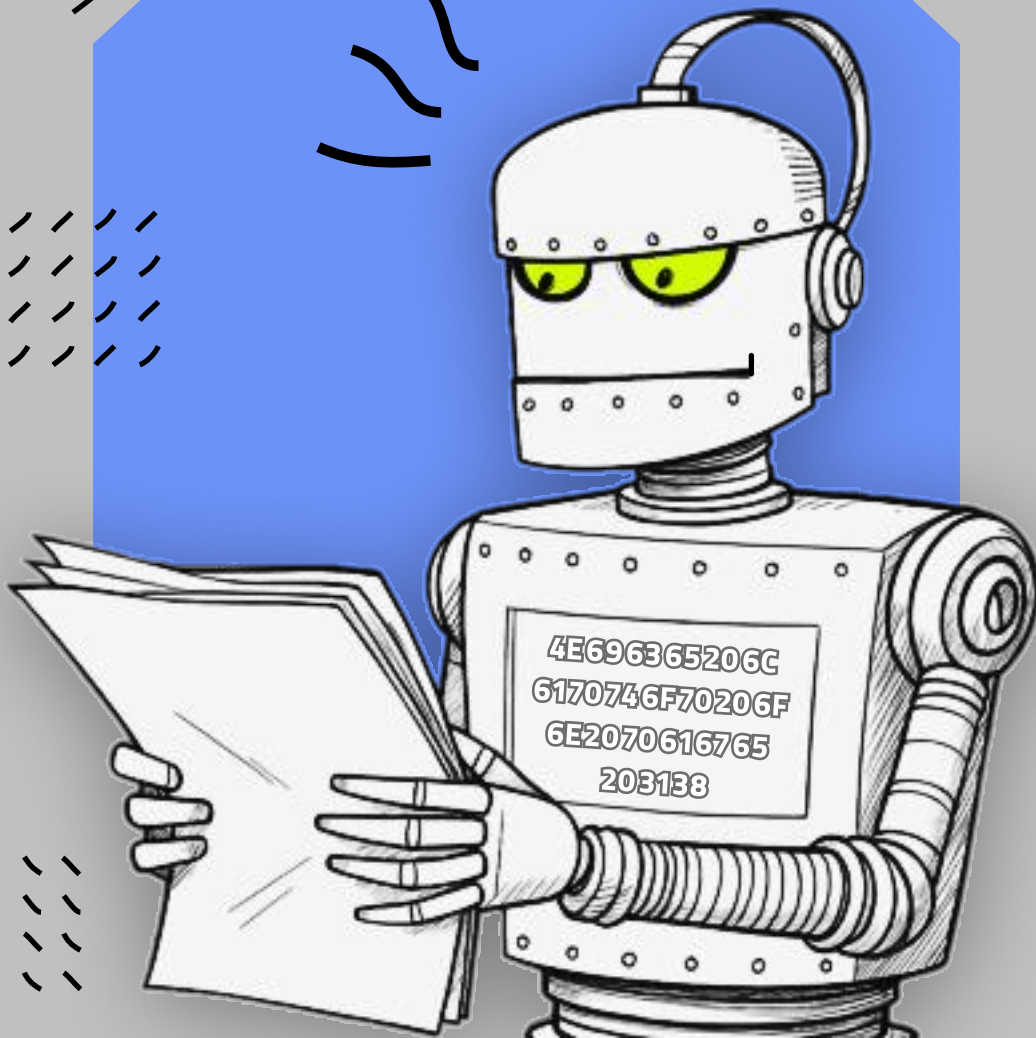


byte

اردیبهشت ۱۴۰۴



فهرست

۱	آرش شاه حسینی	۳ دقیقه	ب
۲	کدهایی که آینده را می نویسند (سرمقاله)		
۳	سیدپارسا نشایی	۵ دقیقه	ب
۴	پیشگیری از «بیرون افتادن» دانشجویان در دوره های آنلاین: چگونه به کمک هوش مصنوعی، نوش دارو را قبل از مرگ سهراب برسانیم؟		
۵	عماد امام جمعه	۴ دقیقه	ب
۶	اسیر درجه آزادی، یک بار برای همیشه؛ چرا $\frac{1}{n-1}$		
۷	آروین طاهری، متین غیائی	۵ دقیقه	ب
۸	منطق کامل		
۹	تحریریه	۳ دقیقه	ب
۱۰	الگوریتم یا خوارزمی		
۱۱	معین آعلی	۵ دقیقه	ب
۱۲	از شمارنده تا زمان بندی؛ بررسی OTP های مبتنی بر زمان و شمارنده		
۱۳	تحریریه	۳ دقیقه	ب
۱۴	دانستنی ها (به شیوه قدیمی)		
۱۵	امیرحسین انصاری	۸ دقیقه	ب
۱۶	سیستم های رایا فیزیکی: معماری، چالش ها و مسیر تحول در صنایع هوشمند		
۱۷	جمعی از دانشجویان	۹ دقیقه	ب
۱۸	برای آزادی روح از خستگی ها بنویس؛ مروری بر روزمره نویسی های دانشجویان		
۱۹	سعید فراتی کاشانی	۴ دقیقه	ب
۲۰	ربات تلگرام به سبک دانشجویی		
۲۱	بردیا رضایی کلانتری	۷ دقیقه	ب
۲۲	قدم به دنیای دواپس		
۲۳	تحریریه	۶ دقیقه	ب
۲۴	داستان احیای امپراطوری درهم شکسته		
۲۵	امیرحسین شهیدی	۷ دقیقه	ب
۲۶	مصاحبه با لیل اسادات علوی؛ هم بنیان گذار گفتا		
۲۷	مهدی علی نژاد	۳ دقیقه	ب
۲۸	مصاحبه با مهدی بحرینی؛ ثابت افزار و اهمیت آن		
۲۹			

صاحب امتیاز: انجمن علمی دانشکده مهندسی کامپیوتر (SSC)

مدیرمسئول: امیرحسین شهیدی

سردبیر: آرش شاه حسینی

صفحه آرایی و گرافیک: آرش شاه حسینی، امیررضا اینانلو، معین آعلی

ویراستاران: امیرمهدی نامجو، امیرحسین صوری

تحریریه: آرمان طهماسبی، معین آعلی، مهدی علی نژاد، امیرمهدی

نامجو، محمد مصیبی، امیرحسین محمدزاده



کدهایی که آینده را می‌نویسند

سرمقاله

گاهی این تغییر آن قدر آرام است که حتی خودمان هم آن را حس نمی‌کنیم؛ یک تصمیم ساده، یک عادت کوچک که ترک می‌شود، یک خط کدی که مسیر اجرای یک برنامه را دگرگون می‌کند. اما همین تغییرهای درونی، اگر جدی گرفته شوند، قدرت آن را دارند که دنیایی را متحول کنند.

«چه کسی خواهان تغییر است...» در ظاهر، همه ما به تغییر تمایل داریم. اما حقیقت این است که تغییر تنها زمانی میسر می‌شود که انسان‌ها جرئت مواجهه با ناشناخته‌ها را داشته باشند. برای پذیرفتن مسئولیت ساختن فردایی متفاوت، باید از دایره امن خود فراتر رفت. تغییر، فرآیندی ساده نیست. در آن شک و ترس و شکست وجود دارد.

اما در دل همین دشواری‌هاست که رشد و پیشرفت شکل می‌گیرد. هر کسی که به دنبال تغییر در جهان است، باید نخستین گام را در تغییر درون خود بردارد.

نوبت اجرای دستور است

حالا که تابع تغییر را نوشته‌ایم، وقت اجراست؛ وقت تماشای این صحنه که دیگر محدود به صفحه مانیتور نمی‌ماند و از مرزهای دیجیتال فراتر رفته و دنیای بیرون را در می‌نوردد...

دستور اجرا صادر شده است. سیستم در حال راه اندازی...

آغاز فرآیند ساخت آینده [INFO] #####
 تمامی تست‌ها با موفقیت پشت سر گذاشته شدند [INFO] #####
 برنامه به درستی اجرا شد [INFO] #####
 فرآیند تکمیل شد، آینده اکنون در دست ماست [SUCCESS] #####

خطای ۴۰۴: آینده یافت نشد؟

تصور کنید یک سیستم در حال اجراست، اما ناگهان با خطای «۴۰۴: آینده یافت نشد» مواجه می‌شود. چرا؟ چون هیچ‌کس برای ساختنش اقدامی نکرده است. این خطا نشان می‌دهد که بدون اقدام و برنامه‌ریزی، آینده هیچ‌گاه به سرانجام نخواهد رسید. در دنیای دیجیتال، این خطا نتیجه ناتوانی در طراحی و ساخت سیستم‌هایی است که می‌توانند به پیشرفت و تغییر منجر شوند. اما در دنیای واقعی هم، تغییر و پیشرفت بدون اقدام، تنها یک آرزو باقی می‌ماند.

شاید دور از ذهن نباشد اگر بگوییم پیشرفت بشر در عصر کنونی حاصل همین چند خط کدی است که در حافظه ماشین‌ها ثبت، در پردازنده‌ها پردازش می‌شود و در شبکه‌ها جریان می‌یابد. هر الگوریتمی که بهینه‌تر می‌شود، هر معماری‌ای که طراحی می‌شود و هر خط برنامه‌ای که نوشته می‌شود، بخشی از همین آینده را شکل می‌دهد. اما سؤال اینجاست: آیا ما فقط نظاره‌گر این این قطار سریع‌السیر «پیشرفت» هستیم و ایستاده در سکوی انتظار؟ یا بر صندلی‌های «مسافرین ویژه» تکیه زده ایم و هم‌سو با تحولات پیش می‌رویم؟ شاید پاسخ در جایی دیگر نهفته باشد؛ جایی در درون خود ما. همان جا که تغییر آغاز می‌شود...

الگوریتم‌های تغییر

ت.غ.پ.ی.ر؛ تماشای غروب یک یگانگی رنج‌آور. نقطه آغاز تغییر، در درون ماست، از لحظه‌ای که در پس ذهن مان ندای اعتراض بلند می‌شود، از زمانی که وضع موجود دیگر ما را راضی نمی‌کند، از دل نارضایتی می‌جوشد. شاید این حرف تکراری به نظر برسد که «تغییر باید از خودمان شروع شود».

اما این تکرار است که حقیقت را باز گو می‌کند.

byte





پیشگیری از «بیرون افتادن» دانشجویان در دوره‌های آنلاین: چگونه به کمک هوش مصنوعی، نوش دارو را قبل از مرگ سهراب برسانیم؟

در گذشته‌های دور (!) برای یاد گرفتن درس‌ها در مدرسه یا دانشگاه مجبور بودیم سر کلاس به شکل حضوری شرکت کنیم. به مرور با رشد تکنولوژی (و رویدادهایی مثل پاندمی کووید-۱۹)، شرکت در کلاس‌های آنلاین محبوبیت روزافزون پیدا کرد. بدون شک بسیاری از شما خوانندگان این متن، در کلاس‌های آنلاین پلتفرم‌هایی مثل کورسرا، Udemey یا موارد مشابه شرکت کرده‌اید؛ به چنین پلتفرم‌هایی، به طور کلی، MOOC (مخفف Massive Open Online Course) هم گفته می‌شود. MOOC ها تجربه یادگیری با کیفیت بالا، با سرعت مطلوب هر شخص، و در برخی موارد رایگان را در دسترس میلیون‌ها نفر از سراسر جهان، مستقل از مکان جغرافیایی‌شان، قرار می‌دهند. در چنین پلتفرم‌هایی، دانش‌آموزان و دانشجویان می‌توانند با روندی که خود ترجیح می‌دهند کلاس‌های درس را در زمان و مکان مناسب دنبال کنند و با آرامش کامل به حل تمرین‌ها بپردازند.

با این وجود، یک مانع بزرگ سر راه استفاده از MOOC ها وجود دارد. بدون شک بسیاری از خوانندگان که تجربه استفاده از یک MOOC را داشته‌اند، تجربه «به پایان نرساندن» یک MOOC را هم داشته‌اند؛ نرخ عدم اتمام یا «بیرون افتادن» (dropout rate) بالای دانشجویان در MOOC ها و درصد پایین کسانی که یک MOOC را با موفقیت به اتمام می‌رسانند، مشکل کوچکی نیست، بلکه نمایانگر یک مانع بزرگ در محیط‌های یادگیری آنلاین و نشانگر تفاوت‌های شدید در انگیزه دانشجویان برای شرکت در یک MOOC است. آمار dropout rate نگران‌کننده است؛ Onah و همکاران نشان دادند که نرخ به پایان رساندن بسیاری از MOOC ها کم‌تر از ۱۳ درصد است، به این معنا که بیش از ۸۷ درصد دانشجویان، این MOOC ها را به پایان نمی‌رسانند. به عنوان مثال، بسیاری از دانشجویان ممکن است صرفاً به دلیل داشتن مقداری زمان خالی در برنامه خود، در یک MOOC ثبت‌نام کنند، بدون آن‌که علاقه شدیدی به موضوع آن درس داشته باشند. برخی دیگر از دانشجویان به هر دلیل از روند درس عقب می‌مانند و پس از انباشته شدن هفته‌ها، انگیزه خود برای بازگشت به مسیر اصلی را از دست می‌دهند.

این اتفاقات ممکن است سبب شوند که دانشجویان مطالب درسی MOOC را به پایان نرسانند و در نتیجه زمان و هزینه صرف شده برای MOOC هدر برود.

از صحبت‌های فوق نتیجه می‌گیریم که درک چگونگی رخ دادن این اتفاق ناگوار، و یافتن راه‌حل‌هایی برای آن، بسیار ضروری است. یک راه‌حل مناسب می‌تواند این باشد که مدرسین و اساتید درس‌های ارائه‌شده بر بستر MOOC بتوانند نشانه‌های دانشجویانی که احتمال dropout بالا دارند را زودتر تشخیص دهند و حمایت لازم را از آن‌ها قبل از این‌که تصمیم به dropout بگیرند، انجام دهند. این حمایت می‌تواند در انواع مختلفی صورت گیرد، از جمله برگزاری کلاس‌های حل تمرین برای کمک به یادگیری مطالب، ارائه فیلم‌های مروری بر هفته‌های قبلی، جلسات رفع اشکال و نظایر آن.

تلاش‌های قبلی در راستای پیش‌بینی دانشجویانی که احتمال dropout دارند، از طریق پرسیدن سوال‌های دوره‌ای در قالب نظرسنجی بوده است، اما این روش چالش‌هایی هم‌چون پر نکردن نظرسنجی توسط دانشجویان با انگیزه پایین‌تر یا عدم توانایی پاسخ‌دهی صادقانه به سوالات نظرسنجی را به همراه دارد. با این وجود، به دلیل رشد استفاده از پلتفرم‌های آنلاین برای یادگیری، محققین می‌توانند داده‌های دقیقی از چگونگی استفاده دانشجویان از پلتفرم MOOC را جمع کرده و به کمک مدل‌های یادگیری ماشین، از آن‌ها برای پیش‌بینی dropout دانشجویان استفاده کنند.

بسیاری از تلاش‌ها در این خصوص در دامنه کاری «داده‌کاوی آموزشی» (Educational Data Mining) قرار می‌گیرند. محققانی هم‌چون Whitehill و همکاران (۲۰۱۷) در مقالات خود بررسی کردند که چگونه می‌توان از ویژگی‌هایی مانند ردیابی رفتار دیجیتال دانشجویان هنگام دیدن ویدیوهای درس و یا مشارکت آن‌ها در آزمونک‌های درسی برای پیش‌بینی احتمال به پایان رساندن درس‌ها استفاده کرد.

محققین شاخه EDM دریافته‌اند که الگوهایمانند pause های متداول یا بازیابی خارج از برنامه فیلم‌های درس، برنامه‌ریزی درسی نامنظم و یا ارسال کردن پاسخ تمرین‌ها با تأخیر می‌توانند نمایانگر مشکلاتی در روند دانشجویان باشند.

یکی از مقالات پژوهشی مهم در این راستا، مقاله Swamy و همکاران (۲۰۲۲) در کنفرانس Learning at Scale بوده است که به طور خاص، در این نوشتار، بررسی دقیق‌تری روی آن انجام می‌دهیم. یکی از چالش‌های کارهای قبلی، عدم گسترش مدل‌های یادگرفته‌شده روی یک دادگان آموزشی به دادگان‌های دیگر بوده است که سبب می‌شده برای هر MOOC نیاز به یادگیری مدل جدیدی وجود داشته باشد. Swamy و همکاران برای حل این مشکل، اطلاعات فعالیت‌ها (interaction ها) دانشجویان با MOOC را با فراداده‌هایی (metadata) در خصوص ویژگی‌های کلی درس ترکیب کردند تا مدلی توسعه دهند که در درس‌ها و موضوعات مختلف قابل استفاده باشد. برای این منظور، آن‌ها از تکنیک مرسوم در یادگیری ماشین موسوم به transfer learning استفاده کردند. در این روش، مدل‌های یادگیری ماشین روی دادگان‌های وسیعی از MOOC های قبلی آموزش می‌بینند تا مدل بتواند به جای وابستگی به ویژگی‌های مخصوص به یک درس یا موضوع خاص، الگوهای رفتاری عمومی دانشجویان که در دروس و موضوعات مختلف معتبر هستند را یاد بگیرد.

در مقاله مذکور، Swamy و دیگران در ابتدا دادگانی متشکل از ۲۶ دوره MOOC ارائه‌شده توسط EPFL شامل اطلاعات بالای ۱۴۵ هزار دانشجو و میلیون‌ها interaction دانشجویان با پلتفرم MOOC را اخذ کردند. سپس، آنان از دادگان خام، «خصوصیات رفتاری» (behavioral features) متعددی را استخراج کردند که هر یک نمایانگر بعد (dimension) متفاوتی از چگونگی تعامل دانشجویان با MOOC مربوطه بوده است. این خصوصیات رفتاری در گروه‌های مختلفی قرار گرفتند که چند نمونه از آنان عبارتند از:

«نظم»: رفتارهای یادگیری دانشجویان تا چه حد منظم است؟

● آیا زمان ورود (login) دانشجویان به پلتفرم هر روز یکسان است یا به مرور زمان تغییر می‌کند؟ آیا دانشجویان بیشتر در طول هفته درس می‌خوانند یا آخر هفته؟

● «تعامل»: دانشجویان به چه میزان روی موارد مختلف در پلتفرم MOOC کلیک کرده و با آن‌ها تعامل می‌کنند؟ چگونگی این رفتار، چه تفاوت‌هایی بین روزهای هفته و آخر هفته‌ها دارد؟

● «رفتار مشاهده فیلم»: آیا دانشجویان فیلم‌ها را از اول تا آخر بدون وقفه تماشا می‌کنند؟ آیا بخش‌های خاصی را skip کرده یا دوباره پخش می‌کنند؟ آیا الگوی مشاهده دانشجویان به صورت پیوسته است یا شامل وقفه، بازپخش یا رد کردن بخش‌هایی از محتواست؟

● «مشارکت در آزمون»: آیا دانشجویان در آزمونک‌های درس شرکت کرده‌اند؟ آیا توانسته‌اند در اولین شرکت در آزمونک، آن را با موفقیت بگذرانند؟

علاوه بر موارد فوق که مربوط به خصوصیات رفتاری دانشجویان می‌شود، نویسندگان مقاله مذکور، ویژگی‌های فراداده‌ای دیگری را نیز که مربوط به کلیت درس می‌شود، به لیست ویژگی‌های فوق افزودند، مانند مدت زمان ارائه شدن درس، سطح درس (مبتدی، متوسط، پیشرفته، ...)، زبان درس (انگلیسی یا فرانسوی)، و نیز توضیحات مربوط به درس که با روش‌های پردازش زبان طبیعی (NLP) به ویژگی‌های قابل پردازش تبدیل شدند.

تمام ویژگی‌های فوق به یک مدل BiLSTM داده شدند. این مدل نوعی از مدل‌های خانواده بزرگ‌تر RNN هاست که برای تحلیل سری‌های زمانی (time series)، مانند مسئله فوق، می‌توان از آن استفاده کرد (در این نوشتار، از بیان جزئیات معماری مدل و نیز فرآیند fine tuning آن، صرف‌نظر می‌کنیم). ارزیابی مدل در دو مرحله انجام گرفت: پس از تکمیل ۴۰ درصد هر MOOC و پس از تکمیل ۶۰ درصد هر MOOC. به این طریق، نویسندگان دریافتند که آیا می‌توان پیش‌بینی‌ای با دقت کافی از موفقیت یا عدم موفقیت (pass or fail) دانشجویان پس از گذشت ۴۰ یا ۶۰ درصد از بازه زمانی یک MOOC داشت یا خیر.

در نهایت، پس از ارزیابی، Swamy و دیگران دریافتند که:

● عملکرد مدل در عملیات transfer به ازای هر درس، به تعداد دفعاتی که درس مذکور قبلاً ارائه شده، تعداد دانشجویانی که قبلاً در درس مذکور مشارکت کرده‌اند، و نرخ به پایان رساندن موفقیت‌آمیز درس توسط آنان بستگی دارد.

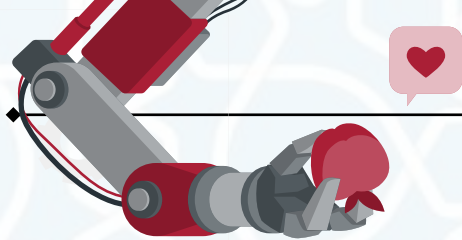
● ترکیب ویژگی‌های رفتاری دانشجویان (مانند نظم، تعامل و غیره) با ویژگی‌های مختص به هر درس (مانند سطح، زبان و غیره) باعث افزایش کارایی مدل برای دروسی که تا کنون در دادگان حضور نداشته‌اند می‌شود. به طور خاص، ویژگی‌های «سطح درس»، «عنوان درس»، و «مدت زمان درس» برای پیش‌بینی موفقیت‌آمیز dropout در مقاطع اولیه تأثیر قابل توجهی دارند.

مقاله Swamy و دیگران نشان می‌دهد که ویژگی‌های قابل جمع‌آوری از یک پلتفرم MOOC توانایی پیش‌بینی نرخ dropout دانشجویان را دارند. اهمیت تحقیقاتی مانند این مقاله، تنها در پیش‌بینی dropout نیست، بلکه یافته‌های این مقاله، نویدبخش آن هستند که اساتید درس می‌توانند در موقع لازم برای دانشجویانی که خطر dropout را دارند وارد عمل شده و در صورت تمایل از این اتفاق جلوگیری کنند. به عنوان مثال، فرض کنید که در یک MOOC، دستیار مجهز به هوش مصنوعی تشخیص می‌دهد که دانشجویی به خصوصی احتمال dropout دارد (مثلاً از طریق الگوهای یادگیری نامنظم یا تحویل ندادن تمرین‌ها و شرکت نکردن در آزمونک‌ها). این دستیار مجهز به هوش مصنوعی می‌تواند اساتید درس را از این حقیقت آگاه کرده و به آن‌ها پیشنهادهایی برای کمک به دانشجویان دهد (به عنوان مثال، کلاس‌های رفع اشکال یا انگیزه‌بخشی به دانشجویان). این «دخالت‌ها» (interventions) در روند درس می‌توانند به افزایش نرخ گذراندن موفقیت‌آمیز MOOC ها توسط دانشجویان کمک ویژه‌ای کنند.

آنچه تا این‌جا دیدیم، نشان می‌دهد که هوش مصنوعی و داده‌کاوی آموزشی می‌توانند نقشی کلیدی در شناسایی و پیشگیری از dropout در دوره‌های آنلاین ایفا کنند. به کمک تحلیل داده‌های رفتاری دانشجویان و درک الگوهای تعامل آن‌ها با پلتفرم، می‌توان پیش از آن‌که دانشجو از مسیر یادگیری خارج شود، وارد عمل شد و حمایت لازم را فراهم کرد. به زبان ساده‌تر، چنین سامانه‌هایی مانند نوشداروهای پیش‌گیرانه عمل می‌کنند: یک پزشک ماهر، لزوماً منتظر نمی‌ماند تا بیماری به مرحله اوج خود برسد و سپس درمان را آغاز کند، بلکه به دنبال نشانه‌های اولیه از احتمال بروز بیماری می‌گردد تا بتواند در زمان مناسب در روند گسترش بیماری «دخالت» کند. به طرز مشابه، اساتید درس نیز می‌توانند مدل‌های یادگیری ماشین پیش‌بینی dropout دانشجویان را به عنوان نوعی از «داروی آموزشی» به کار گیرند و با دخالت در زمان مناسب، اطمینان حاصل کنند که دانشجو از منظر دچار «بیماری درسی» نمی‌شود.

