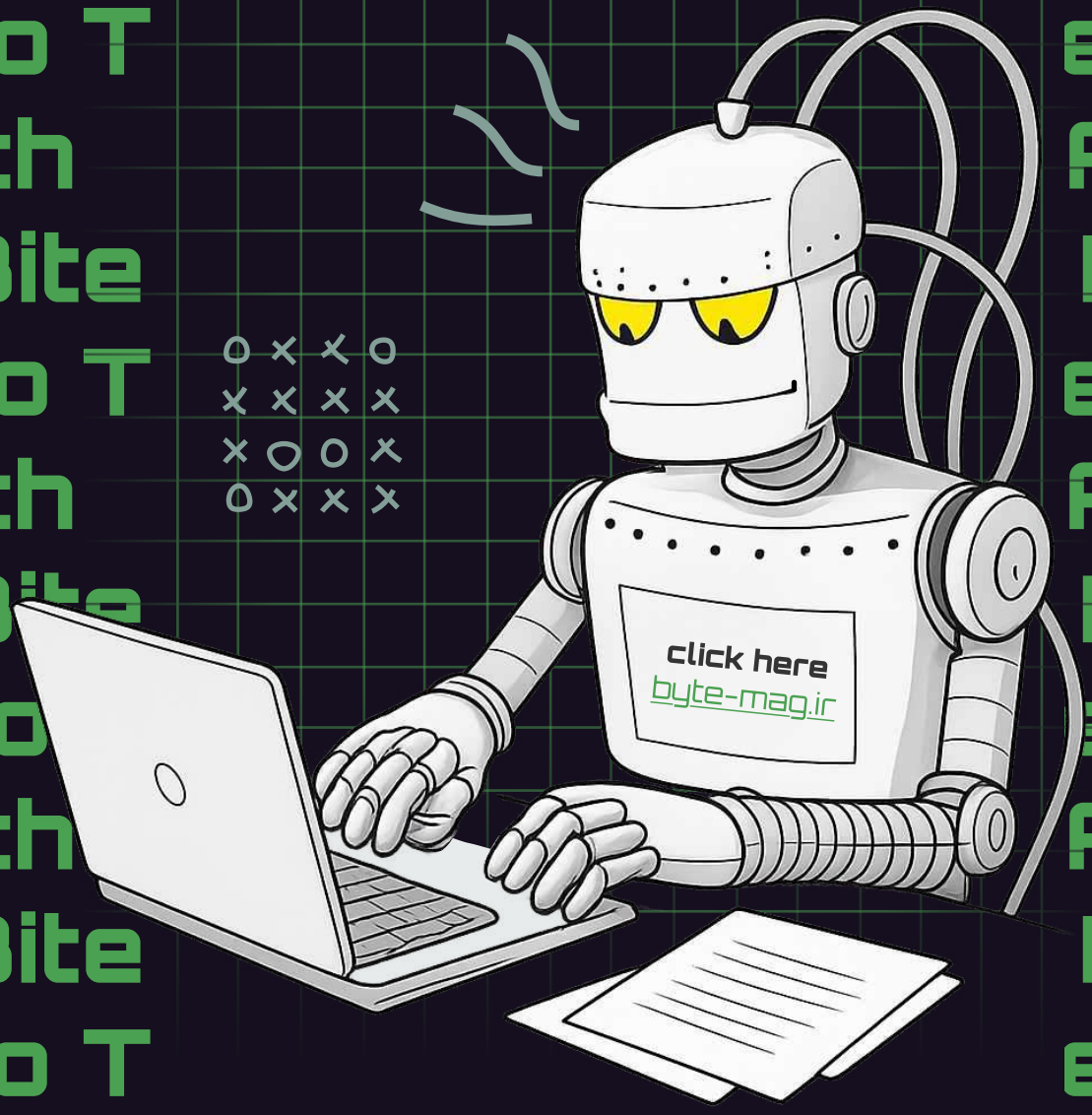
انجمن علمی دانشکدهٔ مهندسی کامپیوتر | نشریهٔ علمی فرهنگی بایـــت | شمارهٔ ۴

byte

مرداد ۱۴۰۴



فهرست

صاحب امتیاز: انجمن علمی دانشکدهٔ مهندسی کامپیوتر (SSC)

مدیرمسئول: امیرحسین شهیدی

سردبیر: آرش شاه‌حسینی

صفحه‌آرایی و گرافیک: آرش شاه‌حسینی، امیررضا اینانلو و معین آعلی

ویراستاران ادبی: امیرحسین صوری، متین غیاثی ، صهیب صادقی، امیرمهدی نامجو

تحریریه: آرمان طهماسبی، معین آعلی، مهدی علی‌نژاد، امیرحسین محمدزاده، متین غیاثی، امیرحسین صوری


آرش شاه حسینی
کارشناسی ۱۴۰۲

در تار و پود فناوری

سرمقاله

بعضی تغییرها آن‌قدر آرام رخ می‌دهند که حتی صدای پایشان را هم نمی‌شنویم. فناوری یکی از آن‌هاست؛ ابزاری که آرام و بی‌صدا شکل انسان را از درون دگرگون می‌کند. ما فناوری را تنها ابزاری برای انجام کار می‌بینیم و از پیامدهای عمیق‌تر آن غافل می‌شویم. این همان خطای شناختی (یا دیدن یک جنبه از واقعیت) است که اجازه نمی‌دهد عصر حاضر را صرفاً دوران پیشرفت فناوری بنامیم چرا که آن‌چه رخ می‌دهد، یک دگرگونی بنیادین در درک ما از انسان بودن است.

فناوری‌های نوین، از شبکه‌های داده‌ای تا سیستم‌های خودکار و ابزارهای هوشمند، دیگر فقط وسیله‌هایی بیرونی نیستند، آن‌ها مانند لایه‌ای نامرئی در زندگی روزمرهٔ ما رخنه کرده‌اند و نکتهٔ قابل‌توجه این‌جاست که ما انسان‌ها آگاهانه و با انتخابی روشن به سمت یکی شدن با فناوری حرکت نمی‌کنیم! بلکه آهسته، تدریجی و بی‌صدا در این مسیر پیش می‌رویم. دیگر صرفاً کاربر ابزارها نیستیم بلکه خود، بخشی از معماری این سیستم‌ها شده‌ایم.

اولین و ملموس‌ترین قربانی این تغییر، حافظهٔ ماست. وقتی موتورهای جست‌وجو در کسری از ثانیه، انبوهی از اطلاعات را پیش روی ما می‌گذارند، انگیزه‌ای برای به خاطر سپردن جزئیات باقی نمی‌ماند. این همان حافظهٔ برون‌سپاری‌شده است که مرزهای دانایی را تحت تأثیر قرار می‌دهد. ادراک ما نیز دگرگون شده‌است. شبکه‌های اجتماعی نسخه‌ای از جهان را به ما نشان می‌دهند که توسط الگوریتم‌ها فیلتر و بازآرایی شده‌است. آن‌چه می‌بینیم، دیگر تجربه‌ای مستقیم از واقعیت نیست؛ بلکه محصول انتخاب‌هایی‌ست که ماشین‌ها برای ما انجام داده‌اند. حتی ارادهٔ ما؛ بیش از هر زمان، به واکنش‌های شرطی وابسته شده‌است. یک اعلان یا لرزش گوشی کافی است تا تمرکز و تصمیم‌مان تغییر کند. اراده، از یک توانایی مستقل، به واکنشی وابسته به اعلان‌ها تقلیل یافته است.

آمارها نیز این روند را تأیید می‌کنند: دامنهٔ توجه انسان امروز به کمتر از ۸ ثانیه رسیده‌است! یعنی کمتر از یک ماهی قرمز. در شهرها، ۸۹ درصد مردم بدون نقشه‌های دیجیتال مسیر خود را پیدا نمی‌کنند. پژوهش‌ها نشان می‌دهد که ظرفیت حافظهٔ رویدادی ما در یک دههٔ گذشته ۴۰ درصد کاهش یافته‌است. این فقط یک تغییر کوچک نیست؛ نشانهٔ یک بحران عمیق‌تر است.

حالا بیاییم یک بازبینی مجدد کنیم؛ آیا این مسیر، مسیر تکامل انسان است یا ساختن نسخه‌ای ناقص از بشر؟

احتمالات پیش روی ما موارد زیر را بیان می‌کند و خالی از لطف نیست اگر بگوییم همین حالا هم دچارشان شده‌ایم:

بهینه‌سازی مخرب: بهره‌وری افزایش می‌یابد، اما احساسات، شک، شهود و خطا که بخش جدایی ناپذیر انسان هستند، کم رنگ می‌شوند.

توهم انتخاب: ما تصور می‌کنیم آزادانه تصمیم می‌گیریم، اما چارچوب‌های الگوریتمی از پیش، مسیرها را تعیین کرده‌اند. حتی مصارف، روابط و اعتراض‌ها نیز در محدوده‌ای کنترل‌شده شکل می‌گیرند.

تمام این‌ها را گفتیم تا بگوییم: پیشرفت فناوری، اگر با بازاندیشی در ما همراه نباشد، نه ما را به آینده‌ای روشن‌تر می‌برد و نه ما را به انسان‌هایی کامل‌تر تبدیل می‌کند؛ بلکه آرام و بی‌صدا، جوهرهٔ ما را می‌گیرد و فقط پوسته‌ای از «انسان» باقی می‌گذارد.

همانند گذشته این شماره از بایت نیز تلاش می‌کند جنبه‌های مختلفی از فناوری و چالش‌های پیش روی آن را به تصویر بکشد و با همراهی شما در میان تار و پود فناوری قدم بگذارد.

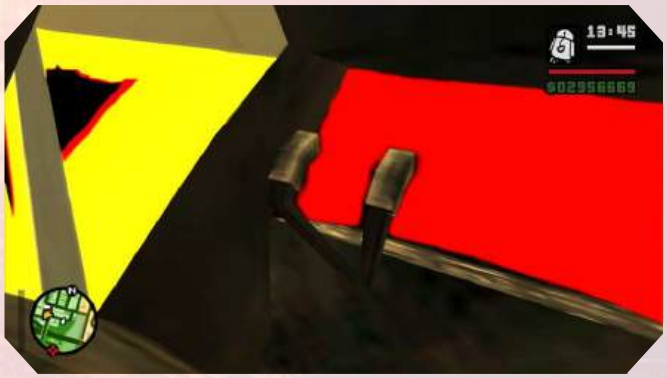
چگونه آپدیت ویندوز GTA: San Andreas را خراب کرد؟


هیربد بهنام
کارشناسی ۱۳۹۹

در یک روز زیبای زمستانی، مایکروسافت تصمیم گرفت که آپدیت 24H2 را برای تمامی کامپیوترهایی که ویندوز ۱۱ دارند منتشر کند. این آپدیت پشتیبانی از HDR، امکان باز کردن فایل‌های 7z و tar در Windows Explorer، دستور sudo در powershell و بهینه شدن مصرف باتری را به این سیستم‌عامل اضافه کرده بود. اما این آپدیت یک مشکل برای گیمرهایی که همچنان بازی نوستالژیک و محبوب Grand Theft Auto: San Andreas را بازی می‌کنند، به‌وجود آورد: هواپیمای آب‌نشین skimmer دیگر ساخته نمی‌شد. این مشکل به حدی بود که بازیکنان حتی نمی‌توانستند این هواپیما را به کمک کد تقلب[1] در بازی بسازند.

در این حین، برنامه‌نویس ماد Silent Patch -که یک ماد برای بهبود سری بازی‌های GTA III، GTA: VC و GTA: SA بر روی سیستم‌های جدید است- دست به کار شد که مشکل اصلی را پیدا و رفع کند. او قبلاً در Silent Patch با مشکلاتی نظیر سیاه شدن صفحه به‌هنگام خروج از ساختمان‌ها، نشتی حافظه‌های[2] مربوط به عکس گرفتن و مشکلات مربوط به چندمانیتوری دست‌وپنجه نرم کرده بود. برای این کار، او در ابتدا به کمک دو ماشین مجازی که یکی نسخهٔ 23H2 و دیگری نسخهٔ 24H2 ویندوز ۱۱ بود، بررسی کرد که آیا واقعاً مشکل نسخهٔ ویندوز است یا خیر و او بعد از استفاده از ماشین‌های مجازی مطمئن شد که ویندوز یک مشکلی برای بازی ایجاد کرده‌است؛ چرا که بر روی نسخهٔ 23H2 ویندوز ۱۱ این مشکل وجود نداشت و هواپیمای skimmer هم در قسمت‌های مختلف نقشه وجود داشت و هم می‌توانست آن را به کمک کدهای تقلب بسازد.

