قمار می‌کنی. کتاب‌خواندن یا دست‌به‌کدشدن هم چیزی نیست که بتواند جبرانش کند. وقتی باگی پیش می‌آید، تنها فرصتت برای یادگرفتن روش حلش است. وقتی حل شد، فقط پازلی حل‌شده برایت می‌ماند که دیگر چیزی یادت نمی‌دهد.

اگر این رشته‌فکر را ادامه بدهی، می‌فهمی بیشتر کاربردهای دیگر هم همین‌اند. کندوکاو در کدبیس برای یادگرفتن، بخش مهمی از مسیر یادگیری توست. در بخش «بازبینی کد جواب مسئله نیست» توضیح دادم چطور با کمک هوش مصنوعی قابلیتی به Godot اضافه کردم. اما نکتهٔ پنهانی که نگفتم این است که نتوانستم این مسیر را ادامه بدهم. چند بار دیگر تلاش کردم باگ‌های حل نشده را رفع کنم یا هر کار دیگری انجام دهم، اما به‌شدت شکست خوردم. فهمیدم موقع استفاده از هوش مصنوعی ــ هوش مصنوعی فرایند فهمیدن کدبیس و زبان برنامه‌نویسی را به عهده گرفته بود. فقط در حال تولید بودم. توانستم آن قابلیت را اضافه کنم، اما همین؛ فقط همان قابلیت را اضافه کردم. آن دانش بیشتر دانشی یک‌بارمصرف بود. عملاً همان مثال دوم‌اسکرول‌کردن بود. جونیورها با هوش مصنوعی همچنان جونیور می‌مانند.

یکی از جنبه‌های مهمی که هوش مصنوعی را با برنامه‌نویس‌ها ناسازگار می‌کند در بخش «پنجرهٔ کانتکست هوش مصنوعی کوچک است و توجه بدی دارد» توضیح داده شده است. نمی‌گویم نمی‌توانی یا نباید در کارت به‌عنوان برنامه‌نویس از هوش مصنوعی استفاده کنی. فقط توضیح می‌دهم چرا اینجا اصطکاک زیادی وجود دارد که ممکن است مردم از آن غافل شده باشند. همه فکر می‌کنند هوش مصنوعی ابزار جدید ماست، درحالی‌که این ابزار از نظر ماهیت کاملاً با ما فرق دارد و به‌نوعی باید گردش‌کار انسانی‌مان را به گردش‌کار هوش مصنوعی تبدیل کنیم و همان بهره‌وری را حفظ کنیم. این‌طور کار نمی‌کند. امیدهای زیادت واقعیت را تغییر نمی‌دهند، حتی اگر دنیا خودش را با تو وفق دهد.

فصل ۴

آیندهٔ هوش مصنوعی

اولش نمی‌خواستم این فصل را بنویسم. با این سرعت و پیش‌بینی‌ناپذیریِ تغییرها، خیلی راحت ممکن است پیش‌بینی‌هایت ــ دست‌کم از نظر زمان‌بندی ــ کاملاً غلط از آب دربیایند. اما به دو دلیل نگهش داشتم: اول اینکه می‌خواهم فکرهای همین امروزم را ثبت کنم تا بعداً به آن‌ها برگردم؛ دوم اینکه می‌خواهم دربارهٔ انتظارهایی از آینده حرف بزنم که ممکن است روی زندگی امروزمان اثر بگذارند، حتی اگر واقعیت نداشته باشند.

دانشگاه هوش مصنوعی

با اینکه در این کتاب بارها با جایگزین‌کردن متخصص‌ها با هوش مصنوعی مخالفت کرده‌ام، می‌توانم تصور کنم اشتباه می‌کنم و به این فکر کنم که با سرعت فعلی هوش مصنوعی آینده چه شکلی می‌شود.

یکی از چیزهایی که در بخش «برای وایب‌کدینگ چه چیزی باید یاد بگیری؟» گفتیم این است که فهمیدن اینکه هوش مصنوعی چه‌کار می‌کند واقعاً سخت است و راهی برای راستی‌آزمایی‌اش نداریم. شاید اگر رشته‌های دانشگاهی بسیار حرفه‌ای‌ای برایش راه بیندازیم، دست‌کم از نظر نظری یکی از این مشکل‌ها حل شود. به حوزه‌هایی مثل جامعه‌شناسی یا حتی طراحی گرافیک نگاه کن. هرچند این حوزه‌ها حتماً از روش‌های علمی استفاده می‌کنند ــ درست مثل وایب‌کدینگ ــ بیشترشان بر تجربه و نتیجه‌ها تکیه دارند. در بخش «کژگراییِ نتیجه‌نگر» توضیح دادم چرا قضاوت بر اساس نتیجه از نظر منطقی غلط است. اما گاهی واقعاً نمی‌توانی فقط با منطق محض قضاوت کنی. جامعه آن‌قدر عامل دارد که نمی‌شود همه‌چیز را با منطق ریاضی توضیح داد. بااین‌حال به جامعه‌شناسی نیاز داری؛ پس توسل به تجربه‌های شخصی، آن هم به‌شکلی حرفه‌ای، موجه است.

برای حل این مشکل روش تازه‌ای به‌کار بردیم: راه‌های ریاضی‌وار معتبری برای ارزیابی نتیجه‌ها پیدا می‌کنیم ــ مثل نمونه‌گیری تصادفی ــ، تعداد زیادی از آن‌ها را گرد می‌آوریم و بعد با فلسفه، منطق، آمار، روان‌شناسی و چیزهای دیگر به نظر می‌رسیم. چون راهی عینی برای راستی‌آزمایی نتیجه‌ها نداریم و فقط می‌توانیم تا حدی مطمئن باشیم، گفته‌های جامعه‌شناختی را هیچ‌وقت واقعیت عینی حساب نمی‌کنیم. نباید به

حرف‌های یک جامعه‌شناس همان‌طور اعتماد کنیم که به این جمله اعتماد داریم: «جاذبه وجود دارد». این به‌هیچ‌وجه از ارزش جامعه‌شناسی کم نمی‌کند. تاریخ معلوم می‌کند باید به یک جامعه‌شناس اعتماد کنیم یا نه. همین‌طور شاید تاریخ پیش‌بینیِ من دربارهٔ هوش مصنوعی را درست از آب دربیاورد؛ اما این معنایش این نیست که از همان اول عینی درست می‌گفتم. حدس محاسبه‌شده‌ای زدم و درست درآمد. جامعه‌شناسی، طراحی گرافیک و حوزه‌های مشابه فقط مجموعه‌ای از حدس‌های محاسبه‌شده.

ممکن است همین اتفاق برای مهندسی هوش مصنوعی بیفتد. چون روش‌های عینی و قطعی‌ای برای راستی‌آزمایی نداریم، باید روی این حوزه سرمایه‌گذاری کنیم و آن را به رشته‌ای دانشگاهی تبدیل کنیم. به‌جای اینکه انتظار داشته باشیم آدم‌ها همهٔ این آزمون‌ها را روی مدل‌ها انجام دهند، خودمان انجامشان می‌دهیم و با ریاضیات مطمئن می‌شویم این آزمون‌ها سوگیری ندارند و درست اجرا می‌شوند. بعد از دلشان قانون درمی‌آوریم. سپس این قانون‌ها را در طول تاریخ بشر شکل می‌دهیم تا روان‌ترین تجربهٔ توسعه را بسازیم.

