IQRO JURNALI , IQRO JOURNAL, ИКРО ЖУРНАЛИ
ISSN:2181-4341| wordlyknowledge.uz/index.php/iqro
Volume 23 Issue 01 AVGUST 2026

SUN'IY INTELLEKT YORDAMIDA DASTURIY TA'MINOTNI TURLI PLATFORMALARGA (CROSS-PLATFORM) MOSLASHTIRISHNI AVTOMATLASHTIRISH IMKONIYATLARI

Yunusov Ahrorbek Baxtiyor o'g'li

Turan International University talabasi

O'zbekiston, Namangan

e-mail: akhror111@gmail.com

Annotatsiya: Maqolada sun'iy intellekt (AI) va yirik til modellari (LLM) yordamida dasturiy ta'minotni turli platformalarga (krossplatformalilik) moslashtirishni avtomatlashtirish imkoniyatlari tadqiq etilgan. Avtomatlashtirilgan sintaktik va semantik kod tarjimasi, ko'p agentli tizimlar va kodning bajarilishi (executability) asosida modellarni o'qitish kabi asosiy yondashuvlar tahlil qilingan. Ushbu usullar dasturlash tillari o'rtasidagi sintaktik va semantik farqlarni bartaraf etish, dasturlash xarajatlarini kamaytirish va operatsion tizimlararo foydalanuvchi interfeysini saqlab qolish imkonini beradi.

Kalit so'zlar: krossplatformali ishlab chiqish, sun'iy intellekt, dasturlashni avtomatlashtirish, til modellari, interfeys adaptatsiyasi, kod tarjimasi.

Abstract: The article explores the opportunities for automating software adaptation to various platforms (cross-platform) using artificial intelligence (AI) and large language models (LLMs). Key approaches, such as automated syntactic and semantic code translation, multi-agent systems, and model training based on executability representation, are analyzed. These methods enable overcoming syntactic and semantic discrepancies between programming languages, reducing manual labor and development costs, and preserving user experience when migrating applications across operating systems.

Keywords: cross-platform development, artificial intelligence, programming automation, large language models, UI adaptation, code translation.

Kirish

Zamonaviy jamiyatda dasturlashni avtomatlashtirish vositalarining jadal rivojlanishi raqamli transformatsiyaning asosiy omili bo'lib xizmat qilmoqda, chunki dasturiy ta'minot inson hayotining barcha sohalarini – kundalik mobil ilovalardan tortib murakkab aerokosmik tizimlargacha qamrab olgan. Dasturiy mahsulotlarni yaratishning an'anaviy jarayoni juda yuqori mehnat talab qilishi bilan ajralib turadi, u katta intellektual, moliyaviy va vaqt resurslarini talab qiladi. Bu esa dasturiy ta'minot yaratish jarayonlarini matematik modellashtirish va optimallashtirish vazifasini umumiy raqobatbardoshlikni oshirish uchun o'ta muhim qilib qo'yadi. O'n yilliklar davomida dasturlash evolyutsiyasi apparat darajasidan uzoqlashish (abstraksiya) yo'lidan bordi: 1950-yillardagi mashina kodi va assamblerdan boshlab Fortran va COBOL kabi birinchi yuqori darajali tillarga, 1970-yillardagi tarkibiy (strukturali) dasturlashga va 1980-1990-yillardagi ob'ektga yo'naltirilgan paradigmaga o'tildi [5]. 1995-yilda virtual mashinaga (JVM) ega Java tilining paydo bo'lishi dasturiy kodning platformalararo ko'chib o'tish konsepsiyasiga poydevor yaratdi.

Zamonaviy dasturiy ta’minot sanoatida texnologik stekni tanlash loyihalarning muvaffaqiyati va iqtisodiy samaradorligiga hal qiluvchi ta’sir ko‘rsatadi [4]. Apparat vositalari va operatsion tizimlarning tezkor rivojlanishi dasturiy mahsulotlarni yangi platformalarga ko‘chirish va moslashtirish (porting) zaruratini muntazam ravishda yuzaga keltirmoqda. Biroq, kod bazasini turli muhitlarga qo‘lda moslashtirish juda ko‘p resurs talab qiladigan, uzoq davom etadigan va xatolarga moyil jarayondir. Eskirgan tizimlarni qo‘lda qayta yozishning yuqori qiymatiga tarixiy misol sifatida Avstraliya Hamdo‘stlik Bankining (Commonwealth Bank of Australia) o‘z platformasini COBOL tilidan Java-ga o‘tkazish uchun taxminan 750 million dollar va dasturchilar guruhlarining besh yillik mehnatini sarflaganini keltirish mumkin [3]. Bundan tashqari, krossplatformali moslashtirish foydalanuvchining grafik interfeysini (UI/UX) ko‘chirish zarurati bilan yanada murakkablashadi. Dasturchilar iOS uchun Apple Human Interface Guidelines va Android uchun Material Design kabi operatsion tizimlarning rasmiy qoidalaridagi fundamental farqlarni hisobga olgan holda, turli qurilmalarda dasturlarning vizual va funksional o‘xshashligini ta’minlashlari kerak. Shu bilan birga, 60% dan ortiq foydalanuvchilar o‘zlari ishlatadigan qurilmaga qarab interfeysi va ishlash mantig‘i farq qiladigan ilovalarni salbiy qabul qilishadi [6].