او در ابتدا سعی کرد که در نسخهٔ بدون ماد بازی، هواپیمای skimmer را بسازد و CJ را داخل هواپیما قرار دهد. زمانی که او این کار را انجام داد، CJ به ارتفاع ۱۰۳۰ متری سطح زمین پرتاب شد که مطمئناً ارتفاع درستی برای هواپیما نیست. همچنین زمانی که او ماد Silent Patch را اجرا کرد CJ به بالا پرتاب نشد، بلکه بازی فریز شد و در نهایت کرش کرد.



سپس او شروع به دیباگ کردن بازی کرد که متوجه شود که مشکل چیست. در ابتدا او در تابعی که هواپیما را در محیط بازی قرار می‌دهد، یک break point گذاشت و متوجه شد که زمانی که بازی می‌خواهد سرعت چرخش پره‌های هواپیما را حساب کند ارتفاع هواپیما عددی در اردر 1030 است که منطقی نیست. پس دو احتمال وجود دارد:

۱. بازی به صورت اشتباهی هواپیما را در یک ارتفاع خیلی زیاد می‌سازد.

۲. هواپیما واقعاً بر روی سطح زمین ساخته می‌شود ولی به دلیلی در آسمان پرتاب می‌شود.

در ادامهٔ بررسی‌هایش او متوجه شد که تابعی به نام CreateCarForScript وجود دارد که وظیفهٔ آن ساختن یک وسیلهٔ نقلیه در نقطهٔ خاصی از نقشه است. در این تابع، خطی به صورت زیر وجود دارد:

```
posZ += newVehicle→GetDistanceFromCentreOfMassToBaseOfModel();
```

همان طور که مشخص است این تکه کد، ارتفاع وسیلهٔ نقلیه را نسبت به مرکز ثقل آن افزایش می‌دهد. سپس او با بررسی داخل تابع GetDistanceFromCentreOfMassToBaseOfModel متوجه شد که در bouding box این هواپیما این مقادیر وجود دارند:

```
- *(RwBBox**)0x00B2AC48   RwBBox *
  - sup    RwV3d
      x    -5.39924574     float
      y    -6.77431822     float
      z    -4.30747210e+33 float
  - inf    RwV3d
      x    5.42313004 float
      y    4.02343750 float
      z    1.87021971 float
```

در بازی‌ها bouding box به مکعبی گفته می‌شود که کل یک شئ درون بازی را در بر می‌گیرد. همان طور که مشخص است مقدار محور عمودی این هواپیما خیلی عدد درستی نیست. اما این موضوع عجیب است چرا که این عدد نه اولین عدد این ساختار[3] است و نه آخرین عدد و اعداد کنار آن هم دست نخورده باقی‌مانده‌اند. پس یک نوع memory corruption عجیب داریم که فقط وسط یک ساختار را دست می‌زند. با بررسی بیشتر و چک کردن هر تکه کد که مقدار این bouding box را عوض می‌کند، او متوجه شد که در ابتدا که هواپیما داده‌هایش را از دیسک لود می‌کند، مقدار ارتفاع bouding box (همان z) درست است ولی بعد از مدتی خراب می‌شود. با بررسی بیشتر می‌توان متوجه شد که تابعی به نام SetupSuspensionLines وجود دارد که مسئولیت مقداردهی اولیه فنربندی هر وسیلهٔ نقلیه را دارد. در این تابع کدی به صورت زیر وجود دارد:

```
colModel→bbox.sup.z = pSuspensionLines[0].p1.z;
```

مقدار bbox.sup.z همان مقدار محور z مربوط به bounding box است. از آنجایی که این مقدار زمان اجرا غلط است، پس می‌توان نتیجه گرفت که مقدار pSuspensionLines[0].p1.z به کل اشتباه است. این مقدار از فایلی به نام vehicles.ide خوانده می‌شود که اطلاعات چرخ‌های هر وسیله نقلیهٔ بازی را در خودش دارد. در صورتی که خط مربوط به هواپیمای skimmer را مشاهده کنیم اطلاعات زیر را می‌بینیم:

```
460,skimmer,skimmer,plane,SEAPLANE,SKIMMER,null,ignore,
5,    0,    0
```


cheat code.۱ memory leak.۲ structure.۳

اما در صورتی که خط مربوط به یک هواپیمای دیگر مانند rustler را نگاه کنیم، متوجه می‌شویم که در آخر این خط چندین پارامتر دیگر نیز وجود دارند:

```
476, rustler, rustler, plane, RUSTLER, RUSTLER, rustler,
ignore, 10, 0, 0, -1, 0.6, 0.3, -1
```

پارامترهای آخر خط که در هواپیمای skimmer وجود ندارند، مربوط به اندازهٔ چرخ‌های هر هواپیما هستند. نکته‌ای که وجود دارد این است که قایق‌ها این پارامترها را ندارند؛ چرا که قایق‌ها چرخ ندارند. در این‌جا هم از آن‌جایی که skimmer چرخ ندارد و یک هواپیمای آب‌نشین محسوب می‌شود، این پارامترها کلاً وجود ندارند! همچنین در بازی GTA: Vice City هواپیمای skimmer واقعاً به‌عنوان یک قایق در دیتای بازی وجود داشت، ولی در بازی GTA: San Andreas این هواپیما از قایق تبدیل به هواپیما شده‌است! اما از شانس بد برنامه‌نویسان rockstar و از آن‌جایی که skimmer یک هواپیماست که چرخ ندارد، خالی گذاشتن فیلدهای مربوط به چرخ‌ها باعث شده‌است که در کد بازی رفتار نامشخص[1] رخ دهد و مقدار bounding box اشتباه شود که باعث می‌شود هواپیما در ارتفاع خیلی زیاد ساخته شود. در صورتی که مقادیر مربوط به چرخ‌های یک هواپیمای دیگر را به skimmer بدهیم، می‌بینیم که skimmer دوباره به بازی بر می‌گردد و بدون هیچ مشکلی می‌توان با آن پرواز کرد!

اما یک سؤال مهم دیگر در این‌جا وجود دارد: چرا این باگ زمانی خودش را نشان داد که ویندوز ۱۱ نسخهٔ 24H2 خودش را منتشر کرد و در ۲۱ سال گذشته خودش را نشان نداده بود؟ برای این کار باید در ابتدا کدی که مربوط به خواندن اطلاعات چرخ‌ها از فایل vehicles.ide است را نگاه کنیم. این کد به‌صورت زیر است:

```
void CFileLoader::LoadVehicleObject(const char* line){
  int objID = -1;
  char modelName[24];
  char texName[24];
  char type[8];
  char handlingID[16];
  char gameName[32];
  char anims[16];
  char vehClass[16];
  int frq;
  int flags;
  int comprules;
  int wheelModelID;// Uninitialized!
  float frontWheelScale, rearWheelScale;// Uninitialized!
  int wheelUpgradeClass = -1;// Funny enough, this one IS initialized
  sscanf(line,
             "%d %s %s %s %s %s %s %s %d %d %x %d %f %f %d",
             &objID, modelName, texName, type, handlingID,
             gameName, anims, vehClass, &frq, &flags,
             &comprules, &wheelModelID, &frontWheelScale,
             &rearWheelScale,
             &wheelUpgradeClass);

  // More processing here...
}
```

همان‌طور که مشخص است، متغیرهای frontWheelScale و rearWheelScale مقداردهی اولیه نشده‌اند و از آن‌جا که از تابع sscanf برای خواندن این اعداد استفاده می‌شود، در صورتی که عددی در رشته وجود نداشته باشد مقدار متغیر پاس‌داده‌شده به sscanf نیز عوض نمی‌شود. اما هنوز این سؤال مطرح است که چرا فقط در آخرین نسخهٔ ویندوز ۱۱ این اعداد به صورتی مقداردهی می‌شوند که بازی نمی‌تواند هواپیمای skimmer را بسازد. به‌عنوان یک فرض اولیه می‌توان به این موضوع فکر کرد که شاید ویندوز کتابخانهٔ libc که شامل توابعی مانند printf و fopen و sscanf است را طوری عوض کرده‌است که باعث بشود این تغییر اتفاق بیفتد، اما نکته‌ای که وجود دارد این است که GTA: SA یک برنامهٔ statically linked است و از کتابخانهٔ libc خود سیستم‌عامل استفاده نمی‌کند؛ بلکه کتابخانهٔ libc مربوط به خودش را دارد که بین نسخه‌های مختلف ویندوز تغییر نمی‌کند. پس مشکل از جای دیگری است. در ادامه، اگر نگاه کنیم که چه تابعی LoadVehicleObject را صدا می‌زند به تابع زیر می‌رسیم:

```
void CFileLoader::LoadObjectTypes(const char* filename){
  // Open the file...
  while ((line = fgets(file)) ≠ NULL){
    // Parse the section indicators...
    switch (section){
      // Different sections...
    case SECTION_CARS:
      LoadVehicleObject(line);
      break;
    }
  }
}
```

همان‌طور که مشاهده می‌کنید، در یک حلقه هر خط از فایل خوانده می‌شود و سپس تابع LoadVehicleObject صدا زده می‌شود. این کد و نحوهٔ صدا شدن توابع باعث می‌شود که متغیرهای frontWheelScale و rearWheelScale از اجرای تابع قبلی به این تابع منتقل شوند. دلیل این موضوع این است که زمانی که یک تابع تمام می‌شود، مقادیر داخل استک آن پاک (صفر) نمی‌شوند و صرفاً stack pointer جابه‌جا می‌شود. این موضوع باعث می‌شود که اگر دوباره همان تابع را فوراً بعد از تمام شدن آن صدا کنیم، تمامی متغیرهای آن تابع مقدار اولیهٔ متغیرهای فراخوانی قبلی تابع را داشته باشند. اما مشخص است که اگر تابع دیگری بین این دو فراخوانی اجرا شود، استک به‌جامانده تابع اول را overwrite می‌کند. در بازی GTA نیز بین هر صدا زدن تابع LoadVehicleObject یک بار تابع fgets صدا زده می‌شود. پس می‌توان حدس زد که در نسخه‌های قبلی ویندوز تابع fgets استفادهٔ زیادی از استک نداشته و به همین دلیل مقدار frontWheelScale و rearWheelScale از فراخوانی قبلی تابع به جا می‌مانند. به همین منظور در صورتی که مقدار این دو متغیر را در نسخه‌های قبلی ویندوز نگاه کنید، به عدد 0.7 می‌رسید، چرا که دقیقاً در خط قبلی فایل که خوانده شده‌است، اطلاعات مربوط به ماشین Top Fun وجود دارد که rearWheelScale آن برابر 0.7 است! پس قطعاً این عدد از فراخوانی قبلی به جا مانده‌است.

