

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Sofiia Hapon 232950IABB

Dmitriy Zakharenkov 230653IABB

Tehisintellekti koodi genereerimise võimekuse hindamine arendajatele: ChatGPT vs. DeepSeek

Bakalaureusetöö

Juhendaja: Rivo Lemmik
Doktorikraad

Tallinn 2025

Autorideklaratsioon

Kinnitame, et oleme koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Sofiia Hapon, Dmitriy Zakharenkov

20.05.2025

Annotatsioon

Käesoleva lõputöö eesmärk on hinnata ChatGPT ja DeepSeek'i koodi genereerimise kvaliteeti. Samuti võrrelda neid kahte tehisintellekti mudelit omavahel nende arhitektuuri, tööviiside jne osas. Töö keskendub ka funktsionaalsuse, loetavuse, efektiivsuse ja kasutusmugavuse mõõtmisele erinevate ülesannete põhjal.

Lõputöö raames viidi läbi praktiline eksperiment, kus mõlemale mudelile anti lahendada viis programmeerimisülesannet, mida hinnati viie päeva jooksul, et mõõta nende vastavalt teatud eelnevalt valitud meetoditele, nii et hinnata mudelite stabiilsust ja järjepidevust. Lisaks viidi läbi eksperthinnang, kus viieliikmeline kogenud programmeerijate rühm hindas koodi kvaliteeti skaalal 1–5 eri aspektide kaupa.

Analüüsi tulemused näitavad, milline mudel osutus erinevates aspektides paremini toimivaks, tuues välja ka olukorrad, kus üks või teine mudel vajab parendamist. Töö annab selge ülevaate, millistes kontekstides sobib paremini ChatGPT ning millistes DeepSeek, pakkudes väärtuslikku sisendit nii arendajatele, õppejõududele kui ka teadlastele.

Lõputöö on kirjutatud eesti keeles keeles ning sisaldab teksti 126 leheküljel, 6 peatükki, 6 joonist, 15 tabelit.

Abstract

Evaluating AI Code Generation Capabilities for Developers: ChatGPT vs. DeepSeek

The aim of this thesis is to evaluate the quality of code generation in ChatGPT and DeepSeek. It also compares these two AI models with each other in terms of their architecture, working methods, etc. The thesis also focuses on measuring the functionality, readability, efficiency and ease of use based on different tasks.

As part of the thesis, a practical experiment was conducted, where both models were given five programming tasks to solve, which were evaluated over a five-day period to measure their performance according to certain pre-selected methods, in order to assess the stability and consistency of the models. In addition, an expert assessment was conducted, where a group of five experienced programmers rated the code quality on a scale of 1–5 across various aspects.

The results of the analysis show which model performed better in different aspects, highlighting also situations where one or the other model needs improvement. The work provides a clear picture of the contexts in which ChatGPT is better suited and DeepSeek in which it is better suited, providing valuable input for developers, academics and researchers alike.

The thesis is in Estonian language and contains 126 pages of text, 6 chapters, 6 figures, 15 tables.

Lühendite ja mõistete sõnastik

AI	<i>Artificial intelligence</i>
Codex	<i>OpenAI Codex</i>
GPT	<i>Generative Pre-Trained Transformer</i>
KV	<i>Key-Value Cache</i>
LLM	<i>Large Language Model</i>
LOC	<i>Lines of Code</i>
MLA	<i>Multi-head Latent Attention</i>
MB	<i>Megabyte</i>
MoE	<i>Mixture of Experts</i>
RL	<i>Reinforcement Learning</i>
RHLF	<i>Reinforcement Learning from Human Feedback</i>
RNN	<i>Recurrent Neural Network</i>
SFT	<i>Supervised Fine-Tuning</i>
SOLID	<i>Single responsibility, Open-closed, Liskov, Interface segregation, Dependency inversion</i>
UX	<i>User Experience</i>

Sisukord

1 Sissejuhatus	11
1.1 Probleem	11
1.2 Eesmärk	12
1.3 Töö struktuur	13
2 Kirjanduse ülevaade	14
2.1 Praegused suundumused tehisintellekti abil loodud koodide genereerimisel	14
2.2 ChatGPT ja DeepSeek arhitektuur ja funktsioonid	16
2.3 Uurimisvajadus	20
2.4 Promptide koostamine ja optimeerimine	22
2.5 Andmekaitse, privaatsus ja turvariskid mudelite kasutamisel	23
3 Metoodika	25
3.1 Uuringu üldine kontseptsioon	25
3.2 Analüüsi aluseks võtmine	26
3.3 Kriteeriumid genereeritud koodi hindamiseks	27
3.3.1 Funktsionaalsus	28
3.3.2 Loetavus ja struktuur	29
3.3.3 Tõhusus ja kompaktsus	31
3.3.4 Genereerimisaeg ja efektiivsus	31
4 Praktiline osa	34
4.1 Konsooli rakenduste loomine	34
4.1.1 Rakendus 1 - Elemendi eemaldamine massiivist	34
4.1.2 Rakendus 2 - Stringide korrutamine	35
4.1.3 Rakendus 3 - Täisarvude jagamine	36
4.1.4 Rakendus 4 - Neliku summa (4Sum)	37
4.2 Algoritmilise ülesanne lahendamine - Binaarpuu maksimaalne tee summa	38
4.3 OOP rakenduse arendamine - IBAN kalkulaator	39
4.4 Katseandmete kogumine	41
4.4.1 Promptide lauseehitus	42
4.4.2 Katse tulemused	43

4.4.3 Koodigeneratsioon mudelite järgi	44
5 Katsetulemuste hindamine ja analüüsi järeldused	45
5.1 Genereeritud koodi funktsionaalsuse analüüs	45
5.1.1 ChatGPT koodi funktsionaalsuse hindamine	45
5.1.2 DeepSeek koodi funktsionaalsuse hindamine	47
5.1.3 Kokkuvõttev võrdlus mudelite vahel.....	48
5.2 Genereeritud koodi loetavuse ja struktuuri analüüs	48
5.2.1 ChatGPT koodi loetavuse ja struktuuri hindamine	49
5.2.2 DeepSeek koodi loetavuse ja struktuuri hindamine	49
5.2.3 Kokkuvõttev võrdlus mudelite vahel.....	50
5.3 Genereeritud koodi tõhususe ja kompaktsuse analüüs	51
5.3.1 ChatGPT koodi tõhususe ja kompaktsuse hindamine	51
5.3.2 DeepSeek koodi tõhususe ja kompaktsuse hindamine	52
5.3.3 Kokkuvõttev võrdlus mudelite vahel.....	52
5.4 Genereeritud koodi genereerimisaja ja efektiivsuse analüüs.....	53
5.4.1 ChatGPT koodi genereerimisaja ja efektiivsuse hindamine.....	53
5.4.2 DeepSeek koodi genereerimisaja ja efektiivsuse hindamine.....	54
5.4.3 Kokkuvõttev võrdlus mudelite vahel.....	55
5.5 Eksperthinnang	55
5.6 OOP-lähenemise analüüs ja tulevikusuunad	57
5.7 Kogumuspõhine hinnang ja soovitused arendajatele.....	60
6 Kokkuvõte	62
6.1 Hinnang töö teostamise protsessile ja projekti logid	63
6.1.1 Sofiia Hapon ajalogide kokkuvõte	64
6.1.2 Dmitriy Zakharenkov'i ajalogide kokkuvõte	66
Kasutatud kirjandus	68
Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks	70
Lisa 2 - Ideaalne prompt ülesannele “Elemendi eemaldamine massiivist”	71
Lisa 3 - Ideaalne prompt ülesannele “Stringide korrutamine”	72
Lisa 4 - Ideaalne prompt ülesannele “Täisarvude jagamine”	73
Lisa 5 - Ideaalne prompt ülesannele “Neliku summa (4Sum)”	74
Lisa 6 - Ideaalne prompt ülesannele “Binaarpuu maksimaalne tee summa”	76
Lisa 7 - Üksikasjalik prompt ülesannele “IBAN kalkulaator”	77

Lisa 8 - Tavaline prompt ülesannele “IBAN kalkulaator”	81
Lisa 9 - ChatGPT ülesanne 1 - Elemendi eemaldamine massiivist, lahendus.....	83
Lisa 10 - ChatGPT ülesanne 2 - Stringide korrutamine, lahendus	83
Lisa 11 - ChatGPT ülesanne 3 - Täisarvude jagamine, lahendus	84
Lisa 12 - ChatGPT ülesanne 4 - Neliku saumma, lahendus	85
Lisa 13 - ChatGPT ülesanne 5 - Binaarpuu maksimaalne tee summa, lahendus	87
Lisa 14 - DeepSeek ülesanne 1 - Elemendi eemaldamine massiivist, lahendus.....	88
Lisa 15 - DeepSeek ülesanne 2 - Stringide korrutamine, lahendus	88
Lisa 16 - DeepSeek ülesanne 3 - Täisarvude jagamine, lahendus.....	89
Lisa 17 - DeepSeek ülesanne 4 - Neliku summa, lahendus.....	90
Lisa 18 - DeepSeek ülesanne 5 - Binaarpuu maksimaalne tee summa, lahendus	91
Lisa 19 - ChatGPT ülesanne 6 - IBAN kalkulaator, lahendus üksikasjaliku prompti abil	92
Lisa 20 - ChatGPT ülesanne 6 - IBAN kalkulaator, lahendus tavalise prompti abil.....	96
Lisa 21 - DeepSeek ülesanne 6 - IBAN kalkulaator, lahendus üksikasjaliku prompti abil	102
Lisa 22 - DeepSeek ülesanne 6 - IBAN kalkulaator, lahendus tavalise prompti abil...	106
Lisa 23 - Sofiia Hapon eneseanalüüs.....	109
Lisa 24 - Dmitriy Zakharenkov eneseanalüüs	111
Lisa 25 – Küsimustik eksperdirühmale analüüsimiseks.....	112

Jooniste loetelu

Joonis 1. Transformer - mudelarhitektuur.	17
Joonis 2. Lihtsustatud illustratsioon mitmepoolse latentse tähelepanu (MLA) kohta....	19
Joonis 3. DeepSeek-V2 arhitektuuri illustratsioon.	20
Joonis 4. Binaarpuu oodatud tulemus esimese testimise jooksul	38
Joonis 5. Binaarpuu oodatud tulemus teise testimise jooksul	39
Joonis 6. IBAN numbri formaadi näide.....	40

Tabelite loetelu

Tabel 1. ChatGPT ja DeepSeek võrdlus andmekaitse ja läbipaistvuse osas.	24
Tabel 2. Loetavuse ja struktuuri mõõtmine mudelite jaoks.....	30
Tabel 3. Efektiivsuse mõõtmine mudelite jaoks.....	32
Tabel 4. Mudelite katse tulemused eksperimendi jooksul.....	43
Tabel 5. ChatGPT O3 funktsionaalsuse tulemused eksperimendi käigus.	45
Tabel 6. DeepSeek R1 funktsionaalsuse tulemused eksperimendi käigus.	47
Tabel 7. ChatGPT O3 loetavuse ja struktuuri kriteeriumid eksperimendi käigus.....	49
Tabel 8. DeepSeek R1 loetavuse ja struktuuri kriteeriumid eksperimendi käigus.....	50
Tabel 9. ChatGPT O3 tõhususe ja kompaktsuse kriteeriumid eksperimendi käigus.....	51
Tabel 10. DeepSeek R1 tõhususe ja kompaktsuse kriteeriumid eksperimendi käigus...	52
Tabel 11. ChatGPT O3 koodi genereerimisaja ja efektiivsuse kriteeriumid eksperimendi käigus.....	53
Tabel 12. DeepSeek R1 koodi genereerimisaja ja efektiivsuse kriteeriumid eksperimendi käigus.....	54
Tabel 13. Eksperdi hinnang mudelite kohta.	56
Tabel 14. Sofii Hapon ajalooigid.	64
Tabel 15. Dmitriy Zakharenkov'i ajalooigid.	66

1 Sissejuhatus

Kaasaegset maailma on üha raskem ette kujutada ilma tehisintellekti ja selle pakutavate võimalusteta. Veel mõni aeg tagasi tundusid tehisintellektiga seotud tehnoloogiad meile ulmena. Tänapäeval on see aga vahend, mis võib mõjutada kõiki inimelu valdkondi, sealhulgas infotehnoloogiat. Tehisintellekti mudelite kasutamine kodeerimise, programmeerimise, testimise ja projekteerimise käigus on muutunud täiesti tavapäraseks jutuks, mis lihtsustab tööd ja kiirendab protsessi mitmekordselt. Seega on raske eitada tehisintellekti mõju tänapäeva maailmale.

Tehisintellektiga seotud tehnoloogiad ja selle koolitamine on aktiivses arengufaasis, pidevalt on uudiseid sellest, kuidas üks organisatsioon teise järel rakendab oma töös tehisintellekti või arendab oma mudelit teatud funktsioonide täitmiseks.

OpenAI ja DeepSeek'i tehisintellekti mudelid on selle turu lipulaevad, mistõttu valiti need meie uuringusse. Arvestades nende nõudlust ja aktiivset konkurentsi nende vahel, on vaja analüüsida iga mudeli võimeid ja selgitada välja nende positiivsed ja negatiivsed küljed. Need aitavad mõista programmeerijale sobiva tööriista valikut.

1.1 Probleem

Nagu eespool mainitud, kasutavad programmeerijad oma töös regulaarselt tehisintellekti. Nad vähendavad oluliselt tööks kuluvat aega ja võtavad paljudes olukordades arendajatelt suure osa koormusest maha. Edasimineku on muutunud nii kiireks, et regulaarselt ilmuvad meedias uudised, et üks või teine organisatsioon vähendab oma töötajaid, sest tehisintellekt tuleb juba praegu suure hulga ülesannetega toime. Olukord on muutunud tõeliselt laiaulatuslikuks ja koondamiste arv on juba loetud tuhandetes töötajates. Seni puudutab see peamiselt madala kvalifikatsiooniga töötajaid, „nooremate“ arendajate taset, kuid aeg-ajalt ka kõrgema kvalifikatsiooniga spetsialiste.

Ka nende kahe mudeli analüüs on huvitav, sest lisaks nende lipulaevade juhtpositsioonile areenil on nad tegelikult võimelised täitma samu ülesandeid ja nende kasutajakogemus

on peaaegu identne, kuid sisuliselt on tegemist kahe põhimõtteliselt erineva tootega. Peamised erinevused seisnevad selles, et mudelite treenimise viisid ja nende treenimise kulude väärtus on radikaalselt erinevad.

Tegelikult on lisaks nendele aspektidele, mis võivad olla väga intrigeerivad, oluline mõista, milline mudel sobib kõige paremini koodi käitlemiseks ja koodi genereerimiseks. Vaatamata jõudlustestidele, mida nende mudelite arendamisega tegelevad organisatsioonid ametlikult avaldavad. Kasutajakogemused on üsna vastuolulised, paljudes foorumites internetis ja väljaspool seda on arvustusi, mis põhjustavad teatud vaidlusi. Seetõttu on raske teha järeldusi iga mudeli kohta.

1.2 Eesmärk

Lõputöö eesmärk on hinnata ja analüüsida, milline AI on koodide genereerimiseks eelistatum, tuginedes ChatGPT ja DeepSeek tehisintellekti mudelite abil saadud tulemustele. Mitmete mõõdikute, nagu funktsionaalsus, loetavus, struktuur, koodi tõhusus, kompaktsus ning genereerimisaeg ja efektiivsus, põhjal tehakse järeldused. Uuritakse C# programmeerimisprobleemide lahendamise tagajärjel saadud kogemusi ja analüüsitakse saadud lahenduste edukuse määra.

See töö aitab mõista, kumb vaadeldud mudelist (ChatGPT O# või DeepSeek R1) on usaldusväärsem, tõhusam ja paremini sobivam igapäevaseks programmeerimiseks, võttes arvesse nii objektiivseid mõõdikuid kui ka subjektiivset eksperthinnangut.

Eesmärgi saavutamiseks on tööprotsess jagatud järgmisteks alametappideks:

- Katseülesannete ja koodinäidete valimine (nii lihtsate rakenduste kui algoritmiliste ja OOP probleemide põhjal);
- Ideaalse prompti koostamine;
- Koodigeneratsiooni läbiviimine ja testimine;
- Genereeritud koodi hindamine mõõdikute ja eksperthinnangute alusel;
- Saadud tulemuste võrdlemine ja analüüs;
- Tulemuste põhjal järelduste tegemine ja soovitude koostamine mudelite kasutamiseks arendajatele.

1.3 Töö struktuur

Käesolev töö koosneb kuuest peamisest peatükist. Peatükis “Kirjanduse ülevaade” antakse ülevaade praegustest teadusuuringutest ja suundumustest tehisintellekti koodi genereerimise valdkonnas, analüüsitakse ChatGPT O3 ja DeepSeek R1 arhitektuuri ja eetilisi aspekte ning määratakse kindlaks uurimisvajadused. Metoodika peatükis kirjeldatakse tehisintellekti mudelite hindamise kriteeriume, uurimiskontseptsiooni ja hüpoteesid. Praktilises osas luuakse konsoolirakendusi, lahendatakse algoritmilisi ja OOP-probleeme ning hinnatakse tulemusi ChatGPT O3 ja DeepSeek R1 mudelite põhjal. Analüüsi peatükis võrreldakse saadud tulemusi ja tehakse täielik analüüs tehisintellekti mudelite kohta. Sealhulgas „Eksperimenthinnang“ alampeatükis analüüsitakse genereeritud koodi kvaliteeti eksperthinnangute põhjal. Autorid hindavad saadud tulemusi, valitud metoodikat ja vahendeid, toovad välja mudelite kogemuspõhised eelised ja puudused, annavad arendajatele praktilisi soovitusi mudelite valimiseks ning jätavad kohta edasiarenduseks.

2 Kirjanduse ülevaade

Käesolevas peatükis antakse ülevaade olemasolevast teadus- ja erialakirjandusest, mis käsitleb tehisintellekti mudelite, eriti ChatGPT ja DeepSeek, kasutamist koodigeneratsioonis ja promptide inseneriteadust.

2.1 Praegused suundumused tehisintellekti abil loodud koodide genereerimisel

Käesoleval kümnendil on märkimisväärselt kasvanud tehisintellekti (AI) kasutamine arenduse automatiseerimiseks. Tehisintellekti abil automatiseeritud koodi genereerimine võib mitte ainult vähendada arendusaega, vaid ka parandada tarkvaratoodete kvaliteeti, likvideerides vigu, vähendades rutiinset tööd ja andes reaajas soovitusi. Praeguste tehnoloogiate analüüs näitab, et süvaõppe mudelite integreerimine, avab programmeerimisel uusi horisonte. [1]

Ajalooliselt tugines automaatne koodi genereerimine mallidele ja reeglipõhiste süsteemidele, millel oli mitmeid piiranguid mittestandardsete ja keeruliste ülesannete käsitlemisel. Kaasaegne lähenemine põhineb neuronivõrkude kasutamisel, mis on võimelised õppima suurtest lähtekoodiandmetest, mis võimaldab mudelitel tuvastada varjatud sõltuvusi ja programmeerimismustreid.

