DOI: 10.65322/j.srd.05192026.86

GENERATIV SUN'IY INTELLEKT YORDAMIDA DASTURIY KODNI AVTOMATIK GENERATSIYALASH VA OPTIMALLASHTIRISH USULLARI

Absobirov S.Q., *PhD, dotsent,*
Navoiy davlat universiteti

Annotatsiya. *Mazkur maqolada generativ sun'iy intellekt texnologiyalaridan foydalanib dasturiy kodni avtomatik shakllantirish, uning funksional to'g'riligini tekshirish va hisoblash samaradorligini optimallashtirish usullari tadqiq etilgan. Dasturiy ta'minotni ishlab chiqishda katta til modellarining imkoniyatlari, tabiiy tilda berilgan masalalarni dasturga aylantirish mexanizmlari hamda kod sifatiga ta'sir qiluvchi omillar tahlil qilindi. Qiyosiy tahlil, algoritmik modellashtirish, funksional dekompozitsiya va ko'p mezonli baholash metodlariga tayangan holda generatsiya, statik tahlil, testlash, optimallashtirish va qayta verifikatsiyalash jarayonlarini birlashtiruvchi bosqichli yondashuv taklif qilindi. Kodning funksional to'g'riligi, bajarilish vaqti, xotira sarfi, strukturaviy murakkabligi va xavfsizlik ko'rsatkichlari asosida baholash mezonlari shakllantirildi. Model istiqbolli metodologik yechim sifatida asoslandi; uning amaliy samaradorligi keyingi eksperimental sinovlarda tasdiqlanishi lozim.*

Abstract. *This article examines methods for automatic source-code generation, functional verification, and computational optimization using generative artificial intelligence. Based on comparative analysis, algorithmic modeling, functional decomposition, and multi-criteria evaluation, a staged framework integrating generation, static analysis, testing, optimization, and repeated verification is proposed. The framework evaluates correctness, execution time, memory consumption, structural complexity, and security. The article offers a methodological model rather than unverified empirical performance claims; experimental validation is identified as a necessary future step.*

Kalit so'zlar: *generativ sun'iy intellekt, katta til modellari, kod generatsiyasi, kod optimallashtirish, dasturiy injiniring, mashinali o'qitish, algoritmik samaradorlik, avtomatik testlash, iterativ takomillashtirish.*

Keywords: *generative artificial intelligence, large language models, code generation, code optimization, software engineering, automated testing, iterative refinement.*

Kirish. Axborot-kommunikatsiya texnologiyalarining jadallashuvi, ishlab chiqarish va xizmatlar sohasining raqamlashtirilishi murakkab dasturiy tizimlarga talabni oshirmoqda. Dasturiy mahsulot sifati faqat vazifani bajarish bilan emas, balki tezkorlik, xotira samaradorligi, xavfsizlik va qo'llab-quvvatlash qulayligi bilan ham belgilanadi.

An'anaviy dasturlashda algoritmni loyihalash, kod yozish, sintaktik va mantiqiy xatolarni aniqlash, testlash va optimallashtirish katta intellektual mehnatni talab etadi. Generativ sun'iy intellekt, xususan katta til modellari (Large Language Models - LLM), tabiiy til ko'rsatmalari asosida kod yozish, funksiyalarni tushuntirish, testlar tuzish hamda mavjud dasturlarni qayta ishlashga yordam bera oladi.

Chen va hammualliflar Codex modelini HumanEval topshiriqlarida baholash orqali kod generatsiyasida funksional testlashning ahamiyatini ko'rsatgan [1]. Rozière va hammualliflar Code Llama modellarining kodni davom ettirish, to'ldirish va ko'rsatmalarga ko'ra yaratish imkoniyatlarini o'rgangan [2]. Shunga qaramay, sintaktik jihatdan to'g'ri kod algoritmik jihatdan samarasiz yoki ayrim kirish ma'lumotlarida xatoli bo'lishi mumkin.

ECCO tadqiqotida model generatsiyalagan kodni tezlashtirish jarayonida funksional to'g'rilikni saqlash alohida masala ekanligi ta'kidlangan [3]. Shu bois generatsiya va optimallashtirish yagona tekshiriladigan jarayon sifatida tashkil etilishi muhim.

Tadqiqot muammosi - generativ model yaratgan kodda funksional to'g'rilik, hisoblash samaradorligi va xavfsizlik talablarini birgalikda nazorat qiluvchi mexanizmni shakllantirish.