شاید برای همین هم ما مهندسان نرم‌افزار در سازگارشدن با هوش مصنوعی مشکل داریم. انتظار دارند ناگهان چندین سال دانش پیدا کنیم، درحالی‌که هیچ‌کس چیزی نمی‌داند. مثل این است که برگردیم به هزاران سال پیش و از آدم‌های آن دوره انتظار داشته باشیم مفهوم‌های رایج جامعه‌شناسی و روان‌شناسی را برایمان توضیح دهند. دلیل اینکه جامعه‌شناسی و حوزه‌های مشابه امروز این‌قدر خوب کار می‌کنند این است که طی هزاران سال به کمال رسیده‌اند. برنامه‌نویسی به‌طور کلی کمتر از صد سال عمر دارد.

انتظارهای کاذب

حتی اگر هوش مصنوعی ذاتاً غیرقابل‌اعتماد باشد، شباهت بیش‌ازحدش به انسان باعث شده ــ و همچنان باعث خواهد شد ــ خیلی از مردم، مدیرها و مدیرعامل‌ها امیدها و انتظارهای کاذبی داشته باشند.

۱. کمبود سینیورها

مدیرها دیگر جونیور استخدام نمی‌کنند و رویشان سرمایه‌گذاری نمی‌کنند. سینیورها بازنشسته می‌شوند و یک زمانی کمبود برنامه‌نویس سینیور خواهیم داشت. به همین سادگی.

۲. بدهی فنی

هرچند انواع دیگر بدهی هم نقش دارند، بدهی فنی بیشتر به چشم می‌آید. کمبود سینیورها، توهم‌زایی و

استفاده از مدل‌های ضعیف‌تر به‌خاطر هزینهٔ هوش مصنوعی می‌توانند بدهی را انباشته کنند و در آینده کلی مشکل بسازند.

۳. خراب‌کردن صنعت برای چند سال

در نهایت ممکن است خیلی راحت دست‌کم برای چند سال صنعت را خراب کنیم. جلویش را نمی‌شود گرفت، چون مشکل‌هایی که در این کتاب گفتم فوراً دیده نمی‌شوند؛ فقط وقتی روی هم جمع شدند خودشان را نشان می‌دهند. امیدوارم بتوانیم زودتر متوقفش کنیم. برای همین شدیداً پیشنهاد می‌کنم دربارهٔ این موضوع‌ها حرف بزنید.

نرم‌افزار می‌میرد

اگر با وجود برداشتم از موضوع، تنها مانع ورود به مهندسی نرم‌افزار ایده باشد و ماشین همهٔ کارهای دیگر را انجام دهد، بازار آن‌قدر اشباع می‌شود که نرم‌افزار عملاً می‌میرد. وقتی خودت می‌توانی نرم‌افزار بسازی، چرا باید بخری‌اش؟ مالکیت فکری دیگر هیچ معنایی ندارد؛ چون می‌توانی ایدهٔ نابغه‌ای را که کسی برایش برنامه ساخته برداری و خودت از صفر، دقیقاً مطابق نیازهایت، برنامه‌اش را بسازی. اگر به کسی نشانش ندهی، کار غیراخلاقی یا غیرقانونی‌ای نکرده‌ای.

چند هفته پیش ایلان ماسک توییت کرد که زبان‌های برنامه‌نویسی تا ده سال دیگر می‌میرند و هوش مصنوعی مستقیماً کد ماشین می‌نویسد. حرفش همه را غافلگیر کرد و مسخره‌اش کردند، اما این همان نگاه حداکثرگرا به هوش مصنوعی است. وقتی می‌گویی کل کدنویسی را به هوش مصنوعی می‌سپاری، انتظار مشابهی را بیان می‌کنی. نمی‌گویم اگر واگذارکردن کد به هوش مصنوعی راه درست از آب دربیاید، حتماً تا ده سال دیگر هوش مصنوعی مستقیم کد ماشین می‌نویسد. اما همین پیش‌بینی را می‌توان دربارهٔ ادعاهایی از این دست هم کرد: «در آینده سیستم‌عامل‌ها منسوخ می‌شوند و همه سیستم‌عامل بومیِ هوش مصنوعی خودشان را می‌سازند». خودم هم یک دسته از این پیش‌بینی‌های ChatGPT را در یک توییت گذاشتم. خواندنشان واقعاً خنده‌دار است.

فروپاشی تمدن

ویدئوی بسیار متفکرانه‌ای از جاناتان بلو با عنوان جلوگیری از فروپاشی تمدن وجود دارد. در این ویدئو دربارهٔ این حرف می‌زند که فناوری خودبه‌خود فرسوده می‌شود و برای بهترشدن واقعی به تلاش مداوم نیاز دارد. در

طول تاریخ هزاران فناوری به همین دلیل از دست رفته‌اند. بسیاری از پیشرفت‌های علمی و فناوری بـه هیچ تبدیل شده‌اند. سال‌ها طـول کشـید تا بفهمیم اهـرام جـیزه چطـور سـاخته شـده‌اند و هنـوز دربـارهٔ خیلی از جنبه‌هایشان مطمئن نیستیم.

نرم‌افزار همین حالا هم در حال فرسوده‌شدن اسـت. این ویـدئو مربـوط بـه سـال ۲۰۱۹ اسـت، خیلی پیش از عرضهٔ عمومی ChatGPT. حرفی که از ذهنم بیرون نمی‌رود این است که مردم می‌گویند: «بله، می‌توانسـتیم این نرم‌افزار را بهتر و کم‌باگ‌تر کنیم، اما بـازار پولش را نمی‌دهـد»؛ امـا از کجـا می‌دانیم واقعـاً می‌توانسـتند؟ وقتی دهه‌ها از ساخت یک نرم‌افزار درست‌وحسابی می‌گذرد، چه چیزی بـاعث می‌شـود فکـر کـنیم هنـوز آن دانش را داریم؟ دانش بایـد مـدام منتقـل و صـیقل داده شـود؛ وگرنـه می‌پوسـد و فرسـوده می‌شـود. من، به‌عنـوان برنـامه‌نویس، می‌تـوانم بگـویم کـد بهـتری می‌نویسـم، امـا واقعیت این اسـت کـه هرچـه بیشـتر انجام‌دادن درست کارها را متوقف کنی، بیشتر از یادگیری و پیشـرفت بازمی‌مـانی و دانشـت بیشـتر منقضـی می‌شود. شاید از نظر فنی هنوز بتوانم کدی بسیار بهتر و مقاوم‌تر بنویسم، اما اگر انجامش صد برابر بیشتر از زمان لازم طول بکشد، احتمالاً دیگر واقعاً بلد نیستم چطور این کار را بکنم.