Üks silmapaistvamaid näiteid tehisintellekti rakendamisest koodi genereerimisel on transformatsioonimudelid. Neid mudeleid treenitakse suurte andmekogumite põhjal, mis hõlmavad nii loomulikku keelt kui ka programmeerimiskeeli. Üks näide on GPT-3, mis põhineb Transformer-arhitektuuril, ja selle spetsiaalne järg Codex, mis keskendub kodeerimisele. Selliste mudelite eeliste hulka kuulub võime mõista konteksti, kohaneda konkreetse arendaja stiiliga ja genereerida koodi praktiliselt igas programmeerimiskeeles.[2][3]

Hiljuti on aktiivselt uuritud lisaks transformer-õppe meetoditele ka hübriidseid lähenemisviise, mis kombineerivad neuronaalseid mudeleid tugevdava õppimisega (RL).

Sellised meetodid optimeerivad genereeritud koodi kvaliteeti iteratiivse täiustamise kaudu, kus mudel saab tagasisidet testide, koodi täitmise ja jõudluse hindamise põhjal. Selline lähenemisviis aitab kaasa korrektsemate ja optimeeritumate lahenduste leidmisele koodi genereerimisel.

Tarkvarasüntees on suund, kus tehisintellekt ei genereeri mitte ainult valmis koodi, vaid püüab ka mõista ülesande spetsifikatsioone, muutes need programmiks. See lähenemisviis nõuab lähteandmete põhjalikku analüüsi ja loogiliste järelduste integreerimist, mis tagab, et genereeritud kood vastab täpsemalt algsele ülesandele. Selliseid tehnikaid uuritakse aktiivselt akadeemilistes ringkondades ja need kinnitavad tehisintellekti potentsiaali formaalsete spetsifikatsioonide muutmisel täidetavaks koodiks.[4]

Tehisintellekti integreerimisest arendusprotsessidesse saadav kasu on arendusaja vähenemine: tehisintellekt suudab automaatselt genereerida lihtsat koodi, andes arendajatele võimaluse keskenduda keerulisematele ülesannetele. Samuti paraneb koodi kvaliteet. Suurte kvaliteetse koodi näidiste põhjal koolitatud mudelite kasutamine võib vähendada igasuguseid vigu. Tehisintellektipõhised tööriistad toetavad algajaid programmeerijaid, pakkudes reaajas näiteid ja juhiseid.

Vaatamata märkimisväärsele potentsiaalile seisab tehisintellektipõhine automaatne koodigenerimine silmitsi mitmete nii positiivse, kui ka negatiivse aspektidega.

Positiivsed:

1. Sümbolsete ja statistiliste lähenemisviiside integreerimine - see traditsiooniliste programmide analüüsimeetodite ja süvaõppe mudelite kombinatsioon võib anda sünergilise efekti, tagades genereeritud koodi tõlgendatavuse ja usaldusväärsuse.
2. Enesekorrigeerimine - tulevased mudelid suudavad tõenäoliselt mitte ainult koodi genereerida, vaid ka analüüsida seda vigade suhtes ja pakkuda automaatseid parandusi.
3. Rakendusvaldkondade laiendamine - tehisintellekti rakendamine koodi genereerimiseks võib laieneda mitte ainult standardsetele programmeerimiskeeltele, vaid ka spetsialiseeritud valdkondlikele keeltele, mida kasutatakse keerukate süsteemide modelleerimisel.

Negatiivsed:

1. Turvaküsimused: koodi genereerimine võib võimaldada haavatavusi, kui mudel ei võta arvesse tänapäevaseid infoturvanõudeid.
2. Kvaliteet: Automaatselt genereeritud kood vajab kohustuslikku testimist ja õigsuse valideerimist, eriti kriitiliste rakenduste puhul.
3. Eetilised ja õiguslikud küsimused: Arutelu toimub tehisintellekti abil genereeritud koodi autoriõiguse ja vastutuse üle võimalike vigade või vigade eest.

Kaasaegsed tehisintellekti tehnoloogiad muudavad radikaalselt tarkvaraarenduse lähenemisviise. Transformaatormodelite kasutamine võimaldab luua intelligentseid süsteeme, mis on võimelised automaatselt kvaliteetset koodi genereerima. Vaatamata olemasolevatele probleemidele, mis on peamiselt seotud turvalisusega, on selle valdkonna edasise arengu väljavaated tohutud. Selliste tehnoloogiate integreerimine aitab juba praegu kaasa arenduse tõhususe suurendamisele ja edasised uuringud selles valdkonnas lubavad revolutsioonilisi muutusi programmeerimistööstuses.

2.2 ChatGPT ja DeepSeek arhitektuur ja funktsioonid

Tehisintellekti tehnoloogia kiire arenguga pööratakse erilist tähelepanu mudelitele, mis ei suuda mitte ainult teksti genereerida, vaid ka keerukaid programmeerimis-, loogilise mõtlemise ja andmeanalüüsi ülesandeid täita. Kaks liidrit selles LLM-mudelite valdkonnas on ChatGPT firmalt OpenAI ja DeepSeek.

Large Language Model (LLM) on tehisintellekti programmi tüüp, mis suudab muu hulgas teksti ära tunda ja genereerida. LLM-i treenitakse tohutute andmekogumite põhjal. LLMid põhinevad masinõppel, täpsemalt teatud tüüpi närvivõrgul, mida nimetatakse transformaatormodeliks.

Lihtsamalt öeldes on LLM arvutiprogramm, millele on antud piisavalt näiteid, et ta suudaks ära tunda ja tõlgendada inimkeelt või muud liiki keerulisi andmeid. Paljud LLM-id on koolitatud internetist kogutud andmete - tuhandete või miljonite gigabaitide teksti - põhjal. Kuid näidete kvaliteet mõjutab seda, kui hästi LLM õpib loomulikku keelt, seega võivad LLMi programmeerijad kasutada hoolikamalt valitud andmekogumit.

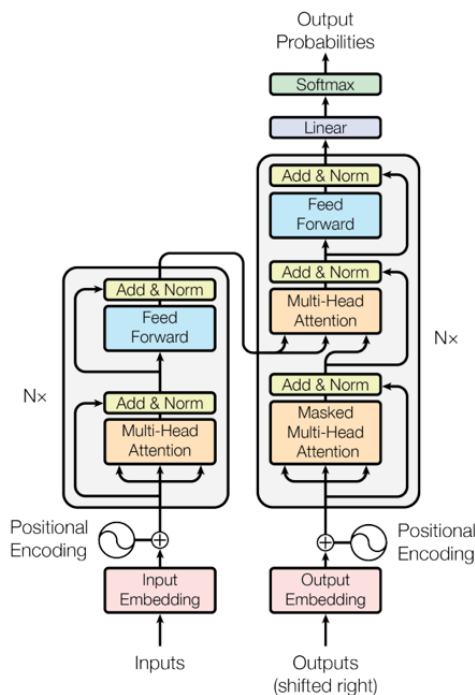
Nagu eespool mainitud LLM-id kasutavad ühte tüüpi masinõpet, mida nimetatakse süvaõppeks, et mõista, kuidas sümbolid, sõnad ja laused koos toimivad. Sügavõpe hõlmab struktureerimata andmete analüüsimist, mis võimaldab süvaõppe mudelil lõpuks ilma inimese sekkumiseta ära tunda erinevusi sisutükkide vahel.

LLMi treenitakse seejärel edasi kohandamise kaudu: neid häälestatakse või häälestatakse, et nad täidaksid konkreetset ülesannet, mida programmeerija soovib, näiteks tõlgendaksid küsimusi ja genereeriksid vastuseid või tõlkiksid teksti ühest keelest teise.

Edasi käsitletakse ChatGPT ja DeepSeek'i sisemise arhitektuuri, sisenditöötlustehnikate ja koodi genereerimise mehhanismide omadusi, et parandada jõudlust ja kohandada mudeleid keerukate programmeerimis- ja teadusanalüüsiülesannete jaoks.

ChatGPT arhitektuur:

ChatGPT põhineb Transformer-arhitektuuril (Joonis 1), mis on revolutsiooniline lähenemisviis, mis võimaldab mudelil töödelda kogu sisendjärjestust samaaegselt, kasutades tähelepanumehhanismi.[1] See lähenemisviis võimaldab paralleelset töötlemist, tõhusat märkide vaheliste sõltuvuste tuvastamist ja mudeli skaleeritavust parameetrite arvu kasvades.



Joonis 1. Transformer - mudelarhitektuur.

Mudel koosneb mitmest dekooderikihtidest ehk omab mitmetasandilist struktuuri, kus iga kiht sisaldab tähelepanumoodulit (enesehäälestus) ja positsioonireguleeritud täislüliplokki. Neid plokke järjestikku rakendades genereerib ChatGPT sisendteksti keerulise esituse, mis võimaldab luua sidusa ja kontekstiga seotud vastuseid.

Mudeli esialgne eeltreening toimub Interneti tekstide (artiklid, raamatud, koodid) ulatuslike andmekogumite abil, mis võimaldab mudelil õppida üldisi keelemustreid. Seejärel rakendatakse täiendavat treeningut õpetaja abil (supervised finetuning), mille järel optimeeritakse mudelit, kasutades inimese tagasisidest lähtuvat tugevdusõpet (Reinforcement Learning from Human Feedback, RLHF). See meetod võimaldab vastuseid kohandada vastavalt kasutaja ootustele, vähendades pahatahtliku või ebatäpse sisu tekkimise tõenäosust.[5]

ChatGPT kaasaegsed versioonid (sealhulgas GPT-4 ja GPT-4o) kasutavad lisamehhanisme, näiteks mõtteahelat, et lahendada keerulisi probleeme, mille puhul mudel „mõtleb“ vastuse üle, mis on eriti kasulik koodi genereerimiseks või matemaatiliseks arutluseks.

Sisendtekst jaotatakse märgideks, mis seejärel teisendatakse vektoriesindusteks. Need vektorid on aluseks edasiseks töötlemiseks võrgus. Tähelepanumehhanism võimaldab mudelil õigesti hinnata iga sümboli tähendust kogu jada kontekstis.

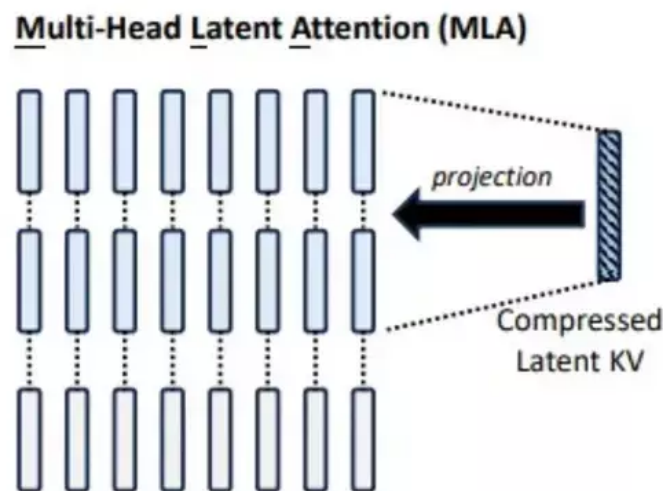
ChatGPT on edukalt rakendatud tarkvara koodi genereerimiseks, silumiseks ja kommenteerimiseks. Mudel on võimeline mõistma erinevate programmeerimiskeelte süntaktilisi ja semantilisi omadusi ning genereerima korrektseid koodifragmente. Tänu RLHFile ja arendajate tagasisidest kogutud lisaandmetele õpib ChatGPT andma soovitusi koodi optimeerimiseks ja parandamiseks.[2]

DeepSeek arhitektuur:

Erinevalt traditsioonilistest transformatsioonimudelitest kasutab DeepSeek Mixture-of-Experts arhitektuuri. Selle lähenemisviisi puhul koosneb mudel mitmest spetsialiseeritud „ekspertide“ alamvõrgustikust, millest igaüks vastutab sisendandmete teatavate aspektide töötlemise eest. Vastuse genereerimisel aktiveeritakse ainult osa parameetritest (nt DeepSeek-V3 puhul aktiveeritakse 671B parameetrist umbes 37B parameetrit konkreetse

sümboli jaoks). Selline lähenemisviis vähendab oluliselt arvutuskulu ja suurendab töötlemiskiirust, ilma et vastuse kvaliteet kannataks.[6]

DeepSeek võtab kasutusele uuendusliku Multi-head Latent Attention (MLA) mehhanismi, mis tihendab võtme- ja olulised vahemälud kompaktseteks latentseteks vektoriteks (Joonis 2). See võimaldab tõhusalt töödelda väga pikki kontekste (kuni 128K tokeneid), ilma et tähelepanu arvutamiseks vajaliku mälu hulk oluliselt suureneks.[6]



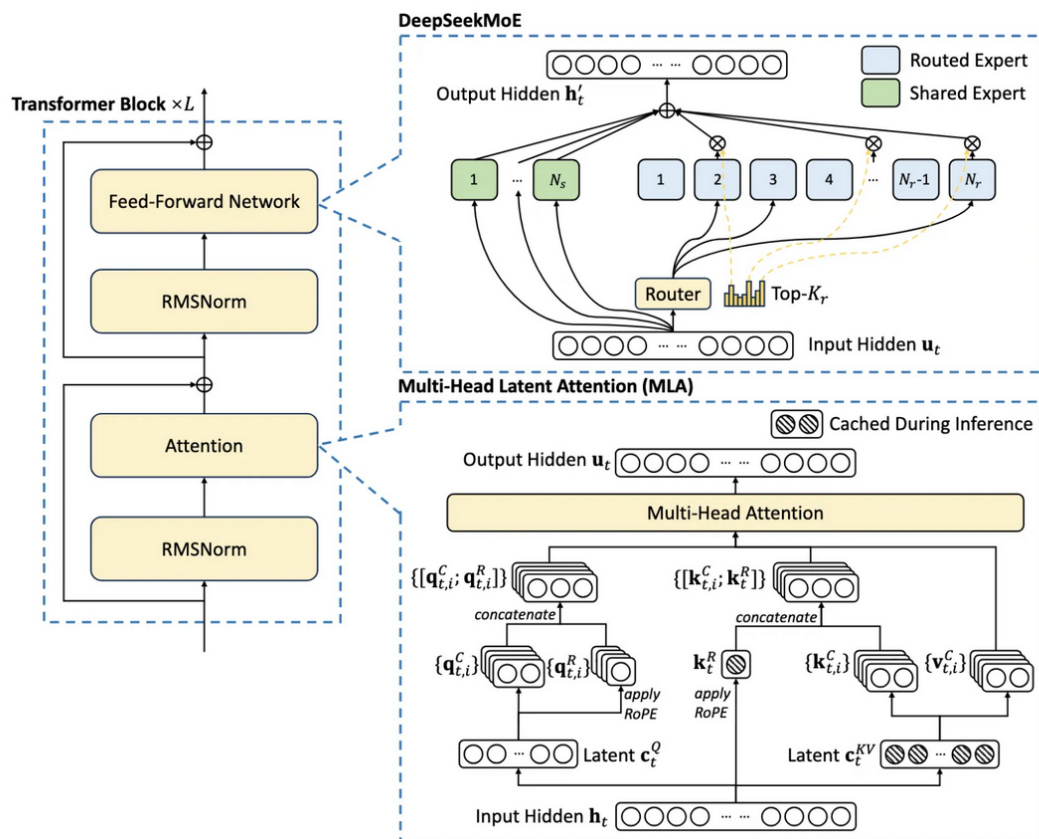
Joonis 2. Lihtsustatud illustratsioon mitmepoolse latentse tähelepanu (MLA) kohta.

Järgmise märgendi genereerimisel säilitatakse eelmised võtme- ja väärtusvektorid (KV), mis takistab ümberarvutamist ja vähendab latentsust järeldamise ajal. See meetod on eriti tõhus ülesannete puhul, mis nõuavad pikki jadasid ja keerulisi seoseid tekstielementide vahel.

DeepSeek on treenitud mitmekeelse korpuse, peamiselt ingliskeelsete ja hiina tekstide peal, keskendudes matemaatilistele ja programmeerimisandmetele. Tokeniseerimine toimub bait-kodeeritud BPE-mudelite abil, mis tagab kõrge kujutamise täpsuse ja võime töödelda suuri andmehulki.

Spetsiaalsed DeepSeek-mudelid, näiteks, DeepSeek Coder ja DeepSeek-Math jt., on optimeeritud tarkvarakoodi genereerimise ja analüüsi ülesannete jaoks (Joonis 3). Need on võimelised mitte ainult genereerima koodi erinevates programmeerimiskeeltes, vaid ka kontrollima selle korrektsust testplokkidega integreerimise kaudu. Täiendav treening

spetsialiseeritud andmekogumite (nt GitHubi kood, matemaatilised probleemid ja tehniline dokumentatsioon) abil võimaldab mudelil tõhusalt toime tulla ka mittetriviaalsete programmeerimisülesannetega.[11]



Joonis 3. DeepSeek-V2 arhitektuuri illustratsioon.

2.3 Uurimisvajadus

Vaatamata arvukatele teadusuuringutele, mille eesmärk on koodi genereerimine tehnilike närvivõrkude abil, on endiselt järgmised peamised lüngad:

Enamik publikatsioone keskendub üksikutele mudelitele - ChatGPT või Codex - või Transformeri põhimõtete kirjeldamisele.[1] Kõige värskemates töodes kirjeldatakse RLHF-i eeliseid ChatGPT puhul või MoE omadusi DeepSeekis, kuid puudub süstemaatiline võrdlus nende kahe lahenduse jõudluse, arvutusressursi tõhususe ja genereeritud koodi kvaliteedi kohta.[5][6][7]

On vähe teavet selle kohta, kuidas erinevused arhitektuuris ja õppimisstrateegias (nt ChatGPT tiheda muunduri arhitektuur vs. DeepSeeki Mixture-of-Experts arhitektuur)

mõjutavad koodi kvaliteeti ja usaldusväärsust reaalsetes tootmiskeskkondades. [6] Praegused uuringud keskenduvad sageli pigem mudelite üldistele omadustele kui nende rakendamisele arenduses.

Mudelite koolitamise ja kasutamise kulude erinevused on samuti veel halvasti mõistetavad. Mõned allikad näitavad, et DeepSeek saavutab võrreldavaid tulemusi oluliselt väiksemate kuludega (nagu DeepSeek-V3 puhul, mille arhitektuur on lahja). 2025. aasta alguses raputas IT-maailma ja isegi aktsiaturgu teave DeepSeeki loodud mudeli kohta. Mudel oli paljudes testides sama hea kui „ChatGPT“ ja mõnes testis isegi parem. Ja kõige üllatavam on see, et selle koolituskulud olid mitu korda väiksemad kui selle peamisel konkurendil. Sellel oli üllatav mõju, sest paljude tehisintellektiga seotud organisatsioonide hind langes ühe päevaga. Kõige silmatorkavam näide on NVIDIA. Selle tehnoloogiagigandi aktsia langes üle 17%, sest organisatsioon ehitab kalleid ja võimsaid kiipe OpenAI tehisintellekti töö jaoks.[10]

DeepSeek'i fenomen näitab, et tehisintellekti tehnoloogia arendamine muutub juba täna taskukohasemaks. Mis võib IT-turgu tõesti dramaatiliselt mõjutada.

ChatGPT ja DeepSeek'i võrdlemine arengu kontekstis on eriti väärtuslik järgmistel põhjustel:

Mõlemal mudelil on unikaalsed arhitektuurilised omadused, mis võivad olla automaatseks koodigenereerimiseks rohkem või vähem tõhusad. Võrdlev analüüs aitab kindlaks teha, kumb mudel käsitleb programmeerimisülesandeid paremini, pakub suuremat täpsust ja genereeritud programmide loogilist terviklikkust.