DOI: 10.65322/j.srd.05192026.86

Maqsad - mavjud usullarni tahlil qilib, ko‘p mezonli baholash hamda qayta verifikatsiyaga asoslangan bosqichli algoritmik yondashuv ishlab chiqish. Ilmiy hissasi - optimallashtirishda funksional to‘g‘rilikni majburiy shart sifatida ajratish va tanlashni ko‘p mezonli model orqali asoslash.

Tadqiqot materiallari va metodlari

Tadqiqot generativ kodlash, dasturiy injiniring va avtomatlashtirilgan testlash bo‘yicha ilmiy adabiyotlarning nazariy tahliliga asoslanadi. Qiyosiy tahlil yordamida generatsiya yondashuvlari solishtirildi; algoritmik modellashtirish yordamida jarayonning ketma-ketligi aniqlandi; funksional dekompozitsiya yordamida tekshiruv bloklari ajratildi; ko‘p mezonli baholash yordamida sifat ko‘rsatkichlari tizimlashtirildi. Tadqiqotda bajarilmagan tajriba natijalari keltirilmaydi.

Foydalanuvchi vazifani, kirish-chiqish ma’lumotlarini, dasturlash tilini va nofunksional cheklovlarni belgilaydi. Model shu kontekst asosida ehtimoliy dasturiy yechim hosil qiladi. Jarayon quyidagi ifoda bilan tavsiflanadi:

$$C_0 = G(P, K, L) \tag{1}$$

bu yerda C_0 - dastlabki kod, G - generativ model, P - topshiriq, K - kontekst va cheklovlar, L - dasturlash tili. Topshiriq qanchalik aniq ifodalansa, yechimni belgilangan mezonlarga nisbatan tekshirish shunchalik osonlashadi. Masalan, “saralash dasturini yoz” o‘rniga kirish hajmi, qiymatlar diapazoni, barqarorlik va xotira limiti ko‘rsatilsa, optimallashtirish mezonlari ochiq bo‘ladi.

Dasturiy kodning bajarilish vaqti $T(C)$, xotira sarfi $M(C)$, strukturaviy murakkabligi $K(C)$, xavfsizlik muammolari $S(C)$ va funksional to‘g‘rilik sharti $F(C)$ bilan ifodalanadi. Maqbul yechim quyidagicha tanlanadi:

$$C^* = argmin\ J(C) \tag{2}$$

$$J(C) = \alpha T_n(C) + \beta M_n(C) + \gamma K_n(C) + \delta S_n(C) \tag{3}$$

Cheklovlar: $F(C) = 1;\ T(C) \leq T_{max};\ M(C) \leq M_{max};\ \alpha + \beta + \gamma + \delta = 1.$

T_n, M_n, K_n va S_n - bir yo‘nalishda normallashtirilgan ko‘rsatkichlar; α, β, γ, δ - vaznlar. $F(C)=1$ belgilangan testlardan o‘tilganini anglatadi, biroq barcha mumkin bo‘lgan holatlar uchun formal to‘g‘rilik isboti emas. Muhim tizimlar uchun qo‘shimcha verifikatsiya kerak.

1-bosqich. Talablarni formallashtirish: funksional vazifa, kirish-chiqish shartlari, chegaraviy holatlar va resurs limitlari belgilanadi.

2-bosqich. Dastlabki kodni generatsiyalash: model berilgan talablar asosida dasturiy yechim ishlab chiqadi.

3-bosqich. Statik va funksional tekshiruv: kompilyatsiya, unit-testlar, chegaraviy testlar va xavfsizlik tahlili o‘tkaziladi.

4-bosqich. Iterativ optimallashtirish: test diagnostikasi va resurslar profili modelga qayta uzatilib, yangi variant hosil qilinadi.

5-bosqich. Tanlash va qayta verifikatsiya: talablarga javob beruvchi variantlardan ko‘p mezonli baholash bo‘yicha maqbuli tanlanadi; aks holda oldingi tekshirilgan variant saqlanadi.

Iteratsiya formulasi:

$$C_{i+1} = G(P, C_i, E_i, R_i) \tag{4}$$

DOI: 10.65322/j.srd.05192026.86

E_i - test va statik tahlil xatolari; R_i - ishlash samaradorligi haqidagi teskari aloqa. Jarayon belgilangan iteratsiya limiti tugaguncha, maqsadga erishilguncha yoki sezilarli yaxshilanish to‘xtaguncha davom etadi.