حال باید بررسی کنیم که در نسخهٔ 24H4 ویندوز چه اتفاقی می‌افتد که این عدد خراب می‌شود. برای این کار می‌توان یک hardware breakpoint بر روی متغیر درون استک قرار داد تا زمانی که آن نوشته می‌شود، دیباگر برنامه را متوقف کند. زمانی که این کار را می‌کنیم، متوجه می‌شویم که دیباگر در فراخوانی تابع LeaveCriticalSection که یک تابع مخصوص سیستم‌عامل است، متوقف می‌شود. این بدین معناست که پیاده‌سازی LeaveCriticalSection در ویندوز 11 نسخهٔ 24H4 طوری عوض شده‌است که از استک بیشتری استفاده می‌کند و در نتیجه متغیرهایی که از فراخوانی قبلی تابع در استک به جا مانده‌اند، overwrite می‌شوند. می‌توانید نحوهٔ صدا زده شدن توابع را در ادامه مشاهده کنید:

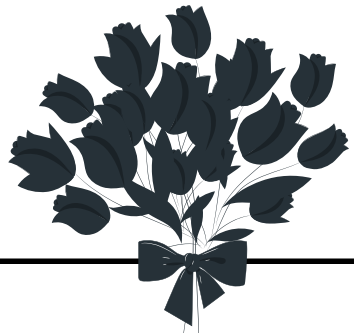
```
> ntdll.dll!_RtlpAbFindLockEntry@4() Unknown
  ntdll.dll!_RtlAbPostRelease@8() Unknown
  ntdll.dll!_RtlLeaveCriticalSection@4() Unknown
  gta_sa.exe!fgets() Unknown
```

شاید جالب باشد اگر بدانید که GTA: SA تنها بازی‌ای نیست که این مشکل گریبان‌گیرش شده‌است. بلکه بازی قدیمی Sid Meier's Alpha Centauri نیز (دقیقاً به همین دلیل که در نسخه‌های جدیدتر ویندوز LeaveCriticalSection فضای بیشتری از استک را اشغال می‌کند) کرش می‌کند! در نهایت نیز یک پیشنهادی که می‌توانم به شما بکنم این است که حتماً حتماً حواستان به رفتارهای نامشخص از این قبیل باشد و حتماً در صورتی که کامپایلر به شما اخطار[1] داد، این اخطار را نادیده نگیرید و به حرف آن گوش کنید!

۱. warning
۲. منبع

به یاد هم دانشکده‌ای عزیزمان

عرفان اسدی زیدآبادی



رضا وحیدی
کارشناسی ۱۳۹۹

من وقتی خبر فوت عرفان رو شنیدم، اصلاً باورم نشد و خشکم زد. عرفان رو من از ترم دوم و پروژهٔ AP می‌شناختم که هم‌تیمی بودیم و اولین دوست دانشگاهم بود. بیشتر از همه‌چی، مهربونی و خوش قلب بودنش بود که شخصیتش رو دوست‌داشتنی می‌کرد و یه جورایی از بقیه متمایزش می‌کرد. تا آخر شب اون روز تمام خاطره‌های خوبی که با هم داشتیم تو ذهنم مرور می‌شد. از پیاده‌روی، سالن، بردگیم، اتاق فرار و دور دور کردن شب‌ها تا هم‌فکری کردن تو حل تمرین و پروژه‌ها.
هنوزم که هنوزه غمش از بین نرفته و بعضی وقت‌ها که بهش فکر می‌کنم یا با دوستامون دربارهٔ خاطرهٔ مشترکی که با عرفان داشتیم صحبت می‌کنیم، خیلی دلتنگش می‌شیم.
امیدوارم روحش شاد باشه.

علی جهانشاهی
کارشناسی ۱۳۹۹

اون که رفته دیگه هیچ وقت نمیاد...
اولش که خبر رو می‌شنوی فقط از خدا میخوای یکی بهت بگه شوخی بوده.
دوست داری سرتو بزاری رو بالش بعد که بیدار شدی بفهمی خواب بوده، ولی نبود. الان بیش از یک ماه میگذره که هرروز داره این کابوس تکرار میشه.
منتظری شمارش دوباره رو گوشیت بیوفته فقط یه بار دیگه صداشو بشنوی.
با خودت میگی چکار کنم تموم شه این کابوس.

هومان کشوری
کارشناسی ۱۳۹۹

من عرفانو از سال اول دانشگاه می‌شناختم و تو درسایی با هم هم‌تیمی بودیم و حتی جزو نفرات اولی بود که حضوری دیدمش.
(سال ما کرونا بود و دو سال اول مجازی بود)
خیلی بچهٔ خوب و خوش‌ذوق و خوش‌قلبی بود و خیلی هم بامرام بود.
کلاً شنیدن این خبر نه‌تنها برای من، بلکه برای کسایی که یه ارتباط ریز با عرفان داشتن هم خیلی سنگین و تلخ بود.

عرفان مجیبی
کارشناسی ۱۳۹۹

ولیک دوست بگرید به زاری از پی دوست
اگرچه باز نگردد به گریهٔ زارش

غمی رسید به روی زمانه از تقدیر
که پشت طاقت گردون دو تا کند بارش

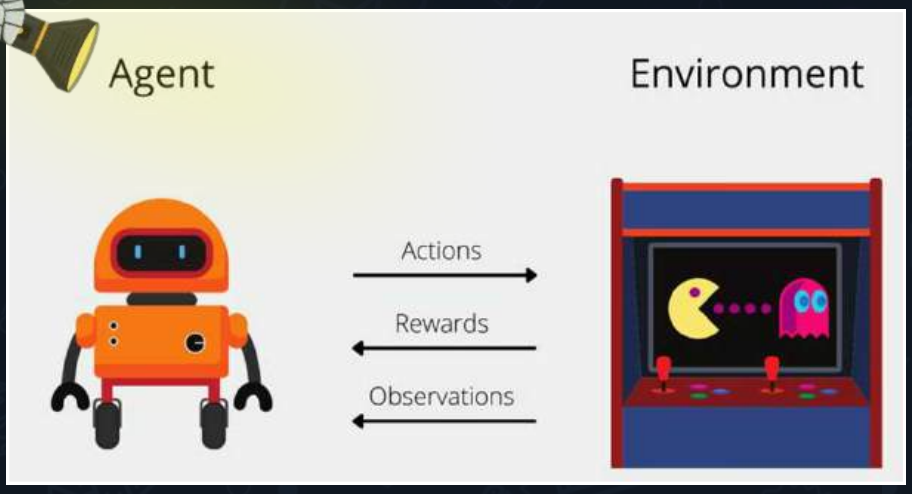
کاش به‌جای کلمه می‌شد اشک‌ها را نوشت؛ لااقل حق غمت ادا می‌شد.

باورم نمی‌شود دارم به یاد عرفان می‌نویسم. برای به یاد او نوشتن خیلی زود است. ما باید هنوز این زندگی را با دوستی‌هامان زندگی می‌کردیم و خاطره می‌ساختیم؛ نه این‌که او برود و ما بمانیم.
عرفان نزدیک فارغ‌التحصیلی بود و مثل همهٔ ما سال آخری‌ها به جستجوی مسیر پیش رویش مشغول بود. با هم که حرف می‌زدیم، امید و ایده و برنامه داشت. من واقعاً برایش خوشحال بودم و امیدوار بودم که روزهای بهتری پیش رویش است؛ روزهایی که به خواسته‌هایش برسد و با علایقش زندگی کند... اما چه بی‌لیاقت بود دنیا که از همهٔ این‌ها بی‌بهره ماند...
عرفان خودش بود؛ همان که می‌نمود. صورتک به رو نداشت و صاف و ساده بود. در این دنیای امروز ما «خودت بودن» هزینهٔ زیادی دارد و رنج زندگی را چندین برابر می‌کند. از ته دل باور دارم خودش بود و خودش ماند و نتوانست به این دنیای «بی‌خود» خو بگیرد. اما حالا امیدوارم جای بهتری باشد؛ در آرامشی به دور از سختی زندگی بی‌مایهٔ ما آدم‌های معمولی.

«رفتی و آدمکا رو جا گذاشتی، قانون جنگلو زیر پا گذاشتی،
این جا قهرن سینه‌ها با مهربونی، تو توی جنگل نمی‌تونستی بمونی،
دلتو بردی با خود به جای دیگه، اون جا که خدا برات لالایی میگه،
می‌دونم می‌بینمت یه روز دوباره، توی دنیایی که آدمک نداره.»

مهدی صابر
کارشناسی ۱۳۹۹

من با عرفان سه سال پیش آشنا شدم، به‌واسطهٔ دوستان مشترکی که داشتیم. بر خلاف اکثر آدمای امروزی، عرفان از همون روز اول خوش‌قلبی‌شو بروز می‌داد. همیشه دوست داشت کنار بقیه باشه و بهشون کمک کنه. یکی از واقعی‌ترین و مهربون‌ترین افرادی که دیدم.
متنوع بودن علایقش به زمینه‌های متفاوت و علوم مختلف هم در نوع خودش جالب بود. کامپیوتر می‌خوند، ولی دلیل بر این نبود که از بیو و نوروساینس هم لذت نبره.
روزی که خبر فوتشو شنیدم، فکر می‌کردم دارم یه شوخی بد رو می‌شنوم و تا یه مدت زیادی باورم نشد. روزای جنگ بارها حال بچه‌ها رو پرسید و دعوت به خونه‌ش (که امن و دور از خطر بود) می‌کرد. دلش نمی‌خواست هیچ اتفاقی برای کسی بیفته و این خیلی پذیرفتن اتفاقی که افتاده رو برام سخت‌تر می‌کرد. حتی هنوزم کامل باورم نشده که چطور چنین آدم پاک‌دلی انقدر زود از بینمون رفته... امیدوارم روحش شاد باشه.



یادگیری ماشین

عوامل یادگیر این قابلیت را دارند که با یادگیری از تجربیات خود، عملکرد خود را در طول زمان بهبود بخشند.
آن‌ها شامل مؤلفه‌هایی برای یادگیری و عملکرد هستند که به آن‌ها امکان می دهد با موقعیت‌های جدید سازگار شوند و تصمیم‌گیری خود را بهبود بخشند.
در یادگیری ماشین هدف این است که رایانه‌ها و سامانه‌ها بتوانند به‌تدریج و با افزایش داده‌ها کارایی بهتری در انجام وظیفهٔ مورد نظر پیدا کنند.

الگوریتم‌های یادگیری ماشین بر سه نوع هستند:

- یادگیری تقویتی[1]
- یادگیری نظارت نشده[2]
- یادگیری نظارت شده[3]

یادگیری تقویتی

یادگیری تقویتی را می‌توان به‌عنوان یک حلقهٔ متشکل از اجزای زیر در نظر گرفت:

- عامل[4]: یادگیرنده یا تصمیم‌گیرنده‌ای که براساس مشاهده‌های خود اقداماتی را انجام می‌دهد.
- محیط[5]: سیستم یا زمینهٔ خارجی که عامل در آن عمل می‌کند.
- حالت[6]: پیکربندی یا نمایش فعلی محیط در یک زمان معین.
- اقدام[7]: تصمیم یا انتخابی که عامل در پاسخ به یک حالت اتخاذ می‌کند.
- پاداش[8]: سیگنال بازخوردی که خوبی یا مطلوبیت عمل عامل را ارزیابی می‌کند.
- سیاست[9]: استراتژی یا رویکردی که عامل برای انتخاب اقدامات براساس حالت‌های مشاهده‌شده به‌کار می‌گیرد.

بیایید مثال واقعی یادگیری حرف زدن یک نوزاد را با هم بررسی کنیم:
در ابتدا، یک نوزاد صداهای مختلف را آزمایش می‌کند. نوزاد مطمئن نیست که کدام صداها پاسخ مراقبین را برانگیزد، هنگامی که کودک کلمهٔ قابل‌تشخیصی را می‌گوید، مراقبان با لبخند، تشویق و تمجید کلامی، پاسخ مثبت می‌دهند. این پاداش‌ها کودک را تشویق می‌کند که به تمرین و یادگیری کلمات جدید ادامه دهد. نوزاد ممکن است وقتی درک نمی‌شود، با لحظاتی از ناامیدی مواجه شود (تنبیه).
حال که نوزاد یک شبکهٔ معنایی (کلمات معنادار) از کلمات به‌دست آورده، شروع به ترکیب کلمات برای تشکیل جملات اساسی می کند (به‌عنوان مثال «شیر می‌خوام»).
نوزاد تعاملات و بازخوردهای اجتماعی پیچیده‌تری را تجربه می کند. والدین با گفتن ترکیبات درست، آن ترکیبات را در ذهن نوزاد وارد می‌کنند و همین‌طور رفته‌رفته کودک تقویت می‌شود و سخنان پیچیده‌تری می‌گوید. جالب است به این توجه کنید که کودک در ادامه بدون دانستن واقعی دستور زبان[10] می‌تواند بر اساس مثال‌های اطرافیانش جمله هایی با گرامر درست تولید کند.