Generativ sun’iy intellekt (AI) va yirik til modellari (LLM) texnologiyalarining paydo bo‘lishi dasturlashni avtomatlashtirishda sifat jihatidan yangi bosqichni ochdi. Intellektual yordamchilarni joriy etish shablon kodlarni yaratish, refaktoring va sintaktik xatolarni tuzatish kabi kundalik vazifalarni avtomatlashtirishga imkon beradi, bu esa dasturlashni inson va neyrotarmoqning samarali hamkorligiga aylantiradi. Generativ sun’iy intellektni qo‘llashning eng istiqbolli yo‘nalishi turli dasturlash tillari va texnologik steklar o‘rtasida, masalan, React-dan Flutter-ga dasturiy kodni avtomatlashtirilgan tarzda tarjima qilishdir [4]. Shunga qaramay, mavjud tijoriy va ochiq LLMlar ko‘pincha tarjima qilingan kodning to‘liq semantik ekvivalentligi va ishlashini kafolatlamaydi, ko‘plab o‘ziga xos xatolarga yo‘l qo‘yadi [3]. Shunday qilib, InterTrans tranzit tarjimonlari, TransAgent ko‘p agentli tizimlari va kodning bajarilishi asosida modellashtirilgan ExeCoder kabi ixtisoslashgan AI arxitekturalari yordamida dasturiy ta’minotni platformalararo avtomatik moslashtirish aniqligini oshirish imkoniyatlarini tadqiq qilish o‘ta dolzarb ilmiy va amaliy yo‘nalish hisoblanadi.

Tadqiqot metodlari

Dasturiy ta’minotni turli platformalarga moslashtirishni avtomatlashtirish va to‘liq funksional o‘xshashlikni ta’minlash uchun kompleks metodologik yondashuv taklif etiladi. Ushbu yondashuv kodni mashina tiliga tarjima qilishning zamonaviy konsepsiyalarini, verifikasiyalashning intellektual ko‘p agentli tizimlarini va dasturning bajarilishi (executability) haqidagi bilimlarni taqdim etishning ilg‘or arxitekturalarini birlashtiradi.

Oraliq tillar yordamida kodni tranzit tarjima qilish (InterTrans). Dasturiy kodni manba tilidan maqsadli tilga to‘g‘ridan-to‘g‘ri tarjima qilish ko‘pincha jiddiy sintaktik va semantik nomuvofiqliklarga duch keladi. Ushbu muammoni hal qilish uchun yirik til modellarining ko‘p tilli qobiliyatlariga asoslangan oraliq tranzit tarjimaning InterTrans usuli qo‘llaniladi [1]. Tarjima jarayoni ikki asosiy bosqichdan iborat: 1) tranzit ketma-ketliklarni rejalashtirish: Tree of Code Translation (ToCT) algoritmi yordamida oraliq tillar orqali tarjima qilishning mumkin bo‘lgan yo‘llari daraxti quriladi (masalan, Python-dan C++ tiliga tarjima qilish Java tilidagi oraliq generatsiya orqali amalga oshiriladi), bu sintaktik farqlarni kamaytirishga imkon beradi; 2) bosqichma-bosqich verifikatsiya: oraliq tarjimalar qat’iy tartibda tekshiriladi, bu esa har bir tranzit o‘tishda kod semantikasining bosqichma-bosqich saqlanishini ta’minlaydi. ToCT-dan foydalanish to‘g‘ridan-to‘g‘ri tarjimaga nisbatan hisoblash aniqligini (Computation Accuracy) 18.3% dan 43.3% gacha oshirish imkonini beradi [1].

