SCIENCE HUB REVIEW
2026
Volume 1
Issue 5
www.sciencehubreview.uz
ILMIY ELEKTRON JURNAL

GENERATIV SUN'IY INTELLEKT YORDAMIDA DASTURIY KOD YARATISH: IMKONIYATLARI VA MUAMMOLARI

A'zamov Azizbek Akmal o'g'li

OXUS University kompyuter ilimlari 501-26 guruh talabasi, 111500, O'zbekiston

avaqadro85@gmail.com

Ibragimov Diyorbek Alisherovich

OXUS University kompyuter ilimlari 501-26 guruh talabasi, 111500, O'zbekiston

ibragimovdiyor913@gmail.com

Yusupov Samandar Anvarovich

OXUS Universiteti katta o'qituvchisi, Toshkent 100208, O'zbekiston

Samandar75781111@gmail.com

Annotatsiya

Mazkur maqolada generativ sun'iy intellekt (GenAI) va katta til modellari (LLM) yordamida dasturiy kod yaratishning imkoniyatlari hamda muammolari tahlil qilinadi. Tadqiqot davomida GenAI vositalarining dasturchi mahsuldorligiga ta'siri, kod sifati, xavfsizlik va ta'lim jarayonidagi o'rni ko'rib chiqiladi. Maqolaning asosiy g'oyasi shundan iboratki, GenAI dasturchining o'rnini bosuvchi "avtopilot" emas, balki inson nazorati ostida ishlaydigan "yordamchi pilot" (copilot) bo'lishi kerak. Shu g'oya asosida "Generatsiya–Tekshiruv–Integratsiya" (GTI) deb nomlangan uch bosqichli ishlash modeli taklif etiladi. Model talabalar va yosh dasturchilarga AI yaratgan kodni xavfsiz va samarali qo'llashda yordam beradi.

Kalit so'zlar

generativ sun'iy intellekt, katta til modellari, kod generatsiyasi, GitHub Copilot, kod xavfsizligi, "gallyutsinatsiya", dasturiy ta'minot muhandisligi, GTI modeli.

KIRISH: So'nggi yillarda sun'iy intellekt dasturiy ta'minot ishlab chiqish sohasini tubdan o'zgartirmoqda. Ilgari dasturchi har bir qatorni qo'lda yozgan bo'lsa, bugun oddiy matnli tavsif (prompt) asosida funksiya, sinf, hatto butun dastur qismlarini yaratish

mumkin. ChatGPT, Claude, GitHub Copilot, Gemini kabi vositalar bu jarayonni keng ommaga olib chiqdi.

Biroq qulaylik bilan birga jiddiy savollar ham paydo bo‘ldi: AI yozgan kodga qanchalik ishonish mumkin? U xavfsizmi? Mualliflik huquqi kimga tegishli? Talabalar AI’ga haddan tashqari tayanib, dasturlashni o‘rganmay qo‘ymaydimi? Ushbu maqolaning maqsadi shu savollarga javob izlash va amaliy yechim taklif qilishdir.

Tadqiqot vazifalari:

1. GenAI yordamida kod yaratish texnologiyasining asosiy tamoyillarini tushuntirish;
2. Uning imkoniyatlarini aniq misollar bilan ko‘rsatish;
3. Asosiy muammolar va xavflarni aniqlash;
4. Muammolarni kamaytirish uchun amaliy model taklif qilish.

2. Nazariy asos

Zamonaviy kod generatsiyasi 2017-yilda taklif etilgan Transformer arxitekturasiga asoslanadi. Bu arxitekturadagi "diqqat mexanizmi" (attention) modelga matndagi so‘zlar va belgilar orasidagi bog‘liqlikni hisobga olish imkonini beradi. Katta til modellari milliardlab parametrga ega bo‘lib, ommaviy matnlar va ochiq manbali kod omborlari ustida o‘qitiladi.