یادگیری نظارت‌نشده

درک این مورد برای ما بسیار سخت است؛ مگر می‌شود بدون این‌که دادهٔ برچسب‌داری داشته باشیم، بفهمیم دادهٔ جدیدی که به ما می‌دهند چیست؟
در این حالت الگوها را منحصراً از داده‌های بدون برچسب یاد می‌گیرند. در چنین رویکردی، یک مدل یادگیری ماشین سعی می‌کند هر شباهت، تفاوت، الگو و ساختار در داده‌ها را به‌تنهایی پیدا کند و هیچ مداخلهٔ انسانی قبلی لازم نیست؛ برای مثال، بدون این‌که کسی اطلاعاتی از دادهٔ بعدی به شما بدهد، می‌توانید عدد بعدی را حدس بزنید.
۸،۶،۴،۲ و...
اما شناخت این الگوها در مغز بسیار سخت است و در واقع کار روانشناس‌ها پیدا کردن این الگوهای فکری بد و درست کردن آن‌ها از طریق یادگیری نظارت‌شده است.

یادگیری نظارت‌شده

یادگیری نظارت‌شده نوعی از یادگیری ماشین است که در آن یک الگوریتم کامپیوتری می‌آموزد که بر اساس داده‌های برچسب‌گذاری‌شده پیش‌بینی یا تصمیم‌گیری کند. داده‌های برچسب‌گذاری‌شده از متغیرهای ورودی شناخته‌شدهٔ قبلی (یا همان ویژگی‌ها) و متغیرهای خروجی (یا همان برچسب‌ها) تشکیل شده‌اند.

۱. Reinforcement Learning (RL)
۲. Unsupervised Learning
۳. Supervised Learning
۴. Agent
۵. Environment
۶. State
۷. Action
۸. Reward
۹. Policy
۱۰. grammer

با تجزیه و تحلیل الگوها و روابط بین متغیرهای ورودی و خروجی در داده‌های برچسب‌دار، الگوریتم یاد می‌گیرد که پیش‌بینی کند. برای مثال تشخیص این که شما لبخند می‌زنید یا نه، با استفاده از این روش انجام می‌شود. بیایید نمونه‌ای از نحوهٔ یادگیری الفبا به کودکان (A، B، C، D) را از طریق یادگیری نظارت‌شده بررسی کنیم. کودکان شکل الفبا را می‌بینند؛ این‌جا دیگر پاداشی نیست و فقط نوعی از تنبیه را داریم، مادر یا معلم شکل حرف A را به کودک نشان می‌دهد و هم‌زمان به او می‌گوید A (صدای A)، حال کودک یک تصویر از A دیده است؛ پس از دیدن همهٔ حروف، به کودکان فرصت‌های متعددی داده می‌شود تا حروف را تشخیص دهند.

تبدیل یک عکس به یک حرف در واقع برچسب دادن به عکس است و ما برای تشخیص این حروف باید ده‌ها عکس با برچسب A ببینیم تا بتوانیم حرف A را تشخیص دهیم.

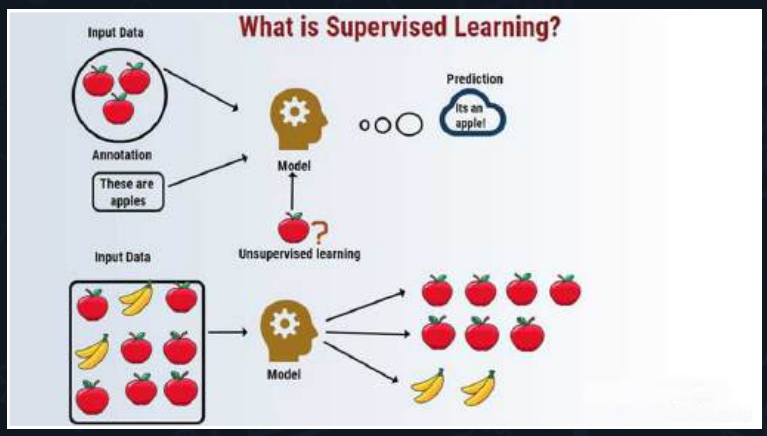
این فرایند برای ما ملموس نیست، اما محاسباتی که مغز برای انجام این کار انجام می‌دهد، بسیار پیچیده و زیاد هستند (ممکن است هر بار یک میلیارد عدد در یک میلیارد عدد ضرب شوند). تبدیل یک عکس به صدا هم فرایندی هست که ما در کلاس با دیدن عکس و صدای آن (برچسب) آموخته‌ایم.

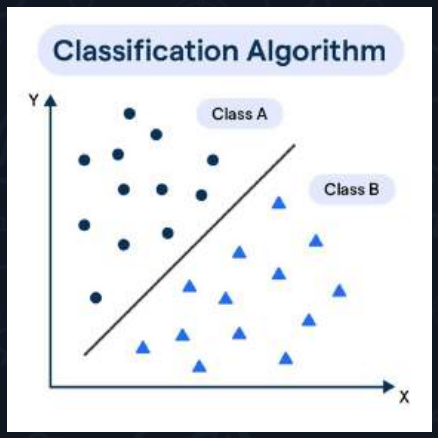
برای جمع‌بندی، چون اقتصاد موضوعی‌ست که جذابیت زیادی دارد، پس مثال اقتصادی می‌زنم؛ فرض کنید می‌خواهید مدلی بسازید که قیمت خانه‌ها در تهران را بر اساس ویژگی‌های مختلف آن‌ها پیش‌بینی کند.

داده‌های مربوط به خانه‌های فروخته‌شده در تهران را جمع‌آوری کنید. این داده‌ها شامل ویژگی‌های مختلف هر خانه و قیمت فروش آن‌ها هستند.

برای این کار می‌توانیم سایت دیوار را بگردیم و تمامی داده‌ها را جمع‌آوری کنیم. حالا از همان سایت دیوار، خانهٔ جدیدی در تهران پیدا می‌کنیم که قیمت ندارد. اکنون می‌خواهیم ببینیم که چگونه می‌توان یک قیمت تقریبی برای این خانه پیدا کرد.

در یک منطقه معمولاً قیمت خانه‌ها بر حسب متراژ بالا رفته و برای ساده‌سازی از بقیهٔ ویژگی ها صرف‌نظر می‌کنیم. برای مثال، یک خانهٔ ۴۰ متری، ۱۰ میلیارد؛ یک خانهٔ ۶۰ متری، ۱۵ میلیارد و همین‌طور بالا می‌رود. این روند در ذهن ما منطقی‌ست، اما به‌دست آوردن شیب دقیق بالا رفتن قیمت با متراژ باید بر حسب داده‌ها انجام شود و ما در واقع باید بهترین خطی که می‌تواند این شیب را داشته باشد پیدا کنیم. به نظر شما ملاک ما برای پیدا کردن این خط چیست؟ منطقی نیست بگوییم هر چه متراژ بیشتر، قیمت بیشتر؛ ما باید بگوییم به ازای هر متر خانه، قیمت خانه تقریباً چقدر اضافه می‌شود.





(دقت کنید که این عدد در یک ساختمان خاص نیست و در ساختمان‌های مختلف با قیمت‌های مختلف به‌ازای متراژ است.)

یکی از ملاک‌هایی که ما در یادگیری نظارت‌شده به‌کار می‌بریم، فاصلهٔ همهٔ نقاط از خط است؛ یعنی ما ابتدا یک خط فرضی اولیه را طوری تغییر دهیم که فاصلهٔ خط تا نقاط کمترین مقدار ممکن شود؛ ابتدا یک خط رسم می‌کنیم؛ حالا دو تصمیم می‌توانیم بگیریم؛ این‌که آن را ساعتگرد بچرخانیم یا پادساعتگرد و بعد از محاسبات می‌بینیم که پادساعتگرد انتخاب بهتری‌ست؛ کمی می‌چرخانیم و دوباره محاسبه می‌کنیم و این‌کار را آن‌قدر ادامه می‌دهیم که دیگر چرخش ما موجب بهبود نشود. این روش یک روش به‌روزرسانی مرحله‌به‌مرحله است، ولی راه‌هایی وجود دارند که نخواهیم مرحله‌به‌مرحله پیش برویم و بتوانیم با فرایند مشتق گرفتن خط اصلی را پیدا کنیم.

حال اگر ویژگی‌های دیگر را نیز در نظر بگیریم، به همان میزان بُعد اضافه می‌شود و این نقاط در فضا در بعدهای مختلف از هم دور می‌شوند و باید دقت شود که تأثیر سال ساخت با تأثیر متراژ شیب یکسانی ندارد.

پس ما باید به هر یک از این داده‌ها یک ضریب بدهیم که به آن میزان، تأثیر هر یک از این ویژگی‌ها محاسبه شود. برای مسائل مختلف نیز یادگیری همین‌گونه است.

دستهٔ دیگر از یادگیری نظارت‌شده به طبقه‌بندی[1] معروف است. در این روش برچسب داده‌ها، داده‌ها را به دو یا چند دسته تقسیم می‌کنند و هدف ما یادگرفتن مرز بین داده ها با توجه به ویژگی‌های داده است؛ برای مثال، تقسیم ایمیل‌ها به دو دستهٔ اسپم (مزاحم-تبلیغاتی) یا غیراسپم؛ یا همان مثال تشخیص حروف الفبا؛ یا تشخیص چهرهٔ افراد با توجه به تصاویری که از آن‌ها داریم.

اما چالش این‌جاست که در خیلی از مسائل دنیای واقعی ما از یک نقطه شروع نمی‌کنیم؛ برای مثال در همین بحث طبقه‌بندی، ما یک عکس ورودی می‌دهیم و هر عکس هزاران پیکسل دارد که آن عکس را ساخته‌اند و ما طبق همهٔ این هزار پیکسل باید تصمیم بگیریم و اگر بخواهیم از یک گوشهٔ تصویر شروع کنیم و جلو برویم و تصمیم بگیریم، کار بسیار سخت و چه‌بسا ناممکن می‌شود و باید میلیاردها درخت را بسازیم.

حال بیایید ببینیم دانشمندان چگونه از نورون مغزی ایده گرفته‌اند تا بتوانند شبکه‌هایی بسازند که این کار طبقه‌بندی را انجام دهند.

ادامهٔ این متن در مورد شبکه‌های عصبی را می‌توانید در کانال تلگرام مرحوم عرفان اسدی مطالعه کنید.

https://t.me/computationalpsychology

MACHINE LEARNING

نگاه دیگری برای یادگیری از داده‌ها


عارف شه‌بخش
کارشناسی ارشد ۱۴۰۰

در دنیای هوش مصنوعی، معمولاً با داده‌هایی روبه‌رو هستیم که در قالب جدول، تصویر یا متن سازمان‌دهی شده‌اند. اما اطلاعات ما همیشه این‌قدر ساختارمند نیستند. مولکول‌ها، شبکه‌های اجتماعی یا گراف‌های استنادی؛ این‌ها داده‌هایی هستند که در قالب ساختارهای پیچیده و درهم‌تنیده قرار دارند. این نوع داده‌ها به‌علت وجود روابط غیر اقلیدسی و نبود روابط هندسی منظم (مانند ترتیب چپ به راست یا روابط فضایی، مثل تصاویر یا جملات) قابل‌استفاده در شبکه‌هایی مانند شبکه‌های چندلایهٔ پرسپترون یا شبکه‌های پیچشی (کانولوشن) و شبکه‌های عصبی برگشتی نیستند. به‌عبارتی این شبکه‌ها با فرض روابط اقلیدسی کار می‌کنند و برای یادگیری مدل از این نوع داده‌ها، به ابزاری قدرتمند نیاز داریم که بتواند این نوع از اطلاعات را به‌خوبی پردازش کند و آن، چیزی نیست جز شبکه های عصبی گرافی[1].

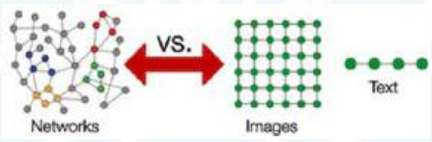
داده‌ساختار گراف

هر گراف از گره‌ها[2] (یا رئوس) و یال‌ها[3] (یا لبه‌ها) تشکیل شده‌است. تصور کنید اتم‌ها گره‌های یک گراف باشند و پیوندهای شیمیایی، یال‌های آن. یا در یک شبکهٔ اجتماعی، افراد گره‌های گراف و روابط دوستی میان آن‌ها یال‌های گراف باشند. نکتهٔ مهم این است که هر گره و یال می‌تواند دارای ویژگی‌هایی[4] باشد. مثلاً یک گره ممکن است نشان‌دهندهٔ یک اتم کربن باشد؛ با ویژگی‌هایی مثل «عدد اتمی: ۶» یا «الکترونگاتیوی: ۲٫۵۵». این ویژگی‌های گره‌ها، یال‌ها و ساختار گراف می‌توانند نقطهٔ شروع الگوریتم‌های یادگیری[5] شبکه‌های عصبی گرافی باشند.

