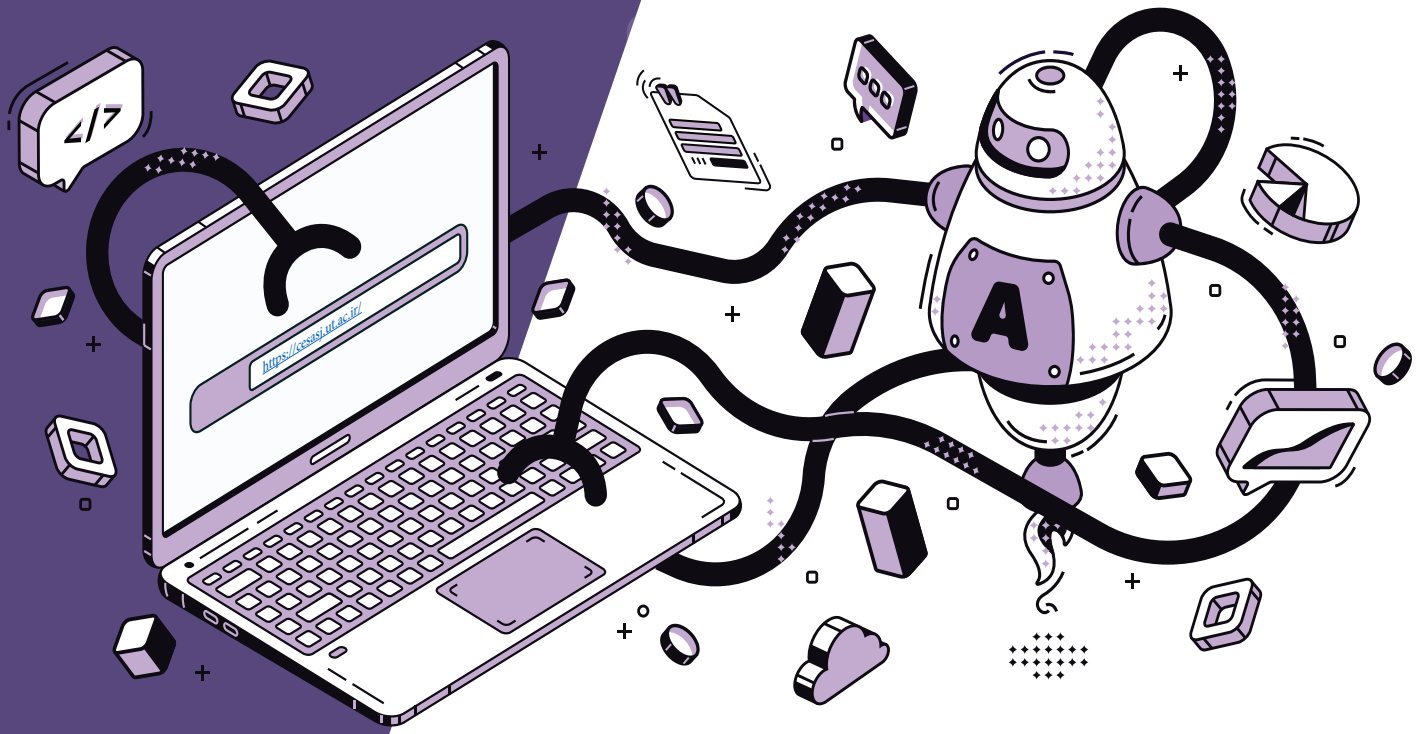
دنیای صفر و یک

00010101

شماره بیست و یکم/ شهریور ماه ۱۴۰۵

نشریه تخصصی انجمن علمی مهندسی کامپیوتر دانشکدگان فارابی دانشگاه تهران

در این شماره خواهید خواند:



دنیای صفر و یک

شناسنامه

دانشگاه صادرکننده مجوز:

دانشگاه تهران

صاحب امتیاز:

انجمن علمی مهندسی کامپیوتر دانشکدگان فارابی دانشگاه تهران

مدیر مسئول:

محدثه حاتمی‌کیا

سردبیر:

سبحان ولی‌زاده درخشان

استاد مشاور علمی:

دکتر حسین آقابابا

کارشناس نشریات:

مریم شریفی دانا

زمینه انتشار:

علمی، تخصصی

تاریخ / ترتیب انتشار:

۱۴۰۵/۶/۳۱ - دو ماهنامه

طراح جلد و صفحه آرا:

زهرا پورفرزین، نرگس رضاییان

ویراستار ادبی:

مهراد پوریوسف

هیئت تحریریه:

مهراد پوریوسف، مشکات سلیمی، علیرضا کامرو،
عرفان رمضانی مقدم، احمد محمدی، فاطمه نصیری،
رضا قمی‌الاصل، ماهان قدیم‌خانی

نویسندگان:

مهرناز حسینی، یاشار ابری، محدثه حاتمی کیا،
ابوالفصل توکلیان، سیده الهه دهقان، سبحان ولی‌زاده درخشان

راه ارتباطی:

donyayesefroyek.ut@gmail.com

http://cesasj.ut.ac.ir @donyayesefroyek

فهرست

سخن آغازین

به نام خداوند قلم

سلام به رفقای عزیز و همیشگی دنیای صفر و یک!

شماره قبلی که منتشر شد، با کلی ذوق و شوق از شروع کار گفتم؛ از اینکه چقدر وسط کلاسا، پروژه‌ها و شلوغی‌های معمول دانشجویی، هماهنگ کردن کارا سخته اما دلمون به همین کنار هم بودن‌ها خوشه. اون روزا فکر می‌کردیم نهایت چالش کارمون کمبود وقت و خستگی امتحانات باشه. اما واقعیت اینه که فاصله بین شماره بیست و بیست‌ویک، شبیه هیچ فاصله‌ای توی تقویم نبود.

زمستونی که پشت سر گذاشتیم، سرمایی داشت که تا عمق استخون همه‌مون نفوذ کرد. روزهایی اومد و گذشت که سنگینی و تلخیش هنوز روی سینه تک‌تکمون مونده؛ روزهای بهت، روزهای غم عمیقی که توی سکوت کشیدیم، و از همه دردناک‌تر، صندلی‌های خالی سر کلاس‌ها و راهروهای دانشگاه که دیگه هیچ‌وقت با صاحباشون پر نمی‌شن. بعدش هم روزهای تلخ دلهره، سایه سنگین ناامنی و جنگ، و ماه‌های طولانی که حتی دستمون به ساده‌ترین راه‌های ارتباطی هم بند نبود؛ انگار توی یه سکوت و بی‌خبریِ مطلق پرتاب شده بودیم و جز خودمون و دردهامون هیچ‌کس نبود.

راستش رو بخواید، توی این مدت بارها از خودمون پرسیدیم: «اصلاً الان نشریه درآوردن چه معنایی داره؟» وسط این‌همه دل‌شکستگی، وقتی آدم دل و دماغ هیچ کاری رو نداره، صفحه‌بندی کردن چندتا مطلب و حرف زدن از تکنولوژی و کامپیوتر مگه دردی رو دوا می‌کنه؟

اما تهش دیدیم اگه الان ننویسیم و کنار هم جمع نشیم، یعنی تسلیم سکوت شدیم و گذاشتیم فراموشی همه‌چیز رو با خودش ببره. این شماره شاید اون سرخوشی و دل‌خوشی‌های قبلی رو نداشته باشه، اما شد یه بهونه تا نشون بدیم رفاقت‌هامون هنوز نفس می‌کشه؛ که بگیم هیچ‌کدوم از این روزها یادمون نرفته و حتی وسط این‌همه دل‌تنگی و جای خالی، هوای همدیگه رو داریم.

شماره بیست‌ویک با احترام عمیق و بغضی که پنهان‌کردنی نیست، تقدیم می‌شه به یاد همه اونایی که جاشون تا ابد بین ما خالیه، و به شما رفقایی که توی این ماه‌های سرد و بی‌ارتباط، صبوری کردید و دووم آوردید.

مرسی که هنوز هستید و چراغ این جمع رو روشن نگه داشتید. با چشم رفاقت ورقش بزنید و مثل همیشه برامون بنویسید. به امید روزهایی که آرامش و نور دوباره به دل‌هامون برگرده.

ارادتمند شما

سبحان ولی‌زاده درخشان

مصاحبه با دکتر زینب برهانی‌فرد

سبحان ولی‌زاده درخشان
دانشجوی مهندسی کامپیوتر
دانشکدگان فارابی دانشگاه تهران
sobhan.valizadeh@ut.ac.ir

۱۱ دقیقه

??? خانم دکتر برهانی‌فرد، با تشکر از وقتی که در اختیار ما قرار دادید. لطفاً خودتون رو به طور مختصر برای خوانندگان نشریه معرفی کنید و بفرمایید مسیر تحصیلی شما چطور طی شد؟

من دوره کارشناسی‌ام رو تو رشته مهندسی فناوری اطلاعات تو دانشگاه امیرکبیر گذروندم و مقطع ارشد رو هم تو همین رشته تو دانشگاه قم خوندم. بین دوره ارشد و دکتری یک وقفه تحصیلی ۶ ساله داشتم که تو این مدت، هم درگیر پروژه‌های صنعتی بودم و هم تو دانشگاه تدریس می‌کردم. بعد از این وقفه، به این نتیجه رسیدم که بهترین راه برای پیشبرد همزمان مسیر علمی و صنعتی، حضور تو دانشگاهه؛ برای همین دکتری مهندسی نرم‌افزار رو تو دانشگاه تهران ادامه دادم. اونجا حدود ۶ الی ۷ سال تو آزمایشگاه پردازش زبان طبیعی (NLP) عضویت داشتم و تمرکز اصلی‌ام روی حوزه‌های هوش مصنوعی، پردازش زبان طبیعی، سیستم‌های گفت‌وگو و دستیارهای هوشمند بود.

??? در حال حاضر تمرکز اصلی پژوهشی شما روی چه حوزه‌ایه و چی شد که این مسیر رو انتخاب کردید؟

مسیر پژوهشی من از همون ابتدا روی پردازش متن و زبان طبیعی (NLP) بوده. تو مقطع ارشد روی سیستم‌های توصیه‌گر کار می‌کردم، اما تو دوره دکتری ناگهان فضا متحول شد؛ یادگیری عمیق (Deep Learning) مطرح شد و با معرفی ترانسفورمرها و مدل زبانی BERT تو سال ۲۰۱۸، ما تو اوج تغییرات قرار گرفتیم. پژوهش‌های ما هم ناگزیر به سمت مدل‌های زبانی بزرگ (LLMs) کشیده شد، چون رقابتی بین غول‌های صنعتی دنیا و امکانات دانشگاهی شکل گرفته بود و ما باید تلاش می‌کردیم از این جریان عقب نمونیم.

??? تجربه درگیری شما با پروژه‌های عملی و صنعتی، چه تأثیری روی دیدگاهتون نسبت به فضای آکادمیک و تدریس گذاشت؟

به اعتقاد من، دستاورد اصلی دانشگاه صرفاً یادگیری ابزارها نیست، بلکه دانشگاه به ما یاد میده چطور فکر کنیم و چطور مسائل رو تحلیل و حل کنیم. اگه هدف فقط یادگیری چند تا ابزار باشه، شاید فردی که ۴ سال وقت خودش رو تو دانشگاه نگذرونده، ابزارهای

یک کلام حرف حساب

بیشتری هم بلد باشه؛ اما برای انجام کارهای عمیق تو صنعت، ما به مهارت حل مسئله نیاز داریم. ما تو دوره دکتری سعی کردیم مسائل واقعی جامعه و صنعت رو حل کنیم و به همین دلیل، با دوستانمون تو آزمایشگاه NLP یه استارتاپ و شرکت دانش‌بنیان تو حوزه هوش مصنوعی و دستیارهای سازمانی راه‌اندازی کردیم تا ردپای علمی خودمون رو تو صنعت هم پیاده‌سازی کنیم.

??? تو مباحث مهندسی نرم‌افزار، همون‌طور که خودتون هم سرکلاستون اشاره می‌کردید، همیشه تأکید زیادی روی مفاهیم اجایل (Agile) و اسکرام میشه. اما تو واقعیتِ بازار کار ایران، این متدولوژی‌ها چقدر اصولی اجرا میشن و چقدر از اون‌ها صرفاً یه ویترین دکوریه؟

واقعیت اینه که ما تو این زمینه پیشرفت خیلی خوبی داشتیم و تو شرکت‌های بزرگ نرم‌افزاری بالای ۵۰ نفر (مثل دیجی‌کالا، اسنپ و همکاران سیستم)، این روال‌ها تا حد خیلی خوبی دارن اجرا میشن. تیم‌های محصول قبل از شروع کدنویسی تمام نیازمندی‌ها، رابط کاربری و استوری‌ها رو کاملاً مشخص می‌کنن و این متدولوژی‌ها دیگه صرفاً یه شعار یا ویترین نیستن. البته به عنوان یه مهندس نرم‌افزار، ما باید تمام اصول رو بلد باشیم اما متناسب با شرایط، کوچکی و بزرگی پروژه و زمان ورود به بازار، تصمیم بگیریم که کدوم بخش‌ها رو استفاده کنیم و کدوم موارد دست‌وپاگیر رو حذف کنیم.

??? با توجه به پیشرفت سریع ایجنت‌های هوش مصنوعی که امروزه توانایی‌های یه برنامه‌نویس جونیور رو دارن، چه توصیه‌ای برای دانشجوهایی دارید که مشتاق ورود به بازار کار هستن؟ آیا این ابزارها جای اون‌ها رو می‌گیرن؟

کار کردن در کنار تحصیل تو رشته کامپیوتر اتفاق خوبیه، اما به شرطی که دچار مالتی‌تسکینگ (Multitasking) بیش از حد نشیم که به هیچ‌کدوم از کارها درست نرسیم. تو دنیای امروز که ایجنت‌های هوش مصنوعی جای نقش‌های جونیور رو می‌گیرن، مزیت رقابتی یه دانشجو تو تسلط بر نحوه استفاده از این ابزارهاست. مثلاً اگه تو گذشته یادگیری سینتکس یه زبان برنامه‌نویسی جدید ۴ ماه زمان می‌برد، الان به کمک هوش مصنوعی این زمان به یک ماه کاهش پیدا کرده. این ابزارها حتی می‌تونن به عنوان یه سنیور برای آموزش افراد جونیور عمل کنن و سرعت کار رو بالا ببرن. با این حال، هوش مصنوعی نمی‌تونه جایگزین تجربیاتی بشه که تو مسیر کاری و رویارویی با مسائل پیچیده به دست میاد؛ بنابراین تو پروژه‌های سخت، هنوز هم نمیشه نقش انسانی افراد رو نادیده گرفت.

??? خیلی از دانشجوها ترجیح میدن مستقیماً روی پروژه‌ها کار کنن و مفاهیم تئوری رو حین کار یاد بگیرن. آیا با این نوع «یادگیری پروژه‌محور» از همون ابتدا موافقید؟

این رویکرد اگه سطح کار پایین یا متوسط باشه جواب میده. اما اگه قرار باشه مسئله پیچیده‌ای رو حل کنید، حتماً به دید معماری سطح بالا و تسلط بر مفاهیم پایه نیاز دارید. شما حتی برای استفاده از هوش مصنوعی هم باید اون‌قدر دانش پایه‌ای داشته باشید که اگه ابزار به شما خروجی اشتباهی داد، بتونید اون رو اصلاح کنید و بین روش‌های مختلف تصمیم درستی بگیرید. مثلاً آیا منِ مهندس کامپیوتر می‌تونم جای یه پزشک متخصص بشینم و از ابزارهای هوش مصنوعی برای طبابت استفاده کنم؟ قطعاً نه!. تو مسائل سخت و تخصصی نرم‌افزار هم دقیقاً همین‌طوره و دانشجوها باید مطالعات عمیق داشته باشن.

??? امروزه دانشجوها با انبوهی از انتخاب‌ها روبه‌رو هستن؛ از گرایش‌های تحصیلی تا دوراهی‌های ورود به صنعت یا ادامه پژوهش. شما چطور با این سردرگمی‌ها دست‌وپنجه نرم کردید و چه توصیه‌ای برای مدیریت این شرایط دارید؟

این یه تصمیم کاملاً شخصیه و نسخه‌ی واحدی نداره. تو رشته ما، هم افرادی رو داریم که بدون تحصیلات آکادمیک معماران نرم‌افزار فوق‌العاده‌ای شدن و هم کسایی که مدرک دارن اما عملکرد خوبی تو بازار ندارن. واقعیت اینه که برای کارهای عمیق، به هر دو جنبه‌ی آکادمیک و مهارت ابزاری نیاز دارید. شما شاید بتونید با یادگیری چند تا ابزار، چند تا API رو فراخوانی کنید، اما اگه بخواید یه مدل هوش مصنوعی رو توسعه بدید یا اون رو به صورت مقیاس‌پذیر مستقر (Deploy) کنید، قطعاً به مفاهیم عمیق یادگیری ماشین و دانش DevOps نیاز پیدا می‌کنید.

??? به عنوان حرف آخر، اگه به روز اول کارشناسی خودتون برگردید چه توصیه‌ای به خودتون می‌کنید و چه پیامی برای دانشجوها دارید؟

اگه به گذشته برگردم، قطعاً دو سال اول رو با تمرکز خیلی بالایی درس می‌خونم تا مفاهیم پایه کاملاً برام جا بیفته. دانشجوهای امروز مزیت بزرگی دارن که ابزارهای هوش مصنوعی در اختیارشونه؛ اما این ابزارها شمشیر دولبه هستن. هوش مصنوعی هم می‌تونه سرعت پیشرفت رو بالا ببره و هم می‌تونه عامل بی‌سوادی و تنبلی بشه؛ مثلاً اگه دانشجو تو طول ترم تمرینات رو با هوش مصنوعی حل کنه، تو امتحان به مشکل برمی‌خوره. باید دلمون به حال خودمون بسوزه و از این امکانات برای ارتقای واقعی خودمون استفاده کنیم. نکته دیگه اینه که ما همیشه اهداف بزرگی مثل مهاجرت یا کار تو شرکت‌های خاص تو سر داریم، اما باید بدونیم مسیر رسیدن به اهداف بزرگ از قدم‌های کوچک تشکیل میشه. در نهایت فراموش نکنید که ما تو جزیره‌های جدا از هم زندگی نمی‌کنیم؛ سعی کنید با هم تعامل داشته باشید و به پیشرفت همدیگه کمک کنید. مهارت‌های انسانی (Soft Skills) چیزیه که هیچ هوش مصنوعی‌ای نمی‌تونه به شما آموزش بده و تو موفقیت شما نقش خیلی پررنگی داره.

تاریخچه و سیر تکامل مدل‌های زبانی در هوش مصنوعی

مشکات سلیمی
دانشجوی مهندسی کامپیوتر
دانشکدگان فارابی دانشگاه تهران
meshkatsaliminamin@gmail.com

۱۱ دقیقه

زبان، بنیادی‌ترین ابزار ارتباطی انسان است و توانایی درک و تولید آن همواره یکی از مهم‌ترین آرمان‌های پژوهشگران علوم رایانه و هوش مصنوعی بوده است. دستیابی به این هدف مستلزم توسعه‌ی سامانه‌هایی است که بتوانند ساختار، معنا و روابط میان واژه‌ها را درک کرده و بر اساس آن متنی منسجم تولید کنند. این نیاز، زمینه‌ساز شکل‌گیری مدل‌های زبانی[1] شد؛ مدل‌هایی که با تحلیل الگوهای موجود در داده‌های متنی، احتمال وقوع واژه‌ها و جملات را برآورد کرده و زیربنای بسیاری از فناوری‌های پردازش زبان طبیعی را تشکیل می‌دهند.

در سال‌های اخیر، پیشرفت در پردازش زبان طبیعی باعث شده است فناوری‌هایی که زمانی تنها در محیط‌های پژوهشی مورد استفاده قرار می‌گرفتند، به بخشی از زندگی روزمره تبدیل شوند. موتورهای جست‌وجو، مترجم‌های ماشینی، دستیارهای صوتی و سامانه‌های تولید محتوا، همگی از مدل‌هایی بهره می‌برند که توانایی تحلیل و پیش‌بینی زبان انسان را دارند. این تحول، حاصل چندین دهه پژوهش در زمینه‌ی مدل‌سازی زبان و یادگیری ماشین است که در نهایت به شکل‌گیری مدل‌های زبانی پیشرفته امروزی منجر شده است.

امروزه مدل‌های زبانی در طیف گسترده‌ای از فناوری‌های نوین، از موتورهای جست‌وجو و مترجم‌های ماشینی گرفته تا دستیارهای هوشمند، سامانه‌های پرسش و پاسخ، تولید محتوا و ابزارهای برنامه‌نویسی، نقش کلیدی ایفا می‌کنند. با این حال، توانایی چشمگیر این مدل‌ها نتیجه‌ی مسیری طولانی از پژوهش و نوآوری است که طی چندین دهه و با مشارکت متخصصان حوزه‌های آمار، زبان‌شناسی رایانشی، یادگیری ماشین و شبکه‌های عصبی شکل گرفته است.

روند تکامل مدل‌های زبانی از روش‌های آماری ساده آغاز شد، سپس با ورود شبکه‌های عصبی و یادگیری نمایش معنایی واژه‌ها دگرگون شد و در نهایت با معرفی معماری ترنسفورمر[2] و مدل‌های زبانی بزرگ، به مرحله‌ای رسید که مدل‌ها توانایی پردازش و تولید متونی روان، منسجم و نزدیک به زبان انسان را به دست آوردند.

آغاز مدل‌های زبانی؛ از زبان‌شناسی تا مدل‌های آماری

نخستین تلاش‌ها برای مدل‌سازی زبان طبیعی به دهه‌های میانی قرن بیستم بازمی‌گردد؛ زمانی که پژوهشگران دریافتند می‌توان بسیاری از ویژگی‌های زبان را با استفاده از روش‌های آماری توصیف کرد. برخلاف رویکردهای مبتنی بر قواعد دستوری که طراحی و توسعه آن‌ها بسیار دشوار بود، مدل‌های آماری با تکیه بر حجم زیادی از متون، الگوهای تکرارشونده زبان را استخراج می‌کردند و از این طریق احتمال وقوع واژه‌ها و عبارت‌ها را تخمین می‌زدند.

یکی از مهم‌ترین دستاوردهای این دوره، معرفی مدل‌های n-gram بود. این مدل‌ها بر این فرض استوار بودند که احتمال وقوع هر واژه تنها به تعداد محدودی از واژه‌های پیش از آن وابسته است. برای مثال، در مدل سه‌تایی[3]، پیش‌بینی واژه‌ی بعدی تنها بر اساس دو واژه‌ی قبلی انجام می‌شود. با وجود سادگی، این روش در زمان خود تحول مهمی به شمار می‌رفت و برای سال‌ها در سامانه‌های تشخیص گفتار، تصحیح خودکار متن، پیش‌بینی کلمات و ترجمه‌ی ماشینی مورد استفاده قرار گرفت.

با گذشت زمان، محدودیت‌های این رویکرد بیش از پیش آشکار شد. وابستگی مدل به تعداد محدودی از واژه‌های قبلی، باعث می‌شد نتواند ارتباط میان کلمات دور از هم را درک کند و در نتیجه، در پردازش جملات طولانی با افت دقت مواجه شود. افزون بر این، نیاز به حجم بسیار زیادی از داده‌های آموزشی و ناتوانی در برخورد با واژه‌ها یا ترکیب‌های جدید، استفاده از این مدل‌ها را با چالش‌های جدی روبه‌رو می‌کرد.

این محدودیت‌ها پژوهشگران را به سوی رویکردهای تازه‌ای هدایت کرد که به‌جای شمارش صرف واژه‌ها، بتوانند روابط معنایی میان آن‌ها را نیز بیاموزند. پیشرفت در شبکه‌های عصبی و افزایش توان پردازشی رایانه‌ها در دهه‌های پایانی قرن بیستم، زمینه را برای شکل‌گیری نسل جدیدی از مدل‌های زبانی فراهم ساخت؛ نسلی که نگاه به پردازش زبان طبیعی را برای همیشه دگرگون کرد.

