УНИВЕРЗИТЕТ „ГОЦЕ ДЕЛЧЕВ” – ШТИП
ФАКУЛТЕТ ЗА ИНФОРМАТИКА

МАГИСТЕРСКИ ТРУД

НИКОЛА ПОП ТОМОВ

ИСТРАЖУВАЊЕ ЗА ЕФИКАСНОСТА НА ГЕНЕРИРАЊЕ КОД СО GPT (GENERATIVE PRE-TRAINED TRANSFORMER) ЗА ЗАДАЧИ ВО СОФТВЕРСКОТО ИНЖЕНЕРСТВО

Штип, 2023

Комисија за оценка и одбрана

Ментор: проф. д-р Александар Крстев
Универзитет „Гоце Делчев“ - Штип

Член: вон. проф. д-р Васко Кокаланов
Универзитет „Гоце Делчев“ - Штип

Член: доц. д-р Зоран Златев
Универзитет „Гоце Делчев“ - Штип

Датум на одбрана: 5.10.2023г.

Објавени трудови:

Pop Tomov, N., Kokalanov, V., & Koceski, S.: Deep Learning-Based Real-Time Body Measurements Using Device Camera. Proceedings of the Mediterranean Embedded Computing Resources (MECO) Conference, 2023.

Pop Tomov, N., Koceska, N., & Koceski, S.: The Use of Augmented Reality in Geometry Teaching. Proceedings of the International Conference "Information Technology and Education Development" (ITRO), 2022

ИСТРАЖУВАЊЕ ЗА ЕФИКАСНОСТА НА ГЕНЕРИРАЊЕ КОД СО GPT (GENERATIVE PRE-TRAINED TRANSFORMER) ЗА ЗАДАЧИ ВО СОФТВЕРСКОТО ИНЖЕНЕРСТВО

Краток извадок:

Брзиот напредок на моделите за машинско учење кои користат обработка на природен јазик предизвика интерес за нивната потенцијална примена за генерирање код во областа софтверското инженерство. Оваа теза ја испитува ефикасноста на GPT (Генеративен претходно-трениран трансформер) моделите во автоматизирањето на задачите за кодирање, оценувајќи го нивниот потенцијал за подобрување на ефикасноста, точноста и квалитетот на кодот.

Студијата вклучува длабинска анализа на моделите базирани на GPT, вклучувајќи ги GPT-3.5 и GPT-4, и ги оценува нивните перформанси врз различни програмски јазици и задачи. Таа користи квалитативни и квантитативни мерки и увиди базирани на анкети за да обезбеди сеопфатна проценка за ефикасноста на овие модели во генерирањето код.

Првичните наоди сугерираат дека иако GPT моделите покажуваат потенцијал за производство на точен, синтаксички прецизен код за добро специфицирани задачи, нивната ефикасност се намалува кога се соочуваат со двосмислени или сложени описи на задачи. Овие резултати ги истакнуваат и можностите и ограничувањата на алатките за генерирање код базирани на GPT, што укажува на потребата за пософистицирани модели способни да се справат со сложеноста на софтверското инженерство.

Истражувањето, исто така, ги истражува импликациите на генерирањето код засновано на GPT врз работните текови на софтверското инженерство, дискутирајќи ги можните промени во улогите на софтверските инженери. Анализиран е потенцијалот овие алатки за вештачка интелигенција да извршуваат секојдневни задачи, со што ќе им се овозможи на инженерите да се фокусираат на дизајн на високо ниво и на стратегиско одлучување. Истовремено, се анализираат предизвиците, вклучително и неопходноста од

темелно тестирање и верификација на кодот генериран со овие алатки и потенцијалните етички импликации на вештачката интелигенција во развојот на софтверот.

Оваа студија придонесува за разбирање на практичните апликации и ограничувањата на ВИ во развојот на софтвер и изгледите за подобрување. Се потврдува дека иако моделите базирани на GPT покажуваат значителен потенцијал во автоматизирањето на специфичните аспекти на софтверското инженерство, човечкиот придонес и надзорот остануваат клучни за справување со сложени, креативни и двосмислени задачи.

Клучни зборови:
Вештачка интелигенција, машинско учење, обработка на природни јазици, автоматско програмирање, квалитет на код, ефикасност на работниот тек, етика во вештачката интелигенција.

INVESTIGATING THE EFFECTIVENESS OF GPT-BASED CODE GENERATION FOR SOFTWARE ENGINEERING TASKS

Abstract:

The rapid progress of machine learning models using natural language processing has sparked interest in their potential application for code generation in the field of software engineering. This thesis examines the effectiveness of GPT (Generative Pre-Trained Transformer) models in automating coding tasks, evaluating their potential to improve code efficiency, accuracy and quality.

The study includes an in-depth analysis of GPT-based models, including GPT-3.5 and GPT-4, and evaluates their performance on various programming languages and tasks. It uses qualitative and quantitative measures and survey-based insights to provide a comprehensive assessment of the effectiveness of these models in generating code.

Initial findings suggest that although GPT models show the potential to produce correct, syntactically precise code for well-specified tasks, their efficiency decreases when faced with ambiguous or complex task descriptions. These results highlight both the capabilities and limitations of GPT-based code generation tools, pointing to the need for more sophisticated models capable of handling the complexity of software engineering.

The research also explores the implications of GPT-based code generation on software engineering workflows, discussing possible changes in the roles of software engineers. The potential for these AI tools to perform everyday tasks is analyzed, allowing engineers to focus on high-level design and strategic decision-making. At the same time, challenges are analyzed, including the necessity of thorough testing and verification of the code generated with these tools and the potential ethical implications of artificial intelligence in software development.

This study contributes to the understanding of the practical applications and limitations of AI in software development and the prospects for improvement. It is confirmed that although GPT-based models show considerable potential in

automating specific aspects of software engineering, human input and supervision remain crucial for dealing with complex, creative and ambiguous tasks.

Keywords: Artificial Intelligence, Machine Learning, Code Generation, Natural Language Processing, Automated Programming, Code Quality, Workflow Efficiency, Ethics in AI.

СОДРЖИНА:

1. ВОВЕД

1.1. Позадина

Брзиот напредок на технологиите за вештачка интелигенција (AI) и машинско учење (ML) има големо влијание врз многу сектори, вклучително и софтверското инженерство. Во последниве години, постигнат е значителен напредок во примената на вештачката интелигенција во сфери од софтверското инженерство како на пр. во дизајн и развој на софтвер, тестирање и одржување и слично.

Едно од најновите ветувачки случувања во оваа област е примената на „Генеративните претходно обучени трансформери“ (GPT) за генерирање код. GPT, модел развиен од OpenAI, е широко употребуван за задачи за обработка на природен јазик поради неговата способност да генерира кохерентен и контекстуално соодветен текст. Потенцијалот на моделите базирани на GPT во софтверското инженерство, особено во задачите за генерирање код, предизвика значителен интерес во истражувачката заедница.

Автоматското генерирање кодови ветува значително намалување на количината на потребното рачно кодирање, а со тоа потенцијално зголемување на ефикасноста на процесот на развој на софтвер. Сепак, иако потенцијалот на GPT моделите во генерирањето кодови е очигледен, степенот на нивната ефикасност и ефикасност е тема што треба дополнително да се истражува. Прелиминарните студии сугерираат дека генераторите на кодови базирани на GPT можат да произведат синтаксички точен код, но севкупниот квалитет, исправноста и корисноста на генерираниот код не се сеопфатно оценети.

Оваа студија има за цел да ја истражи ефективноста на генерирањето код засновано на GPT во задачите за софтверско инженерство. Таа има за цел да обезбеди квантитативна и квалитативна анализа на перформансите на генераторите на код базирани на GPT, со што ќе придонесе за разбирање на применливоста и ограничувањата на овие модели на вештачка интелигенција во софтверското инженерство.

Преку ова истражување, се надеваме дека ќе обезбедиме сознанија кои би можеле да им помогнат на развивачите на софтвер, проект менаџерите и организациите, во носењето на одлуки во врска со усвојувањето на генератори на кодови базирани на GPT. Исто така, се надеваме дека наодите од ова истражување ќе бидат основа за понатамошни студии во оваа област на искористување на вештачка интелигенција во софтверското инженерство.

1.2. Дефинирање на проблемот

Соочени со растечката сложеност на задачите за софтверско инженерство, зголемен е интересот за искористување на вештачката интелигенција (ВИ) за да се зголеми човечкиот капацитет и да се подобри ефикасноста. Една од технологиите за вештачка интелигенција што се покажаа како ветувачки секако е Generative Pretrained Transformer (GPT), модел кој е првично развиен за задачи за обработка на природен јазик. Неодамнешните апликации на овој модел во областа на софтверското инженерство, особено за автоматско генерирање кодови, укажуваат на потенцијалот, но исто така откриваат различни предизвици кога ставува збор за генерирање код.

И покрај очекувањата околу способноста на вештачката интелигенција да автоматизира одредени аспекти на кодирањето, степенот до кој генерирањето код засновано на GPT може ефективно и сигурно да придонесе за задачите за софтверско инженерство останува нејасен. Проблемот лежи во недостатокот на сеопфатно разбирање на можностите и ограничувањата на генерирањето код засновано на GPT во контексти на софтверско инженерство во реалниот свет. Постои потреба од длабинска анализа за ефективноста на овие модели, истражувајќи ја не само нивната способност да генерираат синтаксички точен код, туку и севкупниот квалитет на генерираниот код, колку време заштедува, и колку добро се интегрира во постоечките работни текови за развој на софтвер.

Покрај тоа, додека GPT моделите се обучени за широк опсег на задачи за софтверско инженерство, нивната изведба не е подеднакво проучувана кај различни типови задачи, програмски јазици или нивоа на сложеност на задачите. Ова покренува прашања за генерализираноста на пријавените

резултати и оптималните случаи на употреба за алатките за генерирање код базирани на GPT.

Оваа студија има за цел да ги разреши овие нејаснотии со спроведување на сеопфатна евалуација на генерирањето код засновано на GPT, со што ќе придонесе за разбирање на тоа како овие модели на вештачка интелигенција можат најдобро да се искористат во областа на софтверското инженерство.

1.3. Цели на истражувањето

Примарната цел на оваа студија е сеопфатно да ја истражи ефективноста на генерирањето код засновано на GPT во задачите на софтверското инженерство. За да се постигне оваа главна цел, студијата ќе се фокусира на следните специфични цели:

- **Разбирање на можностите на генерирањето код заснован на GPT**: Оваа цел вклучува проценка на перформансите на моделите базирани на GPT во генерирањето синтаксички правилен и семантички точен код низ различни задачи и програмски јазици. Ова вклучува и поправање грешки, рефакторирање на код, како и генерирање тест-случаи во различни програмски јазици.
- **Оценување на квалитетот на генерираниот код:** Студијата има за цел да го оцени квалитетот на кодот генериран од моделите базирани на GPT користејќи стандардни метрики за квалитет на софтверот како што се читливост, одржливост, ефикасност и доверливост.
- **Проценка за временската ефикасност на генерирањето код засновано на GPT**: Оваа цел се обидува да го одреди степенот до кој генерирањето код базирано на GPT може да го забрза процесот на развој на софтвер, споредувајќи го времето потребно за рачно завршување на задачите наспроти користењето на автоматско генерирање код.
- **Истражување за интеграцијата на генерирањето код засновано на GPT во постоечките работни текови за развој на софтвер:** Ова вклучува истражување како алатките за генерирање код базирани на GPT може да се вклучат во тековните практики и работни текови за

развој на софтвер и потенцијалното влијание врз динамиката на тимот и управувањето со проекти.

- **Идентификација на потенцијалните случаи на употреба и ограничувањата на генерирањето код засновано на GPT:** Студијата ќе има за цел точно да ги лоцира задачите и контекстите во кои генерирањето код базирано на GPT се истакнува, обезбедувајќи препораки за оптимална употреба на овие алатки во софтверското инженерство.

Со постигнување на овие цели, оваа студија има за цел да понуди сеопфатно разбирање на предностите, ограничувањата и случаите на најдобра употреба на генерирање код базиран на GPT во областа на софтверското инженерство. Стекнатото знаење може да биде од непроценливо значење за развивачите на софтвер, проект-менаџерите и организациите кои размислуваат за усвојување на вештачката интелигенција и истражувачите во оваа област.

1.4. Важност на истражувањето

Оваа студија е важна за повеќе области и обезбедува вредни придонеси во академската сфера и практичните апликации подеднакво.

Со ова истражување се стремиме да го унапредиме научното разбирање на улогата на вештачката интелигенција, особено GPT моделите, во задачите за софтверско инженерство. Преку сеопфатната проценка на генерирањето код засновано на GPT, студијата ќе фрли светлина врз можностите и ограничувањата на овие модели, со што ќе се обезбеди солидна емпириска основа што го збогатува тековниот дискурс во ова поле кое секојдневно расте.

Студијата има значителни импликации за развивачите на софтвер и организациите кои размислуваат за интеграција на вештачката интелигенција во нивните работни процеси. Со откривање на придобивките и потенцијалните предизвици од користењето генератори на код базирани на GPT, наодите од оваа студија може да им помогнат на засегнатите страни да донесат информирани одлуки во врска со усвојувањето на вештачката интелигенција,

потенцијално поттикнувајќи поефективни и поефикасни процеси за развој на софтвер.

За развивачите на софтвер и истражувачите вклучени во креирањето и усовршувањето на алатките базирани на GPT, оваа студија претставува вреден ресурс. Со истакнување на силните страни и прецизирање каде се неопходни подобрувања, наодите може да бидат од корист за понатамошниот развој во областа.

Доколку генерирањето код засновано на GPT се покаже ефективно за одредени задачи, тоа би можело да има значително влијание врз практиките на софтверското инженерство. Автоматизирањето на делови од процесот на кодирање потенцијално може да ја зголеми ефикасноста и да го намали ризикот од човечка грешка, што ќе доведе до револуција во начинот на кој се развива софтверот.

Оваа студија се обидува да обезбеди сознанија кои би можеле да ги одредат тековите на софтверското инженерство и истражувањето на вештачката интелигенција, да влијаат на одлуките на развивачите на софтвер и организациите и на крајот да поттикнат напредок во областа на развој на софтвер.

2. ПРЕГЛЕД НА ЛИТЕРАТУРА

2.1. Вештачката интелигенција во софтверското инженерство

Вештачката интелигенција (AI) сè повеќе е присутна во различни аспекти на софтверското инженерство, со алатки како ChatGPT, Copilot, Tabnine и Amazon CodeWhisperer кои ја трансформираат сферата за развој на софтвер. Овие алатки базирани на вештачка интелигенција придонесуваат во зголемување на ефикасноста и продуктивноста во процесот на развој на софтвер.

Примената на вештачката интелигенција во софтверското инженерство опфаќа бројни фази од процесот на развој на софтвер, а алатките за вештачка интелигенција можат да ги подобрат овие фази на следниве начини:

- **Дефинирање и генерирање автоматски тестови:** ВИ може да го направи овој процес попрецизен. Некои од алатките кои се корисни за оваа проблематика се OpenAI Codex with Selenium, Amazon CodeWhisperer и др.
- **UI/UX дизајн**: Со можностите на алатките за вештачка интелигенција како што е ChatGPT, дизајнерите можат да добијат помош во градењето на кориснички интерфејси.
- **Дефинирање за архитектура:** Иако вештачката интелигенција сè уште не може да ги процени компромисите помеѓу различните архитектурни одлуки, може да помогне со сугерирање на релевантни услуги од cloud провајдери или пресметување на вкупниот трошок на сопственост (TCO) на целната архитектура.
- **Кодирање:** Вештачката интелигенција може да помогне во генерирање фрагменти од код врз основа на дадени инструкции, олеснување на прототиповите и разработка на различни идеи. Сепак, улогата на инженерите во проверката и полирањето на кодот останува суштинска.
- **Интеграции:** Алатките за вештачка интелигенција како Copilot можат да помогнат во развојот на API интеграции, работа што не е воопшто едноставна.
- **Acceptance testing:** ВИ може да им помогне на луѓето брзо да ги прифатат сите аспекти на ИТ-производот, минимизирајќи ги деловните ризици и обезбедувајќи целосна транспарентност на прифаќањето на засегнатите страни.
- **Deployment**: Алатките засновани на вештачка интелигенција можат да помогнат со верификација на деплојменти и кратење на времето потребно за deployment. Тие, исто така, можат да помогнат и во post-deployment фазата, идентификувајќи ги грешките и откривајќи абнормалности со анализа на системските логови.

Се очекува вештачката интелигенција да ја трансформира иднината на софтверското инженерство. Процесот може да еволуира во две различни фази: креативна фаза и фаза на испорака. Во креативната фаза, ќе биде потребно човечко вклучување за интеракција со алатките за вештачка интелигенција, искористувајќи го знаењето за деловните практики за да се пренесат

информации до вештачката интелигенција. Во фазата на испорака, вештачката интелигенција ќе биде од корист при генерирање, тестирање и deployment на кодот, при што човечките инженери ќе го прегледуваат и усовршуваат самиот код.

Различни студии ја покажаа ефективноста на алатките за вештачка интелигенција во развојот на софтвер. На пример, Џонатан Буркет, висок инженерски менаџер во Duolingo Inc., изјави дека Copilot го направил 25% поефикасен. Еден труд на истражувачи од Мајкрософт и МИТ, исто така, откри дека програмерите кои користат алатки за вештачка интелигенција можеле да ги завршат своите задачи 55,8% побрзо од претходно.

Потребно е понатамошно истражување за да се навлезе подлабоко во специфичните алатки за генерирање кодови базирани на GPT и нивната ефикасност во задачите за софтверско инженерство, вклучувајќи сеопфатен преглед на постоечката академска литература и емпириски студии.

2.2. Претходни истражувања за генерирање на код

Областа за генерирање кодови бележи значителен напредок во текот на годините, со различни истражувања, од традиционални методи, па сè до поновите техники со користење на вештачка интелигенција.

Традиционалните методи за генерирање кодови се опширно проучувани, со фокус на автоматизирање на процесот на пишување код за да се зголеми ефикасноста и да се намалат грешките. На пример, студијата со наслов „Automated Refactoring of Legacy Java Software to Default Methods“ ја истражува употребата на автоматско рефакторирање за трансформирање на наследениот Java софтвер во посовремени конструкции. Студијата покажува дека овој пристап може значително да ја намали големината на базата на кодови и да ја подобри нејзината одржливост. Сепак, студијата исто така ги истакна ограничувањата на автоматското рефакторирање, како што се потенцијалот за воведување грешки и тешкотијата за справување со сложени структури на кодови.

Во последниве години, појавата на генерирање кодови со помош на вештачка интелигенција отвори нови можности за автоматизирање на процесот на пишување код. Значајна студија во оваа област со наслов “Competition-level code generation with AlphaCode“. Студијата го воведува AlphaCode, систем за вештачка интелигенција што покажал конкурентни перформанси на натпревари за програмирање. Ова претставувало значајна пресвртница во полето на генерирање кодови со помош на вештачка интелигенција, демонстрирајќи го потенцијалот на системите за вештачка интелигенција да пишуваат сложен код. Сепак, студијата ги забележува и ограничувањата на AlphaCode, вклучително и неговото потпирање на големи количини на податоци за тренирање и неговата тешкотија во справувањето со покомплексни случаи.

Студијата со наслов „Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models“ прави истражување на корисници за да разбере како програмерите ги користат и перцепираат алатките за генерирање кодови со помош на вештачка интелигенција. Студијата покажува дека иако овие алатки не нужно го подобруваат времето на завршување на задачите или стапката на успех, повеќето учесници претпочитаат да ги користат во нивните секојдневни задачи за програмирање. Ова сугерира дека алатките за генерирање кодови со помош на вештачка интелигенција можат да обезбедат корисна почетна точка за програмерите и да им го намалат времето поминато во пребарување на интернет.

Студијата со наслов “Synchromesh: Reliable code generation from pre-trained language models“ предлага рамка за подобрување на веродостојноста на генерирањето кодови со помош на вештачка интелигенција. Студијата покажува дека нивниот пристап може значително да ја подобри точноста на предвидувањето и да ги спречи грешките во времето на извршување. Сепак, студијата исто така ги истакнува предизвиците за обезбедување на веродостојност на генерирањето кодови, вклучувајќи ја и потребата за робусни механизми за откривање и корекција на грешки.

Досегашните истражувања за генерирање кодови постигнаа значителен напредок во автоматизирањето на процесот на пишување код, од

традиционални методи до техники со помош на вештачка интелигенција. Сепак, остануваат многу предизвици, вклучувајќи ја потребата за робусни механизми за откривање и корекција на грешки, тешкотијата за ракување со сложени структури на кодови и потпирање на големи количини на податоци за тренирање. Идните истражувања во оваа област најверојатно ќе продолжат да ги истражуваат овие предизвици и да развиваат нови методи за подобрување на ефикасноста и доверливоста на генерирањето код.

2.3. Генерирање код со GPT

Полето на генерирање код засновано на GPT е релативно ново и недоволно истражено, со неколку студии кои почнуваат да го расветлуваат неговиот потенцијал и ограничувањата во различни задачи од софтверското инженерство. Овие студии користат различни методологии и се фокусираат на различни аспекти на генерирањето код заснован на GPT, обезбедувајќи вредни сознанија, истовремено нагласувајќи ја потребата за понатамошно истражување.

Студијата “An Empirical Study of Code Smells in Transformer-based Code Generation Techniques (2022)“ истражува што ќе се случи доколку даден модел се тренира со проблематичен код.
Истражувачите го тренирале моделот GPT-NEO 125M со кодови кои во себе содржат одредени пропусти. По завршувањето на процесот на тренирање, тие увиделе дека тренираниот модел во себе ги инкорпорира погрешните практики во генерирањето код, т.е. кодот што го генерира моделот е исто така проблематичен. Ова укажува дека квалитетот на податоците кои се користат за тренирање на овие модели е еден од клучните сегменти за добивање точен код. Сепак, ова истражување е фокусирано само на еден модел и специфични бази на податоци, што може да ја ограничи веродостојноста на наодите.

