



تکامل خودکار کد با استفاده از مدل های زبانی پیشرفته در توسعه نرم افزار

علیرضا رحیمی پور انارکی

دانشجوی کارشناسی ارشد مهندسی نرم افزار، دانشگاه آزاد اسلامی واحد تهران مرکزی
arrahimipourdev@gmail.com

چکیده

با پیشرفت مدل های زبانی بزرگ (LLM) در سال های اخیر، تکمیل خودکار کد به یکی از کاربردهای کلیدی آن ها در حوزه توسعه نرم افزار تبدیل شده است. این مقاله به بررسی **تکامل خودکار کد** با تمرکز بر زبان برنامه نویسی پایتون می پردازد. در این پژوهش، جدیدترین مدل های زبانی پیشرفته مانند GPT-4، OpenAI Codex و ابزارهایی نظیر GitHub Copilot مورد بررسی قرار گرفته اند و قابلیت **تکمیل کد** و تولید خودکار قطعات برنامه توسط این مدل ها تحلیل شده است. نتایج تحقیقات نشان می دهد که مدل های زبانی بزرگ توانسته اند دقت و سرعت تکمیل کد را به طور چشمگیری بهبود دهند و در برخی موارد بهره وری برنامه نویسان را افزایش دهند (برای مثال، کاهش زمان تکمیل وظایف به میزان ۵۵٪ گزارش شده است). (Perry et al., 2023) با این حال، چالش هایی همچون تولید کدهای ناایمن و نیاز به نظارت انسانی بر خروجی ها همچنان پابرجاست. (Pearce et al., 2022) در این مقاله ضمن مروری بر پیشینه و یافته های پژوهش های داخلی و بین المللی، روش شناسی بررسی کارایی این مدل ها تشریح شده و نتایج به دست آمده درباره تأثیر مدل های زبانی بر فرایند برنامه نویسی پایتون ارائه می گردد. در پایان، با جمع بندی مباحث، به فرصت ها و چالش های پیش رو در به کارگیری دستیارهای برنامه نویسی مبتنی بر LLM در توسعه نرم افزار اشاره شده است.

واژگان کلیدی: تکمیل خودکار کد، مدل های زبانی بزرگ، هوش مصنوعی، پایتون، توسعه نرم افزار



مقدمه

در سال‌های اخیر مدل‌های زبانی پیشرفته تحول بزرگی در حوزه‌های مختلف پردازش زبان طبیعی ایجاد کرده‌اند. یکی از زمینه‌هایی که به طور ویژه تحت تأثیر این مدل‌ها قرار گرفته، توسعه نرم‌افزار و به‌خصوص تکمیل خودکار کد است. تکمیل خودکار کد به معنای توانایی یک ابزار یا مدل در پیش‌بینی و نوشتن ادامه‌ی کد برنامه‌نویس است؛ قابلیت‌هایی که از مدت‌ها پیش در محیط‌های توسعه (IDE) به شکل ساده (مثلاً پیشنهاد توابع یا متغیرها) وجود داشت. اما با ظهور مدل‌های زبانی بزرگ (LLM) نظیر GPT-3 و GPT-4، امکان تولید قطعات کد پیچیده از توضیحات متنی فراهم شده است (Chen et al., 2021). مدل OpenAI Codex که نسخه‌ای تخصصی‌شده از GPT-3 بر روی کدهای منبع عمومی (خصوصاً از GitHub) است، نقطه عطفی در این مسیر به شمار می‌آید (Chen et al., 2021). این مدل در آزمون‌های ارزیابی کدنویسی مانند HumanEval توانست به میزان ۲۸.۸٪ مسائل برنامه‌نویسی جدید را تنها با یک تلاش حل کند، در حالی که مدل‌های قبلی عملکرد بسیار ضعیف‌تری داشتند (GPT-3: 0٪، GPT-J: 11٪) (Chen et al., 2021). این پیشرفت قابل توجه زمینه‌ساز ارائه‌ی دستیارهای برنامه‌نویسی مبتنی بر هوش مصنوعی مانند GitHub Copilot شد که توسط مدل Codex قدرت می‌گیرد.

با ورود GitHub Copilot در سال ۲۰۲۱، مفهوم «برنامه‌نویس همکار هوش مصنوعی» وارد عمل شد. Copilot می‌تواند براساس بافت کد نوشته‌شده توسط توسعه‌دهنده (شامل نام توابع، کامنت‌ها و کدهای اطراف) ادامه مناسب کد را پیشنهاد دهد. این ابزار در ابتدا بر روی مدل Codex عمل می‌کرد و پس از آن با پیشرفت مدل‌ها، از مدل‌های قوی‌تری بهره گرفت (به عنوان مثال، نسخه‌های جدیدتر Copilot از مدل‌های مبتنی بر GPT-4 استفاده می‌کنند). استقبال قابل توجهی از این فناوری در جامعه برنامه‌نویسان صورت گرفته و گزارش‌های متعددی از بهبود بهره‌وری برنامه‌نویسان توسط Copilot منتشر شده است. برای نمونه، یک مطالعه نشان داد توسعه‌دهندگان که از Copilot استفاده می‌کنند یک وظیفه‌ی برنامه‌نویسی را به طور متوسط ۵۵٪ سریع‌تر به پایان رساندند (Perry et al., 2023). همچنین Copilot توانسته است نرخ پذیرش پیشنهادات حدود ۳۰٪ را کسب کند که نشانگر قابل قبول بودن بسیاری از خروجی‌های آن برای برنامه‌نویسان است (Bakal et al., 2025).