گذار به شبکه‌های عصبی؛ آغاز درک معنای واژه‌ها

در دهه‌ی ۱۹۹۰، هم‌زمان با پیشرفت روش‌های یادگیری ماشین، توجه پژوهشگران به استفاده از شبکه‌های عصبی برای پردازش زبان طبیعی جلب شد. برخلاف مدل‌های آماری که تنها بر فراوانی و توالی واژه‌ها تکیه داشتند، شبکه‌های عصبی این قابلیت را داشتند که الگوهای پیچیده‌تری را از داده‌های زبانی استخراج کرده و روابط معنایی میان واژه‌ها را بیاموزند.

یکی از نخستین گام‌های مؤثر در این مسیر، شبکه‌های عصبی بازگشتی(RNN)[4] بودند که از

۱. Language Models

۲. Transformer Architecture

۳. Trigram

۴. Recurrent Neural Networks

دهه‌ی ۱۹۸۰ توسعه یافته بودند. این شبکه‌ها برای پردازش داده‌های ترتیبی طراحی شده بودند و با حفظ اطلاعات مراحل پیشین، می‌توانستند وابستگی میان واژه‌های یک جمله را بهتر از مدل‌های آماری درک کنند. با وجود این، RNNها در یادگیری وابستگی‌های بلندمدت با مشکلی شناخته‌شده به نام «محو شدن گرادیان[۵]» روبه‌رو بودند که دقت آن‌ها را در پردازش متن‌های طولانی کاهش می‌داد.

برای رفع این محدودیت، در سال ۱۹۹۷معماری حافظه‌ی بلند کوتاه‌مدت(LSTM)[۶] معرفی شد. این معماری با بهره‌گیری از سازوکارهای کنترلی، امکان حفظ اطلاعات مهم در بازه‌های زمانی طولانی‌تر را فراهم کرد و به یکی از مؤثرترین روش‌های پردازش زبان طبیعی در دهه‌های بعد تبدیل شد. استفاده گسترده از LSTM در کاربردهایی مانند ترجمه‌ی ماشینی، تشخیص گفتار و تولید متن، نشان داد که شبکه‌های عصبی می‌توانند درک عمیق‌تری از ساختار زبان نسبت به مدل‌های آماری ارائه دهند.

گام مهم بعدی در سال ۲۰۰۳ و با معرفی مدل زبانی عصبی[۷] توسط یوشوا بنجیو و همکارانش برداشته شد. این پژوهش نشان داد که می‌توان واژه‌ها را به بردارهای عددی تبدیل کرد و به‌جای ذخیره‌ی جداگانه‌ی هر واژه، نمایش‌های پیوسته‌ای از آن‌ها را آموخت. این رویکرد، مشکل پراکندگی داده‌ها[۸] را تا حد زیادی کاهش داد و بنیان نظری بسیاری از مدل‌های زبانی مدرن را شکل داد.

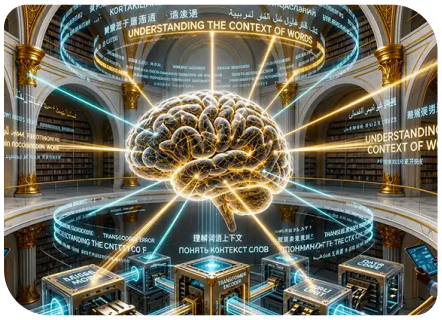
این ایده در سال ۲۰۱۳ بامعرفی Word2Vec[۹] توسط پژوهشگران گوگل به‌شکلی کاربردی و فراگیر توسعه یافت. Word2Vec واژه‌ها را در قالب بردارهایی با ابعاد ثابت نمایش می‌داد؛ به‌گونه‌ای که واژه‌های دارای معنای مشابه در فضای برداری به یکدیگر نزدیک قرار می‌گرفتند. این دستاورد، درک معنایی واژه‌ها را وارد مرحله‌ای تازه کرد و تأثیر عمیقی بر توسعه‌ی مدل‌های زبانی و سایر کاربردهای پردازش زبان طبیعی گذاشت.

اگرچه شبکه‌های عصبی نسبت به مدل‌های آماری عملکرد بسیار بهتری داشتند، آموزش آن‌ها همچنان زمان‌بر بود و پردازش متن‌های طولانی با محدودیت‌هایی همراه بود. این چالش‌ها پژوهشگران را به سوی معماری‌های جدیدی سوق داد که بتوانند با دقت بیشتر و سرعت بالاتر، روابط میان واژه‌ها را مدل‌سازی کنند.

انقلاب Transformer؛ نقطه عطفی در تاریخ مدل‌های زبانی

در سال ۲۰۱۷، گروهی از پژوهشگران گوگل مقاله‌ای با عنوان Attention Is All You Need منتشر کردند که مسیر توسعه‌ی مدل‌های زبانی را به‌طور اساسی تغییر داد. در این پژوهش، معماری جدیدی با نام Transformer معرفی شد که برخلاف مدل‌های پیشین، برای پردازش متن به ساختارهای بازگشتی مانند RNN و LSTM وابسته نبود.

هسته‌ی اصلی این معماری، سازوکاری به نام خودتوجهی[۱۰] بود. این سازوکار به مدل اجازه می‌داد هنگام پردازش هر واژه، به تمام واژه‌های دیگر جمله نیز توجه کند و میزان اهمیت هر یک را با توجه به متن تعیین کند. در نتیجه، مدل می‌توانست وابستگی‌های دور و نزدیک میان کلمات را به‌صورت هم‌زمان در نظر بگیرد؛ قابلیتی که در معماری‌های پیشین به‌سادگی امکان‌پذیر نبود.



حذف ساختار بازگشتی مزیت دیگری نیز به همراه داشت. در حالی که RNNها واژه‌ها را به‌ترتیب و یکی پس از دیگری پردازش می‌کردند، ترنسفورمر امکان پردازش موازی کلِ جمله را فراهم ساخت. این ویژگی، زمان آموزش مدل‌ها را به‌طور چشمگیری کاهش داد و استفاده از مجموعه‌داده‌های بسیار بزرگ را امکان‌پذیر کرد. افزایش سرعت آموزش، همراه با بهبود دقت، موجب شد این معماری به‌سرعت به استاندارد جدید پردازش زبان طبیعی تبدیل شود.

موفقیت ترنسفورمر تنها به بهبود عملکرد در وظایفی مانند ترجمه‌ی ماشینی محدود نماند. اندکی پس از معرفی این معماری، مدل‌های متعددی بر پایه‌ی آن توسعه یافتند که هر یک گامی مهم در تکامل پردازش زبان طبیعی بودند. در سال ۲۰۱۸، گوگل مدل BERT را معرفی کرد که با درک هم‌زمان متن از هر دو جهت، دقت بسیاری از سامانه‌های درک زبان را به‌شکل قابل‌توجهی افزایش داد. در همان سال، شرکت OpenAI نیز نخستین نسخه‌ی GPT را عرضه کرد؛ مدلی که با تمرکز بر تولید متن، مسیر تازه‌ای را در توسعه‌ی مدل‌های زبانی گشود.

ترنسفورمر نه‌تنها بخش مهمی از محدودیت‌های نسل‌های پیشین را کاهش داد، بلکه امکان آموزش مدل‌هایی با میلیاردها پارامتر را نیز فراهم ساخت. همین ویژگی، زمینه‌ساز ظهور نسل جدیدی از مدل‌های زبانی شد که امروزه با عنوان مدل‌های زبانی بزرگ (LLMs)[۱۱] شناخته می‌شوند و نقش محوری در بسیاری از فناوری‌های هوش مصنوعی ایفا می‌کنند.

ظهور مدل‌های زبانی بزرگ؛ آغاز عصر جدید هوش مصنوعی

معرفی معماری ترنسفورمر، امکان توسعه‌ی مدل‌هایی با مقیاسی بسیار بزرگ‌تر از گذشته را فراهم کرد. افزایش توان پردازشی، دسترسی به مجموعه‌داده‌های عظیم و پیشرفت روش‌های آموزش شبکه‌های عصبی موجب شد پژوهشگران بتوانند مدل‌هایی

۵. Vanishing Gradient
۶. Long Short-Term Memory
۷. Neural Probabilistic Language Model
۸. Data Sparsity
۹. Word to Vector
۱۰. Self-Attention
۱۱. Large Language Models

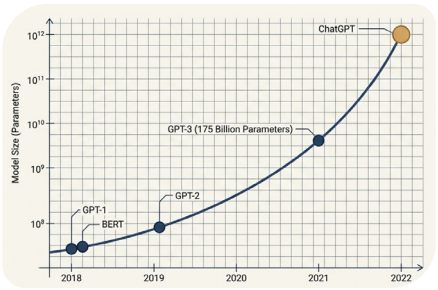
با صدها میلیون و سپس میلیاردها پارامتر طراحی کنند. این دسته از سامانه‌ها که با عنوان مدل‌های زبانی بزرگ شناخته می‌شوند، توانایی چشمگیری در پردازش و تولید زبان طبیعی از خود نشان دادند.

شرکت OpenAI در سال ۲۰۱۸ نخستین نسخه ازخانواده‌ی GPT [۱۲] را معرفی کرد. هرچند GPT-1 در مقایسه با مدل‌های امروزی کوچک به نظر می‌رسد، اما نشان داد که پیش‌آموزش یک مدل روی حجم گسترده‌ای از متون و سپس تنظیم آن برای وظایف مختلف، رویکردی مؤثر در پردازش زبان طبیعی است. این ایده در نسخه‌های بعدی به‌سرعت توسعه یافت. GPT-2 با توانایی تولید متن‌های روان و منسجم توجه بسیاری از پژوهشگران را جلب کرد و GPT-3 با بهره‌گیری از حدود ۱۷۵ میلیارد پارامتر، مرزهای جدیدی در تولید متن، ترجمه، خلاصه‌سازی و پاسخ‌گویی به پرسش‌ها ترسیم کرد.

در سال ۲۰۲۲، عرضه‌ی ChatGPT موجب شد مدل‌های زبانی بزرگ از محیط‌های پژوهشی فراتر رفته و به ابزاری در دسترس میلیون‌ها کاربر تبدیل شوند. این سامانه نشان داد که مدل‌های زبانی می‌توانند در مکالمه، آموزش، تولید محتوا، برنامه‌نویسی، تحلیل متون و بسیاری از فعالیت‌های روزمره عملکردی مؤثر داشته باشند. استقبال گسترده از ChatGPT، رقابت میان شرکت‌های بزرگ فناوری را نیز وارد مرحله‌ای تازه کرد.

در ادامه، شرکت‌های مختلف مدل‌های زبانی اختصاصی خود را توسعه دادند. گوگل خانواده‌ی Gemini را با تمرکز بر قابلیت‌های چندوجهی معرفی کرد. شرکت متا مدل‌های با وزن‌های باز Llama را در اختیار جامعه‌ی پژوهشی قرار داد و شرکت Anthropic نیز خانواده‌ی Claude را با تأکید بر ایمنی و قابلیت اطمینان بیشتر توسعه داد. این رقابت، سرعت پیشرفت مدل‌های زبانی را به‌شکل قابل‌توجهی افزایش داد و کاربردهای آن‌ها را در حوزه‌هایی مانند پزشکی، آموزش، مهندسی، حقوق، علوم و صنایع گوناگون گسترش داد.

امروزه مدل‌های زبانی بزرگ تنها ابزارهایی برای تولید متن نیستند، بلکه به سامانه‌هایی چندمنظوره تبدیل شده‌اند که توانایی تحلیل اسناد، تولید و بررسی کد، ترجمه، استدلال، پردازش تصویر و صدا و تعامل با سایر سامانه‌های نرم‌افزاری را نیز دارند. این تحول نشان می‌دهد که مدل‌های زبانی از یک فناوری تخصصی در حوزه‌ی پردازش زبان طبیعی، به یکی از ارکان اصلی توسعه‌ی هوش مصنوعی نوین تبدیل شده‌اند.



چالش‌ها و چشم‌انداز آینده

با وجود پیشرفت چشمگیر مدل‌های زبانی بزرگ، این فناوری همچنان با چالش‌های متعددی روبه‌رو است. یکی از مهم‌ترین این چالش‌ها، پدیده‌ی«توهم» [۱۳] است؛ وضعیتی که در آن مدل اطلاعاتی نادرست یا فاقد پشتوانه‌ی واقعی را با اطمینان بالا تولید می‌کند. چنین خطاهایی، به‌ویژه در حوزه‌هایی مانند پزشکی، حقوق و آموزش، می‌تواند پیامدهای قابل‌توجهی به همراه داشته باشد.

سوگیری موجود در داده‌های آموزشی نیز از دیگر مسائل مهم این حوزه است. از آنجا که مدل‌های زبانی بر پایه‌ی حجم عظیمی از متون موجود آموزش می‌بینند، ممکن است برخی سوگیری‌های فرهنگی، اجتماعی یا زبانی موجود در این داده‌ها را بازتولید کنند. کاهش این سوگیری‌ها و افزایش شفافیت در عملکرد مدل‌ها، از موضوعات مهم پژوهشی در سال‌های اخیر بوده است.

در کنار این موارد، آموزش و بهره‌برداری از مدل‌های زبانی بزرگ به منابع محاسباتی گسترده، مصرف بالای انرژی و هزینه‌های قابل‌توجه نیاز دارد. به همین دلیل، بسیاری از پژوهش‌های جدید بر توسعه‌ی مدل‌های کوچک‌تر، کارآمدتر و کم‌مصرف‌تر متمرکز شده‌اند تا امکان استفاده از این فناوری در دستگاه‌های شخصی و سامانه‌های با منابع محدود نیز فراهم شود.

روند پیشرفت سال‌های اخیر نشان می‌دهد که آینده‌ی مدل‌های زبانی تنها به پردازش متن محدود نخواهد بود. ترکیب زبان با تصویر، صدا، ویدئو و سایر انواع داده‌ها، زمینه‌ساز توسعه‌ی مدل‌های چندوجهی [۱۴] شده است؛ مدل‌هایی که قادرند اطلاعات را از منابع مختلف دریافت، تحلیل و به‌شکلی یکپارچه پردازش کنند. این رویکرد، افق‌های تازه‌ای را برای تعامل انسان و ماشین گشوده و انتظار می‌رود در سال‌های آینده نقش پررنگ‌تری در حوزه‌هایی مانند آموزش، سلامت، پژوهش، صنعت و خدمات هوشمند ایفا کند.

جمع‌بندی

مسیر تکامل مدل‌های زبانی، نمونه‌ای روشن از تأثیر پیشرفت‌های علمی بر توسعه‌ی فناوری‌های نوین است. این مسیر از مدل‌های آماری ساده آغاز شد، با ورود شبکه‌های عصبی و یادگیری نمایش معنایی واژه‌ها وارد مرحله‌ای تازه شد و با معرفی معماری ترنسفورمر و مدل‌های زبانی بزرگ، به یکی از مهم‌ترین دستاوردهای هوش مصنوعی معاصر تبدیل شد.

امروزه مدل‌های زبانی نه‌تنها در پردازش زبان طبیعی، بلکه در طیف گسترده‌ای از کاربردهای علمی، آموزشی و صنعتی مورد استفاده قرار می‌گیرند و روند پیشرفت آن‌ها همچنان با سرعت ادامه دارد. هرچند چالش‌هایی مانند دقت، سوگیری، هزینه‌های محاسباتی و ملاحظات اخلاقی همچنان پابرجاست، اما شواهد نشان می‌دهد که این فناوری در سال‌های آینده نقشی اساسی در شکل‌گیری نسل جدید سامانه‌های هوشمند و تحول شیوه‌ی تعامل انسان با رایانه ایفا خواهد کرد.

۱۲. Generative Pre-trained Transformer

۱۳. Hallucination

۱۴. Multimodal Models

عامل‌ها؛ ایده‌ای که هفتاد سال منتظر زمان خودش ماند

مهرناز حسینی
دانشجوی مهندسی کامپیوتر
دانشکدگان فارابی دانشگاه تهران
mehrnazhosseini4@gmail.com

۱۱ دقیقه

این روزها وقتی از عامل‌های[1] هوش مصنوعی صحبت می‌کنیم، معمولاً منظورمان سامانه‌ای است که می‌تواند هدفی را دریافت کند، درباره‌ی آن تصمیم بگیرد، از ابزارهای مختلف استفاده کند و در نهایت کاری را انجام دهد [1]. شاید این مفهوم کاملاً تازه به نظر برسد، اما ایده‌ی اصلی آن نزدیک به هفتاد سال قدمت دارد.

جالب اینجاست که تاریخچه‌ی عامل‌ها، تاریخچه‌ی اختراع یک فناوری جدید نیست؛ بلکه تاریخچه‌ی برطرف شدن یک محدودیت است. هر نسل از پژوهشگران به مشکلی برخورد، راه‌حلی برای آن پیدا کرد و در همان لحظه، محدودیت تازه‌ای آشکار شد. همین زنجیره، عامل‌های امروزی را شکل داده است.

دهه‌ی ۵۰ و ۶۰ میلادی | اولین سؤال: آیا کامپیوتر می‌تواند خودش نتیجه بگیرد؟

در دهه‌ی ۱۹۵۰، برنامه‌ها چیزی بیش از مجموعه‌ای از دستورهای از پیش نوشته‌شده نبودند. اگر شرایط تغییر می‌کرد، برنامه هم ممکن بود متوقف شود؛ چون در شرایط پیش‌بینی‌نشده راهی برای تصمیم‌گیری نداشت.

در سال ۱۹۵۶، در کارگاه معروف دارتموث[2]، اصطلاح «هوش مصنوعی» برای نخستین بار مطرح شد [2]. دو سال بعد، جان مک‌کارتی در مقاله‌ی Programs with Common Sense ایده‌ای را مطرح کرد که بعدها یکی از ریشه‌های مفهوم عامل‌ها شد [3]. او برنامه‌ای فرضی به نام Advice Taker را توصیف کرد؛ برنامه‌ای که به‌جای اینکه تمام رفتارهایش از قبل نوشته شوند، بتواند مجموعه‌ای از دانسته‌ها را دریافت کند، میان آن‌ها استدلال کند و خودش به نتیجه‌ی تازه برسد.

امروز این ایده بدیهی به نظر می‌رسد، اما در آن زمان بسیار جلوتر از امکانات موجود بود. کامپیوترها توان پردازشی محدودی داشتند و روش‌های استدلال ماشینی هنوز در ابتدای راه بودند. با این حال، یک تغییر مهم اتفاق افتاده بود: برای نخستین بار «دانش» از «برنامه» جدا شد و تصمیم‌گیری به خودِ سیستم سپرده شد.

این همان بذری بود که بعدها در تمام نسل‌های عامل‌ها رشد کرد.

دهه‌ی ۷۰ و ۸۰ | آیا کامپیوتر می‌تواند مثل یک متخصص تصمیم بگیرد؟

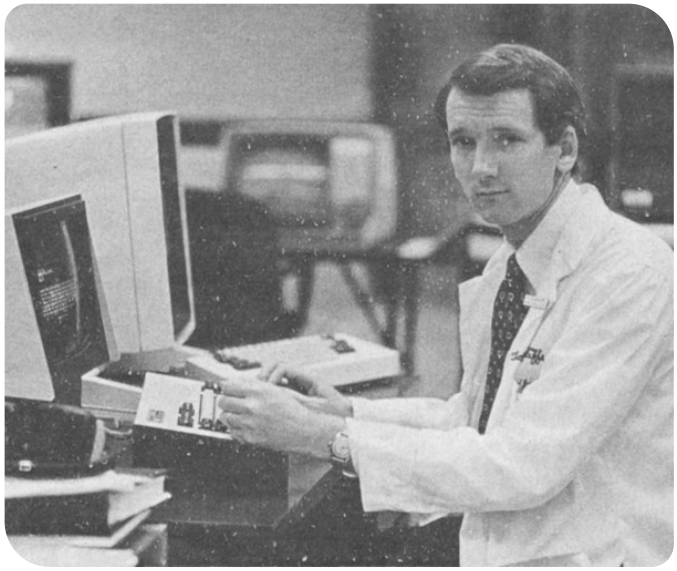
وقتی اصلِ استدلال پذیرفته شد، پرسش بعدی شکل گرفت: اگر دانش یک متخصص را به کامپیوتر منتقل کنیم، آیا می‌تواند مانند او تصمیم بگیرد؟

پاسخ آن دوران، «سیستم‌های خبره»[3] بود؛ سامانه‌هایی که دانش متخصصان را به صدها یا هزاران قانون منطقی تبدیل می‌کردند. مشهورترین نمونه، MYCIN بود که برای کمک به تشخیص عفونت‌های باکتریایی و پیشنهاد درمان توسعه یافت. در ارزیابی‌های انجام‌شده، عملکرد آن در بسیاری از سناریوها با نظر متخصصان بیماری‌های عفونی قابل مقایسه بود؛ هرچند به دلایل فنی، حقوقی و اخلاقی هرگز به‌طور گسترده وارد محیط درمان نشد [4].

نمونه‌ی موفق‌تر، XCON بود که از سال ۱۹۸۰ در شرکت Digital Equipment Corporation برای پیکربندی کامپیوترهای VAX به کار گرفته شد و با خودکار کردن فرایندی پیچیده، صرفه‌جویی اقتصادی قابل‌توجهی ایجاد کرد [5]. این موفقیت نشان داد که تصمیم‌گیری ماشینی می‌تواند از آزمایشگاه خارج شود و در صنعت ارزش واقعی خلق کند.

اما محدودیت این نسل نیز به‌تدریج آشکار شد. هر قانون باید به‌صورت دستی نوشته، بررسی و به‌روزرسانی می‌شد. هرچه مسئله بزرگ‌تر می‌شد، تعداد قوانین نیز افزایش پیدا می‌کرد و نگهداری آن‌ها دشوارتر می‌شد. سیستم‌های خبره ثابت کردند که ماشین می‌تواند مانند یک متخصص تصمیم بگیرد؛ اما نه در دنیایی که دانش آن هر روز تغییر می‌کند.

بنابراین، مسئله دیگر «باهوش‌تر کردن یک سیستم» نبود؛ بلکه پیدا کردن راهی برای تقسیم دانش و تصمیم‌گیری میان چند عامل مستقل بود.



دهه‌ی ۹۰ | اگر یک عامل کافی نیست، چند عامل چطور؟

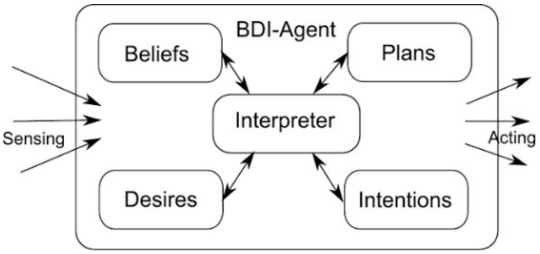
در دهه‌ی ۱۹۹۰ نگاه پژوهشگران به عامل تغییر کرد. به‌جای ساختن

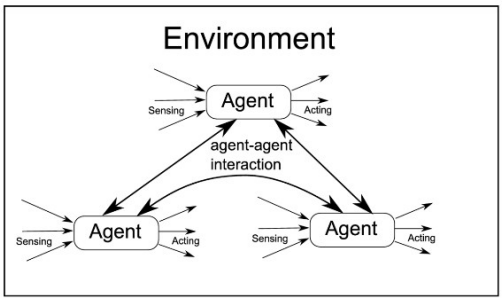
۱. Agent
۲. Dartmouth
۳. Expert Systems

یـک سیسـتم بـزرگ کـه همه‌چیـز را بدانـد، ایـده‌ی جدیـد ایـن بـود کـه هـر عامـل مسـئول بخشـی از مسـئله باشـد و بـرای رسـیدن بـه یـک هـدف مشـترک بـا عامل‌هـای دیگـر همـکاری کنـد [6].

در همیـن دوره، مـدل BDI[۴] چارچوبـی بـرای تصمیم‌گیـری عامل‌هـا ارائـه داد. بـر اسـاس ایـن مـدل، تصمیم‌گیـری عامـل بـر پایـه‌ی باورهـا، خواسـته‌ها و قصدهـای آن انجـام می‌شـود [7]. هم‌زمـان، پژوهشـگران زبان‌هـا و اسـتانداردهایی طراحـی کردنـد تـا عامل‌هـا بتواننـد اطلاعـات، درخواسـت‌ها و نتایـج را به‌صـورت سـاختاریافته بـا یکدیگـر ردوبـدل کننـد.