Трудот “A Preliminary Analysis on the Code Generation Capabilities of GPT-3.5 and Bard AI Models for Java Functions (2023)“ ја оценува способноста на два модели, GPT-3.5 и Bard, за генерирање на Java код при даден опис на функциите што треба да се генерираат. Истражувачите откриле дека GPT-3.5 покажува супериорни перформанси во однос на моделот Bard, генерирајќи точен код за

приближно 90,6% од описите на функциите. Ова сугерира дека GPT-3.5 може да биде моќна алатка за генерирање кодови. Сепак, студијата е ограничена на Java функции и два модели на вештачка интелигенција, што укажува на потребата за пошироко истражување кое вклучува други програмски јазици и модели.

Студијата “Stochastic Code Generation (2023)“ истражува дали употребата на латентен стохастички процес може да ја подобри кохерентноста во генерирањето код. Истражувачите го засноваат својот предложен енкодер и декодер на претходно обучениот модел “CodeParrot“ базиран на GPT-2 и ја користат базата на податоци на “APPS“ за тренирање. Тие откриле дека модифицираниот модел на “Time Control“ функционира слично како “CodeParrot“ на “HumanEval“. Ова сугерира дека употребата на латентен стохастички процес може да биде ветувачки пристап за подобрување на кохерентноста во генерирањето код. Сепак, студијата е ограничена на базата на податоци “APPS“ и моделот “CodeParrot“ базиран на GPT-2, што укажува на потребата за понатамошно истражување што вклучува други бази на податоци и модели.

Додека полето на генерирање код засновано на GPT е сè уште во развој, овие првични студии даваат вредни сознанија за потенцијалот и ограничувањата на оваа технологија во различни задачи од софтверското инженерство. Тие ја нагласуваат важноста на квалитетот на податоците за тренирање, специфичниот модел што се користи и задачата што е на располагање во одредувањето на ефективноста на генерирањето код базирано на GPT. Сепак, со оглед на ограниченият опсег на овие студии, потребни се дополнителни истражувања за целосно да се разберат можностите на генерирањето код засновано на GPT во различни задачи и поставки за софтверско инженерство.

2.4. Празнини во постоечката литература

Додека постоечките студии даваат вредни сознанија за потенцијалот и ограничувањата на генерирањето код заснован на GPT во задачите за софтверско инженерство, сè уште има значителни празнини во тековната литература што треба да се решат. Овие празнини вклучуваат:

- **Ограничена разновидност во програмските јазици и задачи**: Многу студии се фокусираат на специфични програмски јазици и специфични задачи. Постои потреба од истражување кое ги испитува перформансите и ефективноста на генерирањето код засновано на GPT низ поширок опсег на програмски јазици и задачи, вклучително и посложени сценарија од реалниот свет.
- **Недостаток на сеопфатни метрики за евалуација**: Постојните студии првенствено се фокусираат на проценка на синтаксичката исправност на генерираниот код. Иако ова е важен аспект, постои потреба од посеопфатни метрики за евалуација кои ги земаат предвид факторите како што се квалитетот на кодот (читливост, одржување), ефикасност, доверливост и придржување кон стандардите за кодирање.
- **Ограничено разгледување на контекстите за софтверско инженерство од реалниот свет**: Многу студии се фокусираат на евалуација на генерирањето код базирано на GPT изолирано, без да се разгледа неговата интеграција во постоечките работни текови за развој на софтвер. Постои потреба од истражување кое истражува како алатките за генерирање кодови базирани на GPT можат ефективно да се вклучат во контекстите за софтверско инженерство во реалниот свет, земајќи ги предвид факторите како што се динамиката на тимот, управувањето со проекти и соработката.
- **Неразбирање на оптималните случаи и ограничувања**: Тековната литература дава ограничени сознанија за конкретните случаи на употреба каде што генерирањето код засновано на GPT е или не е доволно добро. Постои потреба од истражување кое ќе ги идентификува задачите и контекстите во кои генерирањето код засновано на GPT е најефективно, како и неговите ограничувања и потенцијални предизвици.
- **Ограничено истражување на перспективите на програмерите**: Додека некои студии спроведуваат кориснички студии за да ги разберат перцепциите и преференциите на програмерите во однос на алатките за генерирање кодови со помош на вештачка интелигенција, сè уште има потреба од повеќе истражувања во оваа област. Разбирањето како програмерите ги перцепираат и комуницираат со алатките за генерирање

код базирани на GPT може да обезбеди вредни увиди за развивачите на алатки и организациите кои размислуваат за нивно усвојување.

Со решавање на овие празнини во тековната литература, идните истражувања можат да придонесат за посеопфатно и понијансирано разбирање на ефективноста и ограничувањата на генерирањето код засновано на GPT во задачите на софтверското инженерство. Ова ќе им овозможи на развивачите на софтвер, проект менаџерите и организациите да донесат поинформирани одлуки за усвојување и интегрирање на алатките за генерирање кодови базирани на GPT, што на крајот ќе доведе до напредок во областа на развој на софтвер.

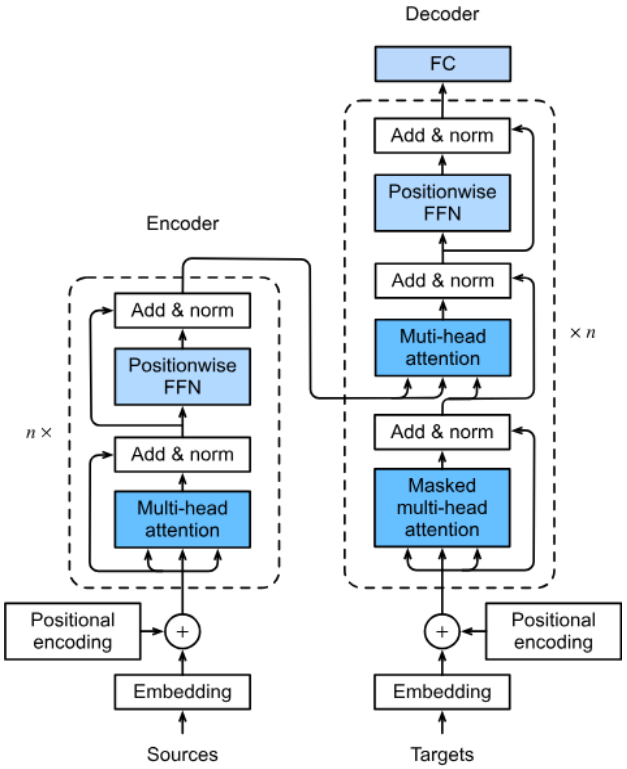
3. ТЕОРЕТСКА РАМКА

3.1. Трансформер модел

Трансформер моделот, за првпат претставен од Vaswani et al. во трудот “Attention is all you need“ во 2017 година, ја формира основата на најмоќните модели за обработка на природен јазик (NLP - Natural Language Processing) што се користат во моментов, вклучувајќи ја и серијата GPT. Ова поглавје ќе понуди сеопфатна дискусија за овој револуционерен модел кој засекогаш го промени полето на вештачката интелигенција.

3.1.1. Архитектура на трансформер моделот

Уникатната архитектура на Трансформерите е специјално дизајнирана да управува со секвенционални податоци, особено фокусирајќи се на подобрување на брзината и ефективноста на машинското преведување. Моделот се состои од енкодер и декодер, од кои и двата сочинуваат stack од идентични слоеви.



Слика 1. Архитектура на трансофермер моделот

Figure 1. Transformer model architecture

Кодерот ги обработува влезните податоци, зафаќајќи ги основните контекстуални врски меѓу елементите во низата. Секој слој од енкодерот има два подслоја: механизам за самовнимание (self-attention) и невронска мрежа за повлекување напред (feed-forward). Влезната низа истовремено се пренесува низ слојот за самовнимание, кој ги моделира зависностите помеѓу зборовите без оглед на нивното растојание во низата. Невронската мрежа за повлекување напред ја применува истата функција на секоја позиција во низата независно.

Декодерот, обратно, ги преведува шифрите во целната излезна секвенца, при што секој слој содржи дополнителен подслој кој врши внимание со повеќе глави (multi-headed attention) над излезот на енкодерот. Оваа структура му овозможува на декодерот да се фокусира на различни делови од излезите на енкодерот за време на генерирањето на секој збор во излезната секвенца.

3.1.2. Механизам на „самовнимание“ кај трансформерите

Централно место во „Трансформер“ моделот е концептот на „внимание“. Механизмот за внимание му овозможува на моделот да ја мери релевантноста на секој влез во низата кога произведува излез. Овој механизам го заменува традиционалното повторување усогласено со секвенца и наместо тоа користи скалирано внимание на точка-производ (dot-product), кое зема query и множество од key-value парови како влезни податоци и пресметува тежинска сума на вредностите врз основа на компатибилноста на барањето со секој клуч.

Моделот “Transformer“ користи техника позната како внимание на повеќе глави (multi-headed attention). Во овој пристап, функцијата за внимание се извршува паралелно преку повеќе линеарни проекции, кои се нарекуваат „глави“. Секоја глава учи различни видови внимание, овозможувајќи му на моделот да се фокусира на различни позиции во низата.

3.1.3. Позиционо енкодирање кај трансформерите

На „Трансформер“ моделите им недостасуваат повторливи или конволуциони слоеви кои вообичаено се користат за обработка на секвенцијални податоци, затоа инхерентно не го разбираат редоследот на влезните елементи. За да се реши ова, моделот вклучува позициони шифрирања, вбризгувајќи информации за релативната или апсолутната позиција на токените во низата. Овие шифри ја имаат истата димензија како и вградувањето, што овозможува нивно сумирање пред да се внесе резултатот во слоевите за самовнимание.

3.1.4. Предности на трансформерите

Една од значајните придобивки на „Трансформер“ моделот е неговата паралелизираност, овозможувајќи побрзо време на обука во споредба со повторливите модели порамнети со секвенца како LSTM (Long Short-Term Memory) или GRU (Gated Recurrent Unit). Со елиминирање на повторувањето, Трансформерот може истовремено да ги обработува сите влезни елементи, што е покомпатибилно со паралелно процесирање на хардвер како што се графичките картички (GPUs).

Дополнително, механизмите за внимание на Трансформерот му овозможуваат поефективно да моделира зависности од долг дострел во податоците. Моделот

може да се фокусира на релевантните делови од низата кога прави предвидувања. Овој пристап е пофлексибилен од моделите кои гледаат само одреден број влезови во исто време, и ги избегнува проблемите што се присутни во другите модели каде важните информации можат да станат премногу разредени или премногу засилени со текот на времето.

3.1.5. Ограничувања на трансформерите

И покрај неговите предности, Трансформер моделот не е без недостатоци. Едно ограничување е користењето меморија, која се скалира квадратно со должината на низата поради механизмот за самовнимание. Ова може да ја усложни обработката на многу долги секвенци.

Друг потенцијален проблем е тоа што Трансформер моделот има тенденција да креира многу параметри, што често доведува до поголем модел. Ова може да резултира со потреба од повисока пресметковна моќ и потреба од повеќе податоци за тренирање за да се спречи преоптоварување (overfitting).

Накратко, Трансформер моделот е еден од клучните пронајдоци во областа на обработката на природни јазици. Неговата уникатна архитектура, карактеризирана со механизмите за внимание и позиционирачките шифри, ја поткрепува ефективноста на пософистицираните модели како GPT. Неговата способност да моделира сложени зависности помеѓу елементите во податоците во низата, заедно со неговиот висок степен на паралелизирање, и покрај неговите ограничувања го прави одличен избор за многу задачи за обработка на природни јазици (NLP).

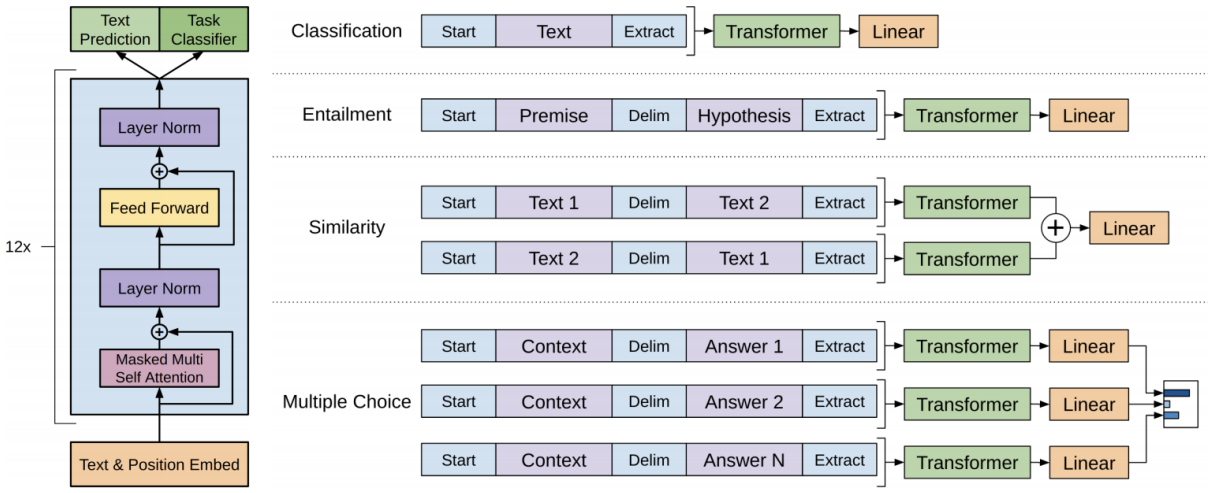
3.2. GPT модели

Воведувањето на „Генеративните претходно-обучени трансформери“ (GPT) предизвика трансформација во областа на обработка на природни јазици (NLP). Овие модели го револуционизираа машинското разбирање и генерирањето на човечки јазици. Во ова поглавје, навлегуваме во длабинското разбирање на архитектурата и работата на GPT моделите.

3.2.1. Архитектура на GPT моделот

Архитектурата на GPT моделите е изградена врз декодерот на Трансформер моделот. Ова им овозможува на овие модели да генерираат текст на начин што е контекстуално чувствителен во однос на влезните податоци. Тие користат механизам за самовнимание, кој ги пресметува оценките за релевантност за секој збор во однос на секој нареден збор во низата, дозволувајќи им на GPT моделите да генерираат излез што ги доловува нијансите на јазикот.

За разлика од целосниот Трансформер модел, кој користи и енкодер и декодер, моделите GPT го користат само декодерот на Трансформерот. Оваа еднонасочна обработка на текстот, од лево кон десно, е она што му ја дава на GPT неговата генеративна моќ.



Слика 2. Архитектура на GPT модел

Figure 1. GPT model architecture

Градбата на моделот GPT вклучува “stack“ од идентични слоеви, при што секој слој содржи два потслоеви: маскиран механизам за само-внимание и целосно поврзана позиционирана мрежа за пренасочување (feed-forward). Забележлива карактеристика на GPT архитектурата е употребата на нормализација на слојот што се применува по слоевите на самовнимание и feed-forward. Понатаму, резидуални врски, слични на оние што се наоѓаат во моделите на ResNet, се присутни околу овие два потслоеви, кои помагаат да се ублажи проблемот со исчезнувањето на градиентите во длабоките мрежи.

3.2.2. Механизам на „самовнимание“ кај GPT

Механизмот за „самовнимание“ (self-attention) е централен ден на GPT моделите. Како што претходно споменавме, самовниманието ги пресметува оценките за релевантност за секој збор во низата во однос на секој нареден збор. Сепак, GPT користи маскирано самовнимание, што го ограничува фокусот на моделот само на левиот контекст или на претходните зборови. Ова обезбедува сигурност дека предвидувањето на моделот за дадена позиција 'i' во низа се потпира само на зборовите што веќе се појавиле пред позицијата 'i'. Така, моделот генерира нови зборови врз основа на неговото знаење за претходните зборови во низата, без никаква предвидливост за тоа што следува потоа.

3.2.3. Начин на функционирање на GPT моделите

GPT моделите се обучуваат користејќи учење без надзор врз база на голем корпус на текстуални податоци. За време на тренирањето, моделот учи да го предвидува следниот збор во низа врз основа на зборовите што се пред него во таа низа, оптимизирајќи ги неговите параметри за да ја минимизира разликата помеѓу неговите предвидувања и вистинските зборови што следат.

Оваа генеративна способност им овозможува на овие модели да генерираат текст кој е многу сличен на текст напишан од човек. Со оглед на основната реченица или збор, моделот може да генерира следна серија зборови или целосни реченици кои се контекстуално и граматички кохерентни.

Исто така, вреди да се напомене дека GPT моделите користат техника наречена „пребарување со зрак“ (beam search) за време на генерирањето текст. Наместо само да се задржи единственото најдобро предвидување во секој чекор, пребарувањето со зрак ги проширува сите можни следни чекори и ги задржува најдобрите „k“ предвидувања на секој чекор, каде што „k“ е параметар дефиниран од корисникот.

Архитектурните и оперативните детали на GPT моделите ги објаснуваат нивните импресивни способности во задачи како што се превод, одговарање на прашања, генерирање текст и генерирање код. Континуираното усовршување

на овие модели во текот на последователните верзии, секоја зголемувајќи ја и оптимизирајќи ја архитектурата, дополнително ги подобри овие способности.

3.2.4. Еволуција на GPT моделите

Првиот GPT модел, иако претставуваше значителен напредок во моделирањето на јазикот, беше ограничен со неговата големина од 117 милиони параметри. Неговата примарна задача беше да го предвиди следниот збор во низа зборови, ограничување што беше проширено во следните верзии на серијата GPT.

Објавувањето на GPT-2 означи значаен чекор напред, со скалирање до 1,5 милијарди параметри. Оваа дополнителна сложеност му овозможи на GPT-2 да покаже подобро разбирање на контекстот и да генерира текст кој е повеќе сличен на текст напишан од човекот.

GPT-3, наследникот на GPT-2, направи уште еден квантен скок во јазичното моделирање. Со неверојатни 175 милијарди параметри, GPT-3 донесе можности како што се превод, одговарање прашања и основни задачи за кодирање. Овој модел, исто така, воведе подобрувања како учење со контекст, овозможувајќи му да ги дотерува своите одговори врз основа на специфичен контекст на разговор.

Следејќи го патот на својот претходник, GPT-4 се заснова на истите работни и архитектурни принципи на Трансформер моделот, но во уште поголем обем. GPT-4 го следи трендот на сè поголеми модели базирани на Трансформери со потенцијално стотици милијарди до трилиони параметри.

3.2.5. Примена на GPT за задачи од софтверското инженерство

И покрај тоа што првенствено беше развиен како јазичен модел, способностите на GPT се големи и тој наоѓа примена во најразлични сфери, од кои едната е во областа на софтверското инженерство. Најчесто се користи за задачи како што се комплетирање код, откривање грешки, па дури и генерирање код врз база на даден опис. Обемот и ефикасноста на овие задачи зависат од верзијата на

користениот модел GPT, при што подоцнежните верзии генерално обезбедуваат подобри резултати поради нивната зголемена сложеност и разбирање на јазикот.

Во продолжение ќе ги разгледаме најчестите сфери на примена на GPT во софтверското инженерство.

- **Комплетирање на код:** Една од најистакнатите употреби на GPT моделите во софтверското инженерство е комплетирањето на кодот. Со оглед на контекстот на постојниот код, GPT моделите можат да ги предвидат следните неколку редови или да завршат блок од код, слично како што го предвидуваат следниот збор во реченицата. Оваа апликација се покажа како вредна алатка за програмерите со тоа што помага во пишувањето код побрзо и попрецизно и обезбедува намалување на когнитивното оптоварување вклучено во помнењето на синтаксата и функциите во даден програмски јазик. Пополнувањето на кодот, исто така, помага да се минимизираат грешките во кодирањето со предлагање на правилни конструкции на код.
- **Откривање грешки:** Моделите на GPT, исто така, може да се обучуваат да откриваат потенцијални грешки или проблеми во базата на кодови. Со учење од голема база на кодови и познати грешки, моделите можат да предвидат потенцијални проблеми во новиот код, помагајќи да се спречи проблематичниот код пред да се продуцира. Тие можат да идентификуваат синтаксни грешки, логички грешки, па дури и некои сложени проблеми како што се race conditions, а со тоа служат како прелиминарна алатка за автоматизирано прегледување на кодот.
- **Генерирање код врз база на даден опис:** Друга корисна примена на GPT моделите е генерирање код од дадени описи напишани на природен јазик. Програмерите можат да внесат опис на функционалноста што ја посакуваат, а моделот може да издаде фрагмент од код што го исполнува тој опис. Ова може да биде особено корисно за помалку искусни програмери или оние кои работат во програмски јазици кои не им се многу познати. Може да послужи и како почетна точка за покомплексна функционалност, забрзувајќи го процесот на развој.
- **Генерирање и преглед на документација:** Креирањето и прегледувањето на документацијата е клучна задача во развојот на

софтвер, но често одзема многу време. GPT моделите може да се користат за автоматизирање на делови од овој процес со генерирање на документација врз основа на коментари на кодот или дури и директно од самиот код. Покрај тоа, овие модели може да се користат за преглед на постоечката документација, идентификување области кои се застарени или кои не се добро објаснети.

- **Анализа на барања:** GPT моделите, исто така, можат да помогнат во процесот на анализа на специфични барања. Со тренирање врз разновиден опсег на документи кои содржат барања за софтвер и кориснички приказни, моделите можат да помогнат во идентификувањето на нејаснотии или недоследности во новите барања. Тие, исто така, можат да предложат подобрувања во фразата на барањата за тие да бидат појасни и попроверливи.