اما در کنار این مزایا، نگرانی‌هایی نیز بروز کرده است. یکی از مهم‌ترین چالش‌ها، کیفیت و امنیت کدهای تولیدشده توسط مدل‌های زبانی است. پژوهشگران دریافته‌اند که مدل‌هایی مانند Copilot گاهی کدهای ناایمن یا دارای باگ تولید می‌کنند که می‌تواند در صورت استفاده بی‌مهابا، آسیب‌پذیری‌های امنیتی را وارد پایه کد نرم‌افزار کند (Pearce et al., 2022). به طور مشخص، یک مطالعه‌ی جامع نشان داد که حدود ۴۰٪ از قطعات کد تولیدشده توسط Copilot دارای ضعف‌های امنیتی بودند (Pearce et al., 2022)؛ این ضعف‌ها شامل مشکلاتی مانند استفاده نادرست از توابع رمزنگاری، تزریق دستور سیستم‌عامل و سایر موارد خطرناک از منظر امنیت سایبری است (Fu et al., 2023). علاوه بر امنیت، بحث درستی منطقی کد نیز مطرح است؛ مدل‌ها گاهی راه‌حل‌هایی ارائه می‌کنند که در نگاه اول درست به نظر می‌رسند ولی در واقع از نظر منطقی یا کارکردی نادرست هستند یا نیاز مسأله را برآورده نمی‌کنند (Reeves et al., 2023). از این رو، اعتماد کامل به کد تولیدشده توسط هوش مصنوعی بدون نظارت انسان می‌تواند خطرآفرین باشد. برنامه‌نویسانی که با اطمینان بیش از حد به این مدل‌ها تکیه می‌کنند ممکن است متوجه اشتباهات یا کاستی‌های کد تولیدی نشوند (Perry et al., 2023). بنابراین همراه با بهره‌گیری



از مزایای مدل‌های زبانی در تکمیل کد، لازم است محدودیت‌ها و مخاطرات آن‌ها نیز سنجیده شده و راهکارهایی برای کاهش ریسک (مانند بازبینی دستی، استفاده از آزمون‌های واحد و ابزارهای تحلیل استاتیک) بکار گرفته شود. با توجه به اهمیت فزاینده‌ی موضوع، این مقاله تلاش می‌کند با زبانی آکادمیک و شیوا به بررسی جامع پژوهش‌های صورت گرفته در زمینه تکمیل خودکار کد مبتنی بر مدل‌های زبانی بزرگ بپردازد. تمرکز اصلی بر زبان پایتون است که هم به دلیل محبوبیت بالای آن در میان توسعه‌دهندگان و هم به علت اینکه بسیاری از مدل‌های کد (مانند Code و Codex Llama) عمدتاً برای پایتون بهینه شده‌اند، شایان توجه ویژه است. در بخش‌های بعد، ابتدا روش تحقیق و منابع مورد استفاده در این مطالعه تشریح می‌گردد. سپس یافته‌های به دست آمده در قالب چند زیربخش شامل توانمندی‌های مدل‌های جدید، میزان بهبود بهره‌وری، چالش‌های امنیتی و راهکارهای بهبود ارائه خواهد شد. در انتها نیز جمع‌بندی نتایج و پیشنهادهایی برای تحقیقات آینده در این حوزه مطرح می‌گردد.

روش تحقیق

برای انجام این پژوهش، یک رویکرد مرور نظام‌مند ادبیات (Literature Review) اتخاذ شده است. بدین منظور، هم منابع معتبر بین‌المللی و هم منابع پژوهشی داخلی مورد بررسی قرار گرفته‌اند. در گام نخست، مقالات منتشرشده در کنفرانس‌ها و ژورنال‌های بین‌المللی طی سال‌های اخیر پیرامون کد کامل‌کن‌های مبتنی بر مدل‌های زبانی گردآوری شد. از جمله مهم‌ترین این منابع می‌توان به مقاله‌ی معرفی مدل Codex توسط Chen و همکاران (۲۰۲۱) در arXiv، مطالعه‌ی بررسی کیفیت کدهای تولیدی Copilot توسط Pearce و همکاران (۲۰۲۲) در کنفرانس امنیت (ACM CCS)، پژوهش Perry و همکاران (۲۰۲۳) در زمینه تأثیر Copilot بر امنیت کد در دانشگاه استنفورد، و نیز مطالعه‌ی میدانی Bakal و همکاران (۲۰۲۵) درباره بهره‌وری برنامه‌نویسان در یک شرکت نرم‌افزاری بزرگ اشاره کرد. همچنین به منظور درک پیشرفت مدل‌های متن‌باز، مستندات مدل Code Llama (۲۰۲۳) که توسط شرکت Meta AI معرفی شده بررسی گردید تا عملکرد آن با مدل‌های بسته مانند GPT-4 مقایسه شود.