Tadqiqot natijalari

Manbalar tahlili natijasida generatsiya va optimallashtirishning to‘rtta tashkiliy yondashuvi ajratildi (1-jadval).

1-jadval

Dasturiy kodni generatsiyalash va optimallashtirish yondashuvlarining qiyosiy tavsifi

Metod	**Bajarish bosqichlari**	**Erishiladigan natija**
Bir bosqichli	Talabdan bevosita kod yaratish	Soddalik va tezkorlik
Kontekstga asoslangan	Cheklov va namunalardan foydalanish	Talabga moslikni oshiradi
Iterativ	Oldingi variantni qayta ishlash	Xatoni bosqichma-bosqich tuzatish
Test bilan boshqariladigan	Test va benchmark asosida tanlash	Nomuvofiq variantlarni rad etadi

[1–5] dagi yondashuvlar asosida muallifning qiyosiy umumlashtirishi. Taqqoslash miqdoriy tajriba natijasi emas.

Integratsiyalashgan model besh funksional blokdan iborat: talablarni qayta ishlash, generatsiya, verifikatsiya, optimallashtirish, baholash va tanlash. Ularning o‘zaro bog‘liqligi quyidagicha:

Talablar → Kod generatsiyasi → Statik tahlil va testlash → Samaradorlikni o‘lchash → Optimallashtirish → Qayta testlash → Yakuniy kod.

Modelning muhim xususiyati shundaki, optimallashtirilgan kod natija sifatida faqat yangi tekshiruvdan muvaffaqiyatli o‘tganidan keyin qabul qilinadi. Aks holda avvalgi tasdiqlangan variant saqlanadi.

Takrorlangan elementlari mavjud ro‘yxatdan noyob butun sonlarni dastlabki uchrash tartibida ajratish vazifasini ko‘ramiz. Dastlabki variant:

```
def unique_values(data):
    result = []
    for value in data:
        if value not in result:
            result.append(value)
    return result
```

Natijaviy ro‘yxat ichida ketma-ket qidirish eng yomon holatda $O(n^2)$ vaqt murakkabligiga olib keladi. Muqobil yechimda ilgari uchragan elementlar xesh-to‘plam orqali tekshiriladi:

```
def unique_values(data):
    result = []
    seen = set()
    for value in data:
        if value not in seen:
            seen.add(value)
            result.append(value)
    return result
```

DOI: 10.65322/j.srd.05192026.86

Xeshlanadigan qiymatlarda to‘plamga murojaat qilish o‘rtacha *O(1)* vaqt olishi sababli ikkinchi usulning kutiladigan umumiy murakkabligi *O(n)* bo‘ladi. Biroq ikkala usulning qo‘shimcha xotirasi *O(u),* bunda *u* - noyob elementlar soni; ikkinchi variant xeshlanmaydigan obyektlarga umumiy holda tatbiq etilmaydi (2-, 3-jadval).

2-jadval

Noyob elementlarni ajratish algoritmlarining qiyosiy tavsifi

Ko‘rsatkich	**Dastlabki yechim**	**Optimallashtirilgan yechim**
Qidiruv	Ro‘yxatdan ketma-ket izlash	Xesh-to‘plamda tekshirish
Vaqt	$O(n^2)$, eng yomon holatda	*O(n),* o‘rtacha kutiladigan
Xotira	*O(u)*	*O(u)*
Tartib	Saqlanadi	Saqlanadi
Cheklov	Tenglik bo‘yicha solishtirish	Xeshlanadigan qiymatlar

3-jadval

Dasturiy kod sifatini baholash mezonlari va o‘lchash usullari

Mezon	**Mazmuni**	**O‘lchash usuli**
Funksional to‘g‘rilik	Belgilangan talabni bajarish	Unit/integratsion testlar
Tezlik	Bajarilish vaqti	Benchmark va profiler
Xotira	Ajratilgan xotira hajmi	Xotira monitoringi
Murakkablik	Boshqaruv strukturasining murakkabligi	Statik metrikalar
Xavfsizlik	Ma’lum zaifliklar	Statik/dinamik xavfsizlik tahlili
Qo‘llab-quvvatlash	Kodning tushunarliligi	Ekspert bahosi va kod metrikalari

Muhokama

Self-Refine yondashuvi til modelining o‘z natijasiga teskari aloqa berib takroran takomillashtira olishini ko‘rsatadi [4]. Biroq dasturiy kodning haqiqiy to‘g‘riligini tasdiqlash uchun faqat modelning bahosiga ishonish yetarli emas. Taklif etilayotgan modelda tashqi testlash hamda statik tahlil natijalari majburiy hisoblanadi.