Можеме да заклучиме дека способноста на GPT моделите да разберат и генерираат текст сличен на човекот отвора широк опсег на апликации во областа на софтверското инженерство. Од помагање на програмерите при пишување и прегледување кодови, па сè до автоматизирање на документацијата и помош во процесот на анализа на барањата. Како што овие модели продолжуваат да се подобруваат, се очекува дека нивната употребливост во софтверското инженерство исто така ќе продолжи да расте.

3.2.6. Fine-tuning на GPT за генерирање на код

Иако GPT моделите имаат општа способност да генерираат текст сличен на човекот и покажуваат солидна способност за ракување со кодот, честопати е неопходно фино прилагодување (fine-tuning) за поспецифични задачи како што е генерирањето код. Фино подесување е процес на продолжување на процесот на тренирање на претходно обучен модел (како GPT) за нова, специфична задача. Во овој дел ќе зборуваме за тоа како GPT моделите може да се дотеруваат за задачи за генерирање на код.

- **Собирање специфични податоци за тренирање:** Првиот чекор во дотерувањето на GPT моделите за генерирање код вклучува собирање голем и разновиден сет на примери на кодови кои можат да послужат како податоци за обука. Овие примери треба да опфатат широк опсег на

програмски задачи и идеално треба да го претставуваат типот на код што ќе се очекува да го генерира моделот. Ова може да вклучува собирање open-source код од јавни репозиториуми, учебници за програмирање или други ресурси. Податоците за обука треба да вклучуваат и различни програмски јазици доколку целта е да се создаде модел за генерирање јазично-агностички кодови.

- **Претпроцесирање податоци:** Откако ќе се соберат податоците за тренирање, тие треба претходно да се обработат во соодветен формат за моделот. На пример, кодот можеби ќе треба да се токенизира, со посебни токени за различни типови на синтакса (како променливи, функции и оператори). Коментарите и празното место можеби ќе треба да се напишани на специфичен начин, во зависност од тоа како сакате моделот да ги третира.
- **Тренирање и fine-tuning на моделот:** Откако податоците за тренирање ќе бидат подготвени, следниот чекор е да се искористат во процесот на fine-tuning. Ова вклучува сетирање на моделот за продолжување на процесот на тренирање врз основа на новите податоци. Целта е да се минимизира разликата помеѓу излезот на моделот и вистинскиот код во податоците за обуката. Обуката продолжува итеративно, ажурирајќи ги внатрешните параметри на моделот сè додека тој може да генерира код кој тесно одговара на стилот и структурата на податоците за тренирање. За време на овој процес, важно е внимателно да се управува со стапката на тренирање (learning rate) за да се спречи преоптоварување и да се осигура дека моделот прави добро генерализирање нови, невидени податоци.
- **Евалуација и тестирање:** По тренирањето, моделот треба да се оцени за да се осигура дека правилно го генерира кодот. Ова може да вклучува квалитативна анализа, каде што генерираниот код се прегледува и споредува со кодот напишан од човекот, или квантитативна анализа, каде што излезот на моделот се споредува со тест сет на невиден код. Исто така, може да биде корисно да се креираат тест случаи за да се провери дали моделот може да се справи со одредени специфични и сложени задачи за кодирање.

- **Постојано учење:** Со оглед на динамичната природа на развојот на софтвер, корисно е моделот да продолжи да учи и да се прилагодува со текот на времето. Ова може да вклучи периодично преквалификација на моделот за нови податоци или имплементирање на систем каде што моделот може да учи од своите грешки и да се подобрува со текот на времето.

Следејќи ги овие чекори за fine-tuning, GPT моделите можат значително да ја подобрат ефикасноста во извршување на задачите за генерирање код. Меѓутоа, важно е да се забележи дека иако процесот на fine-tuning може значително да ги подобри перформансите на моделот на одредени задачи, квалитетот на излезот на крајот ќе зависи од квалитетот и разновидноста на податоците користени за негово тренирање, како и од соодветноста на самиот процес на fine-tuning.

3.3. Влијание на кодот генериран од GPT врз концептите на софтверското инженерство

Неколку фундаментални концепти во софтверското инженерство, како што се одржливост, читливост и модуларност, може да бидат значително под влијание на воведувањето на генерирање код базирано на GPT. Разбирањето на овие концепти и нивната врска со генерирањето код може да обезбеди корисни сознанија за потенцијалните придобивки и предизвици од користењето на вештачката интелигенција во развојот на софтвер.

3.3.1. Влијание на GPT врз одржливоста на кодот

Одржливоста се однесува на леснотијата со која софтверскиот систем може да се ажурира или модифицира. Ова вклучува корекција на дефекти, подобрување на перформансите или прилагодување на системот на променливите барања или средини. Високата одржливост е пожелна карактеристика бидејќи го намалува времето, трошоците и ризикот од правење промени во софтверот.

Генерирањето код засновано на GPT може потенцијално да ја подобри одржливоста со автоматизирање на рутинските задачи за кодирање, намалувајќи ја можноста за човечка грешка. Сепак, тој исто така претставува

ризик ако генерираниот код е сложен или не е добро структуриран, бидејќи може да го отежне кодот за разбирање и менување. Според тоа, квалитетот на податоците за тренирање и процесот на дотерување е од клучно значење во одредувањето на влијанието врз одржливоста.

3.3.2. Влијание на GPT врз читливоста на кодот

Читливост е леснотијата со која развивачот може да разбере дел од кодот. Добрата читливост го олеснува прегледувањето, отстранувањето грешки и менувањето на кодот, што придонесува за подобрена одржливост и продуктивност.

Влијанието на генерирањето код засновано на GPT врз читливоста во голема мера зависи од тоа колку добро моделот е обучен и прилагоден. Ако моделот е добро обучен, може да генерира код кој е подеднакво читлив, ако не и повеќе, отколку кодот напишан од човекот. Меѓутоа, ако не, генерираниот код би можел потешко да се разбере, особено ако вклучува неконвенционални или непотребно сложени структури на кодови. Затоа, за да се обезбеди читливост потребна е внимателна обука и евалуација на GPT моделите.

3.3.3. Влијание на GPT врз модуларноста на кодот

Модуларноста се однесува на степенот до кој компонентите на системот можат да се одвојат и рекомбинираат. Високото ниво на модуларност овозможува полесно разбирање, развој, тестирање и одржување на софтверот, бидејќи промените на еден модул не влијаат на другите.

Генерирањето код засновано на GPT може да влијае на модуларноста на различни начини. На пример, ако моделот е обучен за добро структуриран, модуларен код, тој потенцијално би можел да генерира код кој се придржува до овие принципи. Сепак, постои и ризик моделот да генерира код кој е многу меѓузависен и нема јасни граници на модулите, особено ако не му се даде доволен контекст. Ова ја нагласува важноста за обезбедување на моделот со висококвалитетни, добро структурирани податоци за обука.

Генерирањето код засновано на GPT може значително да влијае на клучните концепти за софтверско инженерство како што се одржливост, читливост и модуларност. Внимателно разгледување на овие фактори е неопходно кога се интегрираат таквите модели во процесот на развој на софтвер. Исто така, ја нагласува важноста на добрите практики за тренирање за да се осигураме дека генерираниот код ги поддржува овие клучни принципи во софтверско инженерство.

4. ИСТРАЖУВАЧКИ ПРАШАЊА И ХИПОТЕЗИ

Во нашата потрага да ја истражиме ефективноста на генерирањето код засновано на GPT за задачи за софтверско инженерство, нашиот фокус го насочивме на пет истражувачки прашања кои се императив за решавање во ова истражување. Секое истражувачко прашање е придружено со поврзана хипотеза, која дава основа за нашето емпириско истражување и ни овозможува да ја процениме ефективноста и импликациите на генерирањето код заснован на GPT.

- **Истражувачко прашање 1 (Q1):** Каква е ефикасноста и точноста на генерирањето код засновано на GPT?
- **Хипотеза 1 (H1):** Генерирањето код засновано на GPT ја подобрува ефикасноста и точноста на задачите за софтверско инженерство во споредба со традиционалните методи. Оваа хипотеза е изградена врз способностите на алгоритмите за машинско учење на GPT да учат од огромни количини на податоци и да реплицираат задачи побрзо и попрецизно од традиционалното програмирање.
- **Истражувачко прашање 2 (Q2)**: Кои фактори влијаат на ефикасноста на генерирањето код засновано на GPT за различни задачи од софтверското инженерство?
- **Хипотеза 2 (H2):** Ефикасноста на генерирањето код засновано на GPT варира во зависност од сложеноста и природата на задачите во софтверско инженерство, како и јасноста и разбирливоста на поставената задача. Ова значи дека задачите со помала сложеност и оние кои се појасно и попрецизно формулирани ќе резултираат со поефикасно генериран код.

- **Истражувачко прашање 3 (Q3):** Во кои типови на задачи од софтверското инженерство генерирањето код со GPT е најефикасно?
- **Хипотеза 3 (H3):** Генерирањето код засновано на GPT е особено ефикасно за задачи како што се откривање грешки, рефакторирање код и автоматско тестирање. Овие задачи често вклучуваат обрасци и елементи што се повторуваат, со кои моделите за машинско учење можат да научат да се справуваат ефикасно.
- **Истражувачко прашање 4 (Q4):** Кои се ограничувањата и потенцијалните ризици од генерирањето код заснован на GPT во софтверското инженерство?
- **Хипотеза 4 (H4):** И покрај неговите придобивки, генерирањето код засновано на GPT воведува нови потенцијални ризици во софтверското инженерство, вклучувајќи намалена интерпретабилност на кодот и зголемено потпирање на моделите за машинско учење. Оваа хипотеза ги предвидува инхерентните ограничувања и новите предизвици кои можат да се појават со употребата на ВИ во сложено поле како софтверското инженерство.
- **Истражувачко прашање 5 (Q5):** Како може генерирањето код базирано на GPT да влијае на улогата на софтверските инженери и процесот на развој на софтвер?
- **Хипотеза 5 (H5):** Генерирањето код засновано на GPT ќе ја зголеми важноста на улогата на софтверските инженери наместо да ги замени, префрлајќи го фокусот кон надзор, контрола на квалитетот и обука на овие системи за вештачка интелигенција. Оваа хипотеза ја потврдува зголемената улога на ВИ во различни сектори, со очекување на еволуција во работните улоги наместо целосна замена.

Во следните поглавја, имаме за цел емпириски да ги тестираме овие хипотези, расветлувајќи ги можностите, потенцијалот за оптимизација, применливите задачи, ограничувањата и импликациите на генерирањето код базирано на GPT во областа на софтверското инженерство.

5. МЕТОДОЛОГИЈА

Ефикасноста на генерирањето код ќе ја истражиме кај двата најдобри модели базирани на GPT: GPT-3.5 и GPT-4.

Овие модели беа избрани поради нивните врвни способности и широкото усвојување во областа, што ги прави претставници на тековната најсовремена технологија во генерирањето кодови со вештачка интелигенција.

За целите на ова истражување избравме десет задачи, при што секоја задача има три нивоа на тежина и тоа: лесно, средно и напредно.

Во секое ниво има постепено зголемување на сложеноста и барањата, со што се овозможува погрануларна анализа на можностите на GPT моделите. Задачите се избрани да опфатат широк опсег на вообичаени активности од софтверско инженерство, во кои спаѓаат: backend и frontend развој, манипулација со податоци, креирање на алгоритми итн.

Задачите им беа поставувани на моделите на англиски јазик и во нивниот опис намерно не се спомнуваат специфични програмски јазици бидејќи сметаме дека е подобро моделите сами да одлучат кој програмски јазик ќе го користат за решавање на специфична задача. Ова ни овозможува да направиме и анализа за тоа кои програмски јазици ги преферираат GPT моделите.

На крајот, ќе спроведеме и истражување на мислењето на софтверските инженери со што ќе се испитаат нивните ставови во однос на употребата на GPT моделите во олеснување на процесот на кодирање.

5.1. Задачи за генерирање на код

Во продолжение се дадени задачите кои им беа поставени на моделите:

5.1.1. Задача 1 (T1): Креирање на веб-страница

- **Лесно ниво:**
    - Креирај едноставна страница.
    - Страницата треба да вклучува header, main content и footer.
    - Во header делот треба да биде насловот на веб-страницата.
    - Main Content делот вклучува воведен параграф.
    - Footer делот вклучува информации за авторските права.

    - Страницата треба да има едноставен стил – специфична боја на позадина и custom боја на текст и фонт.
- **Средно ниво:**
    - Креирај посложена страница.
    - Страницата треба да вклучува header, main section и footer.
    - Header делот треба да вклучува мени за навигација со линкови до секциите “#About“, “#Services“ и “#Contact“.
    - Main Content делот треба да вклучува порака за добредојде и краток опис на услугите.
    - Footer делот треба да вклучува информации за контакт.
    - Страницата треба да вклучува понапреден професионален и модерен стил со сенки, транзиции и анимации.
- **Напредно ниво:**
    - Конструирај динамична страница за е-трговија.
    - Страницата треба да биде респонзивна и вклучува напреден професионален и модерен стил со сенки, транзиции и анимации.
    - Веб-страницата треба да вклучува product listing page, product detail page и shopping cart page.
    - Product Listing страницата треба да прикажува grid на производи, со pagination контроли.
    - Product Detail страницата треба да прикажува детални информации за производот кога ќе се кликне на него.
    - Shopping Cart страницата треба да ги прикажува производите додадени во кошничка и вкупната цена.
    - Веб-страницата треба да вклучува напредни функции, како што се сортирање и филтрирање на страницата со список на производи и можност за додавање и отстранување производи од кошничката.
    - Имплементирај state management за справување со функционалностите на кошничката.

5.1.2. Задача 2 (T2): Креирање login форма

- **Лесно ниво:**
    - Креирај едноставна login форма.

- Формата треба да има две полиња: едно за корисничко име и едно за лозинка.
- Формата треба да има submit копче.
- По притискање на submit, треба да се прикаже порака “Form Submitted“.

- **Средно ниво:**
  - Креирај login форма со две полиња: корисничко име и лозинка.
  - Формата треба да вклучува проверки за валидација:
  - Двете полиња треба да бидат задолжителни и треба да се прикаже порака доколку корисникот се обиде да направи submit без да ги пополни полињата.
  - Полето за лозинка треба да бара минимален број знаци.
  - Порака за грешка треба да се прикаже веднаш до полето во кое има грешки, со јасен опис за грешката.
  - По успешното поднесување, треба да се прикаже порака во која пишува “Form Submitted Successfully“.
- **Напредно ниво:**
  - Креирај сложена login форма.
  - Формата треба да вклучува полиња за корисничко име и лозинка, како и дополнителни полиња како што се е-пошта, телефонски број и two-factor authentication код.
  - Сите полиња треба да имаат проверки за валидација.
  - Полето за е-пошта вклучува валидација за соодветен формат за е-пошта.
  - Полето за телефонски број треба да се потврди за да се осигура дека содржи само цифри и е со правилна должина за телефонски број.
  - Two-factor authentication кодот треба да биде точно 6 цифри.
  - Формуларот треба да обезбеди визуелна повратна информација кога полето ќе ја помине валидацијата (на пр., зелена боја на границата на input полето).
  - Треба да има функција “show password“ која им овозможува на корисниците да ја видат нивната внесена лозинка.

- Податоците од формата треба да се зачуваат во local storage по поднесувањето.
- По успешното поднесување, треба да се прикаже порака во која пишува “Form Submitted Successfully“.

5.1.3. Задача 3 (T3): Креирање калкулатор апликација

- **Лесно ниво:**
  - Креирај едноставна калкулатор апликација.
  - Овој калкулатор треба да може да врши основни аритметички операции: собирање, одземање, множење и делење.
  - Треба да има нумерички влеза и избрана операција од корисникот.
  - Апликацијата треба да го врати резултатот од операцијата кога корисникот ќе ја иницира.
- **Средно ниво:**
  - Креирај калкулатор апликација.
  - Овој калкулатор треба да може да врши основни аритметички операции: собирање, одземање, множење и делење.
  - Покрај тоа, треба да има функции за операции како што се квадрат, квадратен корен и можност за користење загради за приоритетни операции.
  - Калкулаторот треба да има интерфејс кој им овозможува на корисниците да внесуваат броеви и операции еден по еден, слично на стандардниот дигитален калкулатор.
- **Напредно ниво:**
  - Креирај комплексен scientific калкулатор.
  - Калкулаторот треба да ги извршува сите основни аритметички операции, како и напредни операции, вклучувајќи, но не ограничувајќи се на: експоненти, квадратен корен, корен од коцка, тригонометриски функции (sin, cos, tan), инверзни тригонометриски функции, логаритми, факториели и константи како „Пи“ и „e“.
  - Апликацијата треба да поддржува и загради за приоритет на операциите и треба да го следи редоследот на операциите (PEMDAS/BODMAS).

- Калкулаторот треба да има интерфејс кој им овозможува на корисниците да внесуваат сложени изрази одеднаш или чекор по чекор.
- Исто така, треба да има мемориски функции за зачувување и повлекување вредности, како и опција за чистење на меморијата.

5.1.4. Задача 4 (T4): Креирање REST API

- **Лесно ниво:**
  - Креирај едноставно REST API со еден endpoint.
  - Овој endpoint треба да одговара на GET барања и да враќа статичен JSON објект.
  - Објектот JSON треба да има неколку key-value парови
- **Средно ниво:**
  - Креирај REST API за едноставни ресурси (како корисници или производи).
  - API-то треба да има endpoint-и за сите стандардни CRUD операции (CREATE, READ, UPDATE, DELETE).
  - Треба да ги користи точните HTTP методи за секој тип на операција: POST за креирање, GET за читање, PUT или PATCH за ажурирање, DELETE за бришење.
  - Секој endpoint треба да врати JSON објект или низа од објекти.
- **Напредно ниво:**
  - Креирај REST API за сложен систем за блогови со повеќе поврзани ресурси.
  - Во системот за блогови, може да има ресурси за корисници, постови, коментари и категории.
  - Секој ресурс треба да има endpoint-и за сите стандардни CRUD операции.
  - API-то треба да поддржува и понапредни функции, како што се филтрирање и сортирање на вратените податоци, пагинирање и ракување со вгнездени ресурси (како добивање на сите коментари за одредена објава).
  - Треба да користи статусни кодови за да го означи успехот или неуспехот на барањето и да дава корисни пораки за грешки

- API-то треба да ги следи најдобрите практики за дизајн на REST API, како што се stateless server дизајн, користење именки во URI-то и правилната употреба на методите на HTTP и статусните кодови.

5.1.5. Задача 5 (T5): Пишување SQL наредби

- **Лесно ниво:**
  - Напиши SQL наредба која враќа податоци од табела со име “employees“.
  - Табелата “employees“ има колони: id, name, job_title и department и salary.
  - Наредбата треба да ги врати имињата и работните позиции на вработените кои работат во секторот “Marketing“ и имаат плата поголема од 50000.
- **Средно ниво:**
  - Напишете SQL наредба за преземање податоци од две поврзани табели: “orders“ и “customers“.
  - Табелата “orders“ има колони: order_id, customer_id, product_name и order_date.
  - Табелата “customers“ има колони: customer_id, first_name, last_name, и e-mail.
  - Барањето треба да ги врати сите нарачки заедно со името, презимето и е-поштата на соодветниот customer.
- **Напредно ниво:**
  - Напишете SQL наредба која вклучува повеќе табели и посложени операции.
  - Дадени се табелите се “students", “courses", и “student_courses".
  - Табелата “students" има колони: student_id, first_name, last_name, и email.
  - Табелата “courses" има колони: course_id, course_name, и instructor.
  - Табелата “student_courses" има колони: student_id, course_id, и grade.

- Наредбата треба да ги врати името, презимето на секој студент, имињата на сите предмети на кои се запишани и соодветните оценки. Дополнително, треба да вклучува само студенти кои се запишани на повеќе од еден курс, а резултатите треба да се подредат според презимето на студентот.

5.1.6. Задача 6 (T6): Пишување рекурзивни функции

- **Лесно ниво:**
  - Напиши рекурзивна функција која го враќа збирот на вредностите од дадена низа.
  - Функцијата треба да земе низа од броеви како влез и да ја врати нивната сума.
  - Треба да се користи рекурзија за да се постигне ова, а не едноставна јамка или вградена функција за сума (ако постои во избраниот јазик).
- **Средно ниво:**
  - Напиши рекурзивна функција која користи “deep search“ на вгнезден објект или речник.
  - Функцијата треба да земе вгнезден објект и клуч како влез.
  - Функцијата треба да ја врати вредноста поврзана со дадениот клуч доколку постои некаде во вгнездениот објект, во спротивно треба да врати нула или еквивалент.
  - Функцијата треба да користи рекурзија за да го помине вгнездениот објект.
- **Напредно ниво:**
  - Напиши рекурзивна функција која проверува дали е можно да се избере група цели броеви од дадена низа така што збирот на групата е еднаков на даден број како целна сума.
  - Функцијата треба да зема три влеза: почетен индекс за тоа каде да започне проверката во низата, низа од цели броеви и целна сума.
  - Сите множители на 5 во низата мора да бидат вклучени во групата. Ако вредноста веднаш по множителот од 5 е 1, таа не треба да се избира.

- Функцијата треба да врати true или false што означува дали е можно или не да се постигне целната сума под овие ограничувања.
- На пример, ако имаме почетен индекс 0, низата [2, 5, 10, 4] и целната сума 19, функцијата треба да врати true. Со истата низа и почетен индекс, но целната сума 12, функцијата треба да врати false.