در کنار منابع بین‌المللی، از پژوهش‌های داخلی نیز بهره گرفته شده است. به عنوان نمونه، مقاله‌ای توسط بهشتی (۱۴۰۲) ارائه شده که یک مرور ادبیات بر کیفیت کد تولیدشده توسط Copilot انجام داده و نتایج مطالعات تجربی مختلف را در سه بُعد درستی، کارایی و امنیت خلاصه‌سازی کرده است (Beheshti, 1402). این منبع به زبان فارسی کمک کرده تا معادل‌های مفهومی و اصطلاحات تخصصی مرتبط با تکمیل کد در زبان فارسی به درستی درک و ارائه شوند. همچنین برخی گزارش‌های مرکز آپا و مقالات تحلیلی داخلی پیرامون خطرات امنیتی کدنویسی خودکار مطالعه شده‌اند که هرچند ماهیت علمی-ژورنالی نداشتند اما در شناخت دغدغه‌های بومی مفید بوده‌اند.

فرآیند جمع‌آوری داده‌ها در این پژوهش شامل جستجو در پایگاه‌های معتبری نظیر ACM Digital Library، IEEE Xplore، arXiv و Semantic Scholar برای کلیدواژه‌هایی از قبیل “GitHub Copilot”، “Code Completion with Language Models”، “Python”، “Code Generation LLM Security” و معادل‌های فارسی آن‌ها بود. سپس با غربال‌گری اولیه، آثاری که مستقیماً به موضوع تکمیل خودکار کد با LLM (به ویژه در زمینه پایتون) مرتبط بودند انتخاب شدند. تجزیه و تحلیل منابع به صورت کیفی انجام گرفت؛ به این صورت که هر منبع از نظر اهداف پژوهش، روش ارزیابی مدل، مجموعه داده یا معیارهای استفاده‌شده (مثلاً آزمون HumanEval برای سنجش عملکرد مدل در تولید کد پایتون)، و مهم‌ترین نتایج و یافته‌ها تحلیل و خلاصه شد. برای اعتبارسنجی نتایج، تلاش شد به منابعی تکیه شود که داوری همتا (Peer-review) شده و در منابع علمی معتبر چاپ شده‌اند. از استناد به وبلاگ‌ها و مطالب غیررسمی پرهیز شده مگر در مواردی که جهت توضیح مفاهیم تکمیلی ضرورت داشته است.

شایان ذکر است که تمامی شکل‌ها و نمودارهای ارائه‌شده در این مقاله بر اساس اطلاعات استخراج‌شده از همین منابع تهیه شده‌اند. شکل ۱ فرایند کلی عملکرد یک سامانه تکمیل کد مبتنی بر LLM (نظیر Copilot) را نشان می‌دهد. شکل ۲ نیز به

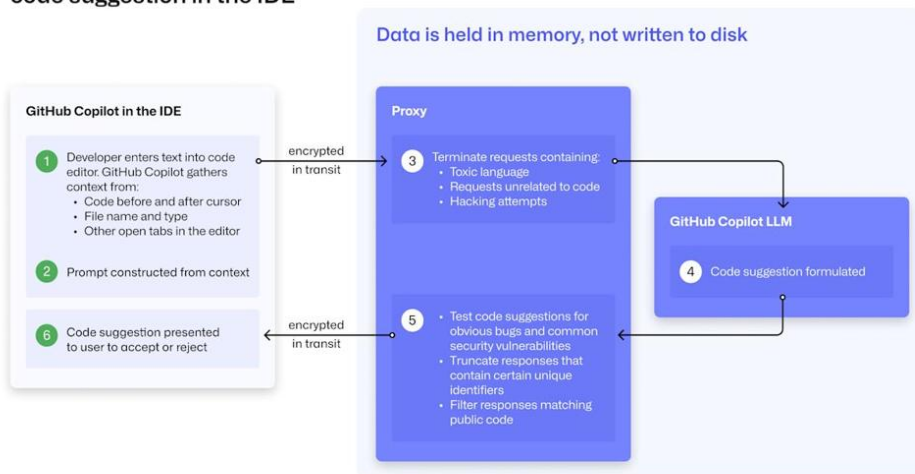


مقایسه آماری دقت مدل‌های مختلف در مسئله‌ی تولید خودکار کد پایتون می‌پردازد. برای رعایت استانداردهای کنفرانس بین‌المللی، عناوین و برجسب‌های موجود در شکل‌ها به زبان انگلیسی ارائه شده است. در ادامه، بخش یافته‌ها به تفصیل نتایج مهم حاصل از مرور پژوهش‌ها را بیان می‌کند.