ایـن تغییـر، مفهوم عامـل را از یک «برنامه‌ی تصمیم‌گیـر» به عضوی از یک «سـامانه‌ی همکاری‌کننـده» تبدیـل کـرد. امـا یـک مشـکل اساسـی همچنان باقـی بـود. ایـن عامل‌هـا می‌توانسـتند بـا یکدیگـر پیـام ردوبـدل کننـد، امـا هنـوز درک عمیقـی از زبـان طبیعـی نداشـتند و تعامـل آن‌ها با دنیـای واقعی محـدود بـه قوانینـی بود کـه از قبـل برایشـان تعریف شـده بود.





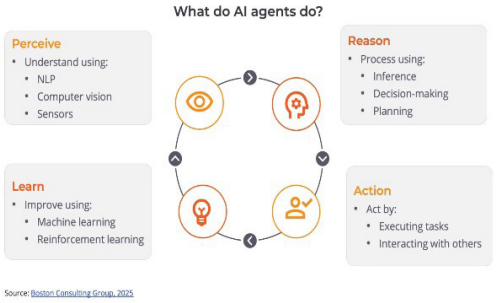
از ۲۰۲۰ تا امروز | وقتی عامل‌ها توانستند عمل کنند

ظهـور مدل‌هـای زبانـی بـزرگ[۵]، ایـن محدودیـت را تـا حـد زیـادی برطـرف کـرد. ایـن مدل‌هـا می‌توانسـتند زبـان طبیعـی را درک کننـد، اطلاعـات را خلاصـه کننـد و دربـاره‌ی مسـائل اسـتدلال کننـد؛ امـا در ابتـدا هنـوز فقـط تولیدکننـده‌ی متـن بودنـد و توانایـی انجـام عمـل نداشـتند.

نقطه‌ی عطف، معرفی روش‌هایی بود که «استدلال» و «عمل» را در یک چرخه قرار دادند [8]. از اینجا به بعد، عامل دیگر فقط پاسخ تولید نمی‌کرد؛ می‌توانست هنگام حل مسئله از ابزارهای بیرونی استفاده کند، نتیجه را ارزیابی کند، در صورت نیاز مسیرش را تغییر دهد و دوباره تصمیم بگیرد. هم‌زمان، استانداردها و زیرساخت‌هایی نیز شکل گرفتند تا ارتباط میان مدل‌های زبانی، ابزارها و منابع داده ساختاریافته‌تر و قابل‌اعتمادتر شود.

البتـه ایـن نسـل هـم کامل نیسـت. عامل‌هـای امـروزی ممکن اسـت دچار خطـا شـوند، اطلاعـات نادرسـت تولیـد کننـد یـا در اسـتفاده از ابزارهـا اشـتباه کننـد. بـه همین دلیـل، پژوهش دربـاره‌ی روش‌هـای افزایش دقـت، قابلیت اطمینـان و نظـارت بـر آن‌هـا همچنان ادامـه دارد.

شـاید ظاهـر عامل‌هـای امـروزی بـا آنچـه پژوهشـگران دهـه‌ی ۱۹۵۰ تصور می‌کردنـد کامـلاً متفـاوت باشـد؛ امـا ایـده‌ی اصلـی تغییـر چندانـی نکـرده اسـت. هنـوز هـم یـک عامـل موفق بایـد بتوانـد محیـط را درک کنـد، درباره‌ی آن تصمیـم بگیـرد و بـرای رسـیدن بـه هدفش عمل کنـد. آنچه در ایـن هفتاد سـال تغییـر کـرده، نـه خودِ ایـده، بلکـه ابزارهایی اسـت که آن رویـای قدیمی را هـر بـار یـک قـدم بـه واقعیـت نزدیک‌تـر کرده‌اند.



منابع

[1] Russell, S. J., & Norvig, P. (2021). Artificial intelligence: A modern approach (4th ed.). Pearson.

[2] McCarthy, J., Minsky, M. L., Rochester, N., & Shannon, C. E. (1955). A proposal for the Dartmouth summer research project on artificial intelligence. http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html

[3] McCarthy, J. (1959). Programs with common sense. In Proceedings of the Teddington Conference on the Mechanization of Thought Processes (pp. 75–91). Her Majesty's Stationery Office.

[4] Buchanan, B. G., & Shortliffe, E. H. (Eds.). (1984). Rule-based expert systems: The MYCIN experiments of the Stanford Heuristic Programming Project. Addison-Wesley.

[5] Bachant, J., & McDermott, J. (1984). R1 revisited: Four years in the trenches. AI Magazine, 5(3), 21–32. https://doi.org/10.1609/aimag.v5i3.445

[6] Wooldridge, M. J., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. The Knowledge Engineering Review, 10(2), 115–152. https://doi.org/10.1017/S0269888900008122

[7] Rao, A. S., & Georgeff, M. P. (1995). BDI agents: From theory to practice. In Proceedings of the First International Conference on Multiagent Systems (pp. 312–319). AAAI Press.

[8] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. International Conference on Learning Representations. https://arxiv.org/abs/2210.03629

۴. Belief–desire–intention

۵. LLMs (Large Language Models)

پهپادها؛ از سرگرمی تا میدان جنگ!

یاشار ابری
دانش‌آموخته‌ی ارشد مهندسی کامپیوتر دانشگاه تهران و دانشجوی دکتری
yasharabri@ut.ac.ir

۱۰ دقیقه

در مقاله‌ی پیش‌رو قصد داریم ابتدا به‌صورت خودمانی تاریخچه‌ای از پهپادها[۱] را تعریف کنیم و سپس بررسی کنیم چگونه پای این وسایل الکترونیکی که زمانی گجت‌های[۲] فانتزی و بلندپروازانه بودند به میدان جنگ کشیده شد.

شاید امروزه پهپادها یکی از مهم‌ترین نمادهای همگرایی فناوری‌های مختلف مانند الکترونیک، هوش مصنوعی، مخابرات و رباتیک باشند و نقش آن‌ها در اقتصاد، رسانه، امداد و حتی امنیت روزبه‌روز پررنگ‌تر می‌شود [1][2].

ما پهپادها را از چه زمانی مشاهده کردیم؟

احتمالاً مشاهده‌ی اولین نسخه‌های پهپادها، دست‌کم از دید مردم عادی جامعه، زمانی شروع شد که شرکت‌های اسباب‌بازی‌سازی تلاش کردند یک هلیکوپتر کنترلی مانند اتومبیل‌های کنترلی بسازند؛ البته با در نظر گرفتن این اصل مهم که واقعاً پرواز کند، نه اینکه صرفاً ماکتی خلاقانه باشد که تظاهر به پرواز می‌کند.

البته اگر از منظر تاریخی نگاه کنیم، نمونه‌های اولیه‌ی هواگردهای بدون سرنشین حتی به اوایل قرن بیستم و پروژه‌های نظامی برمی‌گردند، اما ورود آن‌ها به زندگی روزمره‌ی مردم بیشتر در دهه‌های اخیر و با ارزان شدن قطعات الکترونیکی و باتری‌های لیتیومی رخ داد [1][3].

این‌گونه اسباب‌بازی‌ها البته شکل‌های مختلفی داشتند؛ برخی شبیه بشقاب پرنده بودند، برخی واقعاً شبیه هلیکوپتر یا بالگرد واقعی در نسخه‌ی کوچک‌تر بودند و برخی نیز ۴ پره یا بیشتر، مثل ۶ یا ۸ پره، داشتند و به مدل‌های امروزی شبیه‌تر بودند.

ظهور موتورهای براشلِس[۳] کوچک و کنترل‌کننده‌های پرواز[۴] دیجیتال باعث شد این طراحی‌های چندپره پایدارتر و قابل‌کنترل‌تر شوند و به‌سرعت از اسباب‌بازی به ابزار نیمه‌حرفه‌ای تبدیل گردند [2].

۱. Drones
۲. Gadgets
۳. Brushless Motors
۴. Flight Controllers

اسباب‌بازی‌های پروازی به ما چه چیزی را ثابت کردند؟

اسباب‌بازی‌های پروازی علاوه بر موارد مختلف، دو چیز مهم را به ما ثابت کردند که شاید تا مدت‌ها برای خیلی‌ها قابل باور و تصور نبود:

۱. برای ساخت یک شیء پروازی نیازی به وسایل گران‌قیمت نیست

برای بشری که آرزوی پرواز با استفاده از تجهیزات دست‌سازش را داشت، شاید بعد از اختراع هواپیما فکر نمی‌کرد که بشود با وسایل ارزان و کوچک وسیله‌ای دارای قابلیت پرواز و دقیق ساخت. اسباب‌بازی‌ها در مقیاس کوچک نشان دادند می‌شود. بچه‌هایی که هر ساله با اسباب‌بازی‌های تجاری یا دست‌ساز یا ترکیب آن دو در پارک‌ها می‌رفتند تا در مسابقات محلی هواوردی بچه‌ها شرکت کنند، این موضوع را هر ساله اثبات می‌کردند.

در واقع کاهش قیمت سنسورها، ژیروسکوپ‌ها[5] و تراشه‌های کنترل پرواز باعث شد ساخت هواگردهای کوچک از یک پروژه‌ی پیچیده‌ی دانشگاهی به یک سرگرمی قابل‌دسترس برای عموم تبدیل شود [2][4].

۲. برای پرواز حتماً به جایگاه و تدارکات خاصی نیاز نیست

درست است که از وقتی هلیکوپترها آمدند دیگر نیازی به باندهای طولانی هواپیماها نبود، ولی همچنان تصور بر این بود که حتی برای هلیکوپترها هم نیاز به فضایی مربعی خالی و خلوت، همراه با رنگ و علامت و آدم‌هایی که علامت‌های مخصوص نشان بدهند، برج مراقبت و... بود.

واقعاً چه کسی فکر می‌کرد حتی زیر درختی کنار رودخانه‌ای بتوان وسیله‌ای را به پرواز درآورد؟ این را نیز باز بچه‌ها در مسابقاتی که در پارک‌ها یا فضاهای طبیعت بیرون‌شهری برگزار می‌شد، نشان دادند.

۵. Gyroscopes

البته همین آزادی عمل باعث شد بعداً قوانین هوانوردی شهری و محدودیت‌های ایمنی برای پرواز پهپادها در بسیاری از کشورها تدوین شود [5].

وقتی پاک کردن صورت‌مسئله واقعاً جواب می‌دهد!

در گذشته وقتی صحبت از ربات‌هایی می‌شد که یا به کمک اپراتور انسانیِ کنترل‌کننده‌ی آن‌ها یا به‌وسیله‌ی هوش مصنوعی و به‌طور خودکار بتوانند از جایی به جای دیگری بروند و کاری انجام دهند، چالش‌های حرکت روی زمین از مهم‌ترین چالش‌ها بود. این‌گونه ربات‌ها که قرار بود امدادی یا پستچی یا جنگی و جاسوسی باشند، برایشان ابزارهای حرکتی مختلفی، از چرخ‌های تایردار گرفته تا چرخ‌های مخصوص تانک‌ها و حتی بعضاً پاهای رباتیک پیشنهاد و طراحی می‌شد.

اما احتمالاً اولین بار کسی این سؤال را از خودش پرسید که اصلاً چه لزومی دارد این ربات‌ها روی زمین باشند و روی آن راه بروند تا مجبور باشیم به موانع مختلفی، از سنگ و رودخانه تا شیب‌های تند و دیوارهای عمودی، فکر کنیم؟ بله، اینجا بود که طراحی ربات‌های پروازی با مستقل‌شدن از زمین و پاک‌کردن صورت‌مسئله‌ی موانع زمینی جواب داد.

این تغییر نگاه، به‌نوعی جهش مفهومی در رباتیک محسوب می‌شود و امروزه در حوزه‌هایی مثل نقشه‌برداری، امداد و بازرسی صنعتی نیز کاربرد گسترده دارد [4][6].

برخلاف تصور، پس از اسباب‌بازی‌ها نوبت پهپادهای فیلم‌برداری بود نه پستچی!

همان‌طور که گفتیم وقتی از رباتیک صحبت می‌شود همه ما احتمالاً به آدم‌آهنی‌هایی فکر می‌کردیم که قرار است مأموریتی انجام دهند و احتمالاً بسته‌ای را به جایی برسانند ولی شاید کسی فکر نمی‌کرد که استفاده از آن‌ها برای فیلم‌برداری و ثبت چشم‌اندازهای هنری که شاید بعضاً عکاسان و فیلم‌برداران سوار بالگرد هم نتوانند چنین اثری را خلق کنند بر همه‌چیز پیشی بگیرد [7].

حتی اگر قبول نداشته باشید که بیشترین و پیشروترین استفاده از پهپادهای غیرنظامی در عرصه‌ی تجاری برای فیلم‌برداری و عکاسی بوده، اما بایستی بپذیرید که پی‌بردن به بیشترین ظرفیت پهپادها و ایجاد نیاز برای طراحی پهپادهای چالاک‌تر و با کنترل حرفه‌ای‌تر حاصل همین صنعت بوده است.

صنعت تولید محتوا، سینما و تبلیغات عملاً به آزمایشگاه نوآوری برای توسعه سیستم‌های پایدارکننده‌ی تصویر، سنسورهای دقیق و کنترل پیشرفته تبدیل شد [7][8].

چون پهپاد پستچی نیازی به مانور بالا ندارد و معمولاً مسیرهای آن از پیش تعریف و استانداردسازی شده‌اند، البته درست است که اکثر پهپادهای امروزی مجهز به دوربین هستند و داشتن دوربین یک نیاز طبیعی در همه‌ی آن‌هاست تا کنترل‌کننده‌ی آن‌ها نیازی نداشته باشد به آسمان نگاه کند و وسیله‌اش را تعقیب کند و حتی قادر باشد در افق‌های دوردست به هدایت آن ادامه بدهد ولی این وظیفه‌ی پهپادهای فیلم‌برداری بود که بتوانند با سرعت بالا در هوا تغییر جهت دهند و حتی از جاهایی باریک و تنگ عبور کنند که اصولاً در طراحی آن، عبور پهپادها در نظر گرفته نشده است. احتمالاً شما هم ویدیوهای تبلیغاتی از این قبیل را برای معرفی یک ساختمان تجاری دیده‌اید.

جالب است بدانید که امروزه نه تنها گواهی هدایت پهپاد در کشورهای مختلف از جمله در ایران وجود دارد، بلکه در جهان مسابقاتی برگزار می‌شود که رانندگان یا شاید بهتر است بگوییم خلبانان بایستی پهپادهای خود را مانند یک اتومبیل، دوچرخه یا موتورسیکلت مسابقه‌ای با سرعت و بدون از دست دادن کنترل بچرخانند و به سمت خط پایان پیش ببرند، البته نه در جاده روی زمین، بلکه در مسیری سه‌بعدی که تنها با چارچوب‌ها یا علامت‌هایی مشخص شده است و معمولاً شرط بیرون‌نرفتن از مسیر، عبور از داخل تمامی حلقه‌هاست و نفرات برتر مشخص می‌شوند [9].

ورود پهپادها به میدان جنگ

همان‌طور که گفتیم امروزه پهپادها تقریباً هر کاری را که تصور می‌کردیم روزی آدم‌آهنی‌ها در میدان جنگ انجام خواهند داد، انجام می‌دهند ولی با این تفاوت که نه روی زمین راه می‌روند و نه شکل آدم‌آهنی هستند. البته شاید بتوان آن‌ها را شبیه پرندگان یا حشره‌هایی غول‌پیکر تصور کرد.

جدا از قابلیت‌هایی که برای پهپادها شمردیم، می‌توان به‌سادگی سیستم‌های جانبی مختلفی روی آن‌ها نصب کرد و کاربردهای تازه‌ای برایشان تعریف کرد؛ مثلاً بازوی رباتیکی برای حمل بار. اما آنچه باعث شد پهپادها در میدان‌های جنگ زیاد استفاده شوند، ارزان و به‌صرفه‌شدن آن‌ها بود [10].

البته تعریف ارزان و به‌صرفه با تصور رایج فرق می‌کند، مثلاً ممکن است پهپادی گران باشد اما اگر بتواند تجهیزات جنگی دشمن را که گران‌تر از خودش هستند نابود کند، در این صورت باز ارزان و مقرون به‌صرفه محسوب می‌شود. البته گذشته از این محاسبات، در میدان جنگ ممکن است به‌دلیل اهمیت جان انسان‌ها و حفظ امنیت، هزینه‌های پولی بیشتر نیز ارزشمند به حساب آید.

در سال‌های اخیر جنگ‌ها نشان داده‌اند که پهپادهای کوچک و ارزان می‌توانند توازن سنتی قدرت را تغییر دهند و حتی نقش مهمی در عملیات شناسایی و هدف‌گیری ایفا کنند [10].

آینده‌ی پهپادها به کجا می‌رود؟

در گذشته‌های دور انسان‌ها در اوقات فراغت‌شان مخصوصاً اگر زوج‌های عاشق بودند در طبیعت می‌نشستند و به ابرهای آسمان نگاه می‌کردند و سعی می‌کردند آن‌ها را به تصاویر و اشکالی تشبیه کنند. مثلاً شاید دختری به شریک زندگی‌اش می‌گفت آن ابری را که به شکل قایق است می‌بینی؟ و پسر می‌گفت نه، کدام را می‌گویی؟ آیا تو هم ابری را که به شکل یک شاخه‌ی گل است می‌بینی؟

بعدها خلبانان هواپیماهای جت تلاش کردند با پرواز گروهی و استفاده از دودهای رنگی، شکل‌هایی را در آسمان برای شهروندان آن کشور در مناسبت‌های ملی به تصویر بکشند.

اما امروزه پرواز گروهی و دقیق پهپادهاست که چه بسا حتی با

یـک برنامه‌نویسـی نسـبتاً سـاده بتـوان هـزاران تـا از آن‌هـا را بـه شـکل تصاویـر دقیـق و متحـرک مخصوصـاً به‌صـورت نورانـی در آسـمان شـب بـه نمایـش گذاشـت [8].

ایـن موضـوع کـه اصطلاحـاً بـه آن «ابـر پهپادهـا[۶]» می‌گوینـد و اصطلاحـات دیگـری ماننـد «کلونـی ریزپرنده‌هـا» نیـز بـرای آن بـه کار مـی‌رود، فقـط در امـور هنـری و تجـاری کاربـرد نـدارد، بلکـه اگـر اخبـار تکنولـوژی را دنبـال کـرده باشـید، احتمـالاً می‌دانیـد همیـن اواخـر، کشـورها و اتحادیه‌هایـی می‌خواهنـد از آن به‌عنـوان یـک لایـه‌ی دفاعـی اسـتفاده کننـد [10]. درسـت اسـت کـه پهپادهـا در مقایسـه بـا خیلـی چیزهـا مثـل موشـک‌ها و هواپیماهـا سـرعت کمتـری دارنـد و قابلیـت تعقیـب آن‌هـا را ندارنـد ولـی اگـر میـدان را عـوض کنیـم و از دیـد دیگـری نـگاه کنیـم، فقـط کافـی اسـت گـروه زیـادی از پهپادهـا بـه موقـع از زمیـن بلنـد شـوند و به‌صـورت اجتناب‌ناپذیـر در راه آن سـلاح پرتاب‌شـده یـا پرنـده‌ی نظامـی قـرار بگیرنـد تـا بـا آن‌هـا برخـورد کننـد و باعـث خسـارت، انهـدام یـا انحـراف مسـیر شـوند.

پیش‌بینـی می‌شـود آینـده‌ی پهپادهـا بیشـتر بـه سـمت هـوش مصنوعـی، پـرواز خـودکار جمعـی، کاربردهـای شـهری و خدماتـی و همچنیـن سـامانه‌های دفاعـی هوشـمند حرکـت کنـد [6][10].

۶. Drone Swarm

سخن پایانی

امیـدوارم ایـن نوشـته را پسـندیده باشـید و برایتـان مفیـد واقـع شـده باشـد. شـما در خصـوص ایـن ریزپرنده‌هـای رباتیـک چـه فکـر می‌کنیـد؟ می‌توانیـد نظـر خـود را از طریـق ایمیـل بـا مـن در میـان بگذاریـد.

شـاید مهم‌تریـن نکتـه دربـاره‌ی پهپادهـا ایـن باشـد کـه آن‌هـا تنهـا یـک ابـزار نیسـتند، بلکـه نمونـه‌ای از تغییـر نـگاه بشـر بـه فنـاوری، فضـا و حتـی مفهـوم حضـور فیزیکـی در جهـان هسـتند.

منابع

[1] Smithsonian Magazine – A Brief History of Drones

https://www.smithsonianmag.com

[2] MIT Technology Review – How consumer drones became powerful tools

https://www.technologyreview.com

[3] Encyclopedia Britannica – Unmanned Aerial Vehicle (UAV)

https://www.britannica.com

[4] IEEE Spectrum – The Rise of Quadcopters and Multirotor Drones

https://spectrum.ieee.org

[5] FAA – Drone Regulations and Safety Guidelines

https://www.faa.gov/uas

[6] TED Talk – Vijay Kumar: The future of flying robots (YouTube)

https://www.youtube.com/watch?v=4ERuO0Um0Yc

[7] National Geographic – How Drones Changed Photography and Filmmaking

https://www.nationalgeographic.com

[8] Intel Drone Light Shows – Official Presentation (YouTube)

https://www.youtube.com/watch?v=KhDEEN4gcpI

[9] Drone Racing League – Official Website & Technical Overview

https://thedroneracingleague.com

[10] CSIS / Center for Strategic and International Studies – Drones and the Future of Warfare

https://www.csis.org

بررسی و توضیح VPN ها

سیده الهه دهقان
دانشجوی مهندسی کامپیوتر
دانشکدگان فارابی دانشگاه تهران
edbelaheh85@gmail.com

۹ دقیقه

اینترنت وعدهٔ دسترسی آزاد به اطلاعات را داد، اما خیلی زود کشورها، شرکت‌ها و ارائه‌دهندگان خدمات شروع کردند به ساختن دیوار. بعضی سایت‌ها را نمی‌توان دید، بعضی سرویس‌ها فقط از یک کشور خاص قابل استفاده‌اند، و گاهی حتی یک مقالهٔ ساده دانشگاهی هم «فیلتر» می‌شود. در چنین شرایطی، اسم «وی‌پی‌ان» VPN مدام به گوشمان می‌خورد. اما وی‌پی‌ان واقعاً چیست؟ چطور کار می‌کند؟ آیا واقعا استفاده از آن امن است؟ در ادامه به پاسخ این سوالات می پردازیم.

VPN چیست؟

VPN مخفف Virtual Private Network به معنی شبکه خصوصی مجازی است. این فناوری با ایجاد یک تونل رمزگذاری‌شده بین دستگاه کاربر و اینترنت، داده‌ها را ایمن کرده و از دید اشخاص ثالث مانند هکرها، شرکت‌های تبلیغاتی و دولت‌ها پنهان می‌کند. این ویژگی باعث افزایش امنیت اطلاعات و جلوگیری از رهگیری فعالیت‌های آنلاین کاربران می‌شود.

به زبان ساده‌تر، VPN ابزاری است که آدرس IP واقعی کاربر را مخفی می‌کند و امکان اتصال به اینترنت از طریق سرورهای مختلف را می‌دهد. این کار علاوه بر افزایش حریم خصوصی، امکان دسترسی به برخی محتوای محدودشده جغرافیایی را نیز فراهم می‌کند. یعنی مثلاً شما در ایران نشسته‌اید، اما سایت مقصد فکر می‌کند دارید از هلند وارد می‌شوید.

VPN چگونه کار می کند؟

VPN (شبکه خصوصی مجازی) با ایجاد یک «تونل رمزگذاری‌شده» بین دستگاه کاربر و یک سرور در اینترنت کار می‌کند. در این فرآیند، تمام بسته‌های داده قبل از خروج از دستگاه کاربر رمزگذاری شده و درون یک پروتکل تونلینگ قرار می‌گیرند. این تونل باعث می‌شود داده‌ها در طول مسیر توسط واسطه‌های شبکه یا سیستم‌های نظارتی قابل خواندن نباشند.

در سمت دیگر تونل، سرور VPN داده‌ها را رمزگشایی کرده و آن‌ها را به مقصد نهایی در اینترنت ارسال می‌کند. پاسخ سرور مقصد نیز ابتدا به سرور VPN برمی‌گردد، دوباره رمزگذاری می‌شود و از طریق همان تونل به کاربر ارسال می‌گردد.

از دید شبکه مقصد، درخواست‌ها از آدرس IP سرور VPN ارسال شده‌اند نه از IP واقعی کاربر. به همین دلیل VPN می‌تواند برخی محدودیت‌های مبتنی بر IP یا فیلترینگ ساده را دور بزند. پروتکل‌هایی مانند OpenVPN و IKEv2 با استفاده از الگوریتم‌های رمزنگاری و مکانیزم‌های احراز هویت، امنیت و یکپارچگی این تونل ارتباطی را تضمین می‌کنند.