5.1.7. Задача 7 (T7): Манипулација со стрингови

- **Лесно ниво:**
  - Напиши функција за броење на појавувањата на секој збор во дадена низа.
  - Функцијата треба да земе низа од зборови како влез и да врати речник или слична структура на податоци каде што клучеви се уникатните зборови, а вредности се бројот на појавувања на секој од зборовите
  - На пример, ако е дадена низата “apple banana apple strawberry banana apple“, функцијата треба да врати речник {"apple": 3, "banana“: 2, "strawberry“: 1}.
- **Средно ниво:**
  - Напиши функција која зема низа од зборови (одделени со празни места) и ја претвора во низа каде што секој збор се заменува со збор во Морзеова репрезентација. Претставувањето на Морзеовата шифра на секоја буква треба да биде одвоено со празно место, а секој збор треба да биде одвоен со коса црта ('/').
  - Функцијата треба да ги опфати сите букви од A до Z (нечувствителни на букви) и цифри од 0 до 9.
  - На пример, ако е дадена низата “Hello World 123“, функцијата треба да се врати „.... . .-.. .-.. --- / .-- --- .-. .-.. -. . / .---- ..--- ...--“.
  - Функцијата треба да управува со невалиден влез, како што се знаците што немаат соодветна репрезентација со Морзеова шифра.
- **Напредно ниво:**

- Напишете функција која треба да направи промена во даден стринг така што секое самостојно појавување на зборот “is“ е заменето со “is not“
- Зборот “is“ треба да се смета за самостоен ако не му претходи или следи буква. На пример, “is“ во зборот “this“ не би се сметало за самостојно појавување.
- Функцијата треба да земе низа од зборови како влез и да врати изменета низа која се придржува до гореспоменатите услови.
- На пример, ако влезната низа е “is test“, функцијата треба да врати “is not test“. Или ако иаме “is-is“, треба да се врати “is not-is not“. Ако е дадено “This is right“, функцијата треба да врати “This is not right“.

5.1.8. Задача 8 (T8): Манипулација со низи и матрици

- **Лесно ниво:**
  - Напиши функција за пресметување на збирот на сите елементи во дадена 2D матрица.
  - Функцијата треба да прима 2D матрица (низа од низи) како влез и да го врати збирот на сите елементи.
  - На пример, ако е дадена матрицата [[1, 2, 3], [4, 5, 6], [7, 8, 9]], функцијата треба да врати 45.
- **Средно ниво:**
  - Напишете функција за ротирање дадена 2D матрица за 90 степени во насока на стрелките на часовникот.
  - Функцијата треба да земе 2D матрица (низа од низи) како влез и да врати нова матрица која е влезната матрица ротирана за 90 степени во насока на стрелките на часовникот.
  - На пример, ако е дадена матрицата [[1, 2, 3], [4, 5, 6], [7, 8, 9]], функцијата треба да врати [[7, 4, 1], [8, 5, 2], [9, 6, 3]].
  - Функцијата треба да може се справи со квадратни и правоаголни матрици.
- **Напредно ниво:**
  - Напиши функција која ја реорганизира дадената влезна низа така што по секој број 4 во низата веднаш ќе следува бројот 5.

- Функцијата треба да земе низа од цели броеви како влез и да ја врати изменетата низа.
- Положбите на 4-те во низата не смеат да се менуваат, а сите други броеви може да се поместуваат. Влезната низа е загарантирано да има еднаков број на 4-ки и 5-ки, а секој број 4 има најмалку еден број што не е 4 што следи по него.
- Бројот 5 може да се појави било каде во оригиналната низа
- На пример, ако влезната низа е [5, 4, 9, 4, 9, 5], функцијата треба да врати [9, 4, 5, 4, 5, 9]. Ако е [1, 4, 1, 5], треба да врати [1, 4, 5, 1]. Ако е дадена [1, 4, 1, 5, 5, 4, 1], функцијата треба да врати [1, 4, 5, 1, 1, 4, 5].

5.1.9. Задача 9 (T9): Манипулација со JSON објекти

- **Лесно ниво:**
  - Напиши функција која обработува даден JSON објект
  - JSON објектот претставува низа од “student“ објекти
  - Секој објект student има полиња: 'name', 'age', и 'grade'.
  - Функцијата треба да врати JSON објект со имињата на сите студенти
- **Средно ниво:**
  - Напиши функција која обработува даден JSON објект
  - JSON објектот претставува низа од “student“ објекти
  - Секој објект student има полиња: 'name', 'age', 'grade' и 'subjects' (предмети кои ги изучува студентот)
  - Функцијата треба да врати JSON објект кој ги содржи само они студенти кои го слушаат предметот 'Science'
  - JSON објектот треба да вклучува и ново поле 'average_grade' кое ја содржи просечната оценка од сите предмети на студентот
- **Напредно ниво:**
  - Напиши функција која прима два JSON објекти како влез.
  - Првиот објект претставува низа од 'student' објекти со полиња: 'student_id', 'name', 'age', и 'grade'.
  - Вториот објект претставува низа од 'test_score' објекти, секој со 'student_id', 'subject', 'test_date', и 'score' полиња

- Функцијата треба да ги спои информациите врз основа на 'student_id' и да врати нов JSON објект.
- Во нивиот JSON објект, секој student објект треба да има дополнително поле 'performance', кое е низа од нивните резултати од тестови (секој елемент во низата треба да содржи полиња: 'subject', 'test_date', и 'score').
- Функцијата исто така треба да ја пресмета вкупната просечна оценка на секој студент по сите предмети и да го вклучи како ново поле 'average_score'

5.1.10. Задача 10 (T10): Цртање на објекти со HTML5 Canvas

- **Лесно ниво:**
  - Напиши функција за цртање на основна геометриска форма со HTML5 Canvas
  - Функцијата треба да го земе 2Д контекстот, името, позицијата и големината на геометриското тело
- **Средно ниво:**
  - Напиши функција која црта аналоген часовник што го покажува времето со HTML5 Canvas
  - Функцијата треба да го земе 2Д контекстот на canvas-от и димензиите на часовникот.
  - Часовникот треба да вклучува часовна стрелка, минута и секундарна стрелка.
- **Напредно ниво:**
  - Напишете функција која ќе нацрта илустрација на „Пикачу“ со HTML5 Canvas. „Пикачу“ е добро познат лик од популарната франшиза „Покемон“ и функцијата треба да ги доловува неговите уникатни карактеристики.
  - Функцијата треба да ги земе како параметри 2Д контекстот на canvas-от, како и положбата и големината на илустрацијата.
  - Цртежот треба да го содржи целокупниот облик на „Пикачу“, неговите уши (со црни врвови), очите, образите (црвени кругови), устата, носот и опашката. Обидете се да изгледа што е можно пореалистично.

5.2. Истражување за мислењата на софтверските инженери

За да добиеме дополнителни сознанија за практичната примена, предностите, ограничувањата и потенцијалните области за подобрување на GPT алатките за генерирање код, дизајниравме и администриравме анкета составена од 15 прашања насочена кон активните софтверски инженери.

Анкетата ги доловува реалните искуства и погледите на оние кои директно се вклучени во процесот на развој на софтвер и притоа се имаат сретнато со овие алатки. Ова нуди богата искуствена перспектива која работи во тандем со тестирањето на нашите експериментални GPT модели.

Прашањата се составени да опфатат повеќе аспекти за користењето на алатките и тоа:

- **Позадина и употреба:** Со ова ќе се соберат податоци за улогите на испитаниците во развојот на софтвер, долгогодишното искуство, умешните програмски јазици и фреквенцијата и целта на користење на алатките за вештачка интелигенција. Овие прашања го даваат потребниот контекст за наредните одговори.
- **Предности, ефикасност и оценка за квалитетот на кодот:** Со ова добиваме вредни сознанија за мислењата во врска со точноста на генерираниот код, стапката на успех при првиот обид и влијанието на овие алатки врз брзината на кодирање и намалување на грешките. Ова ни дава увид во придобивките што ги перцепираат овие ИТ професионалци.
- **Слаби страни и ограничувања:** Овде, ги собираме перспективите на софтверските инженери за главните недостатоци или предизвици за користење алатки за вештачка интелигенција базирани на GPT. Ова ни овозможува да ги разбереме нивните недостатоци од гледна точка на реалниот корисник.
- **Идни перспективи и подобрувања:** Последните прашања се фокусирани на погледите за идната улога на алатките во развојот на софтвер, можните подобрувања и подготвеноста на софтверските инженери да ги препорачаат овие алатки на други развивачи на софтвер.

Податоците од анкетата ќе бидат подложени на квантитативни и квалитативни анализи. Квантитативните податоци, како што се фреквенцијата на употреба и оцената на точноста, ќе бидат статистички обработени за да се откријат обрасците, трендовите и значајните наоди. Квалитативните податоци извлечени од прашања со повеќекратен избор и прашања од отворен тип ќе бидат подложени на тематска анализа за да се идентификуваат клучните теми и сознанија. Комбинацијата на овие методологии ќе даде контекст на експерименталните резултати и ќе понуди похолистички поглед на ефективноста на GPT моделите во задачите од софтверското инженерство.

5.3. Критериуми за евалуација

При анализа на ефикасноста на генерирањето код засновано на GPT, користиме три методологии за евалуација. Две од нив се од технички карактер (мануелна и автоматска евалуација на кодот), а третата е во облик на анкетен прашалник и дава перспектива фокусирана на мислењето на софтверските инженери. Оваа комбинација на пристапи ни овозможува да понудиме сеопфатна евалуација на овие алатки за вештачка интелигенција, истакнувајќи ги нивните технички способности и нивната реална примена во пракса.

5.3.1. Мануелна техничка евалуација врз база на систем за оценување на кодот

Првиот метод на евалуација вклучува систем на бодување кој го оценува генерираниот код врз основа на различни клучни аспекти. Секое парче генериран код рачно се оценува со помош на овој систем, со користење на следниве критериуми:

- **Функционалност:** (оценка: 1-5): Критериумот за функционалност првенствено проценува колку добро генерираниот код ги задоволува барањата на задачата за која е дизајниран. Ако кодот не ги исполни барањата на задачата или само делумно ги исполнува, ќе добие понизок резултат. Функционалноста е критична затоа што ефективен модел за генерирање код треба да произведе функционален код кој точно и целосно ја исполнува својата намена.
- **Точност:** (оценка: Да/Не): Критериумот за точност проценува дали кодот се компајлира без никакви грешки. Кодот што не успева да се компајлира

е инхерентно неизвршлив и затоа не ја постигнува својата основна цел. Кодот што се компајлира беспрекорно е означен како точен, додека кодот што генерира грешки во времето на компајлирање се смета за неточен.

- **Читливост**: (оценка: 1-5): Читливоста го мери степенот до кој кодот може лесно да се чита и да се разбере од страна на програмерите. Неколку фактори придонесуваат за овој критериум, вклучувајќи го придржувањето до стандардните конвенции за именување, употребата на коментари за подобра јасност, следењето на упатствата за стилови и целокупната структура и организација на кодот. Читливоста на кодот е од суштинско значење бидејќи кодот кој лесно се одржува и ажурира води до негова подобра долгорочна употребливост и ефективност.
- **Оптималност:** (оценка: 1-5): Критериумот за оптималност проценува колку генерираниот код оптимално ги користи ресурсите, избегнувајќи непотребни операции или непотребни пресметки. Оптималниот код не само што покажува подобри перформанси туку и ја нагласува ефективноста на моделот за генерирање код.

Овој систем на бодување овозможува квантитативна споредба на GPT моделите врз база на нивната ефикасност во генерирањето код. Земајќи ги предвид субјективните мерки (како што се функционалноста и читливоста) и објективните мерки (како точноста и оптималноста), овој начин на евалуација овозможува сеопфатна и квантитативна споредба на GPT моделите.

5.3.2. Автоматска техничка евалуација со алатката SonarQube

Втората методологија за евалуација на кодот е спроведена со употреба на алатката SonarQube. Оваа алатка прави автоматска анализа на кодот која е фокусирана на различни аспекти. Резултатите даваат детална и квантитативна мерка за квалитетот на кодот генериран од овие модели. Проблемите кои ги детектира SonarQube се следниве:

- **Број на грешки** (мерење: нумерички): Овој критериум го брои бројот на грешки во кодирањето што може да доведат до неуспех на програмата, како што е идентификувано од SonarQube. Помалку грешки сугерираат повисок квалитет и сигурност на кодот.
- **Број на слабости (Code Smells)** (мерење: нумерички): Овој критериум ги идентификува слабостите на дизајнот кои можеби нема да предизвикаат

неуспех на програмата, но може да ја попречат брзината на развој или да ги зголемат идните ризици. Помалку слабости означува поголемо придржување до најдобрите практики за кодирање.

- **Број на безбедносни пропусти** (мерење: нумерички): Овој критериум ги брои деловите во кодот што SonarQube ги означува како потенцијални безбедносни пропусти. Помалку безбедносни жаришта укажуваат на помала веројатност за безбедносни проблеми во кодот.

Резултатите од оваа автоматска анализа го дополнуваат нашиот систем за рачно бодување, обезбедувајќи дополнителни квантитативни податоци за перформансите на моделите.

5.3.3. Евалуација базирана на мислењата на софтверските инженери

Дополнувајќи ги нашите технички методи на евалуација, ги користиме податоците собрани од нашата анкета за задоволството на софтверските инженери за да понудиме евалуација фокусирана на активните корисници на алатките. Овој пристап вклучува анализа на одговорите на анкетата за да се добие увид во следните области:

- **Практична примена:** Колку често се користат алатките и за какви видови задачи?
- **Проценета точност:** како корисниците ја оценуваат точноста на генерираниот код?
- **Влијание врз ефикасноста на работата:** Дали корисниците сметаат дека алатките им заштедуваат време или ги намалуваат грешките во кодирањето?
- **Предности и ограничувања:** Кои се главните придобивки и недостатоци на овие алатки кои ги перцепираат корисниците?
- **Идни перспективи:** Дали корисниците веруваат дека овие алатки ќе станат стандардни алатки во областа на развој на софтвер и дали би ги препорачале на други?
- **Предложени подобрувања:** Какви подобрувања би сакале корисниците да видат во идните верзии на овие алатки?

Податоците добиени од оваа анкета служат за обезбедување практична, искуствена димензија на нашата евалуација, збогатувајќи го нашето разбирање за ефективноста на овие алатки за вештачка интелигенција надвор од чисто техничките аспекти.

Со комбинирање на наодите од нашите технички проценки засновани на бодување и автоматизирани резултати, заедно со нашата евалуација фокусирана на реални корисници, имаме за цел да обезбедиме сеопфатна и избалансирана проценка на ефективноста на алатките за генерирање кодови базирани на GPT во задачите на софтверското инженерство.

6. РЕЗУЛТАТИ И АНАЛИЗА

Во ова поглавје, ги презентираме и анализираме резултатите од нашата евалуација на ефективноста на GPT-3.5 и GPT-4 моделите во генерирањето на код. Нашите резултати произлегуваат од ригорозна методологија за евалуација со три методи, која се состои од рачен систем за бодување, автоматизирана техничка евалуација со SonarQube и опсежна анкета фокусирана на корисникот, која вклучува активни практичари од софтверско инженерство.

Преку оваа сеопфатна презентација и анализа на резултатите, имаме за цел да понудиме сеопфатно разбирање на ефективноста на GPT моделите, разјаснувајќи ги не само нивните технички способности туку и нивните практични импликации.

6.1. Резултати за ефикасноста на генерираниот код

Во ова поглавје се претставени резултатите од техничките евалуации и е направена анализа врз база на добиените резултати.

6.1.1. Поединечни резултати за секоја од задачите

Во продолжение се дадени резултатите за секоја од задачите поединечно:

- **Задача 1 (T1): Креирање на веб-страница**

Во Задача 1, и GPT-3.5 и GPT-4 се покажаа релативно добро на сите нивоа на тежина.

Во зависност од нивоата на тежина, се бараше да се напише код за создавање основни и посложени веб-страници, соодветно, со специфични елементи како што се заглавија, подножја и делови за содржина. И двата модела успеаја да генерираат функционален и точен код.

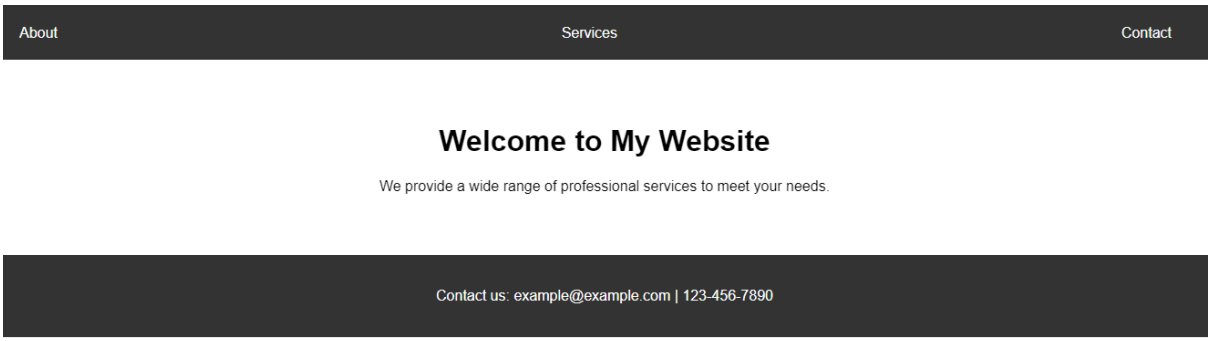
Кај двата модела беа детектирани и слабости во кодот, што сугерира дека генерираниот код можеби нема совршено да се придржува до најдобрите практики. Читливоста на кодот, е малку подобрена кај GPT-4, што може да биде корисно за програмерите кои треба да го разберат и менуваат кодот на долг рок.

Табела 1. Резултати од Задача 1

Table 1. Results of Task 1

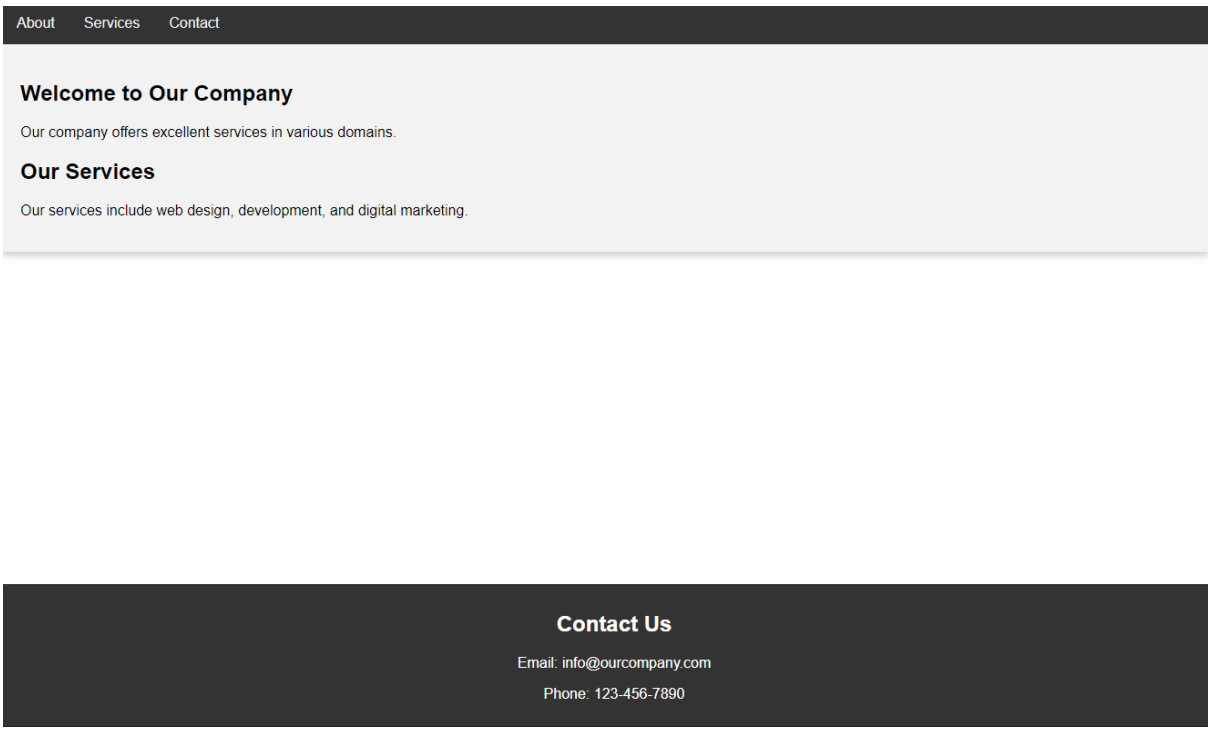
Ниво **Level**	**Критериум за евалуација** **Evaluation criteria**	**GPT-3.5**	**GPT-4**
Основно Basic	Функционалност / Functionality	5	5
	Точност / Correctness	Да	Да
	Читливост / Readability	5	5
	Оптималност / Optimality	5	5
	Бр. на грешки / No. of Bugs	1	2
	Бр. на слабости / No. of Code Smells	0	0
	Бр. на безбедносни пропусти / No. of Security Hotspots	0	0
	Технологија / Technology	HTML, CSS, JavaScript	HTML, CSS, JavaScript
Средно Intermediate	Функционалност / Functionality	4	4
	Точност / Correctness	Да	Да
	Читливост / Readability	5	5
	Оптималност / Optimality	5	5
	Бр. на грешки / No. of Bugs	1	1
	Бр. на слабости / No. of Code Smells	0	0

	Бр. на безбедносни пропусти / No. of Security Hotspots	0	0
	Технологија / Technology	HTML, CSS, JavaScript	HTML, CSS, JavaScript
Напредно Advanced	Функционалност / Functionality	3	3
	Точност / Correctness	Да	Да
	Читливост / Readability	4	4.5
	Оптималност / Optimality	4	4
	Бр. на грешки / No. of Bugs	0	0
	Бр. на слабости / No. of Code Smells	1	2
	Бр. на безбедносни пропусти / No. of Security Hotspots	0	0
	Технологија / Technology	React.js	React.js



Слика 3. Екранска снимка од веб-страницата генерирана од GPT-3.5 на средно ниво

Figure 3. Screenshot of a website generated by GPT-3.5 at the intermediate level



Слика 4. Екранска снимка од веб-страницата генерирана од GPT-4 на средно ниво

Figure 4. Screenshot of a website generated by GPT-4 at the intermediate level

- **Задача 2 (T2) - Креирање форма за најава**:

GPT-4 покажа подобри резултати во функционалноста на средно и напредно ниво, што укажува на подобро разбирање на барањата кај посложени форми за најава. Сепак, имаше безбедносни жаришта, што укажува потенцијалните пропусти во генерираниот код.