Ko‘p agentli verifikasiya va bajarilishni muvofiqlashtirish (TransAgent). Kod generatsiyasi jarayonida yuzaga keladigan sintaktik va mantiqiy xatolarni bartaraf etish uchun TransAgent avtomatik xatolarni tuzatish ko‘p agentli tizimi joriy etiladi [2]. Tarjima va disk raskadrovka (debugging) mojarolari uchun bitta modeldan foydalanish o‘rniga, tizim vazifalarni LLM bazasidagi to‘rtta ixtisoslashgan agent o‘rtasida taqsimlaydi: Boshlang‘ich tarjima agenti (Initial Code Translator) kod bazasining asosiy tarjimasini amalga oshiradi; Sintaktik xatolarni tuzatuvchi agent (Syntax Error Fixer) kompilyatsiya xatolarini aniqlaydi va tuzatadi; Kodni muvofiqlashtiruvchi agent (Code Aligner) manba va maqsadli dasturlarning bloklarini solishtiradi; Semantik xatolarni tuzatuvchi agent (Semantic Error Fixer) dasturning bajarilishi jarayonida mantiqiy xatolarni bartaraf etadi. TransAgent-ning asosiy usuli manba va maqsadli kodlar o‘rtasida bajarilishni batafsil muvofiqlashtirishdir (fine-grained execution alignment) [2]. Bu o‘zgaruvchilar holati va bajarilish jarayonini taqqoslash orqali maqsadli dasturdagi xatoli kod bloklarini aniqlash imkonini beradi, bu esa disk raskadrovka murakkabligini sezilarli darajada kamaytiradi va mantiqni tiklash aniqligini oshiradi.

Bajarilish holatini taqdim etishga asoslangan progressiv o‘qitish (ExeCoder). Oddiy til modellari kodning matnli semantikasini o‘rganadi, ammo uning ishlash holati haqidagi ma’lumotlarni hisobga olmaydi. Tadqiqot doirasida kodning bajarilishi haqidagi bilimlarni (executability representations), ya’ni funksional semantika, sintaktik struktura va o‘zgaruvchilar bog‘liqligini integratsiyalashgan ExeCoder arxitekturasi qo‘llaniladi [3]. Ushbu yuqori darajadagi bilimlarni olish uchun statik tahlil vositalaridan foydalaniladi. Funksional semantika kodning vazifasini tavsiflash ko‘rinishida kodlanadi. Sintaktik struktura tree-sitter tahlilchisi yordamida abstrakt sintaktik daraxt (AST) shaklida olinadi. Model tomonidan graf strukturasini qabul qilish uchun AST soddalashtiriladi va tabiiy tilda aloqalar ko‘rsatilgan holda tasvirlanadi. O‘zgaruvchilar bog‘liqligi ma’lumotlar oqimi grafi (DFG) orqali taqdim etiladi. Bilimlarni integratsiya qilish uchun to‘rt bosqichli sozlashdan iborat bo‘lgan progressiv o‘qitish strategiyasi (Progressive Executability Representation Learning — PERL) qo‘llaniladi [3].

Grafik foydalanuvchi interfeysining (UI/UX) arxitekturaviy adaptatsiyasi. Ilovalarni platformalararo avtomatlashtirilgan ko‘chirish foydalanuvchi interfeysini moslashtirishni ham o‘z ichiga oladi. Asosiy qiyinchilik dizaynning unifikatsiyasi va platformaga xos bo‘lgan qoidalar (iOS uchun Apple HIG va Android uchun Material Design) o‘rtasidagi muvozanatni saqlashdir [6]. Interfeysni avtomatlashtirilgan konvertatsiya qilish (masalan, React-dan Flutter-ga) metodologiyasi quyidagilarni nazarda tutadi: 1) Figma-ning unifikatsiyalangan komponentlari asosida yagona dizayn tizimidan foydalanish; 2) turli ruxsatlar, ekran o‘lchamlari va kiritish usullariga moslashadigan moslanuvchan (adaptiv) maket yaratish; 3) manba platformaning interfeysini maqsadli platformaning ekvivalent elementlariga avtomatik ravishda o‘tkazish uchun AI yordamida UI komponentlarini yaratish [4, 6].

Tadqiqot natijalari

Sun’iy intellekt va ExeCoder arxitekturasi asosida ishlab chiqilgan yondashuv samaradorligini har tomonlama baholash uchun TransCoder-test-X o‘zgartirilgan test ma’lumotlar to‘plamida eksperimental tadqiqot o‘tkazildi [3]. Ushbu test to‘plami C++, Java va Python tillaridagi parallel masalalarning 948 ta yechimini o‘z ichiga oladi va dasturning ishlashini avtomatik ravishda tekshirish uchun modulli testlar bilan ta’minlangan.