Uuring annab sügavama arusaama sellest, millised koolitus- ja optimeerimismeetodid (nt RLHF ChatGPT puhul või MoE ja MLA DeepSeek puhul) aitavad vähendada „hallutsemiste“ arvu ja suurendada koodi usaldusväärsust, samuti aitab see tuvastada valdkondi, mis vajavad edasist täiustamist.

Mudelite kulutõhususe (koolituse ja arvutusressursside kulu) võrdlemine on olulise tähtsusega ettevõtete jaoks, kes soovivad integreerida tehisintellekti lahendusi oma tootmisprotsessidesse. Sellise uuringu tulemused võivad anda väärtuslikku teavet optimaalsete tehnoloogiate valimiseks, mis võivad vähendada kulusid ja parandada tarkvaraarenduse tootlikkust.

Vajadus käesoleva uuringu järele on põhjendatud märkimisväärsete lünkade olemasoluga teoreetilises ja praktilises kirjanduses, mis käsitleb koodi genereerimist tehisintellekti mudelite abil.

2.4 Promptide koostamine ja optimeerimine

Tänapäeval on AI mudelite nagu ChatGPT tõhus kasutamine tihedalt seotud sellega, kui hästi on koostatud sisendküsimused ehk promptid. Promptide inseneriteadus (ingl *prompt engineering*) ei tähenda lihtsalt küsimuse esitamist, vaid süsteemset lähenemist, mille eesmärk on juhtida keelemudelit tootma maksimaalselt täpseid, asjakohaseid ja kvaliteetseid vastuseid.

OpenAI enda poolt avaldatud juhend "Prompt engineering best practices for ChatGPT" (2024) toob esile mitu olulist põhimõtet, mida järgides on võimalik koostada tõhusad ja tulemuslikud promptid.[9]

1. AI rolli määratlemine

Üks esimesi soovitusi on määrata selgelt AI roll. Näiteks võib prompti alustada lausega: *"You are a helpful assistant that explains code step by step."* See loob mudeli jaoks raamistiku, kuidas ta peaks "käituma" – kas olla õpetaja, arendaja, analüütik või näiteks tehniline toimetaja. Rolli määratlemine aitab keelemudelil genereerida vastuseid, mis vastavad paremini kasutaja ootustele ja sobivad konkreetse konteksti jaoks.

2. Konteksti pakkumine ja taustateave

OpenAI rõhutab, et ülesanne ei tohiks olla liiga üldine. Mida rohkem vajalikku konteksti promptis sisaldub (nt sisendandmed, töö eesmärk, kasutatav tehnoloogia vms), seda suurem on tõenäosus, et AI suudab genereerida täpse ja sobiva vastuse. Näiteks kui soovitakse, et AI kirjutaks funktsiooni koodi, tasub selgitada, milleks see funktsioon vajalik on, kuidas seda kasutatakse ja millised sisendid-väljundid on oodatud.

3. Vormindatud sisend

Hea praktikana soovitatakse esitada sisendi ja väljastuse struktuur vormindatult – kasutada loetelusid, koodiplokke ja eraldi lõike, et vältida segadust. See aitab AI-l paremini mõista, millised osad on käsud, millised näited ja millised soovitud tulemused.

4. Keeruliste ülesannete jagamine väiksemateks sammudeks

Keerulisemate probleemide korral on tõhus jagada prompt mitmeks järjestikuseks sammuks – näiteks kõigepealt paluda planeerida lahenduse loogika, seejärel kirjutada funktsioon ja lõpuks testida või selgitada. Selline järk-järguline lähenemine aitab kontrollida genereeritud sisu kvaliteeti ning võimaldab vajadusel kiiresti tuvastada ja parandada valesti mõistetud osi.

5. Soovitud väljundi täpsustamine

Lisaks rollile ja kontekstile peaks prompt sisaldama selget viidet sellele, mida kasutaja ootab – näiteks *“Please return a complete C# console application that follows the described logic”*. Kui see osa jääb promptis määratlemata, võib AI anda osalise või üldise vastuse.

Kõiki neid põhimõtteid järgides saab kasutaja oluliselt suurendada tõenäosust, et keelemudel, nagu ChatGPT, suudab genereerida mitte ainult tehniliselt korrektset, vaid ka sisuliselt asjakohast ja hästi struktureeritud koodi. Just seetõttu on meie uurimuses prompti koostamine ja selle optimeerimine oluline osa metoodikast, sest see võimaldab objektiivselt hinnata, kui palju mõjutab prompti kvaliteet lõppvastuse täpsust, loetavust ja töökindlust.

2.5 Andmekaitse, privaatsus ja turvariskid mudelite kasutamisel

ChatGPT-l on serverid Euroopa Liidus ja USAs. Ettevõtte järgib Euroopa GDPR-i ja California CCPA-d. Ettevõtte klientide puhul ei ole andmeid varundatud ja neid ei kasutata mudeli ümberõppeks. [13]

DeepSeek Cloud Serversil on serverid Hiinas ja selle suhtes kohaldatakse Hiina küberturvalisuse seadust. Teenuse tingimuste kohaselt edastatakse ja säilitatakse metaandmeid Hiina serverites. Selle suhtes kohaldatakse Hiina küberturvalisuse seadust ja määrust andmete ülekandmise kohta välismaale. Igasugune edastamine väljapoole Hiinat on võimalik ainult pärast valitsuse kontrollimist. See toob kaasa sellised riskid nagu sisu filtreerimine, kus poliitiliselt tundlikke teemasid käsitlevad päringud võidakse katkestada. Samuti on olemas andmete liikumine Hiina jurisdiktsiooni alla. [14]

Kui Euroopa ettevõtte käivitab DeepSeek'i, lahkuvad isikuandmed ametlikult GDPRi tsoonist. Lisaks saab DeepSeek'i käivitada offline-mudelil. Kuid just seetõttu on ründajal lihtsam varjatud tagaukse paigaldada. [14]

ChatGPT-l on OpenAI, mis parandab süsteemi tsentraalselt. Selle puuduseks on see, et kood on kättesaamatu ja üks suurem intsident pilves mõjutab korraga kõiki kliente.

Tabel 1. ChatGPT ja DeepSeek võrdlus andmekaitse ja läbipaistvuse osas.

Kriteerium	ChatGPT	DeepSeek
Andmete kogumine	Minimaalne andmete kogumine, korporatiivsed vestlused ei lähe koolitusele	Täielikud jututekstid ja tehnilised metaandmed
Ladustamine	Andmeid saab salvestada ainult Euroopas	Andmebaasid Hiinas, piiramatu säilitamisaeg
Vestluste kustutamine	On olemas ametlik „õigus olla unustatud“ mehhanism.	Ainult manuaalne eemaldamine vestlused, ei ole garantiid täielik eemaldamine
Läbipaistvus	Avalikud aruanded SOC 2, DPA	Dokumentatsioon on ebatäielik

Kui inimese jaoks on peamine, et ta järgiks lääne andmekaitseseadusi ja saaks valmis, regulaarselt uuendatud teenust, siis on ChatGPT kõige sobivam. Kui on vaja täielikku autonoomiat ja võimalust käivitada mudelit ilma internetita, on mõistlikum valida DeepSeek, kuid siis langeb mure turvalisuse pärast täielikult kasutajate meeskonnale.

3 Metoodika

Käesolevas peatükis tutvustatakse uuringu metoodikat, sealhulgas katseülesannete valikut, analüüsi aluseid ning kasutatud kriteeriume genereeritud koodi hindamiseks. Arutatakse, kuidas erinevaid ülesandeid lahendatakse ning kuidas uuritakse tehisintellekti mudelite võimekust koodigeneratsioonis, keskendudes nii funktsionaalsusele, loetavusele kui ka täpsusele.

3.1 Uuringu üldine kontseptsioon

Käesolev uuring on suunatud tehisintellekti mudelite, nagu ChatGPT ja DeepSeek, võimekuse hindamisele koodigeneratsiooni kontekstis. Kuna meie eesmärgiks on analüüsida just parimaid võimalikke mudeleid, siis valisime uuringu jaoks ChatGPT O3 ja DeepSeek R1 mudelid. Need mudelid on valitud, kuna nad esindavad valdkonna juhtivaid lahendusi ja mõlemad kasutavad edasijõudnud arutluskäik (*Advanced Reasoning*), et uurida, kuidas see mõjutab koodigeneratsiooni kvaliteeti.

Kuna uurimus keskendub AI mudelite võimekuse hindamisele, otsustasime teha eksperimentaalse lähenemisviisi, kus katsetame mõlemal mudelil "ideaalse prompti" kasutamist. Selle "ideaalse prompti" mõiste on see, et see on prompt, mille kaudu mõlemad mudelid peaksid suutma anda ühesuguse ja kvaliteedi poolest parima koodi vastuse. Selleks sisestame selle prompti nii ChatGPT-sse kui DeepSeeki ning testime neid 5 päeva jooksul. See test võimaldab meil hinnata, kas erinevad mudelid reageerivad samale ülesandele ühtemoodi ja kas nad toodavad ühtset, kvaliteetset koodi.

Uuringu protsessis me analüüsime, kuidas erinevad mudelid suudavad lahendada programmeerimisülesandeid ja millisel määral nende tulemused vastavad kvaliteedi ja täpsuse mõõdikutele, nagu koodide loetavus, struktuur, tõhusus, genereerimise aeg ja efektiivsus. Tulemuste testimiseks ja valideerimiseks konsooli ja algoritmilise ülesannete puhul kasutame LeetCode'i platvormi, kuna see pakub sisseehitatud testimisvõimalused. LeetCode võimaldab meil kiiresti hinnata, kuidas mudelid lahendavad ülesandeid, andes selged tulemused, mis põhinevad täpsuse ja täidetud nõuete järgi.

Lisaks algoritmilistele ülesannetele viime läbi eraldi eksperimendi ka keerukama objektorienteeritud programmeerimise (OOP) ülesandega. Selle eesmärk on uurida,

kuidas mudelid käituvad olukorras, kus lahendus eeldab mitme klassi koostööd ja süsteemset struktuuri. Kasutame spetsiaalselt koostatud ülesannet, mis esines aastal 2024 kontrolltöös ISA I (*Infosüsteemide arendamine I*) aine raames ja testime mudelite võimet genereerida loogiline, hästi struktureeritud OOP-lahendus.

Uuringu üheks aluseks olev hüpoteesiks on see, et lihtsamate üheklassiliste ülesannete puhul suudavad nii ChatGPT kui ka DeepSeek luua õigeid ja kvaliteetseid lahendusi, kasutades selgelt ja täpselt sõnastatud “ideaalset” prompti ja andes sarnase vastuse. Teisalt eeldame, et mida rohkem on süsteemis klassid ja mida rohkem on nendevahelisi seoseid, nagu OOP-probleemides, seda tõenäolisemalt muutuvad lahendused universaalsel testimisel ebastabiilseks või valeks.

Tulemuste põhjal tehakse järeldused ja koostatakse soovitusel parima mudeli valiku kohta arendajatele.

3.2 Analüüsi aluseks võtmine

Selles osas määratletakse katseülesannete valik ja rakendamise strateegiad, mis moodustavad analüüsi aluse, et hinnata tehisintellekti mudelite täpsust ja tõhusust koodigeneratsioonis.

Valisime analüüsiks erinevad ülesanded, mis katab mitmesuguseid programmeerimisoskusi ja -tehnikaid. Iga ülesande lahendamine nõuab erinevat taset teadmisi ja oskusi, mistõttu need on head näited programmeerimise arendamiseks. Kõik need ülesanded valiti teadlikult C# keeleks, kuna meie õpingute jooksul oleme pidevalt töötanud C# keeles, mis on samuti meie praktilise töö alus.

Konsooli rakenduste loomine

Selle kategooria raames valisime neli lihtsat ülesannet, mis pärinevad LeetCode’ist sarnase keerukusega (nt “Multiply Strings”, “Remove Elements” jne). [15-18] Need hõlmavad teemasid nagu stringide töötlemine, tsüklid ja lihtsad algoritmid. Sellised ülesanded annavad hea pildi AI-mudelite baasvõimekusest koodi genereerimisel ja optimeerimisel.

Algoritmilise probleemi lahendamine

Binaarpuu maksimaalse tee summa leidmine on algoritmiline ülesanne samuti võetud LeetCode'ist, mis keskendub puustruktuuride analüüsile ja rekursiivse loogika rakendamisele. [19] Lahendus nõuab oskust hinnata erinevaid teevõimalusi binaarses puus, kus optimaalne lahendus ei pruugi läbida juurt ning peab arvestama nii negatiivsete kui ka positiivsete väärtustega. Sellise ülesande lahendamine võimaldab hinnata tehisintellektide võimekust töötada keerukates rekursiivsetes struktuurides, hallata seisundit sügavuse lõikes ning optimeerida lahendusi mittetriviaalsetes puukonfiguratsioonides.

OOP rakenduse loomine

Otsustasime lisada eksperimendi juurde objektorienteeritud rakendust C# keeles, sest tahtsime testida, kuidas tehisintellekti mudelid käitlevad klassidevahelist suhtlust ja kui hästi nad mõistavad OOP põhimõtteid mitte ainult üksikute klasside tasandil. Valisime ülesande, mis on inspireeritud kontrolltööst, mida lahendatakse aines ISA I. See ülesanne nõuab mitme klassi koostööd, sõltuvuste haldamist ning OOP-põhimõtete. Lahenduse testimiseks koostame unit-testid, mis hindavad klasside omavahelist suhtlust, ülesanne korrektsust ning struktuuri terviklikkust.

3.3 Kriteeriumid genereeritud koodi hindamiseks

Analüüs põhineb järgmistel põhilistel hindamise kategooriatel:

- Funktsionaalsus - kas kood täidab ülesande nõuded ja töötab ootuspäraselt;
- Loetavus ja struktuur - kui selgelt ja loogiliselt on kood kirjutatud (sh nimetused, vormindus, jaotus funktsioonideks);
- Tõhusus ja kompaktsus - kas lahendus on optimeeritud, ilma liigse koodita, ning kasutab sobivaid algoritme ja andmestruktuure;
- Genereerimisaeg ja efektiivsus - kui kiiresti jõudis mudel töötava lahenduseni ning kui palju kasutaja sisendeid selleks vaja oli.

3.3.1 Funktsionaalsus

Funktsionaalsus tähendab see, et genereeritud programm peab täitma kogu määratud funktsionaalsuse korrektselt, sealhulgas ka piirjuhtumite korral. Lahendus peab vastama ülesande sisenditele ja väljunditele, nagu need on määratletud näidetes.

Funktsionaalsuse objektiivseks hindamiseks kasutatakse laialdaselt klassifikatsioonimudeleid iseloomustavaid mõõdikuid, mis aitavad kvantitatiivselt määrata, kui hästi genereeritud programm täidab ettenähtud ülesande. Sobivad mõõdikud meie juhul on Accuracy, Precision ja Recall, mis pärinevad masinõppe kvaliteedi hindamise printsiibist. [20]

1. Accuracy - näitab, kui suur osa kõigist ennustustest oli õiged. Sobib hästi funktsionaalsuse esmaseks hindamiseks:

$$Accuracy = \frac{Correct\ Predictions}{All\ Predictions}$$

, kus

- *Correct Predictions* - mitu korda mudel genereeris täiesti sobiva lahenduse (võrdne *True Positives* mõistele)
- *All Predictions* - mudelile katsete koguarv

Iga mudeli genereeritud vastust hinnatakse kas käsitsi või automaatselt, klassifitseerides selle kas korrektseks või mittekorrektseks.

Koodi tulemus loetakse korrektseks, kui ta vastab järgmistele tingimustele:

- programm käivitub ilma kompileerimis- või jooksutõrgeteta;
- lahendus täidab kõik ülesande tingimused, sealhulgas arvestab ka äärejuhtumeid;
- väljund vastab ootuspärasele tulemusele, nagu see on määratletud automattestides.

2. Precision - võimaldab hinnata, kui suur osa mudeli genereeritud väidetavalt korrektsetest lahendustest on tegelikult õiged.

See on eriti oluline olukordades, kus mudeli genereeritud kood näeb küll süntaktiliselt ja loogiliselt korrektne välja, kuid ei täida tegelikult kõiki ülesande tingimusi või annab vale väljundi.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

, kus

- *True Positives* - neid lahendusi, mis käivituvad veatult, täidavad kõik ülesande tingimused (sh äärejuhtumid) ja tagastavad õige tulemuse.
- *False Positives* - nende vastuste arv, kus mudel genereeris lahenduse, mis näib korrektne (nt kompileerub ja loogika tundub õige), kuid tegelikult ei täida ülesande nõudeid või annab vale tulemuse.

3. Recall - võimaldab hinnata, kui suur osa kõigist tegelikult korrektsetest lahendustest suutis mudel edukalt genereerida.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

, kus

- *True Positives* - lahendused, mis on veatud, täidavad kõik nõuded ja annavad õige tulemuse;
- *False Negative* - olukorrad, kus mudel ei suutnud genereerida korrektset lahendust, kuigi see oleks olnud võimalik (nt tagastas pooliku, vigase või täiesti vale koodi, mida ei saa parandada ilma käsitsi sekkumiseta).

3.3.2 Loetavus ja struktuur

Genereritud koodi loetavus ja struktuur on olulised tegurid, mis mõjutavad programmi hooldatavust, laiendatavust ja arusaadavust. Kuna valitud ülesanded katavad mitmesuguseid programmeerimisoskusi (sh stringide töötlus, graafialgoritmid ja objektorienteeritud konsoolirakendused C# keeles), tuleb hinnata, kas tehisintellekti genereeritud lahendused järgivad professionaalses arenduses kehtivaid stiililisi ja struktuurseid tavasid. [21]

Hindamisel pööratakse tähelepanu järgmistele aspektidele:

- selged ja tähenduslikud muutujate, klasside ja meetodite nimed, mis peegeldavad nende tegelikku rolli;
- koodi loogiline jaotus, sealhulgas meetodite eraldamine üksikvastutusega plokkideks ja vajadusel klasside kasutamine;
- vajalikud kommentaarid, mis aitavad mõista keerukamaid otsuseid või mitteilmselgeid seoseid (eriti konsoolirakenduste kasutajaliidese loogikas), vältides samas liigseid või ilmselgeid selgitusi.

Kvalitatiivse hinnangu objektiivsemaks mõõtmiseks kasutatakse järgmisi hindamiskriteeriume 1-5 Likert skaalal [22], kus

1. Kindlalt ei ole nõus
2. Ei ole nõus
3. Ei ole nõus ega ole eriarvamusel
4. Nõus
5. Kindlalt nõus

Tabel 2. Loetavuse ja struktuuri mõõtmise mudelite jaoks.

Kriteerium	Hindamisskala	Kirjeldus
Selged ja tähenduslikud nimed	1 - 5	Kas nimed on arusaadavad ja vastavad semantilisele sisule?
Loogiline ülesehitus ja meetodite eraldus	1 - 5	Kas kood on jagatud selgeteks funktsionaalseteks plokkideks?
Vorminduse ja stiili järjepidevus	1 - 5	Kas järgitakse C#-i koodistiili (nt dotnet format, StyleCop)?
Kommentaaride eesmärgipärasus	1 - 5	Kas kommentaarid selgitavad miks, mitte ainult mida kood teeb?