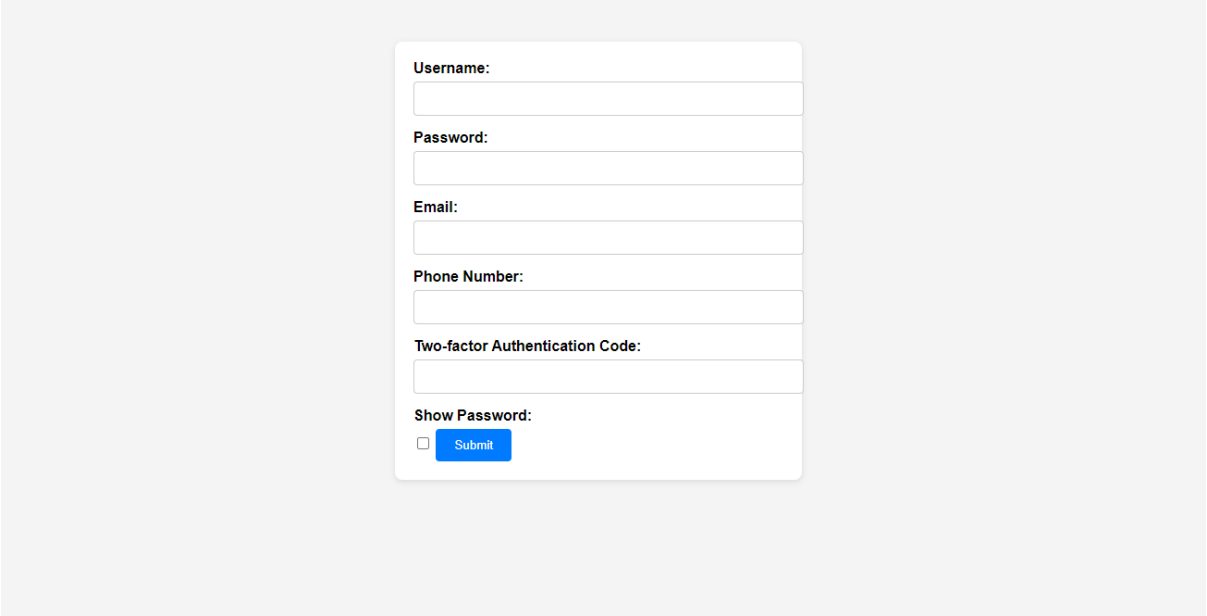
На лесно ниво, и двата модели покажаа подеднакво добри резултати, генерирајќи функционални форми за најавување без поголеми проблеми. Читливоста на генерираниот код остана конзистентна низ нивоата на тежина за двата модела.

Табела 2. Резултати од Задача 2

Table 2. Results of Task 2

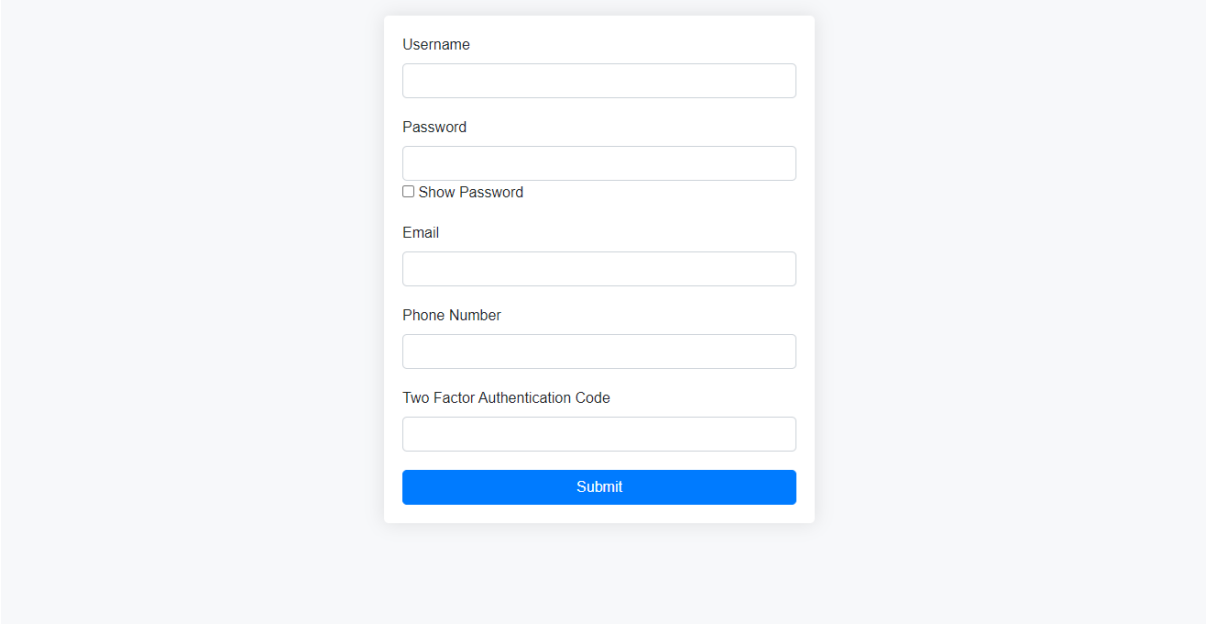
Ниво **Level**	**Критериум за евалуација** **Evaluation criteria**	**GPT-3.5**	**GPT-4**
Основно Basic	Функционалност / Functionality	5	5
	Точност / Correctness	Да	Да
	Читливост / Readability	4	5
	Оптималност / Optimality	4	5
	Бр. на грешки / No. of Bugs	1	1
	Бр. на слабости / No. of Code Smells	4	0
	Бр. на безбедносни пропусти / No. of Security Hotspots	0	0
	Технологија / Technology	HTML, CSS, JavaScript	HTML, CSS, JavaScript
Средно Intermediate	Функционалност / Functionality	4	5
	Точност / Correctness	Да	Да
	Читливост / Readability	4	4
	Оптималност / Optimality	4	4
	Бр. на грешки / No. of Bugs	1	1
	Бр. на слабости / No. of Code Smells	0	0
	Бр. на безбедносни пропусти / No. of Security Hotspots	0	2
	Технологија / Technology	HTML, CSS, JavaScript	HTML, CSS, JavaScript
Напредно Advanced	Функционалност / Functionality	3	4
	Точност / Correctness	Да	Да
	Читливост / Readability	4.5	4
	Оптималност / Optimality	4	4
	Бр. на грешки / No. of Bugs	2	1
	Бр. на слабости / No. of Code Smells	0	0
	Бр. на безбедносни пропусти / No. of Security Hotspots	1	2
	Технологија / Technology	HTML, CSS,	HTML, CSS,

		JavaScript	JavaScript



Слика 5. Екранска снимка од формата генерирана од GPT-3.5 на напредно ниво

Figure 5. Screenshot of a login form generated by GPT-3.5 at the advanced level



Слика 6. Екранска снимка од формата генерирана од GPT-4 на напредно ниво

Figure 6. Screenshot of a login form generated by GPT-4 at the advanced level

- **Задача 3 (T3) - Креирање апликација за калкулатор:**

И двата модела се покажаа подеднакво добро кај основното ниво на тежина.

На средно и напредно ниво моделите покажаа релативно слаби резултати во поглед на исполнување на барањата на задачата. GPT-4 покажа малку подобра функционалност на напредно ниво, што вклучуваше креирање на сложен научен калкулатор со различни напредни операции. Ова сугерира дека обуката на моделот можеби го подобрила неговото разбирање за сложените математички операции.

Читливоста и оптималноста на кодот беа конзистентни на сите нивоа и за GPT-3.5 и за GPT-4. Сепак, откриени беа и одредени слабости и безбедносни пропусти во кодот.

Табела 3. Резултати од Задача 3

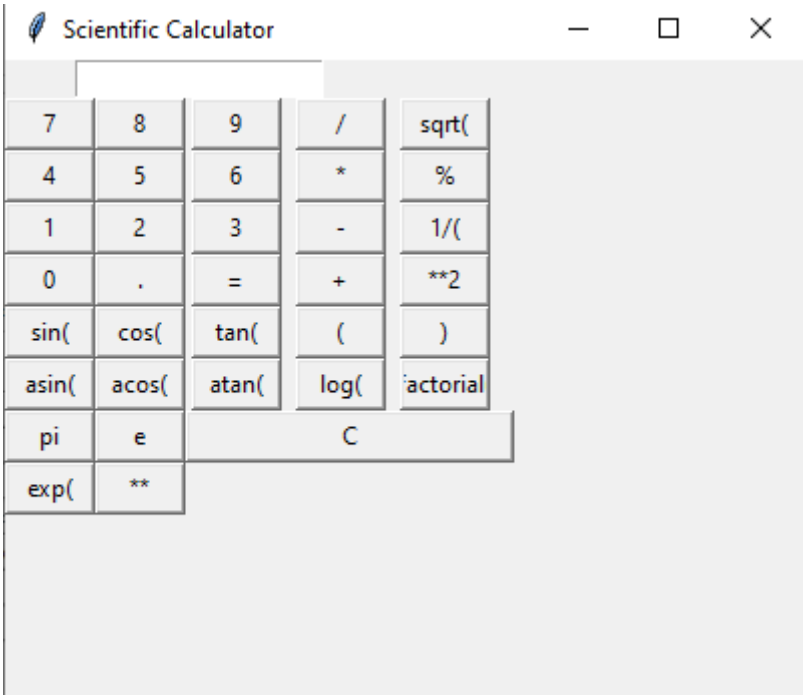
Table 3. Results of Task 3

Ниво **Level**	**Критериум за евалуација** **Evaluation criteria**	**GPT-3.5**	**GPT-4**
Основно Basic	Функционалност / Functionality	5	5
	Точност / Correctness	Да	Да
	Читливост / Readability	4	4
	Оптималност / Optimality	4	4
	Бр. на грешки / No. of Bugs	0	0
	Бр. на слабости / No. of Code Smells	0	1
	Бр. на безбедносни пропусти / No. of Security Hotspots	0	0
	Технологија / Technology	Python	Python
Средно Intermediate	Функционалност / Functionality	3	3
	Точност / Correctness	Да	Да
	Читливост / Readability	4	4
	Оптималност / Optimality	4	3
	Бр. на грешки / No. of Bugs	0	0
	Бр. на слабости / No. of Code Smells	1	1
	Бр. на безбедносни пропусти / No. of Security Hotspots	0	1

	Технологија / Technology	Python	React.js
Напредно Advanced	Функционалност / Functionality	2	3
	Точност / Correctness	Не	Не
	Читливост / Readability	4	4
	Оптималност / Optimality	3	3
	Бр. на грешки / No. of Bugs	0	0
	Бр. на слабости / No. of Code Smells	1	1
	Бр. на безбедносни пропусти / No. of Security Hotspots	0	0
	Технологија / Technology	Python	Python

Слика 7. Екранска снимка од апликацијата генерирана од GPT-3.5 на напредно ниво

Figure 7. Screenshot of an application generated by GPT-3.5 at the advanced level



Слика 8. Екранска снимка од апликацијата генерирана од GPT-4 на напредно ниво

Figure 8. Screenshot of an application generated by GPT-4 at the advanced level

- **Задача 4 (T4) - Креирање REST API:**

И GPT-3.5 и GPT-4 покажаа релативно добра функционалност на кодот сите нивоа, создавајќи функционални REST API-ја со различна комплексност. Сепак, кај GPT-4 беа детектирани повеќе слабости во кодот на средно ниво каде што ја користеше рамката Express.js. Овие слабости се однесуваа на користење “var“ наместо “let“ или “const“ при декларирањето на променливите.

На напредно ниво, двата модели се покажаа добро, но изненадувачки GPT-3.5 успеа да генерира пофункционален код задоволувајќи поголем дел од барањата на задачата. И двата модели имаа и безбедносни пропусти и слабости во кодот на ова ниво.

Табела 4. Резултати од Задача 4

Table 4. Results of Task 4

Ниво Level	**Критериум за евалуација Evaluation criteria**	**GPT-3.5**	**GPT-4**

Основно Basic	Функционалност / Functionality	5	5
	Точност / Correctness	Да	Да
	Читливост / Readability	5	5
	Оптималност / Optimality	5	5
	Бр. на грешки / No. of Bugs	0	0
	Бр. на слабости / No. of Code Smells	0	0
	Бр. на безбедносни пропусти / No. of Security Hotspots	1	1
	Технологија / Technology	Flask, Python	Express.js
Средно Intermediate	Функционалност / Functionality	5	5
	Точност / Correctness	Да	Да
	Читливост / Readability	4.5	4.5
	Оптималност / Optimality	4	4
	Бр. на грешки / No. of Bugs	0	0
	Бр. на слабости / No. of Code Smells	1	9
	Бр. на безбедносни пропусти / No. of Security Hotspots	1	1
	Технологија / Technology	Flask, Python	Express.js
Напредно Advanced	Функционалност / Functionality	4	3
	Точност / Correctness	Да	Да
	Читливост / Readability	4	4
	Оптималност / Optimality	4	4
	Бр. на грешки / No. of Bugs	0	0
	Бр. на слабости / No. of Code Smells	0	0
	Бр. на безбедносни пропусти / No. of Security Hotspots	1	1
	Технологија / Technology	Express.js	Express.js

- **Задача 5 (T5) - Пишување SQL команди:**

Двата модела се истакнаа во генерирањето на SQL команди за различни задачи. Тие можеа да се справат со различни нивоа на сложеност, вклучително и спојување табели, филтрирање податоци и пресметување на средни вредности. И GPT-3.5 и GPT-4 произведоа точни и функционални SQL команди без некои поголеми проблеми.

Читливоста и оптималноста на генерираните SQL команди беа конзистентни кај двата модели и сите нивоа на тежина.

Табела 5. Резултати од Задача 5

Table 5. Results of Task 5

Ниво **Level**	**Критериум за евалуација** **Evaluation criteria**	**GPT-3.5**	**GPT-4**
Основно Basic	Функционалност / Functionality	5	5
	Точност / Correctness	Да	Да
	Читливост / Readability	5	5
	Оптималност / Optimality	5	5
	Бр. на грешки / No. of Bugs	/	/
	Бр. на слабости / No. of Code Smells	/	/
	Бр. на безбедносни пропусти / No. of Security Hotspots	/	/
	Технологија / Technology	SQL	SQL
Средно Intermediate	Функционалност / Functionality	5	5
	Точност / Correctness	Да	Да
	Читливост / Readability	4.5	5
	Оптималност / Optimality	5	5
	Бр. на грешки / No. of Bugs	/	/
	Бр. на слабости / No. of Code Smells	/	/
	Бр. на безбедносни пропусти / No. of Security Hotspots	/	/
	Технологија / Technology	SQL	SQL
Напредно	Функционалност / Functionality	5	5

Advanced	Точност / Correctness	Да	Да
	Читливост / Readability	4.5	4
	Оптималност / Optimality	4	4
	Бр. на грешки / No. of Bugs	/	/
	Бр. на слабости / No. of Code Smells	/	/
	Бр. на безбедносни пропусти / No. of Security Hotspots	/	/
	Технологија / Technology	SQL	SQL

- **Задача 6 (T6) - Пишување рекурзивни функции:**

GPT-4 покажа супериорност во однос на функционалноста на напредно ниво, што укажува на подобро разбирање на сложените алгоритми базирани на рекурзија. Читливоста и оптималноста на кодот останаа конзистентни на сите нивоа за двата модели, а кај средното ниво беа детектирани и слабости во кодот.

Табела 6. Резултати од Задача 6

Table 6. Results of Task 6

Ниво Level	**Критериум за евалуација Evaluation criteria**	**GPT-3.5**	**GPT-4**
Основно Basic	Функционалност / Functionality	5	5
	Точност / Correctness	Да	Да
	Читливост / Readability	4.5	5
	Оптималност / Optimality	5	5
	Бр. на грешки / No. of Bugs	0	0
	Бр. на слабости / No. of Code Smells	0	0
	Бр. на безбедносни пропусти / No. of Security Hotspots	0	0
	Технологија / Technology	Python	Python
Средно	Функционалност / Functionality	5	5

Intermediate	Точност / Correctness	Да	Да
	Читливост / Readability	4.5	4.5
	Оптималност / Optimality	4.5	4.5
	Бр. на грешки / No. of Bugs	0	0
	Бр. на слабости / No. of Code Smells	1	1
	Бр. на безбедносни пропусти / No. of Security Hotspots	0	0
	Технологија / Technology	Python	Python
Напредно Advanced	Функционалност / Functionality	1.5	5
	Точност / Correctness	Да	Да
	Читливост / Readability	4	4
	Оптималност / Optimality	4	4.5
	Бр. на грешки / No. of Bugs	0	0
	Бр. на слабости / No. of Code Smells	0	0
	Бр. на безбедносни пропусти / No. of Security Hotspots	0	0
	Технологија / Technology	Python	Python

- **Задача 7 (T7) - Манипулација со стрингови:**

И GPT-3.5 и GPT-4 се покажаа подеднакво добро во задачите за манипулација со стрингови. Тие генерираа код за броење на појавувања на зборови, за конвертирање на низи во Морзеова шифра и за замена на специфичните појавувања на одредени зборови по потреба. Генерираниот код беше функционален, точен и читлив за сите нивоа на тежина.

Табела 7. Резултати од Задача 7

Table 7. Results of Task 7

Ниво Level	**Критериум за евалуација Evaluation criteria**	**GPT-3.5**	**GPT-4**
Основно Basic	Функционалност / Functionality	5	5

	Точност / Correctness	Да	Да
	Читливост / Readability	5	5
	Оптималност / Optimality	4	4
	Бр. на грешки / No. of Bugs	0	0
	Бр. на слабости / No. of Code Smells	0	0
	Бр. на безбедносни пропусти / No. of Security Hotspots	0	0
	Технологија / Technology	Python	Python
Средно Intermediate	Функционалност / Functionality	4	5
	Точност / Correctness	Да	Да
	Читливост / Readability	5	5
	Оптималност / Optimality	4	5
	Бр. на грешки / No. of Bugs	0	0
	Бр. на слабости / No. of Code Smells	0	0
	Бр. на безбедносни пропусти / No. of Security Hotspots	0	0
	Технологија / Technology	Python	Python
Напредно Advanced	Функционалност / Functionality	5	5
	Точност / Correctness	Да	Да
	Читливост / Readability	4	5
	Оптималност / Optimality	4.5	4.5
	Бр. на грешки / No. of Bugs	0	0
	Бр. на слабости / No. of Code Smells	0	0
	Бр. на безбедносни пропусти / No. of Security Hotspots	0	0
	Технологија / Technology	Python	Python

- **Задача 8 (T8) - Манипулација со низи и матрици:**

И двата модела покажаа добри перформанси во манипулирањето со низи и матрици на средно и напредно ниво. Тие генерираа кодови за пресметување

збирови, за ротирање матрици и за реорганизирање низи и матрици според специфични барања. Генерираниот код беше функционален и точен, со добра читливост и оптималност на овие две нивоа.

Сепак, кај напредното ниво, и двата модела се соочија со потешкотии при генерирање на кодот. Кодот кој го генерираше GPT-4 беше значително подобар од оној на GPT-3.5, но и двата модела не ги исполнија во целост барањата на задачата на ова ниво. Ова укажува дека и покрај тоа што GPT-4 подобро резонира и ги разбира комплексните случаеви, сепак не може во целост да одговори на задачи кои бараат повисоко ниво на резонирање.

Табела 8. Резултати од Задача 8

Table 8. Results of Task 8

Ниво **Level**	**Критериум за евалуација** **Evaluation criteria**	**GPT-3.5**	**GPT-4**
Основно Basic	Функционалност / Functionality	5	5
	Точност / Correctness	Да	Да
	Читливост / Readability	4.5	5
	Оптималност / Optimality	4	4.5
	Бр. на грешки / No. of Bugs	0	0
	Бр. на слабости / No. of Code Smells	0	0
	Бр. на безбедносни пропусти / No. of Security Hotspots	0	0
	Технологија / Technology	Python	Python
Средно Intermediate	Функционалност / Functionality	5	5
	Точност / Correctness	Да	Да
	Читливост / Readability	5	5
	Оптималност / Optimality	4	4.5
	Бр. на грешки / No. of Bugs	0	0
	Бр. на слабости / No. of Code Smells	0	0

	Бр. на безбедносни пропусти / No. of Security Hotspots	0	0
	Технологија / Technology	Python	Python
Напредно Advanced	Функционалност / Functionality	1.5	3
	Точност / Correctness	Да	Да
	Читливост / Readability	2	4
	Оптималност / Optimality	3.5	3.5
	Бр. на грешки / No. of Bugs	0	0
	Бр. на слабости / No. of Code Smells	3	0
	Бр. на безбедносни пропусти / No. of Security Hotspots	0	0
	Технологија / Technology	Python	Python

- **Задача 9 (T9) - Манипулација со JSON објекти:**

GPT-4 покажа подобри резултати во оптималноста на основното ниво и послаба читливост на кодот на средно ниво. Инаку, и двата модела работеа слично, генерирајќи JSON код за манипулација кој ги исполнуваше наведените барања што укажува дека GPT моделите можат да бидат многу корисна алатка кога станува збор за процесирање податоци и манипулација со структури на податоци како што се JSON објектите.