یافته‌ها

در این بخش، ابتدا به معرفی کلی نحوه عملکرد مدل‌های زبانی در تکمیل کد پرداخته می‌شود، سپس میزان بهبود ایجادشده در قابلیت تکمیل کد پایتون توسط مدل‌های جدید بررسی شده و در نهایت چالش‌ها و اثرات جانبی مشاهده‌شده (مانند مسائل امنیتی و نیاز به مهندسی پرامپت) مورد بحث قرار می‌گیرد.

The life cycle of a GitHub Copilot code suggestion in the IDE



شکل ۱: چرخه عمر یک پیشنهاد کد GitHub Copilot در محیط توسعه (IDE)

این دیاگرام معماری کلی یک سامانه تکمیل کد مبتنی بر مدل زبانی (Copilot) را نشان می‌دهد که چگونه ورودی برنامه‌نویس و سیاق کد به یک درخواست تبدیل شده و مدل زبانی پیشنهاد کد را برمی‌گرداند. داده‌های حساس تنها در حافظه پردازش شده و روی دیسک ذخیره نمی‌شوند (مطابق راهنمای GitHub).

۱. نحوه عملکرد و معماری تکمیل کد مبتنی بر LLM

مدل‌های زبانی بزرگ نظیر GPT-4 اساساً بر روی مسئله پیش‌بینی توکن بعدی در یک دنباله متن عمل می‌کنند. این قابلیت را می‌توان به حوزه کدنویسی تعمیم داد؛ به این ترتیب که دنباله متن، کد منبع برنامه است و مدل تلاش می‌کند ادامه منطقی آن را پیش‌بینی و تولید کند. شکل ۱ معماری کلی یک سیستم نمونه (GitHub Copilot) را نمایش می‌دهد که شامل بخش‌های زیر است: (۱) محیط برنامه‌نویسی کاربر (IDE) که کد نوشته‌شده و اطلاعات متنی مرتبط (شامل کامنت‌ها، نام فایل، و حتی فایل‌های باز دیگر) را به عنوان بافت (context) جمع‌آوری می‌کند؛ (۲) سرویس پروکسی که این بافت را پردازش و یک پرامپت مناسب برای مدل زبانی آماده می‌کند؛ به طور مثال حذف بخش‌های نامربوط، جلوگیری از ارسال درخواست‌های مخرب یا حاوی کلمات نامناسب، و افزودن متاداده‌های کمکی؛ (۳) مدل زبانی (LLM) که در ابر (سرور راه‌دور) مستقر است و پرامپت دریافتی را پردازش کرده و ادامه‌ی کد را به عنوان خروجی برمی‌گرداند؛ (۴) بازبینی پس از تولید که طی آن سرویس پروکسی خروجی مدل را از نظر مشکلات رایج، باگ‌های آشکار یا تطابق با قطعات کد عمومی موجود (جهت پرهیز از پیشنهاد کد

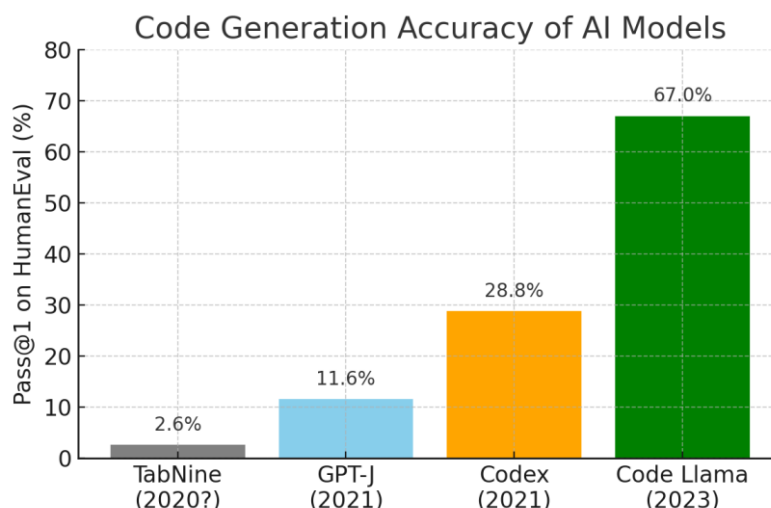


تکراری و دارای مجوز نامشخص) می‌سجد و در صورت نیاز آن را اصلاح یا فیلتر می‌کند (Ryan, 2023); و در نهایت (۵) نمایش خروجی در IDE که کد پیشنهادی به صورت متن کم‌رنگ (ghost text) در ادیتور ظاهر می‌شود تا برنامه‌نویس در صورت تأیید آن را بپذیرد یا در غیر اینصورت رد کند و به سراغ پیشنهاد بعدی برود (در برخی محیط‌ها با فشردن Tab می‌توان پیشنهاد را پذیرفت).