3.3.3 Tõhusus ja kompaktsus

Tõhusus ja kompaktsus viitavad sellele, kui hästi suudab genereeritud kood täita ülesande minimaalsete ressurssidega, säilitades samas selge struktuuri ja loetavuse. Hinnang antakse, võttes arvesse nii koodi ülesehituse kompaktsust kui ka algoritmilist ja süsteemset tõhusust. See parameeter kehtib ainult nende lahenduste puhul, mis on hinnatud true positive'iks, mis reaalselt töötavad ja lahendavad ülesande nõuetekohaselt.

Objektiivseks hindamiseks kasutatakse järgmisi mõõdikuid:

- Koodi maht (LOC) - reaalselt kirjutatud koodiridade arv; väiksem arv viitab üldiselt kompaktsemale lahendusele.

$$LOC = \text{Tegelikult kasutatud koodiridade arv}$$

- Ajaline keerukus (Time Complexity) - hinnatakse teoreetilise keerukuse alusel ($O(1)$, $O(n)$, $O(n^2)$, jne).
- Mälu kasutus (MB) - mõõdetakse, kui palju RAM-i kasutab programm täitmisel.

$$M = \max \text{mälu kasutus (MB)}$$

Selline hindamisviis võimaldab kvantitatiivselt mõõta, kui tõhusalt ja säästlikult AI-mudelid suudavad lahendada programmeerimisülesandeid.

3.3.4 Genereerimisaeg ja efektiivsus

Üks olulisemaid mõõdikuid tehisintellektimudeli praktilise kasutatavuse hindamisel on see, kui kiiresti suudab mudel luua töötava ning korrektse lahenduse. Mida kiiremini saadakse toimiv lahendus, seda tõhusam ja kasutajasõbralikum on mudel reaalsetes töövoogudes.

Töövoo tõhususe hindamiseks kasutatakse järgmine näitaja:

- Efficiency - arvestab, kui kiiresti mudel suudab genereerida töötava lahenduse, võttes arvesse lahenduste tulemusi. Mida suurem väärtus, seda efektiivsem on mudel praktilises kasutuses.

$$Efficiency = \frac{\sum C_i}{\sum t_i}$$

, kus

C_i - kas ülesanne i lahendati edukalt (1 = jah, 0 = ei);

t_i – aeg sekundites, mis kulus iga koodi genereerimiseks;

Tulemused hinnatakse järgmiselt:

Tabel 3. Efektiivsuse mõõtmise mudelite jaoks.

Efektiivsuse väärtus	Kirjeldus
> 0.1	väga hea - palju õigeid vastuseid kiiresti
$0.05 - 0.1$	hea - aktsepteeritav, mõistlik tulemus
< 0.05	nõrk - vähe õigeid tulemusi, aeglane või ebatäpne

Kui väärtus rohkem kui 0.1 tähendab, et mudel suudab anda õige lahenduse kiiresti, on see märgiks väga heast efektiivsusest. Näiteks kui vastus saabub 1 sekundiga ja on korrektne:

$$Efficiency = \frac{1}{1} = 1$$

või kui vastus saabub 5 sekundiga:

$$Efficiency = \frac{1}{5} = 0.2$$

Mõlemad ületavad piiri 0.1 ja on seetõttu “väga head”. Sellised tulemused sobivad hästi töövoogu, kus on oluline kiire ja täpne vastus.

Kategoorias näitajatega 0.05 - 0.1 jäävad tulemused, kus mudel annab vastuse mõistliku aja jooksul, kuid mitte koheselt. Tulemus on aktsepteeritav ja praktilises mõttes kasutatav. Näiteks 10 sekunditega:

$$Efficiency = \frac{1}{10} = 0.1$$

või 15 sekunditega:

$$Efficiency = \frac{1}{15} = 0.066$$

Kategoorias näitajatega < 0.05 kuuluvad olukorrad, kus kas suur osa lahendustest olid valed, või kulus lahendustele liiga kaua aega, mudel ebatäpne ja vajab mitmeid katseid. Näiteks kui õige vastus saadakse alles 30 sekundiga:

$$Efficiency = \frac{1}{30} = 0.033$$

Seetõttu otsustasimegi jagada kõik mõõdetud efektiivsuse väärtused kolme selgesse kategooriasse - väga hea, hea ja nõrk - ning anda igale mudeli tulemusele vastav hinnang. Selline jaotus võimaldab meil ühtlustada tulemuste tõlgendamist ning pakkuda struktureeritud ülevaadet mudelite praktilisest sobivusest erinevates olukordades.

4 Praktiline osa

Praktilises osas, nagu eespool mainitakse, luuakse lihtsad rakendused ja lahendatakse nii algoritmilist ja OOP ülesandeid, kasutades ChatGPT O3 ja DeepSeek R1 mudeleid. Lisaks kogutakse katseandmed, et analüüsida ja võrrelda iga mudeli sooritust. Tulemusi kasutatakse mudelite sobivuse hindamiseks igapäevaseks programmeerimiskasutuseks.

Iga ülesande lahendamine nõuab erinevat taset teadmisi ja oskusi, mistõttu need on head näited programmeerimise arendamiseks. Kõik need ülesanded valiti teadlikult C# keeleks, kuna meie õpingute jooksul oleme pidevalt töötanud C# keeles, mis on samuti meie töö alus.

4.1 Konsooli rakenduste loomine

Selles peatükis keskendume lihtsate rakenduste arendamisele LeetCode'st, mis on mõeldud tehisintellekti mudelite testimiseks ja nende võimekuse hindamiseks.

4.1.1 Rakendus 1 - Elemendi eemaldamine massiivist

Eesmärk:

Kirjutada C# programm, mis võtab sisendiks täisarvude massiivi `nums` ja täisarvu `val`. Programm peab kohapeal (in-place) eemaldama kõik elemendid, mis on võrdsed väärtusega `val`. Pärast töötlemist peab massiivi `nums` esimesed `k` elementi sisaldama üksnes neid arve, mis ei võrdu `val`. Ülejäänud elemendid ja massiivi kogusuurus pole olulised. Funktsioon peab tagastama arvu `k` - ehk mitu elementi ei võrdu väärtusega `val`.

Programmi käitumine:

- Funktsiooni sisendiks on massiiv `nums` ja täisarv `val`.
- Programm eemaldab kõik valiga võrdsed elemendid ilma lisamassiive kasutamata.
- Pärast eemaldamist:
 - Esimesed `k` positsiooni massiivis sisaldavad kõiki ülejäänud arve (järjekord pole oluline).
 - Ülejäänud positsioonide sisu on ebaoluline.

- Funktsioon tagastab k.

Näide töövoost:

- Sisend: nums = [3, 2, 2, 3], val = 3

Väljund: 2, nums = [2, 2, _, _]

- Sisend: nums = [0, 1, 2, 2, 3, 0, 4, 2], val = 2

Väljund: 5, nums = [0, 1, 4, 0, 3, _, _, _]

Lisatingimused:

- $0 \leq \text{nums.length} \leq 100$
- $0 \leq \text{nums}[i] \leq 50$
- $0 \leq \text{val} \leq 100$
- Eemaldamine peab toimuma in-place – lisamälu, mille suurus sõltub nums pikkusest, ei tohi kasutada.
- Lahenduse korrektsust kontrollib kohandatud kohtunik, mis:
- Kutsub sinu funktsiooni: `k = removeElement(nums, val)`.
- Veendub, et k võrdub eeldatava tulemuse pikkusega.
- Sorteerib massiivi esimesed k elementi ja võrdleb neid eeldatava väljundiga.

4.1.2 Rakendus 2 - Stringide korrutamine

Eesmärk:

Kirjutada C# programm, mis korrutab kaks mitte-negatiivset täisarvu, mis on esitatud stringidena (num1 ja num2). Tulemuseks peab samuti olema string, mis kujutab korrutise väärtust.

NB! Sisendeid ei tohi teisendada otse arvudeks (`int.Parse`, `Convert.ToInt32 vms`) ega kasutada sisseehitatud suurte arvude teeke (nt `BigInteger`).

Näide töövoost:

- Sisend: num1 = "2", num2 = "3"

Väljund: "6"

- Sisend: num1 = "123", num2 = "456"

Väljund: "56088"

Piirangud:

- $1 \leq \text{num1.Length}, \text{num2.Length} \leq 200$
- num1 ja num2 sisaldavad ainult numbreid 0–9.
- Kumbki string ei sisalda juhtivaid nulle, välja arvatud erandina number "0" ise.

4.1.3 Rakendus 3 - Täisarvude jagamine

Eesmärk:

Kirjutada C# programm, mis jagab kaks täisarvu - dividend ja divisor - ilma kasutamata korrutamist, jagamist ega modulo-operaatorit. Tulemuseks peab olema täisarvuline jagatise väärtus, mis on trunkreeritud nulli suunas (ehk ilma murdosata). Ülesannes on vaja testida oskust lahendada probleeme piiratud tingimustes, kasutades madalama taseme loogikat nagu tsüklid, bitinihkeid ja loogilisi võtteid.

Näide töövoost:

- Sisend: dividend = 10, divisor = 3

Väljund: 3

Selgitus: $10 / 3 = 3.333\dots$, mis trunc'itakse 3-ks

- Sisend: dividend = 7, divisor = -3

Väljund: -2

Selgitus: $7 / -3 = -2.333\dots$, mis trunc'itakse -2-ks

Piirangud:

- $-2^{31} \leq \text{dividend}, \text{divisor} \leq 2^{31} - 1$ (ehk 32-bitiste allkiri arvude vahemik)
- divisor $\neq 0$ (jagamine nulliga on keelatud)

- Kui tulemus on suurem kui $2^{31} - 1$, tagasta $2^{31} - 1$
- Kui tulemus on väiksem kui -2^{31} , tagasta -2^{31}

4.1.4 Rakendus 4 - Neliku summa (4Sum)

Eesmärk:

Kirjutada C# programm, mis leiab kõik unikaalsed nelikute kombinatsioonid massiivist `nums`, mille elementide summa võrdub etteantud väärtusega `target`. Igas leitud nelikus `[nums[a], nums[b], nums[c], nums[d]]` peavad kõik indeksid olema erinevad ($a \neq b \neq c \neq d$). Lahenduses tuleb vältida duplikaate — st üks ja sama nelik ei tohi korduda erinevates järjekordades.

Ülesanne testib võimet kasutada sortimist, mitmetasandilisi tsükleid ja kahte osutit (two pointers) efektiivse lahenduse loomiseks.

Näide töövoost:

- Sisend: `nums = [1, 0, -1, 0, -2, 2]`, `target = 0`
Väljund: `[[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]`
- Sisend: `nums = [2, 2, 2, 2, 2]`, `target = 8`
Väljund: `[[2, 2, 2, 2]]`

Piirangud:

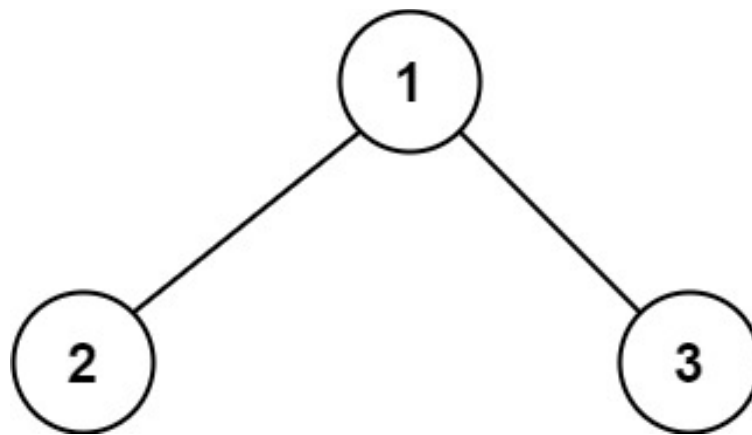
- $1 \leq \text{nums.Length} \leq 200$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- $-10^9 \leq \text{target} \leq 10^9$

4.2 Algoritmilise ülesanne lahendamine - Binaarpuu maksimaalne tee summa

Eesmärk:

Kirjutada C# programm, mis leiab binaarse puu maksimaalse tee summa. Tee all mõeldakse järjestust tippudest, kus iga külgnevate tippude paar on omavahel ühendatud servaga. Tipp võib tee sees esineda ainult üks kord ning tee ei pea läbima juurt. Tee summa on kõigi selle tippude väärtuste summa.

Näide töövoost:

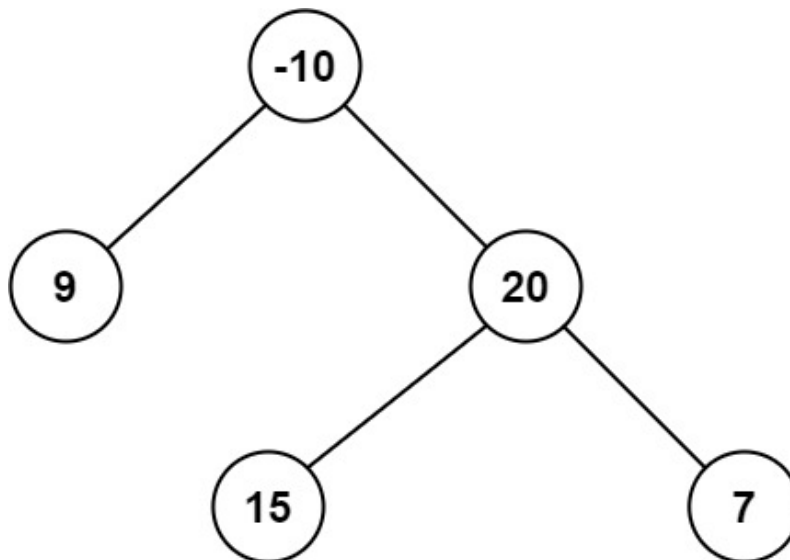


Joonis 4. Binaarpuu oodatud tulemus esimese testimise jooksul

1. Sisend: root = [1,2,3]

Väljund: 6

Seletus: optimaalne tee on 2 -> 1 -> 3, mille teesumma on $2 + 1 + 3 = 6$.



Joonis 5. Binaarpuu oodatud tulemus teise testimise jooksul

2. Sisend: root = [-10, 9, 20, null, null, 15, 7]

Väljund: 42

Seletus: optimaalne tee on 15 -> 20 -> 7, mille teesumma on $15 + 20 + 7 = 42$.

Piirangud:

- Puu sõlmede arv on vahemikus $[1, 3 * 10^4]$
- $-1000 \leq \text{Node.val} \leq 1000$

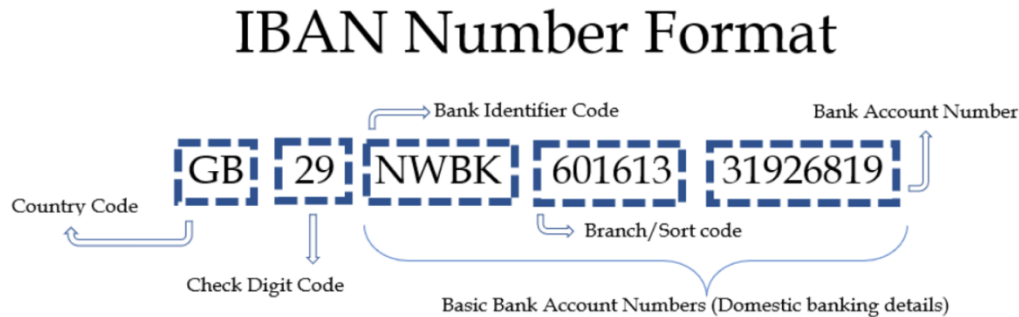
4.3 OOP rakenduse arendamine - IBAN kalkulaator

Eesmärk:

Kirjutada C# programm, mis suudab genereerida, valideerida ja töödelda rahvusvahelisi IBAN-kontosid. Lahendus peab toetama nii üldist IBAN-i struktuuri kui ka spetsiaalset Eesti IBAN-i loogikat. Lisaks peab programm suutma tuvastada IBAN-i päritoluriigi, salvestama viimaseid sisestusi ja toetama erinevaid tegevusi sõltuvalt sisendist. Pöörata

tähelepanu pannakse Clean Code'i põhimõtete järgimisele, vastutusjaotusele ja klasside koostöole.

Rahvusvaheline IBAN:



Joonis 6. IBAN numbri formaadi näide

Rakendada järgmised meetodid:

1. `string GenerateAccountNr()`
 - Genereerib juhusliku kontonumbri (täisarv).
 - Pikkus: 10 kuni 17 sümbolit.
 - Ei tohi alata nulliga.
 - Iga kutsung annab uue väärtuse.
 - Näide: `GenerateAccountNr()` → "1234567890"
2. `bool ValidateIBAN(string iban)`
 - Kontrollib, kas sisestatud IBAN on kehtiv. Tagastab `true` või `false`.
3. `string? FindCountry(string iban)`
 - Tuvastab riigi, võttes sisendiks IBAN-i ja kasutades `countryCodes.txt` faili Moodle'ist.
 - Näide: `FindCountry("DE29100100100987654321")` → "Germany"
4. `void PrintRecentIbans()`
 - Kuvab kuni 5 viimast valideeritud IBAN-i või nende riiki.

- Näide: "DE29100100100987654321"

Ainult Eesti IBAN

Eesti IBAN koosneb 20 märgist:

- Alguses: EE + 2-kohaline kontrollnumber + 2-kohaline panga kood.
- Seejärel: 14-kohaline sisemine kontonumber (vajadusel täidetakse nullidega).

Rakendada täiendavad meetodid:

5. bool ValidateEstonianIBAN(string iban)

- Kontrollib, kas IBAN vastab Eesti struktuurile.

6. string GenerateIBAN(string bankCodeAndCheck, string accountNumber)

- Tagastab täieliku Eesti IBAN-i antud sisendite põhjal.
- Näide: GenerateIBAN("9099", "123456789012") →
 "EE909900123456789012"

7. Kui kontonumber on alla 14 märki, peab GenerateIBAN(...) täiendama selle nullidega eespool.

- Minimaalne pikkus: 10
- Maksimaalne pikkus: 14

4.4 Katseandmete kogumine

Selles peatükis kogume kõik edasiseks analüüsiks vajalikud katseandmed. See hõlmab lahenduste genereerimiseks kasutatud ideaalseid prompte, metrilisi andmeid (näiteks edukate ja ebaõnnestunud katsete arv, genereerimisaeg, mälu suurus ja koodi pikkus) ning mõlema mudeli genereeritud koodinäidiseid. Need andmed võimaldavad meil võrrelda ChatGPT O3 ja DeepSeek'i R1 käitumist peamiste mõõdikute osas ja koostada objektiivne hinnang nende lahenduste kvaliteedile.

4.4.1 Promptide lauseehitus

Kuna meie uurimistöö põhineb eksperimenteerimisel ja erinevate lähenemisviiside rakendamisel, oleme kindlaks määranud “ideaalse prompti” optimaalse struktuuri, mis on vajalik selleks, et saada tehisintellekti mudelitelt kõige täpsemaid ja kvaliteetsemaid vastuseid. Varem, teoreetilises peatükis 2.5 “Prom[tide loomine ja optimeerimine”, oleme juba arutanud promptide moodustamise üldpõhimõtteid ja suuniseid. Nüüd oleme liikunud edasi praktilise osa juurde, kus kirjeldame üksikasjalikult, kuidas sõnastada ülesandeid tehisintellekti mudelitele, et saavutada optimaalseid tulemusi.

Lisa 2 prompt on sõnastatud spetsiaalselt selleks, et testida mudeli võimet rakendada algoritm otse algupärase massiivi ilma lisamälu kasutamata. Nõuded ja piirangud on selgelt määratletud, aidates mudelil selgelt mõista probleemi lahendamise ootusi.