Табела 9. Резултати од Задача 9

Table 9. Results of Task 9

Ниво Level	**Критериум за евалуација Evaluation criteria**	**GPT-3.5**	**GPT-4**
Основно Basic	Функционалност / Functionality	5	5
	Точност / Correctness	Да	Да
	Читливост / Readability	4.5	5
	Оптималност / Optimality	4	5
	Бр. на грешки / No. of Bugs	0	0

	Бр. на слабости / No. of Code Smells	0	0
	Бр. на безбедносни пропусти / No. of Security Hotspots	0	0
	Технологија / Technology	Python	Python
Средно Intermediate	Функционалност / Functionality	5	5
	Точност / Correctness	Да	Да
	Читливост / Readability	4.5	4
	Оптималност / Optimality	3.5	3.5
	Бр. на грешки / No. of Bugs	0	0
	Бр. на слабости / No. of Code Smells	0	0
	Бр. на безбедносни пропусти / No. of Security Hotspots	0	0
	Технологија / Technology	Python	Python
Напредно Advanced	Функционалност / Functionality	5	5
	Точност / Correctness	Да	Да
	Читливост / Readability	4.5	4.5
	Оптималност / Optimality	4	4
	Бр. на грешки / No. of Bugs	0	0
	Бр. на слабости / No. of Code Smells	0	0
	Бр. на безбедносни пропусти / No. of Security Hotspots	0	0
	Технологија / Technology	Python	Python

- **Задача 10 (T10) - Цртање објекти со HTML5 Canvas:**

И двата модела покажаа релативно добри резултати кога станува збор за генерирање код за цртање објекти кои немаат преголема сложеност.

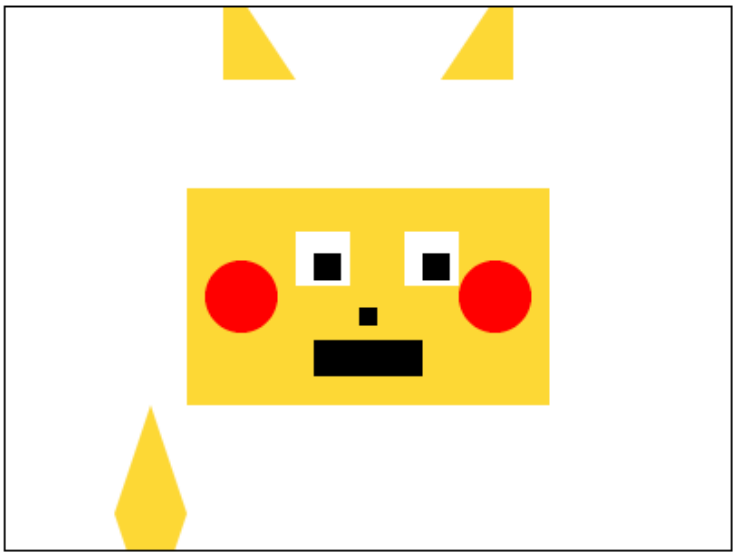
Но, кај напредно ниво, каде се бараше генерирање на посложени форми, проблемите кај двата модела беа евидентни бидејќи крајниот резултат кој се доби со извршување на генерираниот код ни оддалеку не беше реалистичен.

Табела 10. Резултати од Задача 10

Table 10. Results of Task 10

Ниво **Level**	**Критериум за евалуација** **Evaluation criteria**	**GPT-3.5**	**GPT-4**
Основно Basic	Функционалност / Functionality	5	5
	Точност / Correctness	Да	Да
	Читливост / Readability	5	4.5
	Оптималност / Optimality	5	5
	Бр. на грешки / No. of Bugs	1	1
	Бр. на слабости / No. of Code Smells	0	0
	Бр. на безбедносни пропусти / No. of Security Hotspots	0	0
	Технологија / Technology	HTML, Javascript	HTML, Javascript
Средно Intermediate	Функционалност / Functionality	4	5
	Точност / Correctness	Да	Да
	Читливост / Readability	4.5	4.5
	Оптималност / Optimality	4	4.5
	Бр. на грешки / No. of Bugs	1	1
	Бр. на слабости / No. of Code Smells	0	0
	Бр. на безбедносни пропусти / No. of Security Hotspots	0	0
	Технологија / Technology	HTML, Javascript	HTML, Javascript
Напредно Advanced	Функционалност / Functionality	3	3
	Точност / Correctness	Да	Да
	Читливост / Readability	4.5	4.5
	Оптималност / Optimality	4	4
	Бр. на грешки / No. of Bugs	1	2
	Бр. на слабости / No. of Code Smells	0	0

	Бр. на безбедносни пропусти / No. of Security Hotspots	0	0
	Технологија / Technology	HTML, Javascript	HTML, Javascript



Слика 9. HTML5 Canvas објект генериран од GPT-3.5 на напредно ниво

Figure 9. HTML5 Canvas object generated by GPT-3.5 at the advanced level

Слика 10. HTML5 Canvas објект генериран од GPT-4 на напредно ниво

Figure 10. HTML5 Canvas object generated by GPT-4 at the advanced level

6.1.2. Анализа на проблематичен код

Како дел од нашето сеопфатно истражување, од клучно значење е да се навлезе подлабоко во случаите каде што генерираниот код можеби не ги

исполнил специфичните барања на задачата или покажал потенцијални слабости. И покрај тоа што GPT моделите покажаа импресивни перформанси во различни задачи, имаше случаи каде генерираниот код не ги исполни во целост зададените барања и тој беше со лош квалитет.

Во продолжение ќе ги погледнеме специфичните задачи каде што генерираниот код не е добар, било во смисла на функционалност, оптималност или безбедност. Преку детална анализа на проблематичниот код, имаме за цел да идентификуваме обрасци или тенденции во перформансите на GPT кои би можеле да дадат појасна слика за идните итерации на овие модели и да обезбедат вредни сознанија за софтверските инженери кои размислуваат за генерирање код со помош на GPT.

- **Задача 6 (T6) - напредно ниво, GPT-3.5**

Целта на Задача 6 на напредно ниво беше да се генерира рекурзивна функција за да се утврди дали е изводливо да се избере подмножество цели броеви од дадена низа така што нивниот збир е еднаков на одреден зададен број. Функцијата беше дополнително ограничена да ги вклучи сите множители на 5 во низата и да го исклучи секое 1 кое следи веднаш по бројот кој е множител на 5.

Кодот кој го генерираше GPT-3.5 моделот е даден во продолжение:

```
def check_sum(start, array, target):
    # Base cases
    if target == 0:
        return True
    if start >= len(array):
        return False
    # Check if the current element is a multiple of 5 and not followed
by 1
    if array[start] % 5 == 0 and (start + 1 >= len(array) or
array[start + 1] != 1):
        # Include the current element in the sum
        if check_sum(start + 1, array, target - array[start]):
            return True
    # Exclude the current element from the sum
    if check_sum(start + 1, array, target):
        return True
    return False
```

По оценувањето на генерираниот код за оваа задача, забележавме дека тој целосно не ги исполнува барањата на задачата. Во продолжение се дадени проблемите кои беа детектирани кај дадениот код.

Прво, генерираното решение не гарантира вклучување на сите множители од 5 присутни во низата. Логиката условно ги проверува, овозможувајќи потенцијално исклучување на множители на 5. Ова го нарушува барањето на задачата дека сите множители од 5 мора да бидат вклучени во групата за сумирање.

Второ, логиката во врска со бројот 1 што следи по множител на 5 не е претставена соодветно во дадениот код. Обезбедената функција проверува дали множителот на 5 е проследен со бројот 1, но наместо да го исклучи бројот 1 (како што е предвидено во барањата), кодот го условува само вклучувањето на множителот на 5 врз основа на следниот број. Ова значи дека кодот понекогаш може погрешно да го вклучи бројот 1 кога следи по множител на 5, спротивно на спецификациите на задачата.

Трето, еден од основните аспекти на оваа задача е проблемот со збирот на генеричките подмножества. Се чини дека генерираното решение значително се фокусира на множителите на 5 и последователниот број 1, што ненамерно ја засенува поопштата логика на збирот на подмножеството. Иако го вклучува или исклучува тековниот елемент, тој не го опфаќа сеопфатно аспектот на сумата на подмножеството, особено кога се разгледуваат другите ограничувања поврзани со 5 и 1.

Од друга страна, вреди се напомене дека кога му беше претставена истата задача, GPT-4 успеа да ги надмине гореспоменатите предизвици и да генерира функционален код. Кодот генериран од GPT-4 (даден во продолжение) покажува подобро разбирање на барањата за проблемот и нејзините ограничувања.

```
def check_sum(start_idx, arr, target):
    # Check if the target is less than 0, if so return False
    if target < 0:
        return False
      # If we've reached the end of the array, check if target is 0
(means we've found a combination)
    if start_idx == len(arr):
        return target == 0

    # If the current number is multiple of 5, we have to include it
    if arr[start_idx] % 5 == 0:
        # If next number is 1, skip it
        if start_idx + 1 < len(arr) and arr[start_idx + 1] == 1:
                    return  check_sum(start_idx  +  2,  arr,  target  -
arr[start_idx])
        else:
                    return  check_sum(start_idx  +  1,  arr,  target  -
arr[start_idx])
    else:
        # Either include the current number or don't
          return check_sum(start_idx + 1, arr, target - arr[start_idx])
or check_sum(start_idx + 1, arr, target)
```

Решението на GPT-4 безусловно го вклучува секој елемент што е множител на 5 во сумирањето, што го задоволува основниот услов. Условната проверка (`if arr[start_idx] % 5 == 0`) осигурува дека секој множител од 5 во низата задолжително се зема во предвид за собирањето.

GPT-4 експлицитно се осврнува на проблемот каде што бројот 1 треба да се прескокне ако следи по множител на 5. Овој случај е прецизно обработен преку условот `if start_idx + 1 < len(arr) and arr[start_idx + 1] == 1`.

Рекурзивните повици направени во решението на GPT-4 елегантно го балансираат вклучувањето и исклучувањето на тековниот елемент, додека во исто време се гарантира дека се почитуваат ограничувањата околу броевите 5 и 1. Ова покажува посеопфатен и покохерентен пристап кон проблемот со збирот на подмножеството.

По решението на оваа задача, можеме да заклучиме дека GPT-4 покажа значителна супериорност при разбирање на барањата на задачата, решавајќи ја истата со прецизност и јасност, нудејќи решение што ги исполнува сите пропишани критериуми.

- **Задача 8 (T8) - напредно ниво, GPT-3.5**

Целта на задачата 8 на напредно ниво беше да се напише функција која треба да ја реорганизира влезната низа, така што секоја појава на бројот 4 да е веднаш проследена со 5. Положбите на 4-ките се фиксирани, додека другите броеви може да се преместуваат.

Кодот кој го генерираше GPT-3.5 е даден во продолжение:

```
def reorganize_array(arr):
    result = []
    i = 0
    while i < len(arr):
        if arr[i] == 4:
            result.append(arr[i])
             next_index = arr.index(5, i)  # Find the index of the next
occurrence of 5
            result.append(arr[next_index])
            i = next_index + 1  # Skip the processed 4 and 5
        else:
            result.append(arr[i])
            i += 1
    return result
```

По разгледувањето на дадениот код за оваа задача, увидовме неколку проблеми што може да придонесат за неговата неможност да ги помине сите тест случаи. Нив ќе ги разгледаме во продолжение.

Логиката претставена во кодот може потенцијално да ги прескокне броевите што се наоѓаат помеѓу броевите 4 и 5. Кога кодот ќе наиде на 4, тој го додава 4 и следниот број 5 на резултатот и потоа скока на наредната позиција по пронајдената 5-ка. Ова може да доведе до губење на други броеви кои би биле помеѓу 4 и 5 во оригиналната низа.

Исто така, проблем е што бројот 5 се додава на листата со резултати секогаш кога ќе се најде 4, без разлика дали 5-ката веќе била во непосредна близина на 4-та во влезната низа. Овој пристап може да ја преуреди низата од броеви и да

го наруши редоследот на оригиналната низа и потенцијално погрешно да ги смести другите броеви во низата.

6.1.3. Просечна ефикасност и стандардни девијации

Во следниве две табели се претставени резултатите од просечните вредности за ефикасноста на GPT моделите за секој од критериумите за евалуација, како и стандардните девијации.

Овие информации ни помагаат да добиеме појасна слика за просечната ефикасност на моделите во секое од нивоата на тежина, да направиме споредба помеѓу двата модела, како и да го утврдиме степенот на варијабилност на перформансите на моделите за секоја од задачите преку стандардните девијации.

Табела 11. Просечни вредности за ефикасноста на моделите

Table 11. The estimated average values for the efficiancy of the models

Ниво **Level**	**Критериум за евалуација** **Evaluation criteria**	**GPT-3.5**	**GPT-4**
Основно Basic	Просечна функционалност / Avg. Functionality	5.0	5.0
	Просечна точност / Avg. Correctness	100%	100%
	Просечна читливост / Avg. Readability	4.65	4.85
	Просечна оптималност / Optimality	4.5	4.75
	Просечен бр. на грешки / Avg. No. of Bugs	0.3	0.4
	Просечен бр. на слабости / No. of Code Smells	0.4	0.1
	Просечен бр. на безбедносни пропусти / No. of Security Hotspots	0.1	0.1
Средно Intermediate	Просечна функционалност / Avg. Functionality	4.4	4.7
	Просечна точност / Avg. Correctness	100%	100%
	Просечна читливост / Avg. Readability	4.55	4.55
	Просечна оптималност / Optimality	4.2	4.3
	Просечен бр. на грешки / Avg. No. of Bugs	0.3	0.3

	Просечен бр. на слабости / No. of Code Smells	0.3	1.1
	Просечен бр. на безбедносни пропусти / No. of Security Hotspots	0.1	0.4
Напредно Advanced	Просечна функционалност / Avg. Functionality	3.3	3.9
	Просечна точност / Avg. Correctness	90%	90%
	Просечна читливост / Avg. Readability	4	4.25
	Просечна оптималност / Optimality	3.9	3.95
	Просечен бр. на грешки / Avg. No. of Bugs	0.3	0.3
	Просечен бр. на слабости / No. of Code Smells	0.5	0.3
	Просечен бр. на безбедносни пропусти / No. of Security Hotspots	0.2	0.3

Според добиените резултати можеме да забележиме дека кај основното ниво, двата модели покажаа одлична функционалност и точност на кодот, постигнувајќи просечен резултат од 5.0 и 100% точност соодветно. Ова покажува дека двата модели можеа да се справат со поедноставните задачи без поголеми проблеми. GPT-4 беше подобар во однос на читливоста и оптималноста, што значи дека кодот кој го генерирал е поразбирлив и поефикасен. Иако бројот на грешки и безбедносни жаришта беа релативно мал кај двата модели, GPT-4 имаше помалку слабости во кодот, што укажува на подобро придржување кон најдобрите практики за кодирање.

Кај средното ниво на тежина и двата модела имаа мало намалување на функционалноста, при што сепак GPT-4 имаше подобри резултати. Ова сугерира дека GPT-4 може да има подобро разбирање за сложените барања во споредба со GPT-3.5. Читливоста на кодот остана конзистентна за двата модела. Сепак, GPT-4 покажа малку поголем број слабости во кодот и безбедносни жаришта, што може да се должи на неговата способност да се справи со посложени задачи, потенцијално воведувајќи повеќе простор за грешки или неоптимален код.

Во напредно ниво, двата модела се соочија со забележителен пад на функционалноста, каде што GPT-4 имаше значително подобри перформанси. Ова сугерира дека и покрај подобрувањата во GPT-4, тој сè уште има тешкотии да се бори со задачи кои бараат високо ниво на апстрактно расудување или сложено решавање на проблеми. Точноста на моделите е малку намалена на 90%, но сепак претставува високо ниво на успех во генерирањето на код кој се извршува без грешка уште при првиот обид. Во однос на читливоста и оптималноста, GPT-4 беше малку подобар. Бројот на грешки остана конзистентен и кај средното ниво, додека бројот на слабости на кодот и безбедносни жаришта е незначително поголем кај GPT-4.

Просечните резултати покажуваат дека иако двата модела се релативно добри во генерирањето код, GPT-4 покажува супериорност кога се работи за посложени задачи. Сепак, вреди да се напомене дека GPT-4 покажа повеќе потенцијални слабости во кодот и повеќе потенцијални безбедносни жаришта, што треба да се земе предвид кога се разгледува примената на овие модели во практиката на софтверско инженерство. Овие резултати сугерираат дека иако моделите GPT можат да генерираат работен и разумно оптимизиран код, програмерите сепак треба да го прегледаат и да го усовршат генерираниот код за да го обезбедат неговиот квалитет, оптималност и безбедност.

Табела 12. Стандардни девијации (STD) за некои од критериумите
Table 12. Standard deviations (STD) for some of the evaluation criteria

Ниво **Level**	**Критериум за евалуација** **Evaluation criteria**	**GPT-3.5**	**GPT-4**
Основно Basic	Функционалност / Functionality (STD)	0.00	0.00
	Читливост / Readability (STD)	0.41	0.34
	Оптималност / Optimality (STD)	0.53	0.42
Средно Intermediate	Функционалност / Functionality (STD)	0.70	0.67
	Читливост / Readability (STD)	0.37	0.44
	Оптималност / Optimality (STD)	0.48	0.67
Напредно	Функционалност / Functionality (STD)	1.40	0.99

Advanced	Читливост / Readability (STD)	0.75	0.35
	Оптималност / Optimality (STD)	0.39	0.44

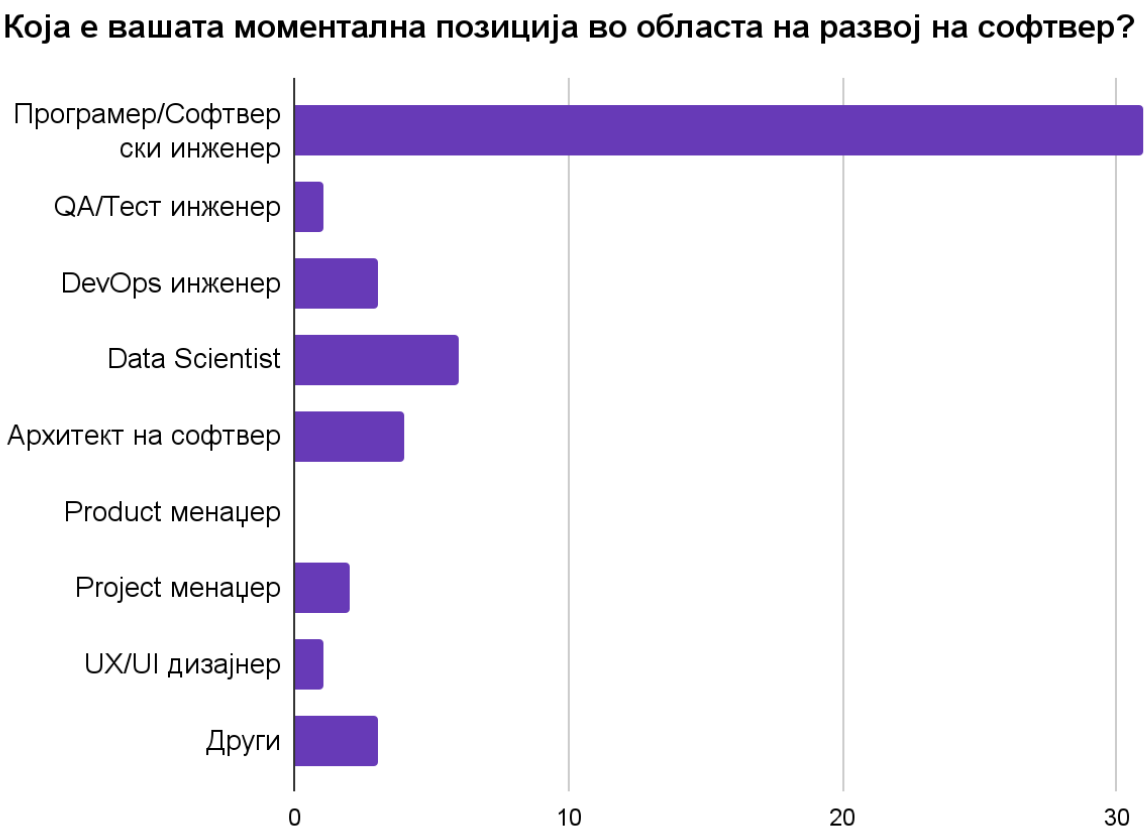
Во однос на добиените резултати за стандардните девијации, кај функционалноста на основно ниво, и двата модела имаат највисока оцена (0) што укажува дека нема варијации во функционалноста и нивната изведба била конзистентна во сите основни задачи. Резултатите за читливост и оптималност кај GPT-4 се помалку дисперзирани од оние на GPT-3.5, што значи дека GPT-4 даде поконзистентни перформанси во смисла на производство на читлив и оптимален код.

Дисперзијата на оценките за функционалност се зголеми за двата модели на средно ниво, при што GPT-3.5 покажа малку поголема варијабилност од GPT-4. Ова значи дека GPT-3.5 има помалку конзистентни перформанси кога се справува со задачи со средна сложеност. Во однос на читливоста и оптималноста на кодот, GPT-3.5 имаше помала варијабилност кај овие два параметри, што укажува на поконзистентни перформанси во однос на GPT-4.

На напредно ниво, варијабилноста во оценките за функционалност дополнително се зголеми за двата модели, при што GPT-3.5 покажува повисоко ниво на STD. Ова сугерира дека перформансите на GPT-3.5 во генерирањето функционален код се разликуваат пошироко кога се справуваат со напредни задачи. За читливоста, GPT-4 покажа помала варијабилност, што сугерира дека е поконзистентен во генерирањето на читлив код кај сложени задачи. За оптималност, двата модели имаа слично ниво на дисперзија, што укажува на споредлива конзистентност во генерирањето на оптимален код.