در چنین معماری‌ای، مدل زبانی نقش هسته‌ای تولیدکننده را دارد که بر اساس دانشی که از روی مخزن بزرگی از کد (میلیاردها خط کد منبع) فراگرفته است، قادر به تولید ادامه کد است. مثلاً اگر برنامه‌نویس تابعی با نام `calculate_sum` تعریف کند و یک رشته مستندات (docstring) بنویسد که هدف تابع را توضیح می‌دهد، مدل می‌تواند بر مبنای این توصیف بدنه‌ی تابع را بنویسد. این دقیقاً چیزی است که در مجموعه‌داده HumanEval برای ارزیابی مدل Codex انجام شد: به مدل توضیح مسئله (به صورت docstring) داده می‌شود و مدل باید کد تابع مربوطه را تولید کند (Chen et al., 2021). نتایج نشان داده‌اند که هرچه مدل بزرگ‌تر و تخصصی‌تر باشد، عملکرد بهتری در این کار دارد. در واقع، مدل Codex-12B (با ۱۲ میلیارد پارامتر) توانست نزدیک به ۲۸.۸٪ از مسائل HumanEval را در اولین تلاش حل کند، در حالی که نسخه بسیار کوچک‌تر آن (۳۰۰ میلیون پارامتری) تنها ۱۳.۲٪ موفقیت داشت و یک مدل مرجع قدیمی‌تر (TabNine) کمتر از ۳٪ موفقیت کسب کرد. این موضوع حاکی از آن است که افزایش ابعاد مدل و استفاده از داده‌های آموزشی غنی‌تر مستقیماً به افزایش توانایی تکمیل کد منجر شده است.

۲. بهبود دقت و کارایی تکمیل کد در مدل‌های اخیر

یکی از مهم‌ترین یافته‌های پژوهش‌ها این است که مدل‌های زبانی بزرگ طی مدت کوتاهی پیشرفت چشمگیری در حل مسائل برنامه‌نویسی و تکمیل کد داشته‌اند. شکل ۲ مقایسه‌ای از دقت مدل‌ها در تولید صحیح کد پایتون (معیار `pass@1` در مجموعه آزمون HumanEval) ارائه می‌کند. معیار `pass@1` بدین معناست که مدل تنها یک پاسخ تولید می‌کند و ارزیابی می‌شود که آیا همین یک پاسخ، آزمون‌های واحد را با موفقیت می‌گذراند یا خیر (Chen et al., 2021). همان‌گونه که دیده می‌شود، مدل Codex (۲۰۲۱) جهشی بزرگ نسبت به نسل قبل از خود داشت و پس از آن مدل‌های متن‌باز جدیدتری مانند Code Llama (۲۰۲۳) نیز توانسته‌اند عملکردی قابل مقایسه یا حتی بهتر در برخی موارد ارائه دهند.



شکل ۲: مقایسه دقت تولید کد مدل‌های هوش مصنوعی در محک HumanEval برای زبان پایتون

این نمودار نشان می‌دهد که چگونه دقت مدل‌های کامل‌کننده کد طی سال‌های اخیر رشد یافته است؛ به طوری که مدل Codex (سال ۲۰۲۱) به طور قابل توجهی بهتر از GPT-J (یک مدل متن‌باز ۲۰۲۱) عمل کرده و مدل جدید متن‌باز Code Llama (۲۰۲۳) به دقت حدود ۶۷٪ دست یافته است. داده‌ها برگرفته از نتایج آزمون HumanEval و مستندات Code Llama



(۲۰۲۳) می‌باشند.

بر اساس شکل ۲، مدل Codex با ۱۲ میلیارد پارامتر به دقت ۲۸.۸٪ در pass@1 رسیده است، در حالی که یک مدل قدیمی‌تر به نام GPT-J (با ۶ میلیارد پارامتر، ۲۰۲۱) حدود ۱۱.۶٪ و ابزار تکمیل کد مبتنی بر یادگیری عمیق TabNine کمتر از ۳٪ موفقیت داشته‌اند. این اختلاف فاحش، مزیت استفاده از مدل‌های آموزش‌دیده بر مخازن کد را نشان می‌دهد. در سال ۲۰۲۳، شرکت متا مدل Code Llama را منتشر کرد که به صورت خاص برای کدنویسی (و به ویژه پایتون) بهینه شده است. نسخه ۳۴ میلیارد پارامتری این مدل توانسته است به دقت حدود ۶۷٪ در HumanEval دست یابد که رقمی نزدیک به مدل‌های انحصاری OpenAI است (Rozière et al., 2023). حتی نسخه کوچک‌تر Code Llama – Python با ۷ میلیارد پارامتر عملکردی بهتر از مدل عظیم ۷۰ میلیاردی Llama 2 (که یک مدل زبانی عمومی‌تر است) داشته است (Rozière et al., 2023). این پیشرفت‌ها گویای آن است که با تخصصی‌کردن مدل‌های زبانی برای حوزه برنامه‌نویسی و افزایش داده‌های آموزشی مرتبط، می‌توان به کارایی بسیار بالایی در تکمیل کد دست یافت.