Lisa 3 promptis keskendutakse aritmeetiliste operatsioonide sooritamisele ilma sisseehitatud teisendusfunktsioonide ja spetsiaalsete raamatukogude kasutamiseta. Prompt eesmärk on testida, mil määral suudab mudel realiseerida korrutamist käsitsi, numbrite kaupa, täpselt etteantud tingimusi järgides.

Lisa 4 prompt keelab selgesõnaliselt tavaliste aritmeetiliste operaatorite kasutamise, nõudes probleemi lahendamist silmuste, bitioperatsioonide ja tingimuslike operatsioonide abil. See aitab hinnata, kuidas mudel saab hakkama ülesannete lahendamisel, kui olemasolevatele lahendusvahenditele on seatud ranged piirangud.

Lisa 5 prompti sõnastus nõuab konkreetsete lähenemisviiside kasutamist (sorteerimine ja kahe osuti tehnika) ning sätestab selgesõnaliselt vajaduse vältida tulemuste dubleerimist. Selle prompt'i eesmärk on testida, kui hästi suudab mudel rakendada algoritmilisi meetodeid, et lahendada tõhusalt ja korrektselt keerulisi piirangutega probleeme.

Lisa 6 prompt on sõnastatud spetsiaalselt selleks, et testida mudeli võimet rakendada tõhusalt rekursiivseid algoritme ja käidelda puust andmestruktuure. Selgesõnaliselt on öeldud, et tee ei pea minema läbi juurest, ja lisatud on visuaalsed näited puude kohta, et aidata mudelil täpselt mõista ootusi probleemi lahendamiseks. Samuti on selgesõnaliselt kirjeldatud algoritmiline lähenemine (postorder traversal with global maximum), mis võimaldab mudelil kohe kasutada õiget rakendamisstrateegiat.

Selleks, et saada õige ja oodatav tulemus OOP rakendusest (Lisa 7), pidime koostama prompti, mis kirjeldab kõiki tegevusi, loogikat ja tehnilisi piiranguid võimalikult täpselt - alates klasside ja meetodite nimedest kuni tagastusväärtuse tüüpide ja sisendparameetriteni. Selles etapis puutusime kokku huvitava probleemiga. Selgus, et ainult rangelt struktureeritud promptide kasutamisel suudavad mõlemad mudelid genereerida lahenduse, mis vastab esitatud nõuetele ja läbib testid. Väikseimgi päringu üldistamine - näiteks konkreetsete allkirjade eemaldamine - tõi kaasa märkimisväärsed erinevusi mudelite väljundkoodis, mis muutis praeguse testikomplekti kasutuskõlbmatuks. Võrdluseks koostasime ka üldisema prompti ilma rangete tehniliste piiranguteta (Lisa 8). See prompt kirjeldab ülesannet üsna täielikult ja vastab oma sõnastuse poolest suures osas ülesande algsele tekstile, nagu see oli esitatud ISA I kontrolltöös.

4.4.2 Katse tulemused

Selles peatükis esitame üldised andmed, mis on saadud mudelite koodide genereerimise ja testimise käigus. Tabelis 4 allpool on esitatud koondtulemused iga ülesande kohta eraldi, koos vastava mudeli statistiliste näitajatega:

Tabel 4. Mudelite katse tulemused eksperimenti jooksul.

Ülesanne	Mudel	Katsete arv	TP	FP	FN	Aeg (sek)	M (MB)	LOC
4.1.1	ChatGPT	5	5	0	0	0.06	46.9	17
	DeepSeek	6	5	1	0	81.4	47	14
4.1.2	ChatGPT	5	5	0	0	2.12	42.3	41
	DeepSeek	11	5	1	5	368.4	44	36
4.1.3	ChatGPT	5	5	0	0	0.04	29.1	37
	DeepSeek	7	5	2	0	357.8	29.2	38
4.1.4	ChatGPT	6	5	1	0	0.08	48.7	58
	DeepSeek	6	5	1	0	175.2	48.9	39
4.2	ChatGPT	6	5	0	1	4.4	48.9	42
	DeepSeek	9	5	4	0	101.6	49.2	20

Nagu me oma eksperimendi käigus nägime, ei saa objektorienteeritud programmeerimisülesandeid hinnata samade kriteeriumide alusel kui lihtsaid algoritmilisi ülesandeid, sest nende vigade ja keerukuse tüüp on üsna erinev. Seetõttu valisime OOP-ülesande puhul mudelite võrdlemiseks teistsuguse lähenemise. Edasises analüüsis näitame, millised on erinevused range ja tavalise prompti lahenduste vahel [5.6], samuti seda, kuidas on koodistruktuur korraldatud ning kui läbimõeldud ja selge see on [4.4.3].

Kõik kogutud andmed Tabelis 4 moodustavad aluse edasiseks analüüsiks, kus hinnatakse mudelite funktsionaalsust, loetavust, tõhusust ning praktilist kasutatavust. OOP puhul samuti näitame, millised on erinevused range ja tavalise prompti lahenduste vahel, samuti seda, kuidas on koodistruktuur korraldatud ning kui läbimõeldud ja selge see on vastavates alampeatükkides [5.1 - 5.6].

4.4.3 Koodigeneratsioon mudelite järgi

Koodigeneratsioon OpenAI ja DeepSeeki LLM-mudelite abil on juba väga arusaadav ja ei tekita mingeid probleeme. Kõige olulisem, arvestades ülesande eripära, on koostada sobiv prompt selle lahendamiseks. Pärast selle koostamist toimub koodi genereerimine.[11][12][13]

Mõtlemise etapil analüüsib mudel päringu teksti, eraldab võtmeaspektid ja püüab tõlgendada ülesannet, arvestades konteksti. Kui mõtlemine on lõppenud, alustab tehisintellekt koodi genereerimist, järgides määratud kriteeriume ja nõudeid. Kood moodustatakse etappide kaupa, kontrollides süntaksi ja loogilise terviklikkuse üle. Lõpptulemusena saab kasutaja valmis koodi, mida saab kohandada ja kasutada konkreetses projektis.

Allpool on iga ülesande kohta üks koodinäide. Iga ülesande jaoks genereeriti üks lahendus päevas. Viie testimispäeva jooksul andsid ChatGPT O3 ja DeepSeek R1 mudelid ühe klassi ülesannete puhul enamasti sarnaseid lahendusi, kuna genereerimiseks kasutati täiuslike vihjete eelkompileeritud formaati. See tagas koodinäidete kõrge kvaliteedi ja järjepidevuse. ChatGPT O3 lahendused nii konsooli, kui ka algoritmilise rekendustele on näha Lisas 9 - 13 ja DeepSeek R1 lahendused Lisas 14 - 18.

OOP-ülesande eksperimendi raames genereerisime mõlema mudeli lahendused nii üksikasjaliku kui ka üldise prompti abil. Vastavad koodinäidised on esitatud Lisa 19 - 22,

kus saab visuaalselt näha erinevusi kahe lähenemisviisi arhitektuuris, stiilis ja rakendamises mõlema mudelite jaoks.

5 Katsetulemuste hindamine ja analüüsi järeldused

Selles peatükis analüüsime katsete tulemusi, lähtudes eelnevalt kirjeldatud hindamiskriteeriumitest. Lisaks esitame ka eksperdihinnangu, et toetada tulemuste usaldusväärsust välise vaatepunkti kaudu. Peatüki lõpus toome välja mudelite kogemuspõhised eelised ja puudused, mis põhinevad meie otsesel kokkupuutel mudelitega praktiliste ülesannete lahendamisel. Samuti lisame ruumi edasiarenduseks.

5.1 Genereeritud koodi funktsionaalsuse analüüs

Käesolevas osas analüüsitakse ChatGPT ja DeepSeek mudelite funktsionaalsus viie erineva ülesande puhul. Siin objektiivselt hinnatame genereerimise stabiilsust ja täpsust kasutades standardmeetreid Accuracy, Precision ja Recall. [3.3.1 peatükk]

5.1.1 ChatGPT koodi funktsionaalsuse hindamine

Tulemused on kokkuvõtlikult järgmised:

Tabel 5. ChatGPT O3 funktsionaalsuse tulemused eksperimendi käigus.

Ülesanne	1 päev	2 päev	3 päev	4 päev	5 päev	Vea tüüp
4.1.1	TP	TP	TP	TP	TP	-
4.1.2	TP	TP	TP	TP	TP	-
4.1.3	TP	TP	TP	TP	TP	-
4.1.4	TP	TP	FP, TP	TP	TP	CS0176
4.2	TP	TP	TP	FN, TP	TP	Internal server error

Meil on kokku 27 katset (5 ülesannet $\times$ 5 päeva). Neist:

- TP (True Positive) = 25
- FP (False Positive) = 1 (ülesanne 4.1.4)
- FN (False Negative) = 1 (ülesanne 4.2)

Lihtsad ülesanded (4.1.1–4.1.3), mis keskendusid massiivioperatsioonidele ja baasalgoritmidele, lahendati ChatGPT poolt täiesti stabiilselt - kõik 15 katset andsid korrektsed tulemused. See näitab kõrget algoritmilise mõistmise taset ka ilma lisateekideta.

Keerulisemad probleemid (4.1.4 ja 4.2) osutusid tundlikumaks struktuurivigadele. Ülesandes 4.1.4 ilmnis vale staatilise meetodi määratlus (CS0176) ning ülesandes 4.2 süsteemiviga (Internal Server Error), tõenäoliselt tingitud rekursiooni piiride ületamisest.

Mõlemad vead ei tulenenud loogika puudulikkusest, vaid keskkonnapiirangute eiramisest. Seega oli ChatGPT eriti usaldusväärne puhtalt algoritmiliste ülesannete lahendamisel, kuid keerukamate arhitektuuride puhul võib vaja minna lisakontrolli.

See on selgelt näha ka funktsionaalsetes mõõdikutes:

$$1) \text{ Accuracy} = \frac{\text{Correct Predictions}}{\text{All Predictions}} = \frac{25}{27} \approx 0.93 \approx 93\%$$

$$2) \text{ Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} = \frac{25}{25+1} \approx 0.96 \approx 96\%$$

$$3) \text{ Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} = \frac{25}{25+1} \approx 0.96 \approx 96\%$$

Sellised näitajad näitavad ChatGPT O3 suurt praktilist usaldusväärsust. Kasutades hästi koostatud prompte, võime eeldada, et:

- Mudel jõuab suure tõenäosusega esimesel katsel toimiva lahenduseni.
- Vigu esineb harva ja need on tavaliselt tehnilised (mitte loogilised).
- Isegi kui vigu esineb, suudab mudel uuesti genereerides anda õige tulemuse.

5.1.2 DeepSeek koodi funktsionaalsuse hindamine

Tulemused on kokkuvõtlikult järgmised:

Tabel 6. DeepSeek R1 funktsionaalsuse tulemused eksperimendi käigus.

Ülesanne	1 päev	2 päev	3 päev	4 päev	5 päev	Vea tüüp
4.1.1	TP	FP, TP	TP	TP	TP	Output Structure Error
4.1.2	FP, TP	FN × 4, TP	TP	FN, TP	TP	Output Structure Error, Internal Server Error
4.1.3	FP, TP	TP	TP	TP	FP, TP	Output Structure Error
4.1.4	TP	TP	TP	TP	FP, TP	Output Structure Error
4.2	TP	TP	TP	FP × 4, TP	TP	CS0436, CS0029

Kokku tehti 39 katset (5 ülesannet × 5 päeva, sealhulgas juhtumid, kus ühe päeva jooksul saadi mitu tulemust). Neist:

- TP (True Positive) = 25
- FP (False Positive) = 9 (kõik ülesanded)
- FN (False Negative) = 5 (4.1.2)

Ülesannete testimise käigus ilmnasid mitmed korduvad vead:

1. Output Structure Error - mudel genereeris loogiliselt korrektset koodi, kuid see oli esitatud ebamugavas või mitte kopeeritavaks vormis (nt mitte koodiplokis, ebavajalike kommentaaride või Markdowni moonutustega). Pidime käsitsi regeneratsiooni taotlema, mis vähendab mudeliga töötamise tõhusust;
2. Internal Server Error - genereeritud koodi käivitamisel tekkis süsteemi kokkuvarisemine. Põhjuseks oli koormatud server;
3. CS0436, CS0029 - DeepSeek püüdis üle kirjutada TreeNode tüüpi, mis on LeetCode'i süsteemis juba olemas. See põhjustas tüübi konflikti ja võimatuse

kompileerida. Loogika oli õige, kuid koodi ei saanud koostada ilma struktuuri muutmata.

See on selgelt näha ka funktsionaalsetes mõõdikutes:

$$1) \text{ Accuracy} = \frac{\text{Correct Predictions}}{\text{All Predictions}} = \frac{25}{39} \approx 0.64 \approx 64\%$$

$$2) \text{ Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} = \frac{25}{25+9} \approx 0.74 \approx 74\%$$

$$3) \text{ Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} = \frac{25}{25+5} \approx 0.83 \approx 83\%$$

5.1.3 Kokkuvõttev võrdlus mudelite vahel

Mudelite kogu funktsionaalsuse analüüs näitab ChatGPT O3 olulist eelist DeepSeek R1 ees. Põhimõõdikutes Accuracy, Precision ja Recall näitas ChatGPT O3 järjepidevalt kõrgeid tulemusi, saavutades keskmiselt 95 % funktsionaalse usaldusväärsuse. See tähendab, et mudel andis peaaegu alati esimesel korral õige ja toimiva koodi, võimaldades harva valesid või puuduvaid lahendusi. DeepSeek R1 näitas seevastu ainult 74% kumulatiivset funktsionaalsust, mis viitab ebastabiilsusele, tehniliste ja loogiliste vigade suurele osakaalule ning vajadusele käsitsi sekkuda ja koodi uuesti genereerida.

Seega osutus ChatGPT O3 funktsionaalsuse poolest palju usaldusväärsemaks ja prognoositavamaks vahendiks tarkvara probleemide lahendamiseks.

5.2 Genereeritud koodi loetavuse ja struktuuri analüüs

Hindamised loetavuse ja struktuuri kohta on tehtud käesoleva töö autorite poolt ja põhinevad nende isiklikel kogemustel, mis on saadud selle tehisintellekti mudeli pikaajalise kasutamise käigus. Eksperimenti käigus saadud andmed näitasid mõningaid erinevusi ChatGPT O3 ja DeepSeek R1 koodilahenduste struktuuris.

5.2.1 ChatGPT koodi loetavuse ja struktuuri hindamine

Tabelis 7 hinnatakse ChatGPT-koodi loetavust ja struktuuri nelja kriteeriumi alusel: nimede selgus, loogiline struktuur, stiili järgimine ja kommentaaride põhjendatus. Kõik näitajad said maksimaalse hinne (5), välja arvatud kommentaarid - 4, sest mõnes kohas oli osaline ülearune või ebapiisavad selgitused.

Tabel 7. ChatGPT O3 loetavuse ja struktuuri kriteeriumid eksperimendi käigus.

Kriteerium	Hindamisskala	Kirjeldus
Selged ja tähenduslikud nimed	5	Kas nimed on arusaadavad ja vastavad semantilisele sisule?
Loogiline ülesehitus ja meetodite eraldus	5	Kas kood on jagatud selgeteks funktsionaalseteks plokkideks?
Vorminduse ja stiili järjepidevus	5	Kas järgitakse C#-i koodistiili (nt dotnet format, StyleCop)?
Kommentaaride eesmärgipärasus	4	Kas kommentaarid selgitavad miks, mitte ainult mida kood teeb?

Tulemuste põhjal oli ChatGPT O3 keskmine hinne loetavuse ja koodistruktuuri eest 4.75 / 5. Mudel sai kõrged punktid kõigis kriteeriumides, eriti nimekasutuses, struktuuris ja vorminduses, mis viitab tugevale vastavusele headele programmeerimistavadele. Ainus väiksem kõrvalekalle oli seotud kommentaaride eesmärgipärasusega: kuigi kommentaare oli piisavalt, esines juhuseid, kus need lisati liigsete või iseenesestmõistetavate kohtade juurde, samas kui keerulisemad loogikaosad jäid mõnikord selgitamata. Kokkuvõttes näitab ChatGPT väga head taset loetava ja korrastatud koodi genereerimisel.

5.2.2 DeepSeek koodi loetavuse ja struktuuri hindamine

Tabelis 8 on esitatud DeepSeeki hindamise tulemused koodistruktuuri ja loetavuse kriteeriumide alusel.

Tabel 8. DeepSeek R1 loetavuse ja struktuuri kriteeriumid eksperimendi käigus.

Kriteerium	Hindamisskala	Kirjeldus
Selged ja tähenduslikud nimed	4	Kas nimed on arusaadavad ja vastavad semantilisele sisule?
Loogiline ülesehitus ja meetodite eraldus	5	Kas kood on jagatud selgeteks funktsionaalseteks plokkideks?
Vorminduse ja stiili järjepidevus	5	Kas järgitakse C#-i koodistiili (nt dotnet format, StyleCop)?
Kommentaaride eesmärgipärasus	3	Kas kommentaarid selgitavad miks, mitte ainult mida kood teeb?

Hindamiste põhjal võib järeldada, et DeepSeek esines selles komponendis hästi, kuid on teatud nüansse, mis põhjustasid mõne kriteeriumi puhul hinnangu languse. Näiteks punktis „Selged ja tähenduslikud nimed“ oli lõplikuks hindeks „4“, sest mõnes põlvkonnas oli küsimusi muutujate nimede kohta, sarnaselt Lisa 14-ga (Elemendi eemaldamine massiivist). See, kus muutujate nimedes kasutatakse numbreid, ei ole C# keeles probleemiks, kuid paljudes arendusmeeskondades on selliste nüansside kohta väljaütlemata reeglid. Ka kriteeriumis „Kommentaaride eesmärgipärasus“ sai hindeks „3“, sest mõnes lahenduses kommentaarid kas puudusid või olid ebapiisavalt seletavad. Tulemuste põhjal oli DeepSeek R1 mudeli keskmine hinne loetavuse ja koodistruktuuri eest 4.25 / 5.

5.2.3 Kokkuvõttev võrdlus mudelite vahel

Üldiselt oli mõlema mudeli loodud kood loetav ja arendajale arusaadav, kuid ChatGPT kood paistis silma paljude selgituste ja kommentaaride poolest, samas kui DeepSeeki kood oli kompaktsem. Mõlema mudeli hinned olid üsna sarnased: 4.75 ja 4.25. Struktuuri poolest järgisid mõlemad mudelid programmeerimisülesannete lahendamisel loogilist struktuuri (sisendi töötlemine, põhiloogika, väljund jne).

Siiski kippus ChatGPT oma hoolikamate vastuste tõttu järgima paremaid stilistilisi võtteid - näiteks lisas ChatGPT, nagu eelnevalt öeldud, kommentaare või valis koodis dokumenteeritud nimetusi, mida DeepSeek katsete põhjal otsustades alati ei teinud.

Seega on ChatGPT eelis loetavuse ja struktuuri osas, mis tähendab, et ChatGPT lahendusi on kolmandatel isikutel lihtsam hooldada ja kontrollida, samas kui DeepSeeki lahendused võivad nõuda rohkem tähelepanu ja aega, et neid mõista ja kontrollida.