چرا شبکه‌های عصبی معمولی کارایی ندارند؟

مدل‌های سنتی یادگیری عمیق[6] انتظار دارند ورودی‌ها بر پایهٔ یک فضای هندسی منظم باشند (مثل یک تصویر ۲۸×۲۸ پیکسلی یا یک جملهٔ همین نوشته!)، ولی گراف‌ها در ساختار، بسیار متنوع هستند، آن‌ها می‌توانند به هر شکل پیچیده‌ای باشند (شکل ۱).

بدتر این‌که گراف‌ها غیر اقلیدسی هستند؛ یعنی نمی‌توان برایشان ترتیب طبیعی مثل ترتیب چپ به راست تعیین کرد. برای یادگیری مدل این نوع داده‌ها به شبکه‌ای نیاز داریم که بتواند بدون از بین بردن روابط غیر اقلیدسی از این داده‌ها یاد بگیرد. در این‌جا شبکه های عصبی گرافی وارد می‌شوند و با توجه به روابط بین گره‌ها، ویژگی‌های گره‌ها یا یال‌ها و ساختار گراف، شروع به یادگیری گراف می‌کنند.



شکل ۱: مقایسهٔ داده‌های ساختارمند -مانند تصاویر و جملات- با داده‌های گرافی بدون ساختار

شبکه‌های عصبی گرافی چگونه از اطلاعات یاد می‌گیرند؟

در قلب شبکه‌های عصبی گرافی فرآیندی به نام انتشار پیام[7] قرار دارد؛ هر گره، اطلاعاتی به نام پیام را از همسایگانش دریافت کرده و به آن‌ها ارسال می‌کند. این پیام‌ها بر اساس ویژگی‌های گره‌ها یا ویژگی‌های یال‌ها یا ساختار گراف ساخته می‌شوند. سپس گره‌ها یا یال‌ها این پیام‌ها را تجمیع کرده و وضعیت خود را به‌روزرسانی می‌کنند. مثلاً یک اتم اکسیژن می‌تواند از هیدروژن‌های متصل به خود اطلاعات بگیرد. این فرایند در چندین مرحله انجام می‌شود تا هر گره یا یال اطلاعاتی غنی و مبتنی بر همسایگان خود را به‌دست آورد.

یک مثال ملموس: پیش‌بینی سمی بودن یک مولکول

فرض کنید می‌خواهیم میزان سمی بودن یک مولکول را پیش‌بینی کنیم. این مولکول‌ها را به‌صورت گراف نمایش می‌دهیم: اتم‌ها گره‌های گراف‌اند، با ویژگی‌هایی مثل جرم اتمی یا هیبریداسیون؛ که این ویژگی‌ها به‌صورت بردار تعبیه[8] نمایش داده می‌شوند و پیوندهای کوالانسی، یال‌های گراف‌اند؛ با ویژگی‌هایی مثل نوع پیوند (پیوند یگانه یا دوگانه). شبکه‌ی عصبی گرافی با بردار تعبیهٔ ویژگی‌های اتم‌ها شروع به یادگیری می‌کنند. ابتدا هر گره، بردارهای تعبیهٔ گره‌های همسایگان خود را با بردار تعبیهٔ خودش جمع می‌کند و این بردار حاصل‌شده را با بردار تعبیهٔ خودش جایگزین می‌کند؛ فرایندی که انتشار پیام نام دارد. البته برای مسئله‌هایی مانند پیش‌بینی یال بین دو گره می‌توان از بردارهای تعبیهٔ یال‌ها استفاده کرد. سپس با انجام چندبارهٔ فرایند انتشار پیام، اتم‌ها اطلاعاتشان را با یکدیگر به اشتراک می‌گذارند. در نهایت مدل نیز می‌تواند بردارهای تعبیهٔ به‌روزشدهٔ گره‌ها را با هم ترکیب کند (مثل عمل میانگین‌گیری بین هر ویژگی بردارها) و به برداری از کل ساختار گراف برسد که می‌تواند برای پیش‌بینی «سمی» یا «غیرسمی» بودن در الگوریتم‌های دسته‌بندی یادگیری ماشین به‌عنوان ورودی استفاده شود.

نگاهی دقیق‌تر به ویژگی‌های گره‌ها

ویژگی‌های گره همان دانشی است که شبکه‌های عصبی گرافی در ابتدا در اختیار دارند. در مولکول‌ها این ویژگی‌ها ممکن است شامل عدد اتمی یا ظرفیت پیوندی باشند. یال‌ها هم می‌توانند ویژگی‌هایی مثل نوع پیوند (تکی/دوتایی) داشته باشند. این ویژگی‌ها به‌صورت بردارهای عددی نمایش داده می‌شوند. با پیشروی انتشار پیام، بردار بازنمایی هر گره از همسایه‌های خود آگاه می‌شود؛ یعنی نه‌تنها خود را می‌شناسد، بلکه نقش خود در کل ساختار گراف را نیز درک می‌کند.

ریاضی در پشت صحنه

فرمول پایه‌ای برای به‌روزرسانی هر گرهٔ v به صورت زیر است:

$$h_v^{(k)} = \text{UPDATE}^{(k)}\left(h_v^{(k-1)}, \text{AGGREGATE}^{(k)}\left(\{h_u^{(k-1)} : u \in N(v)\}\right)\right)$$

در این‌جا $h_v^{(k)}$ بازنمایی گره در لایهٔ k است و $N(v)$ همسایگان آن. تابع AGGREGATE پیام‌ها را جمع‌آوری می‌کند (مثلاً با میانگین‌گیری یا جمع) و تابع UPDATE یک شبکهٔ چندلایهٔ پرسپترون (از لحاظ تئوری می‌توانیم از هر شبکه مشتق‌پذیر استفاده کنیم) است. این شبکه بردار به‌روزرسانی‌شدهٔ هر گره را (X) به صورت ترکیب خطی وزن‌دار با وزن‌هایی برای

۱. Graph Neural Network (GNN)
۲. Node
۳. Edge
۴. Feature
۵. Learning
۶. Deep learning
۷. Message Passing
۸. Embedding

هر ستون بردار (AX + B) در می‌آورد که با برگشت گرادیان این وزن‌ها به‌روزرسانی می‌شوند. این رابطهٔ بازگشتی اجازه می‌دهد که شبکه‌های عصبی گرافی، دانشی چندمرحله‌ای از ساختار گراف را به‌دست آورند.

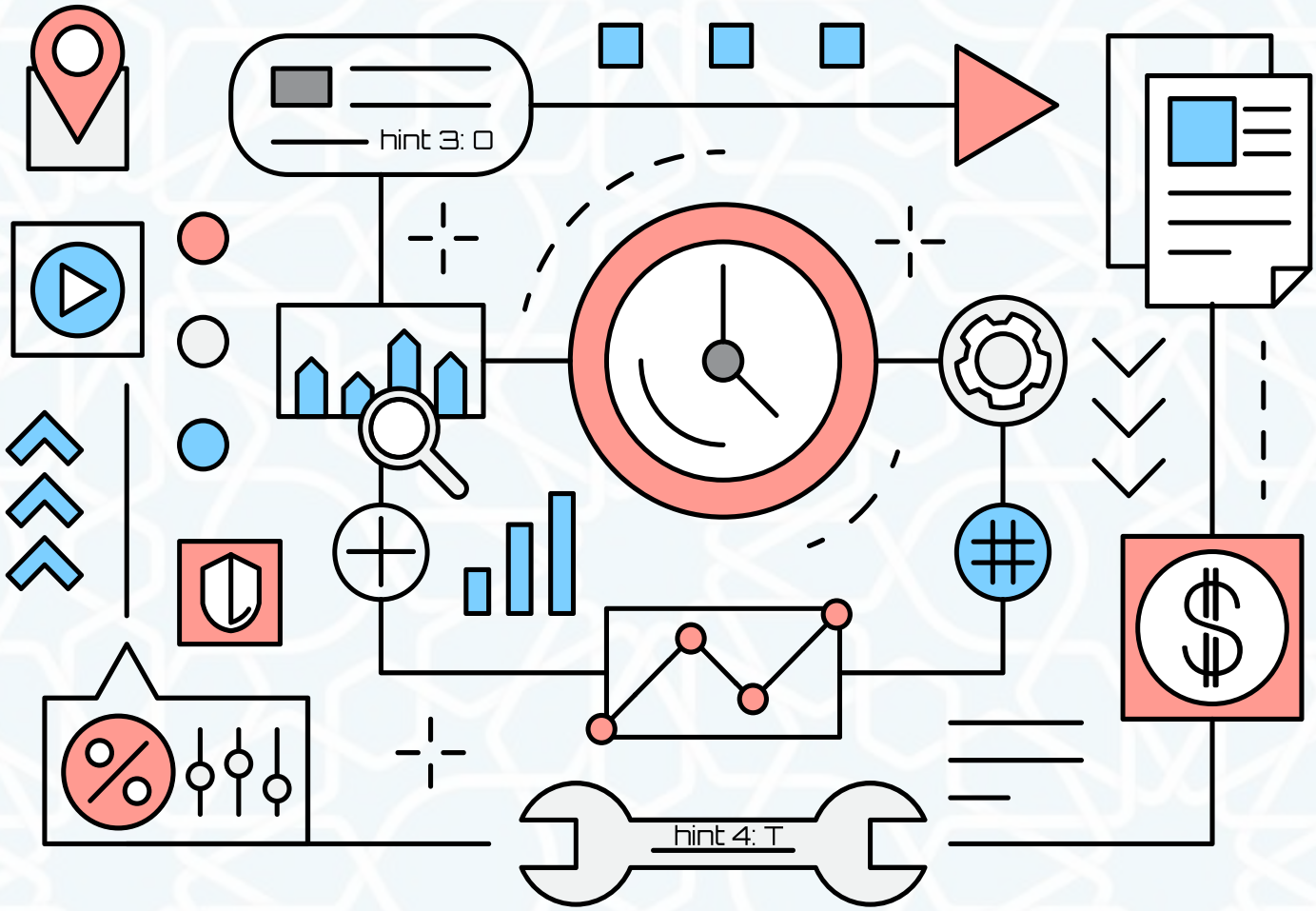
شبکه‌های عصبی گرافی به مفاهیم تئوری محدود نمی‌شوند، آن‌ها در بسیاری از حوزه‌های عملی و دنیای واقعی مورد استفاده قرار می‌گیرند. به‌عنوان مثال در AlphaFold -یک پیش‌بینی‌کنندهٔ ساختار پروتئین انقلابی- شبکه‌های عصبی گرافی به مدل‌سازی نحوهٔ تعامل اسیدهای آمینه کمک می‌کنند. در شبکه‌های اجتماعی، پلتفرم‌هایی مانند فیس‌بوک و لینکدین از شبکه‌های عصبی گرافی برای پیشنهاد اضافه کردن افراد به‌عنوان دوست یا شناسایی حساب‌های جعلی استفاده می‌کنند. حتی موتور جست‌وجوی گوگل نیز برای درک بهتر روابط صفحات وب، از شبکه‌های عصبی گرافی استفاده می‌کند.

چالش‌ها و مسیر آینده

با وجود قدرت بالا، شبکه‌های عصبی گرافی با چالش‌هایی نیز روبه‌رو هستند؛ آن‌ها می‌توانند برای گراف‌های خیلی بزرگ هزینهٔ پردازشی بالایی داشته باشند و عملکردشان به نحوهٔ تعریف گراف بسیار وابسته است. به‌طور مثال می‌توان به سؤالاتی چون «چه چیزی همسایه محسوب می‌شود؟» و «چه ویژگی‌هایی مهم‌ترند؟» اشاره کرد. پژوهشگران مشغول کار روی افزایش مقیاس‌پذیری[1]، مکانیزم‌های توجه[2] و قابلیت تفسیرپذیری[3] هستند، تا شبکه‌های عصبی گرافی را بهتر و قابل‌فهم‌تر کنند.