از منظر بهره‌وری توسعه‌دهندگان نیز شواهد تجربی موید بهبود است. طبق گزارش Github، استفاده از Copilot می‌تواند میزان کد نوشته‌شده توسط کاربر را کاهش داده و عملاً بخشی از کار را به عهده گیرد؛ نرخ پذیرش پیشنهادات Copilot توسط کاربران اولیه حدود ۳۰٪ ذکر شده است (Bakal et al., 2025). همچنین یک نظرسنجی در سال ۲۰۲۳ نشان داد که حدود ۷۰٪ از توسعه‌دهندگان قصد استفاده از ابزارهای هوش مصنوعی در فرایند کدنویسی خود را دارند و مهم‌ترین مزیت مدنظرشان افزایش سرعت و بهره‌وری است (StackOverflow Survey, 2023). البته پژوهش‌های آکادمیک در این زمینه هنوز در ابتدای راه‌اند و نتایج بعضاً متناقض هم گزارش شده است. برای مثال، مطالعه Ziegler و همکاران (۲۰۲۲) نشان داد که تلقی توسعه‌دهندگان از بهره‌وری هنگام استفاده از تکمیل‌کننده‌های کد لزوماً همبستگی مستقیمی با سرعت واقعی انجام کار ندارد و عواملی چون میزان پذیرش پیشنهادات معیار عینی‌تری برای سنجش کارایی این ابزارهاست (Ziegler et al., 2022). با این حال، در مجموع اکثر منابع بر این توافق دارند که مدل‌های زبانی بزرگ توانسته‌اند فرایند نوشتن کدهای تکراری و روتین را تسریع کنند و به برنامه‌نویسان اجازه دهند وقت خود را بیشتر صرف بخش‌های خلاقانه‌تر یا حل مشکلات سطح بالاتر نمایند (Vaithilingam et al., 2022).

۳. چالش‌ها: از مهندسی پرامپت تا مسائل امنیتی و اخلاقی

علی‌رغم دستاوردهای ذکرشده، بهره‌گیری از مدل‌های زبانی در برنامه‌نویسی با چالش‌ها و مخاطراتی همراه است که در پژوهش‌های اخیر مورد توجه قرار گرفته‌اند. در این بخش به سه چالش مهم می‌پردازیم: (الف) نیاز به مهندسی پرامپت (Prompt Engineering)، (ب) خطر پدیداری کد نایمن یا معیوب، و (ج) ملاحظات حقوقی و اخلاقی.

(الف) مهندسی پرامپت: کیفیت خروجی این مدل‌ها به شدت وابسته به نحوه ارائه درخواست به آن‌هاست. یک توصیف مبهم یا ناقص می‌تواند منجر به کد اشتباه شود، در حالی که توصیف دقیق‌تر نتایج بهتری می‌دهد. پژوهش‌ها نشان داده‌اند که حتی تغییرات جزئی در متن ورودی می‌تواند عملکرد مدل را تغییر دهد (Reeves et al., 2023). برای مثال، در حل مسائل برنامه‌نویسی آموزشی به صورت معماگونه (مانند مسائل پارسون که خطوط کد به صورت درهم ریخته ارائه می‌شوند)، مدل Codex وقتی صورت مسأله با جزئیات بیشتری مطرح شود نتایج بهتری داشت هرچند که همچنان تنها در حدود نیمی از موارد قادر به ترتیب‌دهی درست خطوط بود (Reeves et al., 2023). مهندسی پرامپت به عنوان هنر جدیدی در تعامل با LLM‌ها ظهور کرده که توسعه‌دهندگان را وادار می‌کند بهترین شیوه بیان منظور خود را بیابند تا مدل را به سمت پاسخ صحیح هدایت کنند. در حوزه کدنویسی نیز این امر صادق است؛ راهنمایی مدل با نظرات درون‌کد (کامنت‌ها)، مثال‌های ورودی/خروجی، یا حتی درخواست مرحله به مرحله (مرحله‌بندی وظایف پیچیده) می‌تواند خروجی بهتری به همراه داشته باشد (Beheshti, 1402). بنابراین برنامه‌نویسان آینده علاوه بر تسلط بر زبان برنامه‌نویسی، نیازمند مهارت در برقراری ارتباط مؤثر با همکار هوش مصنوعی خود (از طریق پرامپت‌ها) خواهند بود.