با این حال، استفاده از این ابزارها و مدل‌ها، تنها بخشی از راه حل است. طراحی مداخلات مؤثر، تعامل انسانی معنادار میان استاد و دانشجو و ایجاد حس تعلق در کلاس‌های آنلاین از دیگر عوامل مهمی هستند که باید در کنار ابزارهای هوشمند استفاده شوند تا مسیر موفقیت دانشجویان هموار شود. اگر بتوانیم سامانه‌هایی بسازیم که به موقع به اساتید و دانشجویان هشدار دهند، پشتیبانی فراهم کنند و به نیازهای فردی هر دانشجو پاسخ مناسب دهند، آن‌گاه می‌توانیم امیدوار باشیم که درصد بیشتری از دانشجویان، مسیر یادگیری خود را با موفقیت و بدون مشکل به پایان برسانند.





اسیر درجه آزادی

یک بار برای همیشه؛ چرا $\frac{1}{n-1}$ ؟

عماد امام جمعه

کارشناسی ۱۴۰۰



اگر درس آمار و احتمال رو گذروندید، احتمالاً به رابطه $s^2 = \frac{1}{n-1} \sum (x_i - \bar{x})^2$ برای تخمین واریانس مجموعه‌ای از داده‌ها برخورد کرده باشید. برای همه در نگاه اول تقسیم بر $n-1$ اتفاقی غیرمنتظره و غیرشهودی به نظر می‌رسد. عده خوبی شاید با شنیدن جمله « $n-1$ درجه آزادی است» به همراه توضیحاتی غیرمنسجم و گنگ از طرف استاد به راحتی از مسئله عبور کرده باشند. اما شاید هم سؤالات بی‌جوابی مثل «درجه آزادی یعنی چی؟»، «درجه آزادی رو چجوری حساب می‌کنن اصلاً؟»، «درجه آزادی چه ربطی به محاسبه واریانس داره؟» و «این فرمول عجیب درجه آزادی تو تست مربع کای و تست تی^۲ چی میگه؟» گوشه‌ای از ذهن کنجکاوشما آزارتون می‌ده.

تحلیل کامل درجه آزادی و کاربردهاش، برخلاف ظاهر ساده‌اش پیچیدگی‌های زیادی داره و از حوصله متن خارجه. اما تلاش مون رو می‌کنیم که به افکار آشفته‌ای که احتمالاً درباره این موضوع پیدا کردید، کمی ساختار بدیم. ابتدا سعی می‌کنیم تعریف دقیق‌تری از درجه آزادی ارائه بدیم و سپس برای فهم بیشتر در مورد ارتباط درجه آزادی با محاسبه واریانس عمیق می‌شیم.

درجه آزادی چیه؟

«تعداد داده‌هایی در محاسبه نهایی یک آماره، که آزاد هستند تغییر کنند.» این تعریف رسمی‌ایه که از درجه آزادی یا Degree of Freedom توی آمار ارائه میشه. یک تعریف معادل خوب دیگه‌اش میشه: «تعداد داده‌های مستقلی که برای تخمین آماره استفاده می‌شود». ولی خب اینا اصلاً یعنی چی؟

فرض کنید که n متغیر تصادفی مستقل رو مشاهده کردیم، و بهتون می‌گم که میانگین متغیرها برابر μ شده. حالا شما به چندتا از متغیرها نیاز دارید تا مقدار کل متغیرها رو متوجه بشید؟

درسته $n-1$! چون با دونستن $n-1$ ای از متغیرها و میانگین، می‌تونید متغیر n ام رو به شکل یکتا مشخص کنید. پس این‌جا می‌گیم که درجه آزادی متغیرهای من، با داشتن میانگین، برابر $n-1$ هست، چون فقط $n-1$ تا از متغیرها می‌تونن آزادانه تغییر کنند.

به عبارتی $n-1$ تا از متغیرها مستقل هستند، پس می‌گیم که درجه آزادی برابر $n-1$ هست. توجه کنید که این اتفاق به خاطر داشتن دانش اضافه میانگین رخ داد و اگه میانگین متغیرها رو نداشتیم، همه متغیرها مستقل می‌شدند. دقت کنید که موقع محاسبه واریانس هم اطلاعاتی که داریم، دقیقاً مشابه سناریوی بالاست.

برای درک بهتر مفهوم، بیاید درجه آزادی سیستم مورد استفاده در تست مربع کای رو هم ببینیم.

توی این تست، یک جدول n در m داریم، که جمع مقادیر هر سطر و هر ستون رو در اختیار داریم. حالا به نظرتون با اطلاعاتی که فرض گرفتیم، به چند خونه از جدول برای بازسازی کل جدول نیاز داریم؟

با یکم تفکر می‌تونید ببینید که $(n-1) \times (m-1)$ خونه از جدول برای بازسازی کل جدول کافیه، که این با درجه آزادی‌ای که تو تست مربع کای داشتیم مطابقت داره.

↑ ADVANCE KNOWLEDGE = ↓ DEGREES OF FREEDOM

یک شهود خوب دیگه اینه که فرض کنید n متغیر به شکل یک بردار تصادفی n بعدی بوده. این بردار توی یک فضای n بعدی می‌تونه هر مقداری بگیره، اما وقتی که میانگین درایه‌ها (یا درواقع نرم-۱ بردار) رو داشته باشید، با یه کم جبر بازی می‌فهمید که این بردار فقط می‌تونه توی یک زیرصفحه $n-1$ بعدی قرار بگیره. پس یه تعبیر دیگه از درجه آزادی، بعد فضای دامنه این بردار تصادفی از داده‌هاست. از این شهود جلوتر استفاده می‌کنیم.

چرا نه؟

فرض کنید که n تا نمونه^۳ از یک توزیع (مثلاً نرمال با میانگین μ و واریانس σ^2) در اختیار داریم. تخمین زدن μ چالشی برامون نداره و کافیه میانگین سمپل‌ها ($\bar{X}$) رو حساب کنیم، و تخمین گرمون بایاس نداره.

اما برای تخمین σ^2 قضیه سخت‌تره. در نگاه اول به نظر کافیه که مثل فرمول واریانس، میانگین مربع فاصله سمپل‌ها تا میانگین رو حساب کنیم. اما سوال جالب این‌جاست که کدام میانگین؟

میانگین واقعی توزیع $(s^2 = \frac{1}{n} \sum (x_i - \mu)^2)$ اگه میانگین واقعی رو داشتیم، بله. این روش جواب می‌داد و تخمین‌گر بدون بایاسی هست. اما ما میانگین واقعی توزیع رو نمی‌دونیم و فقط یک تخمین‌گر ازش داریم که به احتمال زیاد، برابر میانگین واقعی نیست.

پس، تخمین‌گر میانگین $(s^2 = \frac{1}{n} \sum (x_i - \bar{x})^2)$ در نگاه اول شاید منطقی به نظر برسه، اما این باعث می‌شه که به نوعی رابطه تخمین‌گر خودساخته‌ای از واریانس بسازیم و تخمین گرمون بایاس پیدا کنه.

بایاس از کجا؟

بایاس از اون‌جا میاد که μ مقادیر مختلفی می‌تونه پیدا کنه، و طبق رابطه اصلی واریانس به ازای هر کدوم‌شون واریانس متفاوتی حساب می‌شه. یعنی اگه که واقعاً داشته باشیم $\mu = \bar{X}$ ، همه چی ردیفه. اما معمولاً این‌طور نیست. در واقع می‌شه دید که وقتی میو (μ) با $\bar{X}$ جایگزین می‌کنیم، مقدار s^2 به کمترین مقدار خودش می‌رسه - یعنی داریم:

$$\bar{X} = \arg \min_{\mu} \left(\frac{1}{n} \sum (x_i - \mu)^2 \right)$$

پس همواره داریم واریانس رو underestimate می‌کنیم و این باعث ایجاد بایاس رو به پایین می‌شه. جلوتر این رو بهتر نشون می‌بینیم.

چرا درجه آزادی آره؟

یک روش مرسوم رسیدن به رابطه $s^2 = \bar{X} = \arg \min_{\mu} \left(\frac{1}{n} \sum (x_i - \mu)^2 \right)$ با استفاده از Bessel's Correction و محاسبه Bias هست، که همه‌ش جبریه و ربطی هم به درجه آزادی نداره. این‌جا سعی می‌کنیم روشی رو نشون بدیم که بیشتر ارتباطش با درجه آزادی ملموس باشه.

فرض کنید که n متغیر تصادفی از توزیعی که می‌خواید واریانس‌شو تخمین بزنید گرفتید. حالا اون‌ها رو به شکل یک بردار n بعدی ببینید. هم‌زمان توی شکل‌های زیر مثالی با ۲ داده (و در نتیجه یک بردار ۲ بعدی) داده شده. اگه میانگین واقعی توزیع (μ) رو داشتیم، کافی بود که این بردار رو از بردار $\mu \times 1$ کم کنیم (داده‌ها رو mean-centered کنیم) و به بردار $X - \mu \times 1$ می‌رسیم.



پس دیدیم که چجوری داریم با کم کردن تخمین‌مون از میانگین، به جای میانگین واقعی توزیع، ابعاد داده‌هامون رو یکی کاهش می‌دیم و به‌نوعی تخمین‌مون از واریانس توی یک بعد (درجه آزادی) کمتر داره انجام میشه. (توی شکل زیر توجه کنید که توی فضای n بعدی، بردار residuals همیشه از mean کوچیک‌تره، که underestimate کردن شیوه اشتباه تخمین واریانس رو توجیه می‌کنه. اما وقتی اونا رو به فضای $n-1$ بعدی پروجکت می‌کنیم، هر دوشون یک اندازه‌اند.)

الان می‌تونید ببینید که مربع طول این بردار، n برابر واریانس داده‌ها میشه. پس با محاسبه طول بردار می‌تونیم واریانس رو حساب کنیم.

$$\text{length}(X - \mu 1) = \sqrt{\sum_{i=1}^n (X - \mu 1)_i^2}$$

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^n (X - \mu 1)_i^2$$

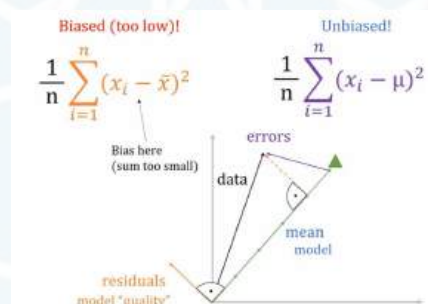
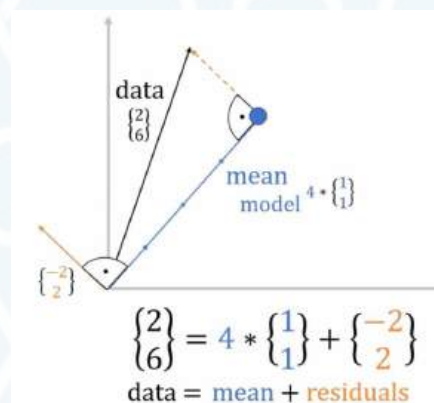
$$\text{length}(X - \bar{X} 1)^2 = n\sigma^2$$

اما حالا که میانگین رو نداریم، مجبوریم از میانگین n متغیری که داریم ($\bar{X}$)، به‌عنوان تخمین میانگین استفاده کنیم. نکته جالب اینه که بردار $X - \bar{X} 1$ همیشه بر بردار $\bar{X} 1$ عموده. این رو با بررسی ضرب داخلی این دو بردار یا نگاه به شکل زیر متوجه می‌شید.

پس وقتی بردار حاصل (residuals) همیشه بر بردار 1 عموده، همیشه توی یک زیرفضای $n-1$ بعدی قرار داره. تعبیر برداری درجه آزادی که معادل کاهش بعد هست رو دقیقاً می‌تونید این‌جا ببینید.

این، یعنی برخلاف حالتی که رو می‌دونستیم و بر اساس داده‌ها واریانس رو برای یک بردار n بعدی حساب می‌کردیم، این‌جا همیشه داریم از روی داده‌ها یک بردار $n-1$ بعدی می‌سازیم. به عبارتی، بردار $n-1$ بعدی residuals، سایه بردار n بعدی errors هست. پس برای محاسبه واریانس باید مربع طول این بردار رو تقسیم بر بعدش ($n-1$) کنیم.

$$\text{len}(X - \bar{X} 1)^2 = (n - 1)\sigma^2$$





آروین طاهری

کارشناسی ۲



متین غیاتی

کارشناسی ۲

منطق کامل

چکیده

در این مقاله ابتدا تعریف منطقی را به همراه تعدادی مثال بیان می‌کنیم و منطق کامل بودن مجموعه‌ای از این توابع را تعریف می‌کنیم. سپس شروطی لازم برای منطق کامل بودن یک مجموعه دلخواه از توابع ارائه می‌دهیم و پس از آن کافی بودن این شروط را ثابت می‌کنیم.

معرفی توابع منطقی

تعاریف

به طور کلی، توابع منطقی دسته‌ای از توابع چندمتغیره هستند که اگر در ورودی آن‌ها ۰ یا ۱ قرار گیرد، خروجی آن‌ها نیز ۰ یا ۱ خواهد بود. این توابع را این‌گونه نشان می‌دهیم:

$$(l) n \in \mathbb{N} \cup \{0\}, f: \{0,1\}^n \rightarrow \{0,1\}$$

از معروف‌ترین این توابع که در ساخت مدارهای منطقی کاربرد دارند، می‌توان به تابع‌های AND، NOT، OR اشاره کرد. از این به بعد برای نشان دادن نحوه کار هر تابع، آن را با جدول درستی آن نشان می‌دهیم. جدول درستی سه تابع ذکرشده را در شکل ۱ می‌بینید. برای مثال، تابع AND تنها در صورتی برابر با ۱ می‌شود که هر دو ورودی آن برابر با ۱ باشند؛ و تابع OR تنها زمانی ۰ می‌شود که هر دو ورودی آن برابر با ۰ باشند.

مسئله ۱. هر سه این توابع را می‌توان به صورت یک چندجمله‌ای دومتغیره از ورودی‌هایشان نمایش داد. برای مثال $AND(x, y) = xy$. آیا می‌توانید بگویید چندجمله‌ای $OR(x, y)$ چه خواهد بود؟

آشنایی با جبر بول

برای نشان دادن ترکیب‌های مختلف توابع AND، OR، NOT که توابع ابتدایی ما هستند، نمادهایی تعریف می‌کنیم.

$$x + y := OR(x, y) \quad xy = x \times y := AND(x, y) \quad (2)$$

$$\bar{x} := NOT(x)$$

حال می‌توانیم توابع مختلفی با استفاده از ترکیب این توابع با هم ایجاد کنیم. تعدادی از خواص این توابع را معرفی می‌کنیم که به دلخواه خود می‌توانید آن‌ها را اثبات کنید.

۱. $(x + y) + z = x + (y + z), x(yz) = (xy)z$
۲. $z(x + y) = zx + zy$
۳. $x + x = x, x \times x = x$
۴. $\overline{xy} = \bar{x} + \bar{y}, \overline{x + y} = \bar{x} \times \bar{y}$

این تابع به ازای چه ورودی‌هایی برابر با ۱ می‌شود؟

$$f(x_1, x_2, x_3) = x_1x_2 + x_2x_3 \quad (3)$$

ساخت همه توابع با توابع ابتدایی

قضیه ۱. هر تابع منطقی را می‌توانیم با ترکیب تعدادی از توابع AND، OR، NOT بسازیم. در این‌جا به اثبات دقیق این گزاره نمی‌پردازیم؛ اما مثالی از تبدیل تابعی به ترکیبی از توابع ساده‌تر در شکل ۲ می‌زنیم تا خودتان با استفاده از آن، گزاره را در حالت کلی ثابت کنید.

منطق کامل

تعریف و مثال

تعریف ۱. به مجموعه‌ای از توابع منطقی، منطق کامل گفته می‌شود اگر که بتوان با استفاده از آن توابع و ترکیبشان با یکدیگر، تمام توابع منطقی را تولید کرد.

برای مثال مجموعه توابع $\{AND(x, y), OR(x, y), NOT(x)\}$ منطق کامل بودند، اما $\{OR(x, y)\}$ توسط دو تابع دیگر قابل ساخت است. داریم:

$$OR(x, y) = NOT(AND(NOT(x), NOT(y))) \quad (4)$$

پس می‌توانیم بگوییم که مجموعه توابع $\{AND(x, y), NOT(x)\}$ نیز منطق کامل هستند. تابع جدیدی معرفی می‌کنیم:

$$NAND(x, y) := NOT(AND(x, y)) \quad (5)$$

با استفاده از تابع NAND، می‌توانیم توابع AND و NOT را بسازیم:

$$NOT(x) = NAND(x, x) \quad (6)$$

$$AND(x, y) = NAND(NAND(x, y), NAND(x, y)) \quad (7)$$

و بدین ترتیب ثابت کردیم که مجموعه $\{NAND(x, y)\}$ نیز یک منطق کامل است.

مسئله ۳. ثابت کنید مجموعه توابع زیر نیز منطق کامل هستند:

۱. $\{OR(x, y), NOT(x)\}$
۲. $\{NOT(OR(x, y))\}$

ناوردا

مسئله ۴. توابع زیر را در نظر بگیرید:

$$f_1(x, y, z) = xy + yz \quad (8)$$

$$f_2(x, y) = x + y \quad (9)$$

$$f_3(x) = 1 \quad (10)$$

آیا مجموعه این توابع می‌تواند منطق کامل باشد؟

جواب این مسئله، خیر است. تابعی در نظر بگیرید که اگر تمام ورودی‌هایش ۱ باشد خروجی آن برابر با ۰ باشد. به نظر شما، آیا این سه تابع قادر به ساختن آن خواهند بود؟

$$f_1(1, 1, 1) = 1 + 1 = 1 \quad (11)$$

$$f_2(1, 1) = 1 + 0 = 1 \quad (12)$$

$$f_3(1) = 1 \quad (13)$$

این سه تابع اگر تمام ورودی‌هایشان برابر باشد،

خروجی‌شان نیز برابر با ۱ خواهد بود. پس هر چقدر این توابع را با هم ترکیب کنیم، تابع ساخته‌شده نیز این خاصیت را خواهد داشت. (چرا؟)

این خاصیت، روی ترکیب توابع ناوردا است. اگر که همه توابع مجموعه در این خاصیت صدق کنند، هر ترکیبی از این توابع نیز در این خاصیت صدق می‌کند. در این بخش، ۵ خاصیت معرفی خواهیم کرد که نسبت به ترکیب توابع دارای آن خاصیت، بسته هستند.

۱. حفظ یک. اگر تمام ورودی‌های تابع برابر با ۱ باشد، خروجی آن نیز ۱ شود.

۲. حفظ صفر. اگر تمام ورودی‌های تابع برابر با ۰ باشد، خروجی آن نیز ۰ شود.

۳. صعودی بودن. به ازای هر وضعیتی از ورودی‌های تابع، اگر که یک ۰ را در میان ورودی‌ها به ۱ تبدیل کنیم، خروجی کم نشود؛ یعنی اگر خروجی ۱ بود به ۰ تبدیل نشود.

۴. خود-دوگان. به ازای هر حالت ورودی‌های تابع، اگر تمام ورودی‌ها را نقیض کنیم (اگر ۰ بود ۱ کنیم و اگر ۱ بود ۰ کنیم)، خروجی تابع نیز نقیض شود.

۵. خطی بودن. اگر تابع در یکی از شرایط زیر صدق کند، به آن خطی گفته می‌شود:

• تابع، برابر XOR تعدادی از متغیرهایش باشد (این تعداد می‌تواند ۰ باشد).

$$f(x_1, x_2, \dots, x_n) = XOR(x_{i_1}, x_{i_2}, \dots, x_{i_m}) \quad (14)$$

• تابع، برابر نقیض XOR تعدادی از متغیرهایش باشد (این تعداد می‌تواند ۰ باشد).

$$f(x_1, x_2, \dots, x_n) = XOR(x_{i_1}, x_{i_2}, \dots, x_{i_m}, 1) \quad (15)$$

ساخت همه توابع منطقی با تعدادی تابع

ابتدا با یک مسئله ساده شروع می‌کنیم. مسئله ۵. فرض کنید تابع منطقی f عضو کلاس توابع حفظ‌کننده یک، حفظ‌کننده صفر و خود-دوگان نباشد. ثابت کنید این تابع منطق کامل است.

اثبات. برای اثبات منطق کامل بودن یک تابع، کافیست مجموعه‌ای از توابع را با آن بسازیم که می‌دانیم منطق کامل هستند. برای مثال اگر بتوانیم NOR یا NAND را بسازیم، اثبات کرده‌ایم که تابع ما نیز منطق کامل است.

می‌دانیم که تابع ما خود-دوگان نیست. این گزاره یعنی ورودی‌هایی وجود دارند که اگر همه آن‌ها را نقیض کنیم، تابع نقیض نشود.

$$(16)$$

$$\exists x_1, \dots, x_n \in \{0,1\}; g(x_1, \dots, x_n) = g(\neg x_1, \dots, \neg x_n)$$

دقت کنید که چون تابع f حفظ‌کننده صفر و حفظ‌کننده یک نبود، پس این ورودی،

ورودی تمام صفر یا تمام یک نیست. حالا به تابع مان به این نحو ورودی دهید: برای ورودی i ام، اگر x_i برابر با یک بود a ، و در غیر این صورت b را قرار دهید. تابع حاصل یک تابع برحسب a و b مانند $g(a, b)$ است. حالا طبق تعریف، مقادیر g به ازای ورودی‌های مختلف را بررسی می‌کنیم:

۱. $g(0, 0) = f(0, \dots, 0) = 1$
۲. $g(1, 1) = f(1, \dots, 1) = 0$
۳. $g(1, 0) = g(0, 1)$

حالا اگر $g(1, 0) = g(0, 1) = 0$ باشد، در واقع تابع g همان NOR است و در غیر این صورت همان NAND است! پس ما با استفاده از این تابع یک مجموعه از توابع منطق کامل ساختیم، در نتیجه خود f منطق کامل است.

حالا می‌خواهیم به مسئله اصلی این بخش بپردازیم:

مسئله ۶. اگر A مجموعه‌ای از توابع منطقی باشد، و به ازای هر کدام از کلاس‌های توابع حفظ‌کننده صفر، حفظ‌کننده یک، صعودی، خود-دوگان و خطی، تابعی در A وجود داشته باشد که عضو آن کلاس نباشد، آن‌گاه مجموعه A منطق کامل است.

زیرمسئله ۱: اگر A مجموعه‌ای از توابع منطقی باشد و به ازای هر کدام از کلاس‌های توابع حفظ‌کننده صفر، حفظ‌کننده یک، صعودی و خود-دوگان، تابعی در A وجود داشته باشد که عضو آن کلاس نباشد، آن‌گاه با این توابع می‌توان صفر، یک و NOT را ساخت.

حالت ۱:

تابعی مانند f در A وجود داشته باشد که حفظ‌کننده صفر و حفظ‌کننده یک نباشد.

ساخت NOT: چون تابع f حفظ‌کننده صفر و حفظ‌کننده یک نیست، داریم:

$$\left. \begin{aligned} f(0, \dots, 0) &= 1 \\ f(1, \dots, 1) &= 0 \end{aligned} \right\} \Rightarrow \forall a \in \{0, 1\}; f(a, \dots, a) = \neg a$$

ساخت صفر و یک: از طرفی تابعی مانند g در A هست که خود-دوگان نباشد، یعنی:

$$\exists x_1, \dots, x_n \in \{0, 1\}; g(x_1, \dots, x_n) = g(\neg x_1, \dots, \neg x_n)$$

حالا به تابع مان به این نحو ورودی دهید: برای ورودی i ام، اگر x_i برابر با یک بود a ، و در غیر این صورت $\neg a$ را قرار دهید. تابع حاصل یک تابع برحسب a مانند $h(a)$ است. طبق تعریف $h(a) = h(1) = h(0)$. حالا اگر $h(a) = 1$ باشد، آن‌گاه یک را ساختیم و با استفاده از تابع NOT که ساختیم، صفر هم داریم. و اگر $h(a) = 0$ باشد، آن‌گاه صفر را ساختیم و با استفاده از تابع NOT که ساختیم، یک هم داریم.

حالت ۲:

دو تابع مختلف مانند f و g در A وجود داشته باشند که f حفظ‌کننده صفر نباشد و g حفظ‌کننده یک نباشد، ولی حفظ‌کننده صفر باشد.

ساخت صفر و یک: طبق فرض $1 = f(a, \dots, a)$ و $0 = g(a, \dots, a)$ که یعنی صفر و یک ساخته شده‌اند.