Avtomatlashtirilgan platformalararo moslashtirish sifati ikki turdagi ko‘rsatkichlar yordamida baholandi: 1) moslik ko‘rsatkichlari (match-based): sintaktik moslikni baholash uchun Exact Match (EM), BLEU va CodeBLEU; 2) bajarilish ko‘rsatkichlari (execution-based): kompilyatsiya aniqligi (Compilation Accuracy, CA), holatlar bo‘yicha hisoblash aniqligi (Case

Computational Accuracy, CCA) va test to‘plamlaridagi umumiy hisoblash aniqligi (Test Computational Accuracy, TCA) [3].

Modellarni bajarilish ko‘rsatkichlari bo‘yicha eksperimental taqqoslash natijalari 1-jadvalda keltirilgan.

1-jadval. TransCoder-test-X to‘plamida bajarilish ko‘rsatkichlari bo‘yicha baholash natijalari.

Model	**Kompilyatsiya aniqligi (CA, %)**	**Holatlar bo‘yicha aniqlik (CCA, %)**	**Umumiy test aniqligi (TCA, %)**
ExeCoder (Ours)	92.68	87.69	83.04
GPT-4	90.63	86.06	81.56
GPT-4o	89.54	84.73	80.01
GPT-3.5	85.37	80.40	76.30
Deepseek-Coder	81.82	76.73	72.22
Magicoder	75.36	70.56	66.20

1-jadvalda ko‘rsatilgan natijalardan ko‘rinib turibdiki, taklif qilingan ExeCoder modeli barcha ko‘rsatkichlar bo‘yicha ochiq va yopiq kodli muqobillaridan ustundir [3]. Ishlab chiqilgan modelning o‘rtacha kompilyatsiya aniqligi 92.68% ni tashkil etadi, bu eng kuchli yopiq tijorat modeli GPT-4 ko‘rsatkichidan 2.05% ga yuqoridir. Testlarni bajarishning o‘rtacha aniqligi (TCA) 83.04% ni tashkil etib, platformalararo kodning barqaror ishlashini ko‘rsatmoqda.

Har bir bajarilish ko‘rsatkichining (funksional semantika FS, sintaktik struktura SS va o‘zgaruvchilar bog‘liqligi VD) alohida hissasini baholash uchun ablatsion tadqiqot o‘tkazildi. Modelning turli konfiguratsiyalari tahlili natijalari 2-jadvalda taqdim etilgan.

2-jadval. Kodning bajarilishi haqidagi bilimlarning hissasini baholash bo‘yicha ablatsion tadqiqoti natijalari.

Konfiguratsiya	**CA (O‘rtacha), %**	**CCA (O‘rtacha), %**	**TCA (O‘rtacha), %**
Code (Boshlang‘ich)	91.01	85.81	80.58
with FS (Semantika)	91.54	86.06	81.01
with SS (Sintaksis AST)	92.64	87.36	82.32
with VD (Bog‘liqlik DFG)	92.39	86.91	81.73

2-jadval tahlili shuni ko‘rsatadiki, modelni faqat manba kodida o‘qitish (Code konfiguratsiyasi) 80.58% darajasida boshlang‘ich TCA aniqligini ta’minlaydi [3]. Funksional-semantik bilimlarning integratsiyasi (with FS) bajarilish aniqligini 0.43% ga oshiradi. Eng muhim hissa abstrakt sintaktik daraxtga asoslangan sintaktik taqdimot (with SS) tomonidan qo‘shiladi, bu esa bajarilish aniqligini (TCA) 1.74% ga (82.32% gacha) va kompilyatsiya aniqligini (CA) 1.63% ga oshiradi. O‘zgaruvchilar bog‘liqligini (with VD) hisobga olish ham test aniqligini 1.15% ga oshirib, mantiqiy xatolarni rad etadi [3].

Muhokama

Olingan eksperimental natijalar kodning bajarilishi haqidagi bilimlarni (executability representations) integratsiyalash krossplatformali dasturiy ta’minotni tarjima qilishda hal qiluvchi omil ekanligini tasdiqlaydi [3]. Umumiy maqsadli til modellariga asoslangan an’anaviy yondashuvlar ko‘pincha tillararo sintaktik va semantik tafovutlar tufayli muvaffaqiyatsizlikka uchraydi [1, 2]. Masalan, tipizatsiya va xotirani boshqarishdagi farqlarni hisobga olmasdan mantiqni oddiy ko‘chirish 15% dan 40% gacha bo‘lgan holatlarda xatolarga olib keladi, bu esa qo‘lda disk raskadrovka qilishni talab qiladi.