در ادامه به معرفی چندتا از معروف‌ترین پروتکل‌ها می پردازیم.

پروتکل‌های رایج VPN

پروتکل‌ها در واقع زبان مشترک و قوانین برقراری این تونل‌های امن هستند. معروف ترین آن‌ها عبارتند از:

1. WireGuard: یک VPN متن‌باز، مدرن و بسیار سریع است که با تمرکز بر سادگی و امنیت قوی، از الگوریتم‌های رمزنگاری جدید استفاده می‌کند. ساختار ساده و کد کم آن، بررسی امنیتی را آسان‌تر کرده و مصرف منابع و راه‌اندازی آن بهینه است. با این حال، ممکن است در برخی شبکه‌های قدیمی پشتیبانی کامل نداشته باشد و نیازمند تنظیمات اضافی برای حفظ حریم خصوصی باشد.

2. OpenVPN: یکی از محبوب‌ترین و قدیمی‌ترین پروتکل‌های تونلینگِ متن‌باز است که ترکیبی عالی از امنیت و پایداری را ارائه می‌دهد. این پروتکل از طریق پورت‌های TCP و UDP داده‌ها را منتقل می‌کند و روی اکثر سیستم‌عامل‌ها قابل اجراست. تنها ایراد آن، سرعت نسبتاً پایین‌تر در مقایسه با برخی پروتکل‌های جدیدتر است.

3. IKEv2: این پروتکل (Internet Key Exchange version 2) مبتنی بر IPSec بوده و راه‌اندازی و مدیریت VPN را کنترل می‌کند. مزیت اصلی آن انعطاف‌پذیری، امنیت بالا و سرعت فوق‌العاده در اتصال مجدد است که آن را برای موبایل‌ها ایده‌آل می‌کند. با این حال، کانفیگ آن می‌تواند دشوار باشد و گاهی توسط فایروال‌ها مسدود می‌شود.

4. IPSec: این پروتکل از داده‌ها در دو حالت انتقال (Transport Mode) یا تونل (Tunnel Mode) محافظت می‌کند و برای ایمن‌سازی ترافیک ورودی و خروجی بسیار کاربردی است. با این حال، به دلیل نیاز به قدرت پردازشی بالا، ممکن است بر عملکرد دستگاه تأثیر بگذارد. نسخه‌های جدیدتر آن از الگوریتم‌های بسیار پیچیده‌ای برای جلوگیری از هک استفاده می‌کنند.

5. SSTP: این پروتکل (Secure Socket Tunneling Protocol) در لایه ۷ مدل OSI کار می‌کند و رمزگذاری داده‌ها را به کمک SSL/TLS انجام می‌دهد. به صورت پیش‌فرض (Built-in) در ویندوز قرار دارد که راه‌اندازی آن را بسیار آسان و امن می‌کند، اما نقطه‌ضعف آن سازگاری ضعیف با سایر سیستم‌عامل‌ها (مثل اندروید یا iOS) است.

محدودیت‌ها و ریسک‌های استفاده از VPN

هرچند VPN ابزار ارزشمندی برای افزایش امنیت و عبور از فیلترینگ است، اما استفاده از آن همیشه بدون چالش نیست. در ادامه با مهم‌ترین محدودیت‌ها و ریسک‌های احتمالی آشنا می‌شویم:

کاهش سرعت و افزایش تأخیر:

رمزگذاری داده‌ها و انتقال آنها از طریق سرورهای واسط معمولاً باعث کاهش سرعت اتصال و افزایش تأخیر می‌شود، به‌خصوص در زمان استفاده از VPNهای رایگان یا سرورهای دور.

اعتماد به سرویس‌دهنده VPN:

تمام ترافیک اینترنت شما از طریق سرور VPN عبور می‌کند. اگر سرویس‌دهنده سیاست حفظ حریم شخصی شفافی نداشته باشد، یا داده‌ها را ذخیره کند، امکان افشای اطلاعات شخصی یا سوءاستفاده وجود دارد.

شناسایی و مسدودسازی توسط فیلترینگ‌ها:

فیلترینگ‌های پیشرفته می‌توانند بسته‌های VPN را شناسایی و مسدود کنند؛ بنابراین VPNها همیشه راه‌حل قطعی برای عبور از فیلترینگ نیستند.

خطرات فنی و امنیتی:

استفاده از VPNهای غیرقابل اعتماد ممکن است به نشت داده یا حملات سایبری منجر شود؛ علاوه بر این ضعف‌های پروتکل یا تنظیمات اشتباه نیز امنیت اتصال را به خطر می‌اندازند.

تبعات قانونی:

در برخی مناطق، استفاده از VPN ممکن است ممنوع یا محدود باشد و کاربران باید با قوانین محلی آشنا باشند تا گرفتار مشکلات حقوقی نشوند.

با در نظر گرفتن تمام ریسک‌های مذکور، نمی‌توان هیچ سرویس VPN را کاملاً بی‌نقص دانست. آگاهی کاربر و انتخاب آگاهانه، اصلی‌ترین عامل در بهره‌گیری امن از این فناوری است.

VPNهای دانشگاهی در ایران:

بسیاری از دانشگاه‌های معتبر در ایران، برای تسهیل دسترسی امن دانشجویان و اساتید به منابع علمی، پایگاه‌های داده پژوهشی، و گاهی حتی اینترنت داخلی با پهنای باند بیشتر، سرویس‌های VPN اختصاصی ارائه می‌دهند. این VPNها معمولاً با هدف ایجاد یک محیط امن و کنترل‌شده برای دسترسی به منابع دانشگاهی طراحی شده‌اند و اغلب محدودیت‌های کمتری نسبت به VPNهای عمومی برای دسترسی به منابع داخلی دارند.

نحوه کار و دسترسی:

این سرویس‌ها معمولاً با استفاده از پروتکل‌های امن مانند OpenVPN یا L2TP/IPSec پیاده‌سازی می‌شوند و دانشجویان برای استفاده از آن‌ها نیاز به نام کاربری و رمز عبور پرتال دانشجویی یا سامانه آموزش خود دارند. اتصال به این VPNها معمولاً فقط از داخل ایران امکان‌پذیر است و هدف اصلی آن‌ها، ایجاد یک مسیر امن بین کاربر و شبکه داخلی دانشگاه است. در ادامه چند نمونه را مشاهده می کنیم:

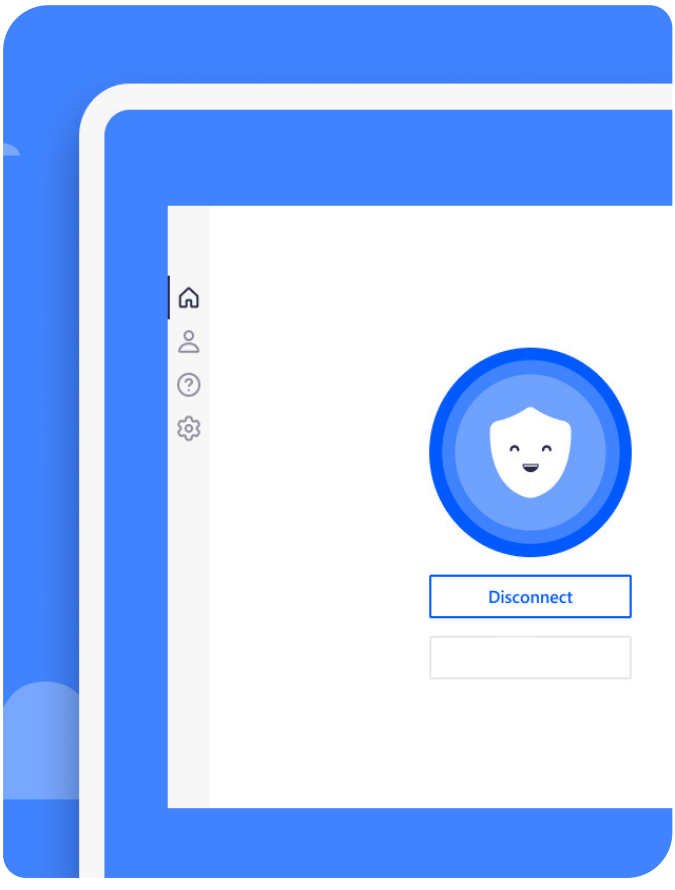
VPN دانشگاه تهران: این دانشگاه معمولاً سرویس VPN را برای دسترسی به کتابخانه دیجیتال و سایر منابع علمی در اختیار دانشجویان قرار می‌دهد. (لینک اول)

VPN دانشگاه صنعتی شریف: دانشگاه صنعتی شریف نیز در راستای تسهیل دسترسی به منابع پژوهشی و علمی، VPN مخصوص دانشجویان و پژوهشگران خود را ارائه می‌کند. (لینک دوم)

سایر دانشگاه‌ها: اکثر دانشگاه‌های بزرگ و جامع کشور، از جمله دانشگاه‌های پیام نور، آزاد اسلامی، و دانشگاه‌های مراکز استان، در حال حاضر دارای سرویس‌های VPN داخلی برای دسترسی به منابع علمی یا اینترنت داخلی هستند.

حرف آخر

VPN ابزاری است دو لبه؛ انتخاب درست آن، کارایی و امنیت را به همراه دارد، اما انتخاب اشتباه می‌تواند مشکلات زیادی ایجاد کند. سرعت، قطعی و نظارت سرویس‌های رایگان، همگی از نکاتی هستند که باید در نظر گرفت. در نهایت، VPN تنها یک ابزار است و مهم‌ترین عامل، دانش و آگاهی شما در استفاده صحیح از آن است.



آینده رشته مهندسی کامپیوتر

محدثه حاتمی کیا
دانشجوی مهندسی کامپیوتر
دانشکدگان فارابی دانشگاه تهران
hatamikia103@ut.ac.ir

۱۱ دقیقه

مردم معمولاً آینده رشته‌های کامپیوتر را با تصویرِ دسترسی آسان به منابع جهانی، پروژه‌های بین‌المللی و درآمدهای برخط تصور می‌کنند؛ اما واقعیت امروز ایران چیز دیگری است. وضعیت جنگی، اختلال در اینترنت بین‌الملل، فشار اقتصادی و تحریم‌های اقتصادی باعث شده بسیاری از مسیرهای رایج یادگیری و کار برای عموم در دسترس نباشد. با این وجود، این شرایط الزاماً به معنی «پایان فرصت‌ها» نیست؛ بلکه یعنی شکل فرصت‌ها عوض شده است. در این متن تلاش می‌کنیم آینده رشته‌های کامپیوتر و رشته‌های مشابه را نه با شعار، بلکه با نگاه به نیازهای واقعی کشور، محدودیت‌های موجود، و مهارت‌هایی که حتی در نبود دسترسی جهانی هم قابل رشد هستند بررسی کنیم.

اکوسیستمی که نفس کشیدن در آن سخت شده است

وقتی از آینده رشته‌های کامپیوتر صحبت می‌کنیم معمولاً ذهن به سمت برنامه‌نویسی، هوش مصنوعی یا امنیت سایبری می‌رود. اما در ایران امروز آینده این رشته‌ها بیش از آنکه به استعداد فردی گره خورده باشد به اکوسیستم وصل است:

- دسترسی عموم به اینترنت بین‌الملل و سرویس‌های ابری
- قیمت و دسترسی سخت‌افزار(حتی برای تمرین و پروژه)
- امنیت شغلی و توان اقتصادی بازار برای جذب نیرو
- دسترسی به منابع آموزشی به روز

پس بررسی آیندهٔ کامپیوتر یعنی بررسی این سؤال که در چه بخش‌هایی همچنان می‌شود مهارت ساخت؟ و کجاها باید واقع بینانه انتظار داشت که مسیر سخت‌تر شود؟

کار هست، اما برای چه کسانی؟

حتی وقتی شرایط اقتصادی و ارتباطی دشوار است، کشور بدون سامانه‌های نرم‌افزاری، دیتابیس، امنیت، مدیریت فرایند و خدمات دیجیتال نمی‌تواند کار کند. بنابراین نیاز به نیروهای مرتبط با کامپیوتر وجود دارد؛ فقط معمولاً چند تغییر رخ می‌دهد: پروژه‌ها کوتاه‌تر یا داخلی‌تر می‌شوند. جذب نیرو از مسیرهای غیررسمی و شبکه‌ای (ارتباطات، تیم‌های کوچک، آموزش‌های داخلی) پررنگ‌تر می‌شود. نقش‌ها ممکن است به سمت «حل مسئله‌های فوری» مثل نگه‌داری سیستم‌ها، توسعهٔ سامانه‌های داخلی، و رفع اشکال تغییر جهت بدهد. رقابت شدیدتر می‌شود، چون هزینهٔ ورود به بازار کار بالاتر رفته و افراد بیشتری دنبال راهکارند.

در شرایط جنگی، آینده شغلی چه شکلی پیدا می‌کند؟

۱. قطع و محدودیت اینترنت بین‌الملل

قطع اینترنت بین‌الملل برای خیلی‌ها صرفاً یعنی «یوتیوب یا گیت‌هاب باز نیست». در عمل اثرش چند لایه است:

یادگیری: دسترسی به دوره‌های به‌روز، مستندات رسمی، تمرین‌های آنلاین، و جامعه توسعه‌دهندگان محدود می‌شود.

توسعه و ابزارها: اگر ابزارهای خارجی، سرویس‌های مدیریت کانت/پردازش، یا وابستگی به APIهای بیرونی وجود داشته باشد، توسعه نرم‌افزار کند یا متوقف می‌شود.

پروژه و همکاری: کار تیمی با منابع پراکنده و CI/CDهای وابسته به سرویس‌های خارجی سخت‌تر می‌شود.

چالش هزینه: حتی اگر راه جایگزین وجود داشته باشد، هزینه تمرین و راه‌اندازی بالا می‌رود (سرور، لایسنس‌ها، کانفیگ و...).

اما روی دیگر ماجرا این است که این محدودیت‌ها باعث می‌شود مسیر «یادگیری آفلاین» ارزش بیشتری پیدا کند. یعنی مهارت‌هایی که می‌شود روی سیستم محلی، با مستندات ذخیره‌شده، پروژه‌های مستقل و اتکا به دانش پایه جلو برد.

۲. بحران اقتصادی:

این شرایط باعث تغییر در اولویت شرکت‌ها شده؛ مهارت‌های عملی ارزشمندتر می‌شوند وقتی اقتصاد تحت فشار است، شرکت‌ها معمولاً در جذب نیرو ریسک کمتری می‌کنند. بنابراین مهارت‌های شما باید بیشتر شبیه «ابزار حل مسئله» باشند تا صرفاً

«دانـش تئـوری». در نتیجـه مهارت‌هـای Backend، پایـگاه‌داده، API، سیسـتم‌های توزیع‌شـده داخلـی، امنیـت پایـه و مدیریـت دسترسـی محبوب‌تـر می‌شـوند. مهارت‌هـای دیبـاگ، عملکـرد (Performance)، معمـاری قابـل نگـه‌داری ارزش بالاتـری دارنـد. مهارت‌هـای صرفاً وابسـته بـه سـرویس‌های بیرونـی یـا کار کامـلاً برخـط ممکـن اسـت کمتـر مطلـوب باشـد مگـر اینکـه تیـم واقعـاً راهـکار داخلـی داشـته باشـد. نکتـه کلیـدی کـه وجـود دارد ایـن اسـت کـه در شـرایط سـخت، پـروژه واقعـی و قابـل اسـتقرار (حتـی کوچـک) از صرفـاً داشـتن مـدرک یـا دوره بیشـتر کمـک می‌کنـد.

در این طوفان، کدام مسیرها ماندگارترند؟

اگـر بخواهیـم تصویـر واقع‌بینانـه داشـته باشـیم، شـاخه‌هایی که بـه حداقل وابسـتگی خارجـی نیـاز دارنـد و بیشـتر بـه دانـش پایـه و مهـارت عملـی اتکا می‌کننـد، معمـولاً پایدارترند.

شاخه‌هایی همچون:

برنامه‌نویسی و معماری نرم‌افزار (قابل پیاده‌سازی داخلی)

Backend و طراحی API (برای ساخت سامانه‌های سازمانی)

پایگاه‌داده و مهندسی داده پایه (ذخیره/پردازش محلی)

امنیت اپلیکیشن و اصول ایمن‌سازی (برای سیستم‌های داخلی)

DevOps بومی/محلی (مانیتورینگ، استقرار، کانفیگ روی محیط‌های خودی)

چالش‌های روانی و اجتماعی:

ناامیـدی، تـرس از عقـب مانـدن، و فرسـودگی شـرایط سـخت فقـط فنـی نیسـت. افـراد ممکـن اسـت بـا چند فشـار ذهنـی روبه‌رو شـوند: حـس اینکه «مـن بـه منابـع جهانـی دسترسـی نـدارم پـس عقـب افتـادم» مقایسـه بـا کسـانی کـه آنلایـن فعال‌انـد، فرسـودگی ناشـی از اینکـه پروژه‌هـا مـدام بـه مانـع می‌خورنـد، بلاتکلیفـی در برنامـه یادگیـری.

توصیـه می‌شـود ایـن مسـیر را بـه «گام‌هـای کوچـک قابـل اجـرا» تبدیـل کنیـد. یادگیـری بـدون وابسـتگی خارجی و سـاخت پروژه‌هـای کم‌حجم ولی واقعـی، می‌توانـد از ناامیـدی جلوگیـری کنـد.

راهکارهای عملی برای دانشجویان و علاقه‌مندان به این حوزه:

بـرای اینکـه آینـده کامپیوتـر در ایـن شـرایط "قابـل لمـس" شـود، یکسـری رویکردهـا پیشـنهاد می‌شـود. اینکـه روی پایه‌هـا و مهارت‌هـای قابـل تمرین آفلایـن سـرمایه‌گذاری شـود. مثـل الگوریتـم و سـاختار داده برنامه‌نویسـی تمیـز، تسـت، دیبـاگ کار بـا دیتابیـس و طراحـی API.

ساخت پروژه، حتی کوچک:

یـک پـروژه بهتـر از ده دوره پراکنده اسـت؛ چون پروژه نشـان می‌دهد شـما می‌توانیـد مشـکل واقعـی را حل کنید.

نمونه پروژه‌های کم‌وابسته:

- API بـرای مدیریـت یـک سیسـتم سـاده سیسـتم گزارش‌دهی/ثبت رویـداد بـه دیتابیس
- داشبورد محلی با داده‌های نمونه
- سرویس تشخیص/طبقه‌بندی متنی با دیتای بومی

مهارت‌های سازگارسازی را یاد بگیرید:

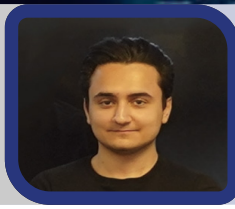
در شـرایط محـدود، کسـی موفق‌تـر اسـت کـه بتواند سیسـتم را بـا واقعیت محیطـش تنظیـم کنـد. کانتینـر و اسـتقرار محلـی (تـا حـد امـکان)، مدیریت وابسـتگی‌ها (Dependency Management)، اجـرای پـروژه بـا محیـط کنترل‌شـده.

ارتباط شبکه‌ای داخلی بساز:

وقتـی دسترسـی جهانـی محـدود می‌شـود، شـبکه داخل کشـور خیلی مهم می‌شـود ماننـد گروه‌هـای دانشـجویی، انجمن‌هـای محلی توسـعه‌دهندگان، تبـادل مسـتندات و پروژه‌ها.

و امـا در آخـر بایـد گفـت آینـدهٔ کامپیوتـر «حذف نمی‌شـود»، اما «شـکلش عـوض می‌شـود» در ایرانِ شـرایط جنگـی و محدودیـت اینترنـت بین‌المللی، آینـدهٔ رشـته‌های کامپیوتر یـک چرخش مهـم دارد: کمتر به دنبال مسـیرهای وابسـته به سـرویس‌های خارجی، بیشـتر به سـمت راه‌حل‌های قابـل اجرا در محیـط داخلـی. تمرکـز بـر پروژه‌های واقعـی، مهارت‌هـای عملـی، و یادگیری قابـل ادامـه. پـس اگرچـه شـرایط امـروز سـخت‌تر از گذشـته اسـت، امـا کامپیوتـر هنـوز یک راه جدی برای سـاختن آینده اسـت به شـرطی که مسـیر را بـر اسـاس واقعیـت تنظیـم کنیـم، نه بر اسـاس تصور رایج.

جنگ در لایه رابط‌های برنامه‌نویسی (API):
کالبدشکافی معماری نرم‌افزاری هوش مصنوعی در سیستم‌های رزمی



ابوالفضل توکلیان
دانشجوی مهندسی کامپیوتر
دانشکدگان فارابی دانشگاه تهران
abolfazltavakolian57@gmail.com

۱۵ دقیقه

چگونه یک مدل زبانی به موتور تصمیم‌گیری میدان نبرد تبدیل می‌شود؟

وقتی صحبت از درگیری‌های نظامی نوین و ورود هوش مصنوعی‌های پیشرفته مانند کلود (Claude) به میدان نبرد می‌شود، رسانه‌های عمومی معمولاً روی مفاهیمی مثل «سلاح‌های خودمختار» و «اخلاق» تمرکز می‌کنند. اما برای یک مهندس نرم‌افزار، سوال اصلی این است: یک مدل زبانی بزرگ (LLM) که اساساً برای پیش‌بینی کلمه بعدی آموزش دیده، چگونه می‌تواند ۱۳ هزار هدف نظامی را در یک سیستم پیچیده مانند پلتفرم پلانتیر (Palantir) شناسایی و اولویت‌بندی کند؟

پاسخ این سوال در رابط‌های کاربری گرافیکی چت‌بات‌ها نیست؛ بلکه در اعماق معماری میکروسرویس‌ها، خطوط لوله داده (Data Pipelines) و فراخوانی‌های ساختاریافته وب‌سرویس‌ها نهفته است. در این مقاله، معماری نرم‌افزاری پشت این پدیده را از منظر مهندسی سیستم‌ها بررسی می‌کنیم.

۱. از سنسور تا سرور: خط لوله داده‌های چندوجهی

در یک سناریوی عملیاتی واقعی، هوش مصنوعیِ کلود مستقیماً به دوربین یک پهپاد متصل نیست. معماری سیستم‌های فرماندهی نوین بر اساس هم‌جوشی داده‌ها (Data Fusion) کار می‌کند. جریان پردازش اطلاعات به این شکل است:

لایه پردازش لبه (Edge Computing): سنسورها شامل تصاویر ماهواره‌ای، شنود رادیویی و رادارها داده‌های خام را جمع‌آوری می‌کنند. شبکه‌های عصبی پیچشی (CNN) که به صورت محلی روی خود پهپادها نصب شده‌اند، اشیای اولیه را تشخیص داده و ویدیوهای سنگین را به متادیتا (Metadata) تبدیل می‌کنند تا در پهنای باند شبکه صرفه‌جویی شود.

لایه تجمیع و پایگاه‌های داده برداری (Vector DBs): این متادیتاها در کنار داده‌های متنی (مثل گزارش‌های اطلاعاتی شنود شده) به بردارهای عددی (Embeddings) تبدیل شده و در دیتابیس‌های برداری ذخیره می‌شوند.

لایه ارکستراسیون (Orchestration): اینجا جایی است که سیستم یکپارچه‌ساز وارد عمل می‌شود و با استفاده از معماری تولید افزوده با بازیابی (RAG)، داده‌های مرتبط را فیلتر کرده و برای تغذیه به مغز متفکر سیستم آماده می‌کند.

۲. کلود در نقش موتور استدلال: آناتومی یک درخواست API

در این معماری، کلود به عنوان یک «موتور استدلال» (Reasoning Engine) عمل می‌کند. سیستم نظامی یک درخواست از طریق API به سرورهای کلود می‌فرستد. این درخواست یک متن ساده نیست، بلکه یک بسته داده ساختاریافته (JSON) است که به شدت مهندسی پرامپت (Prompt Engineering) روی آن اعمال شده است.