ساخت NOT: از طرفی تابعی مانند h در A وجود دارد که صعودی نیست. یعنی به ازای یک ورودی، برابر یک است، ولی اگر یکی از ورودی‌های صفر آن را به یک تبدیل کنیم خروجی تابع برابر با صفر می‌شود (بدون

کاهش کلیت مسئله فرض می‌کنیم این اتفاق برای ورودی اول می‌افتد). یعنی:

$$h(0, x_2, \dots, x_n) = 1 \wedge h(1, x_2, \dots, x_n) = 0 \quad (19)$$

حالا که صفر و یک را ساخته‌ایم، می‌توانیم به تابع مان همان x_i های خاص را ورودی دهیم. تابع زیر را در نظر بگیرید:

$$k(a) = h(a, x_2, \dots, x_n) \quad (20)$$

طبق تعریف داریم:

$$k(0) = h(0, x_2, \dots, x_n) = 1 \quad (21)$$

$$k(1) = h(1, x_2, \dots, x_n) = 0$$

که یعنی این همان تابع NOT است.

زیرمسئله ۲: اگر تابع f خطی نباشد، مجموعه $\{NOT, f, 1, 0\}$ منطق کامل است. ابتدا یک تعریف معادل برای خطی بودن یک تابع می‌آوریم، اما به اثبات آن نمی‌پردازیم.

قضیه ۲. تابع منطقی f خطی است اگر و تنها اگر ورودی‌های آن به دو دسته افراز شوند. ورودی‌هایی که مستقل از بقیه ورودی‌ها، تأثیری در خروجی نمی‌گذارند و ورودی‌هایی که مستقل از بقیه ورودی‌ها، نقیض‌کردنشان منجر به نقیض شدن خروجی تابع می‌شود. حالا تابع f مطابق فرض خطی نیست. این یعنی یکی از ورودی‌های f (بدون کاستی از کلیت مسئله، در نظر می‌گیریم ورودی اول) هست که نقیض کردن آن به‌ازای رشته‌ای از سایر ورودی‌ها تأثیری در خروجی تابع نداشته باشد و به‌ازای رشته‌ای دیگر، نقیض کردن آن خروجی تابع را نیز نقیض کند، یعنی:

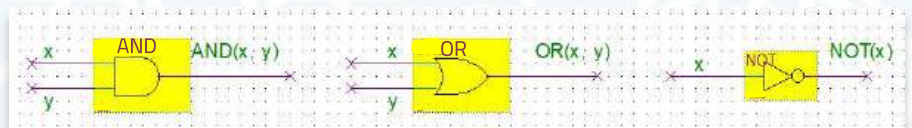
(۲۲)

$$\begin{aligned} \exists (x_1, \dots, x_n); f(a, x_1, \dots, x_n) &= f(\neg a, x_1, \dots, x_n) \wedge \\ \exists (y_1, \dots, y_n); f(a, y_1, \dots, y_n) &= \neg f(\neg a, y_1, \dots, y_n) \end{aligned}$$

حالا تابع $g(a, b)$ را به این صورت تعریف می‌کنیم: a را به ورودی اول f بدهید. برای ورودی i ام f ، اگر $x_i = y_i$ ، خود همان عدد را ورودی می‌دهیم (چون به صفر و یک دسترسی داریم). اگر $x_i \neq y_i$ ، آن‌گاه b را به عنوان ورودی می‌دهیم. و اگر $x_i \neq y_i$ ، آن‌گاه $\neg b$ را ورودی می‌دهیم (چون به NOT دسترسی داریم).

طبق این تعریف $g(0, 1) = g(1, 1) = g(1, 0) \neq g(0, 0)$ این یعنی جداول درستی این تابع در سه تا از ستون‌ها یکسان و در دیگری متفاوت است که یعنی تابع g به یکی از صورت‌های $a+b, a+\neg b, \neg a+b, \neg a+\neg b, a \cdot b, a \cdot \neg b, \neg a \cdot b, \neg a \cdot \neg b$ است. با استفاده مناسب از تابع NOT در ورودی‌های این تابع می‌توان این تابع را به یکی از توابع $a+b$ یا $a \cdot b$ تبدیل کرد. چون تابع NOT هم در اختیار داریم، با توابع مان یک مجموعه منطق کامل ساخته‌ایم.

شکل ۱

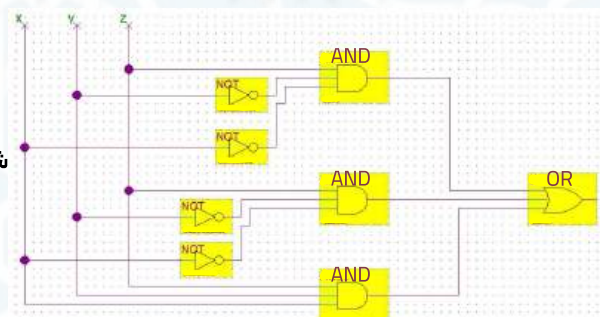


x	y	AND(x, y)
0	0	0
0	1	0
1	0	0
1	1	1

x	y	OR(x, y)
0	0	0
0	1	1
1	0	1
1	1	1

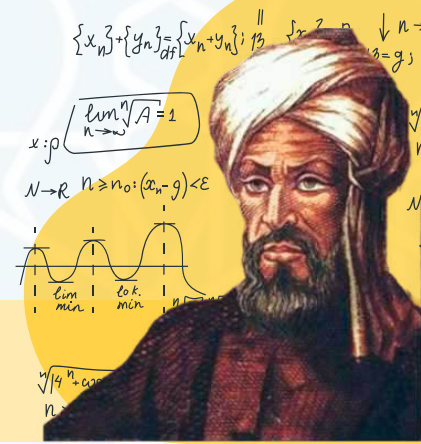
x	NOT(x)
0	1
1	0

شکل ۲



x	y	z	f(x, y, z)
0	0	0	0
0	0	1	1 $\rightarrow z\bar{y}\bar{x}$
0	1	1	1 $\rightarrow \bar{z}y\bar{x}$
0	1	0	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1 $\rightarrow zy\bar{x}$

$$f(x, y, z) = z\bar{y}\bar{x} + \bar{z}y\bar{x} + zy\bar{x}$$

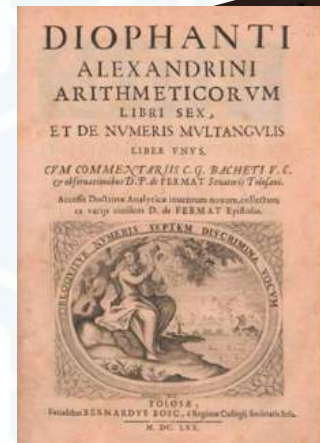


الگوریتم یا خوارزمی

محمد بن موسی خوارزمی، ریاضی‌دان، ستاره‌شناس، فیلسوف، جغرافی‌دان و مورخ شهیر ایرانی در قرن هشتم و نهم میلادی است.

او در زمینه جبر، کتابی با عنوان «حساب الجبر و المقابله» نوشت که در آن برای اولین بار حل نظام‌مند معادلات خطی و درجه دوم ارائه شد. خوارزمی از روش‌های جبری و هندسی برای حل شش نوع معادله استاندارد استفاده کرد. این اثر را می‌توان مشهورترین و مهم‌ترین اثر خوارزمی دانست. این کتاب در حقیقت اولین کتاب نوشته‌شده در زمینه جبر در تاریخ است. در نتیجه، خوارزمی با تگارش این کتاب لقب «پدر جبر» را از آن خود کرد.

کتاب دیگر او که در مورد انجام محاسبات با اعداد هندو است، در حدود سال ۸۲۵ میلادی نوشته شد. این کتاب را می‌توان اولین کتاب سیستم شماره‌گذاری هند در خاورمیانه و سپس اروپا دانست که در قرن دوازدهم با نام «Algoritmi de numero Indorum» به لاتین ترجمه شد. در این آثار، در نگاه اول ردپای چندانی از ریاضیات پیشرفته مشاهده نمی‌شود، اما با نگاهی عمیق‌تر درمی‌یابیم که مفاهیم مطرح‌شده بسیار انتزاعی و عمیق هستند. او در آثار ریاضیاتی خود مفهوم اعداد طبیعی را معرفی کرد که این مفاهیم همچنان در علم ریاضیات پرکاربرد هستند.



اعداد هندو-عربی برای تاریخ محاسبات مهم هستند، زیرا از سیستم اعشاری با پایه ده استفاده می‌کنند. این سیستم عددی زیربنای فناوری محاسبات مدرن است و انقلابی در ریاضیات بود که هرگونه اعمال محاسباتی پیشرفته را مقدور ساخت. نوشته‌های او نه تنها به پیشرفت ریاضیات کمک شایانی کرد، بلکه بر پیشرفت زبان نیز تأثیر گذاشت.



خوارزمی برای حل سوالات، مفهوم الگوریتم‌ها را پایه‌گذاری کرد. وی اولین توضیحات مفصل را برای استفاده از نماد اعشاری برای انجام عمل اساسی (جمع، تفریق، ضرب و تقسیم) و محاسبات اعشاری را ارائه داد. آن چه که زندگی پربار خوارزمی را در میان دیگران برجسته می‌سازد، شیوه وی برای گسترش هر دانشی بر پایه سنجش، گزینش و ترکیب همه یافته‌های موجود در یک ساختار قابل گسترش است؛ تفکر نظام‌مند او برای حل مسائل از طریق تریبی از مراحل، مفهوم الگوریتم را ایجاد کرد.

واژه «الگوریتم» نیز از نوشته‌های او سرچشمه می‌گیرد که درحقیقت ترجمه کلمه «algarismo» در زبان پرتغالی است و برگرفته از تلفظ لاتین اسم الخوارزمی است. در واقع، الگوریتم به قوانین مورد استفاده برای انجام محاسبات با استفاده از سیستم عددی هندو-عربی اشاره دارد.

در نتیجه، علاوه بر «پدر جبر»، از او به عنوان «پدر بزرگ علوم کامپیوتر» نیز یاد می‌شود.

از آن جایی که علم در قرون وسطی در اروپا در خفقان بود و حتی کلیسا استفاده از اعداد غیر یونانی را ممنوع کرده بود، ترجمه‌های علمی او از علم حساب و جبر، چند قرن بعد از درگذشت او به اروپا رسید و سیستم اعداد اعشاری و جبر را به آنان معرفی کرد. آثار او در ریاضیات، منبع اصلی دانش ریاضی قرن‌های آینده در شرق و غرب بود.

اواخر قرن نوزدهم بود که الگوریتم به معنای مجموعه‌ای از قوانین گام‌به‌گام برای حل یک مسئله معرفی شد. در قرن بیستم آلن تورینگ، ریاضیدان و دانشمند کامپیوتر بریتانیایی، از نظر تئوری بررسی کرد که چگونه یک ماشین می‌تواند دستورالعمل‌های الگوریتمی را دنبال کرده و ریاضیات پیچیده را حل کند که پس از خوارزمی یک نقطه عطف تاریخی برای علوم کامپیوتر به حساب می‌آید.



از شمارنده تا زمان بندی؛ بررسی OTP های مبتنی بر زمان و شمارنده

امروزه، همه ما در دنیای دیجیتال نیاز داریم که امنیت حساب های کاربری مان تضمین شود. شرکت های مختلف، هر یک راهکارهای خاص خود را برای تأمین این امنیت در نظر گرفته اند، اما چیزی که بیشتر شرکت ها بر سر آن توافق دارند، استفاده از احراز هویت دوعاملی (2FA - Two-Factor Authentication) است که به خط دفاع اصلی ما در برابر هکرها و افرادی تبدیل شده است که رمز عبور حساب های ما را در اختیار دارند. احراز هویت دوعاملی روش های مختلفی دارد؛ برای مثال، تلگرام از یک گذرواژه ثابت برای این منظور استفاده می کند. اما رایج ترین روش های احراز هویت دوعاملی، استفاده از خانواده رمز یک بار مصرف (OTP - One-Time Password) است؛ یعنی رمزهای کوتاهی که تنها یک بار و در بازه زمانی محدودی قابل استفاده هستند. در این متن قرار است با سه عضو معروف این خانواده آشنا شویم و مزایا و معایب هر یک را بررسی کنیم.

الگوریتم HMAC-Based HOTP: One-Time Password

این الگوریتم بر مبنای HMAC و یک شمارنده مشترک بین کلاینت و سرور Authenticator طراحی شده است. کلمه HMAC مخفف Hash-based Message Authentication Code است که یک الگوریتم رمزنگاری برای تضمین اصالت داده ها هنگام انتقال آن ها به شمار می رود. بنابراین، زمانی که از HMAC استفاده می کنیم، مطمئن هستیم که داده ها واقعاً از سوی فرستنده اصلی ارسال شده اند و در مسیر انتقال کسی آن ها را تغییر نداده است (Integrity). به طور کلی، HMAC از دو عنصر استفاده می کند: یک کلید مخفی که بین طرفین تعیین شده و یک تابع هش مانند SHA-256. الگوریتم HMAC به این صورت عمل می کند که ابتدا داده را با یک کلید داخلی که از کلید مخفی اولیه مشتق شده ترکیب می کند و سپس آن را Hash می کند. پس از آن، نتیجه به دست آمده را با یک کلید داخلی دیگر، که آن هم از کلید مخفی اولیه مشتق شده، ترکیب کرده و مجدداً Hash می کند و نتیجه نهایی را برمی گرداند. این نتیجه نهایی همان HMAC ماست! اما دلیل استفاده از این ساختار Hash چیست؟ این ساختار باعث می شود حتی اگر کسی تابع Hash ما را شناسایی کند، بدون داشتن کلید مخفی نتواند HMAC را جعل کند.

باید بدانید که HMAC بین کلاینت و سرور هنگام ثبت نام تبادل می شود و در سیستم کلاینت ذخیره می شود.

همچنین، به ازای هر کاربر، در سرور و سیستم کاربر یک شمارنده وجود دارد که با هر بار تولید OTP، این شمارنده یک عدد افزایش می یابد. نحوه کارکرد این الگوریتم به این صورت است که اپلیکیشن سمت کلاینت و سرور، با استفاده از شمارنده (C) و کد مخفی (K) یک هش SHA تولید می کند و آن را به یک تابع Truncate می سپارد. تابع Truncate نیز بر اساس قرارداد، شش یا پنج رقم آخر آن را به عنوان OTP شناسایی کرده و بازمی گرداند. همچنین، با هر درخواست، شمارنده (C) یک عدد افزایش می یابد.

تولید OTP در این روش کاملاً آفلاین است و هیچ نیازی به اینترنت یا همگام سازی زمانی ندارد که این موضوع یک مزیت محسوب می شود. با این حال، ناهمگام شدن شمارنده در این روش یکی از مشکلات اساسی به شمار می رود. فرض کنید کلاینت چندین بار رمز تولید کند اما از آن ها استفاده نکند. در این حالت، شمارنده کلاینت از شمارنده سرور جلوتر خواهد بود و دیگر نمی توان از OTP ها استفاده کرد. در چنین وضعیتی، نیاز است شمارنده کلاینت و سرور مجدداً با هم همگام شوند؛ البته این کار همچنان به دلیل وجود کلید مخفی، امن است، اما تجربه کاربری مناسبی ارائه نمی دهد.



الگوریتم TOTP: Time-Based One-Time Password

قطعاً تا به حال برایتان پیش آمده است که قصد ورود به حساب کاربری گیت هاب خود را داشته باشید و پس از وارد کردن رمز عبور، با پیغامی جدید مواجه شوید که از شما می خواهد برای حساب خود احراز هویت دوعاملی (2FA) را فعال کنید.

ناچار می شوید یک نرم افزار Authenticator روی سیستم یا موبایل خود نصب کنید و سپس یک کد QR که گیت هاب به شما داده است را اسکن نمایید. پس از انجام این کار، OTP مربوط به گیت هاب نیز به لیست OTP های آن برنامه Authenticator اضافه می شود.

در دفعات بعدی که قصد ورود به حساب گیت هاب خود را دارید، مجبور خواهید بود یک کد OTP از داخل آن برنامه بردارید و در سایت گیت هاب وارد کنید. شاید تا به حال به این نکته توجه نکرده باشید، اما می توانید کدهای OTP موجود در برنامه Authenticator را بدون اتصال به اینترنت یا دریافت ایمیل و پیامک دریافت کنید! گیت هاب برای سیستم 2FA خود از الگوریتم TOTP استفاده می کند، که در ادامه با آن آشنا خواهیم شد.

این الگوریتم نسخه بهبودیافته ای از HOTP است که مشکل ناهمگام شدن شمارنده را با اضافه کردن پارامتر زمان حل کرده است. مانند الگوریتم قبلی، در اینجا هم یک کلید مخفی داریم که هنگام ثبت نام بین کلاینت و سرور تبادل می شود (همان QR Code که گیت هاب به شما ارائه می دهد تا آن را اسکن کنید). در این روش، OTP بر اساس کلید مخفی و تعداد بازه های زمانی از یک زمان مبدأ مشترک تولید می شود. رمزها در بازه های زمانی ۳۰ ثانیه ای معتبر هستند و پس از آن دیگر اعتبار ندارند. تولید شمارنده الگوریتم HMAC در این روش به این صورت است که زمان فعلی یونیکس را بر یک عدد ثابت از پی تعیین شده که نشان دهنده طول عمر OTP است، تقسیم می کنیم.

زمان فعلی یونیکس (Unix Timestamp) به عددی گفته می شود که نشان می دهد از اول ژانویه سال ۱۹۷۰ میلادی (معروف به Epoch) چند ثانیه گذشته است! این زمان سنجی در دنیای کامپیوتر بسیار پرکاربرد است و یک روش ساده و دقیق برای ذخیره و مقایسه زمان در برنامه های مختلف به شمار می رود.

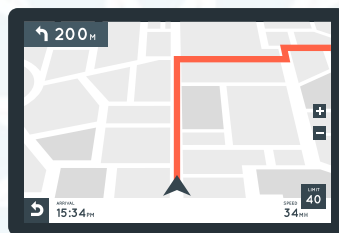
استفاده از Unix Timestamp، پیچیدگی همگام سازی زمان بین کلاینت و سرور را نیز از بین می برد و دیگر نیازی به همگام کردن زمان کلاینت و سرور نیست. باقی فرآیند تولید OTP کاملاً مشابه با الگوریتم HMAC است و می توان به صورت کاملاً آفلاین OTP تولید کرد.

دانستنی‌ها (به شیوه قدیمی)



اولین ویروس، بی‌خطر بود! در سال ۱۹۷۱، اولین ویروس رایانه‌ای جهان ساخته شد. این ویروس که Creeper داشت، صرفاً به عنوان یک آزمایش طراحی شده بود تا ببینند این ویروس چگونه بین رایانه‌ها پخش می‌شود. عملکردش فقط این بود که پیامی نمایش دهد: «من Creeper هستم، اگر می‌تونی منو بگیر!» اولین ویروسی که واقعاً در فضای آزاد منتشر شد، ویروسی به نام Brain بود. این ویروس هم آسیبی نداشت، اما اولین ویروس مخصوص رایانه‌های IBM PC به حساب می‌آمد.

تعداد خطوط کد در یک خودروی معمولی از فیسبوک نیز بیشتر است. امروزه خودروها به‌طور فزاینده‌ای به فناوری وابسته شده‌اند، از سامانه‌های سرگرمی گرفته تا قابلیت‌های رانندگی خودکار، خودروها با فناوری‌های پیشرفته‌ای به بازار عرضه می‌شوند. بازار خودرو امروزه بیش از هر زمانی به فناوری وابسته شده است.



CAPTCHA مخففی طولانی است. این واژه مخفف عبارت: "Completely Automatic Public Turing test to Tell Computers and Humans Apart" به معنای: «آزمون عمومی کاملاً خودکار تورینگ برای تشخیص انسان از رایانه» است.

در حال حاضر، شرکت‌های متعددی مانند Github، Microsoft و دیگران از این روش برای پیاده‌سازی 2FA استفاده می‌کنند.



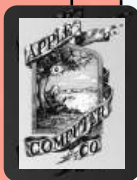
استفاده از Push-OTP و SMS-OTP و Email-OTP

معروف‌ترین و قدیمی‌ترین روش استفاده از OTP، همین نوع است که همه ما روزمره از آن استفاده می‌کنیم. این روش به این صورت عمل می‌کند که یک کد یک‌بارمصرف در سمت سرور تولید شده و از طریق ایمیل، پیامک (SMS) یا Push-Notification به دست کاربر می‌رسد و می‌توان از آن به عنوان احراز هویت دو عاملی (2FA) استفاده کرد. از معایب این روش، نیاز به اتصال به اینترنت یا شبکه تلفن همراه است.

این روزها OTP دیگر فقط محدود به TOTP و HOTP نیست، بلکه روش‌های جدیدتری معرفی شده‌اند که هم امنیت بالاتری دارند و هم استفاده از آن‌ها راحت‌تر است. مثلاً کاری که جیمیل در Push-based OTP انجام داده این است که به جای تایپ کردن یک کد شش‌رقمی، فقط کافی است یک اعلان را روی گوشی خود تأیید کنید. Biometric OTP یک قدم جلوتر رفته و از چهره یا اثر انگشت فرد برای ساخت یک رمز مخصوص استفاده می‌کند که جعل آن بسیار سخت است.

استفاده از QR Code OTP نیز یکی از روش‌های جدید است که تلگرام برای ورود به حساب‌های کاربری از آن بهره می‌برد. این روش به این صورت عمل می‌کند که یک QR Code را با اپلیکیشن مخصوص اسکن می‌کنید و رمز یک‌بارمصرف همان‌جا تولید می‌شود؛ این روش در ورود به تلویزیون‌های هوشمند نیز بسیار کاربرد دارد. در کنار همه این‌ها، OTP‌هایی وجود دارند که مبتنی بر کلیدهای سخت‌افزاری هستند. به عنوان مثال می‌توان به YubiKey اشاره کرد که یک کلید سخت‌افزاری بسیار معروف و ایمن است.





نخستین لوگوی شرکت اپل چیزی نبود که انتظار دارید. در ابتدا، تصویر آیزاک نیوتن را نشان می‌داد که زیر درختی

نشسته بود و سیبی در آستانه افتادن روی سرش قرار داشت.

این لوگو در سال ۱۹۷۶ طراحی شد و عبارتی در حاشیه آن نوشته شده بود: «نیوتن... ذهنی که تا ابد در دریاهای عجیب اندیشه سفر می‌کند... تنها»



چیدمان QWERTY صفحه‌کلید، در ابتدا برای کاهش سرعت تایپ طراحی شده بود.

در دوران ابتدایی ماشین‌تحریر، تایپ سریع باعث گیر کردن دکمه‌ها می‌شد. چیدمان QWERTY با فاصله دادن میان حروف پرکاربرد، سرعت تایپ را کاهش می‌داد تا از گیر کردن کلیدها جلوگیری شود.

اگر به دنبال یادگیری چیدمانی سریع‌تر هستید، صفحه‌کلید Dvorak برای سرعت بالا طراحی شده است.



نشان مرورگر فایرفاکس در واقع تصویر روباه نیست! چون نام این روباه باشد، تصویر موزیلاست!

تا حدی هم درست است. بسیاری تصور می‌کنند مرورگر «Firefox» است، تصویر لوگو نیز باید این تصور هم درست و هم نادرست است؛ و البته در حقیقت، حیوان پشمالوی موجود در لوگو در ابتدا یک پاندا قرمز بود. اما با گذر زمان، تیم سازنده مرورگر آن را «روباه» نامیدند.

دولت ایالات متحده از پلی‌استیشن ۳ استفاده می‌کرد اما نه برای بازی. در سال ۲۰۰۵، نیروی هوایی ایالات متحده از ۱۷۶۰ دستگاه پلی‌استیشن ۳ برای ساخت یک آبرایانه برای وزارت دفاع استفاده کرد. دلیل این انتخاب، صرفه جویی در هزینه، انرژی و سازگاری بیشتر آن با محیط زیست بود.

تا سال ۲۰۰۵، کبوترهای نامه‌بر از اینترنت سریع‌تر بودند!

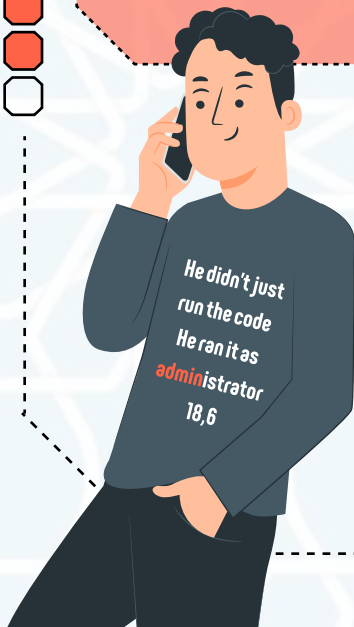
در یک آزمایش برای مقایسه سرعت آپلود، یک کبوتر نامه‌بر با یک فلش USB حاوی ۴ گیگابایت اطلاعات، مسافت ۵۰ مایل (۸۰ کیلومتر) پرواز کرد تا اطلاعات را به شرکت ارائه‌دهنده اینترنت برساند و همان فایل از طریق اینترنت نیز ارسال شد. کبوتر در یک ساعت به مقصد رسید، اما آپلود بیش از دو ساعت طول کشید!