چرا این موضوع برای شما اهمیت دارد؟

چه دانشجوی زیست‌شناسی باشید و به کشف دارو علاقه‌مند باشید، چه دانشجوی علوم کامپیوتر باشید و در حال یادگیری مدل‌های جدید؛ شبکه‌های عصبی گرافی افق جدیدی در هوش مصنوعی می‌گشایند. زیرا آن‌ها داده‌ها را نه به‌صورت مسطح و ساده، بلکه پیچیده، متصل و پویا می‌بینند و امکان یادگیری مدل از داده‌های غیر اقلیدسی را فراهم می‌کنند، و شاید همین توانایی باشد که راه را برای کشفیات علمی آینده باز می‌کند.



۱. Scalability
۲. Attention
۳. Interpretability
۴. مرجع

فایل‌سیستمی بدون محدودیت تعداد inode


نیما شیرزادی
کارشناسی ارشد ۱۴۰۱

فایل‌سیستم[1] یک ساختار مدیریت داده است که وظیفهٔ سازمان‌دهی، ذخیره‌سازی و مدیریت داده‌ها روی حافظه‌های ذخیره‌سازی مانند HDD، SSD، حافظه‌های جانبی و... را بر عهده دارد. فایل‌سیستم تعیین می‌کند که چگونه داده‌ها به صورت فایل‌ها و دایرکتوری[2]ها ذخیره شده و چگونه می‌توان به آن‌ها دسترسی داشت. هر فایل‌سیستم دارای ساختار مشخصی برای نام‌گذاری فایل‌ها، مدیریت فضا، سطح دسترسی کاربران و نگهداری فراداده[3] (مانند تاریخ ایجاد یا اندازهٔ فایل) است. انواع مختلفی از فایل‌سیستم‌ها وجود دارند که هر کدام برای نیازها و پلتفرم‌های خاصی طراحی شده‌اند؛ مانند ext4، NTFS، FAT32 و غیره. انتخاب فایل‌سیستم مناسب می‌تواند بر عملکرد، امنیت و قابلیت اطمینان سیستم تأثیر چشم‌گیری داشته باشد.

در دنیای سیستم‌های عامل، به‌خصوص در سیستم‌های عامل مبتی‌بر یونیکس[4] و شبه‌یونیکسی[5] مانند لینوکس، مفهوم inode یکی از اصولی‌ترین و مهم‌ترین مفاهیم در مدیریت فایل‌هاست. به زبان ساده inode ساختاری است که اطلاعات مربوط به هر فایل یا دایرکتوری را در خود ذخیره می‌کند. این اطلاعات، همان فرادادهٔ فایل است که شامل مواردی مانند مالک فایل، گروه مالک، مجوزهای دسترسی، اندازهٔ فایل، زمان‌های مختلف مرتبط با فایل (مثل زمان ایجاد، آخرین دسترسی و آخرین تغییر) و مهم‌تر از همه، اشاره‌گرهایی به محل واقعی بلوک‌های دادهٔ فایل بر روی دیسک است. باید توجه داشت که داده‌های اصلی فایل، به صورت مستقیم در inode ذخیره نمی‌شوند؛ بلکه inode به آدرس‌هایی اشاره می‌کند که داده‌ها در آن‌ها قرار دارند.

در لینوکس، فایل‌سیستم ext4 یکی از پرکاربردترین و محبوب‌ترین گزینه‌هاست. در ext4 تعداد inodeها هنگام ایجاد فایل‌سیستم تعیین می‌شود و این تعداد ثابت باقی می‌ماند. به این معنا که اگر تعداد inodeها تمام شود، دیگر نمی‌توان فایل جدیدی ایجاد کرد؛ حتی اگر فضای فیزیکی کافی روی دیسک موجود باشد. این موضوع به خصوص در سیستم‌هایی که تعداد زیادی فایل کوچک دارند، می‌تواند به یک محدودیت جدی تبدیل شود.

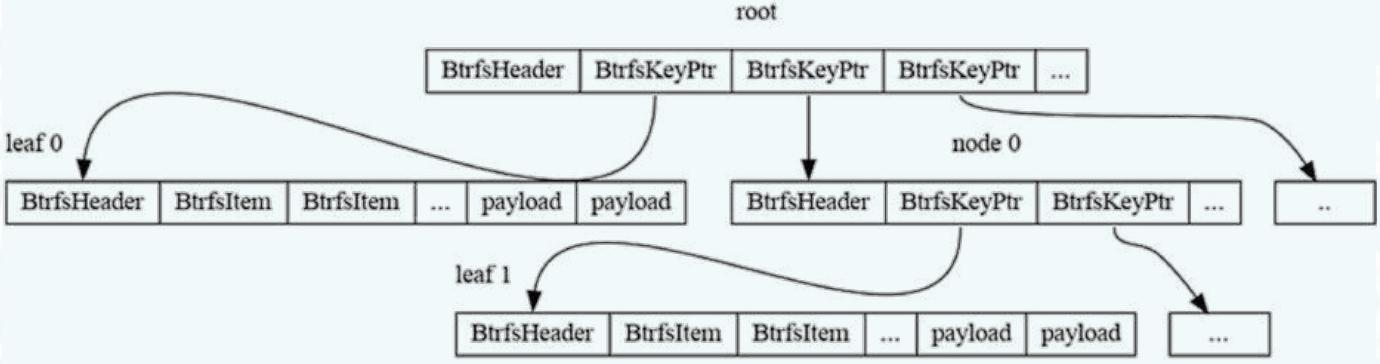
از سوی دیگر، در ویندوز، فایل‌سیستم NTFS به عنوان فایل‌سیستم پیش‌فرض، ساختارهای پیشرفته‌تری برای مدیریت فایل‌ها دارد.

NTFS علاوه‌بر مدیریت امنیتی و پشتیبانی از مجوزهای دسترسی پیشرفته، قابلیت‌هایی مثل فشرده‌سازی فایل‌ها، رمزنگاری و قابلیت‌های journaling (ثبت تغییرات) را نیز ارائه می‌دهد. هرچند مفهوم inode به آن شکل در NTFS وجود ندارد؛ اما ساختارهای مشابهی برای نگهداری فرادادهٔ فایل‌ها در آن تعریف شده‌است. همچنین فایل‌سیستم‌هایی مانند FAT32 و exFAT که بیشتر در حافظه‌های قابل‌حمل کاربرد دارند، محدودیت‌هایی در حداکثر اندازهٔ فایل و تعداد کل فایل‌ها دارند و به‌هیچ عنوان برای استفاده در سیستم‌های پیچیده و با تعداد زیاد فایل مناسب نیستند.

با افزایش پیچیدگی و نیازهای مدرن در ذخیره‌سازی داده‌ها، فایل‌سیستم‌های جدیدی توسعه یافته‌اند که بتوانند محدودیت‌های فایل‌سیستم‌های سنتی را برطرف کنند. یکی از این موارد، Btrfs (B-tree file system) است. ساختار کلی این فایل‌سیستم در تصویر زیر مشهود است؛ این فایل‌سیستم در لینوکس به عنوان یک گزینهٔ پیشرفته و مدرن مطرح شده‌است. مهم‌ترین ویژگی Btrfs، مدیریت پویا و غیرمحدود inodeهاست.

Btrfs نسبت به فایل‌سیستم‌هایی مانند ext3 و ext4، ساختار کاملاً متفاوتی دارد و inodeها را به صورت پویا و در قالب یک ساختار داده‌ای مبتنی بر B-tree مدیریت می‌کند. هر inode در Btrfs به‌عنوان یک رکورد مستقل در درخت B-tree ذخیره می‌شود و تعداد inodeها به‌صورت خودکار و بر اساس نیاز افزایش می‌یابد. این بدین معناست که برخلاف ext4، هیچ آرایهٔ ثابت و محدودیت عددی برای inodeها در Btrfs وجود ندارد.

علاوه‌بر این، Btrfs از مکانیزم Copy-on-Write بهره می‌برد؛ بدین‌معنی که هنگام تغییر داده‌ها یا فراداده (شامل inode)، نسخهٔ جدیدی از آن‌ها ایجاد می‌شود؛ و نسخهٔ قبلی تا زمانی که تغییرات به‌صورت کامل اعمال و وضعیت پایدار نشده‌است، حذف نمی‌شود. این رفتار، اگرچه ممکن است در نگاه اول هدررفت فضا به‌نظر برسد، اما مزایای مهمی دارد؛ تاریخچه‌ای از نسخه‌های پیشین فایل‌ها باقی می‌ماند که امکان تهیهٔ سریع snapshot و بازگشت به وضعیت‌های قبلی را بدون هزینهٔ اضافه فراهم می‌کند.



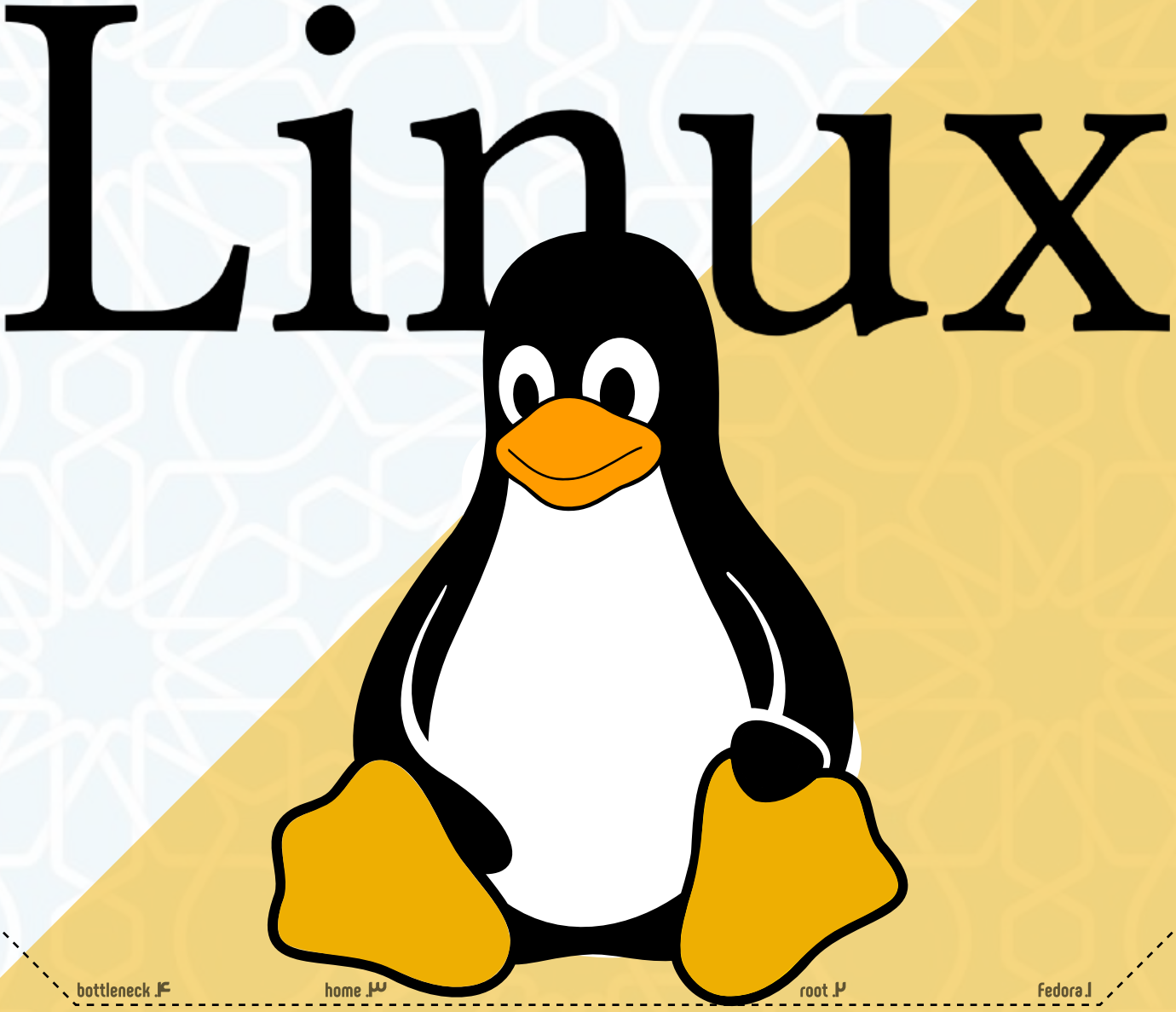
۱. File System
۲. Directory
۳. Metadata
۴. Unix-Based
۵. Unix-Like

در Btrfs، به ازای هر بلوک داده، مقدار checksum (مثل CRC32) نگهداری می‌شود تا صحت داده‌ها بررسی شود. این کار، باعث افزایش اطمینان در برابر خرابی‌های داده می‌شود. همچنین ساختار درختی Btrfs به آن اجازه می‌دهد تا عملیات جست‌وجو، درج و حذف inodeها را به صورت بسیار کارآمد انجام دهد؛ و مدیریت فایل‌ها را در مقیاس وسیع به‌خوبی پشتیبانی کند. این ویژگی‌ها باعث شده‌است که Btrfs نه‌تنها محدودیت inode سنتی را حذف کند، بلکه قابلیت‌هایی مانند checksum برای تشخیص و اصلاح خطاها، فشرده‌سازی داده‌ها و مدیریت RAID داخلی را نیز ارائه دهد که ext4 فاقد آن‌هاست.