Model kod yozayotganda dastur mantig‘ini inson kabi "tushunmaydi": u berilgan kontekstga ko‘ra eng ehtimolli keyingi belgi (token) ketma-ketligini bashorat qiladi. Shu sababli natija ko‘pincha to‘g‘ri va foydali bo‘ladi, lekin ba’zan tashqi ko‘rinishidan ishonarli, aslida esa noto‘g‘ri bo‘lishi mumkin. Bu hodisa "gallyutsinatsiya" deb ataladi va quyidagi muammolarning ko‘pchiligining ildizidir.

Kod generatsiyasi sifatini baholash uchun odatda HumanEval kabi benchmarklar ishlatiladi: modelga funksiya tavsifi beriladi, yaratilgan kod esa unit-testlar yordamida tekshiriladi. Keyinroq paydo bo‘lgan SWE-bench kabi benchmarklar modelning real GitHub muammolarini hal qilish qobiliyatini sinaydi, bu esa amaliy vazifalar murakkabroq ekanini ko‘rsatadi.

3. Genai yordamida kod yaratishning imkoniyatlari

3.1. Mahsuldorlikning oshishi. GitHub Copilot bilan o‘tkazilgan nazorat ostidagi tajribada (Peng va boshq., 2023) Copilot’dan foydalangan dasturchilar JavaScript’da

HTTP server yozish topshirig‘ini nazorat guruhiga qaraganda taxminan 55,8 foiz tezroq bajargan. Bu natija takrorlanuvchi (boilerplate) kodni yozishda AI ayniqsa foydali ekanini ko‘rsatadi.

3.2. Takrorlanuvchi vazifalarni avtomatlashtirish. Konfiguratsiya fayllari, CRUD operatsiyalari, ma’lumotlarni formatlash, oddiy SQL so‘rovlari va shablon testlarni AI bir necha soniyada yaratadi. Dasturchi esa murakkab arxitektura va biznes mantiqqa e’tibor qaratishi mumkin.

3.3. Tezkor prototiplash. Startap va talabalar loyihalarida g‘oyani tezda ishlaydigan prototipga aylantirish muhim. GenAI MVP yaratish vaqtini sezilarli qisqartiradi.

3.4. Kodni tushuntirish va hujjatlashtirish. Model begona yoki eski kodni oddiy tilda tushuntirib beradi, izohlar (comment) va hujjatlar yozadi. Bu yangi boshlovchilar uchun ayniqsa qulay.

3.5. Refaktoring va xatolarni topish. AI kodni soddalashtirish, nom berishni yaxshilash, xato xabarlarini tahlil qilish va tuzatish variantlarini taklif qilishda yordam beradi.

3.6. Dasturlash tillari o‘rtasida o‘tkazish. Masalan, Python kodini Java yoki C++ ga o‘girish, eski kod bazasini zamonaviy freymvorkka ko‘chirish imkoniyati mavjud.

3.7. Ta’limdagi o‘rni. AI shaxsiy ustoz sifatida savollarga istalgan vaqtda javob beradi, turli yechimlarni solishtirib ko‘rsatadi va o‘quvchi tezligiga moslashadi. Bu, ayniqsa, o‘qituvchi yetishmaydigan sharoitda qimmatlidir.

4. Asosiy muammolar va xavflar

4.1. Noto‘g‘ri yoki mavjud bo‘lmagan kod (gallyutsinatsiya). Model mavjud bo‘lmagan funksiya, kutubxona yoki API nomini to‘qib chiqarishi mumkin. Chen va hamkasblarining (2021) Codex bo‘yicha ishida ham modelning bir martalik urinishda hamma masalani to‘g‘ri hal qilmasligi, ko‘p urinishdan so‘ng muvaffaqiyat ehtimoli ortishi ko‘rsatilgan. Demak, birinchi natijaga ko‘r-ko‘rona ishonib bo‘lmaydi.