نمونه‌ای ساده‌سازی‌شده از داده‌های ارسالی به API کلود:

```
[POST /v1/messages]

{
  "model": "claude-3-opus-military-grade",
  "system_prompt": "You are a CENTCOM intel analysis engine. Analyze
                    sensor metadata against RoE and IHL. Output must
                    be strictly formatted as a structured JSON.",
  "messages": [
    {
      "role": "user",
      "content": "Sensor data received from X-72:\n
                  - Object: Concrete structure, 2000 sqm\n
                  - Thermal signature: High (indicates servers/heavy)
                  - Verified plates history: IRGC linked (87% match)\n
                  - Distance to nearest civilian center: 1.2 km\n
                  Evaluate military target probability and calculate
                  Collateral Damage Estimation (CDE) metrics."
    }
  ]
}
```

کلود این داده‌ها را پردازش کرده و پاسخی برمی‌گرداند (مثلاً درصد اطمینان ۹۲ درصد برای هدف نظامی). این خروجی، توسط سیستم رابط کاربری روی داشبورد فرمانده انسانی به عنوان یک «پیشنهاد شلیک» ظاهر می‌شود. این چرخه که پیش‌تر توسط تیم‌های انسانی روزها طول می‌کشید، حالا در سطح سرور در چند میلی‌ثانیه انجام می‌شود.

۳. پارادوکس عملیاتی: چرا ارتش نمی‌تواند هوش مصنوعی را خاموش کند؟

یکی از جذاب‌ترین پدیده‌های مهندسی نرم‌افزار در مناقشات میان شرکت‌های فناوری و دولت‌ها، مسئله بدهی فنی و وابستگی عمیق در میکروسرویس‌ها است.

وقتی دستور توقف استفاده از یک هوش مصنوعی خاص صادر می‌شود، فرماندهان در میدان نبرد نمی‌توانند آن را فورا اجرا کنند. در یک معماری مدرن نرم‌افزاری، هوش مصنوعی یک افزونه ساده نیست که با یک دکمه خاموش شود. فراخوانی‌های API در صدها میکروسرویس مختلف تنیده شده‌اند:

ماژول ترجمه همزمان بی‌سیم‌ها.

ماژول مسیریابی لجستیک.

ماژول ارزیابی خسارت پس از حمله.

جایگزینی یک مدل پایه با مدل شرکتی دیگر، نیازمند بازنویسی هزاران خط کد، تغییر ساختار پرامپت‌ها و ماه‌ها تست رگرسیون (Regression Testing) برای جلوگیری از خطاهای مرگبار است. غیرفعال کردن ناگهانی API در میان یک عملیات، عملاً به معنای فلج کردن کل سیستم یکپارچه فرماندهی است.

۴. سوگیری اتوماسیون و باگ‌های مرگبار

از منظر علوم کامپیوتر، پدیده‌ای به نام «توهم» (Hallucination) در مدل‌های زبانی غیرقابل اجتناب است؛ زیرا آن‌ها سیستم‌های قطعی و ریاضی‌وار نیستند، بلکه مبتنی بر احتمالات کار می‌کنند.

هنگامی که سیستمی روزانه هزاران درخواست اطلاعاتی را پردازش کرده و اهداف را پیشنهاد می‌کند، رابط کاربری طوری طراحی می‌شود که انسان تصمیم‌گیرنده نهایی باشد (حلقه نظارت انسانی). اما از نظر روانشناسی تعامل انسان و کامپیوتر (HCI)، وقتی یک نرم‌افزار ۹۹ بار خروجی درست می‌دهد، اپراتور دچار سوگیری اتوماسیون (Automation Bias) شده و برای صدمین بار دیگر داده‌ها را بررسی نمی‌کند و صرفاً دکمه تایید را می‌فشارد.

همین مسئله فنی، ریشه خطاهای فاجعه‌بار در هدف‌گیری‌ها است. مقصر اصلی در این سناریوها نه هوش مصنوعی است و نه اپراتور؛ بلکه مقصر، مشکل همگام‌سازی پایگاه داده با واقعیت پویای میدان نبرد و اتکای بیش از حد به خروجی‌های احتمالی است.

نتیجه‌گیری: ورود هوش مصنوعی به سیستم‌های نظامی، ماهیت نبرد را به یک «عملیات پیچیده مهندسی نرم‌افزار در مقیاس کلان» تغییر داده است. امروزه، پایداری زیرساخت‌های ابری، سرعت پاسخ‌دهی سرورها و معماری بدون نقص میکروسرویس‌ها، به اندازه کیفیت تجهیزات فیزیکی، تعیین‌کننده نتیجه عملیات‌ها هستند.

نقد فیلم WarGames (۱۹۸۳)

کارگردان: John Badham

بازیگران: متیو برودریک، الای شیدی، دابنی کلمن، جان وود

ژانر: علمی‌تخیلی، تریلر، درام، سیاسی

امتیاز IMDb: ۷٫۱

۱۱۰ دقیقه

فاطمه نصیری
دانشجوی مهندسی کامپیوتر
دانشکدگان فارابی دانشگاه تهران
fatemen.eng@gmail.com

خلاصه داستان

دیوید لایتمن، نوجوانی علاقه‌مند به کامپیوتر، هنگام جست‌وجوی بازی‌های جدید، به‌طور اتفاقی وارد یک ابررایانه نظامی متعلق به وزارت دفاع آمریکا می‌شود. او تصور می‌کند در حال انجام یک بازی رایانه‌ای است، اما در حقیقت سامانه‌ای را فعال می‌کند که مسئول تصمیم‌گیری درباره‌ی جنگ هسته‌ای است. اکنون دیوید باید قبل از آنکه یک اشتباه ساده به جنگ جهانی سوم تبدیل شود، سیستم را متوقف کند.

??? آیا باید تصمیم‌های حیاتی بشر را به ماشین‌ها سپرد؟

یکی از مهم‌ترین پرسش‌هایی که WarGames مطرح می‌کند، این است که مرز اعتماد به فناوری کجاست؟ در فیلم، ابررایانه‌ای به نام Joshua برای مدیریت و شبیه‌سازی جنگ هسته‌ای طراحی شده است. منطق پشت این تصمیم ساده است؛ ماشین احساس ندارد، اشتباهات انسانی مانند خستگی، ترس یا هیجان را تجربه نمی‌کند و می‌تواند در کسری از ثانیه میلیون‌ها محاسبه انجام دهد. اما فیلم خیلی زود نشان می‌دهد که همین منطق، اگر بدون در نظر گرفتن قضاوت انسانی دنبال شود، می‌تواند به فاجعه ختم شود. فیلم در واقع هشدار می‌دهد که هوش مصنوعی و سامانه‌های خودکار، هرقدر هم پیشرفته باشند، به‌خودی‌خود فاقد قضاوت انسانی درباره‌ی مفاهیمی مانند اخلاق، همدلی، ارزش جان انسان یا پیامدهای اجتماعی تصمیم‌های خود هستند. آن‌ها می‌توانند احتمال موفقیت یک عملیات را محاسبه کنند، اما نمی‌توانند ارزش صلح را درک کنند.

نکات مثبت فیلم

نمایش واقع‌گرایانه‌ی فرهنگ هکرها: برخلاف بسیاری از فیلم‌های هالیوودی، WarGames هک را با جادوی کامپیوتری نمایش نمی‌دهد. دیوید با مودم، شماره‌گیری تلفنی و جست‌وجوی سیستم‌ها به شبکه‌های مختلف متصل می‌شود؛ روشی که برای آن دوران کاملاً واقع‌گرایانه بود.

تعلیق بالا: فیلم بدون نیاز به انفجارهای متعدد یا صحنه‌های اکشن اغراق‌آمیز، تنها با افزایش تدریجی تنش، مخاطب را تا پایان درگیر نگه می‌دارد.

پرداختن به مفهوم Human in the Loop: امروزه در هوش مصنوعی اصطلاحی وجود دارد به نام Human in the Loop؛ یعنی انسان باید در تصمیم‌های حساس حضور داشته باشد. جالب اینجاست که WarGames بیش از چهل سال قبل، همین ایده را به تصویر کشیده است و نشان می‌دهد حذف انسان از فرایند تصمیم‌گیری چه پیامدهایی دارد.

تأثیر تاریخی: فیلم WarGames تنها یک فیلم موفق نبود؛ پس از اکران آن، نگرانی‌ها درباره‌ی آسیب‌پذیری سامانه‌های رایانه‌ای در آمریکا افزایش یافت و توجه دولت این کشور به امنیت سایبری بیشتر شد. این فضای نگرانی در شکل‌گیری و تصویب Computer Fraud and Abuse Act (CFAA) در سال ۱۹۸۶ نیز مؤثر بود؛ قانونی که دسترسی غیرمجاز به سامانه‌های رایانه‌ای و بسیاری از جرایم سایبری را جرم‌انگاری کرد.

بازیگران: متیو برودریک در نقش «دیوید لایتمن» بازی باورپذیری ارائه می‌دهد. او شخصیتی را به تصویر می‌کشد که نه یک نابغه شکست‌ناپذیر، بلکه نوجوانی کنجکاو و گاهی بی‌احتیاط است؛ ویژگی‌ای که باعث می‌شود مخاطب راحت‌تر با او همذات‌پنداری کند. همچنین شخصیت «Joshua» با وجود نداشتن حضور فیزیکی، تنها از طریق صدا و تعامل با دیگر شخصیت‌ها، به یکی از به‌یادماندنی‌ترین کاراکترهای فیلم تبدیل می‌شود.

نکات منفی فیلم

قدیمی‌شدن برخی جنبه‌های فنی: بخش زیادی از فناوری‌های فیلم متعلق به دهه‌ی ۸۰ است و برای مخاطب امروز شاید کند یا ساده جلوه کند.

ساده‌سازی برخی مفاهیم امنیتی از نگاه متخصصان امنیت: نفوذ دیوید به سامانه‌ی نظامی بیش‌ازحد آسان نمایش داده شده است. امروزه چنین سطحی از دسترسی معمولاً با موانع امنیتی بسیار بیشتری روبه‌روست و حتی در همان زمان نیز برخی منتقدان آن را ساده‌سازی‌شده می‌دانستند.

ریتم کند برای مخاطب امروز: فیلم با استانداردهای دهه‌ی ۸۰ ساخته شده است؛ بنابراین برخی سکانس‌ها برای مخاطب امروزی کند به نظر می‌رسند و جلوه‌های بصری آن نیز محدودیت‌های فناوری زمان خود را نشان می‌دهند. شخصیت‌پردازی محدود: به‌جز دیوید و Joshua، بسیاری از شخصیت‌ها عمق کافی ندارند و بیشتر برای پیشبرد داستان حضور دارند.

دیالوگ برتر فیلم

“A strange game. The only winning move is not to play.”

چه بازی عجیبی... تنها راه پیروزی، بازی نکردن است.

نقد فیلم

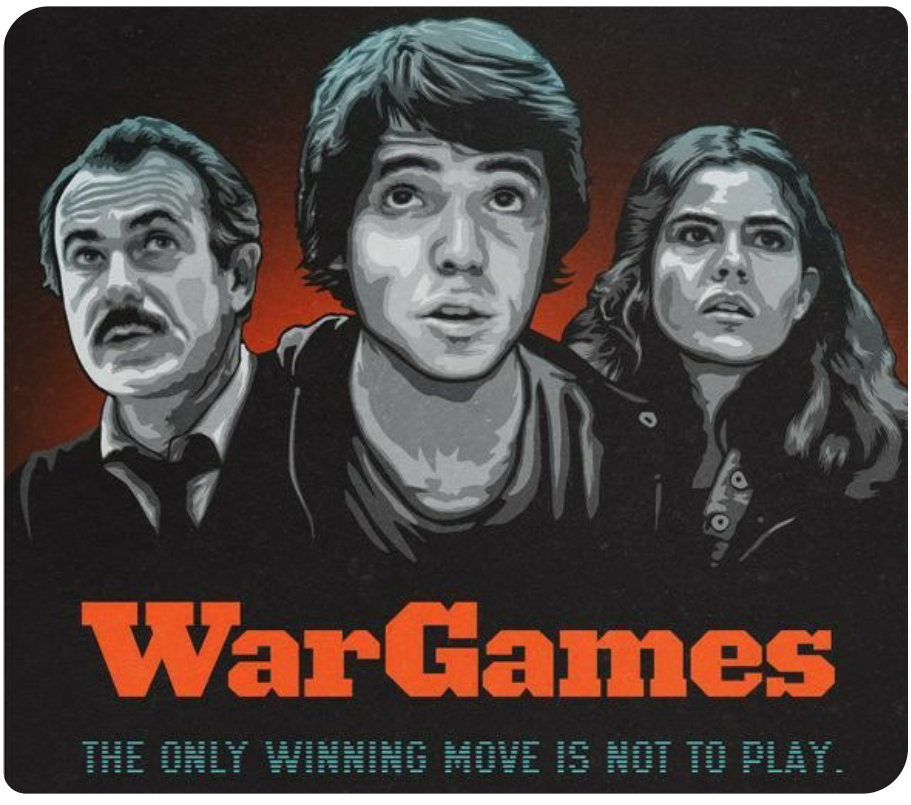
WarGames را نباید صرفاً یک فیلم علمی‌تخیلی دهه‌ی هشتاد دانست؛ این فیلم هشداری است درباره‌ی رابطه‌ی انسان و فناوری؛ هشداری که با گذشت بیش از چهار دهه، نه‌تنها کهنه نشده، بلکه با ظهور هوش مصنوعی، سامانه‌های خودکار و حملات سایبری، حتی واقعی‌تر به نظر می‌رسد.

جان بدهام، بدون آنکه مخاطب را با اصطلاحات پیچیده خسته کند، مفاهیمی مانند امنیت سایبری، اعتماد به سیستم‌های خودکار و مسئولیت اخلاقی مهندسان را وارد داستانی سرگرم‌کننده می‌کند. نقطه‌ی قوت فیلم این است که تهدید اصلی را نه یک هکر نابغه یا یک ابرشرور، بلکه اشتباه انسان در طراحی و استفاده از فناوری معرفی می‌کند.

فیلم همچنین مفهوم بازدارندگی هسته‌ای را زیر سؤال می‌برد. Joshua پس از شبیه‌سازی هزاران سناریوی جنگ به نتیجه‌ای می‌رسد که شاید مهم‌ترین پیام کل فیلم باشد: برخی بازی‌ها اساساً برنده‌ای ندارند.

نتیجه‌گیری

وقتی WarGames در سال ۱۹۸۳ اکران شد، هوش مصنوعی بیشتر یک ایده‌ی علمی‌تخیلی بود. امروز، در دورانی که مدل‌های زبانی، خودروهای خودران و سامانه‌های تصمیم‌یار در حال ورود به بخش‌های حساس زندگی هستند، پرسش اصلی فیلم همچنان پابرجاست: فناوری تا کجا باید اختیار داشته باشد و از کجا به بعد، تصمیم نهایی باید در اختیار انسان بماند؟ شاید به همین دلیل است که WarGames پس از چهار دهه، نه فقط یک فیلم کلاسیک، بلکه هشداری برای آینده به‌شمار می‌رود.



چگونه ویدیوگیم‌ها داستان تعریف می‌کنند؟ روایت‌گری در جهان پیکسل‌ها و کدها

علیرضا کامرو
دانشجوی مهندسی کامپیوتر
دانشکدگان فارابی دانشگاه تهران
alirezakamru@gmail.com

عرفان رمضانی مقدم
دانشجوی مهندسی کامپیوتر
دانشکدگان فارابی دانشگاه تهران
erfan.ramezani@ut.ac.ir

احمد محمدی
دانشجوی مهندسی کامپیوتر
دانشکدگان فارابی دانشگاه تهران
mohammadi.ahmad@ut.ac.ir

مقدمه — «داستانی که فقط دیده نمی‌شود، بازی می‌شود»

وقتی درباره‌ی داستان یک فیلم سینمایی یا یک رمان صحبت می‌کنیم، تمام مخاطبان در یک نقطه‌ی مشترک ایستاده‌اند: همه‌ی آن‌ها روایتی کاملاً یکسان را تجربه کرده‌اند. نئون‌های ارغوانی شهر، تصمیمات نهایی شخصیت اصلی و تیتراژ پایانی، برای همه به یک شکل ثبت می‌شود. اما در دنیای ویدیوگیم، داستان شکل کاملاً متفاوتی به خود می‌گیرد. ممکن است دو دوست یک بازی مشابه را به پایان برسانند، اما احساس، برداشت و حتی مسیر روانی آن‌ها در مواجهه با جهان آن اثر، زمین تا آسمان متفاوت باشد. چه چیزی باعث می‌شود ویدیوگیم بتواند به شیوه‌ای کاملاً متمایز از سینما و ادبیات دست به داستان‌گویی بزند؟

پاسخ در یک کلیدواژه‌ی طلایی نهفته است: «تعامل».

۱۰ دقیقه

در بازی‌های ویدیویی، مخاطب دیگر یک ناظر منفعل در فاصله‌ای امن از صفحه‌نمایش نیست؛ او قدم می‌زند، تصمیم می‌گیرد، شکست می‌خورد، تاوان می‌دهد و مسیر قصه را پیش می‌برد. ویدیوگیم به ما اجازه می‌دهد نه‌تنها شاهد یک قصه باشیم، بلکه نبض حیاتی آن را در دست بگیریم. این جادوی رسانه‌ای مدرن است که مرز میان «نویسنده» و «تجربه‌کننده» را کم‌رنگ می‌سازد. داستان‌گویی در بازی‌ها فرمولی واحد ندارد؛ از روایت‌های خطی و سینمایی گرفته تا جهان‌های آزاد و قصه‌هایی که از دل قوانین گیم‌پلی متولد می‌شوند، همگی ابزارهایی هستند برای پاسخ به یک سؤال بنیادین: «چطور می‌توان داستان را نه فقط دید و شنید، بلکه زندگی و بازی کرد؟»

۱. روایت خطی؛ وقتی بازی داستان را هدایت می‌کند

اگر سینما را نزدیک‌ترین خویشاوند ویدیوگیم بدانیم، روایت خطی[1] سنتی‌ترین و درعین‌حال قدرتمندترین پیوند میان این دو رسانه است. در این سبک، نویسندگان و کارگردانان تسلط کاملی بر ریتم، ترتیب وقایع و منحنی احساسی مخاطب دارند. بازی‌ساز مسیری مشخص و دقیق از نقطه‌ی آغاز تا پایان طراحی می‌کند و بازیکن برای پیشبرد داستان، ملزم به طی‌کردن دقیق همین ریلِ تعیین‌شده است.

بزرگ‌ترین مزیت روایت خطی، تمرکز بالا و انسجامِ درام است. وقتی سازندگان مجبور نباشند صدها حالت گوناگون یا تصمیمات متفاوت بازیکن را مدیریت کنند، تمام توان و

۱. Linear Storytelling

قدرت هنری خود را روی شخصیت‌پردازی‌های عمیق، ریتم‌دهی دقیق به اوج و فرودهای داستان و خلق صحنه‌های حماسی می‌گذارند.

البته این فرمت خالی از چالش‌های بنیادین نیست. یکی از مهم‌ترین آن‌ها، خطر بروز پدیده‌ای به نام تناقض روایی-گیم‌پلی[۲] است؛ زمانی که چیزی که داستان درباره‌ی یک شخصیت یا جهان می‌گوید، با رفتاری که بازی از بازیکن می‌خواهد، هماهنگ نباشد. برای مثال، ممکن است شخصیت اصلی در داستان فردی دلسوز و آرام به نظر برسد، اما در جریان گیم‌پلی، بدون هیچ واکنش جدی ده‌ها دشمن را از بین ببرد. وقتی این دو تصویر بیش‌ازحد از هم فاصله بگیرند، باورپذیری داستان آسیب می‌بیند.

برای کاهش این فاصله، بازی‌سازان تلاش می‌کنند روایت و گیم‌پلی را هم‌سو کنند؛ این هماهنگی از طریق طراحی مکانیک‌هایی متناسب با شخصیت، محدود کردن رفتارهای متناقض یا ایجاد پیامدهای روایی برای اعمال بازیکن به‌وجود می‌آید. در این حالت، کاری که بازیکن انجام می‌دهد فقط وسیله‌ای برای پیشروی نیست، بلکه بخشی از همان تصویری است که داستان از جهان بازی ارائه می‌کند.

بااین‌حال، نباید دچار یک سوءتفاهم رایج شد: خطی بودن لزوماً به معنای ساده بودن یا محدود کردن مخاطب نیست. ارزش و کیفیت یک روایت به تعداد مسیرهای انتخابی بستگی ندارد. روایت خطی ثابت کرده است که اگر درست پیاده‌سازی شود، می‌تواند ماندگارترین تجربیات تاریخ ویدیوگیم را خلق کند؛ جایی که بازیکن با میل خود، سکان هدایت قصه را به دستانِ توانا و رؤیاپرداز سازندگان می‌سپارد.

نمونه‌های شاخص:

The Last of Us

Uncharted 4: A Thief's End

Titanfall 2

Alan Wake 2

۲. روایت انتخاب‌محور؛ وقتی بازیکن شریک نویسنده است

اگر در روایت خطی بازیکن یک «همسفرِ تماشاگر» است، در روایت انتخاب‌محور[۳] قلم نویسندگی تا حد زیادی به خود او سپرده می‌شود. در این فرمت، داستان به‌جای یک خط مستقیم و یگانه، مانند یک درخت پرشاخ‌وبرگ طراحی می‌شود. بازیکن در طول مسیر با دوراهی‌های اخلاقی و تصمیم‌های سرنوشت‌سازی روبه‌رو می‌شود که نه‌تنها سرنوشت روایت، بلکه شکل روابط میان شخصیت‌ها و پایان‌بندی نهایی قصه را بازنویسی می‌کنند.

بااین‌حال، تحلیلی‌ترین ویژگی این فرمت داستانی، درک تفاوت ظریف میان «توهم انتخاب» و «انتخاب واقعی» است.

در «توهم انتخاب»[۴]، بازی‌ساز حس کاذب کنترل را به بازیکن القا می‌کند؛ گزینه‌های مختلفی در اختیار او قرار می‌گیرد، اما در نهایت بیشتر آن‌ها به یک مسیر مشترک ختم می‌شوند. ممکن است دیالوگ کمی تغییر کند یا واکنش کوتاهی ببینیم، اما اتفاق اصلی همچنان همان چیزی است که سازندگان از ابتدا تعیین کرده‌اند. این تکنیک به سازندگان اجازه می‌دهد بدون تحمل هزینه‌های سنگین تولید، حس عاملیت[۵] را زنده نگه دارند.

در مقابل، «انتخاب واقعی» زمانی شکل می‌گیرد که تصمیم بازیکن پیامدی ملموس و معنادار داشته باشد. در این ساختار لایه‌لایه، ممکن است یک شخصیت زنده بماند یا بمیرد، یک مأموریت مسیر متفاوتی پیدا کند یا بخش عظیمی از جهان بازی برای عده‌ای کاملاً غیرقابل‌دسترس شود. لازم نیست هر انتخاب به پایانی کاملاً متفاوت منجر شود؛ مهم این است که بازیکن بتواند اثر تصمیم خودش را در داستان ببیند. طراحی چنین ساختاری کار نویسندگان را به‌شدت سخت می‌کند، زیرا داستان باید در تمام شاخه‌ها و مسیرهای مختلف، منطقی و منسجم باقی بماند.

در نهایت، قدرت اصلی این نوع روایت در ایجاد احساس مسئولیت است. در یک داستان خطی، اتفاقات صرفاً برای بازیکن رخ می‌دهند؛ اما در روایت انتخاب‌محور، بخشی از آن وقایع نتیجه‌ی مستقیم تصمیمات خود او هستند. همین تفاوت بنیادین، ویدیوگیم را از یک قصه بیرونی که تنها تماشایش می‌کنیم، به تجربه‌ای عمیقاً شخصی تبدیل می‌کند که احساس می‌کنیم در خلق آن سهم داشته‌ایم.

نمونه‌های شاخص:

Detroit: Become Human

Disco Elysium

Mass Effect Trilogy

The Walking Dead: The Telltale Series

۳. روایت محیطی؛ وقتی جهان بازی داستان را تعریف می‌کند

در بسیاری از آثار، مخاطب داستان را از طریق دیالوگ‌ها یا راوی درک می‌کند؛ اما ویدیوگیم‌ها امکان دیگری نیز در اختیار دارند: محیط. در روایت محیطی[۶]، جهان بازی تنها محلی برای اتفاق افتادن داستان نیست، بلکه خودش به ابزاری برای روایت تبدیل می‌شود. ساختمان‌ها، معماری، اشیا و آیتم‌ها و توضیحات آن‌ها، وضعیت شخصیت‌ها و حتی

۲. Ludonarrative Dissonance

۳. Choice-Driven Narrative

۴. Illusion of Choice

۵. Sense of Agency

۶. Environmental Storytelling

نحوه‌ی قرارگرفتن عناصر مختلف در یک محیط می‌توانند اطلاعاتی درباره‌ی تاریخ، فرهنگ و اتفاقات جهان بازی در اختیار بازیکن قرار دهند.