هرچند Btrfs ویژگی‌های برجسته‌ای دارد، اما با چالش‌هایی نیز همراه است. Btrfs به دلیل ساختار پیچیده‌تر و امکانات گسترده‌تر نسبت به ext4، نیازمند منابع سیستمی بیشتری است و مدیریت آن به تخصص و دانش فنی بیشتری نیاز دارد. همچنین در سال‌های ابتدایی توسعهٔ Btrfs، به‌خاطر برخی مشکلات پایداری، مورد انتقاد قرار گرفته بود؛ هرچند اکنون به مراتب پایدارتر شده و در توزیع‌های مطرح لینوکس، به‌عنوان گزینهٔ پیش‌فرض برای فایل‌سیستم برخی از توزیع‌های لینوکس به‌کار گرفته می‌شود. با این حال، در مقایسه با ext4 که سال‌ها مورد آزمون قرار گرفته و بسیار پایدار است، هنوز در برخی سناریوهای خاص، احتمال مواجهه با خطاهای پیچیده در Btrfs وجود دارد که باید مدیریت شود.

از این رو، من هم از این فایل‌سیستم استفاده می‌کنم که فدورا[1] از نسخهٔ ۳۳ و بالاتر، تصمیم گرفت آن را به‌عنوان فایل‌سیستم پیش‌فرض خود برگزیند. ویژگی‌ای که بیش از همه من را مجذوب خود کرد، قابلیت subvolume است. از آن‌جا که برای مسیرهای ریشه[2] (/) و خانه[3] (/home) همواره می‌بایست دو پارتیشن جداگانه ایجاد می‌شد و تخصیص حافظهٔ مناسب برای هر یک، خود چالشی دشوار محسوب می‌شد، با بهره‌گیری از این قابلیت می‌توان یک پارتیشن کلی ایجاد کرد و دو subvolume برای / و home/ ساخت؛ که تا زمانی که فضای کلی partition به اتمام نرسیده‌است، هر دو می‌توانند از آن فضا استفاده کنند.

در نهایت، inode به‌عنوان یک ساختار کلیدی در مدیریت فایل‌ها، نقش مهمی در کارکرد فایل‌سیستم‌ها ایفا می‌کند. فایل‌سیستم‌های سنتی مانند ext4، محدودیت‌های سخت‌گیرانه‌ای در تعداد inodeها دارند که می‌تواند در سیستم‌های دارای تعداد زیاد فایل کوچک، به‌عنوان یک گلوگاه[4] مطرح شود. سیستم‌های جدیدتری مانند Btrfs، این محدودیت‌ها را با مدیریت پویا و پیشرفتهٔ inodeها رفع کرده و امکاناتی فراتر از یک فایل‌سیستم سنتی ارائه می‌دهند. انتخاب بین این سیستم‌ها بستگی به نیازهای خاص هر محیط و میزان تخصص مدیریتی دارد؛ اما قطعاً Btrfs با قابلیت‌های نوین خود یکی از پیشگامان تحول فایل‌سیستم‌ها در اکوسیستم لینوکس محسوب می‌شود.



۱. Fedora
۲. root
۳. home
۴. bottleneck

داستان نبرد سخت

شبیه‌سازی نزدیک به واقعیت، انگیزهٔ پشت سؤالات هاردوار


مهدی علی‌نژاد
کارشناسی ۱۴۰۱

بخش علمی هاردوار نیز مانند تمامی رویدادهای دیگر، سختی‌های خودش را دارد. از سؤالات گرفته تا داوری، توزیع قطعات و حتی محاسبهٔ امتیازات، هر کدام اندازهٔ یک مسئولیت جداگانه، به انرژی نیاز داشتند. اما چیزی که این هاردوار را با دفعات قبلی متمایز می‌کرد، حجم تغییراتی بود که در روند مسابقه به‌وجود آورد. تغییراتی مانند فروشگاه، سؤالات انتخابی و تعداد بالای داوران برای نمره‌دهی، همگی تغییراتی بودند که طبق بازخوردهایی که از سال‌های پیش به ما رسیده بود، عملیاتی شدند. با تمام این‌ها، چیزی که خیلی برایمان مهم بود، کیفیت سؤالات بود. پشت تک‌تک سؤالات فکر شده بود و هر کدام قصد داشتند شما را با یک بخش از جهان وسیع سخت‌افزار آشنا کنند.

ما در هاردوار امسال، تلاشمان بر این بود که سؤالات بیشترین تنوع ممکن را داشته باشند. شروع کار ما از دی‌ماه بود. یادم می‌آید اولین جلسه‌ای که برگزار کردیم، دو شب قبل از ICPC بود و تعداد زیادی (حدود ۲۴ تا) سؤال علمی آماده کرده بودیم که به نسبت هاردوارهای قبلی، عدد بسیار بزرگی بود. هاردوار، هرساله یک مسیر مشخص با حداکثر ۴ بخش -که نیاز به داوری حضوری داشتند- برای همه ترسیم می‌کرد؛ ولی ما قصد داشتیم ناممکن را ممکن کنیم. جالب است بدانید که از سمت ادوار هم مدام به ما گفته می‌شد که این حجم از لود علمی، قابل‌تحمل نیست؛ و تنها راه‌حل، کم کردن سؤالات است.

جا دارد اشاره کنم که من اوایل کار برای سازمان‌دهی ایده‌ها، یک کانال تلگرامی ایجاد کردم تا ایده‌ها را داخل آن قرار دهم. البته به جز روز اول، چیز دیگری در آن کانال ارسال نشد؛ اما موردی که بسیار برایم جذاب است، این بود که که یک‌سری افراد، این کانال را قبل از مسابقه پیدا کردند؛ که همت این دوستان واقعاً ستودنی بود.

در ذهن ما، سؤالات داخل چندین دستهٔ اصلی قرار داشتند. سؤالات معمایی، امنیت و شبکه، رباتیک، هوش مصنوعی و برنامه‌نویسی که هر کدام در ادامه توضیح داده خواهند شد. این سؤالات چندین‌بار دچار تغییر و بازبینی شدند؛ چرا که هرچه جلوتر می‌رفتیم و قطعات را تست می‌کردیم، بیشتر متوجه سختی یا امکان‌پذیر نبودن سؤال می‌شدیم.

از جملهٔ این تغییرات، می‌توان به اضافه شدن یک سؤال برنامه‌ریزی‌نشده در بخش دوم روز اول، زیر و رو شدن یکی از المان‌های بازی و تبدیل آن به سؤال معدن، اضافه شدن قابلیت بازگرداندن قطعه‌ها و... اشاره کرد.

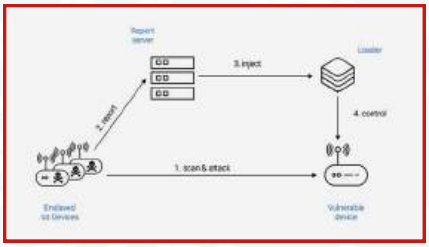
سؤالات معمایی

این دسته از سؤالات، حال‌وهوایی مانند سؤالات رویداد وارم‌آپ داشتند؛ البته با چاشنی سخت‌افزار. مطمئنم برایتان سؤال شده که حالا این سؤالات معمایی ما کجاست؟ در جواب به این سؤال، با تأسف می‌گویم که هیچ‌کدام از سؤالات معمایی، به موقع آماده نشدند :). خیلی به دوستانی که برای این سؤال‌ها وقت می‌گذاشتند افتخار می‌کردم. از یک‌شنبهٔ قبل از رویداد، کارشان را شروع کرده بودند و واقعاً داشتند در طراحی سؤالات پیش می‌رفتند؛ ولی مشکلات پیش‌بینی‌نشده‌ای سر راه‌شان قرار گرفت. مشکلاتی در حد پریدن فیوز پریزهای کلاس ۲۰۲. قرار بود با این سؤالات، کمی از زیبایی‌های فیلد سخت‌افزار را به شما نشان دهیم؛ که به امید خدا در دورهٔ بعدی در خدمتتان خواهیم بود! با این سؤالات قصد داشتیم با استفاده از انواع قطعات سخت‌افزاری، جرقهٔ خلاقیتی در شما بزنیم و همچنین دست شما را گرم کنیم که ببینید چه کارهایی را می‌توانید با این قطعات انجام دهید!

سؤالات امنیت و شبکه

بخش امنیت و شبکه، سؤالات Capture The Flag -یا به اختصار CTF- مانند سؤال «کلاغ سیاه» یا «دامبلدور» و سؤالات DigiSpark یعنی «حملهٔ بدون دست»، «کارآگاه» و «AGCC» را شامل می‌شود. هدف اصلی این سؤالات، آشنا شدن با ضعف این سیستم‌هاست. دستگاه‌های Internet of Things -یا به اختصار IoT-، یکی از بزرگترین هدف‌ها برای هکرها هستند. به‌دلیل جدید بودن و در دسترس نبودن، اکثراً ملاحظات امنیتی در آن‌ها لحاظ نمی‌شود و همین موضوع، آن‌ها را بسیار خطرناک می‌کند.

به‌عنوان مثال، یکی از حملات معروف منع سرویس در سال ۲۰۱۶ با نام Mirai، با بهره‌گیری از دوربین‌ها، مودم‌های شبکه و سایر تجهیزات اینترنت‌اشیاء (IoT) اجرا شد. این حمله توانست ترافیکی با حجمی بیش از یک ترابایت بر ثانیه به سمت سرویس‌دهندگانی مانند OVH -یکی از بزرگ‌ترین شرکت‌های میزبانی وب در اروپا- ارسال کند. حتی گزارش‌هایی مبنی بر تلاش برای حمله به کل شبکهٔ یک کشور نیز منتشر شده‌است. چیزی که به این حمله قدرت داده بود، ضعف‌های امنیتی موجود در همین دستگاه‌ها بود. فرایند تحت سلطه گرفتن این دستگاه‌ها به این صورت بود که ابتدا دستگاه‌های آلوده در شبکه، به دنبال دستگاه‌های جدیدی می‌گشتند، سپس مشخصات آن دستگاه را به سرور گزارش می‌دادند، سرور نیز با وارد کردن یک کد مخرب در ثابت‌افزار این دستگاه، آن را تحت سلطهٔ خود در می‌آورد.