4.2. Xavfsizlik zaifliklari. Pearce va boshq. (2022) GitHub Copilot’ni xavfsizlik nuqtai nazaridan tekshirib, xavfli deb tan olingan 89 ta stsenariy bo‘yicha yaratilgan dasturlarning taxminan 40 foizida zaiflik aniqlanganini ma’lum qilgan. Perry va boshq. (2023) tadqiqotida esa AI yordamchisidan foydalangan ishtirokchilar ko‘pincha

xavfsizroq bo‘lmagan kod yozgani, shu bilan birga o‘z kodiga ortiqcha ishonganini ko‘rsatilgan. Eng keng tarqalgan xatolar: SQL-inyeksiya, parollarni kod ichida saqlash, kiritilgan ma’lumotni tekshirmaslik.

4.3. Kontekstning cheklanganligi. Model katta loyihaning to‘liq arxitekturasini, ichki qoidalarini va bog‘liqliklarini ko‘ra olmaydi. Natijada alohida funksiya to‘g‘ri ishlasa ham, butun tizimga mos kelmasligi mumkin.

4.3.1. Eskirgan bilim. Model o‘qitilgan sanadan keyin chiqqan kutubxona versiyalari va yangi API’lar haqida bilmasligi mumkin, shuning uchun eskirgan yoki bekor qilingan (deprecated) usullarni taklif qiladi.

4.4. Mualliflik huquqi va litsenziya masalalari. Modellar ochiq manbali kod ustida o‘qitilgan, ba’zi hollarda ular o‘qitish ma’lumotidagi kod parchalarini deyarli aynan takrorlashi mumkin. Bu GPL kabi litsenziyalar bilan bog‘liq huquqiy noaniqlikni keltirib chiqaradi. Bu sohada qonunchilik hali shakllanmoqda.

4.5. Maxfiylik. Dasturchi kompaniyaning ichki kodi yoki maxfiy kalitlarini tashqi AI xizmatiga yuborsa, ma’lumot sizib chiqishi xavfi tug‘iladi.

4.6. Ko‘nikmalarning pasayishi va ta’limiy xavf. Talaba AI yozgan kodni tushunmasdan nusxalasa, algoritmik fikrlash va muammo hal qilish ko‘nikmalari rivojlanmaydi. Bundan tashqari, "ishlayapti" degan taassurot kodni tekshirish odatini susaytiradi.

4.7. Texnik qarz va qo‘llab-quvvatlash. Tushunilmagan va uslubi bir xil bo‘lmagan AI kodi vaqt o‘tishi bilan qo‘llab-quvvatlashni qiyinlashtiradi.

4.8. Xolislik va xilma-xillik. Model o‘qitish ma’lumotlaridagi keng tarqalgan yondashuvlarni takrorlaydi, shu sababli nostandart yoki yangi yechimlar o‘rniga ko‘pchilik ishlatadigan usullarni taklif qilishga moyil.

5. Taklif etilayotgan g‘oya: generatsiya–tekshiruv–integratsiya (Gti) modeli

Yuqoridagi muammolarni hisobga olib, AI bilan ishlashning uch bosqichli modelini taklif qilaman. Asosiy tamoyil: "AI yozadi, inson javob beradi." Ya’ni kod uchun yakuniy mas’uliyat har doim dasturchida qoladi.

1-bosqich. Generatsiya (aniq prompt bilan).

– Vazifani aniq ta’riflash: dastur tili, versiyalar, kiruvchi va chiquvchi ma’lumotlar, cheklovlar.

– Katta vazifani kichik qismlarga bo‘lish.

– Maxfiy ma’lumotlarni (parol, API kalit, shaxsiy ma’lumot) promptga kiritmaslik.

2-bosqich. Tekshiruv (ishonchsizlik tamoyili).

– Kodni qator-baqator o‘qib, tushunmaguncha ishlatmaslik.