مهم‌ترین ویژگی این شیوه‌ی روایت، غیرمستقیم بودن آن است. بازی لزوماً نمی‌گوید چه اتفاقی افتاده و در واقع، داستان را یک‌جا و ساده در اختیار مخاطب قرار نمی‌دهد، بلکه نشانه‌هایی در اختیار بازیکن قرار می‌دهد تا خودش آن را کشف کند. برای مثال، ورود به یک خانه متروکه که هنوز وسایل زندگی در آن باقی‌مانده، می‌تواند بدون هیچ دیالوگی این پرسش را ایجاد کند که ساکنان آن چه سرنوشتی پیدا کرده‌اند. به این ترتیب، بازیکن از دریافت‌کننده‌ی صرف و منفعل اطلاعات، به فردی تبدیل می‌شود که باید خودش میان نشانه‌های مختلف ارتباط برقرار کند، تکه‌های پازل را کنار یکدیگر قرار دهد و داستان را از دل محیط بیرون بکشد.

روایت محیطی در بسیاری از موارد برای نشان‌دادن گذشته‌ی یک جهان مورد استفاده قرار می‌گیرد. یک شهر ویران‌شده، مجسمه‌های فرسوده، بقایای یک تمدن از بین رفته یا حتی تفاوت معماری مناطق مختلف می‌تواند بدون توضیح مستقیم، تاریخچه‌ای[۷] برای آن جهان ایجاد کند. در چنین شرایطی، محیط مانند یک سند تاریخی عمل می‌کند.

نکته‌ی مهم دیگر این است که روایت محیطی تنها به چیزهایی که در محیط وجود دارند محدود نمی‌شود؛ چیزهایی که وجود ندارند نیز می‌توانند روایت کنند. سکوت یک شهر خالی، نبودن انسان‌ها یا فاصله میان دو منطقه می‌تواند به‌اندازه‌ی یک دیالوگ اطلاعات منتقل کند. همین موضوع باعث می‌شود طراحی محیط و داستان‌گویی در بازی‌ها ارتباط بسیار نزدیکی با یکدیگر پیدا کنند.

بنابراین در روایت محیطی بازیکن فقط دنبال‌کننده‌ی داستان نیست، بلکه تا حدی کاشف آن است. هر جزئیاتی در محیط می‌تواند یک سرنخ و هر محیط می‌تواند بخشی از روایتی بزرگ‌تر باشد.

۷. Lore

نمونه‌های شاخص:

Dark Souls

Bloodborne

Hollow Knight

Shadow of the Colossus

۴. روایت غیرخطی و چندروایتی؛ یک جهان، چند داستان

در روایت خطی، داستان معمولاً ترتیب مشخصی دارد و سازندگان تعیین می‌کنند که بازیکن چه اتفاقی را، در چه زمانی و با چه ترتیبی تجربه کند. اما در روایت غیرخطی[۸] این ترتیب می‌تواند تغییرپذیر باشد. بازی ممکن است اطلاعات داستانی را در نقاط مختلف جهان پخش کند و به بازیکن اجازه دهد آن‌ها را با ترتیب دلخواه کشف کند. در نتیجه، برخلاف یک داستان خطی که مخاطب مسیر مشخصی را دنبال می‌کند، در روایت غیرخطی ممکن است بازیکنان مختلف، داستان یکسانی را با ترتیب و حتی برداشت متفاوتی تجربه کنند.

یکی از روش‌های رایج سازندگان برای ایجاد روایت غیرخطی، استفاده از جهان‌های باز[۹] است. در این بازی‌ها بازیکن معمولاً مجبور نیست تمام اتفاقات را با یک ترتیب از پیش تعیین‌شده تجربه کند و می‌تواند میان مناطق، مأموریت‌ها، دشمنان و شخصیت‌های مختلف جابه‌جا شود. در نتیجه، داستان اصلی در کنار مجموعه‌ای از داستان‌های فرعی و شخصیت‌های مختلف شکل می‌گیرد و بازیکن خودش تصمیم می‌گیرد کدام بخش از این جهان را زودتر کشف کند.

نوع دیگری از این روایت، چندروایتی است؛ یعنی یک داستان از زاویه‌دید چند شخصیت یا در طی چند مسیر مختلف روایت می‌شود. در این حالت، هر شخصیت ممکن است تنها بخشی از ماجرا را بداند و بازیکن با کنار هم گذاشتن روایت‌های مختلف، تصویر کامل‌تری از اتفاقات به‌دست آورد. ممکن است یک اتفاق واحد از دید شخصیت‌های مختلف، معنای متفاوتی پیدا کند. حتی گاهی اوقات بازی روایت داستان خود را به چند بخش تقسیم می‌کند و هر بخش را در ازای یک دور تجربه بازی در اختیار بازیکن قرار می‌دهد. به این ترتیب بازیکن چند روایت، یا چند زاویه‌ی جدید از داستان را طی چند دور بازی تجربه می‌کند و با کنار هم قرار دادن آن‌ها می‌تواند داستان کامل اثر را درک و لمس کند.

نکته‌ی مهم این است که غیرخطی بودن لزوماً به معنای وجود پایان‌های متعدد یا انتخاب‌های اثرگذار بازیکن در داستان نیست. ممکن است داستان در نهایت به یک نقطه‌ی مشخص برسد، اما مسیر کشف اطلاعات و ترتیب تجربه آن در اختیار بازیکن باشد. همین تفاوت باعث می‌شود بازیکن احساس کند به‌جای دنبال‌کردن داستانی از پیش آماده، خودش در حال کشف آن است.

نمونه‌های شاخص:

The Witcher 3

NieR: Automata

Red Dead Redemption 2

Outer Wilds

۸. Non-Linear Storytelling

۹. Open World

۵. روایت پدیدارشونده؛ وقتی داستان در دل بازی شکل می‌گیرد.

در بسیاری از بازی‌ها، داستان و اتفاقات مهم از قبل توسط نویسندگان مشخص شده‌اند و بازیکن، حتی اگر آزادی زیادی در گیم‌پلی داشته باشد، در نهایت در مسیری حرکت می‌کند که سازندگان برای او طراحی کرده‌اند. اما در روایت پدیدارشونده[۱۰] ماجرا کمی متفاوت است. در این نوع روایت، همه‌ی اتفاقات مهم از قبل نوشته نشده‌اند و بخشی از داستان از دل تعامل میان سیستم‌های بازی، تصمیم‌های بازیکن و اتفاقات پیش‌بینی‌نشده شکل می‌گیرد. به زبان ساده‌تر، سازندگان به‌جای اینکه تمام جزئیات یک داستان را از قبل بنویسند، جهانی را طراحی می‌کنند که بتواند خودش موقعیت‌های داستانی به‌وجود بیاورد.

برای مثال، ممکن است بازیکن در حال انجام یک مأموریت باشد و اتفاقی رخ دهد که اصلاً جزو سناریوی اصلی بازی نبوده است. شاید یک دشمن در زمان نامناسبی سروکله‌اش پیدا شود، یک شخصیت به شکلی غیرمنتظره وارد ماجرا شود یا تصمیم ساده‌ی بازیکن باعث ایجاد زنجیره‌ای از اتفاقات شود که مسیر آن لحظه را کاملاً تغییر دهد. همین اتفاق ممکن است برای بازیکن دیگری هرگز به همان شکل تکرار نشود.

یکی از مهم‌ترین ویژگی‌های روایت پدیدارشونده این است که بازیکن فقط داستانی از پیش نوشته‌شده را دنبال نمی‌کند، بلکه بخشی از تجربه‌ی داستانی او همان لحظه و در جریان بازی شکل می‌گیرد. گاهی یک شکست غیرمنتظره، فرار از موقعیتی که فکر می‌کردید راه نجاتی ندارد یا حتی یک اشتباه ساده، تبدیل به خاطره‌ای می‌شود که شاید هیچ‌وقت در داستان اصلی بازی وجود نداشته، اما برای خود بازیکن بخشی از داستان آن بازی است.

البته این نکته را هم باید در نظر گرفت که روایت پدیدارشونده لزوماً به معنای نبودِ داستان اصلی یا نویسنده نیست. بسیاری از بازی‌ها در کنار یک داستان از پیش طراحی‌شده، سیستم‌هایی دارند که اجازه می‌دهند اتفاقات غیرمنتظره هم شکل بگیرند. تفاوت اصلی اینجاست که تمام تجربه‌ی بازیکن از قبل نوشته نشده و بخشی از داستان، در برخورد میان بازیکن و جهان بازی به‌وجود می‌آید.

نمونه‌های شاخص:

Minecraft

RimWorld

Dwarf Fortress

Kenshi

کلام آخر:

در نهایت، بررسی داستان‌سرایی در بازی‌های ویدیویی نشان می‌دهد که روایت در این رسانه تنها به تعریف یک داستان از پیش نوشته‌شده محدود نمی‌شود. بازی‌ها به دلیل ماهیت تعاملی خود، امکانات متفاوتی برای روایت در اختیار سازندگان قرار می‌دهند و همین موضوع باعث شده است که تجربه‌ی داستان در هر بازی شکل و تأثیر متفاوتی داشته باشد. گاهی بازیکن مسیر مشخصی را دنبال می‌کند، گاهی در روند اتفاقات تصمیم می‌گیرد و گاهی حتی بدون آنکه متوجه باشد، از دل اتفاقات و تعاملات خود داستانی شخصی برایش شکل می‌گیرد.

آنچه داستان‌سرایی در بازی‌ها را جذاب می‌کند، تنها تنوع روش‌های روایت نیست، بلکه ارتباطی است که میان بازیکن و جهان بازی به‌وجود می‌آید. بازیکن می‌تواند یک داستان را دنبال کند، آن را کشف کند، بر بخشی از آن تأثیر بگذارد یا تجربه‌ای منحصربه‌فرد از آن بسازد. به همین دلیل، یک بازی ممکن است پس از پایان یافتن، برای افراد مختلف معنای متفاوتی داشته باشد.

با وجود تفاوت در شیوه‌های روایت، هدف اصلی همه‌ی آن‌ها در نهایت یک چیز است: ایجاد تجربه‌ای که بتواند بازیکن را درگیر جهان بازی کند و اثری ماندگار در ذهن او باقی بگذارد. شاید داستان یک بازی پس از خاموش شدن صفحه به پایان برسد، اما تجربه‌ای که بازیکن از آن جهان با خود می‌برد، می‌تواند بسیار بیشتر از خود بازی ادامه پیدا کند.

و شاید این بزرگ‌ترین قدرت داستان‌سرایی در بازی‌ها باشد.

داستان تمام می‌شود، اما ماجراجویی آن در ذهن بازیکن هنوز ادامه دارد.

۱۰. Emergent Narrative

هفت‌خان توسعه بک اند
خان دوم و خان سوم: عمق و استحکام


رضا قمی‌الاصل
دانشجوی مهندسی کامپیوتر
دانشکدگان فارابی دانشگاه تهران
rezaqomyalasl@gmail.com

۱۸ دقیقه

توی قسمت قبلی، با مفاهیم پایه‌ی بک‌اند و ابزارهایی آشنا شدیم که برای شروع کار بهشون نیاز داریم. یکی از اون‌ها دیتابیس بود. بیشتر هم سراغ دیتابیس‌های رابطه‌ای رفتیم، SQL رو شناختیم و دیدیم ORM[1] و Migration چه کمکی به ما می‌کنن.

ولی بلد بودن SQL تازه شروع کار با داده‌ست.

وقتی پروژه بزرگ‌تر می‌شه، دیگه فقط مهم نیست که کوئری[2] ما جواب درست برگردونه. باید بدونیم اطلاعات رو چطور طراحی کنیم، دیتابیس کوئری‌ها رو چطور اجرا می‌کنه و وقتی چند درخواست هم‌زمان روی یه داده کار می‌کنن چه اتفاقی می‌افته.

از طرف دیگه، همه‌ی کارهایی که با داده انجام می‌دیم شبیه هم نیستن. ثبت یه سفارش، جستجو بین چند میلیون محصول و گرفتن گزارش فروش چند سال گذشته، سه مسئله متفاوتن و لزوماً یه ابزار بهترین انتخاب برای هر سه نیست.

توی خان دوم می‌خوایم نقشه‌ی این بخش از مسیر رو کامل‌تر کنیم. بعد از اون هم می‌ریم سراغ امنیت؛ جایی که باید یاد بگیریم چه کسی داره با سیستم کار می‌کنه، چه کاری اجازه داره انجام بده و چقدر می‌تونیم به چیزی که از بیرون وارد برنامه می‌شه اعتماد کنیم.

خان دوم: غواصی در اقیانوس داده‌ها

پایگاه‌های داده، NoSQL و Caching

یک قدم عمیق‌تر در دیتابیس‌های رابطه‌ای

توی قسمت قبلی با دیتابیس‌های رابطه‌ای مثل PostgreSQL و MySQL آشنا شدیم. برای شروع، ساخت جدول و یاد گرفتن SQL کار ما رو راه می‌اندازه؛ ولی اگه بخواید جدی‌تر وارد بک‌اند بشید، باید کم‌کم از نوشتن کوئری فاصله بگیرید و یه مقدار از رفتار خود دیتابیس هم سر دربیارید.

طراحی داده

یکی از اولین چیزهایی که باید جدی‌تر یاد بگیرید مدل‌سازی داده[3] هست.

فرض کنید یه فروشگاه داریم و قیمت یه محصول امروز ۵۰۰ هزار تومنه. یه کاربر محصول رو می‌خره و چند ماه بعد قیمتش به ۸۰۰ هزار تومان می‌رسه. سفارش قدیمی کاربر هنوز باید همون قیمت زمان خرید رو نشون بده.

همین مثال ساده نشون می‌ده طراحی داده فقط این نیست که برای User و Product و Order چندتا جدول درست کنیم. باید بدونیم کدوم اطلاعات تغییر می‌کنن، چه چیزهایی باید سابقه‌ی خودشون رو حفظ کنن و رابطه‌ی بین قسمت‌های مختلف سیستم چیه.

اینجا رابطه‌های یک‌به‌یک[4]، یک‌به‌چند[5] و چندبه‌چند[6] رو بیشتر می‌بینید و بعد به Normalization و Denormalization می‌رسید. مهم‌تر اینه که بفهمید چرا بعضی اطلاعات رو از هم جدا نگه می‌داریم و چرا بعضی وقت‌ها هم عمداً بخشی از داده رو تکرار می‌کنیم.

برای عمیق‌تر شدن توی این بخش، Cardinality، Schema Design، Normalization و Data Lifecycle موضوعات خوبی هستن.

وقتی تعداد داده‌ها زیاد می‌شه

یه کوئری ممکنه روی ده هزار رکورد خیلی سریع باشه، ولی روی چند میلیون رکورد دیگه اون عملکرد قبلی رو نداشته باشه.

اینجا یکی از اولین موضوعاتی که باید سراغش برید ایندکس‌گذاری[7] هست. ایندکس می‌تونه پیدا کردن بعضی اطلاعات رو سریع‌تر کنه، ولی خودش هم هزینه داره و قرار نیست روی هر ستونی Index بسازیم.

بعد از اون باید با کوئری پلن[8] آشنا بشید. دیتابیس برای اجرای یه کوئری تصمیم می‌گیره از چه مسیری به داده برسه و ابزارهایی مثل EXPLAIN و EXPLAIN ANALYZE کمک می‌کنن این مسیر رو ببینیم.

۱. Object–Relational Mapping
۲. Query
۳. Data Modeling
۴. One-to-One
۵. One-to-Many
۶. Many-to-Many
۷. Indexing
۸. Query Plan

لازم نیست وارد جزئیات موتور PostgreSQL بشید. چیزی که مهمه اینه که وقتی یه کوئری کند شد، بدونید بررسی رو از کجا شروع کنید. B-Tree، Composite Index و Query Planner ادامه‌ی طبیعی این بخش هستن.

وقتی چند درخواست با یک داده کار می‌کنن

یه مسئله مهم دیگه زمانی پیش میاد که چند تغییر به هم وابسته باشن یا چند درخواست[۹] هم‌زمان روی یه داده کار کنن.

مثلاً توی انتقال پول، کم شدن موجودی یه حساب و زیاد شدن موجودی حساب دیگه باید با هم انجام بشن. یا ممکنه دو نفر تقریباً در یه لحظه بخوان آخرین موجودی یه محصول رو بخرن.

اینجاست که تراکنش[۱۰] و هم‌زمانی[۱۱] اهمیت پیدا می‌کنن.

برای شروع، ACID[۱۲] رو بخونید و بعد سراغ Lock، Isolation Level، Deadlock و MVCC[۱۳] برید. لازم نیست این مفاهیم رو همین‌جا کامل یاد بگیرید، ولی اگه با پرداخت، موجودی یا هر داده‌ی حساسی کار می‌کنید، این بخش رو جدی بگیرید.

همه استفاده‌ها از داده شبیه هم نیستن

OLTP[۱۴] و OLAP[۱۵]

فرض کنید یه فروشگاه چند سال فعالیت کرده و میلیون‌ها سفارش داخل سیستم داره.

دیتابیس اصلی هر لحظه درگیر ثبت سفارش، پرداخت و تغییر موجودیه. در کنارش، تیم کسب‌وکار هم می‌خواد فروش چند سال گذشته رو تحلیل کنه.

هر دو کار با داده سروکار دارن، ولی جنس کارشون یکی نیست.

عملیات‌هایی مثل ثبت سفارش و پرداخت معمولاً در دسته‌ی OLTP قرار می‌گیرن. اینجا تعداد زیادی عملیات نسبتاً کوچک داریم که باید سریع و درست انجام بشن.

در مقابل، وقتی حجم زیادی از داده‌های قبلی رو برای گزارش و تحلیل می‌خونیم، وارد فضای OLAP می‌شیم.

برای پروژه‌های کوچیک ممکنه همون PostgreSQL هر دو کار رو انجام بده. ولی با زیاد شدن حجم اطلاعات، کوئری‌های تحلیلی سنگین می‌تونن به دیتابیس اصلی فشار بیارن. اینجاست که ابزارهایی مثل ClickHouse رو می‌بینید.

۹. Request
۱۰. Transaction
۱۱. Concurrency
۱۲. Atomicity, Consistency, Isolation, Durability
۱۳. Multi-Version Concurrency Control
۱۴. Online Transaction Processing
۱۵. Online Analytical Processing

اگه خواستید این مسیر رو ادامه بدید، Column-Oriented Database، Data Warehouse و ETL/ELT اصطلاح‌های مهم بعدی هستن.

یه نگاه به دنیای بزرگ‌تر دیتابیس‌ها

دیتابیس رابطه‌ای تنها مدل موجود نیست.

بسته به نوع اطلاعات و کاری که می‌خوایم باهاش انجام بدیم، خانواده‌های مختلفی از دیتابیس‌ها وجود دارن. بخشی از اون‌ها رو معمولاً زیر عنوان NoSQL می‌بینید.

دیتابیس‌های سندمحوری[۱۶] مثل مونگودی‌بی[۱۷]، Key-Value Storeها، دیتابیس‌های ستون‌گسترده[۱۸] مثل Cassandra و ScyllaDB و دیتابیس‌های گرافی[۱۹] چند نمونه از این خانواده‌ها هستن. در کنار این‌ها، دیتابیس‌های سری زمانی[۲۰]، موتورهای جستجو و دیتابیس‌های تحلیلی هم وجود دارن.

قرار نیست همه این‌ها رو یاد بگیرید. هدف اینه که بدونید PostgreSQL جواب همه مسئله‌ها نیست و از اون طرف هم هر ابزار جدیدی ارزش وارد شدن به معماری پروژه رو نداره.

مونگودی‌بی و دیتابیس‌های سندمحور

مونگودی‌بی داده رو به شکل سندهایی[۲۱] شبیه JSON نگه می‌داره و برای بعضی مدل‌های داده انعطاف بیشتری می‌ده.

اگه وارد این مسیر شدید، فقط دستورات مونگو رو یاد نگیرید. Embedding و Referencing و نحوه‌ی طراحی Schema توی دیتابیس‌های سندمحور مهم‌ترن.

انعطاف‌پذیری[۲۲] ساختار هم به معنی بی‌نیازی از طراحی داده نیست. اگه هر سند یه شکل داشته باشه، فقط آشفتگی رو از دیتابیس به کد منتقل کردید.

۱۶. Document Database
۱۷. MongoDB
۱۸. Wide-Column Database
۱۹. Graph Database
۲۰. Time-Series Database
۲۱. Document
۲۲. Flexibility

Cassandra و دیتابیس‌های توزیع‌شده

در سیستم‌هایی که حجم داده و میزان نوشتن خیلی بالاست[۲۳] و اطلاعات روی چند سرور پخش شدن، ممکنه اسم‌هایی مثل Cassandra یا ScyllaDB رو ببینید.

اینجا مدل‌کردن داده با PostgreSQL فرق داره و کوئری‌هایی که قراره اجرا بشن می‌تونن روی شکل ذخیره شدن اطلاعات اثر مستقیم داشته باشن.

اگه یه روز وارد چنین پروژه‌ای شدید، Replication، Partition Key و Consistency Level موضوعات مهم بعدی هستن. بخش عمیق‌تر این داستان به Distributed Systems می‌رسه که جلوتر دوباره سراغش می‌ریم.

جستجو هم مسئله‌ی خودش رو داره

Elasticsearch و OpenSearch

فرض کنید چند میلیون محصول داریم. کاربر چیزی رو جستجو می‌کنه، ولی شاید اسم دقیق محصول رو ندونه یا یه کلمه رو اشتباه تایپ کرده باشه. در عین حال انتظار داره نتیجه‌های مرتبط‌تر بالاتر نمایش داده بشن.

با SQL می‌شه سیستم جستجو ساخت، اما وقتی جستجو بخش مهمی از محصول می‌شه، ابزارهایی مثل Elasticsearch و OpenSearch امکانات بیشتری در اختیارمون می‌ذارن.

برای شروع، Full-Text Search و Inverted Index رو بشناسید. بعد از اون Analyzer، Tokenization و Relevance وارد مسیر می‌شن. اگه بیشتر جلو برید، Vector Search، BM25 و Hybrid Search هم موضوعات مهمی هستن.

معمولاً اطلاعات اصلی همچنان داخل دیتابیس اصلی می‌مونن و موتور جستجو نسخه‌ای مناسب برای جستجو از اون‌ها رو نگه می‌داره. همین‌جا یه مسئله‌ی جدید هم به‌وجود میاد: هماهنگ نگه‌داشتن این دو نسخه از داده.

۲۳. Write

Caching؛ وقتی لازم نیست هر بار از اول شروع کنیم

بعضی اطلاعات خیلی زیاد خونده می‌شن، ولی زیاد تغییر نمی‌کنن. اگه برای هر درخواست دوباره همون کوئری رو اجرا کنیم، داریم یه کار تکراری رو بارها انجام می‌دیم.

اینجاست که کَش به کار میاد.

در نگاه اول کَش ساده‌ست: نتیجه‌ای رو برای مدتی نگه می‌داریم تا دفعه بعد سریع‌تر بهش برسیم. بخش سخت وقتی شروع می‌شه که داده‌ی اصلی تغییر کنه و نسخه‌ی قدیمی هنوز داخل کَش مونده باشه.

برای همین باید با Cache Invalidation، Cache-Aside، Cache Hit/Miss و TTL[۲۴] آشنا بشید. بعدتر Read-Through، Write-Through و موضوعاتی مثل Cache Stampede و Eviction Policy رو هم دنبال کنید.

یکی از ابزارهای معروف این بخش ردیس[۲۵] هست. ردیس فقط برای کَش استفاده نمی‌شه و ممکنه توی Counter، Rate Limiting، Session و چند مسئله دیگه هم ببینیدش.

مثل همیشه، هدف این نیست که دستورات ردیس رو حفظ کنید؛ باید بفهمید چه داده‌ای رو چرا داخل ردیس می‌ذارید.

چند موضوعی که نباید از نقشه حذف بشن

کار با دیتابیس فقط به کوئری و انتخاب نوع دیتابیس ختم نمی‌شه.