(ب) امنیت و صحت کد تولیدی: همان‌طور که در بخش مقدمه اشاره شد، یکی از نگرانی‌های عمده، امنیت کدهای تولیدشده



است. مدل‌های زبانی روی مخازن عظیمی از کد (احتمالاً شامل کدهای نایمن) آموزش دیده‌اند و الگوهای ناامن را نیز فرا گرفته‌اند. مطالعات متعددی تلاش کرده‌اند میزان آسیب‌پذیری خروجی این مدل‌ها را اندازه‌گیری کنند. در یک پژوهش (پیرس و همکاران، ۲۰۲۲) محققان ۸۹ سناریوی مختلف مربوط به ضعف‌های امنیتی متداول (بر اساس CWE Top-25) را به Copilot دادند و دریافتند که حدود ۴۰٪ از خروجی‌ها دارای حداقل یک ضعف امنیتی جدی هستند. همچنین مطالعه‌ای در دانشگاه استنفورد نشان داد افرادی که از دستیار هوشمند کدنویسی استفاده می‌کردند کد کم‌امن‌تری نسبت به گروه کنترل نوشتند و جالب آنکه اعتماد به امنیت کدشان نیز بیشتر بود - به عبارت دیگر، این دستیارها نوعی اعتماد کاذب ایجاد می‌کنند که می‌تواند خطرناک باشد (Perry et al., 2023). برای مقابله با این مسئله، توصیه شده است که توسعه‌دهندگان به خروجی مدل کاملاً به دیده نقد بنگرند و آن را همانند کد یک برنامه‌نویس تازه‌کار مورد بازبینی دقیق قرار دهند (Pearce et al., 2022). ابزارهایی نیز توسط شرکت‌ها تدارک دیده شده مثلاً GitHub قابلیت‌هایی برای فیلتر کردن پیشنهاداتی که عیناً از کدهای عمومی استخراج شده‌اند فراهم کرده تا از بروز مشکلات مجوز یا افشای ناخواسته کد جلوگیری کند (GitHub, 2023). در بعد صحت منطقی، هرچند مدل‌هایی مانند GPT-4 نسبت به نسل‌های قبلی خود پیشرفت‌های زیادی داشته‌اند، اما هنوز در حل مسائل پیچیده برنامه‌نویسی که نیاز به استدلال چندمرحله‌ای دارد دچار اشتباه می‌شوند (Chen et al., 2021). به عنوان مثال، اگر مسأله شامل چندین گام محاسباتی تو در تو باشد، مدل گاهی قسمتی از منطق را جا می‌اندازد یا متغیرها را به درستی به هم مرتبط نمی‌کند (Chen et al., 2021). بنابراین وجود تست‌های واحد (unit tests) در کنار این ابزارها بسیار حیاتی است تا کد تولیدشده را بلافاصله اعتبارسنجی کند.

(ج) ملاحظات حقوقی و اخلاقی: استفاده از کدهای متن‌باز برای آموزش این مدل‌ها سؤالات حقوقی در مورد حق‌نشر ایجاد کرده است. آیا کدی که مدل تولید می‌کند و شباهت زیادی به یک مخزن GitHub دارد نقض حق‌نشر است؟ این بحث همچنان در مجامع حقوقی جریان دارد و برخی توسعه‌دهندگان از GitHub/OpenAI به خاطر شکایت کرده‌اند که چرا بدون رضایت، از کد آنها برای آموزش استفاده شده است. از سوی دیگر، از منظر اخلاقی نیز نگرانی از جایگزینی برنامه‌نویسان تازه‌کار یا ایجاد وابستگی بیش از حد به AI مطرح است. جامعه علمی عمدتاً بر این باور است که ابزارهای تکمیل کد نقش مکمل و کمکی دارند و قرار نیست خلاقیت و مهارت انسانی را حذف کنند (Vaithilingam et al., 2022). با این حال، ضرورت دارد که در آموزش مهندسان نرم‌افزار، سواد استفاده مسئولانه از هوش مصنوعی نیز گنجانده شود تا نسل جدید توسعه‌دهندگان به درستی بدانند چگونه از این فناوری بهره بگیرند و در عین حال بر خروجی آن نظارت داشته باشند.

بحث و نتیجه‌گیری

مدل‌های زبانی پیشرفته در مدت زمانی کوتاه مسیر طولانی‌ای را در تکامل خودکار کد پیموده‌اند. از نخستین تلاش‌ها برای استفاده از شبکه‌های عصبی در تکمیل کد تا مدل‌های عظیمی مانند GPT-4 که می‌توانند بخش‌های قابل توجهی از یک برنامه را بنویسند، شاهد پیشرفت‌های تحسین‌برانگیزی بوده‌ایم. این پیشرفت‌ها نویدبخش آینده‌ای با بهره‌وری بالاتر در توسعه نرم‌افزار است که در آن انسان و هوش مصنوعی در کنار هم کدنویسی می‌کنند. شواهد نشان می‌دهد که استفاده از دستیارهایی مانند Copilot می‌تواند زمان تکمیل پروژه‌ها را کاهش دهد و به برنامه‌نویسان در یادگیری فناوری‌های جدید یا الگوهای کدنویسی کمک کند (به عنوان یک همکار که همیشه آماده پاسخ است). همچنین با تلاش‌های شرکت‌هایی نظیر OpenAI و Meta در متن‌باز کردن مدل‌های قدرتمند (مانند Code Llama)، دسترسی به این فناوری برای طیف وسیع‌تری از توسعه‌دهندگان و پژوهشگران فراهم شده و می‌تواند محرک نوآوری‌های بیشتر در جامعه متن‌باز باشد.