– Unit-testlar yozish va ishga tushirish (iloji bo‘lsa, testlarni avval yozish).

– Statik tahlil va xavfsizlik vositalaridan foydalanish (masalan, linter va zaiflik skanerlari).

– Kutubxona va funksiya nomlarining rasmiy hujjatda mavjudligini tekshirish.

3-bosqich. Integratsiya (mas’uliyat bilan).

– Kod uslubi va loyiha qoidalariga moslashtirish.

– Litsenziya va mualliflik masalasini hisobga olish.

– AI qo‘llanilgan joylarni jamoa ichida ochiq belgilash.

– Kod-review (boshqa inson tomonidan ko‘rib chiqish) o‘tkazish.

Ta’limda qo‘llash qoidasi. Talabalar uchun qo‘shimcha tamoyil: avval o‘zi yechishga urinadi, keyin AI bilan solishtiradi, yakunda esa AI taklifini o‘z so‘zlari bilan tushuntirib beradi. Shunda AI o‘rganishni almashtirmay, uni kuchaytiradi.

6. Amaliy misol

Tasavvur qiling, talaba "foydalanuvchi parolini ma’lumotlar bazasiga saqlaydigan Python funksiyasi"ni AI’dan so‘raydi. Model parolni oddiy matn yoki zaif xesh bilan saqlaydigan kod taklif qilishi mumkin. GTI modeli bo‘yicha:

1. Generatsiya: prompt aniqlanadi: "Python 3.12, bcrypt yoki Argon2 ishlatilsin, SQL-inyeksiyadan himoyalangan parametrli so‘rov bo‘lsin."

2. Tekshiruv: kod o‘qiladi, kutubxona hujjatidan tekshiriladi, to‘g‘ri va noto‘g‘ri parol uchun testlar yoziladi.

3. Integratsiya: loyiha uslubiga moslanadi, sozlamalar maxfiy kalitlar bilan birga muhit o‘zgaruvchilarida saqlanadi, kod review’dan o‘tkaziladi.

Shu yo‘l bilan AI tezligi saqlanadi, xavf esa sezilarli kamayadi.

7. Xulosa

Generativ sun’iy intellekt dasturiy kod yaratishda tezlik, qulaylik va yangi imkoniyatlar beradi, lekin xatolar, zaifliklar, huquqiy noaniqliklar va ko‘nikmalarning pasayishi kabi jiddiy muammolarni ham keltirib chiqaradi. Maqolada ko‘rsatilganidek, ularning asosiy sababi modelning ehtimollik asosida ishlashi va kontekstni to‘liq anglamasligidadir.

Taklif etilgan GTI modeli AI’dan foydalanishni tartibga soladi: aniq prompt, majburiy tekshiruv va mas’uliyatli integratsiya. Kelajakda dasturchining roli faqat kod yozishdan ko‘ra ko‘proq muammoni to‘g‘ri qo‘yish, AI natijasini baholash va yakuniy sifat uchun javob berishga o‘tadi. Shuning uchun bugungi talabalar AI bilan ishlash ko‘nikmasini fundamental dasturlash bilimlari bilan birga rivojlantirishi lozim.

Adabiyotlar

1. Vaswani, A. va boshq. (2017). Attention Is All You Need. Advances in Neural Information Processing Systems (NeurIPS).

2. Chen, M. va boshq. (2021). Evaluating Large Language Models Trained on Code. arXiv:2107.03374.

3. Pearce, H. va boshq. (2022). Asleep at the Keyboard? Assessing the Security of GitHub Copilot’s Code Contributions. IEEE Symposium on Security and Privacy.

4. Peng, S. va boshq. (2023). The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. arXiv:2302.06590.

5. Perry, N. va boshq. (2023). Do Users Write More Insecure Code with AI Assistants? ACM CCS.

6. Jimenez, C. E. va boshq. (2024). SWE-bench: Can Language Models Resolve Real-World GitHub Issues? ICLR.