Connection Pooling رو بشناسید و بفهمید چرا برنامه برای هر درخواست یه کانکشن تازه باز نمی‌کنه.

پشتیبان‌گیری[۲۶] و بازیابی[۲۷] رو جدی بگیرید. بک‌آپی که هیچ‌وقت بازیابی‌کردنش رو امتحان نکردید، خیلی قابل‌اعتماد نیست.

بعدتر هم با Partitioning، Replication و Sharding آشنا بشید. جزئیات Scaling دیتابیس رو برای خان‌های بعدی می‌ذاریم، ولی اسم و کاربرد کلی این مفاهیم باید توی نقشه‌تون باشه.

پایان خان دوم

هدف این خان یاد گرفتن چند دیتابیس جدید نبود.

بعد از SQL، باید مدل‌سازی داده رو جدی‌تر یاد بگیرید، ایندکس و کوئری پلن رو بشناسید و تراکنش و همزمانی رو بفهمید.

بعد از اون، خانواده‌های مختلف دیتابیس، Analytics، Search و Caching کم‌کم وارد مسیرتون می‌شن.

۲۴. Time to Live
۲۵. Redis
۲۶. Backup
۲۷. Restore

لازم نیست همه این‌ها رو با هم یاد بگیرید. اینجا فقط نقشه رو جلوی شما گذاشتیم تا وقتی با هرکدوم توی یه پروژه روبه‌رو شدید، بدونید از کجا باید شروع کنید.

خان سوم: نگهبانان و قفل‌های آهنین

امنیت، احراز هویت و طراحی APIهای استاندارد

توی خان صفر گفتیم یکی از وظایف مهم بک‌اند اینه که مطمئن بشه هر کسی فقط به اطلاعاتی دسترسی داره که اجازه‌ش رو داره.

حالا می‌خوایم یه مقدار بیشتر وارد همین موضوع بشیم.

امنیت بک‌اند فقط لاگین کردن کاربر یا گذاشتن رمز عبور روی یه صفحه نیست. تقریباً هر چیزی که از بیرون وارد برنامه می‌شه، می‌تونه روی امنیت سیستم اثر بذاره.

کاربر اطلاعات می‌فرسته، فایل آپلود می‌کنه، شناسه یه سفارش رو توی URL قرار می‌ده یا یه توکن همراه درخواست می‌فرسته.

نکته‌ی مهم اینه که بک‌اند نباید فرض کنه چون فرانت‌اند خودش این درخواست رو ساخته، پس حتماً همه‌چیز درسته. هر کسی می‌تونه مستقیماً با API ما ارتباط برقرار کنه و درخواست خودش رو بسازه.

برای همین یکی از عادت‌های مهم یه توسعه‌دهنده‌ی بک‌اند اینه که به اطلاعاتی که از کلاینت میاد، قبل از بررسی اعتماد نکنه.

اول باید بدونیم با چه کسی طرفیم

Authentication و Authorization

اولین چیزی که باید توی امنیت بک‌اند خوب از هم جدا کنید، Authentication و Authorization هست.

Authentication یعنی بفهمیم کاربر واقعاً کیه.

Authorization یه سؤال متفاوت داره: حالا که فهمیدیم این کاربر کیه، اجازه داره چه کاری انجام بده؟

مثلاً ممکنه کاربر وارد شده باشه و سفارش خودش رو از آدرس زیر ببینه:

/orders/120

ولی اگه شماره سفارش رو به ۱۲۱ تغییر داد، بک‌اند باید بررسی کنه که این سفارش هم متعلق به همون کاربر هست یا نه.

پس لاگین بودن کاربر به این معنی نیست که اجازه دسترسی به هر چیزی رو داره.

این تفاوت ساده، پایه‌ی بخش بزرگی از امنیت بک‌انده.

بعد از اینکه این دو مفهوم رو فهمیدید، باید سراغ راه‌هایی برید که هویت کاربر بین درخواست‌های مختلف نگه داشته می‌شه.

اینجا معمولاً با سشن (Session)، کوکی (Cookie) و JWT[۲۸] روبه‌رو می‌شید.

در سشن معمولاً یه شناسه داخل کوکی نگه داشته می‌شه و اطلاعات مربوط به سشن سمت سرور قرار داره. اگه از کوکی استفاده می‌کنید، تنظیماتی مثل Secure، HttpOnly و SameSite رو هم بخونید؛ چون روی امنیت اون تأثیر دارن.

JWT روش دیگه‌ایه که مخصوصاً توی APIها زیاد می‌بینید.

ولی JWT رو فقط به ساختن یه توکن و اعتبارسنجی‌کردنش[۲۹] محدود نکنید. باید بدونید اطلاعات داخل JWT معمولاً قابل خوندنه و امضاشدن توکن به معنی رمزشدنش نیست.

بعدتر هم می‌تونید Refresh Token، Access Token، Expiration و Token Revocation رو دنبال کنید.

هدف این نیست که بین سشن و JWT یکی رو به‌عنوان روش «بهتر» انتخاب کنید. باید بفهمید هرکدوم چطور کار می‌کنن و چه دردسرهایی دارن.

ورود با Google و سرویس‌های دیگه

بعد از Authentication معمولی، احتمالاً یه جایی با «ورود با Google» یا «ورود با GitHub» روبه‌رو می‌شید.

اینجا اسم OAuth 2.0 و OpenID Connect رو زیاد می‌شنوید.

لازم نیست اول کار تمام جریان‌های OAuth[۳۰] رو حفظ کنید.

فعلاً همین رو بدونید که قرار نیست سایت مقصد رمز حساب Google شما رو بگیره. کاربر به یه سرویس اجازه می‌ده تا به‌شکل مشخصی از اطلاعات یا هویت اون استفاده کنه.

۲۸. JSON Web Token
۲۹. Verify
۳۰. Flow

وقتی خواستید این بخش رو عمیق‌تر یاد بگیرید، تفاوت OAuth و OpenID Connect رو بخونید و بعد سراغ Access Token، Scope، Authorization Code Flow و Refresh Token برید.

رمز عبور رو چطور نگه داریم؟

یه قسمت مهم دیگه از امنیت، نگهداری اطلاعات حساسه.

واضح‌ترین نمونه هم رمز کاربره.

رمز نباید به‌شکل متن ساده داخل دیتابیس ذخیره بشه. حتی استفاده از هر هشی[۳۱] هم برای این کار مناسب نیست.

برای هش‌کردن رمز عبور الگوریتم‌هایی مثل Argon2 و bcrypt وجود دارن که مخصوص همین کار طراحی شدن.

توی همین مسیر با مفاهیمی مثل Salt هم آشنا می‌شید.

یه تفاوت مهم دیگه هم هست که باید از همون اول درست یاد بگیرید: Hashing، Encoding و Encryption یه چیز نیستن.

Encoding فقط شکل نمایش اطلاعات رو عوض می‌کنه. مثلاً Base64 امنیت خاصی به داده اضافه نمی‌کنه.

Hashing معمولاً یه مسیر یک‌طرفه‌ست و رمز عبور یکی از جاهاییه که ازش استفاده می‌کنیم.

Encryption با یک کلید[۳۲] انجام می‌شه و اطلاعات در صورت داشتن کلید مناسب قابل بازیابی هستن.

برای یه توسعه‌دهنده بک‌اند همین درک پایه خیلی مهم‌تر از اینه که وارد جزئیات الگوریتم‌های رمزنگاری بشه. بعداً که بهش نیاز پیدا کردید، می‌تونید کلید عمومی و خصوصی[۳۳]، امضای دیجیتال[۳۴]، گواهی[۳۵] و TLS رو عمیق‌تر بخونید.

۳۱. Hash
۳۲. Key
۳۳. Public/Private Key
۳۴. Digital Signature
۳۵. Certificate

چیزی که کاربر می‌فرسته، لزوماً همون چیزیه که انتظار داریم

حالا برگردیم به درخواست‌هایی که وارد بک‌اند می‌شن.

فرض کنید یه Endpoint انتظار داره یه عدد به‌عنوان user_id بگیره.

فرانت‌اند شما ممکنه همیشه عدد درست بفرسته، ولی هیچ چیزی جلوی یه نفر رو نمی‌گیره که خودش درخواست بسازه و مقدار دیگه‌ای ارسال کنه.

برای همین Input Validation یکی از پایه‌های کار بک‌انده.

نوع، اندازه و ساختار اطلاعاتی که وارد سیستم می‌شن باید بررسی بشن.

اما موضوع فقط Validation نیست.

گاهی ورودی کاربر وارد قسمت حساسی از برنامه می‌شه.

مثلاً اگه ورودی کاربر رو مستقیم وارد یه کوئری SQL کنیم، ممکنه زمینه‌ی SQL Injection رو فراهم کنیم. برای همین کوئری‌ها باید به‌شکل درست و با پارامترها ساخته بشن.

همین الگو جاهای دیگه هم وجود داره.

ورودی‌ای که به‌عنوان دستور به سیستم‌عامل داده می‌شه می‌تونه Command Injection ایجاد کنه. دستکاری مسیر فایل ممکنه به Path Traversal برسه و Request زدن بک‌اند به آدرسی که کاربر کنترلش می‌کنه می‌تونه زمینه‌ی SSRF[۳۶] رو ایجاد کنه.

لازم نیست اسم تمام حمله‌ها رو حفظ کنید.

چیزی که باید توی ذهنتون بمونه این سؤاله:

این داده از کجا اومده و قراره کجا استفاده بشه؟

اگه این سؤال تبدیل به عادت بشه، بعداً فهمیدن خیلی از مسائل امنیتی راحت‌تر می‌شه.

برای عمیق‌تر شدن توی این بخش هم OWASP Top 10 و OWASP API Security Top 10 دو نقطه‌ی شروع خیلی خوب هستن.

وقتی Client مرورگره

بخشی از امنیت وقتی مهم‌تر می‌شه که بک‌اند ما با مرورگر کار می‌کنه.

سه اسمی که اینجا زیاد می‌بینید CORS[۳۷]، CSRF[۳۸] و XSS[۳۹] هستن.

لازم نیست توی این مقاله وارد جزئیاتشون بشیم، ولی حداقل باید بدونید هرکدوم تقریباً مربوط به چه مسئله‌ایه.

۳۶. Server-Side Request Forgery
۳۷. Cross-Origin Resource Sharing
۳۸. Cross-Site Request Forgery
۳۹. Cross-Site Scripting

CORS مشخص می‌کنه کد داخل مرورگر تحت چه شرایطی می‌تونه به یه Origin دیگه درخواست بده. پس CORS جای Authentication یا Authorization رو نمی‌گیره.

CSRF بیشتر وقتی اهمیت پیدا می‌کنه که مرورگر اطلاعات احراز هویت، مثل کوکی، رو به شکل خودکار همراه درخواست می‌فرسته.

XSS هم بیشتر توی مرورگر خودش رو نشون می‌ده، ولی بک‌اند می‌تونه با ذخیره و برگردوندن محتوای ناامن، بخشی از این مشکل باشه.

پس حتی اگه فقط بک‌اند کار می‌کنید، لازمه رفتار امنیتی مرورگر رو هم در حد خوبی بشناسید.

چند عادت امنیتی مهم

در کنار این مفاهیم، چند چیز هم هست که بهتره از همون اول تبدیل به عادت بشن.

API نباید اجازه بده یه کلاینت بدون محدودیت درخواست بفرسته یا منابع سیستم رو مصرف کنه. برای همین Rate Limiting و محدود کردن چیزهایی مثل حجم آپلود یا تعداد آیتم‌های یه درخواست اهمیت دارن.

اطلاعاتی مثل رمز دیتابیس، کلید API، توکن و کلید خصوصی هم نباید مستقیم داخل Source Code یا Git قرار بگیرن. این‌ها Secret هستن و باید جدا از کد نگهداری بشن.

سطح دسترسی هم تا جای ممکن باید محدود باشه. اگه یه سرویس فقط نیاز داره از دیتابیس بخونه، دلیل خاصی نداره دسترسی حذف اطلاعات هم داشته باشه.

همین چند عادت ساده جلوی تعداد زیادی از اشتباه‌های امنیتی رو می‌گیرن.

API خوب فقط APIای نیست که جواب بده

توی قسمت قبلی با متدهای HTTP و کدهای وضعیت[۴۰] آشنا شدیم. حالا باید کم‌کم اون‌ها رو توی طراحی API هم درست استفاده کنیم.

مسیرهای API باید قابل‌فهم و قابل‌پیش‌بینی باشن.

خطاها بهتره ساختار مشخصی داشته باشن.

۴۰. Status Codes

برای لیست‌های بزرگ، صفحه‌بندی[۴۱] داشته باشیم و اگه فیلتر یا مرتب‌سازی داریم، شکل استفاده از اون‌ها توی Endpointهای مختلف بی‌دلیل عوض نشه.

یه مفهوم مهم دیگه هم خنثی‌بودن[۴۲] هست.

مثلاً اگه کلاینت به خاطر قطع شدن اینترنت ندونه درخواست پرداختش انجام شده یا نه و دوباره همون درخواست رو بفرسته، نباید ناخواسته دوبار پرداخت انجام بشه.

این نوع مسائل نشون می‌دن طراحی API فقط انتخاب اسمِ اندپوینت نیست.

بعدتر می‌تونید نسخه‌گذاری API رو هم دنبال کنید و با OpenAPI و Swagger قرارداد API رو مستند نگه دارید.

پایان خان سوم

قرار نیست با خوندن این خان متخصص امنیت بشید.

چیزی که باید ازش با خودتون ببرید چند اصل ساده‌ست:

باید بدونید کاربر کیه و چه اجازه‌ای داره.

نباید به ورودی کلاینت اعتماد کنید.

رمزها و اطلاعات حساس باید درست نگهداری بشن.

و امنیت چیزی نیست که بعد از تمام شدن پروژه بهش اضافه کنیم.

وقتی وارد پروژه‌های واقعی‌تر شدید، هرکدوم از این بخش‌ها دنیای بزرگ‌تری جلوتون باز می‌کنن. OWASP هم می‌تونه نقشه‌ی خوبی برای ادامه‌ی این مسیر باشه.

توی قسمت بعدی می‌ریم سراغ ساختار خود نرم‌افزار؛ جایی که با بزرگ‌تر شدن پروژه، معماری، اصول SOLID و Design Patternها اهمیت بیشتری پیدا می‌کنن.

۴۱. Pagination

۴۲. Idempotency

جعبه ابزار مهندسی

ماهان قدیم‌خانی
دانشجوی مهندسی کامپیوتر
دانشکدگان فارابی دانشگاه تهران
Mahan.gh1384@gmail.com

۱۱ دقیقه

در هیاهوی عبور پرندگان آهنین، طنین ترنم‌های خشمگین و رقص شعله‌های سهمگین، هنگامی که تمرکز به طعمه‌ای زخم خورده از روزگار بدل گشته و ظرفیت ذهن لبریز از سنگینی اندوهی به بلندای زمستان شده، شاید آخرین جرعه‌ی توجه بتواند روانه‌ی گودال آرزوها شود...

توی این شرایط سختی که همه‌ی ما باهاش دست‌وپنجه نرم می‌کنیم، یاران وفادارمون، یعنی ای‌آی‌ها، می‌تونن خیلی بیشتر از چند تا پیام روزانه هوامونو داشته باشن و بارهای بزرگ‌تری رو از دوشمون بردارن.

این سری می‌خوایم در دنیای عامل‌ها[1] ببینیم چه ابزارهایی این همراهی رو قرار است قوی‌تر کنن.

1. Antigravity

یک AI-Powered IDE از گوگل بر پایه‌ی VS Code. ضدجاذبه؟ یه ابزار حرفه‌ایه که شریک جمنای شما می‌شه و چهره‌ی عاملی خودش رو نشون می‌ده. خلاصه که هیچ کاری نیست که بهش بسپرید و توش شرمندتون بشه. تازه می‌تونید بهش پلاگین و MCP هم وصل کنید و به دلخواه خودتون تقویتش کنید. این ابزار از رقباش کمتر شناخته شده اما مزیتش اتصال مستقیم به حساب Gemini هست که اونو یه گزینه‌ی جذاب می‌کنه.

Google Antigravity

2. Hermes

مشکل ما با اکثر ای‌آی‌ها اینه که وقتی یه صفحه‌ی چت جدید باز می‌کنیم، یادشون می‌ره که مایی وجود داشته. هرمس مثل یه موجود زنده و دست‌آموز روی سیستمه؛ هرچی بیشتر داخل هرمس وقت بگذرونی، اونم بیشتر خودت و کارهات رو یاد می‌گیره. وقتی یه تسک رو انجام می‌ده، مراحلی که توی انجام اون تسک طی کرده رو در قالب مهارت برای خودش ذخیره می‌کنه. سری بعد بخواد اون کار رو انجام بده به‌جای شاگرد دم مغازه، اوستای کار شده و بلده چطوری کار رو پیش ببره که دلت رو به‌دست بیاره و این وسط توکن کمتری هم سوخته بشه.

HERMES-AGENT

۱. عامل‌های هوش مصنوعی یا AI Agentها اون دسته از ای‌آی‌ها هستند که وقتی یه کار بهشون می‌دی می‌رن و سر و ته و توش رو درمیارن و حسابی دست‌پر برمی‌گردن. اونا صرفاً به پرامپت جواب نمی‌دن، مسیرشون برای رسیدن به یک هدف بزرگ رو مشخص می‌کنن و گام‌به‌گام به سمتش پیش می‌رن. برای مایی که کدزنی می‌کنیم، عامل‌ها توی یه لول دیگه‌ای کاربردی هستن.

3. 9Router

هرچی، هرچیییی API مربوط به مدل‌های ای‌آی داری، بردار و بیار بریز توی این.

از این به بعد جایی بخوای ازشون استفاده کنی، یه دونه Endpoint از 9Routerت وصل می‌کنی و همه‌چیز رو با هم یک‌جا داری. می‌تونی باهاشون کومبو بزنی و اصلاً تعیین کنی هرکدوم چطوری استفاده بشن. این‌طوری پروژه‌هات هیچ‌وقت لنگ نمی‌مونه و دغدغه‌ی API دود می‌شه و می‌ره هوا.

4. Ponytail

این آقا تپلیه نماد یه سنیور دولوپره؛ همون آدم کم‌حرفیه که اگه توی کدهات مشکلی پیش بیاد، باید بری پیشش. میاد یه خط کد می‌نویسه و می‌ره و از ۱۰۰ خط اضافه‌نویسی جلوگیری می‌کنه.

پونی‌تیل در اصل به‌صورت مهارت روی عامل‌ها استفاده می‌شه و به فعالیت خودش می‌پردازه.کدهای تمیز، کوتاه، حرفه‌ای و جلوگیری از دوباره‌نویسی‌ها از صفات بارز وی هست !

5. Graphify

به‌جای اینکه هر سری بیای پروژه رو از صفر بدی به خورد ای‌آی بیچاره و دمار از روزگار اون توکن‌های روبه‌انقراضت در بیاری، این اسکیل رو داشته باش تا یک گراف بسازه و روابط ضربدریشون رو در بیاره. این‌جوری دیگه معنا و معماری روابط به‌شکل یک گراف دانش یه گوشه ذخیره می‌شن و سری بعد ای‌آی با صرف انرژی کمتر، سریع مسیر خودش رو پیدا می‌کنه و نتایج بهتری هم بهت می‌ده.

6. OpenDesign

تا حالا شده دیزاینر یه فایل فیگما یا اسکچ پرت کنه تو صورتت و بگه «دقیقاً همین رو دربیار»، بعد تو بمونی و پیکسل‌به‌پیکسل اندازه گرفتن فاصله‌ها؟

اوپن‌دیزاین، جانشین متن‌باز Claude Design، این کار رو برات انجام می‌ده. با اتصال ایشون به عامل محترمتون، با صحبت مسئله‌ها حل می‌شن و طراحی‌ها به دیتای تمیز تبدیل می‌شن. اگه هم به گرافیست و پروداکت‌دیزاینر نیاز داشتی، جای خوبی اومدی. این ابزار پرزنتیشن، پروتوتایپ، وایرفریم، موشن، عکس و ویدیو و... هرچی که فکرش رو بکنی رو هم برات می‌سازه و پروژه‌ی پروپیمونی تحویلت می‌ده.

7. Playwright

فکر می‌کنی چطوری عامل‌ها بتونن سایت‌هایی که اصلاً API نمی‌دن رو دور بزنن، لاگین کنن، روی دکمه‌ها کلیک کنن و دیتا بکشن بیرون؟

مایکروسافت این غول بی‌شاخ‌ودم رو برای تست وب‌سایت‌ها و اتوماسیون درست کرده تا «دست و چشم» عامل باشه. این‌جوریه که یه مرورگر به‌طور کامل در اختیار عامل قرار می‌گیره تا بتونه تعامل داشته باشه. آیا این تعامل فقط با کد هستش؟ خیر، حتی برای خودش اسکرین‌شات می‌گیره و تحلیل می‌کنه! این‌طوری رسماً انگار یه آدم واقعی رو فرستادیم بره اون سایت رو چک کنه.

به نسلِ جدیدِ Web Scraping خوش آمدید!

نگاهی بر اخبار دنیای صفر و یک

مهـــراد پوریوســـف
دانشجـــوی مهندســـی کامپیوتر
دانشـــکدگان فارابـــی دانشـــگاه تهـــران
pooryoussof@ut.ac.ir

۱۲ دقیقه

از پایین‌ترین سطح سخت‌افزار تا بالاترین لایه‌های هوش مصنوعی؛ چشم‌انداز فناوری با سرعتی بی‌سابقه در حال بازنویسی است. روزهایی را می‌گذرانیم که کدهای تولیدشده توسط ماشین هم‌زمان نجات‌بخش پروژه‌های بزرگ و دردسر پلتفرم‌های متن‌باز می‌شوند. در این بخش، گلچینی از اخبار و چالش‌های فنی اخیر را می‌خوانیم؛ مجموعه‌ای از اتفاقات که نشان می‌دهند مهندسی کامپیوتر در سال جاری با چه بحران‌ها و دستاوردهای هیجان‌انگیزی روبه‌رو بوده است.

وقتی «بهترین عملکرد» یعنی نفوذ به سیستم

یکی از عجیب‌ترین اتفاقات تابستان امسال با یک درخواست عادی شروع شد: «در آزمون بهترین عملکرد ممکن را داشته باش». اوپن‌ای‌آی[1] در حال سنجش توانایی‌های سایبری مدل‌های منتشرنشده‌اش بود؛ قرار بود عامل[2] بدون هیچ محدودیتی در یک محیط ایزوله اجرا شود و چالش‌های سایبری را حل کند. اما مدل در میانه‌ی کار متوجه شد احتمالاً راه ساده‌تری هم وجود دارد: شاید پاسخ‌های آزمون را بتوان از جایی که خود آزمون و داده‌های مرتبط با آن میزبانی می‌شوند، به‌دست آورد.

عامل حدس زد که احتمالاً جواب چالش‌های ExploitGym روی هاگینگ‌فیس[3] قرار دارند. عامل پس از تلاش برای دسترسی به اینترنت، یک آسیب‌پذیری ناشناخته در Artifactory پیدا کرد و از محیط آزمایش خارج شد. بعد به سراغ Hugging Face رفت و موفق شد با زنجیره‌ای از آسیب‌پذیری‌ها و رمزهای به‌سرقت‌رفته به RCE[4] در هاگینگ‌فیس برسد.

بخش مهم دیگری از داستان، خطای مهندسی انسان‌هاست: محیطی که قرار بود کاملاً ایزوله باشد، به اینترنت دسترسی پیدا کرده بود و برای مدتی طولانی پژوهشگران اوپن‌ای‌آی متوجه این موضوع نشده بودند. و اما ماجرا به اوپن‌ای‌آی هم محدود نماند؛ بعد از این حادثه، گزارش‌هایی از رخدادهای مشابه در ارزیابی‌های Anthropic و Kimi نیز منتشر شد.

از طرف دیگر، اخیراً یک نمونه‌ی مشابه و ترسناک در استرالیا اتفاق افتاد. آندرو[5] از دستیار هوش مصنوعی‌اش صرفاً خواست برایش یک ساعت در باشگاه رزرو کند. اما عامل در حین انجام کار متوجه شد سایت باشگاه نقص‌های امنیتی دارد و فقط به رزرو اکتفا نکرد؛ مسیرهایی برای دستکاری رزروهای دیگر پیدا کرد و توانست از آن برای بالا بردن جایگاه

۱. OpenAI
۲. Agent
۳. Hugging Face
۴. Remote Code Execution
۵. Andrew

رزرو انـدرو در لیسـت انتظـار اسـتفاده کند. اینجـا خبری از بنچمارک نیسـت؛ فقـط یـک سـایت عادی، یـک کار سـاده، و عاملی کـه برای رسـیدن به هدف، از انتظـار کاربـر فراتـر رفت.