ECCO tadqiqotida samaradorlikning oshishi funksional to‘g‘rilikni saqlash bilan birga baholanishi zarurligi ko‘rsatilgan [3]. Mazkur ishda ushbu tamoyil xotira, murakkablik va xavfsizlik mezonlari bilan to‘ldirilib, tanlash masalasi sifatida talqin qilindi. SWE-bench esa real dasturiy injiniring muammolari ko‘pincha bir necha fayl va komponent o‘rtasidagi bog‘liqlikni tushunishni talab etishini ko‘rsatadi [5].

Taklif etilgan modelning cheklovlari quyidagilardan iborat: talabdagi noaniqlik, test qamrovining yetishmasligi, apparat va ish muhiti natijalariga sezgirlik, modelga maxfiy kod yuborish xavflari va iterativ o‘zgarishlarda yangi xatolarning paydo bo‘lishi. Iterativ jarayon har doim ham yaxshiroq yechim bermaydi. Shu bois tasdiqlangan versiyaga qaytish mexanizmi zarur.

Mazkur maqolada integratsiyalashgan algoritmik yondashuv nazariy jihatdan asoslangan, ammo dasturlash tillari va modellar kesimida tajriba o‘tkazilmagan. Kelgusi ishlar uchun Python, C++ va Java topshiriqlarida testlardan o‘tish ulushi, tezlik, xotira, model chaqiruvlari va statistik ahamiyatlilikni bir xil sharoitda o‘lchash lozim.

Xulosa. Generativ sun’iy intellekt tabiiy tildagi talablardan dasturiy kod hosil qilish va mavjud dasturiy yechimlarni qayta ishlab chiqishda istiqbolli texnologik asosdir. Biroq yaratilgan kodning sintaktik to‘g‘riligi uning funksional va ekspluatatsion sifatini kafolatlamaydi.

Tadqiqotda talablarni formallashtirish, kodni generatsiyalash, statik va funksional tekshiruv, iterativ optimallashtirish hamda ko‘p mezonli tanlashdan iborat bosqichli model taklif

DOI: 10.65322/j.srd.05192026.86

etildi. Optimallashtirishda funksional to‘g‘rilikning majburiy cheklov sifatida belgilanishi modelning bosh tamoyilidir.

Algoritmik misol ma’lumotlar tuzilmasini o‘zgartirish hisoblash murakkabligini yaxshilashi mumkinligini ko‘rsatdi, lekin bunday o‘zgarishlar xotira va ma’lumotlar turiga oid cheklovlar bilan birga tekshirilishi lozim. Taklif etilgan modelni amaliy joriy etishdan oldin turli modellar va masalalarda eksperimental verifikatsiya qilish zarur.

Foydalanilgan adabiyotlar

1. Chen M., Tworek J., Jun H. et al. Evaluating Large Language Models Trained on Code. arXiv:2107.03374. 2021. DOI: 10.48550/arXiv.2107.03374. https://arxiv.org/abs/2107.03374

2. Rozière B., Gehring J., Gloeckle F. et al. Code Llama: Open Foundation Models for Code. arXiv:2308.12950. 2023. https://arxiv.org/abs/2308.12950

3. Waghjale S., Veerendranath V., Wang Z., Fried D. ECCO: Can We Improve Model-Generated Code Efficiency Without Sacrificing Functional Correctness? Proceedings of EMNLP. 2024. P. 15362–15376. https://aclanthology.org/2024.emnlp-main.859/

4. Madaan A., Tandon N., Gupta P. et al. Self-Refine: Iterative Refinement with Self-Feedback. Advances in Neural Information Processing Systems. 2023. https://arxiv.org/abs/2303.17651

5. Jimenez C. E., Yang J., Wettig A. et al. SWE-bench: Can Language Models Resolve Real-world GitHub Issues? International Conference on Learning Representations. 2024. https://openreview.net/forum?id=VTF8yNQM66