فرض کن چیزی یاد می‌گیرم و سال‌ها از زندگی‌ام را صرف فهمیدنش می‌کنم. بعد آن را به زبـان سـاده بـرای نسل‌های جوان‌تر و آدم‌های دیگر توضیح می‌دهم و آموزش می‌دهم. آن‌ها می‌توانند از جایی که من متوقـف شده‌ام شروع کنند. ذهنشان را با داده‌های نامربوط پر نکرده‌اند و می‌توانند از نقطهٔ بسیار جلـوتر آغـاز کننـد. دلیل اینکه می‌توانی یک زبان را این‌قدر روان صـحبت کـنی این اسـت کـه طی سـال‌ها کامـل و منتقـل شـده است. لازم نیست برای ارتباط با آدم‌های اطـرافت این‌همـه وقت صـرف سـاختن مفهوم‌هـا و کلمه‌هـای تازه کنی.

به همین شکل، اگر یاد بگیرم و تمـرین کنم چطـور نرم‌افـزار بهـتری بسـازم، عرف‌هـا به‌مـرور تغییر می‌کننـد. دانشی که امروز اصلاً از آن خبر ندارم بعدها تبدیل به حافظهٔ عضـلانی می‌شـود. حـتی اگـر بـازار بـابتش پول ندهد، چیزی عوض نمی‌شود، چون بهره‌وری من بیشتر می‌شود.

این پارادوکس اسـتفاده از هـوش مصـنوعی اسـت. فکـر می‌کـنیم بهره‌وری‌مـان بیشـتر شـده، امـا اگـر همـهٔ بخش‌های این تجربه را حـذف کـرده باشـیم و فقـط در حـال تولیـد باشـیم، در نهـایت دانشـمان را از دسـت می‌دهیم. حتی اگر آن را از دست ندهی، تسلطت بر آن کم می‌شود؛ نقص‌هایت بیشتر می‌شـوند، تیزبینـی‌ات را از دست می‌دهی و این وضعیت وقتی روی هم جمـع شـود بـه کـاهش بهـره‌وری می‌انجامـد. تِـرِنس تائو (Terence Tao)، شاید بزرگ‌تـرین ریاضـی‌دان زنـده، ویـدئویی دارد کـه در آن دربـارهٔ مـاهیت پارادوکسـیکال

استفاده از هوش مصنوعی برای کار حرف می‌زند. شدیداً پیشنهاد می‌کنم تماشایش کنی.

قانون مور

چیزی وجود دارد به‌نام قانون مور. Gordon Moore گفت تعداد ترانزیستورهای روی تراشهٔ ریزپردازندهٔ کامپیوتر تقریباً هر دو سال دو برابر می‌شود. یعنی اندازهٔ تراشه‌ها و CPUها مدام کوچک‌تر می‌شود. این قانون تا حدود سال ۲۰۱۳ همچنان برقرار بود و آن زمان تخمین زده شد که منسوخ می‌شود. اما گروه درخشانی از دانشمندان موفق شدند ناممکن را ممکن کنند و فناوری تازه‌ای اختراع کنند که قانون مور را نجات داد. اگر تلاش مداوم دانشمندان در طول تاریخ و سرمایه‌گذاری شرکت‌های سخت‌افزاری (که به محصولاتشان نیاز داشتند) نبود، این فناوری را نداشتیم. تصور کن هیچ‌کس روی آن سرمایه‌گذاری نکرده بود و رکود ادامه پیدا می‌کرد تا مقاله‌های چنددهه‌پیشی که این فناوری را ممکن کردند فراموش می‌شدند یا، بدتر از آن، به‌نحوی حذف می‌شدند (کتابخانهٔ اسکندریه را فراموش نکن). این دانش به‌راحتی می‌توانست مثل بسیاری از فناوری‌های دیگر در تاریخ گم شود.

آخرالزمان

خب، قبلاً درباره‌اش حرف زدیم. به‌نظر من هیچ مانع نظری‌ای جلوی آخرالزمان هوش مصنوعی نیست. چه می‌دانیم؟ شاید در طول عمرمان اتفاق بیفتد. دیگر تجربهٔ نهاییِ نسل زد همین است.

پیش‌بینی خوب

اگر پیش‌بینی‌های بد از آب درنیایند، عصر طلاییِ توسعهٔ مستقل ــ مخصوصاً در حوزه‌هایی مثل بازی‌سازی که ورود به آن‌ها سخت است ــ تبدیل به عصر پلاتینی می‌شود. چیزهای خوبی که قبلاً ساختنشان ناممکن بود، حالا ساخته می‌شوند. چیزهای خوبی هم که از قبل وجود داشتند بهتر می‌شوند. هوش مصنوعی همین حالا هم این صنعت را دموکراتیزه کرده و باز هم می‌کند.

فصل ۵

هوش مصنوعی برای خلاقیت

احتمالاً این فصل حسابی بحث‌برانگیز می‌شود. بیشتر وقت‌ها وقتی مردم به جنبه‌های منفی هوش مصنوعی فکر می‌کنند، فقط همین حوزه به ذهنشان می‌آید. پس سعی می‌کنم تا جایی که می‌توانم توضیح بدهم. اگر به کارت نمی‌آید، راحت از آن بگذر.

اول بگویم که با اینکه کار اصلی‌ام برنامه‌نویسی است، مدت زیادی هم هست که هنر کار می‌کنم. همین حالا هم دارم طراحی گرافیک می‌خوانم و شغل‌های شرکتی‌ام هم در همین حوزه بوده‌اند. دو سال است هنر دیجیتال کار می‌کنم ــ و ادامه‌اش می‌دهم ــ و از بچگی هم شکل‌های دیگری از هنر دیجیتال و فیزیکی انجام داده‌ام. برای همین در حرف‌زدن دربارهٔ این موضوع به‌اندازهٔ کافی اعتمادبه‌نفس دارم.

اما هم‌زمان، چون هنرمند سینیوری نیستم ــ خودم را فقط طراح گرافیک جونیور می‌دانم ــ نمی‌خواهم وارد جزئیات فنی عمیق شوم. همان‌طور که ابتدای فصل ۳ گفتم، می‌توانی حرف‌هایی را که زده‌ام یا خواهم زد به تخصص خودت ربط بدهی و ببینی در حوزهٔ کاری تو چطور کاربرد پیدا می‌کنند.

مالکیت

یکی از رایج‌ترین بدفهمی‌هایی که دیده‌ام به همین موضوع مربوط است.

با اینکه می‌دانم بیشتر مردم با این دیدگاه موافق نیستند، باید بگویمش چون فکر می‌کنم حقیقت دارد. واقعیت این است که اخلاق فقط مجموعه‌ای از قراردادهاست. شاید نتیجه بگیری اخلاق آن‌قدرها هم مهم نیست و باید بی‌خیالش شویم. اما من کاملاً مخالفم. این تعریف از اخلاق حس ما را نسبت به آن عوض نمی‌کند. حسی که دربارهٔ اخلاق داریم بر پایهٔ فایدهٔ آن است؛ بگذار توضیح بدهم.