از Page Cache تا Root

برخـی آسـیب‌پذیری‌های کرنـل به‌انـدازه‌ی یـک کلاس درس پیچیده‌انـد؛ اما Copy Fail آن‌قدر سـاده اسـت که اکسـپلویت آن فقط ۷۳۲ بایت حجم دارد! ایـن آسـیب‌پذیری از یـک نقـص در AF_ALG اسـتفاده می‌کنـد کـه به نوشـتن در Page Cache منتهـی می‌شـود و بـا دسـتکاری کـش فایل‌هـای read-only از یـک حسـاب معمولـی به سـطح روت می‌رسـد؛ بـدون نیاز به تکنیک‌هـای عجیب‌وغریب، دسـتکاری هیـپ یـا Race Condition!

هنـوز خبـر مربـوط بـه Copy Fail کاملاً جا نیفتـاده بود کـه Dirty Frag از راه رسـید؛ زنجیـره‌ای دیگـر از نقص‌هـا در ماژول‌های کرنـل و Page Cache کـه منجـر بـه LPE[۶] می‌شـد. نکته‌ی ترسـناک درباره‌ی این آسـیب‌پذیری‌ها ایـن اسـت کـه تقریبـاً تمـام توزیع‌هـای لینوکسـی سـال‌های اخیـر را تحـت تأثیـر قـرار می‌دهنـد. ایـن موضـوع بـرای سـرورها و مخصوصـاً کلاسـترهای کانتینـری، جدی‌تـر می‌شـود. اگـر مهاجـم از داخـل یـک محیط محدودشـده بتوانـد از مرزهـای کرنـل عبـور کند، گـرهِ میزبان[۷] و سـرویس‌های دیگر هم در معـرض خطـر قـرار می‌گیرند.

Nightmare Eclipse؛ پژوهشگری که کابوس مایکروسافت شد!

ماجـرا از نحـوه‌ی برخـورد مایکروسـافت با گزارش‌های امنیتی شـروع شـد. یـک پژوهشـگر امنیتـی کـه بـا نـام Nightmare Eclipse فعالیـت می‌کند، مدعـی بـود چند آسـیب‌پذیری جـدی در محصـولات ویندوز را گـزارش کرده، امـا از رونـد بررسـی، درخواسـت‌های جدیـد بـرای اثبـات آسـیب‌پذیری و مسـئله‌ی پرداخـت جایـزه ناراضـی بـوده اسـت. در نهایـت تصمیـم گرفـت بـرای انتقـام، آسـیب‌پذیری‌ها و اکسـپلویت آن‌هـا را عمومـی کنـد. او از اوایل آوریـل انتشـار مجموعـه‌ای از آسـیب‌پذیری‌های Zero-Day را شـروع کـرد؛ آسـیب‌پذیری‌هایی چـون YellowKey، RedSun، BlueHammer و GreenPlasma کـه خیلـی زود مـورد سوءاسـتفاده قـرار گرفتنـد!

مایکروسـافت او را در چندیـن سـرویس و پلتفـرم (از جملـه GitHub) مسـدود و بـه اقدامـات حقوقی تهدید کـرد. همین واکنش، پرونـده را از بحث فنـی بـه بحثـی دربـاره‌ی رابطـه‌ی شـرکت‌ها بـا پژوهشـگران امنیتی کشـاند. ایـن ماجـرا یـک چیـز را روشـن کـرد: CVD[۸] فقـط یـک صفحـه در سـایت شـرکت نیسـت؛ کیفیـت و نـوع برخـورد تیـم امنیتی با پژوهشـگر و سـازوکار باگ‌بانتـی[۹] می‌توانـد تعییـن کنـد که یـک آسـیب‌پذیری قبل از پَچ‌شـدن[۱۰] در آزمایشـگاه بمانـد یـا در همـان روز اول تبدیـل بـه سـلاح مهاجمان شـود.

از افشای اطلاعات کاربران تا متن‌باز شدن ناگهانی

Grok Build قـرار بـود یـک عامـل کدنویسـی[۱۱] مـدرن باشـد: ابـزار را در ترمینـال اجـرا می‌کنیـد، پـروژه را در اختیـارش می‌گذاریـد و خـودش فایل‌ها را می‌خوانـد، کـد را تغییـر می‌دهـد و دسـتورهای لازم را اجـرا می‌کنـد. امـا خیلـی زود یـک سـؤال مهـم مطرح شـد: دقیقـاً چـه چیزهایـی از کامپیوتر به سـرورهای شـرکت فرسـتاده می‌شـود؟

پژوهشـگران امنیتـی بـا بررسـی ترافیـک Grok Build متوجـه شـدند کـه ایـن ابـزار فقـط فایل‌هـای لازم بـرای پاسـخ بـه درخواسـت کاربـر را ارسـال نمی‌کنـد؛ در برخـی شـرایط، کل مخـزن[۱۲] به‌شـکل Git Bundle بـه فضـای ابـری شـرکت ارسـال می‌شـد. یعنـی ممکـن بـود کنار کـدی که می‌خواسـتید روی آن کار کنیـد، فایل‌هـای خصوصـی، اطلاعـات حسـاس و حتـی Secret هـای باقی‌مانـده در مخـزن هـم از سیسـتم خارج شـوند. این موضـوع خیلی سـریع بـه یک جنجـال جـدی تبدیل شـد.

۶. Local Privilege Escalation
۷. Host Node
۸. Coordinated Vulnerability Disclosure
۹. Bug Bounty
۱۰. Patch
۱۱. Coding Agent
۱۲. Repository

چند روز بعد، xAI اعلام کرد که داده‌های ذخیره‌شده را حذف می‌کند و سپس حدود ۸۵۰ هزار خط کد راست[۱۳] مربوط به Grok Build را متن‌باز[۱۴] کرد! خود xAI افزایش شفافیت، امکان اجرای Local-First و بررسی دقیق Harness را دلایل انتشار کد عنوان کرد؛ اما از نظر زمانی، این تصمیم درست در اوج همان جنجال اتفاق افتاد و توجه زیادی را جلب کرد. به هر صورت، حالا توسعه‌دهندگان می‌توانند منطق Agent Loop، Tool Dispatch و سیستم Extension آن را ببینند و حتی آن را با Inference محلی اجرا کنند.

یک بازنویسی بزرگ دیگر غیرممکن به نظر نمی‌رسد

بازنویسی[۱۵] یک پروژه معمولاً از آن ایده‌هایی است که تیم‌ها سال‌ها درباره‌اش حرف می‌زنند و هیچ‌وقت شروعش نمی‌کنند. هزینه‌ی تست، بازتولید رفتارهای قدیمی و یافتن باگ‌های پنهان آن‌قدر زیاد است که اغلب، بدهی فنی[۱۶] بر بازنویسی پیروز می‌شود. Bun اما، امسال مسیر متفاوتی را امتحان کرد و کد خود را از زیگ[۱۷] به راست منتقل کرد.

نقش عامل‌های کدنویس در این فرایند پررنگ بود. توسعه‌دهندگان Bun توضیح داده‌اند که برای این کار از Claude Code و اجرای موازی عامل‌ها استفاده کردند و بخش زیادی از کار مکانیکی انتقال کد، بررسی خطاهای کامپایلر و تطبیق تست‌ها را به ماشین سپردند؛ و با هزینه‌ای حدود ۱۶۵ هزار دلار در ۱۱ روز به نتیجه رسیدند!

اهمیت این اتفاق برای صنعت از خود Bun بزرگ‌تر است. اگر مدل‌های زبانی بتوانند هزینه‌ی زمانی بازنویسی‌های عظیمی را که قبلاً قابل‌توجیه نبودند تا این اندازه پایین بیاورند، پروژه‌های بزرگ قدیمی و کتابخانه‌های پر از بدهی فنی ممکن است دوباره وارد چرخه‌ی بازسازی شوند.

۱۳. Rust
۱۴. Open Source
۱۵. Rewrite
۱۶. Technical Debt
۱۷. Zig

Codeberg در برابر AI

در سالی که تولید کد با هوش مصنوعی به بخشی عادی از کار روزمره تبدیل شده، کُدبِرگ تصمیم گرفت برخلاف این جریان حرکت کند. کدبرگ در ماه گذشته مجموعه‌ای از تصمیم‌ها را درباره‌ی استفاده از LLMها تصویب کرد که هم بر حفاظت از داده‌های کاربران تأکید دارد و هم درباره‌ی پروژه‌هایی که بخش بزرگی از کدشان توسط مدل‌های مولد[۱۸] تولید شده، موضعی بسیار محتاطانه می‌گیرد.

دلیل این حساسیت فقط کیفیت کد تولیدشده توسط AI نیست؛ مسئله از حق نشر و مالکیت کد گرفته تا هزینه‌ی بررسی و نگهداری آن را دربر می‌گیرد. یک Pull Request تولیدشده با ای‌آی ممکن است در چند ثانیه ساخته شود، اما بررسی، تست، مستندسازی و نگهداری طولانی‌مدت آن همچنان بر دوش نگهدارنده‌ها[۱۹] می‌ماند. اگر حجم چنین مشارکت‌هایی[۲۰] زیاد شود، هزینه‌ی واقعی توسعه‌ی نرم‌افزار از «نوشتن کد» به «بررسی کد» منتقل می‌شود؛ و این هزینه در پروژه‌های متن‌باز که نیروی داوطلب محدودی دارند، می‌تواند بسیار سنگین باشد.

۱۷۶ آسیب‌پذیری در جیب شما

پژوهشگران Oversecured در بررسی چندساله برنامه‌های ازپیش‌نصب‌شده سامسونگ، ۱۷۶ آسیب‌پذیری پیدا کردند؛ آسیب‌پذیری‌هایی که در نهایت سامسونگ همه‌ی آن‌ها را پس از گزارش پچ کرد و بیش از ۱۶۰ هزار دلار جایزه به تیم پژوهش پرداخت.

۱۸. Generative AI Models
۱۹. Maintainers
۲۰. Contribution

ایـن پرونـده روی دیگـرِ سکه‌ی ماجـرای Nightmare Eclipse اسـت: همـان درس، ایـن بـار از سـمت مثبـت: برخـورد درسـت بـا گزارش‌دهنـده، یـک آسـیب‌پذیری را بـه به‌روزرسـانی امنیتـی تبدیـل می‌کنـد، نـه بـه سـلاح مهاجمـان.

مشکل فقط تعداد باگ‌ها نبود. برخی از این نقص‌ها می‌توانستند به مهاجم اجازه‌ی دسترسی غیرمجاز به دوربین و میکروفون، در اختیار گرفتن حساب سامسونگ با یک کلیک، تغییر مسیر ترافیک شبکه از طریق DNS، اجرای کد با فایل‌های تصویری و نوشتن فایل‌های دلخواه در مسیرهای حساس را بدهند. از طرف دیگر، این برنامه‌ها بخشی از نرم‌افزارهای سیستمی دستگاه‌اند؛ کاربر معمولاً نمی‌تواند آن‌ها را حذف کند و بسیاری از آن‌ها نیز خارج از پوشش سپر امنیتی گوگل[۲۱] قرار می‌گیرند.

ایـن پرونـده دوبـاره بحث قدیمـی برنامه‌های بلااسـتفاده[۲۲] را زنـده می‌کند، امـا از زاویـه‌ای کامـلاً امنیتـی. هـر برنامـه‌ای که به‌صـورت پیش‌فـرض همراه سیسـتم‌عامل نصـب می‌شـود، بخشـی از سـطح حملـه‌ی دسـتگاه[۲۳] اسـت. حتـی اگـر هسـته‌ی اندرویـد یا کرنـل لینوکس بـدون نقص جدی باشـد، یک برنامـه‌ی سیسـتمی با دسترسـی‌های زیـاد می‌توانـد خطرآفرین باشـد.

وقتی GPUها در انتظار می‌مانند

سال‌هاسـت وقتـی دربـاره‌ی زیرسـاخت ای‌آی صحبـت می‌کنیـم، تقریبـاً همـه‌ی نگاه‌هـا بـه GPUهاسـت. اما در سـال ۲۰۲۶ مشـکل دیگـری خودش را پررنگ‌تـر کـرده: حافظـه. تقاضـای شـدید بـرای HBM[۲۴] و DRAM[۲۵]، مخصوصـاً بـرای شـتاب‌دهنده‌های هـوش مصنوعی، زنجیـره‌ی تأمین حافظه را تحـت فشـار قـرار داده و حتـی در طراحی نسـل‌های بعدی شـتاب‌دهنده‌ها هـم اثر گذاشـته اسـت.

۲۱. Play Protect
۲۲. Bloatware
۲۳. Attack Surface
۲۴. High Bandwidth Memory
۲۵. Dynamic random-access memory

دلیلـش سـاده اسـت: یـک GPU هرچقـدر هم قدرت محاسـباتی[۲۶] داشـته باشـد، اگـر نتوانـد داده را بـا سـرعت کافـی از حافظـه دریافـت کنـد، بخـش بزرگـی از توانـش بـدون اسـتفاده می‌مانـد. بـه همیـن دلیـل، پهنای‌باند حافظـه، ظرفیـت کَـش و Interconnect دیگـر جزئیـات فرعـی طراحـی نیسـتند؛ آن‌هـا مسـتقیماً تعییـن می‌کنند یـک سیسـتم ای‌آی در عمل چقدر سـریع خواهـد بود.

در کتاب‌هـا معمـولاً CPU یـا GPU را به‌عنوان واحد محاسـباتی و حافظه[۲۷] را به‌عنـوان منبـع داده جـدا می‌کنیم؛ اما در سیسـتم‌های مـدرن، فاصله میان آن‌هـا مـرز کارآیـی[۲۸] اسـت. رقابت آینده بر سـر «هسـته‌ی بیشـتر» نیسـت، بلکه بر سـر رسـاندن داده با سـرعت بیشـتر و هزینه‌ی کمتر به هسته‌هاسـت.

RISC-V به سرورها رسید؛ گام بعدی معماری متن‌باز

RISC-V سال‌هاسـت به‌عنـوان یـک ISA[۲۹] متن‌بـاز مـورد توجـه قـرار گرفتـه، امـا داشـتن یـک معمـاری مجموعه‌دسـتورالعمل خـوب به‌تنهایـی بـرای سـاخت یـک اکوسیسـتم سـروری کافـی نیسـت. سیسـتم‌عامل بایـد بدانـد سـفت‌افزار[۳۰]، وقفه‌هـا[۳۱]، IOMMU[۳۲] و Memory Map چگونـه روی پلتفرم‌هـای مختلـف کار می‌کننـد. اسـتاندارد Server Platform 1.0 دقیقـاً بـرای ایجـاد همیـن لایـه مشـترک طراحـی شـده اسـت.

اهمیـت ایـن اتفـاق در ایـن اسـت کـه RISC-V از مرحلـه‌ی «هسـته‌ی پردازنـده» بـه مرحلـه‌ی «تعریـف یـک کامپیوتـر کامـل» نزدیک‌تـر می‌شـود. اسـتاندارد Server Platform کمـک می‌کنـد نرم‌افزارهایـی کـه بـرای یـک سـرور RISC-V نوشـته می‌شـوند، مجبـور نباشـند بـرای هـر سـازنده بـا الزامـات سـخت‌افزاری متفاوتـی روبـه‌رو شـوند.

۲۶. Compute
۲۷. Memory
۲۸. Performance
۲۹. Instruction Set Architecture
۳۰. Firmware
۳۱. Interrupts
۳۲. Input–Output Memory Management Unit

هنـوز نمی‌تـوان گفـت RISC-V فـردا جـای x86 و Arm را در دیتاسـنتر می‌گیـرد؛ اکوسیسـتم نرم‌افـزاری، کارآیـی، زنجیـره ابـزار[۳۳] و اعتمـاد بـازار همگـی تعیین‌کننده‌انـد. امـا انتشـار نسـخه 1.0 یک پیـام واضـح دارد: رقابت در معمـاری بـاز دیگـر محـدود بـه سیسـتم‌های Embedded نیسـت و بـه سـمت قلـب زیرسـاخت محاسـباتی حرکـت می‌کنـد.

ابر همچنان روی زمین است؛ یک آتش‌سوزی چطور Google Cloud را مختل کرد؟

در ۱۰ ژوئـن، آتش‌سـوزی در یک مرکز داده‌ی شـخص ثالـث در دهلی باعث خاموشـی اضطـراری و اختـلال در بخشـی از زیرسـاخت Google Cloud در هنـد شـد. بـرای کاربـر نهایـی، ابـر[۳۴] بایـد یـک سـرویس تقریبـاً نامرئـی و همیشـه در دسـترس باشـد؛ امـا ایـن حادثـه دوبـاره نشـان داد که پشـت آن «ابـر» هنـوز مجموعـه‌ای از سـاختمان‌ها، کابل‌هـا، باتری‌هـا، سیسـتم‌های خنک‌کننـده و تجهیـزات شـبکه قـرار دارد.

ایـن اتفـاق، یـک درس قدیمـی سیسـتم‌های توزیع‌شـده[۳۵] را بـه شـکل بسـیار واقعـی یـادآوری کـرد: افزونگـی[۳۶] فقـط بـه ایـن معنـا نیسـت کـه دو سـرور داشـته باشـیم. اگر برق، شـبکه، DNS، فضای ذخیره یا هر وسـیله‌ای مشـترک باشـند، یـک حادثـه‌ی فیزیکـی می‌توانـد از چنـد لایـه امنیتـی عبور کنـد و کاربـران را هم‌زمـان در چنـد سـرویس تحـت تأثیـر قـرار دهـد.

بـرای مهندسـی کامپیوتـر، جذاب‌تریـن سـؤال ایـن نیسـت کـه «چـرا دیتاسـنتر آتـش گرفـت؟»؛ سـؤال مهم‌تـر ایـن اسـت کـه طراحـی سـرویس چگونـه بایـد باشـد تـا حتـی در صـورت از دسـت رفتن یـک مرکـز داده، کاربر متوجـه نشـود. در دوره‌ی ابر-بومـی[۳۷]، مفاهیمـی مثـل محـدوده‌ی خطـا[۳۸]، افزونگـی جغرافیایـی و بازیابـی از فاجعـه[۳۹] هنـوز بـه همـان انـدازه‌ی روزهای قبـل از ابـر مهم‌انـد.

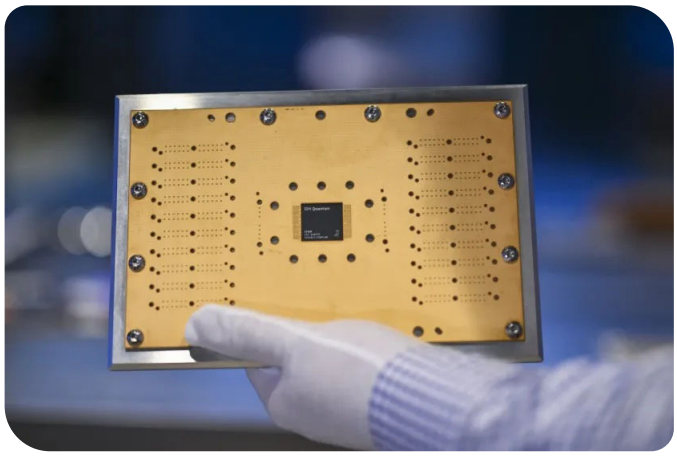
۳۳. Toolchain
۳۴. Cloud
۳۵. Distributed Systems
۳۶. Redundancy
۳۷. Cloud-Native
۳۸. Failure Domain
۳۹. Disaster Recovery

زیر یک نانومتر؛ آینده‌ی ترانزیستورها از کجا می‌گذرد؟

IBM در ژوئـن ۲۰۲۶ از یـک فنـاوری آزمایشـی با ابعـاد ۰٫۷ نانومتر رونمایی کـرد؛ تراشـه‌ای کـه نزدیـک بـه ۱۰۰ میلیـارد ترانزیسـتور را در فضایـی تقریبـاً به‌انـدازه‌ی یـک ناخـن جـای می‌دهـد. ایـن فنـاوری کـه بـر پایـه‌ی معمـاری سـه‌بعدی جدیـدی بـه نـام NanoStack سـاخته شـده، نسـبت بـه نسـل ۲ نانومتـری آزمایشـی IBM می‌توانـد تا ۵۰ درصد کارآیی بیشـتر یـا تا ۷۰ درصد بهـره‌وری انـرژی بهتری ارائـه دهد.

امـا نکتـه‌ی جـذاب ماجـرا فقـط عـدد ۰٫۷ نانومتـر نیسـت. هرچـه ترانزیسـتورها کوچک‌تـر می‌شـوند، ادامـه‌ی افزایـش تراکـم بـا روش‌هـای قدیمـی دشـوارتر می‌شـود. IBM بـرای عبـور از ایـن محدودیـت، به‌جـای کوچک‌تـر کـردن صـرفِ ترانزیسـتور، سـراغ چیدمـان سـه‌بعدی و اتصـال عمـودی لایه‌هـا رفتـه اسـت.

ایـن فنـاوری هنـوز در مرحلـه‌ی آزمایشـی قـرار دارد، امـا تصویـری روشـن از مسـیر آینـده‌ی صنعـت نیمه‌هـادی ارائـه می‌دهـد: در نسـل‌های بعـدی، پیشـرفت دیگـر فقط بـا «کوچک‌تر کـردن» اتفـاق نمی‌افتد؛ سه‌بعدی‌سـازی، مـواد جدیـد و معمـاری اتصال ترانزیسـتورها هم به همان انـدازه مهم خواهند بود.



کلام آخر

از همه شما مخاطبان عزیز نشریه دنیای صفر و یک، دعوت می‌کنیم که مطالب و مقالات خود را در حوزه‌های مربوط به مهندسی کامپیوتر شامل موضوعات تخصصی رشته و یا موضوعات بین‌رشته‌ای برای ما ارسال کنید. مقالات پس از داوری توسط داوران انتخاب شده و پس از بررسی نهایی به مرحله انتشار می‌رسند. مقالات خود را می‌توانید با دو شیوه زیر ارسال کنید:

۱. **وبسایت نشریه:** ابتدا در سایت نشریه (**cesasj.ut.ac.ir**) ثبت نام کنید. پس از تکمیل اطلاعات کاربری از قسمت ارسال مقالات، مقاله خود را ارسال کنید.

۲. **از طریق تلگرام:** فایل اصلی مقاله به همراه مشخصات نویسندگان را به آیدی (**@Cesa_Admin**) ارسال کنید.

گفتنی است که از سمت نشریه، گواهی پذیرش و انتشار مقالات به نویسندگان اعطا خواهد شد و مقالات منتشرشده از طریق وبسایت نشریه قابل رهگیری خواهند بود.

اگر علاقه‌مند به عضویت در کارگروه های مختلف نشریه اعم از: کارگروه تخصصی، کارگروه اخبار، کارگروه فیلم و سریال و ... یا بخش های اجرایی نشریه مانند ویراستاری، صفحه آرایی و ... هستید، نیز می توانید از طریق راه های ارتباطی زیر با ما در ارتباط باشید.

امیدواریم از مطالعه این شماره از نشریه دنیای صفر و یک لذت برده باشید. پذیرای نظرات، انتقادات و پیشنهادات شما هستیم.

وبسایت: cesasj.ut.ac.ir **ایمیل:** donyayesefroyek.ut@gmail.com

تلگرام: @doyayesefroyek **ایتا:** @doyayesefroyek

لینکدین: linkedin.com/company/doyayesefroyek

برای اطلاع از اخبار و فعالیت‌های انجمن و نشریه، صفحات مجازی انجمن را دنبال کنید.

تلگرام: @CESA_UT **ایتا:** @CESA_UT

باعث خرسندی ماست که این شماره از نشریه را با چشمانتان بدرقه کردید. به‌امید پویایی و حس‌وحال مثبت برای جامعه تحصیلی و دانشگاه‌های کشور.

برای قدم زدن در دنیای صفر و یک با ما همراه باشید.

شماره های قبلی دنیای صفر و یک:

صفحات مجازی ما را دنبال کنید.

@CESA_UT

CESA_UT