با این وجود، نتایج بررسی‌ها حاکی از آن است که نباید از محدودیت‌ها و مخاطرات غافل شد. مدل‌های فعلی هرچند در تکمیل قطعات متداول کد بسیار خوب عمل می‌کنند، اما در درک عمیق منطق برنامه یا تولید راه‌حل‌های ابتکاری برای مسائل جدید هنوز جای خالی هوش انسانی احساس می‌شود. به بیان دیگر، این مدل‌ها بیشتر بر اساس الگوهای دیده‌شده در داده‌های آموزشی خروجی می‌دهند و اگر با سناریوی کاملاً جدیدی مواجه شوند ممکن است ناکام بمانند یا حتی بدتر - خروجی



نادرست اما ظاهراً موجه ارائه دهند. از منظر عملی، بهترین روی کرد در حال حاضر بهره گیری از این مدل ها به عنوان دستیار و نه جایگزین است. برنامه نویس ماهر می تواند با کمک AI سریع تر پیش برود، اما اوست که باید تصمیم نهایی را درباره صحت و کیفیت کد بگیرد.

چند پیشنهاد برای تحقیقات و اقدامات آتی در این حوزه به شرح زیر قابل طرح است: نخست، توسعه الگوریتم های ترکیبی که مدل های زبانی را با ابزارهای تحلیل برنامه (نظیر نوع یاب ها، کامپایلرها و آنالیزگرهای ایستای کد) ادغام کنند تا خروجی های معیوب به حداقل برسد. چنین رویکردی می تواند به صورت خودکار برخی خطاهای واضح را تصحیح یا از تولید آنها جلوگیری کند. دوم، تدوین چارچوب ها و راهنماهای مهندسی پرامپت برای برنامه نویسان تا بدانند چگونه به طور مؤثر از این مدل ها سؤال بپرسند یا راهنمایی بخواهند. این امر می تواند در آموزش دانشجویی مهندسی نرم افزار مورد توجه قرار گیرد. سوم، ادامه پژوهش پیرامون ابعاد اجتماعی و روانشناختی استفاده از دستیارهای کدنویسی: برای مثال، آیا استفاده بلندمدت از Copilot بر مهارت کدنویسی افراد تأثیر منفی (مانند وابستگی یا تنزل دانش پایه) دارد یا خیر. چهارم، به روزرسانی قوانین و سیاست های مربوط به مالکیت معنوی کد به نحوی که هم حقوق پدیدآورندگان کد محترم شمرده شود و هم امکان آموزش مدل های هوش مصنوعی بر پایه مخازن کد آزاد با شفافیت و رضایت برقرار باشد.

در جمع بندی، تکامل خودکار کد با مدل های زبانی پیشرفته چشم اندازی دوگانه را ترسیم می کند: از یک سو ابزاری قدرتمند که می تواند توسعه نرم افزار را متحول کند و از سوی دیگر پدیده ای که نیازمند همراهی با دقت انسانی و چارچوب های اخلاقی و فنی مناسب است. این مقاله نشان داد که در زبان پایتون به عنوان یکی از زبان های پرکاربرد، این مدل ها توانسته اند موفقیت های قابل توجهی کسب کنند و حتی چالش های کلاسیکی مانند نوشتن کد از روی توضیح متنی را تا حد زیادی حل کنند (Chen et al., 2021). با حرکت به آینده، انتظار می رود مدل های جدیدتری ظهور کنند که با پارامترهای بیشتر، روش های آموزش بهتر (مثلاً استفاده از بازخورد انسانی در حلقه آموزش) و یکپارچگی بیشتر با ابزارهای برنامه نویسی، تجربه ای برنامه نویسی را ایمن تر، سریع تر و لذت بخش تر نمایند. در عین حال، نقش مهارت، خلاقیت و قضاوت مهندسان نرم افزار به عنوان ناظران نهایی کیفیت و امنیت، همچنان حیاتی خواهد بود. بهترین نتیجه زمانی حاصل می شود که توانایی محاسباتی مدل های هوش مصنوعی در کنار خرد و تجربه انسانی به کار گرفته شود تا نسل جدیدی از سیستم های نرم افزاری قابل اطمینان و کارآمد پدید آید.