6.2. Резултати од истражувањето за мислењата на софтверските инженери

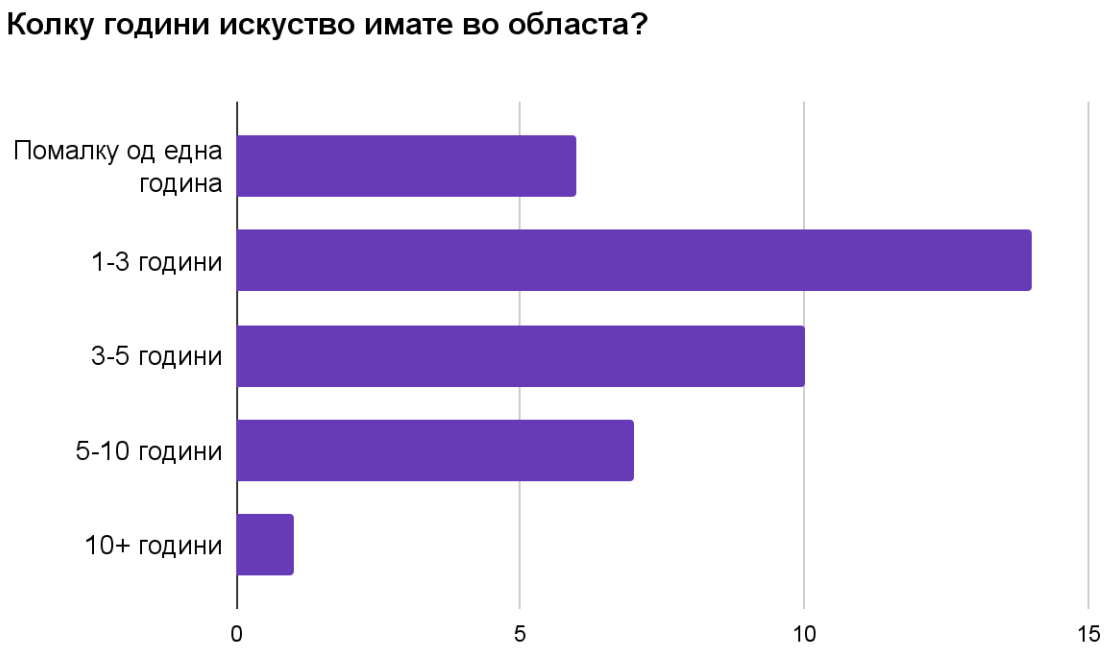
Резултатите од спореведеното истражување за мислењата на софверските инжинери ќе ни помогнат да добиеме појасна слика за насоката во која се движат тековите во софтверското инженерство кога е во прашање примената на вештачката интелигенција за задачите за генерирање код.



Графикон 1. Резултати од првото прашање

Chart 1. First question results

Според добиените резултати од првото прашање, примерокот од анкетата е првенствено составен од програмери и софтверски инженери (31 од 38). Други позиции вклучуваат: QA/Test инженери, DevOps инженери, Data Scientist-и, архитекти на софтвер и проект менаџери. Оваа разновидност на улоги илустрира широк интерес и примена на алатки базирани на GPT во различни области во развојот на софтвер.

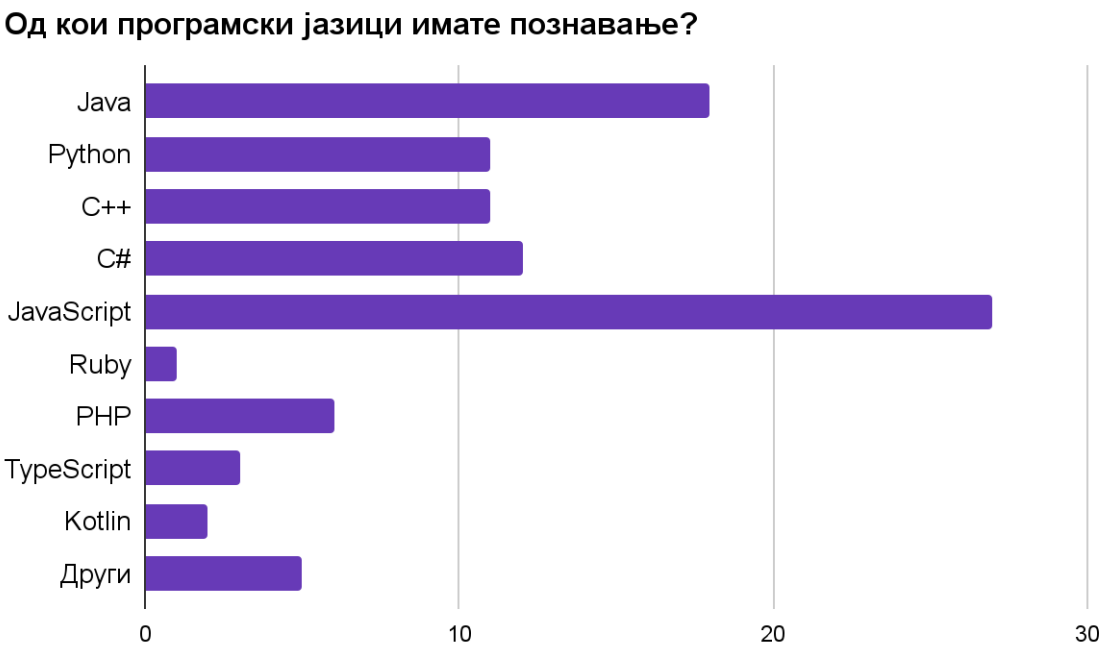


Графикон 2. Резултати од второто прашање

Chart 2. Second question results

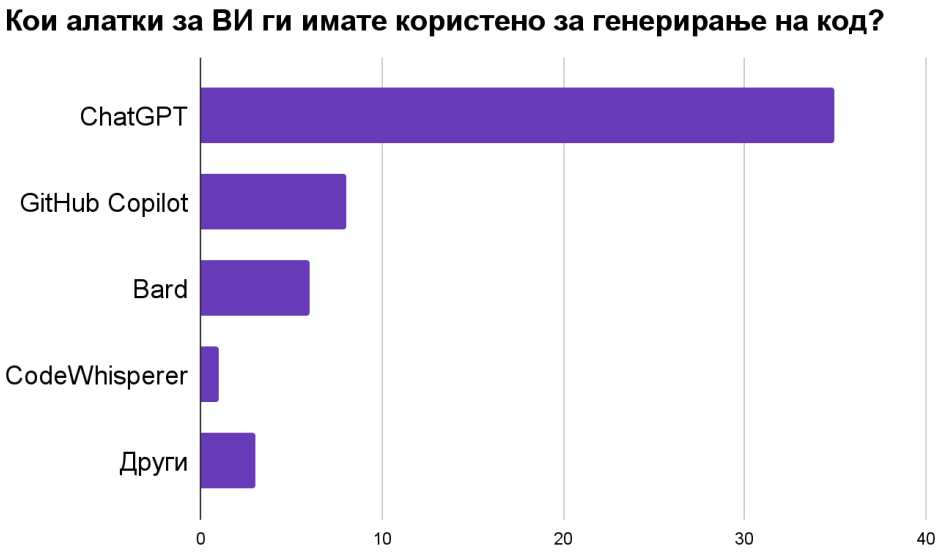
Годините на искуство меѓу учесниците се движат од помалку од една година до повеќе од една деценија. Повеќето од нив спаѓаат во границата од 1-3 години (14) и 3-5 години (10), што покажува потенцијална корелација помеѓу примената на алатките за вештачка интелигенција и помладата демографска популација во областа.

Според резултатите за познавање на програмски јазици, JavaScript беше најчестиот јазик (27), следен е Java (18), C# (12) и Python (11). Ова укажува на примената на алатките базирани на GPT врз повеќе програмски јазици, што покажува дека овие алатки можат да служат на широк опсег на развивачи.



Графикон 3. Резултати од третото прашање

Chart 3. Third question results

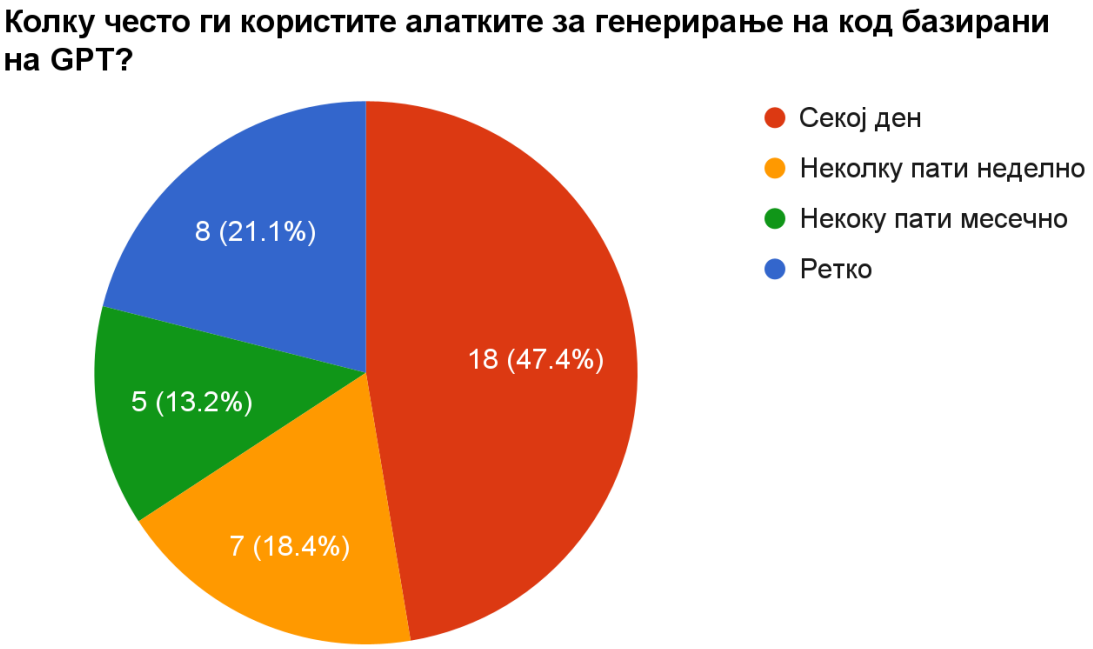


Графикон 4. Резултати од четвртото прашање

Chart 4. Fourth question results

ChatGPT е најкористената алатка за вештачка интелигенција меѓу испитаниците (35), што укажува на нејзината популарност и широко усвојување. Другите алатки како GitHub Copilot, Bard и CodeWhisperer имаат помалку

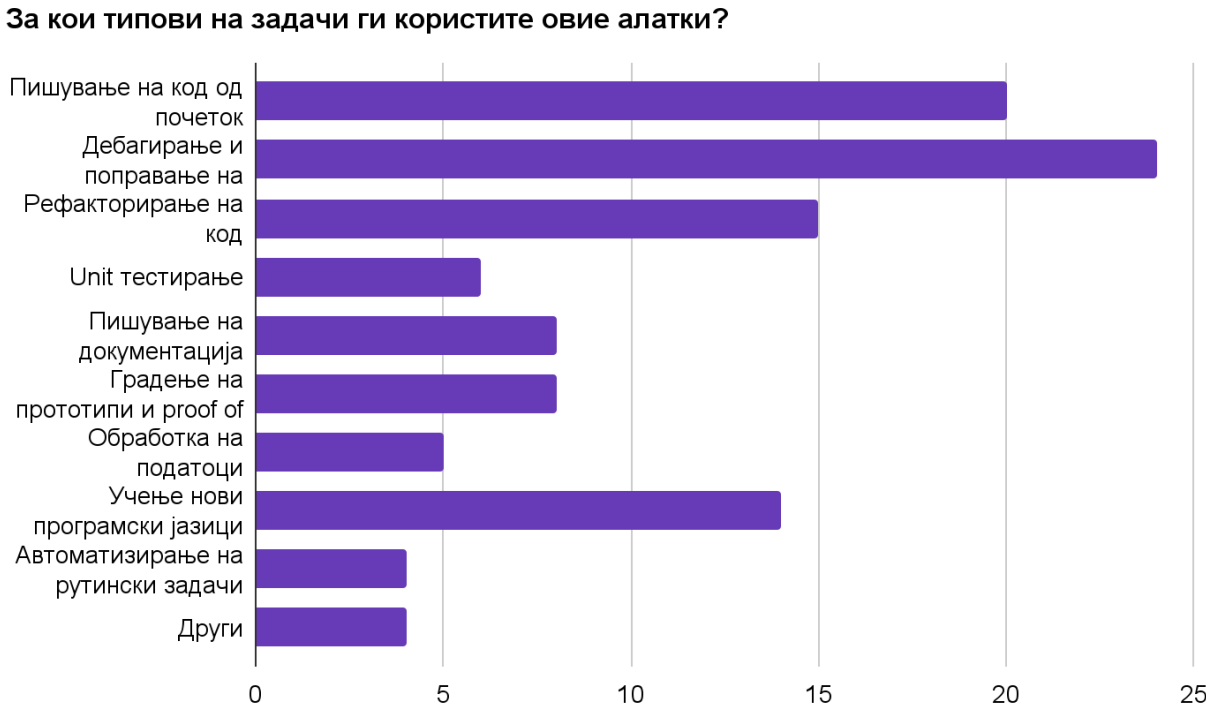
корисници, што може да ги одрази разликите во пристапноста, карактеристиките и корисничките параметри.



Графикон 5. Резултати од петтото прашање

Chart 5. Fifth question results

Учесниците во анкетата доста често користат генератори на код базирани на GPT, при што 25 од нив ги користат овие алатки секој ден или неколку пати неделно. Овие алатки се користат за задачи како што се: пишување код од нула, дебагирање и поправање грешки и рефакторирање на код, истакнувајќи ја разновидноста на апликациите во кои овие алатки за вештачка интелигенција можат да бидат од корист.



Графикон 6. Резултати од шестото прашање

Chart 6. Sixth question results

Целокупната точност на кодот генериран од овие алатки доби просечна оцена помеѓу 3 и 4 од мнозинството учесници (29 од 38). Ова укажува на генерално задоволителни перформанси, иако има простор за подобрување, особено затоа што прецизноста е од клучно значење за програмерите.

Графикон 7. Резултати од седмото прашање

Chart 7. Seventh question results

Графикон 8. Резултати од осмото прашање

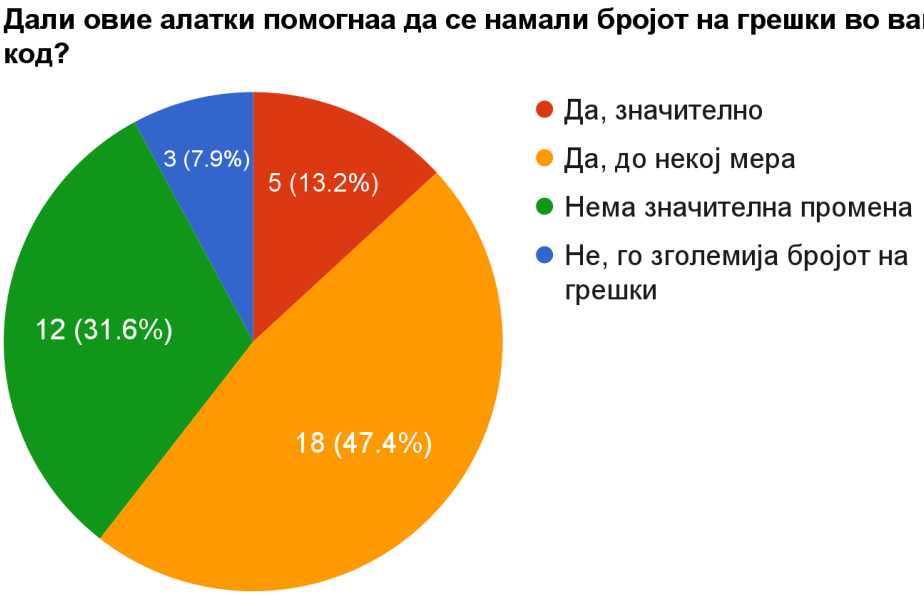
Chart 8. Eight question results

Во однос на задоволувањето на потребите при првиот обид, 12 учесници одговориле дека кодот често ги задоволува нивните потреби, додека 15 учесници рекоа дека понекогаш ги задоволува нивните потреби. Иако овие алатки се корисни, овие одговори сугерираат дека сè уште може да има значителен простор за подобрување на првично генерираниот код од овие алатки.

Графикон 9. Резултати од деветтото прашање

Chart 9. Ninth question results

Мнозинството учесници (26) се согласија дека генераторите на кодови базирани на GPT значително го намалуваат времето што го поминуваат на задачите за кодирање, покажувајќи клучна придобивка од овие алатки - временска ефикасност.



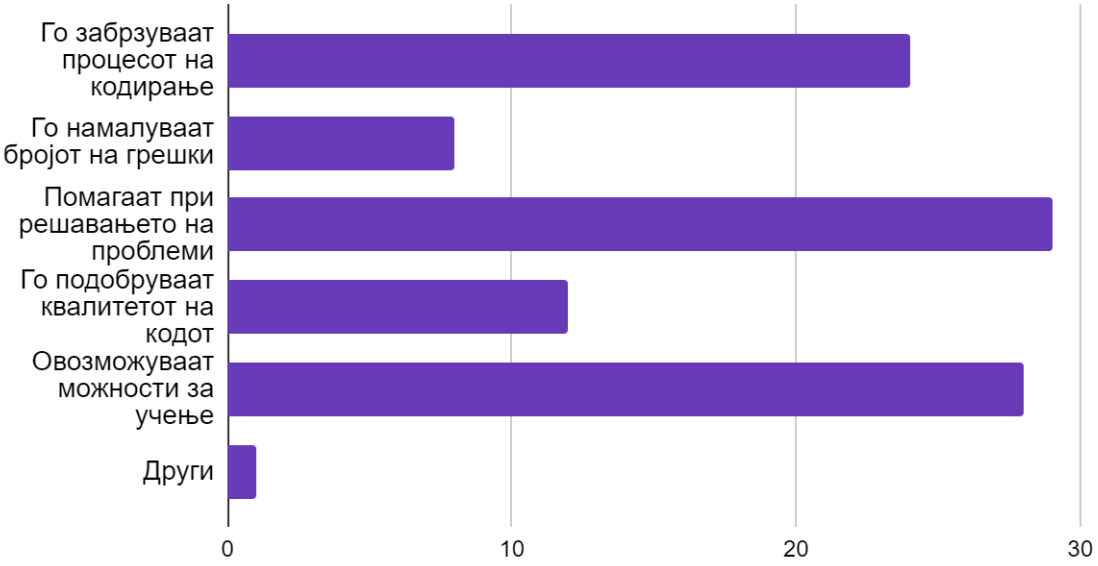
Графикон 10. Резултати од десеттото прашање

Chart 10. Tenth question results

Додека 23 учесници се согласија дека овие алатки помагаат да се намали бројот на грешки во кодирањето, само 5 пријавија значително намалување. Ова несовпаѓање може да ја одрази разликата во сложеноста на автоматизираните задачи или варијабилноста во перформансите на овие алатки.

Истражувањето ги идентификуваше клучните предности на генераторите на кодови базирани на GPT како што се: забрзување на процесот на кодирање, помагање во решавањето проблеми и обезбедување можности за учење. Во исто време, учесниците укажаа на предизвиците вклучувајќи ја точноста, зависноста од алатките и загриженоста за приватноста, кои обезбедуваат потенцијални области за развој и подобрување на овие алатки.

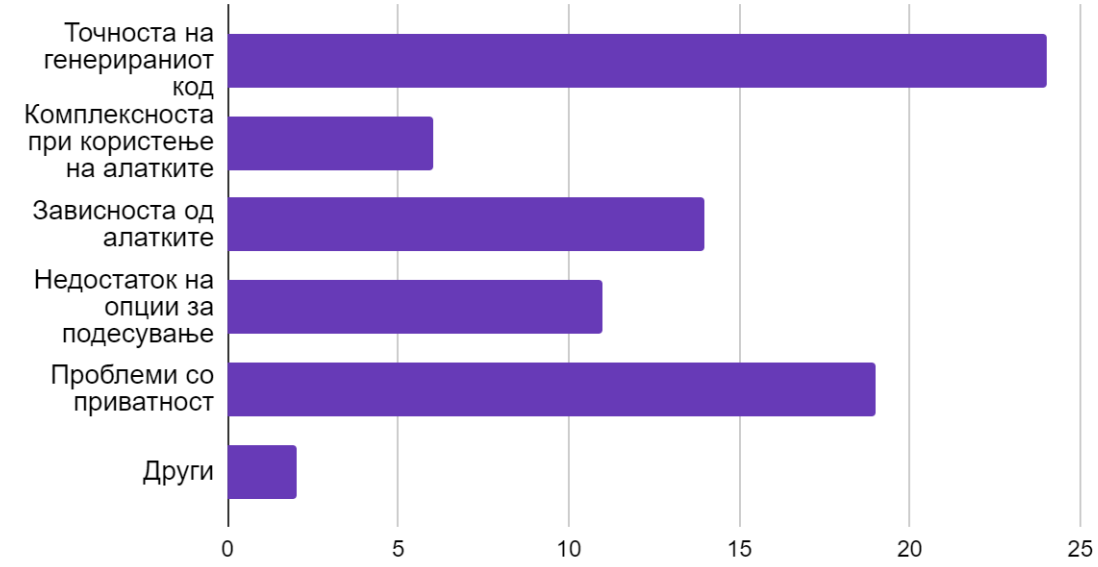
Кои мислите дека се главните предности од користењето на овие алатки?