قدیم‌ترها، پیش از شکل‌گرفتن جوامع و اجتماعات یا هر نوع رابطهٔ صمیمانه، هرکسی تنهای تنها بود. باید نه‌فقط میان حیوانات وحشی، بلکه میان آدم‌های دیگر هم زنده می‌ماندی. این روش به‌صرفه نبود. از چنین وضعی سودی نمی‌بردیم. پس به هم‌نشینی و همراهی رو آوردیم. من برایمان غذا می‌آورم و در برابر خطرها از ما محافظت می‌کنم؛ تو مراقب خانه و بچه باش. اگر با هم همکاری کنیم بیشتر از وقتی که تنها دنبال

منافع خودمان برویم سود می‌بریم. مفهوم «اعتماد» را خیلی منطقی این‌طوری ساختیم. به هم اعتماد می‌کنیم چون می‌خواهیم سود بیشتری ببریم. این همکاری کم‌کم تبدیل شد به چیزی که امروز به آن «جامعه» می‌گوییم. در جامعه بیشتر از وقتی سود می‌بریم که هرکس دنبال منفعت خودش باشد. سود همگانی به سود هر فرد تبدیل می‌شود. دلیل اینکه با یک سرماخوردگی ساده نمی‌میریم این است که همکاری کرده‌ایم: پزشک‌ها برای جامعه دارو می‌سازند، جامعه به آن‌ها پول می‌دهد و آن‌ها هم می‌توانند لپ‌تاپی خفن بخرند و الی آخر.

اخلاق جوهرهٔ جامعه است. مجموعه‌ای از قانون‌ها و قراردادها ساختیم تا جامعه‌مان را حفظ و تقویت کنیم. هر کار غیراخلاقی مستقیم یا غیرمستقیم **با مفهوم جامعه تناقض دارد** و درنتیجه به جامعه آسیب می‌زند. اگر کسی را بکشی و مجازات نشوی، بقیه نمی‌توانند احساس امنیت کنند؛ پس اول حمله می‌کنند و ناگهان همه‌چیز به هرج‌ومرج می‌کشد. اگر به کسی دروغ بگویی شاید به‌نظر نرسد به جامعه آسیب زده‌ای، اما طرز فکر تو مهم است. این دروغ‌ها و کارهای غیراخلاقی ممکن است در گذر زمان فاجعه‌ای به بزرگی چرنوبیل بسازند. برای همین هم روی عبارت «با مفهوم جامعه تناقض دارد» پافشاری کردم؛ بند بعدی بیشتر توضیحش می‌دهم:

از نظر من، کافی است معنی اخلاق را *بدانی*؛ اما برای استدلال‌کردن لازم نیست همیشه مستقیم به آن فکر کنی. برای فهمیدن اینکه کاری اخلاقی است یا نه، لازم نیست خط مستقیمی بین آن کار و پایداری جامعهٔ خودت پیدا کنی. دلیل اینکه بیشتر آدم‌ها وجدان دارند این است که نیاز زیستی به اخلاق و زنده‌نگه‌داشتن جامعه در وجودشان جا افتاده؛ درست مثل گرایش طبیعی به تشکیل خانواده و بچه‌دارشدن. می‌توانی از این حس و درک پایه‌ای استفاده کنی و بیش‌ازحد به ایدهٔ دشوارِ وصل‌کردن هر عمل به آسیبی مستقیم و آشکار برای جامعه تکیه نکنی.

در نهایت، حس‌هایی که دربارهٔ اخلاق داری معتبرند. من فقط دلیلی برایشان پیشنهاد کردم.

این موضوع، موضوع بسیار بزرگی است که قرن‌ها درباره‌اش بحث کرده‌اند و قصد ندارم عمیق‌تر واردش شوم. پس جمع‌بندی‌اش کنیم و ربطش بدهیم به حق نشر (کپی‌رایت).

مالکیت هم درست مثل اخلاق، به‌خودیِ خود معنایی ندارد. قراردادی تمام‌عیار است. وقتی می‌گویی «این قاشق مال من است»، دلیلش این نیست که همه‌چیزش را از صفر ساخته‌ای. فلزش را از هیچ خلق نکرده‌ای؛ پیش از تو در طبیعت وجود داشته. حتی قاشق را هم خودت نساخته‌ای؛ آن را از شرکت خریده‌ای.

امـا بـاز هم خـودت را مالـک واقعی‌اش می‌دانی، فقـط چـون پولش را داده‌ای (گـاهی هم در رسـتوران پول می‌دهی کـه قاشـق را موقتاً اسـتفاده کـنی). همـان‌طور کـه پول فقـط یـک قـرارداد اسـت، این مـالکیت هم قراردادی است. اگر قاشق را از تو بدزدم، صرفاً چون در دستم است به طرز جادویی صـاحبش نمی‌شـوم. دزد حساب می‌شوم و تو هنوز صاحبش هستی. این قانون را سـاختیم چـون بـدون آن جامعـه از هم می‌پاشـید. یادت باشد وقتی توی جنگل بودی هم به‌اصطلاح روی وسایلت «مالکیت» داشـتی؛ امـا هرکسـی می‌توانسـت آن‌ها را بردارد و کاری هم از دستت برنمی‌آمد. برای جلوگیری از همین وضع جامعه و قانون‌هـای مـالکیت را ساختیم. معنی مالکیت را از نو تعریف کردیم؛ فقط یک قرارداد است.

سبک هنری و مالکیت فکری

همین موضوع دربارهٔ مالکیت‌های فکری هم صدق می‌کند. حقوق هنری با انواع دیگر مالکیت فرقی ندارند.

حالا باید یک واقعیت ساده را بگویم: «سبک هنری» مشمول حق نشر نیست. نمی‌توانی از کسی شکایت کنی چون سبک هنری‌اش مثل توست. نمی‌تـوانی از کسـی شـکایت کـنی چـون از همـان پـالت رنـگ تـو اسـتفاده می‌کند. این موضوع تازه‌ای هم نیست؛ در تمام دسته‌های هنری از قدیم همین‌طور بوده.

هرچه می‌خواهی بگـو، این واقعیت بـه دلیلی وجـود دارد. راه قابل‌اعتمـادی نـداریم کـه بفهمیم اصـلاً سـبک چیست، چرا باید «مال تو» باشد یا آن را از کس دیگری گرفته‌ای یا نه. فرض کن می‌توانسـتی به‌نحـوی سـبک هنری را نوعی مالکیت فکری کنی؛ تقریباً همهٔ هنرمندان فعلی از نظر قانونی بـه خطـر می‌افتادنـد. تصـور کن تمام حقوق اثرت را به شرکتی بفروشی و حالا آن سبک مال آن‌ها باشـد، پس دیگـر نتـوانی از سـبکی کـه طی سال‌ها صیقل داده‌ای استفاده کنی.