در تمامی تبلیغات آیفون، ساعت ۹:۴۱ نمایش داده می‌شود. علت این موضوع آن است که در سال ۲۰۰۷، استیو جابز برای نخستین بار آیفون را در همین ساعت معرفی کرد. ارائه او دقیقاً ۴۱ دقیقه به طول انجامید و دقیقاً در ساعت ۹:۴۱ صبح، نخستین تصویر آیفون نمایش داده شد. از آن زمان، اغلب تصاویر تبلیغاتی آیفون با ساعت ۹:۴۱ همراه هستند.

بیشتر شرکت‌های موفق امروزی فعالیت خود را از یک گاراژ آغاز کرده‌اند.

جالب است بدانید که افزون بر شرکت اپل، شرکت‌های بزرگ دیگری مانند HP، گوگل و مایکروسافت نیز در ابتدا در یک گاراژ ساده تأسیس شده‌اند.



سیستم‌های رایافیزیکی: معماری، چالش‌ها و مسیر تحول در صنایع هوشمند

امیرحسین انصاری

کارشناسی ارشد ۱۴۰۴

مقدمه

سیستم‌های رایافیزیکی^۱ (CPS) به عنوان یکی از پیشرفته‌ترین و تأثیرگذارترین فناوری‌های نوظهور در مهندسی دیجیتال شناخته می‌شوند که نقش کلیدی در ادغام دنیای فیزیکی و سایبری ایفا می‌کنند. این سیستم‌ها ترکیبی از حسگرها، پردازنده‌ها، عملگرها، شبکه‌های ارتباطی و الگوریتم‌های هوشمند هستند که با استفاده از تحلیل داده‌های بی‌درنگ، کنترل دقیق و تصمیم‌گیری خودکار در محیط‌های پیچیده و پویا امکان‌پذیر می‌سازند. تعامل هم‌زمان بین اجزای فیزیکی و محاسباتی در این سیستم‌ها، باعث شده‌است که CPS از مدل‌های سنتی کنترل و نظارت فراتر برود و بستری را برای بهینه‌سازی هوشمند فرآیندها در مقیاس گسترده فراهم کند. ظهور CPS نتیجه پیشرفت‌های موازی در علوم محاسباتی، سیستم‌های توزیع‌شده، ارتباطات بی‌سیم، هوش مصنوعی و حسگرهای پیشرفته است. این سیستم‌ها قادرند مجموعه وسیعی از داده‌های محیطی را جمع‌آوری و تحلیل کنند، فرآیندهای پیچیده را مدل‌سازی کنند و واکنش‌های بی‌درنگ و هوشمندانه ارائه دهند. چنین قابلیتی، CPS را به یک پارادایم نوین در مهندسی سامانه‌های هوشمند و دیجیتال‌سازی صنایع تبدیل کرده‌است. از این رو، سیستم‌های رایافیزیکی در صنایعی مانند حمل‌ونقل هوشمند، پزشکی دیجیتال، اتوماسیون صنعتی، کشاورزی هوشمند و مدیریت انرژی جایگاه ویژه‌ای پیدا کرده‌اند و موجب افزایش بهره‌وری، کاهش هزینه‌ها و ارتقاء کیفیت خدمات در این حوزه‌ها شده‌اند. یکی از ویژگی‌های برجسته CPS، توانایی تعامل پویا و تطبیق‌پذیر با محیط‌های متغیر و غیرقابل پیش‌بینی است. برخلاف سیستم‌های سنتی که بر پایه الگوریتم‌های ایستا و ازپیش‌تعریف‌شده کار می‌کنند، سیستم‌های رایافیزیکی قادرند به صورت خودمختار، بر اساس داده‌های جمع‌آوری‌شده از محیط، تغییرات لازم را در تصمیمات کنترلی اعمال کنند. این قابلیت نه تنها باعث کاهش خطاها و افزایش دقت عملکرد سیستم‌ها می‌شود، بلکه زمینه را برای خودکارسازی پیشرفته و توسعه فناوری‌های نوین در حوزه‌های هوش مصنوعی و یادگیری ماشین فراهم می‌کند.

- که برای پایش وضعیت محیط‌های صنعتی، شهری و پزشکی استفاده می‌شوند.
- سیستم‌های بینایی کامپیوتری: دوربین‌های نظارتی، حسگرهای LiDAR و رادار؛ که در سیستم‌های حمل‌ونقل هوشمند، خودروان‌ها و نظارت تصویری پیشرفته کاربرد دارند.
- GPS و سیستم‌های موقعیت‌یابی: برای مسیریابی در خودروهای خودران، رهگیری اشیاء و مدیریت ناوگان استفاده می‌شود.
- فناوری RFID و NFC: در لجستیک، زنجیره تأمین و کنترل دسترسی هوشمند برای شناسایی خودکار و ردیابی اشیاء به کار می‌روند.

چالش‌های این لایه:

- انرژی مصرفی بالا در حسگرهای توزیع‌شده که نیازمند طراحی حسگرهای کم‌مصرف و فناوری‌های برداشت انرژی^۳ است.
- دقت و صحت داده‌های جمع‌آوری‌شده، که با توسعه حسگرهای پیشرفته و استفاده از روش‌های کالیبراسیون بهبود می‌یابد.
- هماهنگ‌سازی حسگرهای ناهمگون، که نیازمند استانداردهای یکپارچه‌سازی داده‌ها و پروتکل‌های ارتباطی کارآمد است.

لایه پردازشی^۴

این لایه نقش حیاتی در تبدیل داده‌های خام به اطلاعات معنادار دارد و بر تحلیل، مدل‌سازی و پیش‌بینی‌های هوشمندانه تمرکز دارد. سرعت و دقت پردازش در این بخش، تأثیر مستقیم بر عملکرد کل سیستم دارد. اجزای کلیدی این لایه:

- پردازنده‌های نهفته^۵: پردازنده‌هایی مانند ARM Cortex-M، FPGA و DSP که به عنوان واحدهای محاسباتی کم‌مصرف در سیستم‌های بی‌درنگ برای تحلیل داده‌های محلی به کار می‌روند.
- سرورها و محاسبات ابری^۶: پردازش داده‌های حجیم و اجرای الگوریتم‌های هوش مصنوعی، یادگیری ماشین و مدل‌سازی پیچیده در محیط‌های ابری یا

با توجه به افزایش پیچیدگی و مقیاس سیستم‌های مدرن، توسعه و پیاده‌سازی سیستم‌های رایافیزیکی با چالش‌های متعددی همراه است. مسائلی مانند امنیت سایبری، انرژی مصرفی، قابلیت اطمینان، یکپارچه‌سازی با سیستم‌های قدیمی و کاهش تأخیر در پردازش داده‌ها از جمله چالش‌های اصلی این حوزه محسوب می‌شوند. به همین دلیل، پژوهش‌های گسترده‌ای در زمینه بهبود الگوریتم‌های پردازشی، توسعه معماری‌های امن، بهینه‌سازی مصرف منابع و طراحی پروتکل‌های ارتباطی پیشرفته در حال انجام است تا سیستم‌های رایافیزیکی را بیش از پیش کارآمد، هوشمند و ایمن سازند.

ساختار و معماری سیستم‌های رایافیزیکی

سیستم‌های رایافیزیکی دارای معماری چندلایه و توزیع‌شده هستند که هدف اصلی آن ایجاد تعامل یکپارچه بین دنیای فیزیکی و سایبری است. این سیستم‌ها از اجزای مختلفی همچون حسگرها، پردازنده‌های توزیع‌شده، شبکه‌های ارتباطی پیشرفته و عملگرهای هوشمند تشکیل شده‌اند که به صورت هماهنگ کار می‌کنند تا داده‌های محیطی را جمع‌آوری، پردازش و تحلیل کنند و در نهایت، تصمیمات هوشمند را به اقدامات فیزیکی تبدیل کنند. طراحی این معماری باید به گونه‌ای باشد که ویژگی‌هایی مانند تأخیر کم، مصرف بهینه انرژی، امنیت سایبری و قابلیت اطمینان بالا را تضمین کند. در ادامه، لایه‌های اصلی معماری CPS مورد بررسی دقیق قرار می‌گیرند.

لایه حسگری^۲

این لایه نقطه ورود اطلاعات به سیستم رایافیزیکی است و وظیفه جمع‌آوری داده‌های بی‌درنگ از محیط فیزیکی را بر عهده دارد. داده‌های دریافتی از طریق انواع حسگرها، دوربین‌های هوشمند، سیستم‌های GPS و RFID و... به دست آمده و سپس به لایه پردازشی برای تحلیل و تصمیم‌گیری ارسال می‌شوند. اجزای کلیدی این لایه:

- حسگرهای محیطی: حسگرهایی نظیر حسگر دما، رطوبت، فشار، لرزش، نور و گاز

توزیع شده انجام می‌شود.

- محاسبات لبه‌ای^۲: پردازش داده‌ها در نزدیک‌ترین نقطه به حسگرها، که باعث کاهش تأخیر در تصمیم‌گیری و افزایش بهره‌وری سیستم می‌شود.
- الگوریتم‌های هوش مصنوعی و یادگیری ماشین: برای تشخیص الگوها، پیش‌بینی خرابی‌ها، کنترل تطبیقی و تصمیم‌گیری خودمختار در CPS به کار می‌روند.

چالش‌های این لایه:

- بهینه‌سازی انرژی مصرفی در پردازنده‌های توزیع‌شده، به دلیل محدودیت منابع در سیستم‌های تعبیه‌شده.
- مدیریت داده‌های حجیم^۸ که نیازمند روش‌های فشرده‌سازی و فیلترگذاری کارآمد است.
- یکپارچه‌سازی مدل‌های یادگیری ماشین با سیستم‌های بی‌درنگ، که نیاز به طراحی الگوریتم‌های سبک و بهینه دارد.

لایه ارتباطی^۹

این لایه نقش پل ارتباطی بین تمامی بخش‌های سیستم رایفیزیکی را ایفا کرده و انتقال داده‌ها بین حسگرها، پردازنده‌ها و عملگرها را تسهیل می‌کند. فناوری‌های ارتباطی کلیدی در این لایه:

- شبکه‌های بی‌سیم: Wi-Fi, Zigbee, LoRa, بلوتوث، 5G و 6G برای اتصال دستگاه‌های اینترنت اشیا و ارتباطات بی‌درنگ.
- شبکه‌های سیمی: Ethernet, CAN Bus, Modbus و پروتکل‌های صنعتی برای ارتباطات پایدار و سریع در محیط‌های صنعتی.
- پروتکل‌های امنیتی و رمزنگاری داده‌ها: مانند TLS, AES و پروتکل‌های مبتنی بر بلاکچین برای تضمین امنیت اطلاعات.

چالش‌های این لایه:

- کاهش تأخیر ارتباطی در سیستم‌های بی‌درنگ و صنعتی.
- افزایش ظرفیت و پهنای باند شبکه‌ها برای مدیریت حجم بالای داده‌های CPS.
- افزایش امنیت ارتباطات در برابر حملات سایبری و تهدیدات هکری.

لایه کنترل و عملیاتی^{۱۰}

این لایه نهایی‌ترین بخش سیستم رایفیزیکی است که پس از پردازش داده‌ها و تصمیم‌گیری،

خروجی‌ها را به صورت دستورات کنترلی به عملگرها و دستگاه‌های فیزیکی ارسال می‌کند. اجزای کلیدی این لایه:

- موتورهای و عملگرهای هوشمند، همچون سروو موتورها، پیزوالکتریک‌ها و درایورهای صنعتی که وظیفه اجرای دستورات کنترلی را دارند.
 - سیستم‌های کنترلی بی‌درنگ^{۱۱} مانند کنترل‌کننده‌های PID، شبکه‌های عصبی تطبیقی و کنترل پیش‌بین مدل (MPC) برای اجرای دستورات با دقت بالا.
 - رابط‌های انسان-ماشین^{۱۲} (HMI) شامل نمایشگرهای لمسی، داشبوردهای صنعتی و سیستم‌های واقعیت افزوده (AR)، که امکان نظارت و کنترل فرآیندها را برای کاربران فراهم می‌کند.
- چالش‌های این لایه:
- دقت و سرعت اجرای دستورات کنترلی در محیط‌های حساس به زمان.
 - هماهنگی بین سیستم‌های کنترلی و پردازشی برای جلوگیری از تأخیر و ناپایداری در عملکرد.

مدیریت خطاها و تشخیص ناهنجاری‌ها برای جلوگیری از خرابی‌های ناگهانی و افزایش ایمنی سیستم.

چالش‌های سیستم‌های رایفیزیکی (CPS)

این سیستم‌ها به دلیل ترکیب پیچیده زیرساخت‌های فیزیکی و سایبری با چالش‌های متعددی مواجه‌اند. این چالش‌ها می‌توانند عملکرد، امنیت، کارایی و قابلیت اطمینان این سیستم‌ها را تحت تأثیر قرار دهند. از آن‌جا که CPS در حوزه‌های حیاتی مانند حمل‌ونقل هوشمند، سلامت دیجیتال، شهرهای هوشمند و زیرساخت‌های صنعتی استفاده می‌شود، مدیریت این چالش‌ها برای اطمینان از عملکرد ایمن و پایدار این سیستم‌ها ضروری است. در ادامه، مهم‌ترین چالش‌های CPS و راهکارهای پیشنهادی برای مقابله با آن‌ها مورد بررسی قرار می‌گیرند.

امنیت و حریم خصوصی در CPS

یکی از بزرگ‌ترین چالش‌های CPS، امنیت و حریم خصوصی داده‌هاست. سیستم‌های رایفیزیکی به دلیل ارتباط مستقیم با زیرساخت‌های فیزیکی و استفاده از شبکه‌های بی‌سیم و اینترنت اشیا (IoT)،

به شدت در معرض حملات سایبری قرار دارند. حملات سایبری به CPS می‌توانند منجر به اختلال در عملکرد سیستم، سرقت داده‌های حساس و حتی تهدیدهای جانی و مالی شوند. برخی از تهدیدهای امنیتی رایج در CPS عبارت‌اند از:

- حملات تزریق داده^{۱۳}: در این نوع حملات، مهاجم داده‌های نادرست را به حسگرها تزریق می‌کند و باعث می‌شود که سیستم تصمیمات اشتباهی بگیرد. به عنوان مثال، در یک سیستم حمل‌ونقل هوشمند، تزریق داده نادرست می‌تواند باعث تغییر مسیر وسایل نقلیه خودران و ایجاد تصادف شود.
- حملات کنترل از راه دور^{۱۴}: مهاجم می‌تواند با نفوذ به دستگاه‌های متصل به شبکه، کنترل اجرای سیستم را به دست بگیرد. این نوع حملات در سیستم‌های صنعتی و بیمارستانی می‌تواند خطرات جدی ایجاد کند.
- حملات اختلال سرویس^{۱۵}: با ارسال حجم زیادی از درخواست‌های غیرضروری، مهاجمان می‌توانند عملکرد سیستم را مختل کرده و باعث از کار افتادن اجزای حیاتی آن شوند.

راهکارهای امنیتی پیشنهادی

- برای مقابله با تهدیدهای امنیتی، استفاده از مکانیزم‌های امنیتی پیشرفته ضروری است:
- رمزنگاری داده‌ها: استفاده از روش‌های رمزنگاری قوی مانند AES و RSA برای جلوگیری از دستکاری و سرقت اطلاعات.
 - احراز هویت چندمرحله‌ای^{۱۶}: افزایش امنیت دسترسی به سیستم‌ها با تأیید هویت چندلایه‌ای.
 - بلاکچین: استفاده از بلاکچین برای ایجاد سیستم‌های غیرمتمرکز و مقاوم در برابر نفوذ.
 - هوش مصنوعی و یادگیری ماشین: به کارگیری الگوریتم‌های یادگیری ماشین برای تشخیص ناهنجاری‌ها و حملات سایبری به صورت بی‌درنگ.

انرژی مصرفی و بهینه‌سازی منابع در CPS

سیستم‌های رایفیزیکی معمولاً در محیط‌هایی با منابع محدود، مانند حسگرهای بی‌سیم، دستگاه‌های نهفته و شبکه‌های توزیع‌شده اجرا می‌شوند. یکی از چالش‌های مهم در این سیستم‌ها بهینه‌سازی انرژی مصرفی و منابع محاسباتی است. محدودیت منابع می‌تواند تأثیر مستقیمی بر طول عمر باتری، تأخیر در پردازش و کیفیت سرویس (QoS) داشته باشد.

روش‌های بهینه‌سازی انرژی مصرفی در CPS

- بهینه‌سازی زمان‌بندی وظایف^{۱۷}: استفاده از الگوریتم‌های زمان‌بندی وظایف کم‌مصرف برای کاهش انرژی مصرفی پردازنده و حافظه.
- استفاده از سخت‌افزارهای کم‌توان^{۱۸}: به‌کارگیری پردازنده‌های کم‌مصرف مانند ARM Cortex-M، FPGA و ASIC برای کاهش توان مصرفی.
- بهینه‌سازی پروتکل‌های ارتباطی: طراحی پروتکل‌های ارتباطی کارآمد، که تبادل داده‌های غیرضروری را کاهش داده و انرژی را حفظ کنند.
- محاسبات لبه‌ای: انتقال بخشی از پردازش‌ها به لبه شبکه^{۱۹} برای کاهش وابستگی به مراکز داده ابری و کاهش انرژی مصرفی.

هماهنگی میان سیستم‌های پیچیده و مدیریت بی‌درنگ

این سیستم‌ها شامل اجزای مختلفی مانند حسگرها، عملگرها، شبکه‌های ارتباطی و الگوریتم‌های کنترلی هستند. هماهنگ‌سازی این اجزا یک چالش اساسی محسوب می‌شود، زیرا ناهمگونی سخت‌افزاری، تفاوت در استانداردهای ارتباطی و نیاز به پردازش‌های بی‌درنگ مشکلاتی را ایجاد می‌کنند.

چالش‌های هماهنگ‌سازی در CPS

- استانداردسازی پروتکل‌های ارتباطی: عدم وجود استانداردهای واحد برای ارتباطات بین اجزای CPS باعث ناسازگاری بین سیستم‌ها و افزایش پیچیدگی در یکپارچه‌سازی آن‌ها می‌شود.
- مدیریت داده‌های حجیم و پردازش بی‌درنگ: سیستم‌های CPS حجم عظیمی از داده‌ها را از حسگرهای مختلف جمع‌آوری می‌کنند که نیاز به تحلیل و پردازش بی‌درنگ برای تصمیم‌گیری‌های دقیق و سریع دارند.
- افزایش تحمل‌پذیری در برابر اشکال^{۲۰}: خرابی در یکی از اجزای سیستم نباید باعث از کار افتادن کل سیستم شود. برای این منظور، استفاده از الگوریتم‌های مقاوم در برابر اشکال و بازیابی خودکار^{۲۱} ضروری است.

مزایا و کاربردهای سیستم‌های رایافیزیکی

سیستم‌های CPS به دلیل یکپارچه‌سازی هوشمندانه زیرساخت‌های فیزیکی و محاسباتی، تأثیرات گسترده‌ای بر بهبود کارایی، کاهش هزینه‌ها، افزایش ایمنی و توسعه فناوری‌های پیشرفته در صنایع مختلف دارند. این سیستم‌ها با ترکیب حسگرها، پردازش بی‌درنگ داده‌ها، ارتباطات بی‌سیم و الگوریتم‌های پیشرفته کنترل، امکان خودکارسازی، پیش‌بینی و تصمیم‌گیری هوشمندانه را فراهم می‌کنند. در این بخش، ابتدا به مهم‌ترین مزایای CPS پرداخته و سپس کاربردهای آن در صنایع مختلف بررسی خواهند شد.

مزایای سیستم‌های رایافیزیکی

افزایش دقت و کارایی عملیاتی

یکی از مهم‌ترین مزایای سیستم‌های رایافیزیکی تحلیل بی‌درنگ داده‌ها و بهینه‌سازی فرآیندهای صنعتی و شهری است. CPS با جمع‌آوری داده‌های دقیق از حسگرها و پردازش آن‌ها به‌وسیله الگوریتم‌های پیشرفته یادگیری ماشین و کنترل تطبیقی، امکان اجرای فرآیندها را با دقت بسیار بالا و خطای کم فراهم می‌کند. این مزیت در صنایعی مانند تولید صنعتی، پزشکی و حمل‌ونقل بسیار حائز اهمیت است.

کاهش هزینه‌های عملیاتی و بهینه‌سازی انرژی مصرفی

سیستم‌های رایافیزیکی با استفاده از بهینه‌سازی مصرف منابع، کاهش هدررفت انرژی و افزایش بهره‌وری در تولید و مدیریت خدمات، هزینه‌های عملیاتی را کاهش می‌دهند. مدیریت هوشمند انرژی، استفاده از روش‌های پیش‌بینی مبتنی بر هوش مصنوعی و اجرای فرآیندهای نگهداری پیش‌گیرانه^{۲۲} باعث کاهش هزینه‌های تعمیر و نگهداری در صنعت، حمل‌ونقل و ساختمان‌های هوشمند می‌شود.

افزایش ایمنی و امنیت سیستم‌ها

CPS با به‌کارگیری حسگرهای پیشرفته، ارتباطات امن و الگوریتم‌های شناسایی تهدیدها، امنیت و ایمنی سیستم‌های فیزیکی را افزایش می‌دهد. این سیستم‌ها با تحلیل بی‌درنگ وضعیت محیط و پیش‌بینی خرابی‌ها می‌توانند از بروز حوادث و آسیب‌های جدی جلوگیری کنند. در حوزه‌هایی مانند خودروهای خودران، شبکه‌های انرژی هوشمند و زیرساخت‌های صنعتی حساس، این مزیت تأثیر قابل‌توجهی در جلوگیری از تهدیدهای سایبری و فیزیکی دارد.

کاربردهای CPS در صنایع مختلف

حمل‌ونقل هوشمند و وسایل نقلیه خودران

سیستم‌های رایافیزیکی در حوزه حمل‌ونقل هوشمند و خودروهای خودران کاربردهای گسترده‌ای دارند. یکپارچه‌سازی داده‌های حسگرهای خودرو، ارتباطات^{۲۳} V2X و الگوریتم‌های کنترل پیشرفته باعث افزایش ایمنی و کاهش تصادفات جاده‌ای می‌شوند. برخی از کاربردهای CPS در حمل‌ونقل در ادامه آمده‌اند:

- مدیریت هوشمند ترافیک از طریق تجزیه و تحلیل داده‌های حسگرهای شهری.
- وسایل نقلیه خودران که با استفاده از بینایی کامپیوتری و شبکه‌های عصبی عمیق، مسیرهای بهینه را انتخاب می‌کنند.
- سیستم‌های هشدار تصادف و کنترل تطبیقی سرعت در خودروهای مدرن.

پزشکی و سلامت دیجیتال

- در حوزه پزشکی، سیستم‌های رایافیزیکی به تحول در جراحی از راه دور، نظارت مستمر بر وضعیت بیماران و پزشکی مبتنی بر داده کمک کرده‌اند. در ادامه به برخی از کاربردهای مهم در این حوزه اشاره شده است:
- پایش سلامت بیماران در زمان واقعی با استفاده از دستگاه‌های پوشیدنی و اینترنت اشیا پزشکی (IoMT).
- جراحی از راه دور با کمک ربات‌های جراح مانند داوینچی^{۲۴} که دقت جراحی را بهبود می‌بخشند.
- سیستم‌های مدیریت دارو و نسخه‌های هوشمند برای بهینه‌سازی درمان بیماران.

شهرهای هوشمند و مدیریت زیرساخت‌ها

- در شهرهای هوشمند، سیستم‌های رایافیزیکی نقش مهمی در مدیریت انرژی مصرفی، کنترل آلودگی هوا و بهینه‌سازی حمل‌ونقل عمومی دارند. برخی از کاربردهای کلیدی در این زمینه عبارت‌اند از:
- سیستم‌های مدیریت هوشمند انرژی مصرفی که برق و آب را بر اساس الگوی مصرف کاربران تنظیم می‌کنند.
- نظارت بر کیفیت هوا و مدیریت آلاینده‌ها از طریق شبکه‌های حسگری گسترده.
- سیستم‌های کنترل ترافیک تطبیقی برای بهبود حمل‌ونقل عمومی و کاهش ازدحام در کلان‌شهرها.