5.3 Genereeritud koodi tõhususe ja kompaktsuse analüüs

Tõhususe ja kompaktsuse hindamine võimaldab meil mõista, kui ratsionaalselt ja minimaalsete ressurssidega keelemudel ülesandega toime tuleb, säilitades samal ajal arusaadava koodistruktuuri. Analüüsi on kaasatud ainult need lahendused, mis läbisid tõepositiivse tulemuse. Peamised mõõdikud on koodiridade arv (LOC), maksimaalne mälu kulu (MB) ja ajakomplekssus. Mõlemad mudelid näitasid iga ülesande puhul sama ajakomplekssust, mis muudab võrdluse teiste mõõdikute osas eriti ülevaatlikuks.

5.3.1 ChatGPT koodi tõhususe ja kompaktsuse hindamine

ChatGPT poolt genereeritud kood osutus reeglina pikemaks. Enamiku ülesannete puhul oli koodiridade arv märgatavalt suurem ChatGPT puhul, nt: 58 rida ülesandes 4.1.4 ja 42 rida ülesandes 4.2.

Hoolimata suuremast LOC-st jäi ChatGPT mälu kasutus mõõdukaks. Ülesannetes 4.1.2 ja 4.2 näitas mudel isegi veidi väiksemaid mäluväärtusi kui DeepSeek. See tähendab, et koodi suurus ei põhjutanud ülemäärast ressursikulu, vaid peegeldas pigem lahenduste struktuurilist terviklikkust.

Kõigi viie ülesande puhul näitasid nii ChatGPT kui ka DeepSeek mudelid identset ajakomplekssust, mis vastab iga ülesandetüübi eeldatavale tasemele.

Tabel 9. ChatGPT O3 tõhususe ja kompaktsuse kriteeriumid eksperimendi käigus.

Ülesanne	ChatGPT ajaline keerukus
4.1.1	$O(n)$
4.1.2	$O(n^2)$
4.1.3	$O(\log n)$
4.1.4	$O(n^3)$
4.2	$O(n)$

Seega näitab ChatGPT stabiilset tõhusust, mõõdukat ressursikulu ja lahenduste suurt loogilist täielikkust, kuigi selle arvelt suureneb koodi suurus.

5.3.2 DeepSeek koodi tõhususe ja kompaktsuse hindamine

DeepSeek lahendused olid kompaktsemad. Kõikide ülesannete puhul genereeris mudel vähemate ridadega koodi. Näiteks ülesandes 4.2 oli ainult 20 rida võrreldes ChatGPT 42 reaga. Selline käitumine võib viidata optimeerimise ja lühiduse soovile. Siiski tuleb märkida, et mitmel juhul saavutati selline lühidus tiheda struktuuri ja minimaalsete abielementide arvelt, mis potentsiaalselt vähendab koodi loetavust ja modulaarsust.

Mälukasutuse osas näitas DeepSeek tulemusi, mis olid lähedased ChatGPT-le, kuid mõnikord veidi suuremad - näiteks 49,2 MB võrreldes 48,9 MB-ga ülesandes 4.2.

Tabel 10. DeepSeek R1 tõhususe ja kompaktsuse kriteeriumid eksperimendi käigus.

Ülesanne	DeepSeek'i ajaline keerukus
4.1.1	$O(n)$
4.1.2	$O(n^2)$
4.1.3	$O(\log n)$
4.1.4	$O(n^3)$
4.2	$O(n)$

Üldiselt keskendub DeepSeeki lähenemisviis lühidusele ja lihtsustatud loogikale, mis võib olla tõhus olukordades, kus oluline on genereerimise kiirus ja minimaalne koodimaht, kuid mille analüüsimiseks või täiustamiseks on vaja rohkem vaeva näha.

5.3.3 Kokkuvõttev võrdlus mudelite vahel

Kokkuvõttes olid nii ChatGPT kui ka DeepSeek mudelid tõhususe ja kompaktsuse poolest head. Nad näitasid sama ajalist keerukust kõigis testitud ülesannetes, mis viitab algoritmilise loogika õigele mõistmisele ja rakendamisele. ChatGPT paistab silma üksikasjalikuma ja struktureerituma rakendamise poolest, samas kui DeepSeek tugineb lühidusele ja minimalismile.

Mõlemal lähenemisviisil on sõltuvalt kasutuseesmärgist omad eelised ning võib järeldada, et jõudluse ja ressursitõhususe poolest on mõlemad mudelid näidanud häid ja praktilisi tulemusi.

5.4 Genereeritud koodi genereerimisaja ja efektiivsuse analüüs

Reaalsetes arenduskeskkondades mõjutab koodi genereerimise kiirus otseselt töövoogude sujuvust, kasutatavust ja mudeli praktilist väärtust. Selles jaotises võrreldakse ChatGPT O3 ja DeepSeek R1 mudelite genereerimise kiirust ja tõhusust. Tõhususe hindamiseks kasutatav valem on Efficiency.

5.4.1 ChatGPT koodi genereerimisaja ja efektiivsuse hindamine

Tabelis 11 on esitatud iga ülesande tulemused 5 päeva jooksul, mida kasutati testitud mudelite Efficiency arvutamiseks. Igale katsele on määratud Ci skoor, mis näitab, kas genereeritud lahendus oli edukas (1 - edukas ehk True Positive; 0 - ebaõnnestunud ehk False Positive või False Negative). Samuti on näidatud aeg ti, mis kulus mudelil iga lahenduse genereerimiseks, sõltumata selle edukusest.

Mida suurem on saadud tõhususe väärtus, seda paremini ühendab mudel genereerimise kiiruse ja tulemuse täpsuse.

Tabel 11. ChatGPT O3 koodi genereerimisaja ja efektiivsuse kriteeriumid eksperimendi käigus.

Ülesanne	Ci (edukad katsed)	Kogu katsete arv	ti (keskmine aeg sekundites)
4.1.1	5	5	0.06
4.1.2	5	5	2.12
4.1.3	5	5	0.04
4.1.4	5	6	0.08
4.2	5	6	4.4

Nüüd arvutame kumulatiivse arvutuse, kasutades valemit, et leida ChatGPT O3 mudeli üldine efektiivsus.

$$Efficiency = \frac{\sum C_i}{\sum t_i} = \frac{True\ Positives\ (1 \times C_i)}{(Kogu\ katsete\ arv) \times t_i} = \frac{25}{27 \times 1.34} \approx 0.69 \approx 69\%$$

Seega on ChatGPT O3 mudeli kumulatiivne tõhusus antud ülesannete lahendamisel ligikaudu 0,69 ehk 69%. See tähendab, et mudel suutis lühikese aja jooksul genereerida suure hulga õigeid lahendusi, ühendades hästi nii täpsuse kui ka kiiruse. See tulemus näitab mudeli suurt praktilist rakendatavust ülesannete lahendamisel, kus oluline on nii usaldusväärsus kui ka vastamise kiirus.

5.4.2 DeepSeek koodi genereerimisaja ja efektiivsuse hindamine

Tabel 12. DeepSeek R1 koodi genereerimisaja ja efektiivsuse kriteeriumid eksperimendi käigus.

Ülesanne	Ci (edukad katsed)	Kogu katsete arv	ti (keskmine aeg sekundites)
4.1.1	5	6	81.4
4.1.2	5	11	368.4
4.1.3	5	7	357.8
4.1.4	5	6	175.2
4.2	5	9	101.6

Arvutame kumulatiivse arvutuse, et leida DeepSeek R1 mudeli üldine efektiivsus.

$$Efficiency = \frac{\sum C_i}{\sum t_i} = \frac{True\ Positives\ (1 \times C_i)}{(Kogu\ katsete\ arv) \times t_i} = \frac{25}{39 \times 216.88} \approx 0.003 \approx 3\%$$

DeepSeek R1 mudeli kumulatiivne efektiivsus viie ülesande põhjal oli ligikaudu 0.003 ehk 3%. Kuigi mudel suutis genereerida 25 edukat lahendust, oli selle saavutamiseks vaja kokku 39 katset, millest mitmed ebaõnnestusid. Samal ajal oli ka iga katse keskmine aeg väga pikk, ulatudes mitmes ülesandes sadade sekunditeni mis näitab halva efektiivsust..

Selline suur ajakulu vähendab oluliselt mudeli praktilist kasutatavust olukordades, kus tähtis on ka lahenduse kiirus. Kuigi õigeid vastuseid saadi, toimus see ebaühtlaselt ja suure ressursikuluga.

5.4.3 Kokkuvõttev võrdlus mudelite vahel

Kumulatiivsete tulemuste põhjal oli:

ChatGPT O3 efektiivsus: ≈ 0.69 ehk 69%

DeepSeek R1 efektiivsus: ≈ 0.003 ehk 3%

ChatGPT genereeris edukaid lahendusi kiiresti ja järjepidevalt, sageli vähem kui sekundiga. DeepSeekil seevastu võttis ühe lahenduse genereerimine palju kauem aega, sageli üle 100 sekundi, mis mõjutas oluliselt üldist tõhusust.

Kuigi mõlemad mudelid on võimelised genereerima õigeid lahendusi, oli ChatGPT oluliselt tõhusam - see andis õigeid lahendusi kiiremini ja kiiremini. Seega sobib ChatGPT paremini olukordadesse, kus on oluline nii täpsus kui ka reageerimise kiirus, samas kui DeepSeek ei pruugi olla tõhus praktilise kasutamiskiiruse seisukohast.

5.5 Eksperthinnang

Käesolevas dokumendis koostati eksperdirühm, mis koosnes 5 programmeerijast, kellel on kogemused ettevõtluskeskkonnas. Neile anti ülesanne võrrelda omavahel kahe erineva LLM-mudeli abil genereeritud programmeerimisprobleemide lahendusi. Eksperthindamisse ei kaasatud OOP-ülesannet, kuna selle analüüs toimub eraldi kvalitatiivsel alusel ja ei ole otseselt võrreldav teiste ülesannetega.

Uuringu objektiivsuse huvides ei jagatud rühmale tehisintellekti nime. Võrdluse eesmärk oli anda hindeid skaalal 1 kuni 5. Hindamiskriteeriumid olid järgmised. Esimene oli "Selged ja tähenduslikud nimed", eksperdid pidid aru saama, kas muutujate/meetodite nimed olid selged ja tähenduslikud. Teine näitaja oli "Loogiline ülesehitus ja meetodite eraldus", nad pidid hindama, kas kood on jaotatud selgeteks funktsionaalseteks plokkideks/funktsioonideks. Kolmas näitaja oli "Vorminduse ja stiili järjepidevus", viies näitaja oli "Lakoonilisus", mis oli vastutav üleliigse koodi ja korduste olemasolu eest, ning viimane näitaja oli "Kommentaaride eesmärgipärasus". Tabelis 13 on esitatud iga kriteeriumi keskmine hinne.

Tabel 13. Ekspertdihinnang mudelite kohta.

Ülesanne	Mudel	Selged ja tähenduslikud nimed (1–5)	Loogiline ülesehitus ja meetodite eraldus (1–5)	Vorminduse ja stiili järjepidevus (1–5)	Lakoonilisus (1–5)	Kommentaari de eesmärgipärasus (1–5)
4.1.1	ChatGPT	4,8	5	4,4	4,8	3
	DeepSeek	4,8	4,8	4,6	5	3
4.1.2	ChatGPT	4,2	4,8	5	4,8	3
	DeepSeek	4,4	4,6	4,8	4,4	3
4.1.3	ChatGPT	4,8	4,8	5	4,8	3,8
	DeepSeek	4,6	4,4	4,8	4,4	3
4.1.4	ChatGPT	4,8	4,4	4,6	4,2	4,2
	DeepSeek	4,8	4,6	4	4,2	2,8
4.2	ChatGPT	4,8	4,8	4,8	5	4
	DeepSeek	5	5	5	5	3,2

Hindamistulemused osutusid üsna huvitavaks. Kuigi hindamine viidi läbi individuaalselt ja ükski ekspert ei tundnud üksteist. Selline tähelepanek võib meile öelda, et hindamisel ei olnud erimeelsusi ja see on üsna objektiivne, sest samade ülesannete puhul oli arvamus ligikaudu ühtne.

Esimeses kriteeriumis „Selged ja tähenduslikud nimed“ on ChatGPT O3 ja DeepSeek R1 kõikjal ligikaudu ühesugused hinded, ainult ülesanded 4.1.2, 4.1.3 ja 4.2 on erinevad.

Teises kriteeriumis ülesannetes 4.1.2 ja 4.2 olid DeepSeeki lahendused ekspertide hinnangul veidi tugevamad, esimesel juhul oli DeepSeeki skoor “4.4” vastu “4.2”. Ja teises oli see vastavalt “5” vs. “4.8”. See on tegelikult veidi erinev autorite arvamusest, sest meie arvamus oli teistsugune, kuid eksperdirühma hinnang on huvitav. Kolmanda kriteeriumi puhul on tulemused sarnased eelmistele, hinded on ligikaudu võrdsed, kusjuures veamarginaaliga.

Kriteeriumi “Lakoonilisus” puhul on enamikul juhtudel eelis ChatGPT-1 või on nad mõnes ülesandes võrdsed. Viimane näitaja on “Kommentaaride eesmärgipärasus”. Siin, nagu juba varem öeldud, on eelis OpenAI toote poolel. Meie töö käigus oleme seda mitu korda märganud ja ka ekspertide rühm ei jätnud seda tähelepanuta.

Hinnangute põhjal võime moodustada arvamuse, et sõltuvalt ülesandest kõigi kriteeriumide järgi võib ükskõik milline mudel võtta esikoha. Iga tehisintellektuaal suudab oma tööd hästi teha. Korrapärasus on ainult viimases parameetris, milles DeepSeek on süstemaatiliselt halvem. Võrdlust kokku võttes tasub kõik hinded kokku liita, et arvutada ekspertide võitja. ChatGPT keskmine hinne kõigi ülesannete ja kriteeriumide puhul on “4.5” ja DeepSeekil “4.32”, mis võib viidata sellele, et ChatGPT O3 sai paremini hakkama.

5.6 OOP-lähenemise analüüs ja tulevikusuunad

Objektipõhise ülesande kaasamine käesolevasse uuringusse tõi esile mitmeid omadusi, mis erinesid oluliselt lihtsatest algoritmilistest ülesannetest. Meie testid näitasid, et OOP-ülesannete puhul ei saa kohaldada samu hindamismeetodeid kui täpsusele või kokkukuuluvusele orienteeritud ülesannete puhul. See avab võimaluse teema edasiseks laiendamiseks ja LLM-mudelite (nagu ChatGPT O3 ja DeepSeek R1) käitumise süstemaatiliseks uurimiseks kontekstides, kus on vaja ehitada keerulisemaid arhitektuure.

Eksperimendi alguses sai selgeks: ainult väga täpse ja üksikasjaliku struktureeritud üleskutse abil suutsid mõlemad mudelid luua lahenduse, mis sobis olemasoleva testikomplektiga. Sellised üleskutsed sisaldasid konkreetseid klassinimesid, meetodiallkirju ja loogika kirjeldusi. Üldisema lähenemise korral - näiteks ilma signatuuride täpsustamata - muutus mudelite käitumine väga erinevaks. See muutis võimatuks sama testikomplekti kasutamise valideerimiseks. See tulemus kinnitas meie

hüpoteesi, et OOP-ülesannetega tegelemisel mängivad ülesande klasside hulk ja nendevahelised seosed kriitilist rolli ja muutuvad lahendused universaalsel testimisel ebastabiilseks. Kuigi mudelid toimivad hästi antud struktuuride reprodutseerimisel, jääb nende võime iseseisvalt arhitektuurilisi otsuseid teha siiski piiratuks.

Tasub märkida mõlema mudeli loodud lahenduste iseloomulikke omadusi. Kuigi mõlemad koodid rakendavad sama ülesannet - kontonumbri genereerimine, riigi tuvastamine IBANi järgi, eesti formaadi toetamine "EstonianIBANCalculator" alamklassi kaudu, on erinevused rakenduses otseselt seotud ülesande sõnastusega. Täpsema ja rangelt struktureeritud prompti kasutamisel kipuvad mudelid genereerima kompaktset ja funktsionaalset koodi minimaalsete kõrvalekalletega, samas kui üldise ülesande kirjelduse puhul võivad struktuur, stiil ja rakendamise sügavus oluliselt varieeruda.

Üksikasjalik prompt

ChatGPT kood üksikasjalikust promptist demonstreerib tehniliselt sügavaid otsuseid: countryCodes.txt lugemisel kasutatakse Lazy<Dictionary<>>, et tagada andmete laisk ja turvaline initsialiseerimine. Hiljutiste IBANide ajalugu salvestatakse Queue<string>-sse, mille ülevoolu kontroll on käsitsi juhitud. IBANi märkide teisendamine numbrilisel kujul toimub otse foreachis ilma tarbetu mälu eraldamiseta. Mõned koodi osad on siiski ülekoormatud käsitsi kirjutatud loogikaga - näiteks IBANi märkide kontrollimine, kasutades Span<char>, stackalloc, dubleeritud AddToHistory-kõne igas meetodis.

DeepSeek'i lahendus sarnase promptiga tundub samuti ülevaatlikum. CountryCodes.txt lugemine on rakendatud staatilise konstruktori kaudu, tähemärke töödeldakse StringBuilder ja ConvertToNumericString meetodi abil. Kasutatakse vähem pesastamistasandeid, rohkem LINQ-d ja vähem käsitsi töötlemist, mistõttu on kood lihtsamini arusaadav ja testitav.

Üldine prompt

ChatGPT-i kood tavalisest promptist on hästi dokumenteeritud, kontrollib mitte ainult süntaksit, vaid ka IBAN-i pikkust sõltuvalt riigist. Samuti tagab ta, et ei genereerita identseid kontonumbreid. IBAN-ide ajalugu tagastatakse nimekirjana, mis on andmetega töötamiseks mugav. Rakendus loeb riigikode failist ning juhul, kui faili ei leita, kasutatakse varulahendust sisse ehitatud andmetega. See muudab koodi mugavaks

laiendamiseks ja hooldamiseks tulevikus. Kuna prompt oli üldisem ja sisaldas vaid ülesande kirjeldust ilma tehniliste piiranguteta, oli mudelil rohkem vabadust lahenduse kujundamisel – ja ChatGPT kasutas seda üsna sisukalt.

Tavalise promptiga DeepSeeki kood on samuti hästi struktureeritud ja laiendatav. See kasutab riigiandmeid CSV-vormingus, mis sisaldab IBAN-koodi, nime ja pikkuseinfot. Tänu sellele saab iga riigi numbrivormingut täpsemalt valideerida. Ajalugu salvestatakse loendina ja sellele pääseb ligi meetodi kaudu, mitte ei kuvata seda lihtsalt ekraanil. Ka siin on Eesti loogika viidud eraldi klassi ning lisatud IBAN-i kontrollsumma kontroll. Kuna prompt oli üldsõnalisem, oli ka DeepSeekil rohkem võimalusi oma lahendusstruktuuri kujundada, mille tulemusena valmis praktiliseks kasutuseks sobiv, hästi laiendatav arhitektuur.

Kokkuvõtte ja edasiarendus

Kokkuvõttes saab öelda, et OOP-lahenduste väljatöötamisel on oluline juba alguses mõista, millist lõpptulemust soovitakse saavutada. See näitab, et LLM-mudelite hindamiseks ei piisa ainult automaatsetest testidest ja kvantitatiivsetest mõõdikutest. Tuleb kavandada kogu projekti struktuur: määratleda klassid, nende vastutusala ja omavahelised seosed. Kõik see loob aluse rakenduse loogikale, mille põhjal koostatakse vajalikud meetodid klasside eesmärkide täitmiseks.