حتی اگر عمدی هم در کار نباشد، باز سبک تو شـبیه سـبک هنرمنـدهای دیگـری اسـت. نمی‌تـوانی ادعـا کـنی سبک هنری‌ات «منحصربه‌فرد» است. دست‌کم دیگر هیچ سـبک هـنریِ منحصـربه‌فردی وجـود نـدارد. یـادم هست پیش از هوش مصنوعی هر هنرمند خـوبی می‌گفت (و بعضی‌هایشـان هنـوز هم می‌گوینـد) هنرمنـد خوب، دزد خوبی است: آن‌قدر از آدم‌های مختلفی می‌دزدی تا هیچ‌کس نتوانـد تشـخیص دهـد از چـه کسـی دزدیده‌ای. این هم همین است.

تنها چیزی که می‌توانی ادعا کنی اثر هنری خودت است. می‌تـوانی نقاشـی کسـی را کـپی کـنی بی‌آنکـه سـبک هنری‌اش را کپی کنی، و باز هم از تو شکایت کنند. شاید بپرسی چطور ممکن است اثر هنری را بـدون سـبک

هنری‌اش کپی کرد؟ جوابش عوض‌کردن مدیوم هنری (رسانه یا ابزار مادیِ خلق اثر) است. پروندهٔ مشهوری هست به‌اسم پروندهٔ حق نشر لوکزامبورگ علیه Jeff Dieschburg. یک نقاش اثر هنری یک عکاس را نقاشی کرد. حتی کپیِ موبه‌مو هم نبود، اما آن‌قدر واقعی و شبیه بود که بتوان آن را کپی تشخیص داد؛ همین باعث شد پای شکایت به میان بیاید. حتی می‌توانی واکنش‌های مردم را همان صفحه بخوانی و ببینی آدم‌ها چقدر می‌توانند بی‌ادب باشند. قضاوتت را بر اساس «مقدار زحمتی» که کشیده شده نگذار. حتی اگر پنجاه سال صرف کپی‌کردن اثر کسی کنی، باز هم حق نشر را نقض کرده‌ای (مگر اینکه اجازه داشته باشی).

تناقض

فرض کن اثری هنری ساخته‌ای و آن را به کسی فروخته‌ای. آیا آن شخص می‌تواند اثر تو را اسکن کند و به‌عنوان اثر خودش در اینترنت بگذارد؟ معلوم است که نه؛ اجازه ندارد اثر تو را بفروشد و بگوید سازنده‌اش خودش بوده است. حتی لازم نیست ادعا کند اثر را ساخته؛ در حالت عادی اصلاً اجازه ندارد اسکن آن را در اینترنت بارگذاری کند. دربارهٔ چیزهای دیگر هم همین‌طور است. نمی‌توانی کتابی را که خریده‌ای اسکن کنی و در اینترنت بگذاری؛ چه پولی باشد چه رایگان.

فکر می‌کنی دلیلش چیست؟ شاید فکر کنی این کار به اعتبار و بازار هنرمند آسیب می‌زند و رقابت را ناعادلانه می‌کند. اما چرا می‌توانی مثلاً قاشقی را اسکن و بفروشی؟ حتی دربارهٔ کتاب هم می‌توانی آن را در بازار آزاد دوباره بفروشی. آیا این کار به اعتبار، شرکت و بازار آسیب نمی‌زند؟

اگر به‌اندازهٔ کافی درباره‌اش فکر کنی، خیلی از این تناقض‌ها پدیدار می‌شوند. می‌فهمی قانون‌های حق نشر با هم تفاوت زیادی دارند، اما حتی حس نمی‌کنی ناعادلانه باشند. طبیعی به‌نظر می‌رسند، با اینکه چندان منسجم نیستند.

دلیلش همان چیزی است که در این فصل توضیح دادم: مالکیت مفهومی ساختگی است که برای حفظ جامعه به آن نیاز داریم. چیزی نیست که ذاتاً به تو داده شده باشد؛ با عضویت در جامعه به دستش می‌آوری (کارهای مجرمانه تو را از عضویت در جامعه محروم می‌کنند و برای همین حقوقت را از دست می‌دهی). این‌طور نیست که همهٔ قانون‌های مالکیت بی‌نقص باشند؛ آن‌ها زیاد تغییر می‌کنند.

پس چه چیزی باعث می‌شود فکر کنی هنر هوش مصنوعی هنر واقعی نیست؟ فقط چون بیشتر بخش‌هایش خودکار شده‌اند؟ هنر مولد چه (generative art)، با هنر هوش مصنوعی مولد اشتباه گرفته

نشوند)؟ هنر کامپیوتری چه؟ این حوزه از دههٔ ۱۹۶۰ وجود داشته است. همان‌طور که برنامهٔ Notepad اثر هنری نیست اما گوسالهٔ طلایی هست، و درِ چوبی اثر هنری نیست اما آینهٔ چوبی هست، تصویر ارزان و تولیدشده با هوش مصنوعی اثر هنری نیست، اما تصویری که با دقت ساخته شده باشد می‌تواند اثر هنری باشد.

هوش مصنوعی و سرقت

اگر قبول کنیم خودِ سبک هنری از حق نشر حمایت نمی‌شود، به‌راحتی می‌توانیم ببینیم مدل‌های زبانی بزرگ «بر پایهٔ سرقت» نیستند. وقتی بفهمی حتی نمی‌توانی وجود روح را ثابت کنی ــ همان‌طور که در فصل ۲ گفتیم ــ این موضوع روشن‌تر هم می‌شود.

می‌فهمم که همه نه دربارهٔ روح و نه، عجیب‌تر از آن، دربارهٔ سبک با من موافق نیستند. اما باید قبول کنیم که «سرقت» اتهام بسیار سنگینی است. اگر می‌خواهی کسی را دزد بنامی، باید خیلی مطمئن‌تر از این حرف‌ها باشی. متهم‌کردن آدم‌ها به کارهای وحشتناک خطرناک و غیراخلاقی است. مثل اتهام دروغینِ تجاوز است؛ فقط کمی کم‌ویرانگرتر. شاید بعضی‌ها بگویند فقط مدل‌های زبانی بزرگ را دزد می‌نامند، اما با این کار همهٔ این آدم‌ها را هم حامی سرقت می‌خوانند: سازندگان مدل‌های زبانی بزرگ، شرکت‌هایی که مالکشان هستند و کارمندهایشان، کسانی که از رابط برنامه‌نویسیِ مدل‌های زبانی بزرگ برای ساخت نرم‌افزار به‌عنوان خدمت (SaaS) استفاده می‌کنند، کسانی که از آن‌ها برای تولید متن، کد، تصویر یا هرچیزی به هر دلیلی استفاده می‌کنند، کسانی که مشتاقانه درباره‌شان حرف می‌زنند و دیگران را به استفاده از آن‌ها دعوت می‌کنند، و حتی کسانی که مدل‌های زبانی بزرگ را روی دستگاه خودشان اجرا می‌کنند. اگر فکر می‌کنی دارم اغراق می‌کنم، شاید یکی از رایج‌ترین جرم‌ها، یعنی دزدی، را دست‌کم گرفته‌ای. نمی‌خواهی فقط به‌خاطر اینکه به اشباح باور داری، آدم‌ها را به جرم‌های کیفری متهم کنی، نه؟

هیچ مدل زبانی بزرگی از نظر اخلاقی ساخته نشده است

مدل‌های زبانی بزرگ اساساً حاصل آموزش‌دادن ماشین با حجم زیادی از ورودی‌اند. مثلاً ماشین را با ده میلیون تصویر از خودرو آموزش می‌دهی و بعد مدلی زبانی داری که می‌تواند تصویر دقیق خودرو تولید کند. مدل زبانی بزرگ آن تصاویر را در خود نگه نمی‌دارد؛ فقط الگوها را ذخیره می‌کند و می‌توان از آن الگوها به هر شکلی برای تولید هر چیزی از صفر استفاده کرد.