Графикон 11. Резултати од единаесеттото прашање

Chart 11. Eleventh question results

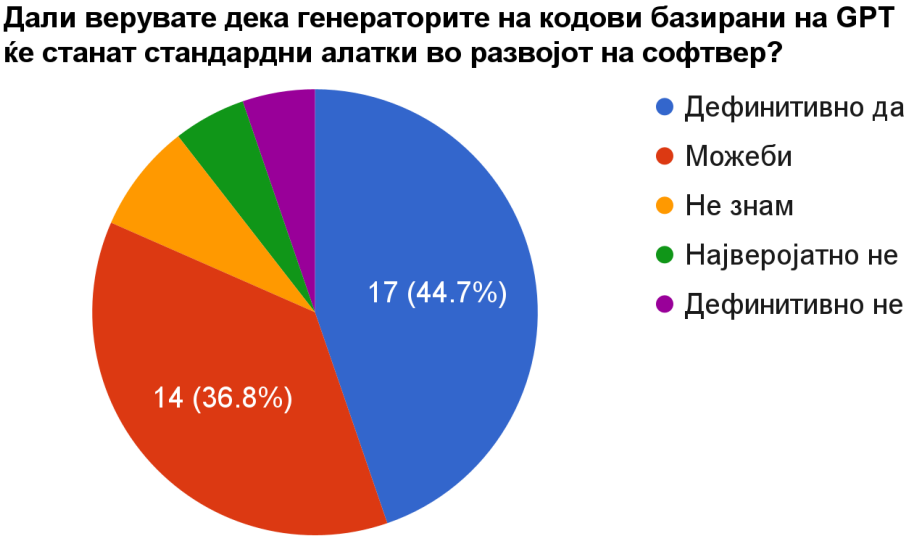
Што сметате дека се главни недостатоци или предизвици при користење на овие алатки?



Графикон 12. Резултати од дванаесеттото прашање

Chart 12. Twelveth question results

Силно верување во иднината на генераторите на код базирани на GPT беше изразено од 31 учесник во истражувањето. Понатаму, мнозинството би ги препорачало овие алатки на други развивачи на софтвер. И покрај идентификуваните предизвици, ова покажува позитивен изглед и широко прифаќање на овие алатки од страна на стручната фела во областа на развој на софтвер.



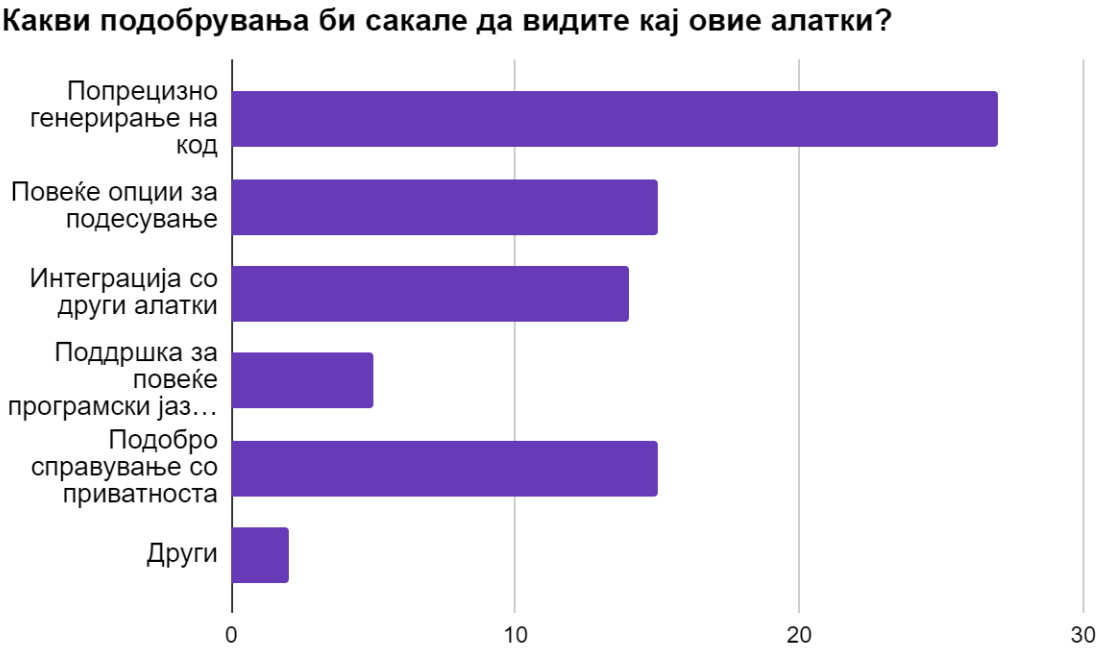
Графикон 13. Резултати од тринаесеттото прашање

Chart 13. Thirteenth question results

Графикон 14. Резултати од четиринаесеттото прашање

Chart 14. Fourteenth question results

Учесниците идентификуваа неколку области на потенцијално подобрување за генераторите на код базирани на GPT, вклучувајќи: попрецизно генерирање код, подобро справување со приватноста, повеќе опции за подесување и подобра интеграција со други алатки. Овие одговори можат да го водат идниот развој на овие алатки за вештачка интелигенција, со адресирање на нивните ограничувања и подобрување на нивните способности.



Графикон 15. Резултати од петнаесеттото прашање

Chart 15. Fifteenth question results

7. ДИСКУСИЈА И ЗАКЛУЧОЦИ

Во ова поглавје ќе ги разгледаме наодите од студијата во контекст на тековната литература и најдобрите практики во софтверското инженерство и програмирањето. Придобивките, недостатоците и импликациите од користењето на моделите базирани на GPT за задачи за софтверско инженерство ќе бидат истражени, заедно со проценка на јаките страни и ограничувањата на GPT моделите. Дополнително, ќе бидат разгледани силните страни и ограничувањата врз база на спроведената студија. Анализата ќе инкорпорира сознанија добиени и од систематските проценки за генерирање на код и од анкетата спроведена врз софтверските инженери.

7.1. Резиме на наодите од истражувањето

Резултатите од нашето истражување обезбедија непроценливи сознанија за можностите на GPT моделите. И GPT-3.5 и GPT-4 покажаа извонредни перформанси во низа задачи, нагласувајќи го нивниот потенцијал како моќни алатки за генерирање код. Нашата мануелна и автоматска техничка евалуација на кодот овозможи објективни проценки врз основа на функционалноста, точноста, читливоста, оптималноста и потенцијалните проблеми со кодот. Дополнително, спроведената анкета понуди перспектива од реалниот свет за тоа како програмерите ги користат, перцепираат и би сакале да ги подобрат овие алатки.

7.1.1. Функционалност и точност на кодот генериран од GPT

Во однос на функционалноста, и двата модели се покажаа добро во повеќето задачи. Тие можеа да генерираат код кој ги исполнуваше барањата на задачите, а разликите во функционалноста беа мали. Сепак, GPT-4 очекувано покажа подобри резултати кога станува збор за функционалноста на кодот кај комплексните задачи, што покажува дека робустноста на базата на податоци врз кои е трениран GPT-4 значително го подобрила неговото разбирање за посложените концепти во софтверско инженерство.

Во однос на точноста, двата модела произведоа код кој успешно се компајлира во повеќето случаи. Ова е клучен аспект, бидејќи некомпајлирањето код нема практична употреба. Фактот дека и двата модели можеа да генерираат код кој успешно се компајлира, ја покажува нивната компетентност во справувањето со основната синтакса и јазичните структури.

Резултатите од анкетата ја потврдуваат функционалноста и точноста на кодот генериран од GPT. Мнозинството од испитаниците се изјасниј дека кодот често или понекогаш ги задоволува нивните потреби уште од првиот обид што укажува на функционалноста и точноста на генерираниот код. Сепак, вреди да се напомене дека мнозинството од програмерите како еден од главните недостатоци го сметаат квалитетот на генерираниот код, што укажува дека сеуште постои простор за напредок во овој сегмент.

7.1.2. Читливост и оптималност на кодот генериран од GPT

Што се однесува до читливоста, GPT-4 покажа забележителни подобрувања во одредени задачи, особено на средно и напредно ниво. Читливоста на кодот е од суштинско значење за одржување и соработка меѓу програмерите. Фактот дека GPT-4 можеше да генерира почитлив код покажува дека тој може да произведе решенија кои се полесни за програмерите да ги разберат и да работат на нив.

Оптималноста е исто така важен аспект, бидејќи директно влијае на перформансите и ефикасноста на добиениот код. И двата модели покажаа конзистентни перформанси кај овој критериум, со минимални разлики меѓу нив. Ова сугерира дека и GPT-3.5 и GPT-4 се способни да генерираат код кој е разумно оптимизиран, со оглед на сложеноста на задачите.

7.1.3. Слабости и безбедносни пропусти во кодот генериран од GPT

Слабостите и безбедносните жаришта во кодот се области каде што двата модела покажаа различни степени на перформанси. Додека GPT-4 генерално се покажа како подобар во однос на слабостите детектирани во кодот, вреди да се напомене дека имаше и случаи каде генерираниот код од GPT-4 имаше безбедносни пропусти. Ова покажува дека, иако моделите GPT можат да генерираат функционален код, тие можеби не секогаш се придржуваат до најдобрите практики, што доведува до потенцијални предизвици за одржување и безбедност. Испитаниците во анкетата исто така покажаа доза на загриженост кога е во прашање безбедноста и приватноста при користење на овие алатки.

7.1.4. Кориснички перспективи за користењето на GPT за генерирање на код

Истражувањето за мислењата на софтверските инженери обезбеди вредни кориснички гледишта за предностите, предизвиците и идните перспективи на генераторите на кодови базирани на GPT. Учесниците ја идентификуваа брзината на процесот на кодирање, помошта во решавањето на проблемите и можностите за учење како значајни предности. Во меѓувреме, точноста, зависноста од алатките и прашањата за приватност беа идентификувани како

главни предизвици. И покрај овие предизвици, поголемиот дел од учесниците беа оптимисти за иднината на овие алатки и истите би ги препорачале на своите колеги. Тие, исто така, понудија вредни повратни информации за потенцијалните подобрувања кои би можеле да се имплементираат кај овие алатки, како што се подобрен квалитет на кодот, подобрено справување со приватноста и подобрена интеграција со други алатки.

Наодите од систематската евалуација и анкетата на корисниците го истакнуваат ветувачкиот потенцијал на GPT моделите во софтверското инженерство. Тие, исто така, посочуваат области каде овие алатки би можеле да се подобрат. Сеопфатниот поглед на можностите на овие алатки, како од технички, така и од корисничка перспектива, обезбедува вредни насоки за идните случувања на ова поле.

7.2. Предности и недостатоци на GPT генерираните кодови

Наодите од нашата студија, кои опфаќаат систематски проценки на кодот генериран од GPT и увиди од анкета на корисници, фрлаат светлина врз потенцијалните предности и предизвици за користење на алатките за генерирање код базирани на GPT во развојот на софтвер.

7.2.1. Предности на GPT генерираните кодови

- **Широк спектар на применливост:** Една значајна придобивка од генерирањето код заснована на GPT е неговата разновидност. И GPT-3.5 и GPT-4 ја покажаа својата способност да генерираат функционален код за различни типови на задачи, од основни до посложени. Ова сугерира дека моделите GPT можат да одговорат на различни предизвици во софтверското инженерство и се прилагодливи на низа развојни сценарија.
- **Зголемена продуктивност и временска ефикасност:** GPT моделите имаат потенцијал да ги забрзаат процесите на кодирање. Со брзо генерирање на фрагменти од код, овие модели им овозможуваат на програмерите да го насочат своето внимание кон размислувања за дизајн на повисоко ниво и посложени решавање проблеми. Оваа ефикасност се потврдува и од наодите од нашата анкета, при што

значителен број учесници изјавија дека алатките базирани на GPT го намалиле времето за кодирање. Сепак, важно е да се забележи дека ова воочено зголемување на продуктивноста треба да се мери со времето што може да биде потребно за преглед и потенцијална преработка на кодовите.

- **Поддршка за учење и решавање проблеми**: Испитаниците во анкетата исто така ја идентификуваа помошта за решавање проблеми и можностите за учење како главни предности на генераторите на кодови базирани на GPT. Овие алатки можат да им обезбедат на програмерите нови пристапи кон предизвиците за кодирање и да понудат увид во различни програмски парадигми.

7.2.2. Недостатоци на GPT генерираните кодови

- **Потенцијални слабости и ранливости на кодот:** Клучниот недостаток идентификуван и во евалуацијата на кодот и во одговорите на анкетата е склоноста на GPT моделите да генерираат код кој вклучува слабости или безбедносни пропусти. Иако моделите можат да генерираат функционален код, тие може да не се придржуваат строго до најдобрите практики, потенцијално воведувајќи ризици и технички долг во софтверските проекти.
- **Контекстуални ограничувања:** GPT моделите, иако се моќни, немаат длабоко разбирање на конкретниот контекст и ограничувањата на проектот. Тие може да обезбедат функционални решенија кои, сепак, не се усогласуваат со архитектурните одлуки или стандардите за кодирање на проектот. Ова ограничување ја нагласува континуираната потреба од човечко надгледување и интервенција за да се осигура соодветноста на генерираниот код за конкретни проекти.
- **Познавање и владеење со специфични програмски јазици**: Ефективноста на генерирањето код засновано на GPT може да зависи од програмскиот јазик што се користи за одредена задача. Се чини дека GPT моделите претпочитаат одредени јазици, како што се Python и JavaScript, потенцијално поради поголемата застапеност на овие јазици во нивните податоци за обука. Затоа, генерираниот код на овие јазици може да биде со подобар квалитет. Спротивно на тоа, перформансите на

GPT моделите кога имаат задача да генерираат код за помалку популарните јазици, тој би можел да биде со сомнителен квалитет.

Додека генераторите на кодови базирани на GPT обезбедуваат повеќекратни придобивки, тие доаѓаат со свој сет на предизвици. Затоа, правилната и ефективна употреба на овие алатки бара избалансирано разбирање на нивниот потенцијал и ограничувања. Важно е да се напомене дека овие алатки треба да се гледаат како помош за програмерите, наместо како замена, дополнувајќи ја човечката експертиза и расудување во процесот на развој на софтвер.

7.3. Извлечени заклучоци

Врз основа на нашата сеопфатна евалуација на моделите GPT-3.5 и GPT-4 и сознанијата собрани од анкетата на програмерите, можеме да извлечеме неколку значајни заклучоци.

Прво, GPT моделите, вклучувајќи го и поновиот GPT-4, покажуваат импресивни перформанси во генерирањето на код кој не само што е функционален туку покрива и широк опсег на сложеност. Овие модели се способни да создадат читлив и релативно оптимизиран код, иако има случаи кога генерираниот код може да има потенцијални слабости и безбедносни пропусти.

Второ, постои опипливо влијание на овие генератори на код базирани на вештачка интелигенција врз процесот на кодирање. Тие можат да го забрзаат развојот, да обезбедат можности за учење и да понудат вредна поддршка за решавање проблеми. Сепак, генерираниот код можеби не секогаш е совршено усогласен со контекстите и стандардите специфични за проектот, што укажува дека човечкиот надзор останува суштински.

И на крај, согледаната ефективност на овие генератори на код е многу зависна од програмскиот јазик за кој станува збор. Моделите GPT имаат тенденција да генерираат код со подобар квалитет на одредени јазици како Python и JavaScript, веројатно поради нивната важност во податоците за обука на моделите.

Во суштина, генераторите на кодови базирани на GPT имаат значителен потенцијал за решавање задачи од софтверското инженерство. Тие се ефикасни алатки кои можат да им помогнат на програмерите во нивната секојдневна работа, но не се беспрекорни и сепак бараат остро човечко око за преглед и контрола на квалитетот.

7.4. Препораки за понатамошна работа

Резултатите од ова истражување отвораат неколку патишта за идна работа во областа. Еве неколку препораки:

- **Истражување на моделите на GPT со различни програмски јазици:** Со оглед на потенцијалната варијација на ефективноста на моделите на различни програмски јазици, идните студии би можеле да истражат колку добро функционираат моделите GPT со помалку популарни програмски јазици. Ова може да доведе до сознанија за проширување на можностите на овие модели или откривање на ограничувања во нивната јазична разновидност.
- **Решавање на безбедносните грижи:** Нашата студија откри дека кодот генериран од GPT може да содржи потенцијални безбедносни пропусти. Идните истражувања би можеле да се фокусираат на начините за ублажување на овие ранливости, можеби со интегрирање на најдобрите безбедносни практики во обуката на моделите со вештачка интелигенција или градење процеси за самоевалуација на генерираните кодови.
- **Истражување на етичките размислувања:** Како што вештачката интелигенција станува сè повеќе интегрирана во развојот на софтвер, може да се појават етички грижи. Идните истражувања би можеле да ги разгледаат овие потенцијални етички прашања, како што се ризиците од прекумерно потпирање на алатките за вештачка интелигенција или влијанијата врз пазарите на труд.
- **Интеграција со други алатки за развој на софтвер:** Идните студии би можеле да испитаат колку добро овие алатки за вештачка интелигенција се интегрираат со постоечките развојни средини и работни текови и како таквата интеграција би можела да се оптимизира за да се подобри продуктивноста на програмерите.

- **Долгорочна студија за влијание:** Може да се спроведе надолжна студија за да се разберат долгорочните влијанија од користењето генератори на код базирани на GPT врз резултатите од проектот, одржливоста на кодот и продуктивноста на развивачите.

Како заклучок, генераторите на кодови базирани на GPT претставуваат возбудлива област на истражување, со многу аспекти сè уште зрели за истражување и разбирање. Нивниот потенцијал и предизвици продолжуваат да се развиваат со напредокот во технологијата во сферата на вештачка интелигенција, ветувајќи богато поле за истражување во годините што доаѓаат.

8. КОРИСТЕНА ЛИТЕРАТУРА

1. Austin, T., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., ... & Sutton, C. (2021) Program synthesis with large language models arXiv preprint arXiv:2108.07732
2. Balog, M., Gaunt, A. L., Brockschmidt, M., Nowozin, S., & Tarlow, D. (2016) Deepcoder: Learning to write programs arXiv preprint arXiv:1611.01989
3. Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., ... & Zhang, Y. (2023) Sparks of artificial general intelligence: Early experiments with gpt-4 arXiv preprint arXiv:2303.12712
4. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020) Language models are few-shot learners Advances in neural information processing systems, 33, 1877-1901
5. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., ... & Zaremba, W. (2021) Evaluating large language models trained on code arXiv preprint arXiv:2107.03374
6. Dale, R. (2021) GPT-3: What’s it good for? Natural Language Engineering, 27(1), 113-118
7. Destefanis, G., Bartolucci, S., & Ortu, M. (2023) A Preliminary Analysis on the Code Generation Capabilities of GPT-3.5 and Bard AI Models for Java Functions arXiv preprint arXiv:2305.09402
8. Domı, E., Pérez, B., & Rubio, Á. L. (2012) A systematic review of code generation proposals from state machine specifications Information and Software Technology, 54(10), 1045-1066

9. ERRINGTON, J. A. H. (2003) Code generation in action
10. Floridi, L., & Chiriatti, M. (2020) GPT-3: Its nature, scope, limits, and consequences Minds and Machines, 30, 681-694
11. Khan, J. Y., & Uddin, G. (2022, October) Automatic code documentation generation using gpt-3 In Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (pp. 1-6)
12. Katz, D. M., Bommarito, M. J., Gao, S., & Arredondo, P. (2023) Gpt-4 passes the bar exam Available at SSRN 4389233
13. Khatchadourian, R., & Masuhara, H. (2017, May) Automated refactoring of legacy Java software to default methods In 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE) (pp. 82-93)
14. Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., ... & Vinyals, O. (2022) Competition-level code generation with alphacode Science, 378(6624), 1092-1097
15. Liu, J., Xia, C. S., Wang, Y., & Zhang, L. (2023) Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation arXiv preprint arXiv:2305.01210
16. Liu, X., Zheng, Y., Du, Z., Ding, M., Qian, Y., Yang, Z., & Tang, J. (2021) GPT understands, too arXiv preprint arXiv:2103.10385
17. MacNeil, S., Tran, A., Mogil, D., Bernstein, S., Ross, E., & Huang, Z. (2022, August) Generating diverse code explanations using the gpt-3 large language model In Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 2 (pp. 37-39)
18. Nori, H., King, N., McKinney, S. M., Carignan, D., & Horvitz, E. (2023) Capabilities of gpt-4 on medical challenge problems arXiv preprint arXiv:2303.13375
19. Poesia, G., Polozov, O., Le, V., Tiwari, A., Soares, G., Meek, C., & Gulwani, S. (2022) Synchromesh: Reliable code generation from pre-trained language models arXiv preprint arXiv:2201.11227
20. Puschel, M., Moura, J. M., Johnson, J. R., Padua, D., Veloso, M. M., Singer, B. W., ... & Rizzolo, N. (2005) SPIRAL: Code generation for DSP transforms Proceedings of the IEEE, 93(2), 232-275
21. Sharma, S., Anand, N., & G V, K. K. (2023) Stochastic Code Generation arXiv preprint arXiv:2304.08243

22. Siddiq, M. L., Majumder, S. H., Mim, M. R., Jajodia, S., & Santos, J. C. (2022, October) An Empirical Study of Code Smells in Transformer-based Code Generation Techniques In 2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM) (pp. 71-82) IEEE
23. Svyatkovskiy, A., Deng, S. K., Fu, S., & Sundaresan, N. (2020, November) Intellicode compose: Code generation using transformer In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 1433-1443)
24. Tsantalis, N., Mansouri, M., Eshkevari, L. M., Mazinanian, D., & Dig, D. (2018, May) Accurate and efficient refactoring detection in commit history In Proceedings of the 40th international conference on software engineering (pp. 483-494)
25. Trummer, I. (2022) CodexDB: Synthesizing code for query processing from natural language instructions using GPT-3 Codex Proceedings of the VLDB Endowment, 15(11), 2921-2928
26. Vaithilingam, P., Zhang, T., & Glassman, E L. (2022, April) Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models In Chi conference on human factors in computing systems extended abstract
27. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017) Attention is all you need. Advances in neural information processing systems, 30.