سیستم‌های رایفیزیکی در Industry ۴.0 باعث تحول در تولید و زنجیره تأمین شده‌اند. این سیستم‌ها امکان اتصال ماشین‌آلات صنعتی، بهینه‌سازی تولید و اجرای نگهداری پیش‌گیرانه را فراهم می‌کنند. موارد زیر برخی از کاربردهای مهم CPS در Industry ۴.0 را بیان می‌کنند:

- خودکارسازی فرآیندهای تولید با استفاده از ربات‌های صنعتی پیشرفته.
- بهینه‌سازی زنجیره تأمین با تحلیل داده‌های بی‌درنگ و کاهش زمان تأخیر در توزیع.
- نگهداری پیش‌بینی‌شده^{۲۵} که با استفاده از یادگیری ماشین و پردازش داده‌های حسگرها، خرابی تجهیزات را قبل از وقوع پیش‌بینی می‌کند.

روندهای آینده در سیستم‌های رایفیزیکی

با پیشرفت فناوری‌های دیجیتال و توسعه روزافزون حسگرها، محاسبات ابری، ارتباطات پرسرعت و هوش مصنوعی، سیستم‌های رایفیزیکی نیز در مسیر تکامل و بهبود عملکرد قرار دارند. این سیستم‌ها در آینده نه تنها کارایی و امنیت بالاتری خواهند داشت، بلکه با استفاده از فناوری‌های نوظهور، تصمیم‌گیری هوشمندانه‌تر، پردازش سریع‌تر و هماهنگی بهینه میان اجزای فیزیکی و دیجیتال خواهند داشت. در این بخش، سه روند کلیدی که نقش مهمی در آینده سیستم‌های CPS ایفا می‌کنند، بررسی خواهند شد.

یکپارچه‌سازی با هوش مصنوعی و یادگیری ماشین

یکی از مهم‌ترین مسیرهای آینده CPS، ادغام هوش مصنوعی (AI) و یادگیری ماشین (ML) در فرآیندهای تصمیم‌گیری و کنترل است. این فناوری‌ها باعث می‌شوند که سیستم‌های رایفیزیکی رفتار خود را با محیط تطبیق داده و به صورت خودکار الگوهای بهینه‌تری را برای عملکرد خود انتخاب کنند. برخی از کاربردهای کلیدی هوش مصنوعی در CPS در ادامه آمده‌اند:

۱. پیش‌بینی رفتار سیستم‌ها با مدل‌های LSTM و CNN: شبکه‌های عصبی بازگشتی طولانی‌مدت (LSTM) و شبکه‌های عصبی کانولوشنی (CNN) به طور گسترده برای تحلیل و پیش‌بینی داده‌های زمانی در CPS به کار می‌روند. این مدل‌ها می‌توانند الگوهای رفتاری پیچیده را در سیستم‌های حمل‌ونقل هوشمند، مدیریت انرژی و نظارت صنعتی شناسایی کرده و تصمیم‌گیری‌های خودکار را بهبود بخشند.

۲. بهینه‌سازی زمان‌بندی وظایف با الگوریتم‌های مبتنی بر یادگیری تقویتی^{۲۶} (RL): الگوریتم‌های RL می‌توانند نحوه تخصیص منابع و زمان‌بندی فرآیندها را بهینه کنند. به عنوان مثال، در سیستم‌های تولید هوشمند، این فناوری می‌تواند مسیر حرکت ربات‌های صنعتی، فعالیت‌های ماشین‌آلات و مدیریت ترافیک اطلاعاتی را بر اساس داده‌های بی‌درنگ بهینه‌سازی کند.

۳. افزایش قابلیت‌های خودمختاری در سیستم‌های رایفیزیکی: سیستم‌هایی مانند خودروهای خودران، ربات‌های صنعتی و شبکه‌های هوشمند توزیع انرژی با استفاده از یادگیری تقویتی عمیق^{۲۷} (DRL) می‌توانند تصمیمات خود را بر اساس بازخوردهای محیطی بهبود بخشند و عملکردی تطبیقی ارائه دهند.

محاسبات لبه‌ای و مه‌آلود برای پردازش بی‌درنگ

یکی از چالش‌های اصلی سیستم‌های رایفیزیکی، حجم عظیم داده‌هایی است که حسگرها تولید می‌کنند و نیاز به پردازش سریع دارند. با ظهور محاسبات لبه‌ای و محاسبات مه‌آلود^{۲۸}، این سیستم‌ها قادر خواهند بود بخش زیادی از پردازش‌ها را به جای ارسال به مراکز داده ابری، در نزدیکی منبع داده انجام دهند. این فناوری‌ها چندین مزیت کلیدی دارند:

۱. کاهش تأخیر و افزایش سرعت پردازش داده‌ها: در سیستم‌های کنترل بی‌درنگ، مانند خودروهای خودران یا ربات‌های جراحی از راه دور، تأخیر در ارسال داده‌ها به سرورهای ابری می‌تواند منجر به حوادث ناگوار شود. محاسبات لبه‌ای با پردازش اطلاعات در نزدیکی حسگرها، امکان واکنش‌های سریع‌تر را فراهم می‌کند.

۲. بهینه‌سازی مصرف پهنای باند و کاهش هزینه‌های ارتباطی: به جای ارسال حجم زیادی از داده‌ها به مراکز داده ابری، فقط اطلاعات حیاتی و پردازش‌شده به سرورهای اصلی منتقل می‌شوند. این روش، به ویژه در شبکه‌های حسگر گسترده

در شهرهای هوشمند و زیرساخت‌های صنعتی، باعث کاهش مصرف پهنای باند می‌شود.

۳. ترکیب پردازش‌های توزیع‌شده و متمرکز: معماری محاسبات مه‌آلود امکان توزیع وظایف پردازشی را بین دستگاه‌های لبه، گره‌های مه‌آلود و مراکز داده ابری فراهم می‌کند. این ترکیب باعث بهبود کارایی و افزایش تحمل‌پذیری سیستم‌ها در برابر خطاهای احتمالی خواهد شد.

توسعه ارتباطات 6G و فناوری‌های بی‌سیم پیشرفته

نسل ششم شبکه‌های ارتباطی (6G) یکی از کلیدی‌ترین فناوری‌هایی است که آینده CPS را تحت تأثیر قرار خواهد داد. ارتباطات 6G با افزایش پهنای باند، کاهش تأخیر و بهبود امنیت امکان ایجاد سیستم‌های توزیع‌شده پیشرفته‌تر را فراهم می‌کنند. برخی از ویژگی‌های کلیدی 6G در سیستم‌های رایفیزیکی عبارت‌اند از:

۱. کاهش تأخیر ارتباطات به کمتر از یک میلی‌ثانیه برای کنترل بی‌درنگ: در سیستم‌های حساس به زمان مانند کنترل تجهیزات صنعتی، خودروهای خودران و زیرساخت‌های بهداشتی هوشمند تأخیر ارتباطات یک چالش اساسی است. شبکه‌های 6G با کاهش چشمگیر تأخیر، امکان پردازش بی‌درنگ و ارتباط سریع‌تر میان اجزای CPS را فراهم می‌کنند.

۲. افزایش امنیت با استفاده از بلاکچین و رمزنگاری کوانتومی: حملات سایبری به سیستم‌های رایفیزیکی یکی از تهدیدهای جدی است. فناوری‌های جدید در شبکه‌های 6G، از جمله رمزنگاری کوانتومی و استفاده از بلاکچین برای احراز هویت و حفاظت از داده‌ها، باعث افزایش سطح امنیت در CPS خواهند شد.

۳. افزایش چگالی اتصال برای پشتیبانی از میلیاردها دستگاه متصل: اینترنت اشیا صنعتی (IIoT) و حسگرهای توزیع‌شده در شهرهای هوشمند نیاز به ارتباطات بی‌سیم با ظرفیت بالا و تأخیر کم دارند. شبکه‌های 6G قادر خواهند بود تعداد بیشتری از دستگاه‌ها را بدون کاهش کیفیت ارتباطی مدیریت کنند.





The key to Artificial Intelligence has always been the representation.

Jeff Hawkins



Computers are incredibly fast, accurate, and stupid. Human beings are incredibly slow, inaccurate, and brilliant. Together they are powerful beyond imagination.

A computer scientist is someone who fixes things that aren't broken.



برای آزادی روح از خستگی‌ها بنویس؛ مروری بر روزمره‌نویسی‌های دانشجویان

خرده‌دانش‌ها و خرده‌پرسش‌ها اون بخشی از دانش ما هستند که یک گوشه‌ای از ذهنمون دارن خاک میخورن ولی یه جرقه کافیه تا خودشون رو نشون بدن. یه مکالمه با دوستا، یه بحث سر کلاس، یک فیلم و بسیاری مورد دیگه می‌تونن اون جرقه رو بزنین؛ تا حدی که به خاطرش ساعت‌ها تو اینترنت می‌گردیم و درمورد این جور مسائل دانش جمع می‌کنیم ولی بعد از مدتی دوباره برمی‌گردن تو انباری مغزمون و خاک می‌خورن یا فراموش می‌شن!

به قولی مغز ما استعداد بی‌نهایتی برای یادگیری داره ولی Data Loss زیادی هم داره و توی نمونه‌ای که انسان عملاً توسط اخبار و اطلاعات محاصره شده این ضعف شدیدتر هم خودش رو نشون میده! راه حل چیه؟ نوشتن! بنویسید! از روزمرگی‌هاتون در کار و درس و زندگی بنویسید؛ بنویسید و دانشتون رو با بقیه به اشتراک بگذارید و در آخر یادتون باشه «کم‌رنگ‌ترین جوهرها از قوی‌ترین حافظه‌ها ماندگارترند.»



Open to open source

امیرحسین رازلیقی

کارشناسی ۱۳۹۹



و کامنت داد که «به نظرم جای x و y رو عوض کن. چون اتفاق افتادن y در فلان کیس‌ها محتمل‌تره و باعث میشه سرعت ساخت tensor کند نشه».

خلاصه که انگار یه منتور اختصاصی داری و بهت کمک می‌کنه کدهای اصولی‌تر و بهتری بزنی.

از اون موقع واقعاً جذب این کار شدم. البته که کتاب‌خونه‌اش ظرافت‌های خیلی زیادی داره که جذابش می‌کنه. مثلاً، توی سیستم‌های apple silicon چیزی به اسم to(device). که توی پایتورچ همش درگیرشیم نیاز نداریم! چرا؟ چون یه چیزی هست به اسم unified memory که همه‌رم شما، مشترکاً توسط cpu و gpu استفاده میشه. یعنی اگه رم شما ۱۶ گیگه، انگار ۱۶ گیگ رم gpu هم داری! و هر تنسوری توسط mlx ساخته میشه، هم در دسترس cpu عه و هم gpu.

درنهایت، پیشنهاد می‌کنم توی وقتی آزادتون، یکمی open source رو بگردین و investigate کنین: خیلی چیزای جذابی برای یادگیری و ارائه دادن داره! و البته حواس‌تون باشه، چون به شدت اعتیاد

آورده 🤖🔥

خلاصه که پروژه رو clone کردم و شروع کردم به خوندن کدهای cpp. کاری که باید می‌کردم نسبتاً ساده بود، ولی قسمت سخت ماجرا، فهمیدن ساختار کلی و کارکرد بخش‌های مختلف و پیاده‌سازی اون پروژه بزرگ بود. اونم برای منی که همیشه cpp کد زدن برام عذابه!

خلاصه که بعد از تلاش‌های بسیار و نوشتن تعدادی تست، PR رو ثبت کردم. نکته جذابش اینه که با فاصله زمانی کمی، یکی از اعضای اون تیم (که research scientist توی اپله) اومد و کامنت گذاشت. سطح دقتش به جزئیات واقعاً جذاب بود! مثلاً یک بار یه جای کدم چنین چیزی بود:

```
if (x || y || z) {  
    Do something  
}
```

امروز می‌خوام یه کمی از تجربه شروع کارم توی دنیای open source بنویسم. توی عید، با یک کتاب‌خونه آشنا شدم که یه تیم کوچیکی چندماه پیش شروع کرده بودن به اسم MLX. (<http://github.com/ml-explore/mlx>) این کتاب‌خونه یک چیزی شبیه PyTorch ولی بصورت خاص طراحی شده که با GPU های Apple Silicon کار کنه. بعد چندماه، اپل این تیمو کامل استخدام می‌کنه توی تیم تحقیقاتش و این پروژه میشه یکی از پروژه‌های این سورس تازه‌اِپل.

منم همین‌جوری داشتم با این کتاب‌خونه یه کمی کار می‌کردم که ببینم چه فرقایی با پایتورچ داره، چقدر بهینه‌تره و این صحبتاً، به یه باگ خوردم (بدیهتاً چون هنوز stable release نیست و روی ورژن 0.9.0 ش هست، باگ کاملاً محتمله). رفتم روی ریپوش یه ایشو باز کردم. افراد اون تیمه اومدن کامنت گذاشتن و ازم سوال کردن و بعد هم تأیید کردن ایشو رو. گفتم حالا که تا اینجا اومدیم، بذار ببینم می‌تونم خودم یه PR بزنم و حلش کنم یا نه.





و همین‌طور این خونه‌ها وقتی خراب میشن، خود حافظه به سری خونه رو دست نخورده نگه می‌داره تا موقعی که به خونه خراب میشه ازش استفاده بکنه.

اون ۱۰ درصد و ۷ درصد که اون پایین می‌بینید هم نشون‌دهنده همین مسئله است و خبر نکته این‌جاست که وقتی این درصد از ۱ و ۲ رد میشه، کارایی ssd تون هم شروع به افت شدید می‌کنه!

راستی اگه خواستید بلایی که سر حافظه‌تون آوردید رو ببینید می‌تونید از دستور زیر استفاده کنید.

<code>sudo smartctl -a /dev/nvme0</code>	
Available Spare:	100%
Available Spare Threshold:	10%
Percentage Used:	7%
Data Units Read:	77,385,464 [39.6 TB]
Data Units Written:	71,859,698 [36.7 TB]
Host Read Commands:	1,152,495,229

باتوجه به همین مسئله از بالا به پایین ارزون‌تر و با توجه به مدارات و نرخ نویز افزوده برای تمایز بیت‌ها، از پایین به بالا کارایی و سرعت بالاتر هست و دوام خونه‌ها افزایش پیدا می‌کنه (این افزایش‌ها گاهی نمودارهای نمایی داره) حافظه‌های با کیفیت عادی که دست من و شماست معمولاً از نوع TLC و معمولی‌ترها از نوع QLC هستن.

کلاً این‌طوره که کارخونه‌ها برای این که دوام حافظه‌شونو با به عدد بتونن بیان کنن میان میگن در این حافظه کلاً در مجموع چقدر می‌تونن بنویسن و همون‌طوری که مشاهده می‌کنید، من در طی ۵ سالی که این لپ‌تاپ رو دارم تقریباً ۳۷ ترابایت بهش نوشتم (احتمالاً ۲۳ ترابایتش به خاطر OS و چند هسته‌ای باشه:)))

قابل توجه دوستان عزیزو و فنیم به چیزایی رو توضیح بدم بهتون جالبه.

حافظه‌های ssd طبق نرخ تحمل‌شون قیمت‌گذاری میشن و عملاً بعد از به مدتی خونه‌های حافظه‌شون خورده میشه! (حالاتی در درس معماری دکتر اسدی به سر کوچیکی می‌زنید به این داستان) ssd ها بر مبنای تعداد بیت نگهداری شده در هر خونه‌شون دسته‌بندی میشن.

SLC: Single Layer Cell

DLC: Double Layer Cell

TLC: Triple Layer Cell

QLC: Quad Level Cell

این‌ها انواع این نوع حافظه‌ها هستن که به ترتیب از اولی تا پایینی این شکلیه که یک بیت به ازای هر خونه، دو بیت، سه بیت و چهار بیت داده رو توی هر خونه نگهداری می‌کنند.



بسی قوی هستن در Cover Letter نویسی. عدد دقیق ندارم ولی گویا این مورد درصد قبولی رو مقادیر خوبی بالا می‌بره.

■ سعی کنید در رزومه، همه تجربیات‌تون رو بنویسید. فکر می‌کنم تعداد خوبی‌تون تجربه تدریس و اینا رو حتی به مقدار کم دارید؛ حتما بنویسیدش.

■ حواستون شدیداً به فرمتینگ رزومه‌تون باشه. بسیار زیاد پیشنهاد می‌کنم که از قالب‌های canva برای ساختن رزومه‌تون استفاده کنید.

■ در آخر هم این‌که بسی زیاد توصیه می‌کنم رزومه‌تون رو به دو سه نفر فرد با تجربه‌تر نشون بدید و ازشون فیدبک بگیرید.



و اگه شمام اونجا بودید، می‌تونید با این حرکت، حس همزاد‌پنداری‌شون رو جریحه‌دار کنید.

■ معمولاً HR های فنی گیت‌هاب رو چک می‌کنن و اگر کامیت‌های مداومی داشته باشید تأثیر مثبتی داره در رزومه‌تون. پیشنهاد می‌کنم روزانه LeetCode رو چک کنید و چالش‌هاشو بررسی و در گیت‌هاب‌تون قرار بدید. تو شرکت‌ها باید روزانه بخش کوچکی کار رو جلو ببرید و فعال بودن گیت‌هاب نشانه‌گر اینه که در این زمینه اوکی‌اید.

■ یک تریک خیلی خوب اینه که کتاب‌ها یا کورس‌های مطرح حوزه خودتون رو ببینید و بخوانید و در بخشی به اسم Related Courses and books بیفزایید. این مورد به شخص من بسی کمک کرده. فرضاً همین کتاب INSPIRED یکی از کتب مطرح حوزه پروداکته و در مصاحبه‌هام معمولاً افراد اشاره می‌کنن که «به INSPIRED هم که خوندی».

■ اگر امکان‌ش هست حتماً یک Cover Letter به انضمام رزومه‌تون ارسال کنید. برای نوشتنش هم مقادیر خوبی از جی‌پی‌تی و دوستان کمک بگیرید.

در چند سال اخیر تعداد خوبی رزومه‌لول جونیور و تازه‌کار به آچارهای سازمان‌های مختلف دادم. این پیام چکیده‌ای از پرتکرارترین فیدبک‌ها و نکاتیه که دریافت کردم:

■ داشتن تعدادی پروژه خوب، پس از دانشگاه خوب، بیشترین وزن رو در رزومه داره. سعی کنید که از یوتیوب و جاهای دیگه پروژه بیابید و بنزید و حتما هم در گیت‌هاب قرارش بدید و لینک ریپوی مربوطه رو در رزومه بذارید.

■ در همین راستا اگر پروژه‌ای که می‌زنید علاوه‌بر زمینه تکنولوژی، در زمینه موضوعی هم با کار شرکت نزدیکی داشته باشه، تأثیر مثبتی داره. فرضاً پروژه کلون اسنپ‌فود (که اگه اشتباه نکنم پروژه یکی از دروس دانشکده بود) قطعی برای کسی که می‌خواد اسنپ فود اپلائی کنه تأثیر مثبت داره؛ هر چقدر هم که ساده باشه.

■ اگر در دانشگاه خوبی تحصیل می‌نمایید، بالای بالای رزومه‌تون بنزیدش. عمده افرادی که در شرکت‌های بزرگ‌تر تکنولوژی هستن، در دانشگاه‌های بهتر و دبیرستانی قوی‌تر بودن





محصول نیازی به مهندس نرم‌افزار نباشد و همه چیز با یک توضیح زبان طبیعی قابل انجام باشد. ظهور ابزارهای کدنویسی اتوماتر مثل cursor و تلاش برای استفاده از ILM ها در جهت پیاده‌سازی صفر تا صدی یک محصول در این جهت‌اند. در واقع ممکن است چندی مهندسین نرم‌افزار دلخوش به رشد هوش مصنوعی، محصولات بیشتر و هوشمندانه‌تری را راحت‌تر طراحی و پیاده کنند، اما شاید در ده سال خودشان توسط AI بلعیده شوند.

وظیفه ما چیست؟ در وهله اول که ما نقشی در تحولات این دنیا نداریم، بایستی شل کنیم. در وهله دوم اما اگر فکر می‌کنید شما را برای شل کردن نساخته‌اند، می‌توانید امیدوار باشید که هر چه قدر شغل‌هایی از زیر این هرم حذف شوند، شغل‌های جدیدتری نیز به رأس آن به مرور اضافه می‌شوند. باید سعی کنیم با تغییرات هم‌سو شویم و جایگاه خودمان را در این هرم بسازیم. این یعنی چند کار مهم: اول، یادگیری مداوم و به‌روزرنگ داشتن مهارت‌ها، مخصوصاً در بخش‌هایی که انقلابی‌ترند. دوم، توسعه توانایی حل مسئله، یعنی این‌که فقط بلد بودن یک ابزار کافی نیست، بلکه باید بتوان مسائل واقعی را فهمید و آن‌ها را به مسائل قابل‌حل تبدیل و در نهایت راه‌حل ممکن برایشان ارائه کرد. سوم و نکته مهم اما تمرکز بر ارتباطات انسانی است. با اتوماسیون شدن بسیاری از فرایندهای فنی، ارزش کارهایی بالا می‌رود که نیازمند تعامل (در عین اختلافات سازنده) و ارتباط مؤثر و البته جلب اعتماد هستند. توانایی کار گروهی و ارتباط مؤثر با ذی‌نفعان پروژه به همان اندازه مهارت‌های تکنیکی مهم هستند. به هر صورت ما اکنون کمابیش در همان موقعیت دهه ۹۰ عکاسان آنالوگ هستیم...

امروز که این آگهی فرصت شغلی را دیدم برایم جالب بود. سابقاً عنوان مشابهی مثل ML Engineer هم وجود داشته اما آن بیشتر ناظر به استفاده از مهارت‌های مهندسی نرم‌افزار برای Deploy مدل‌های IML ای است. اما عنوان AI Software Engineer به عکس، بیان می‌کند که نقش شما مهندسی نرم‌افزار است و باید بلد باشید چگونه از مدل‌های هوش مصنوعی در جهت ساخت محصول استفاده کنید. به نظر من این تازه آغاز یک مسیر است و احتمالاً آرام طی چند وقت آتی بقیه شرکت‌های محصول‌محور هم به چنین role‌هایی متمرکز خواهند شد. پس بروید کوبر یاد بگیرید که خربزه آب است.

برای مثال، در دهه ۱۹۹۰، با گسترش فناوری دوربین‌های دیجیتال، خیلی از شرکت‌ها و افرادی که سال‌ها در زمینه عکاسی با نگاتیو و چاپ عکس‌های آنالوگ تخصص داشتند، به تدریج بخش بزرگی از بازار کار خود را از دست دادند. لابراتوارهای چاپ عکس کم‌کم ورشکسته شدند و عکاسان قدیمی، که کار با دوربین‌های مکانیکی و نگاتیو را خوب بلد بودند، دیگر مثل قبل تقاضا نداشتند. در عوض، رشد فناوری دیجیتال باعث ایجاد فرصت‌های شغلی جدیدی شد: افرادی که دانش و مهارت کار با نرم‌افزارهای ویرایش عکس دیجیتال مثل فتوشاپ داشتند، یا کسانی که می‌توانستند دوربین‌های دیجیتال پیشرفته‌تر را برنامه‌نویسی و به‌روزرسانی کنند، بسیار مورد توجه قرار گرفتند. مثال خیلی انتزاعی که من دوست دارم در مورد اختراع ماشین حساب است. تا قبل از اختراع و گسترش ماشین حساب، محاسبان و حسابداران ماهر با ابزارهایی مثل چرتکه و جداول لگاریتمی و حتی با محاسبه ذهنی یا قلم و کاغذ، سفارش‌های محاسباتی (مثلاً در بانک‌ها، اداره‌ها و شرکت‌های بزرگ) را انجام می‌دادند. این افراد برای انجام محاسبات پیچیده، آموزش خاصی می‌دیدند، اما با ورود ماشین‌حساب‌های الکترونیکی و ارزان شدن آن‌ها، نیاز به این تخصص و شغل کاهش پیدا کرد. بسیاری از کسانی که شغل‌شان صرفاً انجام محاسبات دستی و عددی بود، کار خود را از دست دادند یا مهارت‌شان دیگر مزیت محسوب نمی‌شد. در عوض، شغل‌هایی برای افراد متخصص در زمینه طراحی، تولید و تعمیر ماشین‌حساب و بعدها برنامه‌نویسی کامپیوترها و ماشین‌حساب‌های پیشرفته‌تر ایجاد شد.