می‌توانی مدل زبانی بزرگی را با آثار خودت آموزش بدهی تا با سبک تو هنر بسازد، اما برای اینکه آن مدل از همان ابتدا مفید باشد باید با انبوهی از تصاویر دیگر آموزش دیده باشد. برای فهمیدن معنای آثار تو به تمام آن الگوها نیاز دارد. مگر اینکه میلیون‌ها اثر هنری کشیده باشی، آثارت به‌تنهایی فایده‌ای برای آموزش ندارند. پایه، آموزش مدل با داده‌های ازپیش‌موجود است. اگر این کار از نظر تو دزدی است، پس هیچ مدل زبانی بزرگی از نظر اخلاقی ساخته نشده است. مثل این است که با برده‌داری سرطان را درمان کنی. حتی اگر با برده‌داری سرطان را درمان کرده باشی، این کار تو را کم‌تر غیراخلاقی نمی‌کند. یادت باشد بر اساس نتیجه قضاوت نمی‌کنی.

نمی‌تواند سرقت باشد

همان‌طور که توضیح دادم، مدل‌های زبانی بزرگ با آموزش‌دیدن روی داده‌های دیگران ساخته می‌شوند. اگر از آن دسته آدم‌هایی هستی که فکر می‌کنند این موضوع ثابت می‌کند سرقت رخ داده، یک سؤال از تو دارم: فکر می‌کنی مغز انسان چطور کار می‌کند؟ چرا فکر می‌کنی مغز ما اساساً همین کار را نمی‌کند؟ برای پیشرفت باید همین کار را بکنی. «مطالعهٔ هنری» همین است: چیزها را کپی می‌کنی، از شاهکارها گرفته تا اشیا و چیزهای دیگر. اگر مطالعه‌ات از نوع کپی‌کردن از استاد باشد، سعی می‌کنی اثر هنری را کپی کنی و هم‌زمان حواست باشد چرا آن هنرمند فلان کار مشخص را کرده است. الگوها را یکی‌یکی در ذهنت حک می‌کنی. در انواع دیگر مطالعه هم همین کار را تکرار می‌کنی. تمام اساس درس‌خواندن یادگرفتن الگوهاست. تصویر را در ذهنت ذخیره نمی‌کنی که هر بار آن را کپی کنی؛ یاد می‌گیری آن تصویر چطور ساخته شده است. شاید حتی بتوانی آن اثر هنری را کپی کنی، اما این به معنی دزدیده‌شدن دانشت نیست.

سرقت مفهومی کاملاً مشخص است. اگر چاقویی به تو بفروشم و با آن کسی را بکشی، من مسئول نیستم. حتی اگر با آن چاقو به ۱۰۱ روش مختلف به آدم‌ها آسیب بزنی، باز هم شریک جرم تو نیستم، چون چاقو هدف مشخصی دارد. کشتن فقط یکی از کاربردهای آن است و ربطی به خود چاقو ندارد؛ آدم می‌تواند با دست خالی هم بکشد. به همین شکل، می‌توانی از هوش مصنوعی برای کپی‌کردن اثر هنری کسی و کمی تغییردادنش استفاده کنی، اما این به معنی مقصر بودن ابزار نیست؛ شخصی که این کار را انجام می‌دهد مقصر است. همین حالا هم در یوتیوب کلی ویدئو دربارهٔ حذف واترمارک از تصویر وجود دارد. مردم از زمان عرضهٔ Photoshop از آن برای جعل اسناد و مهرهای مهم استفاده کرده‌اند. پس چرا همین حس را نسبت به آن نداری؟ فقط چون انجامش سخت‌تر است؟ می‌توانم استدلال کنم جعل سند با Photoshop حتی آسان‌تر و امن‌تر است. اگر مدل‌های زبانی بزرگ ذاتاً دزدند، پس Photoshop هم قطعاً ذاتاً دزد است.

هنر هوش مصنوعی

اگر از مسئله‌های اخلاقی عبور کنیم، هنوز هم بحث‌های زیادی هست دربارهٔ اینکه هنر واقعی چیست و چـرا بعضی‌ها هنرِ هوش مصنوعی را هنر نمی‌دانند.

هنر دو بخش دارد:

1. **بخش خلاقانه**: ایدهٔ پشت اثر، معنایش، حسی که می‌خواهی برانگیزی و چیزهای دیگر.
2. **بخش فنی**: ساخت اثر، استفاده از ابزارهای هـنری و دانش علمی دربـارهٔ ترکیب‌بنـدی، نظریـهٔ رنـگ، نحوهٔ ضربه‌زدن با قلم‌مو، استفاده از دست‌ها و چیزهای دیگر.

شاید اصطلاح «فنی» را دوست نداشته باشی، مخصوصاً اگر بعنوان هنرمند شغلی نداشته‌ای، امـا بااین‌حـال درست است. فنی است چون فقط دانشی علمی یا روشی است که باید یاد بگیری و به‌کار ببری. بله، می‌توانی آن را با تصمیم‌های خلاقانه ترکیب کنی، اما در نهایت این دو بخش جدا هستند. اینکه چه چیزی می‌خـواهی بسازی و چطور می‌سازی‌اش، دو بخش از یک شغل‌اند (البته می‌توانند روی هم اثر هم بگذارنـد). شـاید ایـدهٔ فوق‌العاده‌ای داشته باشی اما ندانی چطور پیـاده‌اش کـنی. در عین حـال، شـاید بخش فـنی را بلـد باشـی امـا خلاقیت لازم برای دانستن اینکه چه چیزی بسازی نداشته باشی (سبک هایپررئالیسم اساساً همین است).

حتی خودِ واژهٔ «فنی» از واژهٔ یونـانی باسـتان «techne» می‌آیـد کـه معنـایش هـنر بـود. آن زمـان هـنر فقـط پیشه‌ای بود. نقاش‌ها و مجسمه‌سازها فرقی با نجارها و آهنگرها نداشتند. برای سـاختن چیزهـا از قواعـدی پیروی می‌کردنـد. بـا اینکـه امـروز خیلی خلاق به‌نظـر می‌رسـند، آن روزهـا خلاق‌بـودن حـتی تابو بـود («تاریخ مفاهیم بنیادین زیبایی شناسی» اثر ووادیسواف تاتارکیویچ، ۱۹۸۰، فصل سوم: «تاریخ رابطهٔ هنر بـا شـعر»). با این حساب، پیوند را روشن می‌بینیم: بخش فنی جایی است که کارها را تا حد ممکن عینی انجام می‌دهی. لزوماً خلاق نیستی، اما بااین‌حال هنرمندی.