حال، هدف اصلی سؤالات «حملهٔ بدون دست» و «کارآگاه» همین بود که بتوانید به کمک ابزارهایی که در دست دارید، کنترل یک دستگاه را به‌دست آورید. سؤالات CTF، بیشتر از این نظر حائز اهمیت بودند که هنگامی که یک حمله رخ می‌دهد، در اکثر مواقع، فرد یا ارگان هدف حمله چیزی جز یک سیستم آلوده در اختیار ندارد و نیاز است تا با مهندسی معکوس و تکنیک‌های دیگر بتوانید آسیب‌پذیری را پیدا کرده و آن را برطرف کنید. سوال OTA نیز که مخفف Over The Air update هست، شما را با یک مکانیزم دفاعی دربرابر حملات ثابت‌افزار آشنا می‌کرد. سؤال MimeDogs نیز مهارت شما برای مقابله با قابلیت‌های یک سیستم را می‌سنجید و سؤال AGCC یا Air Gap Covert Channel نیز از مسائل پرکاربرد امنیتی بود که به پیدا کردن راه‌هایی برای انتقال اطلاعات بین دو دستگاه که یک فاصلهٔ فیزیکی آن‌ها را از یکدیگر جدا کرده است، اشاره داشت که یک موضوع بسیار کاربردی است.

سوال UART نیز، طریقهٔ مواجههٔ شما با یک مسئلهٔ مبهم را نشان می‌داد که چگونه می‌توانید یک راه‌حل بهینه برای مسئله پیدا کنید.

سؤالات رباتیک

از مجموعهٔ سؤالات رباتیک، تنها دو سؤال موفق شدند که به روز مسابقه برسند! این‌گونه سؤالات مکانیکی، اولین بار بود که امتحان می‌شدند. (البته کاشف به عمل آمد که به موازات ما، مسابقات رباتیک دانش‌آموزی نیز در سالن جباری در حال برگزاری بود). این سؤالات برای خودمان هم خیلی جدید بودند. بچه‌های تیم علمی از هفتهٔ قبل از رویداد در تلاش بودند که رباتی بسازند که بتواند روی چرخ‌هایش راه برود و همین تسک، از ما دو شبانه‌روز زمان برد. هدف از این سؤالات، این بود که با چالش‌های بیشتری از جنس برق و کنترل روبه‌رو شوید.

سؤال «دستهٔ بازی»، قصد داشت دید اولیه‌ای به شما بدهد که کار با قطعات سخت‌افزاری چه مشکلات و معضلاتی دارد. پس از آن، سؤال «سلام WALL-E» بود؛ این تنها سؤالی بود که شما را موظف به ساخت یک شیء مکانیکی می‌کرد و ترکیب زیبای این دو دنیا با یک‌دیگر بود.

در انتها نیز، سؤال «ماز» قرار داشت؛ در تیم رویداد حقیقتاً فکر نمی‌کردیم کسی تصمیم به حل این سؤال بگیرد و به حل آن برسد! که البته یک تیم به این نقطه رسید. هدف از سؤال در ابتدا تجربهٔ ساخت یک دستگاه مانند moon rover بود که شما را به طور هم‌زمان با چالش انتقال اطلاعات و کنترل مواجه کند.

سؤالات هوش مصنوعی

سؤالات هوش مصنوعی نیز دستهٔ دیگری از سؤالات هستند؛ ایدهٔ اولیه پیدایش این بخش، الهام گرفته از یکی از سؤالات دورهٔ قبل بود که البته شک و تردید زیادی نیز نسبت به آن وجود داشت. سؤال مذکور در دورهٔ قبل، به کابوس بچه‌ها تبدیل شده بود و تمامی تیم‌ها را گیر انداخته بود؛ بچه‌هایی که عموماً ترم ۲ یا ۴ بودند، وادار به حل سؤالی در مباحث ML شدند؛ که از قضا مدل آن نیز مدل ساده‌ای نبود.

قوی‌ترین انگیزهٔ «انتخابی شدن» سؤالات هاردوار امسال همین بود که دیگر چنین اتفاقی رخ ندهد! ولی در عین حال، بتوانیم شما را با استفاده‌های هوش مصنوعی در سخت‌افزار آشنا کنیم.

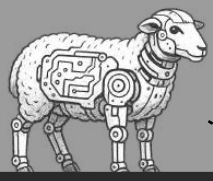
سؤال «حاکم نامرئی»، تنها سؤال حاضر از این گروه بود و هدف از آن، آشنا شدن با یک کاربرد یادگیری ماشین[1] در سخت‌افزار بود، البته سعی کردیم که این بار، کسی با تجربهٔ تلخی که از سؤال هوش مصنوعی پارسال داشتیم، روبه‌رو نشود.

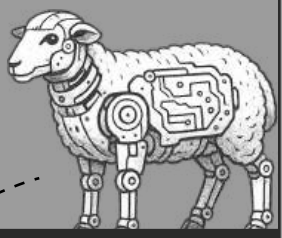
سؤالات برنامه‌نویسی

ما در سؤالات برنامه‌نویسی نیز با چالش روبه‌رو شدیم. اما این چالش برعکس بقیهٔ زمینه‌ها بود؛ هرچه ما سر سؤالات دیگر وقت می‌گذاشتیم و به سختی زیاد آن‌ها پی می‌بردیم، سؤالات برنامه‌نویسی برعکس بودند و وقتی این سؤالات ریلیز شدند، دیدیم که به‌سادگی توسط شرکت‌کنندگان حل می‌شوند! سؤالاتی که قرار بود صرفاً حل یکی از آن‌ها، چهار ساعت وقت بگیرد، همه در کمتر از ۴۰ دقیقه توسط بعضی شرکت‌کنندگان حل شدند. البته به جز سؤال «بازی چند نفره» که برای حل نکردن طراحی شده بود! اما همچنان یک‌سری افراد بودند که چالش را پذیرفتند و سربلند از آن بیرون آمدند.

هدف از این سؤالات، بیشتر آشنا شدن با خود ESP بود و این که چگونه می‌توان برای ساخت وسایل کوچک از آن‌ها استفاده کرد. این بود داستانی از تلاش ما برای آشنا کردن شما با زمینهٔ شیرین سخت‌افزار! اگر بخواهم هدف خود از هد علمی شدن در این رویداد را در یک جمله خلاصه کنم، آشنا کردن شما با کارهایی است که می‌توانید به دست خودتان انجام دهید و با استفاده از این دستگاه‌ها، لامپ اتاقتان را هوشمند کنید، یک ماشین کنترلی بسازید؛ یا خیلی چیزهای دیگر؛ ببینید که اسباب‌بازی‌هایی که در بچگی عاشق آن‌ها بودید را چقدر ساده می‌توانید خودتان بسازید و یک زمینه برای استفاده از خلاقیت خود پیدا کنید.

do androids dream of electric sheep?



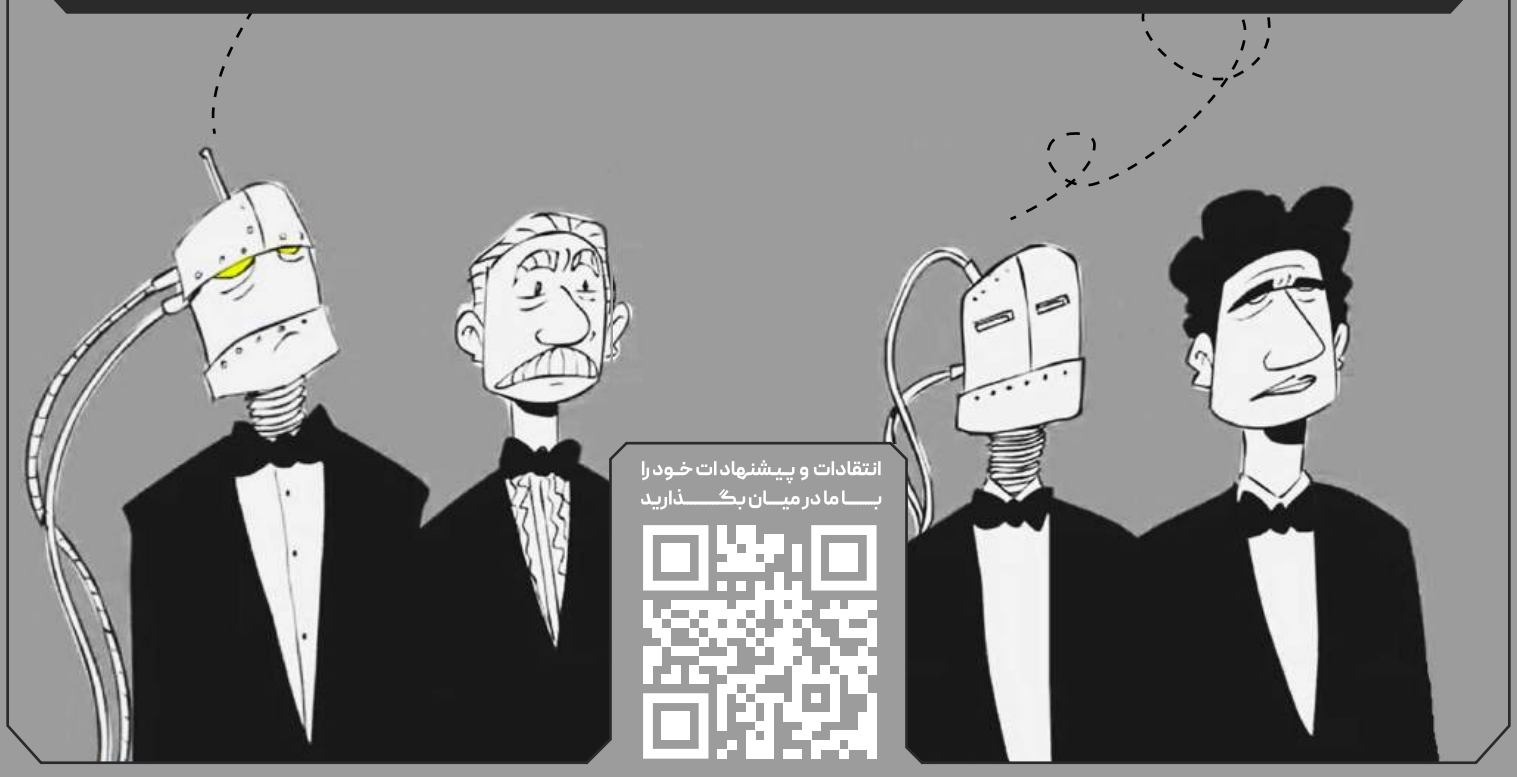


«چشمانش را بست و به بره ای کوچک با پشم های برق آسا فکر کرد...»

در شمارهٔ دوم نشریه اندکی با تفکر ماشین‌ها آشنا شدیم و حالا به مفهومی گسترده‌تر می‌پردازیم؛ با پرسشی جدید: آیا ماشین‌ها هم خواب گوسفندهای الکتریکی می‌بینند؟

در پس هر عامل ساخته‌شده توسط یک الگوریتم هوش مصنوعی، شبکه‌ای عظیم از نورون‌ها نهفته‌است که داده‌ها را با سرعتی فراتر از تصور انسان پردازش می‌کند. این شبکه‌ها از مغز ما الهام گرفته‌اند، اما برخلاف ما، هیچ‌گاه خسته نمی‌شوند و میلیون‌ها تجربه (که در این‌جا همان داده ها هستند) را طی چند ثانیه مرور می‌کنند. مدل های مولد تصویر -مانند DALL·E یا Midjourney- می‌توانند تنها با یک جمله، دنیایی خیالی را به تصویری دقیق تبدیل کنند؛ نه به این خاطر که «می‌بینند»، بلکه چون میلیارد ها تصویر را دیده‌اند و الگو های پنهان میان آن‌ها را یافته‌اند. در پزشکی، الگوریتم‌ها با بررسی میلیون‌ها عکس رادیولوژی، توده‌های سرطانی را گاهی با دقتی بیشتر از متخصصان تشخیص می‌دهند. در موسیقی، سازنده‌هایی مانند AIVA نه تنها قطعات کلاسیک می‌سازند، بلکه سبک آهنگ‌سازان را بازآفرینی می‌کنند، تا جایی که شنونده شاید نتواند خالق واقعی اثر را تشخیص دهد.

مرز تازه‌ای شکل گرفته‌است؛ نه بین انسان و ماشین، بلکه میان دانستن و آگاهی. ماشین‌ها از لحظهٔ غروب خورشید آگاهند، اما آیا لذت آرامش آن را درک می‌کنند؟



انتقادات و پیشنهادات خود را
با ما در میان بگذارید

STUDENTS' SCIENTIFIC CHAPTER
انجمن علمی دانشکده مهندسی کامپیوتر
دانشگاه صنعتی شریف