و البته همه این‌ها به این معنی نیست که نقش دییتاساینیست، توسط مهندسین نرم‌افزار جایگزین می‌شود. خود SE ها از دو جهت تحت فشار برای تغییر هستند. جهت اول از این حیث که به مرور این عامل‌های (Agent) هوش مصنوعی تبدیل به لگوهای جدیدی برای ساختن سرویس‌ها و محصولات می‌شوند. همان‌طور که گفته شد مهندسین نرم‌افزاری که ندادند چطور لگوها رو باید در کنار هم قرار بدهند از دور بازی خارج می‌شوند (تحولاتی که در سمت MCP رخ می‌دهد، هر چند ضعیف اما آغازی بر همین سیل تغییرات عظیم‌تر است). جهت دوم فشار از سمت مدل‌های جنریتور اما به نظر من آخرالزمانی‌تر خواهد بود و آن هم وقتی است که دیگر برای فرآیند ساخت

در این حدود یک سال، هر وقت کسی از من می‌پرسید که چه طور دییتاساینیست بهتری شویم، پاسخ می‌دادم که مهارت‌های مهندسی نرم‌افزارتان را بهبود ببخشید. چرا که معمولاً کسی که چنین سوالی می‌کند یک سطح پایه‌ای از دانش هوش مصنوعی را دارد اما علاقه‌اش به هوش مصنوعی همچنان فراتر از SE است و برای همین دنبال بهتر شدن در آن است. در واقع روند این گونه بوده که AI ابتدا کار خود AI کارها را بیشتر و بیشتر اتومات کرده و کاری کرده که افراد بیشتری بتوانند توسعه‌دهنده AI باشند. این پدیده مخصوصاً برای نیازمندی‌هایی که با داده‌های غیرساختارمند که خوراک foundation model ها هستند، واضح‌تر است.

فرض کنید چند سال پیش، برای طبقه‌بندی آگهی‌های دسته‌بندی‌شده در یک پلتفرم مثل دیوار، باید ویژگی‌ها یا Feature های مربوطه را به صورت دستی استخراج می‌کردید و برای هر دسته الگوریتم‌های مختلفی امتحان می‌کردید تا بهترین نتیجه را بگیرید. مثلاً برای تشخیص اینکه یک آگهی مربوط به فروش خانه است یا خودرو، باید با استفاده از NLP ساده، کلمات کلیدی را جدا می‌کردید، و شاید مدل‌های سنتی مثل SVM یا Random Forest را با دنگ و فنگ زیاد آموزش می‌دادید. بعداً حالا شبکه‌های از پیش آموزش‌دیده مثل Bert آمدند که می‌توانستند در حکم یک نقطه شروع عمل کنند و اندکی از کار را ساده‌تر کنند. اما امروز به لطف مدل‌های پایه مثل GPT شما می‌توانید فقط با چند خط کد و تجربه و داده کمتری، مدل‌هایی بسازید که نه فقط دقت بالاتر، بلکه قابلیت تعمیم بیشتری دارند. این اتوماسیون موجب شده که دیگر صرف دانستن الگوریتم‌های پایه‌ای کافی نباشد؛ بلکه مهم‌تر این است که بدانیم چطور صورت مسئله را تعریف کنیم و راه‌حلی برایش پیشنهاد دهیم (جنبه محصولی قضیه) و چطور راه‌حل را پیاده‌سازی و اجرا کنیم (جنبه نرم‌افزاری قضیه).

اگر بخواهم خلاصه کنم، امروز «مهندس بودن» مهم‌تر از «دییتاساینیست بودن» است (و البته از آن ورش هم هست، مهندس نرم‌افزاری که گستره قابلیت‌های هوش مصنوعی را نداند، شبیه حسابدار است که از وجود ماشین‌حساب آگاه نیست). در واقع گسترش یک فناوری می‌تواند تأثیر معکوسی روی آدم‌های آن حوزه داشته باشد.



حالا چجوری ازش استفاده کنیم؟ شما اول می‌تونین وارد `sheets.new` بشین که یک شیت جدید براتون ساخته بشه. توی پراپرتیز این رو هم بگم که اگر با این روش آشنا نبودین، به‌طور مشابه می‌تونین وارد `docs.new`، `slides.new` و... هم بشید که یک فایل جدید براتون باز کنه. درگام بعدی کافیه از تب `Extensions` وارد `Apps Script` بشین. توی این محیط شما می‌تونین کدهاتون رو بزنین. برای اجرای تابع‌ها به صورت تکی، کافیه از منوی بالا انتخاب‌شون کنین و دکمه `Run` رو بزنین؛ اما اگر می‌خواین تغییراتون روی خود اون `WebApp` هم اعمال بشه، باید `Deploy` کنین. اولین بار کافیه `New Deployment` بزنین و تنظیماتش رو به‌صورت زیر انتخاب کنین:

Web app

Execute as

Me

The web app will be authorized to run using your account data.

Who has access

Anyone

برای دفعه‌های بعدی هم کافیه توی `Manage Deployments` دکمه `Edit` رو بزنین و یک ورژن جدید تعریف کنین. راستی، توی این محیط بهتره که بی‌خیال ورودی گرفتن به صورت دستی بشین!:

ما می‌خوایم توی این آموزش به روش پایه‌ای بات رو پیاده‌سازی کنیم و کاری به چیزهای آماده نداشته باشیم. حالا شیوه کلی کارمون چیه؟ این ربات ما هر کاری که انجام می‌ده (پا در اصل پیام‌هایی که می‌فرسته)، با `fetch` کردن یک `URL` در قالب زیر صورت می‌گیره:

`https://api.telegram.org/bot<TOKEN>/<METHOD>`

حالا ما به سادگی می‌تونیم این `URL` ها رو تشکیل بدیم. برای دریافت توکن که به سادگی می‌تونین وارد `@BotFather` بشین و بعد از ساخت ربات‌تون، اون رو دریافت کنین. برای متد هم خود مستندات تلگرام، بهترین منبع به حساب می‌آن. بریم با هم یک نمونه تابعش رو پیاده‌سازی کنیم.

sendMessage			
Use this method to send text messages. On success, the sent <code>Message</code> is returned.			
Parameter	Type	Required	Description
<code>business_connection_id</code>	String	Optional	Unique identifier of the business connection on behalf of which the message will be sent.
<code>chat_id</code>	Integer or String	Yes	Unique identifier for the target chat or username of the target channel in the format <code>@channelusername</code>
<code>message_thread_id</code>	Integer	Optional	Unique identifier for the target message thread (topic) of the forum, for forum supergroups only
<code>text</code>	String	Yes	Text of the message to be sent, 1-4096 characters after entities parsing
<code>parse_mode</code>	String	Optional	Mode for parsing entities in the message text. See formatting options for more details.

شاید شما هم تا حالا به سرتون زده باشه که بخواین یک ربات تلگرام بسازین، اما با روش‌های ساخت اون زیاد آشنا نباشین. توی این متن می‌خوایم به چند روش برای ساخت ربات تلگرام از جمله روش‌های رایگان اشاره کنیم.

الان احتمالاً اگر از یک بچه ۴ ساله هم پرسیم که یک زبان برنامه‌نویسی معرفی کن، بی‌درنگ به پایتون اشاره می‌کنه. ربات تلگرام هم از این قاعده مستثنی نیست و یکی از دم‌دستی‌ترین روش‌هاش، استفاده از کد پایتون به حساب می‌آد. برای ساخت ربات تلگرام با پایتون می‌تونین از کتابخانه‌های زیادی استفاده کنین که معروف‌ترین‌هاش `python-telegram-bot`، `elethon aiogram` و `telebot` هستن.

یکی از نکته‌هایی که جا داره برای `telethon` بهش اشاره کنیم، نیاز به `api_id` و `api_hash` هست. شما می‌تونین با مراجعه به سایت `my.telegram.org`، بخش مربوط به `API Dev tools` رو پر کنین و در نهایت `api_id` و `api_hash` رو دریافت کنین، اما این پایان ماجرا نیست! چرا؟ چون احتمالاً از داخل ایران نتونین به این سادگی این کار رو انجام بدین و ممکنه نیاز داشته باشین که شخصی دیگه از کشوری که تلگرام داخلش محدودیتی نداره، این کار رو براتون انجام بده. البته کاربرد این `api` بسیار فراتر از ربات تلگرام هست و با استفاده از این `api` می‌تونید به اکانت خودتون هم دسترسی داشته باشین. از اون جایی که `api_id` و `api_hash` قابل غیرفعال کردن نیستن، توصیه می‌کنم اگر نیاز داشتید، یک شماره مجازی تهیه کنین و از دوست‌هاتون بخوایید که برای اون اکانت `api` بگیرن.

درباره زدن کدهای پایتون بیشتر توضیح نمی‌دم، چون خیلی منبع براش زیاده. از سورس‌های موجود در اینترنت گرفته تا یار عزیزمون (`ChatGPT`). حالا که کد پایتون رو زدین، باید اجراش کنین. خودم هم می‌دونم که چشم‌پسته غیب گفتم، ولی خب معمولاً دستگاه خودتون برای این کار جوابگو نیست، چون تا کوچک‌ترین مشکلی در اتصال به اینترنت به وجود بیاد قطع می‌شه. خب حالا چه کاری می‌تونیم کنیم؟ یکی از راه‌های موجود، تهیه سرور مجازی که کد پایتون‌تون رو داخلش اجرا کنید، اما معمولاً برای استفاده‌های جزئی هزینه‌بره. پس انتخاب بهتر چیه؟ یک انتخاب دیگه، تهیه همون سرورها، اما به صورت ساعتیه و در نهایت گزینه اصفهانی‌مون برای پایتون، سایت `pythonanywhere` هست. این سایت یک‌سری قابلیت‌های مقدماتی رو به‌صورت رایگان ارائه می‌ده که تا حدی شبیه به همون سرور مجازی، اما یکی از مشکلاتش اینه که احتمالاً کدتون بعد از گذشت زمانی حدود یک روز، متوقف بشه و مجبور بشید دوباره اجراش کنید؛ اما به‌طور کلی گزینه جالبی به حساب می‌آد.

نوبتی هم که باشه، نوبت روش اصلی‌ای هست که این‌جا می‌خوایم بهش اشاره کنیم و اون هم سرویس سرویس سه مزیت کاملاً رایگانه. همیشه به سادگی میشه استفاده کرد.

مورد آخر شاید کمی عجیب میاد، چون شما می‌تونین ایجاد کنین و خیلی راحت اون‌ها رو در حالی که بات بالا هست، بخونین. راستی تا یادم نرفته، زبانش هم تقریباً همون `JavaScript` هست.



این `Google Script` هست. اصلی رو به ما ارائه می‌ده: بالاست.

از `Google Sheets` داخلش


```
function handleMessage(chat_id, text) {
    var sender_id = chat_id;
    var content_text = text;
    var userProperties = PropertiesService.getUserProperties();
    var state = userProperties.getProperty(chat_id);

    if (text === '/start') {
        userProperties.setProperty(chat_id, 'started');
        sendMessage(sender_id, START_MESSAGE);
    } else if (state === 'started') {
        userProperties.setProperty(chat_id, 'awaiting_name');
        sendMessage(chat_id, GET_NAME_MESSAGE);
    } else if (state === 'awaiting_name') {
        userProperties.setProperty(chat_id, 'awaiting_last_name');
        sendMessage(chat_id, GET_LAST_NAME_MESSAGE);
    } else if (state === 'awaiting_last_name') {
        userProperties.deleteProperty(chat_id);
        sendMessage(chat_id, SUCCESS);
    }
}
```

اگر یادتون باشه، گفتیم یکی از مزایای این روش، سهولت استفاده از Google Sheets هست. الان می‌خوایم با هم‌دیگه کمی درباره این هم صحبت کنیم. برای این کار همون‌طور که انتظار می‌ره، تابع‌های آماده خیلی خوبی موجودن. ما این‌جا صرفاً به دو تابع مقدماتی برای خوندن و نوشتن اشاره می‌کنیم. یکی از ساده‌ترین روش‌ها برای خوندن، استفاده از قطعه کد زیره. شما با این کد می‌تونین کل محتوای شیت مدنظر رو داخل یک آرایه دو بعدی بریزین و از این به بعد صرفاً به چشم یک آرایه دو بعدی بهش نگاه کنین. مثلاً می‌تونین روی اون حلقه برزین و یا کارهای دیگه‌ای بکنین.

```
var spreadsheet = SpreadsheetApp.openById(SPREADSHEET_ID);
var sheet = spreadsheet.getSheetByName(SHEET_NAME);
var data = sheet.getDataRange().getValues();

for (var i = 0; i < data.length; i++) {
    // loop
}
```

برای نوشتن هم یکی از روش‌های ساده می‌تونه به صورت زیر باشه که به طور مستقیم روی یک سلول بنویسین:

```
var cell = sheet.getRange(row + 1, column + 1);
cell.setValue(value);

var row = [Field_A, Field_B, Field_C];
sheet.appendRow(row);
```

بی‌شمار تابع کاربردی دیگه هم هستن که با کمی جست‌وجو، به‌سادگی پیدا‌شون می‌کنین.

یکی دیگه از چیزهایی که ممکنه توی ساخت ربات با این روش نیاز داشته باشین، کار با فایل‌ه. توی محیط گوگل اسکریپت، به طور مستقیم نمی‌تونین بارگذاری یا بارگیری فایل رو انجام بدین، اما باز هم یک جایگزین اصفهانی روبه‌روی شماست. شما می‌تونین از استوریج استفاده کنین. به عنوان مثال، بعضی از شرکت‌ها به شما استوریج ۵ گیگی رو به صورت رایگان ارائه می‌دن. شما می‌تونین داندلود و آپلود فایل رو با استوریج انجام بدین.

و بالاخره این آموزش تموم شد. همون‌طور که گفته شد، ما این‌جا صرفاً ساده‌ترین گزینه‌ها رو بررسی کردیم تا با مسیر اولیه آشنا بشین و بعدش متناسب با نیازتون، موارد کاربردی‌تر بیشتری پیدا کنین که از امکانات خاص‌تر بهره‌مند بشین.

توی این تصویر، بخشی از توضیحات مربوط به تابع sendMessage آورده شده. خب همون‌طور که می‌بینین، پارامترهای اجباری اون، chat_id و text هستن. پس پیاده‌سازی تابع sendMessage می‌تونه به این شکل باشه:

```
var TOKEN = 'ENTER_YOUR_TOKEN_HERE';
var TELEGRAM_URL = 'https://api.telegram.org/bot' + TOKEN;

function fetchAndLogUrl(url) {
    var response = UrlFetchApp.fetch(url);
    Logger.log(response.getContentText());
}

function sendMessage(receiverID, textMessage) {
    var url = TELEGRAM_URL + '/sendMessage?' +
        'chat_id=' + receiverID +
        '&text=' + encodeURIComponent(textMessage);
    fetchAndLogUrl(url);
}
```

پس چی کار کردیم؟ اول Telegram URL رو تشکیل دادیم. بعد اسم تابع مون (sendMessage) رو نوشتیم. بعدش علامت سوال گذاشتیم و در ادامه پارامترها رو دونه دونه بهش دادیم. این پارامترها با نماد & از هم‌دیگه جدا می‌شن. حالا یک سوال اساسی پیش میاد. تلگرام از کجا باید بفهمه اتفاقاتی که سمت ربات می‌افته رو به ما توی این‌جا نشون بده؟ این‌جاست که بحث Webhook می‌آد وسط. ما اول باید یک بار پروژه مون رو Deploy کنیم که بتونیم WebApp URL بگیریم ازش. بعد با استفاده از تابع setWebhook به تلگرام می‌گیم که هرچی ربات دریافت کرد رو به این WebApp بفرست:

```
function setWebhook() {
    var url = TELEGRAM_URL + "/setWebhook?url=" + WEBAPP_URL;
    fetch(url);
}
```

به‌طور مشابه اگر بخوایم این سرویس رو قطع کنیم، باید از deleteWebhook استفاده کنیم. حالا شاید براتون سوال بشه که وقتی WebApp ما پیام رو دریافت کنه، چه تابعی صدا زده میشه؟ این تابع doPost هست. شما به صورت زیر می‌تونین ارزش استفاده کنین:

```
function doPost(e) {
    var contents = JSON.parse(e.postData.contents);
}
```

یک بار که برای خودتون contents رو چاپ کنین، متوجه ساختار محتوای می‌شین و دستتون راه می‌افته. مثلاً برای پیدا کردن آیدی عددی فرستنده و محتوای پیام، می‌تونین از متغیرهایی به صورت زیر استفاده کنین:

```
var sender_id = contents.message.from.id;
var text = contents.message.text;
```

خب دیگه از این‌جا به بعد دستتون بازه که هر کاری خواستین با رباتون انجام بدین. قابلیت‌های زیادی هم داخل این سرویس قابل استفاده هستن. یک نمونه ساده‌ش که احتمالاً به کارتون بیاد، استفاده از state برای وضعیت کاربره‌است. مثلاً فرض کنین می‌خوانین اول به کاربر بگین اسمش رو بفرسته و بعدش هم بگین فامیلی‌ش رو بفرسته. خب توی چنین شرایطی باید وضعیت کاربر رو هم نگه دارین. مثلاً وضعیت انتظار برای وارد کردن نام خانوادگی. یکی از روش‌ها به این شکله:

قدم به دنیای دواپس

بر دیارهایی کلاتری

کارشناسی ۱۳۹۹



خودش رو در حالت عملیاتی و قابل استفاده می گذرونه.

وظایف مهندس DevOps چیه؟

مهندس دواپس باید بتونه یه زیرساخت خوب برای شرکت فراهم کنه و بعدش ارزش نگهداری کنه. لزوماً هم این کارها یک نفره انجام نمیشن و گاهی یک تیم DevOps باید روش کار کنه.

زیرساخت خوب یعنی چی اصلاً؟

زیرساخت خوب زیرساختیه که:

- هر اتفاقی توش بیفته، تحت هیچ شرایطی کاربرها اختلالی حس نکنن. (که البته حرف سنگینه و همیشه ۱۰۰٪ تضمین داد که همین اتفاقی نمی افته.)

- قابل گسترش باشه. یعنی به صورتی پیاده سازی بشه که اگر فردا، شرکت گسترش پیدا کرد و خواست صدها محصول دیگه تولید کنه، بتونه به راحتی اونهارو کنار زیرساخت فعلی مستقر کنه. (منظور از محصول، محصولات نرم افزاریه! نه مثلاً قاشق، چنگال)

- همه چی مشخص و مستند باشه. یعنی اگر نیروی جدیدی به شرکت اضافه شد، بتونه به راحتی با زیرساخت ارتباط بگیره و بفهمه چه خبره. برای این کار نیاز به lac داریم که پایین تر توضیح میدم.

- خطاهاش از هم مستقل باشن. یعنی مثلاً اگر اعتبار سیستم ارسال SMS مون تموم شد و سیستم پیامک دهی دیگه کار نمی کنه، اختلالی توی سیستم ورود به پنل کاربری به وجود نیاد، چون ربطی به هم ندارن!

- همه چیزش خودکارسازی شده باشه. زیرساخت سیستم باید به صورت خودکار فعالیت کنه، تغییرات جدید رو به روز کنه، تعداد سرورها رو کم و زیاد کنه و...

وقتی سیستم از یه حدی بزرگتر میشه، انجام دادن بسیاری از کارها به صورت دستی خیلی کار سختی می شه. مثلاً یه برنامه نویس می آد رنگ یه بخشی از سایت رو از آبی به قرمز تغییر می ده! آیا واقعاً نیازه یه مهندس دواپس بیاد کد رو از اول build و اجرا کنه و بیره روی سرور؟ آیا ابزاری وجود نداره که این کارها رو برامون انجام بده؟ قطعاً وجود داره و باید در زیرساخت شرکت ها پیاده سازی بشه.

چالش های دواپس

- تغییر فرهنگ سازمانی: پیاده سازی دواپس نیاز به تغییر فرهنگ سازمانی داره که ممکنه برای برخی شرکت ها چالش برانگیز باشه.

- نیاز به مهارت های جدید: دواپس نیاز به یادگیری مهارت های جدیدی مثل خودکارسازی، مدیریت زیرساخت به عنوان کد و استفاده از ابزارهای CI/CD داره.

- پیچیدگی فناوری: استفاده از ابزارها و فناوری های دواپس ممکنه در ابتدا پیچیده به نظر برسه.

به طور کلی، دواپس یک فرهنگ و مجموعه ای از روشهاست که به شرکت ها کمک می کنه تا نرم افزارها رو با سرعت، کیفیت و قابلیت اطمینان بیشتری تحویل بدن. این فرهنگ بر همکاری، خودکارسازی و بهبود مستمر تأکید داره و با استفاده از ابزارهای مناسب، می تونه به شرکت ها کمک کنه تا در دنیای رقابتی امروز موفق تر عمل کنند. هرچند پیاده سازی دواپس ممکنه چالش هایی به همراه داشته باشه، اما مزایای اون به قدری قابل توجه هستن که بسیاری از سازمان ها رو به سمت پذیرش این فرهنگ سوق داده.

پس (Site Reliability Engineering) چیست؟

خیلی اوقات واژه های Site Reliability Engineer (مهندس قابلیت اطمینان سایت) و DevOps Engineer (مهندس دواپس) به جای هم استفاده میشن، ولی در واقعیت تفاوت دارن. در واقع SRE فردی هست که وظیفهش بالا بردن اعتمادپذیری^۱، دسترسی پذیری^۲ و کمک به بهبود عملکرد^۳ سیستمه.

این کلمات پاراگراف بالا یعنی چی؟
اعتمادپذیری: این کلمه در واقع بیانگر یک احتمال و با درصد نشون داده می شه. اعتمادپذیری به توانایی یک سیستم برای انجام وظایفش به طور صحیح و بدون خطا در یک بازه زمانی مشخص اشاره داره. یعنی در واقع سیستم باید بتونه به طور مداوم و پایدار کار کنه و نتایج مورد انتظار رو تولید کنه.

- دسترسی پذیری: این کلمه هم مثل قبلی یه احتمال. دسترسی پذیری به میزان دسترسی بودن سیستم برای کاربران در زمانی که به آن نیاز دارن، اشاره داره. این مفهوم نشون میده که سیستم، چه مقدار از زمان کلی

دواپس چیست؟

دواپس (DevOps) ترکیبی از دو کلمه Development (توسعه) و Operations (عملیات) هست و به عنوان یک فرهنگ، فلسفه و مجموعه ای از روش ها و ابزارها در دنیای فناوری اطلاعات شناخته می شه. هدف اصلی دواپس، بهبود همکاری بین تیم های توسعه نرم افزار و تیم های عملیات فناوری اطلاعات، به طوری که بتونن نرم افزارها رو با سرعت بیشتر، کیفیت بهتر و با قابلیت اطمینان بالاتر به دست کاربران برسونن.

دواپس یه فرهنگه که بر همکاری، ارتباطات و یکپارچه سازی بین تیم های مختلف تأکید داره. در گذشته، تیم های توسعه و عملیات به صورت جداگانه کار می کردن. تیم توسعه مسئول نوشتن کد و ایجاد نرم افزار بود، در حالی که تیم عملیات مسئول نگهداری و اجرای نرم افزار در محیط های عملیاتی بود. این جدایی گاهی اوقات منجر به مشکلاتی مثل تأخیر در تحویل، تطابق نداشتن محیط های توسعه و عملیات و حتی اختلافات بین تیم ها می شد.

دواپس با ایجاد یک فرهنگ مشترک و استفاده از ابزارهای خودکارسازی، این شکاف رو پر می کنه. در فرهنگ دواپس، توسعه دهندگان و مهندسين عملیات با هم همکاری نزدیکی دارن تا اطمینان حاصل کنند که نرم افزارها نه تنها به سرعت توسعه پیدا می کنند، بلکه به راحتی و با اطمینان در محیط های عملیاتی نیز اجرا می شن.

مزایای دواپس چیه؟

- سرعت بیشتر در تحویل نرم افزار: با استفاده از دواپس، شرکت ها می تونن نرم افزارها رو با سرعت بیشتری به بازار عرضه کنن.
- کیفیت بالاتر: خودکارسازی تست ها و نظارت مستمر باعث میشه که نرم افزارها با کیفیت بالاتری تحویل داده شوند.
- قابلیت اطمینان: با استفاده از روش های CI/CD و نظارت مستمر، نرم افزارها با قابلیت اطمینان بیشتری اجرا می شن.
- همکاری بهتر: دواپس باعث بهبود همکاری بین تیم ها و کاهش تنش ها و اختلافات می شه.
- کاهش هزینه ها: خودکارسازی و بهبود فرآیندها باعث کاهش هزینه های عملیاتی و افزایش بهره وری می شه.