برای همین هوش مصنوعی فقط یک ابزار است. هوش مصنوعی هوشیار نیست. خودش نمی‌تواند چـیزی را برانگیزد. مدل زبانی بزرگ ماشینی الگومحور است و از الگوهای پرامپت‌هایت پیروی می‌کند. حتی اگـر بـه آن بگویی «یک نقاشی بکش»، باز هم برای ساختن به همان دستور نیاز دارد. اگـر چـیز دیگـری بگـویی، خـروجی عوض می‌شود. خروجی انسان لزوماً به همین شکل تغییر نمی‌کند.

پس بله، معلوم است می‌توانی فقط از هوش مصنوعی بخواهی چیزی بسازد و اسمش هـنر نباشـد. درسـت

مثل اینکه کسی مدادی بردارد و روی کاغذ خط بکشد؛ این لزوماً هنر نیست. هنر عامدانه و آگاهانه است و به خلاقیت نیاز دارد؛ چیزهایی که همه‌شان را خود هنرمند می‌تواند فراهم کند. همان‌طور که هنرمند هایپررئالیست هنوز هنرمند است، کسی که از هوش مصنوعی استفاده می‌کند هم می‌تواند هنرمند باشد. طبق منطق معمول، نفر دوم حتی بیشتر هنرمند است، چون کارش تماماً دربارهٔ خلاقیت است، نه چیزی که «هرکسی بتواند انجام دهد».

شاید متوجه شده باشی گفتم «فردی که از هوش مصنوعی استفاده می‌کند»، نه «هنرمند هوش مصنوعی». چون هوش مصنوعی سبک یا مدیوم هنری نیست. ویژگی اصلیِ یک مدیوم این است که آثار هنریِ متعلق به آن چیزی مشترک داشته باشند. وقتی می‌توانی با آن هر سبکی بسازی، منطقی نیست آن را یک دسته‌بندیِ سبکی بدانیم. می‌شود هنر هوش مصنوعی را دسته‌بندی کرد، اما فقط به‌عنوان دسته‌ای از ابزارها؛ درست مثل اینکه به کسی بگویی هنرمند Photoshop.

اختیار بر اثر هنری

یکی از بزرگ‌ترین دلیل‌هایی که هنرمندها ــ مخصوصاً تازه‌کارها ــ با ابزارهای هنریِ هوش مصنوعی کنار نمی‌آیند این است که روی خروجی احساس کنترل ندارند. فکر می‌کنند: «چیزهای تصادفی تولید می‌کند، نمی‌داند من چه می‌خواهم و فقط دستگاهی برای تولید اسلاپ و خروجی‌های بی‌روح است». بعد از همین استدلال استفاده می‌کنند تا ثابت کنند برای بقیه هم همین‌طور است و درنتیجه هنر واقعی نیست.

شاید با خودت فکر کرده باشی حرفم جور درنمی‌آید. دربارهٔ برنامه‌نویسی ــ کاری که آشکارا خیلی مکانیکی‌تر از هنر است ــ می‌گویم بر خروجی هوش مصنوعی اختیار نداری. اما دربارهٔ هنر، که چیزی بسیار شخصی و صمیمی است، می‌گویم هوش مصنوعی ابزار است و می‌توانی بر خروجی‌اش اختیار داشته باشی.

اول از همه باید بگویم بیشتر ــ اگر نگوییم همهٔ ــ جنبه‌های منفی‌ای که گفتم ممکن است به حوزه‌های دیگری از جمله هنر *به‌عنوان کسب‌وکار* هم مربوط باشند. من در صنعت هنر چندان حرفه‌ای کار نکرده‌ام. همیشه در حال یادگیری بودم و هنوز با کار حرفه‌ای فاصله داشتم. پس نمی‌توانم دربارهٔ آن حوزه ادعایی بکنم.

دوم، به‌نظر من ماهیت هنر تفاوتی دارد که اثر می‌گذارد: هنر ذاتاً تصادفی است. هر حرکت قلم‌مو با حرکت بعدی فرق دارد. جملهٔ معروفی هست که می‌گوید: «هر بار چیزی را نقاشی می‌کنی، اثر هنریِ متفاوت

ساخته‌ای». اساساً به همین خاطر است که آثار هنری سنتی این‌قدر گران‌اند. اثر اصلی را فقط یک بار می‌شود کشید و همین کمیابش می‌کند.

این تصادفی‌بودن برای بیشتر هنرمندها هنگام خلق اثر آشناست. اگر هنرمندی، از خودت بپرس: چند بار ایده‌ای در سرت داشته‌ای، شروع به ساختنش کرده‌ای و بعد دیده‌ای طور دیگری از آب درآمده؟ بیشتر وقت‌ها حتی تصویر واضحی از ظاهر نهایی‌اش نداشتی. در همان حال که از نظر فنی می‌ساختی‌اش، ایده را هم شکل دادی.

برای همین با تصادفی‌بودن در هنر هوش مصنوعی مشکلی ندارم؛ همان‌طور که هنرمند David Hockney هم نداشت (روحش شاد). نکته این است: آیا *تو* می‌توانی این تصادفی‌بودن را کنترل کنی تا به ایده‌ات نزدیک شوی؟ می‌توانی احساساتت را کنترل کنی و چون نتیجه با تصورت فرق داشت نگویی «آشغاله»؟ بعضی‌ها ادعا می‌کنند می‌توانند. ما از کجا بدانیم دروغ می‌گویند؟ فکر می‌کردم هنر مسئله‌ای شخصی است.

حتی می‌گویم همین ماهیت تصادفیِ هنر باعث می‌شود هوش مصنوعی برای این حوزه مناسب‌تر باشد تا حوزه‌ای علمی و حساس. مقیاس‌پذیری و نگهداشت‌پذیری در هنر آن‌قدر که در مهندسی نرم‌افزار مهم‌اند مهم نیستند؛ البته اگر اصلاً بتوانیم آن‌ها را در بعضی پروژه‌های هنری تعریف کنیم.

دشمن هم نیستیم

همه داریم راه‌های واردکردن هوش مصنوعی به حوزه‌های کاری‌مان را پیدا می‌کنیم؛ بیایید دربارهٔ قضاوت‌ها و نظرهای مختلف پذیراتر باشیم. در بعضی حوزه‌ها مثل بازی‌سازی، یکی از بزرگ‌ترین دلیل‌های سخت‌بودن ورود به بازار این بود که خود برنامه‌نویسی وقت زیادی می‌گرفت؛ چه رسد به ساخت آثار هنری که سال‌ها یادگیریِ متمرکز و پربازده می‌خواهد. بازی‌سازی سه‌بعدی برای بازی‌سازهای مستقل تقریباً غیرممکن بود، چون ورود به آن به‌شدت سخت بود. هوش مصنوعی کاری کرده آدم‌ها بتوانند چیزهایی بسازند که قبلاً واقعاً نمی‌توانستیم. چه درست باشد چه غلط، بیایید پیش از اینکه به هم برچسب بزنیم درباره‌اش به نتیجه برسیم.