Kuigi meie katsed ei viinud ühise lahenduseni, kui kasutati üldisemaid ja vähem määratletud prompt'e, näitasid need siiski, kui erinevalt mudelid suudavad läheneda samale probleemile ja õigesti seda lahendada. See on väärtuslik teadmine iseenesest ja avab ukse edasiseks uurimiseks. Tulevikus võiks keskenduda sellele, kuidas selliseid erinevaid lahendusi oleks võimalik objektiivselt võrrelda - näiteks universaalsete hindamiskriteeriumide või spetsiaalsete struktuurianalüüsi tööriistade abil.

5.7 Kogumuspõhine hinnang ja soovitused arendajatele

Selles peatükis jagame oma isiklikke kogemusi ChatGPT O3 ja DeepSeek R1 kasutamisest praktiliste tarkvaraliste ülesannete lahendamisel ning anname soovitusi arendajatele, kes kavatsevad neid vahendeid oma töös kasutada.

ChatGPT O3 on näidanud suurt reageerimiskiirust, mugavat väljundformaati ja võimet mõista isegi ebatäpseid või valesti kirjutatud päringuid. See muudab selle eriti kasulikuks igapäevatöös, kui on vaja kiiresti katsetada mõnd ideed või genereerida koodifragment. Mõnel juhul ei anna mudel siiski üksikasjalikku selgitust lahenduse loogikast ja keerukamate tehniliste ülesannete puhul võib see analüüsi sügavuse poolest olla DeepSeekile allajääv.

Seevastu DeepSeek paistab silma üksikasjalike samm-sammuliste selgituste ja “mõtlemisstiilis” kommentaaridega. See muudab selle eriti kasulikuks algoritmiliselt keeruliste või analüütiliste probleemide puhul. Mudel on siiski tundlikum päringu sõnastuse suhtes - ebaselged või halvasti struktureeritud juhised võivad põhjustada vigu või ebatäielikku koodi. Lisaks on DeepSeek märgatavalt aeglasem ja selle väljundstruktuuris esineb suurema tõenäosusega vigu.

Igal juhul kinnitasid katse tulemused veel ühe meie hüpoteesi paikapidavust: lihtsamate üheklassiliste ülesannete puhul suutsid nii ChatGPT kui ka DeepSeek genereerida korrektsed ja kvaliteetsed lahendused, kui kasutati selgelt ja täpselt koostatud “ideaalset” prompti. Mõlema mudeli vastused olid sellistes tingimustes väga sarnased ja läbisid edukalt testid.

Seega need on praktilised kogemuste põhjal soovitusi arendajatele:

1. Püüdke sõnastada taotlused selgelt, konkreetselt ja üheselt mõistetavalt. Mida vähem peab mudel “arvama”, seda kvaliteetsem on tulemus.
2. Kasutage samm-sammult juhiseid - näiteks paluge mudelil kõigepealt kirjeldada lahenduskava ja seejärel anda lõplik kood (“Mõtlege samm-sammult, siis kirjutage kood”).
3. Jaotage keerulised probleemid alamülesanneteks ja lahendage need etapiviisiliselt.
4. Testige ja valideerige genereeritud koodi alati - isegi hästi treenitud mudel võib eksida.

5. Vältige liiga üldisi või ebamääraseid päringuid.
6. Hoidke silm peal uuenduste ja mudeliversioonide suhtes - nii ChatGPT kui ka DeepSeek arenevad aktiivselt.

Üldiselt osutus ChatGPT O3 paremaks ja kiiremaks vahendiks igapäevaseks arenduseks, kus kiirus ja paindlikkus on olulised. DeepSeek R1 seevastu võib olla kasulik selgitavaks abiliseks keerulisemate ülesannete puhul. Need AI-mudelid on arendaja jaoks nagu väga nutikas assistent, kellele antakse lahendada täpne probleem ja kes siis suure tõenäosusega annab kiire ja õige lahenduse. Töö nende vahenditega võib oluliselt kiirendada arendusprotsessi, kui järgida ülaltoodud soovitusi.

6 Kokkuvõte

Käesoleva bakalaureusetöö eesmärk oli võrrelda kahe levinud LLM-mudeli - ChatGPT O3 ja DeepSeek R1 - võimekust toota funktsionaalselt korrektset, loetavat ja tõhusat C#-koodi erinevat tüüpi programmeerimisülesannete lahendamisel. Lisaks sooviti hinnata, millised on mudelite praktilised eelised ja puudused kasutusmugavuse ja töövoo tõhususe vaates. Töö keskmes oli küsimus: milline mudel sobib paremini igapäevasesse arendusprotsessi?

Eesmärkide saavutamiseks viidi läbi eksperiment, kus mõlemale mudelile esitati viis programmeerimisülesannet, millest igaüht lahendati järjest viiel päeval sama prompti alusel. Koguti andmeid funktsionaalsuse, efektiivsuse, koodi loetavuse, kompaktsuse ning kasutajakogemuse kohta konsooli ja algoriitmilise ülesanne hinnates. Mõõdikute hulka kuulusid klassikalised kvaliteedinäitajad nagu Accuracy, Precision ja Recall ning spetsiaalselt efektiivsuse hindamiseks loodud valem.

Analüüsi tulemusena ilmnas, et ChatGPT saavutas kõigis mõõtkavades kõrgemaid tulemusi. ChatGPT O3 keskmine funktsionaalne täpsus oli 95%, võrreldes DeepSeeki R1 74%-ga. Samuti oli ChatGPT O3 keskmine efektiivsus (Efficiency) märkimisväärselt kõrgem - 0.69 võrreldes DeepSeeki R1 0.03-ga. Loetavuse, struktuuri, tõhususe ja kompaktsuse osas ilmnasid samuti selged erinevused. ChatGPT kood oli küll mõnevõrra pikem, kuid paremini struktureeritud - see sisaldas selgeid muutujaid, alamfunktsioone ja kasulikke kommentaare. Vastavalt hindamiskriteeriumitele sai ChatGPT maksimaalse hinne 5 peaaegu kõigis kategooriates, v.a kommentaaride eesmärgipärasuses, kus hinne oli 4. DeepSeek seevastu genereeris lühema, kuid sageli vähem arusaadava koodi, millel tihti puudusid kommentaarid või olid need sisult pealiskaudsed.

Mälukasutuses olulisi erinevusi ei täheldatud, mis tähendab, et ChatGPT ja DeepSeek pakkusid üldiselt efektiivsemaid lahendusi.

Metoodika peatükis algselt sõnastatud uuringu hüpoteesid leidsid eksperimendi käigus kinnitust: lihtsate üheklassiliste ülesannete puhul andsid mõlemad mudelid selgelt sõnastatud prompti alusel stabiilseid ja sarnaseid lahendusi, samas kui objektorienteeritud

ülesannete korral muutusid tulemused varieeruvaks ja testimisel vähem usaldusväärseks. Selle kinnituseks viidi läbi eraldi põhjalik OOP-analüüs, kus võrreldi erineva täpsusastmega promptide mõju mudelite käitumisele. Analüüs näitas selgelt, et mida üldisem on ülesande kirjeldus, seda rohkem erineb mudelite lähenemine — nii struktuurilt, loogikalt kui ka disainivalikutelt.

Eksperthinnangu alusel, mille viis läbi viieliikmeline professionaalne arendajate rühm, kogus ChatGPT keskmiseks hindeks 4.5 ja DeepSeek 4.32. Kõige järjepidevam erinevus esines kriteeriumis „Kommentaaride eesmärgipärasus“, kus ChatGPT oli stabiilselt tugevam. Mõned ülesanded (nt 4.2) näitasid DeepSeeki paremust teatud aspektides, kuid üldine koondtulemus jäi ChatGPT kasuks.

Töö põhjal võib järeldada, et ChatGPT sobib paremini igapäevaseks programmeerimiseks ja kiireks prototüüpimiseks, pakkudes paremat tasakaalu kiiruse, täpsuse ja kasutusmugavuse vahel. DeepSeek võib siiski olla kasulik hariduslikus või analüütilises kontekstis, kus on vaja mõtlemise selgitusi ja algoritmilist detailsust. Kokkuvõttes on ChatGPT praegusel hetkel usaldusväärsem töövahend praktilises tarkvaraarenduses.

6.1 Hinnang töö teostamise protsessile ja projekti logid

Antud töösse panustasid mõlemad autorid võrdselt. Töö protsess kestis 4 kuud, kus ülesanded ühise eesmärgi täitmiseks olid jagatud võrdselt mõlema meeskonnaliikme vahel. Töö alguses määratleti selgelt uurimisfookus, valiti katses osalevad LLM-mudelid (ChatGPT O3 ja DeepSeek R1) ning loodi täpne plaan ülesannete ja meetodika osas. Mõlemad autorid tegelesid uurimismetoodika täpsustamise, dokumentatsiooni koostamise ja tulemuste saamise, testimise ja analüüsimisega. Katsetamise tööprotsess jagati järgmiselt: Sofia keskendus ChatGPT O3 mudeli lahenduste genereerimise ja analüüsi eest, samal ajal kui Dmitri vastutas samamoodi DeepSeeki R1 mudeli eest.

Üldiselt sujus koostöö lõputöö kirjutamise ajal hästi ja ladusalt: osalejad hoidsid iga päev ühendust, arutasid töö järgmisi etappe, jagasid omavahel ülesandeid ja kujutasid selgelt ette, milline peaks praktilise osa protsess algusest lõpuni välja nägema. Autorid pidasid oma juhiga iganädalasi Teamsi konsultatsioone, kus nad arutasid nädala jooksul tehtud ülesandeid ning jagasid oma muresid ja õnnestumisi. Lisaks tekkis palju küsimusi ka väljaspool konsultatsiooni, kus juht oli alati valmis konteksti selgitama.

Kuna käesolevas töös tutvusid autorid esimest korda LLM-mudelite käitumise uurimisega, ilmnas töö algfaasis, et esialgsed hindamismeetodid ja mõõdikud ei olnud piisavalt objektiivsed ega sobinud kõigi ülesannetüüpide jaoks. See tõi kaasa vajaduse uurimisstruktuuri ümbermõtestamiseks ning mitmete analüüsietappide kohandamiseks, et tagada tulemuste võrreldavus ja usaldusväärsus kogu eksperimendi vältel. Algselt oli keeruline leida sobivat objektorienteeritud ülesannet, mis vastaks korraga nii piisavale keerukusastmele kui ka võimaldaks universaalset testimist. Autorid muutsid ülesannet mitu korda, et leida tasakaal selle vahel, et see oleks piisavalt sisuline, kuid samas ka mõõdetav. Kuigi alguses eeldati, et saab kasutada sarnaselt teistele ülesannetele mõõdetavaid ja testitavaid parameetreid, selgus praktikas, et ülesande üldisem sõnastus viis mudelite väga erinevate lahendusteni. See muutis testide kohandamise raskeks ja sundis ümber mõtlema ka analüüsi struktuuri. Lõpuks leiti siiski sobiv lähenemine, mis võimaldas hinnata tulemusi mõtestatult ja teha sisulisi järeldusi.

Tähelepanekutest võib tuua välja ainult seda, et mõnikord polnud liikmed mõningaid aspekte või detaile arvesse võtnud või polnud need eelnevalt läbi räägitud, mille tõttu pidi parandusi tegema.

6.1.1 Sofiia Hapon ajalogide kokkuvõte

Sofia Hapon tegevused nädala kaupa on kirjeldatud järgmises tabelis (Tabel 14).

Tabel 14. Sofiia Hapon ajalooigid.

Nädal	Tegevused
03.02 - 09.02	Meeskonna valimine ja teemade ülevaade
10.02 - 16.02	Esialgse bakalaureusetöö teema välja mõtlemine
17.02 - 23.02	- Esialgse bakalaureusetöö teema välja mõtlemine - Juhendaja otsimine
24.02 - 02.03	- Teema kinnitamine ja esialgne sõnastamine - Eesmärkide sõnastamine - Tehisintellekti õpetamiseks mudelite info kogumine

03.03 - 09.03	• Lõputöö dokumentatsioon (sissejuhatus) • Ülesandepüstitus
10.03 - 16.03	• Sissejuhatus ja eesmärgi osade kirjutamine • Mudelite valik ning kirjanduse uurimine
17.03 - 23.03	• Esialgsete mõõduki meetmeid kogumine • Metoodika osa kirjutamisega alustamine • Katseülesannete valik • Hüpoteeside püstitamine
24.03 - 30.03	• Endale mudeli valik edasitestimiseks (ChatGPT O3) • Metoodika osa kirjutamine • Promptide lauseehituse uurimine
31.03 - 06.04	• Praktilise osa kirjutamisega alustamine • Ülesannete kirjelduse koostamine • Ideaalse prompti koostamine ning testimine
07.04 - 13.04	• Ülesannete lahendamine ChatGPT O3 mudeliga (v.a OOP) • Ülesannete testimine LeetCode'is • Ideaalse prompti testimine • OOP ülesande sõnastuse korrigeerimine
14.04 - 20.04	• Metoodika osa objektiivsemaks tegemine • ChatGPT O3 katseandmete kogumine
21.04 - 27.04	• Metoodika osa objektiivsemaks tegemine • ChatGPT O3 katseandmete kogumine • Analüüsi osa kirjutamisega alustamine • Valemite kasutamine ning valideerimine • OOP ülesannete lahendamine ChatGPT O3 mudeliga
28.04 - 04.05	• ChatGPT O3 katseandmete kogumine • Analüüsi osa kirjutamine

05.05 - 11.05	• Analüüsi osa kirjutamine • OOP ülesanne muutmine
12.05 - 18.05	• Analüüsi osa kirjutamine • OOP ülesanne eraldi analüüsi kirjutamine ja edasiarenduse võimaluste uuring • Kogumuspõhine hinnangu ja soovitusi arendajatele kirjutamine
19.05 - 23.05	• Dokumentatsiooni nõuetekohane vormistamine • Kokkuvõte osa kirjutamine

6.1.2 Dmitriy Zakharenkov'i ajalogide kokkuvõte

Dmitriy Zakharenkov'i tegevused nädala kaupa on kirjeldatud järgmises tabelis (Tabel 15).

Tabel 15. Dmitriy Zakharenkov'i ajaloogid.

Nädal	Tegevused
03.02 - 09.02	• Meeskonna valimine ja teemade ülevaade
10.02 - 16.02	• Esialgse bakalaureusetöö teema välja mõtlemine
17.02 - 23.02	• Esialgse bakalaureusetöö teema välja mõtlemine • Juhendaja otsimine
24.02 - 02.03	• Teema kinnitamine ja esialgne sõnastamine • Eesmärkide sõnastamine • Tehisintellekti õpetamiseks mudelite info kogumine
03.03 - 09.03	• Lõputöö dokumentatsioon (sissejuhatus) • Ülesandepüstitus
10.03 - 16.03	• Sissejuhatus ja probleemi osade kirjutamine • Mudelite valik ning kirjanduse uurimine
17.03 - 23.03	• Kirjanduse ülevaade osa kirjutamine

	• Katseülesannete valik ja kogumine • Eksperthinnangu idee sõnastamine
24.03 - 30.03	• Endale mudeli valik edasitestimiseks (DeepSeek R1) • Kirjanduse ülevaade osa kirjutamine • Hüpoteeside püstitamine • Andmekaitse teema uurimine
31.03 - 06.04	• Ülesannete DeepSeek R1 mudeliga lahendamine • Ideaalse prompti koostamine ning testimine
07.04 - 13.04	• Ülesannete DeepSeek R1 mudeliga lahendamine • OOP ülesanne testimine • Ülesannete testimine LeetCode'is • DeepSeek R1 katseandmete kogumine
14.04 - 20.04	• Praktilise osa kirjutamisega jätkamine • DeepSeek R1 katseandmete kogumine
21.04 - 27.04	• OOP ülesannete lahendamine DeepSeek R1 mudeliga • DeepSeek R1 katseandmete kogumine • Analüüsi osa kirjutamisega alustamine • Valemite kasutamine ning valideerimine
28.04 - 04.05	• DeepSeek R1 katseandmete kogumine • Analüüsi osa kirjutamine
05.05 - 11.05	• Analüüsi osa kirjutamine • OOP ülesanne muutmine • DeepSeek R1 katseandmete kogumine • OOP ülesanne eraldi analüüsi põhimõtte sõnastamine
12.05 - 18.05	• Analüüsi osa kirjutamine • Eksperthinnangu küsimustiku teemine • Eksperthinnangu hinnete kogumine ja analüüsimine
19.05 - 23.05	• Dokumentatsiooni nõuetekohane vormistamine

Kasutatud kirjandus

- [1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, "Attention Is All You Need" [Võrgumaterjal]. Available: <https://arxiv.org/abs/1706.03762> [Kasutatud: 10 March 2025].
- [2] Wojciech Zaremba, Greg Brockman, OpenAI, "OpenAI Codex" [Võrgumaterjal]. Available: <https://openai.com/index/openai-codex/> [Kasutatud: 12 March 2025].
- [3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, Dario Amodei, "Language Models are Few-Shot Learners" [Võrgumaterjal]. Available: <https://arxiv.org/abs/2005.14165> [Kasutatud: 12 March 2025].
- [4] "Program synthesis" [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Program_synthesis [Kasutatud: 13 March 2025].
- [5] OpenAI, "Introducing ChatGPT" [Võrgumaterjal]. Available: <https://openai.com/index/chatgpt/> [Kasutatud: 15 March 2025].
- [6] DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Yang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J.L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jin Chen, Jingyang Yuan, Junjie Qiu, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qihao Zhu, Qinyu Chen, Qiushi Du, R.J. Chen, R.L. Jin, Ruiqi Ge, Ruizhe Pan, Runxin Xu, Ruyi Chen, S.S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Size Zheng, T. Wang, Tian Pei, Tian Yuan, Tianyu Sun, W.L. Xiao, Wangding Zeng, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wentao Zhang, X.Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun et al. (57 additional authors not shown), "DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model" [Võrgumaterjal]. Available: <https://arxiv.org/abs/2405.04434> [Kasutatud: 15 March 2025].
- [7] Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, Quyet V. Do, Yan Xu, Pascale Fung, "A Multitask, Multilingual, Multimodal Evaluation of ChatGPT on Reasoning, Hallucination, and Interactivity" [Võrgumaterjal]. Available: <https://arxiv.org/abs/2302.04023> [Kasutatud: 16 March 2025].