زیرساخت به عنوان کد^۶ یا Iac چیست؟

یکی از مسئولیت‌های مهم مهندسين دواپس، انجام دادن کارها به صورت کدمحوره. ما می‌تونیم به راحتی بریم از هر جایی سرور بخریم، روش کلی پکیج نصب کنیم، کلی کانفیگ و هزاران کار دیگه انجام بدیم. حالا اگر دو سال دیگه برگشتیم و خواستیم همین کارهارو انجام بدیم، ولی یادمون نبود چی کار کرده بودیم چی؟ اگر عضو جدیدی به تیم اضافه شد، باید همه چی رو دونه دونه برایش توضیح بدیم؟ اگر بخوایم زیرساخت‌مون رو تغییر بدیم و بریم از یه جای دیگه سرور بخریم، دوباره باید همه کارها و کانفیگ‌هایی که انجام دادیم رو از اول انجام بدیم؟

این جاست که ابزارهایی مثل Terraform و Ansible Helmchart values به کمکمون میان. با استفاده از این ابزارها کافیه یه سری فایل بنویسیم و داخلشون مشخص کنیم چه چیزهایی می‌خوایم. بعد اون‌ها همین کارها رو برامون انجام می‌دن، بدون این‌که ما در کارشون دخالتی داشته باشیم. با این کار اگر زیرساخت جدیدی بخریم، فقط همین کدها رو اجرا می‌کنیم، اگر عضو جدیدی بیاد، فقط همین کدها رو می‌خونه. هر وقت نیاز داشته باشیم هم می‌تونیم برگردیم ببینیم چی کار کردیم.

لازمه همه این‌ها هم اینه که کاری رو دستی انجام ندیم. وگرنه برمی‌گردیم می‌بینیم یه چیز این وسط کمه که باز هم هیچ‌کس نمی‌دونه چی بوده!))

داکر^۷ چیه؟

وقتی صحبت از برنامه‌ها یا application ها می‌شه، هزاران خط کد و نیازمندی‌ها و وابستگی‌ها خودشون رو نشون می‌دن. این مفاهیم فقط در حوزه برنامه نویسی مطرح هستن و یک مهندس دواپس که می‌خواد این برنامه‌ها رو روی سرور مستقر کنه و تحویل مشتری بده، نیازی به درگیر بودن باهاشون نداره. از سال ۲۰۱۳ یه ابزاری به زبان Golang و به نام داکر نوشته شد که فلسفهش ایزوله‌سازی برنامه‌ها از هم‌دیگه و اجرای راحت اون‌ها در هر فضایی بود. داکر با استفاده از کانتینرها^۸ می‌آد برنامه‌هامون رو اجرا می‌کنه. حالا این یعنی چی؟

ببینید وقتی شما داکر رو نصب می‌کنین، یه Engine باهاش نصب می‌شه که این Engine تقریباً همه‌جا ثابت. یعنی داکری که من روی سیستم خودم نصب می‌کنم، با اون‌ی که روی سرور (و خیلی فضاهای دیگه) هست،

تقریباً یکیه. این باعث می‌شه من مطمئن شم که اگر یه کدی روی سیستم من با استفاده از داکر اجرا میشه، روی یک سیستم دیگه هم قطعاً اجرا می‌شه، چون Engine مون یکیه.

مراحل کار با داکر چیه؟

۱. اول شما یه برنامه‌ای، به هر زبانی که دوست دارید می‌نویسید.

۲. بعد با استفاده از داکر اون رو build می‌کنین. خروجی این مرحله اصطلاحاً یه ایمج^۹ هست. فرض کنین یه چیزی روی CD نوشتین و همه‌جا می‌تونین ببریدش، به هر دستگاهی بزنین و اجراش کنین. پس اون کد رو build می‌کنین و حالا یه ایمج دارید که حاوی کد شماست، ولی خب نمی‌دونین توش چه خبره! نیازی هم ندارید که بدونید. چون دیباگ‌ها تون رو انجام دادید و مطمئنید کار می‌کنه. صرفاً می‌خواین ببرین روی سرور یا هر دستگاه دیگه‌ای اجراش کنین.

۳. حالا که ما یه ایمج آماده برای اجرا داریم، کافیه اجراش کنیم. بعد از این‌که اجراش کردیم، یه مفهومی به اسم کانتینر ایجاد میشه. در واقع کانتینرها فضاهایی برای اجرا شدن ایمج‌ها هستن. حالا کد ما در حال اجراست و می‌تونیم مثل قبل باهاش کار کنیم. اما فایده این کار چی بود اصلاً؟

فایده داکر چیه؟

تا این‌جا فهمیدیم که برنامه‌هامون در واقع در کانتینرهایی اجرا می‌شن. نکته مثبت داکر این جاست که این کانتینرها از هم مستقل هستن. استقلال و ایزوله بودن این کانتینرها نسبت به هم دیگه به ما این امکان رو می‌ده که بتونیم به راحتی، همه‌جا اجراشون کنیم. حالا خطاهاشون به هم دیگه ربطی ندارن و از هم دیگه کاملاً جدا هستن. از طرف دیگه، یک مهندس دواپس درگیر کد نیست و کاری به هزاران خط کد و نیازمندی‌هاش نداره و صرفاً با ایمج‌های داکر طرفه. بزرگترین مزیت داکر برای مهندسین دواپس، عدم درگیری‌شون با کده. مدیریت و نگهداری ایمج‌ها که صرفاً انتزاعی (abstraction) از هزاران خط کد هستن، بسیار راحت‌تره.

کوبرنتیز^{۱۰} چیه؟

حالا که ما کد هامون رو به صورت ایمج درآوردیم و شروع کردیم به اجرا کردنشون، نیاز به یه ابزاری داریم که بتونیم این کانتینرهای در حال اجرا رو مدیریت کنیم. بعضی وقت‌ها برنامه‌ها با خطاهایی مواجه میشن

و Crash می‌کنن. بعد از خرابی چه مکانیزمی باید در پیش بگیریم؟ همه این کارها و سناریوها با کوبرنتیز قابل مدیریت هست که ابزار بسیار پیچیده‌ای هست و یادگیری‌ش اصلاً به همین آسونی‌ها نیست. برای همین بیشتر از این توضیح نمی‌دم. اگر دوست داشتید، درباره‌ش بیشتر بخونین!))

دیگه تو دواپس چیا نیازه؟

یکی از مهم‌ترین مفاهیمی که وجود داره، مشاهده‌پذیری^{۱۱} هست. ابزارهای monitoring به ما این قابلیت رو میدن که حواسمون به همه‌جای سیستم باشه و بتونیم رفتار سیستم رو دائماً رصد کنیم. سه مفهوم خیلی جدی در رصد کردن رفتار سیستم وجود داره:

Trace: تریس‌ها به ما کمک می‌کنن تا بفهمیم وقتی یه درخواستی وارد سیستم‌مون می‌شه، چه مرحله‌ای رو طی می‌کنه تا خارج بشه، و این‌که در هر مرحله چه قدر کارش طول کشیده. این جوری می‌تونیم بفهمیم کجای سیستم‌مون کنده و داره رفتار غیرعادی نشون می‌ده. مثلاً درخواستی از طرف کاربر اومده که می‌خواد سبد خرید بسازه، پرداخت کنه و محصول برایش فرستاده بشه. با رصد کردن این فرآیند می‌تونیم بفهمیم که مثلاً فرآیند ساخت سبد خرید بسیار کند انجام شده و مشکل رو رفع کنیم.

Log: لاگ‌ها در واقع اون اتفاق‌هایی هستن که در برنامه‌های شما می‌افتن. وقتی یه برنامه‌ای ارور می‌ده، اونو توی لاگ خودش می‌نویسه. ما با بررسی این لاگ‌ها می‌تونیم بفهمیم در طول زمان چه اتفاقی برای چه بخشی از سیستم‌مون افتاده. به خصوص توی سیستم‌های بانکی این لاگ‌ها به شدت مهم می‌شن و ما باید ازشون backup بگیریم. در سیستم‌های مالی حساس باید این اتفاقات ثبت و نگهداری بشن که چه کسی در چه زمانی چه کاری انجام داده، چون هر لحظه ممکنه نیاز بشن. به قول یه عزیز می‌گه ممکنه اداره اطلاعات با یه گونی بیاد، اگه لاگ داشته باشین لاگو می‌بره، وگرنه خودتون می‌بره.

Metric: متریک‌ها یک سری شاخص هستن که فواید بسیاری دارن، ولی خب پایه‌ای‌ترین استفاده اون‌ها در بررسی وضعیت CPU سرورها و مصرف RAM و... هست. نیازه که ما این شاخص‌هارو از سرورهامون جمع‌آوری کنیم و بعد حواسمون بهش باشه که چیز غیرعادی در اون به وجود نیاد.

پرده سوم: هرکسی به یک فرشته نگاهبان نیاز دارد!

پیش از آن که از اقدامات منجی AMD در این شرکت سخن بگوییم، خوب است یک آشنایی اولیه با او نیز داشته باشیم.

لیسا سو، متولد ۱۹۶۹ در تایوان است که در دو سالگی به همراه خانواده‌اش به نیویورک مهاجرت کرد. پدرش تحلیل‌گر آماری و مادرش حسابدار بود. لیسا در مورد دوران کودکی خود می‌گوید:

«من مثل بسیاری از کودکان آسیایی تربیت شدم. سخت درس می‌خواندم، سخت کار می‌کردم و مجبور بودم همیشه بهترین باشم. بزرگترین شکست من در چشم پدر و مادرم این بود که دست‌خط چینی‌ام افتضاح بود. همیشه در دست‌خط نمره C می‌گرفتم و این، آن‌ها را دیوانه می‌کرد.»

لیسا در دوران تحصیل در دبیرستان به نظم، پشتکار و داشتن روحیه رقابتی مشهور بود و همیشه جزو دانش‌آموزان برتر قرار داشت.

لیسا از نوجوانی به مهندسی برق و یادگیری درباره نیمه‌رساناها و قطعات الکترونیکی علاقه زیادی نشان می‌داد و در نهایت بعد از پایان دبیرستان به MIT -برترین دانشگاه مهندسی دنیا- رفت تا در یکی از سخت‌ترین رشته‌ها، یعنی مهندسی برق، تحصیل کند.

لیسا تا مدرک دکترا در این رشته و دانشگاه تحصیل کرد و در مورد آن می‌گوید:

«شاید تعجب کنید، اما پدر و مادرم ترجیح می‌دادند من پزشک شوم. زمان ما، منزلت پزشکی از هر شغل دیگری بالاتر بود، اما من از دیدن خون اذیت می‌شدم. آخر این شد که دکترا گرفتم که خب فکر می‌کنم جایگزین قابل‌قبولی برای حرفه پزشکی باشد.»

لیسا پس از فارغ‌التحصیلی و مدتی کار در یک آزمایشگاه تحقیقاتی به شرکت IBM رفت. وی ۱۳ سال در IBM کار کرد و به‌عنوان مهندس و رهبر یک تیم نقش خود را در این شرکت بزرگ -که در آن دوران در اوج خود قرار داشت- به بهترین شکل ایفا کرد. او به نایب‌رئیس بخش تحقیقات و توسعه نیمه‌هادی‌ها ارتقاء یافت و محصولات نوآورانه‌ای را برای IBM بنیان‌گذاری کرد.

لیسا از اولین پژوهش‌گرانی بود که به فناوری سیلیکون روی عایق یا SOI می‌پرداخت، موضوعی که در آن زمان هنوز در مراحل تحقیقاتی اولیه بود. از نمونه نوآوری‌های او جابگزینی اتصالات آلومینیومی

داستان یک منجی به نام لیسا سو و نجات‌یافته‌ای به نام AMD!

پرده دوم: داستان AMD

AMD در سال ۱۹۶۹ در ایالات متحده آمریکا تأسیس شد و کار خود را با هدف تولید پردازنده‌های ارزان و رقابتی آغاز کرد. این شرکت به مرور زمان با معرفی محصولات خلاقانه و باکیفیت، به یکی از بزرگ‌ترین بازیگران در صنعت پردازش کامپیوتری تبدیل شد. این شرکت فراز و فرودهای زیادی را پشت‌سر گذاشت و اشتباهاتی کرد که در دهه ۲۰۱۰ میلادی، شرکت را تا مرز نابودی پیش برد.



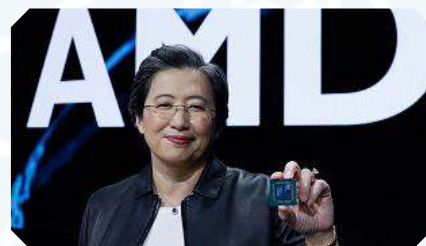
جری سندرز هم‌بنیان‌گذار AMD

این شرکت در رقابت با Intel، گاهی اوقات محصولات بهتری ارائه می‌داد، گاهی هم شکست می‌خورد، اما در دوره‌ای گاهی کارهایی کردند که فقط اشتباه نبودند بلکه عجیب‌غریب هم بودند!

سال‌های ابتدایی قرن ۲۱ بود که AMD با تراشه‌های ارزان و قدرتمندش از Intel قوی‌تر ظاهر شده بود، اما سبک مدیریتی حاکم بر این شرکت، آن‌ها را تله‌های ذهنی‌ای گرفتار کرده بود که قدرت درک موقعیت فعلی و آینده شرکت را از آنان گرفته بود.

این شرکت آن قدر بر رقابت با Intel متمرکز بود که از توسعه زیرساخت‌هایش جا مانده بود. زمان زیادی نگذشت که Intel با معرفی پردازنده‌های جدید و سریع‌تر، جایگاه خود را در بازار پس گرفت، حال آن‌که در AMD، مشکلات تولیدی و کیفیتی پردازنده‌ها باعث کاهش سهم بازار این شرکت شد. خریدهای غیرمعقول این شرکت نیز اوضاع را وخیم‌تر کرده بود و با بحران‌های مالی آن دهه و تصاحب بازار از سوی Intel، سبب مشکلات AMD کامل شده بود.

نوآوری کلید بقاست!



پرده اول: سیستم‌ها چگونه خود را نابود می‌کنند؟

برای یک کارآفرین اهمیتی ندارد که چرا یک نیاز برای انسان‌ها ایجاد شده‌است، او فقط به دنبال پاسخی برای آن نیاز می‌رود تا منافع مالی و روحی خود را تأمین کند! اما این شرط همان قدر که شرطی لازم برای تولد یک کارآفرین است، به همان اندازه شرط لازم برای بقای او است! این شرط در نتیجه رعایت یک اصل دنبال می‌شود که در یک جمله خلاصه می‌شود به «هرچه مشتری می‌خواهد» و با یک رویکرد قابل اجراست، و آن نوآوری است! بزرگ‌ترین نقطه قوت یک کارآفرین، ایده نوآورانه او نیست، بلکه توان و انگیزه او برای اجرایی کردن آن ایده است! اما این نقطه قوت، چاقویی است که می‌تواند دسته‌اش را نیز ببرد! گاهی اوقات کارآفرین‌ها و به بیان دقیق‌تری استارت‌آپ آن‌ها، آن قدر درگیر کارهای اجرایی می‌شوند که این اصول را فراموش می‌کنند و بدون این‌که بدانند، ناگهان خود را در باتلاق ایده‌هایی که روی آن قسم می‌خورند، می‌بینند!

داستان ما از این‌جا آغاز می‌شود! جایی پس از شروع، که تمام انرژی خود را آن‌جا خرج کردید! جایی پس از قله، که فکر می‌کنید جاودانه هستید! و در واقع جایی در دره نابودی که فکر می‌کنید اتفاقات خارج از اختیار شما هستند! جایی که فقط ذهنیت یک کارآفرین توانست تفاوتی از جنس میلیارد‌ها دلار رقم بزند و شرکتی را که در سطوحی زباله‌وال استریت ۲ دلار معامله می‌شد را به ارزش هر سهم بیش از ۱۰۰ دلار برساند!

- که در آن زمان بیشترین استفاده را در تراشه داشت- بامس بود.

این تغییر با توجه به فرصت‌هایی که به وجود می‌آورد، در نوع خود انقلابی در تراشه‌های IBM بود و این شرکت را تبدیل به یک شرکت پیشرو در این حوزه با مزیت‌های رقابتی بالا تبدیل کرد.

سو به عنوان یک مدیر و مهندس، دارای سطحی از دانش و توانایی بود که او را به فردی خاص در حوزه کاری تراشه‌ها تبدیل می‌کرد. او از دانش و همچنین تجربه کار با افراد بزرگ برخوردار بود. در واقع، سو به عنوان فردی شناخته می‌شود که هم علوم انسانی را خوب درک کرده و هم علوم طبیعی. از دیدگاه لیس، مدیریت فقط مهارت نیست، بلکه یادگیری و آموزش نقش مهمی را در مدیریت ایفا می‌کند.

پرده چهارم: فرشته نگهبان وارد می‌شود.

لیسا سو در سال ۲۰۱۴ و پس از دو سال فعالیت به عنوان مدیر ارشد در AMD به عنوان رئیس و مدیر عامل این شرکت منصوب شد. همان‌طور که خواندید، این شرکت در آن زمان اصلاً وضعیت مناسبی نداشت و به جز چند بازار محدود که در آن‌ها هم سود چشم‌گیری عایدش نمی‌شد، حرفی برای گفتن در حد و اندازه پیشروهای این صنعت نداشت.

در همین وضعیت، لیس سو مدیر عاملی این شرکت را برعهده گرفت. لیس مسئولیت شرکتی را برعهده گرفت که عملاً در حال غرق شدن در باتلاق ایده‌ها و تفکرات قدیمی بود، اما او این مسئولیت را این‌گونه تفسیر کرد:

«من تمام زندگی حرفه‌ای‌ام را در حوزه نیمه‌هادی‌ها گذراندم، اما در آمریکا تعداد شرکت‌های بزرگ حوزه نیمه‌هادی بسیار کم است. برای همین از این‌که قرار بود مدیر AMD شوم، بسیار هیجان‌زده بودم.»

لیسا عاشق چالش و کارهای دشوار بود، کارهایی که به افرادی جسور با تصمیماتی قاطع نیاز داشتند؛ چالش‌هایی که هر یک به تنهایی کمر یک شرکت را خرد می‌کرد؛ در همین حال، سو با همان ویژگی‌های شخصیتی‌اش که نمونه بارز یک کارآفرین موفق است، وارد میدان شد تا بار دیگر ثابت کند او فقط یک دکتر، مهندس یا حتی مدیر نیست؛ بلکه او یک الهه کارآفرینی است!

لیسا در مورد وضعیت AMD در آن زمان می‌گوید:

«ما با کمتر از یک درصد از سهم بازار شروع کردیم و آن زمان به نظر می‌رسید به پیشرفت بزرگی نیاز داشتیم. اما واقعیت این است که موفقیت در حوزه فناوری به شرط‌بندی‌های درست بستگی دارد و ما جزو اولین‌هایی بودیم

که به سیر تحول محاسبات کامپیوتری در ۱۰ سال آینده فکر کردیم. همین باعث شد پیشرفت چشم‌گیری داشته باشیم.» سو در روز اول کاری خود در AMD، نامه‌ای برای کارمندان نوشت که به تنهایی یک کلاس درس کامل برای استارت‌آپ‌ها بود. سو در مورد این نامه می‌گوید:

«در آن یادداشت طرحی ساده و سه‌اولویی برای همه موارد بعدی وجود داشت: تمرکز مجدد بر ساخت محصولات عالی، بازسازی روابط عمیق با مشتری و ساده کردن هر کاری که انجام می‌دهیم، زیرا این احساس را داشتیم که ما شرکتی کوچک تا متوسط هستیم که مانند یک شرکت واقعاً بزرگ عمل می‌کنیم. بنابراین بخشی از فرهنگی که می‌خواستیم بسازم، وضوح، تمرکز و جاه‌طلبی بود، اما همچنان ما باید سریع حرکت کنیم. ما باید چابک باشیم. این دستور ما برای موفقیت است.»

سو فهمید این شرکت در ساخت پردازنده برای محاسبات سنگین آینده روشن‌تری دارد، پس پتانسیل‌های این شرکت را با وجود این‌که هر چرخه توسعه محصول ۳ تا ۵ سال طول می‌کشد، به همین سمت هدایت کرد. سو برای رابطه شرکت با مشتریان خود ارزش زیادی قائل بود و می‌گفت مشتری‌هایمان باید قانع شوند ما یک نقشه جامع فناوری داریم که نیازهای آن‌ها را برطرف می‌کند. او در مورد این سیاست‌های خود می‌گوید:

«آن روزها مهم‌ترین موضوع برای ما این بود که اول بازارهایی را شناسایی کنیم که آینده خوبی داشتند. این‌که امروز چه کار می‌کنی مهم نیست؛ مردم به کاری که در طول سال‌ها انجام می‌دهی، توجه می‌کنند.»

سو عملاً یک ساختار جدید برای این شرکت ایجاد کرد تا بتواند راهبرد او در چابکی و نوآوری را به بهترین شکل اجرا کند. گویی خونی که در این شرکت جریان داشت، همان خونی بود که در بدن او بود! او هویت AMD را به درستی فهمیده بود و نوآوری‌ها و سیاست‌هایش نیز در همین راستا به نتایج درستی منتهی می‌شد.

«برای یک شرکت فناوری، تصمیم‌گیری درباره این‌که در چه حوزه‌هایی مهارت زیادی دارد، بسیار مهم است؛ چرا که باید در جایگاه خود بهترین باشی؛ یا رتبه اول یا دوم. کاری که باید بکنیم این است که DNA شرکت را تعریف کنیم و آن را تا جای ممکن با محصولاتی که می‌توانیم به بازار عرضه کنیم، بی‌عیب و نقص کنیم.»

سو در واقع یک مبادله اساسی بین فرصت‌ها و تهدیدها در اهداف این شرکت کرد و بر حوزه‌های برتر سرمایه‌گذاری کرد و بعضی بخش‌ها که در ظاهر فرصت‌های خوبی بودند

اما با منافع، هویت و توانایی‌های شرکت در تضاد بودند را کنار گذاشت.

«اول لازم بود مشتریانمان را متقاعد می‌کردیم که به ما اعتماد کنند. روشی که برای این کار انتخاب کردیم، این بود که به آن‌ها می‌گفتیم قرار است دقیقاً چه کار کنیم. می‌گفتیم سه سال طول خواهد کشید تا محصول ما برای عرضه آماده شود، اما تا آن موقع ما شما را در جریان تک‌تک مراحل فرایند ساخت قرار می‌دهیم و این دقیقاً کاری بود که کردیم.»

پرده آخر: امپراطوری سو در قامت AMD

AMD پیش از ورود سو شرکتی بود که در بازار وال استریت به عنوان یک زباله تکنولوژی شناخته می‌شد. ساختارهای درون‌سازمانی و مدل کسب و کاری کهنه، تصمیمات اشتباه مدیریتی و ده‌ها دلیل دیگر، این شرکت را در آستانه نابودی قرار داده بود تا این‌که فرشته نگهبان AMD وارد شد و جانی دوباره به یکی از قدیمی‌ترین شرکت‌های حوزه تکنولوژی بخشید.

AMD اکنون بیش از ۲۰ میلیارد دلار در سال درآمد دارد و سهامش بیش از ۱۲۰ دلار می‌ارزد. در بازار تراشه‌های کامپیوتری و گرافیکی حرف زیادی برای گفتن دارد و سال‌های متعددی بعد از ورود سو، هر سال در حال فتح قله‌های جدید است.

لیسا سو نمادی از یک کارآفرین واقعی است. کارآفرینی تبدیل به بخشی از ژن او شده که از روی آن می‌توان ده‌ها کلاس کارآفرینی، مدیریتی و مهندسی برگزار کرد. لیس در یک کلام، یک فرشته کسب و کار است.

«آن‌چه کار را برایم لذت‌بخش می‌کند، این است که فناوری‌ای که روی آن کار می‌کنیم، زندگی میلیاردها نفر را تحت تأثیر قرار می‌دهد. اکثر کارهایی که در طول روز انجام می‌دهید، به نوعی از دل یک پردازنده AMD عبور می‌کند.»





لیلا سادات علوی

کارشناسی ارشد
نرم افزار آه ۱۴۰۴

مصاحبه با لیلا سادات علوی هم‌بنیان‌گذار گفتا

امیر حسین تنهیدی

کارشناسی آه ۱۴۰۴



لطفاً یک معرفی اولیه از خودتان،
تحصیلات و فعالیت‌های دانشجویی و
غیردانشجویی‌تون ارائه می‌دهید.

من لیلا سادات علوی هستم، دانشجوی سال آخر کارشناسی ارشد مهندسی نرم‌افزار دانشگاه صنعتی شریف و دوره کارشناسی را در دانشگاه صنعتی اصفهان گذراندم. در انجمن علمی دانشکده فعال بودم و رویدادهای زیادی برگزار کردیم. برای مثال، یکی از رویدادهایی که برگزار کردیم، رویدادی شبیه AI Challenge بود که چند سال متوالی برگزار شد. همچنین یک نشریه دانشجویی مشابه بایت داشتیم. کارآموزی را در شرکت مهیمن اصفهان به عنوان مهندس نرم‌افزار گذراندم. مدتی نیز در شرکت ترب به عنوان مدیر محصول فعالیت داشتم و اکنون مدیر محصول گفتا هستم.