نتیجه‌گیری

وقتی چیزی تأثیرهای بنیادینی بر یک حوزه می‌گذارد، باید بنیان‌های آن حوزه را زیر سؤال ببری. بازنگری صنعت (مخصوصاً صنعت فناوری) چیزی نیست که بتوان ساده از کنارش گذشت، چون ممکن است خیلی چیزها را نبینی؛ تو جزو اولین آدم‌هایی هستی که این گردش‌کار کاملاً تازه را امتحان می‌کنند. هوش مصنوعی ابزار بسیار مفیدی است، اما مفیدبودن به معنی امن‌بودن نیست؛ چاقو هم مفید است و هم‌زمان ناامن.

مفهوم «ارشدیت» همیشه به تجربه گره خورده است. اما حالا باید بنیان‌ها را زیر سؤال ببریم: تجربه چطور به دانش وصل می‌شود؟ اصلاً چه چیزی تجربه را به تجربه تبدیل می‌کند؟ فرق تجربه و خاطره چیست؟

یکی از چیزهایی که ارشدیت به ما برنامه‌نویس‌ها یاد داده این است که «اگر کار می‌کند، دستش نزن». شاید احمقانه به‌نظر برسد، اما فقط روش بامزه‌ای برای توضیح مفهومی بسیار مهم است: تغییرها خطرناک‌اند. هر بار چیزی را در سامانه‌ات تغییر می‌دهی، راه‌های تازه‌ای برای خراب‌کردن چیزها ایجاد می‌کنی. هرچه سطح تغییر بزرگ‌تر باشد، فرایند پرخطرتر می‌شود. اما حالا همه می‌خواهند توسعهٔ نرم‌افزار را با روشی کاملاً تازه انجام دهند؛ با ابزارهای غیرقطعی‌ای که توهم می‌زنند و فقط چند سال یا حتی چند ماه است از آن‌ها استفاده می‌کنیم. این شدیدترین تغییری است که تا امروز دیده‌ایم، اما بعضی‌ها تعجب می‌کنند که چرا نگرانش هستی؛ انگار ارشدیتی که تمام این مدت به دست آورده‌ای خودش یک توهم بوده است.

هوش مصنوعی قطعاً توانمند است. کمکمان کرده به چیزهایی فراتر از خیالمان برسیم و من به‌هیچ‌وجه مخالفش نیستم. اما استفاده از هوش مصنوعی به‌عنوان ابزاری تازه یک چیز است و استفاده از آن به‌عنوان راهی برای قمارکردن با آینده‌ات چیز دیگری. این... چیزی که هوش مصنوعی انجام می‌دهد اصلاً شبیه ابزارهایی نیست که می‌شناختم. بیشتر شبیه کارمند مستی است که هر چند ساعت یک بار همه‌چیز را فراموش می‌کند.

مهندسی نرم‌افزار جزو شغل‌هایی است که بیشترین تغییر را کرده‌اند. ما برنامه‌نویس‌ها در مقایسه با بیشتر حرفه‌ها به تغییرهای صنعت خودمان عادت بیشتری داریم. با جایگزین‌کردن «ابزارمان» با «ابزارهای جدیدتر» اصلاً مشکلی نداریم. بااین‌حال گفت‌وگوها دربارهٔ هوش مصنوعی در حوزه برنامه‌نویسی بسیار پرسروصدا است. حتی انعطاف‌پذیرترین آدم‌ها هم با این هجوم ناگهانی تغییرها کنار نمی‌آیند و تقریباً همه

ــ از جمله کسانی که هوش مصنوعی را با تمام وجود در شغلشان پذیرفته‌اند ــ می‌گویند کاری که یک سال پیش می‌کردند حالا به‌مراتب لذت کمتری دارد. سرعت دائمیِ همه‌چیز واقعاً طاقت‌فرساست.

شاید اشتباه می‌کنم. شاید کل این کتاب فقط مرحلهٔ چانه‌زنیِ پنج مرحلهٔ سوگ من باشد. اما الان دقیقاً بهترین زمان برای مهندس نرم‌افزاربودن نیست. حتی اگر خودت تکلیف هوش مصنوعی را روشن کنی ــ همان‌طور که حس می‌کنم من تازه کرده‌ام ــ باز هم واقعیت اطرافت یا انتظارهای بازار کار از تو را عوض نکرده‌ای. پیمانکار برایش مهم نیست از چه ابزاری استفاده می‌کنی؛ اما از تو می‌خواهد محصول را در زمانی آن‌قدر کوتاه تحویل بدهی که بدون واگذارکردن همه‌چیز به هوش مصنوعی عملاً انجام‌دادنش ناممکن است.

شغلت از قبل هم پراسترس بود، حالا می‌خواهند مجبورت کنند تمام معنا و لذتش را کنار بگذاری. اگر بخواهی شغلت را نگه داری، باید با سیاست‌های تازه‌شان کنار بیایی؛ سیاست‌هایی که بی‌آنکه خودت بدانی جوری چیده شده‌اند که شکست بخوری. این سیاست‌ها مجبورت می‌کنند فراموش کنی چه‌کسی هستی، زیر پرامپت و تیکت دفنت می‌کنند، نمی‌گذارند حس بهره‌وری داشته باشی و باعث می‌شوند حالت غرقگی (flow state) را که قبلاً داشتی از دست بدهی. خنده‌دار اینجاست که ماندن در این وضعیت قطعاً تخصصت را هم پایین می‌آورد و ترس از جا ماندنت را بیشتر می‌کند. ولی اگر این را بگویی، فقط توصیه می‌کنند «برو یاد بگیر» و هیچ‌وقت نمی‌گویند چطور یا چه چیزی یاد بگیری.

ما فقط می‌خواستیم برنامه‌نویس‌های خوبی بشویم؛ در کارمان بهتر شویم، به برنامه‌نویس میدل‌لول، سینیور، سینیورِ بهتر، مهندس ارشد و الی آخر تبدیل شویم. اما حالا همه‌چیز معنایش را از دست داده است. مدام در حال تولیدی، بی‌آنکه حس پیشرفت داشته باشی. نمی‌دانیم چه چیزی یاد بگیریم. در این مرحله، قرار دادن خودت در جایگاهی که اصول را یادبگیری، وقتی می‌دانی هوش مصنوعی می‌تواند کاری را که تو به‌زور در تلاش برای یادگرفتنش هستی در چند ثانیه انجام دهد، به‌شدت سخت و بی‌معنی به‌نظر می‌رسد.

چه کسی می‌داند چه اتفاقی می‌افتد؟ این دیوانه‌وارترین دهه‌ای است که تجربه می‌کنیم؛ از پاندمیِ کووید شروع شد و یکراست وارد عصر هوش مصنوعی شدیم. و اگر بخواهم منصف باشم، این آغاز دورهٔ تازه‌ای است: عصر فرسودگی شغلی.