ExeCoder-ning taklif qilingan usuli ushbu cheklovlarni bartaraf etadi. Ablatsion tahlil ko‘rsatganidek, eng yuqori aniqlik AST yordamida olinadi. Bu maqsadli tilning sintaktik qoidalarini bilish modelga to‘g‘ri konstruktsiyalarni yaratish va kalit so‘zlar ziddiyatidan qochish imkonini berishi bilan izohlanadi.

Bundan tashqari, TransAgent kabi ko‘p agentli tizimlar sinergik samara beradi. Vazifalarning ixtisoslashgan agentlar o‘rtasida taqsimlanishi xatolarni mahalliy darajada aniqlash imkonini beradi. Bajarilish holatlarini taqqoslash xatolarni qidirish maydonini qisqartiradi va mantiqni tiklash samaradorligini oshiradi [2].

Ushbu yondashuvning amaliy ahamiyati, ayniqsa, mobil ilovalarni React-dan Flutter-ga ko‘chirishda yaqqol namoyon bo‘ladi. AI vositalaridan foydalanish biznes mantig‘ini va foydalanuvchi interfeysi elementlarini minimal og‘ishlar bilan avtomatik ravishda tarjima qilishga imkon beradi [4]. Shu bilan birga, krossplatformalilikning asosiy muammosi iOS va Android operatsion tizimlarining rasmiy qoidalariga rioya qilgan holda, foydalanuvchi interfeysining (UI/UX) bir xilligini saqlab qolishdir. Ushbu jarayonni Figma-ning yagona dizayn tizimlari va AI yordamida adaptiv komponentlar yaratish orqali avtomatlashtirish, turli qurilmalarda bir xil ilovalarni kutayotgan 60% dan ortiq foydalanuvchilarning talablarini qondiradi [6].

Xulosa

Tadqiqot doirasida sun’iy intellekt yordamida dasturiy ta’minotni turli platformalarga moslashtirishni avtomatlashtirish imkoniyatlari batafsil o‘rganildi. InterTrans tranzit tarjimasi, TransAgent ko‘p agentli tizimi va ExeCoder progressiv o‘qitish usuli tahlil qilindi.

TransCoder-test-X to‘plamidagi eksperimental baholash ExeCoder modelining barcha muqobillaridan, shu jumladan GPT-4 va GPT-4o-dan ustunligini ko‘rsatdi [3]. U 92.68% kompilyatsiya aniqligi va 83.04% test aniqligiga erishdi. Ablatsion tahlillarda AST va DFG kabi tuzilmaviy bilimlarning mantiqiy xatolarning oldini olishdagi hal qiluvchi rolini isbotladi.

Intellektual avtomatlashtirishni joriy etish platformalararo ishlab chiqish va dasturlarni ko‘chirish xarajatlarini sezilarli darajada kamaytiradi, dastur mantig‘ining funksional ekvivalentligini kafolatlaydi [4, 6]. Istiqbolli yo‘nalishlar sifatida yanada ixcham kod bog‘liqligi

modellarini yaratish va qo‘llab-quvvatlanadigan dasturlash tillari hamda freymvorklar doirasini kengaytirish rejalashtirilgan.

Foydalanilgan adabiyotlar ro‘yxati:

1. Macedo M., Tian Y., Nie P., Cogo F.R., Adams B. InterTrans: Leveraging Transitive Intermediate Translations to Enhance LLM-based Code Translation // arXiv preprint arXiv:2411.01063. – 2024.
2. Yuan Z., Chen W., Wang H., Peng X., Chen Z., Lou Y. TransAgent: Enhancing LLM-Based Code Translation via Fine-Grained Execution Alignment // arXiv preprint arXiv:2409.19894. – 2024.
3. He M., Chen Y., Yang F., Zhao P., Yin W., Kang Y., Lin Q., Rajmohan S., Zhang D. ExeCoder: Empowering Large Language Models with Executability Representation for Code Translation // Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP). – 2025. – Pp. 7099-7109.
4. Ms. Namrata Soni, Ms. Dipali Kirange, Aftab Mulani. Leveraging Artificial Intelligence for Technology Stack Conversion // International Journal of Innovative Research in Technology (IJIRT). – 2024. – Vol. 10, Issue 11. – Pp. 342-348.
5. Бершадский А.М., Евсеева Ю.И., Гудков А.А. Автоматизация программирования: от первых компиляторов до генеративного искусственного интеллекта // Моделирование, оптимизация и информационные технологии. – 2025. – Т. 13, № 4. – URL: https://moitvivt.ru/ru/journal/pdf?id=2031 (дата обращения: 13.08.2026).
6. Асадуллин Р.М., Копылов Я.К. Важность проектирования идентичного интерфейса кроссплатформенного приложения // Вестник науки. – 2025. – № 7 (88), т. 2. – С. 372-377.