گفتا چیست و دقیقاً چه کاری انجام می‌دهد؟

گفتا در اصل قرار است به یک فروشنده هوش مصنوعی تبدیل شود. هدف ما این است که فرایند فروش را برای فروشندگان آسان‌تر و اثربخش‌تر کنیم، که ابعاد مختلفی را دربرمی‌گیرد.

برای مثال، هدایت خریداران واقعی به سمت فروشندگان، افزایش ارتباط مؤثر با مشتریان، بالا بردن نرخ تبدیل بازدیدکننده به خریدار و پیشنهاد محصولات مناسب به خریداران بر اساس نیازشان.

تمام این فعالیت‌ها را تلاش کرده‌ایم با استفاده از هوش مصنوعی مکالمه‌محور (Conversational AI) انجام دهیم، چرا که این روش هم بسیار رایج شده و هم در مقایسه با روش‌های قبلی عملکرد بهتری دارد.

چطور ایده گفتا به ذهن شما رسید؟

راستش ایده اولیه گفتا این چیزی که الان هست نبود. زمانی که GPT-4o معرفی شد و بحث چت‌بات‌ها داغ شده بود، در دانشگاه یک هکاتون حول این قضایا داشت برگزار می‌شد.

من بدون ثبت‌نام قبلی در این رویداد شرکت کردم و این موضوع برام جالب شد. ایده اولیه ما این بود که گفتا یک چت‌بات باشد که

مثل ساختار تیمی میکروسافت یا تیم کوچک تلگرام.

بر همین اساس، ترجیح دادیم به جای جذب افراد سینیور، افراد جونیور مشتاق به یادگیری را به تیم بیاوریم و خطاهای احتمالی را به عنوان هزینه‌های یادگیری بپذیریم.

تصمیم گرفتیم استارتاپ را به روش Bootstrap (تأمین مالی از منابع شخصی) پیش ببریم، چون باور داشتیم این روش چابکی تیم را حفظ می‌کند و به ما اجازه می‌دهد تصمیمات سریع و مستقل بگیریم.

به قول مارک زاکربریگ، افراد جونیور ارزش بیشتری برای تیم به همراه دارند، چرا که هم علاقه زیادی به یادگیری دارند و هم در مقابل تغییر انعطاف‌پذیرترند؛ در حالی که بسیاری از سینیورها دیدگاه‌های علمی و دانشی تثبیت‌شده‌ای دارند و به سختی می‌توانند خود را با تغییرات وفق دهند. یک نیروی جونیور اگر مرتکب اشتباهی شود، معمولاً تمایل دارد سریعاً آن را یاد بگیرد و اصلاح کند. در حالی که اگر فردی سینیور وارد تیم شود، معمولاً اصرار دارد که نظر خودش اعمال شود. در مقابل، ما ترجیح می‌دادیم اعضای تیم در کار مشارکت فعال داشته باشند.

شروع یا گسترش استارتاپ بدون جذب سرمایه سخت نیست؟

از نظر جذب سرمایه نیز ما با جذب سرمایه از سرمایه‌گذاران خطرپذیر (VC) موافق نبودیم. تجربه نشان داده اکثر VCها ارزش افزوده‌ای ایجاد نمی‌کنند و در بسیاری موارد حتی ضرر می‌زنند.

در نگاه ما، زمانی که محصول هنوز به تناسب بازار نرسیده (Product-Market Fit)، بهتر است سرمایه جذب نشود. VCها معمولاً فشار زیادی برای ورود سریع‌تر محصول به بازار وارد می‌کنند، در حالی که ممکن است زمان بیشتری برای بلوغ محصول نیاز باشد و چابکی تیم را می‌گیرند.

مثلاً ما الان قصد داریم مدل درآمدی خود را تغییر بدهیم. اگر یک سرمایه‌گذار در تیم ما حضور داشت، مجبور بودم هفته‌های متوالی در جلسات هیئت‌مدیره شرکت کنم و توضیح بدهم که چرا این مدل درآمدی بهتر است؛ چون سرمایه‌گذار معمولاً درکی از جزئیات کسب‌وکار ما ندارد. اما در شرایط فعلی، تیم ما راحت‌تر و چابک‌تر تصمیم می‌گیرد. حتی اگر تصمیم اشتباهی بگیریم،

پیام خریداران را دریافت کند و با درک خودش پاسخ دهد؛ زیرا در دیوار این نیاز را احساس می‌کردیم.

تحقیقات بازار انجام دادیم و داده‌هایی از فروشندگان جمع‌آوری کردیم و دیدیم بخشی اصلاً مکالمات را به خارج از پلتفرم هدایت می‌کنند که به نوعی برای هر دو ایده‌آل نبود و از طرفی هم سؤالات تکراری را باید ده‌ها بار با کم و زیاد پاسخ می‌دادند اما وقتی وارد کار شدیم، متوجه شدیم که صرفاً داشتن یک چت‌بات کافی نیست و احساس کردیم رسالت گفتا این هست که به افزایش فروش کمک کند. ابتدا گفتا قرار بود اطلاعات آگهی‌ها را دریافت کند و به سؤالات مشتریان پاسخ دهد، اما در ادامه با شناخت بهتر محصول، تصمیم گرفتیم گفتا را گسترش دهیم چون دیدیم این محصول پتانسیل دارد در تجارت الکترونیک بسیار اثرگذار باشد.

تیم گفتا چطور شکل گرفت؟

هرکسی که مرا بشناسد می‌داند که چند سالی بود علاقه‌مند به راه‌اندازی استارتاپ بودم، هرچند دقیقاً نمی‌دانستم چرا. دوران کارشناسی به دلیل مشغله‌های درسی و دیگر مسائل فرصت مناسبی نبود، اما در دوران ارشد که فشار درس‌ها کمتر شد، جدی‌تر به این موضوع پرداختم.

با چند نفر درباره هم‌بنیان‌گذاری صحبت کردم. ابتدا ایده‌های مختلفی مانند استفاده از AI برای فیلتر کردن رزومه‌ها داشتیم و حتی روی یکی از پروژه‌ها کمی کار کردیم، اما تیم زیاد به کار متعهد نبود و ادامه‌اش ندادیم. در نهایت با یکی از دوستان قدیمی‌ام صحبت کردم، یک تفاهم‌نامه هم‌بنیان‌گذاری نوشتیم و کار را شروع کردیم. بر خلاف برخی استارتاپ‌ها که ابتدا ایده را ارائه می‌کنند و سپس به دنبال جذب سرمایه می‌روند، ما از همان ابتدا روی فروش محصول تمرکز داشتیم و واقعاً هم فروش خوبی داشتیم.

هدف‌مان این بود که تیمی چند مهارته و چابک داشته باشیم؛ در روزهای اول که دو نفر بودیم، هر کدام در زمینه‌های مختلف از فرانت‌اند تا بک‌اند و مباحث فنی دیگر کار می‌کردیم.

بر این باوریم که محصولات بزرگ نیز با تیم‌های کوچک و چابک رشد کرده‌اند؛



سریعاً آن را می‌پذیریم و اصلاح می‌کنیم. در حالی که یک VC (سرمایه‌گذار خطرپذیر) اولویتش سود و سرمایه خودش است و دائم این پرسش را مطرح می‌کند که «سود و بازگشت سرمایه من کجاست؟»

از طرفی، با رشد هوش مصنوعی، امروزه اکثر تیم‌های جدید استارت‌آپ خود را به صورت Bootstrap (با سرمایه شخصی) راه‌اندازی می‌کنند و با یک تیم کوچک کار را آغاز می‌کنند. در گذشته، برای تأسیس پروژه‌هایی مثل کافه‌بازار، نیاز بود تعداد زیادی نیروی سینیور درگیر توسعه، طراحی و مدیریت کسب‌وکار شوند؛ و وقتی کسب‌وکار کمی اسکیل می‌شد، مدیریت آن بسیار سخت بود. اما اکنون کدنویسی برای کسانی که دید مهندسی دارند، بسیار آسان‌تر شده و به نظرم بسیاری از استارت‌آپ‌ها به این سمت حرکت خواهند کرد. در نتیجه، نقش VC‌ها در محصولات نرم‌افزاری کم‌رنگ‌تر خواهد شد؛ گرچه در استارت‌آپ‌هایی که با حمل‌ونقل و لجستیک سروکار دارند، همچنان به سرمایه‌گذاری‌های بزرگ نیاز است.

هزینه توسعه و چند مهارتی شدن نیروها هم کاهش یافته‌است. از نظر بازاریابی هم من معتقدم که یک محصول خوب خودش بهترین تبلیغ برای خودش است. محصول باکیفیت به‌صورت دهان‌به‌دهان شناخته می‌شود، اما محصولی که کیفیت متوسطی دارد یا محصول دوم یک بازار باشد، برای جا افتادن نیاز به تلاش بازاریابی بیشتری دارد.

مدل‌های بازاریابی مختلفی وجود دارد و مدل ما هم بیشتر بر همین مبناست؛ به همین دلیل، در مجموع نیازی به سرمایه‌گذاری VC ندیدیم و با وجود سختی‌ها، تصمیم گرفتیم به روش Bootstrap پیش برویم. در نهایت هم توصیه اکیدم به استارت‌آپ‌ها این است که در مرحله MVP اصلاً به سراغ جذب سرمایه نروند. بهتر است ابتدا در بازار جایگاه خود را تثبیت کنند و سپس جذب سرمایه را از یک Angel Investor آغاز کنند و بعد به سمت VC بروند، چرا که VC‌ها معمولاً کنترل زیادی روی تیم اعمال می‌کنند و چابکی تیم را کاهش می‌دهند.

گفتا چه اثری بر بازار گذاشته است؟

محصول ما پس از عرضه تأثیر قابل‌توجهی گذاشت. به حدی که پس از معرفی گفتا، دیوار نیز به سمت راه‌اندازی دستیار خرید حرکت کرد.

هدف اصلی ما ایجاد اثرگذاری بود و خوشبختانه احساس می‌کنیم به این هدف دست یافته‌ایم و باعث شده‌ایم ایده‌های نوآورانه بیشتری در این حوزه شکل بگیرد.

در مورد چالش‌های فنی و روند توسعه محصول گفتا توضیح می‌دهید؟

به نظر من کدنویسی نسبت به ده سال پیش بسیار آسان‌تر شده و چالش‌های فنی کم‌رنگ‌تر شده‌اند.

چالش اصلی ما بیشتر در حوزه محصول و کار با مدل‌های زبانی بزرگ (LLM) بود. باید فکر می‌کردیم چگونه این فناوری جدید را طوری به دست کاربر برسانیم که برایش قابل‌فهم و جذاب باشد. بسیاری از کاربران نحوه درست استفاده از پرامپت‌ها را نمی‌دانستند، بنابراین دامنه پرامپت‌ها را محدود کردیم تا کاربران عامی هم بتوانند به راحتی از محصول استفاده کنند. در محصول ما UX اهمیت بسیار بیشتری پیدا کرده و چالش اصلی این است که به کاربر بفهمانیم این اطلاعات از کجا می‌آید و چگونه باید از محصول استفاده کند تا بیشترین بازدهی را از قابلیت‌های LLM محصول بتوانند دریافت کنند.

امروزه بیشتر چالش‌ها از مسائل فنی به سمت چالش‌های محصولی (Product Challenges) حرکت کرده‌اند. چون فناوری‌هایی مثل LLM (مدل‌های زبانی بزرگ) بسیاری از قابلیت‌ها را آنلاک کرده‌اند و دست ما را باز گذاشته‌اند. اما مسئله این‌جاست که استفاده درست از این ابزارها نیاز به آموزش و مهارت دارد.

مثلاً خود مشتری‌ها دقیقاً نمی‌دانند یک LLM چه کارهایی می‌تواند انجام دهد. یا در زمینه‌های کسب‌وکاری، خیلی وقت‌ها مفاهیم ساده‌ای مثل Cross-sell برایشان ناشناخته است؛ در حالی که در عمل، همین رفتار را دارند.

ما باید ابتدا به مشتریان بفهمانیم که با استفاده از محصول ما، می‌توانند چنین کارهایی را ساده‌تر و هوشمندتر انجام دهند. بنابراین طراحی یک تجربه کاربری (UX) خوب اهمیت زیادی پیدا می‌کند؛ به طوری که فاصله بین درک جامعه و رشد تکنولوژی را کمتر کنیم و کاربر به راحتی متوجه شود چه کاری انجام می‌دهد و چه ارزشی دریافت می‌کند.

مدیریت تیم در گفتا چگونه انجام می‌شود؟

به نظر من مهم‌ترین عامل موفقیت یک استارت‌آپ تیم آن است؛ ایده‌ها و محصول را بسیاری می‌توانند ارائه دهند، اما ساختن تیمی با انگیزه، همدلی و تاب‌آوری در برابر بحران‌ها کار آسانی نیست و باید تیمی را تشکیل دهید که بتواند اختلاف نظر را باهم حل کنند و در شرایط مختلف تمرکز کافی روی محصول داشته باشند.

ما سعی کردیم انعطاف‌پذیری (Flexibility) را برای تیم بیشتر کنیم، چون به‌خصوص اول کار، خیلی چیزها رفاقتی پیش می‌رفت. مثلاً یک تسک تعریف می‌شد، یک نفر می‌گفت «باشه من می‌زنم»، ولی بعداً گفتیم به‌کم سبک مدیریتی را بهتر کنیم و برای همین استندآپ روزانه گذاشتیم و کم‌کم باگ‌های دیگر هم حل کردیم و به مرور ساختار مدیریتی تیم را پخته‌تر کردیم.

دقیقاً به همین خاطر است که نیروی جونیور از این لحاظ بهتر است. چون مثلاً بچه‌ها راحت فیدبک می‌دادند که فلان مشکل وجود داشت. این بازخوردها به ما کمک کرد روندها را اصلاح کنیم.

ما از همان اول سراغ استخدام اسکرام‌مستر نرفتیم. سبک کاری ما این‌طور بود که از فکر و نظر خود بچه‌ها استفاده کنیم. مخصوصاً الان که با دهه هشتادی‌ها کار می‌کنیم، این فضاهای تعاملی و انعطاف‌پذیر خیلی اهمیت دارد. چون واقعیت این است که بچه‌های این نسل، به‌خصوص بچه‌های شریف، همیشه یک گزینه بهتر برای کار کردن دارند؛ برای همین باید یک تمایز واقعی ایجاد کرد که باعث شود بچه‌ها حس رضایت و انگیزه برای کار را داشته باشند.

برای آینده گفتا چه برنامه‌ای دارید؟

ما یک بک‌لاگ بزرگ از ایده‌ها داریم و هنوز مشخص نیست کدام مسیر را انتخاب خواهیم کرد، اما فیلتر نوآوری در این انتخاب برایمان اهمیت زیادی دارد.

در صورت رشد تیم، برنامه داریم ساختار شرکت را به تیم‌های کوچک‌تر و مستقل تقسیم کنیم تا هر تیم مسئولیت محصول خود را بر عهده داشته باشد و با تیم‌های دیگر در تعامل باشند.

فعلاً تمرکزمان بر توسعه محصول فعلی است، اما در آینده دوست داریم پروداکت‌لاین‌های جدیدی راه‌اندازی کنیم.

همچنین علاقه به حضور در بازارهای بین‌المللی داریم، اما به دلیل رقابت شدید در این حوزه‌ای که کار می‌کنیم، بازارهای جهانی برای استارت‌آپ‌های کوچک خطرات زیادی دارند و فعلاً تصمیم داریم در بازار داخلی تمرکز کنیم تا به ثبات بیشتری برسیم.





معدی بحرینی

دکتر ۱۴۰۱

مفهوم ثابت افزار و اهمیت آن



معدی علی نژاد

کارشناسی ۱۴۰۱

یکی از تکنیک‌های پیشرفته برای جلوگیری از استخراج ثابت افزار، تغییر کد باینری در زمان اجراست تا فرآیند تحلیل مهندسی معکوس را پیچیده تر کند. همچنین روش‌های مبهم سازی کد^۱ مانع از تبدیل کد باینری به کد منبع می‌شوند.

ابزارهای تحلیل مانند Ghidra و IDA Pro برای استخراج اطلاعات از ثابت افزار مورد استفاده قرار می‌گیرند. برای مثال، در یک پروژه تحقیقاتی، امکان استخراج UUID و رمز عبور WiFi از ESP که در حافظه فلش بدون رمزنگاری ذخیره شده بود، بررسی شد.

اهمیت امنیت ثابت افزار: نمونه‌های واقعی

حملات علیه ثابت افزار در دنیای واقعی نمونه‌های مهمی دارد. یکی از این نمونه‌ها، حمله استاکسنت^۲ بود که با هدف تخریب تأسیسات هسته‌ای ایران انجام شد. استاکسنت با آلوده کردن سیستم‌های PLC^۳ باعث شد که سانتریفیوژهای غنی سازی اورانیوم از کنترل خارج شوند.

نمونه دیگر، حملات به ثابت افزار UEFI است که در مادربوردها مورد استفاده قرار می‌گیرد. برخی از بد افزارهای سطح ثابت افزار (Bootkits) از این ضعف بهره می‌برند و می‌توانند کلیدهای فشرده شده توسط کاربر را به سروری از راه دور ارسال کنند، بدون این‌که حتی آنتی ویروس‌ها قادر به شناسایی آن باشند.

فازینگ^۴ در ثابت افزار

فازینگ یک روش تست خودکار برای کشف آسیب پذیری‌های نرم افزار و سیستم‌های ثابت افزار است. در این روش، ورودی‌های تصادفی مختلف به سیستم داده می‌شود تا واکنش آن بررسی شود.

به عنوان مثال، اگر یک کد C با مقدار ورودی "۴" دچار خطا شود، فازینگ به دنبال یافتن چنین نقاط ضعفی در برنامه است. این ایده در سطح ثابت افزار نیز کاربرد دارد و با استفاده از الگوریتم‌های تولید داده^۵ یا جهش داده^۶، سیستم را تست کرده و نقاط ضعف آن را مشخص می‌کند.

به روز رسانی امن ثابت افزار

ثابت افزار در دستگاه‌های اینترنت اشیا (IoT) اغلب نیاز به به روز رسانی دارد. برخی از چالش‌های امنیتی آن عبارت‌اند از:

- جلوگیری از این‌که افراد غیرمجاز، خود را به عنوان سرور ثابت افزار معرفی کرده و نسخه‌ای آلوده از ثابت افزار را روی دستگاه نصب کنند.
- جلوگیری از دسترسی غیرمجاز به ثابت افزار هنگام ارسال آن از طریق شبکه.
- در گذشته، به روز رسانی ثابت افزار فقط از طریق اتصال فیزیکی انجام می‌شد، اما امروزه بیشتر به صورت به روز رسانی از راه دور (OTA)^۷ انجام می‌شود. چالش اصلی این روش، به روز رسانی بدون ایجاد اختلال در عملکرد فعلی دستگاه است.

روش‌های جدید برای به روز رسانی ایمن شامل استفاده از بلاکچین و رمزنگاری کلید عمومی و خصوصی برای جلوگیری از خواندن ثابت افزار توسط مهاجمان است.

تحمل پذیری در برابر خطا

تحلیل سیستم از نظر واکنش به خطاها، یکی از مباحث مهم در طراحی ثابت افزار است. در سیستم‌های بی درنگ^۸ مانند FreeRTOS، بررسی می‌شود که اگر خطایی مانند وارونگی بیت یا تغییر مقدار داده رخ دهد، سیستم چه واکنشی نشان می‌دهد.

یکی از روش‌های افزایش قابلیت اطمینان سیستم، استفاده از TMR^۹ است که از سه ماژول موازی برای انجام عملیات استفاده کرده و نتیجه با بیشترین رأی را انتخاب می‌کند. در کنار این روش، رویکردهای ترکیبی و پویا نیز برای تحمل پذیری در برابر خطا مورد استفاده قرار می‌گیرند.

روش‌های جلوگیری از استخراج ثابت افزار

رمزنگاری فایل‌های باینری ثابت افزار یکی از راهکارهای امنیتی در این حوزه است. این روش نه تنها در سیستم‌های ثابت افزار بلکه در سایر سیستم‌های نرم افزار نیز کاربرد دارد.

مفهوم ثابت افزار و اهمیت آن

ثابت افزار چیست؟

ثابت افزار تعاریف متعددی دارد. برخی آن را به عنوان نرم افزار تعبیه شده^{۱۰} می‌شناسند، برخی دیگر آن را واسطی میان سخت افزار و نرم افزار می‌دانند، اما رایج ترین تعریف آن، نزدیک ترین لایه نرم افزاری به سخت افزار است.

عملکرد ثابت افزار

ثابت افزار یک نرم افزار برنامه پذیر است که با جزئیات سخت افزار در ارتباط است، مانند تغییر مقادیر رجیسترهای خاص یا تنظیمات سخت افزار. در سیستم‌هایی همچون لپ تاپ‌ها و رایانه‌های شخصی که سیستم عامل دارند، ثابت افزار اولین نرم افزاری است که اجرا می‌شود. به عنوان مثال، در لپ تاپ‌های امروزی ثابت افزار UEFI استفاده می‌شود که هنگام فشردن دکمه پاور، ابتدا تست‌های اولیه مانند تست حافظه (Mem Test) و تست سلامت سیستم (POST) را انجام می‌دهد و سپس کنترل را به سیستم عامل واگذار می‌کند. برنامه‌ای که روی سیستم‌های نهفته اجرا می‌شود، ثابت افزار نام دارد. فایل باینری‌ای که روی یک میکروکنترلر بارگذاری می‌شود و نرم افزاری که روی یک روتر اجرا شده و پروتکل‌های ارتباطی را مدیریت می‌کند، مثال‌هایی از ثابت افزار هستند.

حوزه‌های مرتبط با ثابت افزار

امنیت ثابت افزار

یکی از مسائل مهم در امنیت ثابت افزار، دسترسی به فایل باینری آن و مهندسی معکوس برای استخراج ویژگی‌های مختلف است.

یک سال دیگه هم در انجمن علمی کنار هم کار کردیم، خندیدیم، خسته شدیم، بعضی مواقع به خاطر بعضی دلخوری‌ها با هم بد صحبت کردیم، کف خوار زمی تو سرما و گرما خوابیدیم و سر هر رویداد اتاق انجمن رو جو ری به فنا دادیم که شتر با بارش توش گم میشه!

ولی بیاید حالا که گذشته یکم از دورتر به همه‌اش نگاه کنیم؛ الان که فکر می‌کنم می‌بینم کنار هم خیلی کارای بزرگی کردیم؛ از کوچک‌ترین فعالیت هامون تا بزرگترین هاش واقعاً همه انرژی مون رو گذاشتیم تا به بهترین شکل برگذارش! صاف، ساده و خالص با هم جمع شدیم و با خامی‌های جوانی ولی با روح‌های پاک و بی‌شیله پيله مون رویدادهارو از صفر بالا آوردیم.

برای خسته شدن که زوده ولی پایه‌ای به بار دیگه از صفر شروع کنیم؟



The Imitation Game



حدود ۷۰ سال پیش، یعنی در سال ۱۹۵۰، آلن تورینگ، نابغهٔ ریاضیات و علوم کامپیوتر که به عنوان «پدر علم نوین کامپیوتر» شناخته می‌شود، در مقاله‌ای انقلابی با عنوان "Computing Machinery and Intelligence" پرسشی بنیادین مطرح کرد: «آیا ماشین‌ها می‌توانند فکر کنند؟» این سوال ساده، اما ژرف، آغازگر یکی از بزرگ‌ترین چالش‌های فلسفی و علمی عصر مدرن شد. تورینگ در این مقاله برای اولین بار مفهوم «آزمون تورینگ» را معرفی کرد؛ معیاری که پیشنهاد می‌کند اگر یک ماشین بتواند انسان را در مکالمهٔ متنی فریب دهد تا باور کند که در حال صحبت با یک انسان است، آن‌گاه می‌توان گفت که آن ماشین دارای «هوش» است. ایدهٔ تورینگ نه تنها دیدگاه‌ها را نسبت به ماشین‌ها تغییر داد، بلکه بنیان‌گذار تحقیقات پیشرفته در زمینهٔ هوش مصنوعی شد.

امروزه، با پیشرفت‌های حیرت‌انگیز در عرصهٔ فناوری، سوال تورینگ همچنان معتبر است. آیا ماشین‌های هوشمند امروزی توانسته‌اند به مفهوم «فکر کردن» نزدیک شوند؟ یا این که هوش مصنوعی تنها انعکاسی پیچیده از برنامه‌ریزی انسانی است؟

انتقادات و پیشنهادات خود را
بما در میان بگذارید