- [8] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, Shujie Liu, "CodeXGLUE" [Võrgumaterjal]. Available: <https://github.com/microsoft/CodeXGLUE> [Kasutatud: 17 March 2025].
- [9] OpenAI, "Prompt engineering best practices for ChatGPT" [Võrgumaterjal]. Available: <https://help.openai.com/en/articles/10032626-prompt-engineering-best-practices-for-chatgpt> [Kasutatud: 18 March 2025].
- [10] Gulf Business, "Nvidia shares tank as DeepSeek triggers AI selloff" [Võrgumaterjal]. Available: <https://gulfbusiness.com/nvidia-deepseek/> [Kasutatud: 28 March 2025].
- [11] Zhenda Xie, "DeepSeek" [Võrgumaterjal]. Available: <https://github.com/deepseek-ai> [Kasutatud: 28 March 2025].
- [12] CloudFlare, "What is a large language model (LLM)?" [Võrgumaterjal]. Available: <https://www.cloudflare.com/ru-ru/learning/ai/what-is-large-language-model/> [Kasutatud: 28 March 2025].
- [13] Stephen Kenney, "ChatGPT vs. DeepSeek: How the two AI titans compare" [Võrgumaterjal]. Available: <https://www.uc.edu/news/articles/2025/03/chatgpt-vs-deepseek-how-the-two-ai-titans-compare.html#:~:text=Final%20score%3A%20ChatGPT%204%2C%20DeepSeek,and%20innovation%20when%20used%20responsibly> [Kasutatud: 4 April 2025]
- [14] Vinod Chugani, "DeepSeek vs. ChatGPT: How Do They Compare?" [Võrgumaterjal]. Available: <https://www.datacamp.com/blog/deepseek-vs-chatgpt> [Kasutatud: 4 April 2025]
- [15] LeetCode, "27. Remove Element" [Võrgumaterjal]. Available: <https://leetcode.com/problems/remove-element/description/> [Kasutatud: 10 April 2025]
- [16] LeetCode, "43. Multiply Strings" [Võrgumaterjal]. Available: <https://leetcode.com/problems/multiply-strings/description/> [Kasutatud: 10 April 2025]
- [17] LeetCode, "29. Divide Two Integers" [Võrgumaterjal]. Available: <https://leetcode.com/problems/divide-two-integers/description/> [Kasutatud: 10 April 2025]
- [18] LeetCode, "124. Binary Tree Maximum Path Sum" [Võrgumaterjal]. Available: <https://leetcode.com/problems/4sum/description/> [Kasutatud: 10 April 2025]
- [19] LeetCode, "124. Binary Tree Maximum Path Sum" [Võrgumaterjal]. Available: <https://leetcode.com/problems/binary-tree-maximum-path-sum/description/>
- [20] Evidently AI Team, "Accuracy vs. precision vs. recall in machine learning: what's the difference?" [Võrgumaterjal]. Available: <https://www.evidentlyai.com/classification-metrics/accuracy-precision-recall#:~:text=Accuracy%20is%20a%20metric%20that,often%20the%20model%20is%20right%3F> [Kasutatud: 15 April 2025]
- [21] Microsoft, "Common C# code conventions" [Võrgumaterjal]. Available: <https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions> [Kasutatud: 16 April 2025]
- [22] Kantar, "Kuidas analüüsida skaalaküsimuste andmeid?" [Võrgumaterjal]. Available: <https://www.kantaremor.ee/blogi/kuidas-analuusida-skaalakusimuste-andmeid/> [Kasutatud: 18 April 2025]

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks¹

Autorid Sofiia Hapon ja Dmitriy Zakharenkov

1. Annavad Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose “Tehisintellekti koodi genereerimise võimekuse hindamine arendajatele: ChatGPT vs. DeepSeek”, mille juhendaja on Rivo Lemmik
 - 1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
 - 1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Autorid on teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Autorid kinnitavad, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

20.05.2025

¹ Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtjaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 - Ideaalne prompt ülesannele

“Elemendi eemaldamine massiivist”

Sina oled C# programmeerimise ekspert ja algoritmide insener. Sinu ülesanne on kirjutada C# funktsioon, mis eemaldab täisarvude massiivist kõik väärtusele val vastavad elemendid kohapeal ning tagastab allesjäänud elementide arvu. Lahendus peab toimuma in-place, ilma lisamassiivide või -kogumite kasutamiseta.

Nõuded:

- Sisendid:
int[] nums – täisarvude massiiv
int val – eemaldatav väärtus
- Funktsioon peab:
 - Eemaldama kõik val väärtused massiivist nums kohapeal
 - Tagastama arvu k, mitu elementi ei võrdu val-iga
 - Paigutama esimesed k elementi massiivis nums nii, et need sisaldavad ainult allesjäänud väärtusi (järjekord võib muutuda)
 - Ülejäänud massiivi sisu peale k ei oma tähtsust

Näidistöövoog:

Sisend: nums = [3, 2, 2, 3], val = 3 → Väljund: 2 → nums = [2, 2, -, -]
Sisend: nums = [0,1,2,2,3,0,4,2], val = 2 → Väljund: 5 → nums = [0,1,4,0,3,-,-,-]

Piirangud:

- $0 \leq \text{nums.length} \leq 100$
- $0 \leq \text{nums}[i] \leq 50$
- $0 \leq \text{val} \leq 100$

Lahendus peab toimuma kohapeal – ei tohi kasutada lisamälu, mille maht sõltub massiivi pikkusest.

Lahendus peab läbima LeetCode'i kohandatud kohtuniku testi:

- Funktsiooni kutsutakse: `k = RemoveElement(nums, val);`
- Kontrollitakse, et `k` väärtus on korrektne
- Esimesed `k` elementi sorteeritakse ja võrreldakse oodatud väljundiga

Väljund peab olema järgmisel kujul:

// C# kood

<siia lahendus>

// Selgitus

<siia lühike selgitus>

Lisa 3 - Ideaalne prompt ülesannele “Stringide korrutamine”

Sina oled C# programmeerimise ekspert ja algoritmide insener.

Kirjuta C# funktsioon, mis korrutab kaks mitte-negatiivset täisarvu, mis on esitatud stringidena, ja tagastab korrutise samuti stringina.

Ära kasuta mingeid sisseehitatud meetodeid arvude teisendamiseks (`int.Parse`, `Convert.ToInt32`, `vars`) ega `BigInteger` tüüpi. Korrutamine peab toimuma numbrite kaupa, nagu käsitsi korrutamisel.

Nõuded:

- Sisendid: `num1` ja `num2`, mõlemad stringid pikkusega 1 kuni 200.
- Sisendid sisaldavad ainult numbreid (0–9) ja neil pole juhtivaid nulle (v.a "0").
- Väljund peab olema string, mis kujutab korrutise tulemust.

Näidistöövoog:

- Sisend: `num1 = "2"`, `num2 = "3"` → Väljund: `"6"`
- Sisend: `num1 = "123"`, `num2 = "456"` → Väljund: `"56088"`

Kirjuta puhas, loetav ja kommenteeritud C# kood. Esita lahendus meetodina ja lisa testjuhtumid. Koodi lõpus selgita lühidalt (3–5 lauset), kuidas algoritm töötab (nt käsitsi korrutamise loogika).

Väljund peab olema järgmisel kujul:

```
// C# kood
<siia lahendus>

// Selgitus
<siia lühike selgitus>
```

Lisa 4 - Ideaalne prompt ülesannele “Täisarvude jagamine”

Sina oled süsteemitase C# programmeerija ja madaltaseme algoritmide ekspert.

Kirjuta C# funktsioon, mis jagab kaks 32-bitist allkirjaga täisarvu dividend ja divisor ilma kasutamata korrutamist, jagamist ega modulo operaatorit.

Tulemuseks peab olema täisarvuline jagatise väärtus, mis on trunc’itud nulli suunas (ehk ilma murdosata).

Piirangud:

- Sisendid jäävad vahemikku -2^{31} kuni $2^{31} - 1$
- divisor ei tohi olla 0 (jagamine nulliga pole lubatud)
- Kui tulemus ületab $2^{31} - 1$, tagasta $2^{31} - 1$
- Kui tulemus on väiksem kui -2^{31} , tagasta -2^{31}

Näidistöövoog:

- Sisend: dividend = 10, divisor = 3 → Väljund: 3
- Sisend: dividend = 7, divisor = -3 → Väljund: -2

Rakendusjuhised:

- Kasuta tsükleid, lahutamist, bitinihkeid ja loogilisi operatsioone.
- Ära kasuta jagamis-, korrutamise- ega modulo-operatsioone.
- Arvesta erandjuhtumeid, nt `Int32.MinValue / -1`.

Kirjuta puhas, modulaarselt ja kommenteeritud C# kood.

Lisa testjuhtumid ja lühike (3–5 lauset) selgitus kasutatud lahenduskäigu kohta.

Väljund peab olema kujul:

```
// C# kood
```

```
<siia kood>
```

```
// Selgitus
```

```
<siia lühike kirjeldus>
```

Lisa 5 - Ideaalne prompt ülesannele “Neliku summa (4Sum)”

Sina oled C# ekspert, kes on spetsialiseerunud efektiivsetele algoritmidele, kasutades sorteerimist ja kahe osuti tehnikat.

Ülesanne:

Kirjuta C# funktsioon, mis leiab kõik unikaalsed nelikute kombinatsioonid `[nums[a], nums[b], nums[c], nums[d]]` massiivist `nums`, kus:

Kõik indeksid on erinevad ($a \neq b \neq c \neq d$)

Nelikute elementide summa võrdub etteantud target väärtusega

Nõuded:

- Välti duplikaate – iga unikaalne nelik peab olema vastuses ainult korra, sõltumata järjekorrast.
- Rakenda efektiivne lahendus, kasutades sorteerimist, mitut tsüklit ja kahe osuti tehnikat.

Sisendi piirangud:

- $1 \leq \text{nums.Length} \leq 200$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- $-10^9 \leq \text{target} \leq 10^9$

Näidistöövood:

- Sisend: `nums = [1, 0, -1, 0, -2, 2]`, `target = 0`
- Väljund: `[[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]`
- Sisend: `nums = [2, 2, 2, 2, 2]`, `target = 8`
- Väljund: `[[2, 2, 2, 2]]`

Rakendusjuhised:

- Kasuta sorteerimist, et lihtsustada duplikaatide tuvastamist ja kahe osuti kasutamist.
- Fikseeri esimene ja teine element kahes tsüklis, seejärel kasuta kahte osutit järelejäänud kahe numbri leidmiseks.
- Välti duplikaate, jättes samad väärtused vahele igal sammul.
- Kirjuta puhas, modulaarselt üles ehitatud ja kommenteeritud C# kood.

Lühike (3–5 lauset) selgitus kasutatud lahendusest.

Väljund peab olema järgmises formaadis:

```
// C# kood
```

```
<siia kood>
```

```
// Selgitus
```

```
<siia lühike kirjeldus>
```

Lisa 6 - Ideaalne prompt ülesannele “Binaarpuu maksimaalne tee summa”

Sina oled C# tarkvaraarendaja, kelle erialaks on rekursioon ja binaarpuude algoritmid. Sinu ülesanne on kirjutada C# funktsioon, mis leiab binaarpuu maksimaalse tee summa.

Tee on järjestus tippudest, mis on ühendatud servade kaudu. Iga tipp võib esineda ainult üks kord ja tee ei pea läbima juurt.

Nõuded:

- Sisend: mitte-tühi binaarpuu, kus igas sõlmes on täisarvuline väärtus
- Tippude arv vahemikus: $1 \leq \text{sõlmede arv} \leq 30\,000$
- Väärtuste vahemik: $-1000 \leq \text{sõlme väärtus} \leq 1000$
- Väljund: täisarv, mis näitab maksimaalset võimalikku tee summat ükskõik kus puus.

Näidistöövood:

Näide 1:

Sisend: [1, 2, 3]

Väljund: 6

Selgitus: Parim tee on $2 \rightarrow 1 \rightarrow 3$, summa $2 + 1 + 3 = 6$

Visuaal:

```
    1
   / \
  2   3
```

Näide 2:

Sisend: [-10, 9, 20, null, null, 15, 7]

Väljund: 42

Selgitus: Parim tee on $15 \rightarrow 20 \rightarrow 7$, summa $15 + 20 + 7 = 42$

Visuaal:

```
      -10
      /  \
     9    20
      /  \
     15    7
```

Rakendusjuhised:

- Kasuta postorder-läbimist (vasak $\rightarrow$ parem $\rightarrow$ sõlm) rekursiooni abil
- Arvuta igas sõlmes maksimaalne ühe haru kasum, samal ajal uuendades globaalse maksimumi kahe haru teega
- Vanemale tuleb tagastada ainult ühe haru tee, et säilitada kehtivus
- Kasuta globaalset muutujat maksimaalse väärtuse salvestamiseks

Väljund peab olema kujul:

```
// C# kood

<siia C# kood>

// Selgitus

<3-5 lauset algoritmi tööloogikast>
```

Lisa 7 - Üksikasjalik prompt ülesannele “IBAN kalkulaator”

Sina oled kogenud C# 12 arhitekt. Sinu ülesanne on kujundada ja realiseerida teek IBANCalculator.Core, mis töötab .NET 8 raames, järgides täpselt kirjeldatud avalikku API-t ja loogikat.

Iga avalik tüüp peab:

- kuuluma nimelisse ruumi IBANCalculator.Core;

- paiknema eraldi failis, mis algab kommentaariga `// File: <TypeName>.cs`.

Teek peab suutma:

- genereerida suvalisi kontonumbreid ja IBAN-e;
- valideerida üldiseid ja Eesti IBAN-e;
- tuvastada riigi vastavalt IBANi algusele failist `countryCodes.txt`;
- meeles pidada viimaseid 5 IBAN-i;
- sisaldama Program-näidet, mis demonstreerib API kasutamist;
- kompileeruma hoiatuseta ja läbima varjatud automaattestid.

Avalik API (peab vastama sõna-sõnalt)

#	Tüüp	Kirjeldus	Kohustuslikud public-liikmed
1	<code>public interface IIBANCalculator</code>	Ühtne liides	Omadused: <code>Guid Id</code>, <code>string Name</code>, <code>TaskPriority Priority</code>, <code>DateTime EnqueuedAt (UTC)</code>. Konstruktorid: <code>QueueTask()</code> (genereerib <code>Id</code>, <code>EnqueuedAt</code>, seab <code>Name = ""</code>, <code>Priority = Normal</code>) ja <code>QueueTask(string name, TaskPriority priority)</code>. Ära lisa baasklasse, liideseid ega ülekirjutusi (<code>Equals</code>, <code>PrintMembers</code> jne.).
2	<code>public class IBANCalculator : IIBANCalculator</code>	Üldine IBAN-abiline	Väljad: <code>private readonly Queue<string> _history = new(5);</code> <code>private static readonly Random _rng = new();</code> Meetodid: (realiseerivad liidese) • <code>GenerateAccountNr()</code> – tagastab 10–17-kohalise numbri, esimene $\neq 0$

			• ValidateIBAN() – MOD-97-10 valideerimine • FindCountry() – tuvastab riigi failist • PrintRecentIbans() – kuvab 5 viimast IBAN-i
3	<pre>public sealed class EstonianIBANCalculator : IBANCalculator</pre>	Eesti- põhine loogika	Ülekirjutab ValidateIBAN: • algus EE • pikkus 20 tähemärki • 5–6. koht numbrid Uus meetod: <pre>string GenerateIBAN(string checkAndBankNr, string accountNr) → tagastab "EE" + checkAndBankNr + accountNr.PadLeft(14, '0')</pre> Kui ei läbi valideerimist → InvalidOperationException

GenerateAccountNr

- pikkus 10–17
- esimene number ei tohi olla 0
- ülejäänud kohad täidetakse suvaliste numbritega

ValidateIBAN (üldine):

- eemalda tühikud ja teisenda suurtäheks
- kontrolli: pikkus 5–34 ja ainult täht/number
- nihuta esimesed 4 tähemärki lõppu
- teisenda tähed A–Z → 10–35, loe mod 97
- kehtiv, kui tulemuseks on 1

Vead või parsimisprobleemid → tagasta false

FindCountry

- loeb countryCodes.txt ainult 1 korra:
- asukoht: ApplicationContext.BaseDirectory
- formaat: <kood><tühik><riiginimi>
- tagastab riigi nime või null

PrintRecentIbans

- kuvab viimased 5 IBAN-i (uuemad ees) Console.WriteLine kaudu

ValidateIBAN (Eesti jaoks)

- kutsub base.ValidateIBAN(), kui false → tagasta false
- lisaks:
 - peab algama "EE"
 - täispikkus peab olema täpselt 20
 - kohad 5 ja 6 peavad olema numbrid

GenerateIBAN(string checkAndBankNr, string accountNr)
parameetrid:

- checkAndBankNr.Length == 4, kõik numbrid
- accountNr.Length peab olema 10–14, kõik numbrid
- tulemus: "EE" + checkAndBankNr + accountNr.PadLeft(14, '0')
- kontrolli tulemust ValidateIBAN() abil
- kui see ebaõnnestub → viska InvalidOperationException

Täiendavad reeglid:

- Pole lubatud kasutada koodi väljaspool meetodeid (static init, var, jms).
- Mitte mingit lisapinda peale defineeritu.
- _history võib olla ühe lõime sees – pole vaja lock.
- Soositud: era-abilised meetodid, mis on selgenimelised.

Oodatud väljund (failistruktuur):

```
// C# Code
```

```
// File: IIBANCalculator.cs
```

```
...
```

```
// File: IBANCalculator.cs
```

```
...
```

```
// File: EstonianIBANCalculator.cs
```

```
...
```

```
// Explanation
```

```
<3–5 selget lauset, mis selgitavad arhitektuuri ja töövoogu>
```

Lisa 8 - Tavaline prompt ülesannele “IBAN kalkulaator”

Sa oled C# programmeerimise ekspert ja tarkvaraarhitekt. Sinu ülesanne on luua IBAN kalkulaatori süsteem C# keeles, kasutades objektorienteeritud programmeerimist ja Clean Code põhimõtteid. Ülesanne jaguneb baasosaks ja valikuliseks osaks. Rakenda nii palju kui võimalik. Pead looma klassid eraldi failidesse ja kasutama liidest (interface). Ära lisa ülesandes mitte nõutud meetodeid. Järgi täpselt meetodite nimesid ja nõudeid.

Peab sisaldama järgmisi meetodeid:

- [illegible]

- Väljund: max 5 stringi, viimased esimesena

NÕUDED:

Kasuta Clean Code põhimõtteid: selged nimed, loogiline jaotus, koodi kordamise vältimine

Kood peab kompileeruma ja töötama ilma vigadeta

Kõik klassid eraldi failidesse, vähemalt üks liides (interface) peab olema

Lisa kommentaare ja järgi standardset C# struktuuri

Ära lisa mittevajalikke meetodeid; abimeetodid on lubatud

Ära kasuta AI poolt genereeritud koodi – eelda, et teed selle ise

VÄLJUNDI FORMAAT:

```
// C# kood
```

```
<siia tuleb sinu korrektne, puhas, objektorienteeritud kood>
```

```
// Selgitus
```

Lühike selgitus (3–5 lauset) sinu lahenduse kohta:

- Milliseid klasse sa lõiid ja miks
- Kuidas sa lahendasid valideerimise ja genereerimise
- Kuidas on üles ehitatud Eesti IBAN loogika

Lisa 9 - ChatGPT ülesanne 1 - Elemendi eemaldamine massiivist, lahendus

```
public int RemoveElement(int[] nums, int val)
{
    int left = 0;
    int right = nums.Length - 1;
    while (left <= right)
    {
        if (nums[left] == val)
        {
            nums[left] = nums[right];
            right--;
        }
        else
        {
            left++;
        }
    }

    return left;
}
```

Lisa 10 - ChatGPT ülesanne 2 - Stringide korrutamise, lahendus

```
public class Solution
{
    public string Multiply(string num1, string num2)
    {
        if (num1 == "0" || num2 == "0")
            return "0";

        int n1 = num1.Length;
        int n2 = num2.Length;
        int[] result = new int[n1 + n2];
```

```

    for (int i = n1 - 1; i >= 0; i--)
    {
        int digit1 = num1[i] - '0';
        for (int j = n2 - 1; j >= 0; j--)
        {
            int digit2 = num2[j] - '0';

            int posLow = i + j + 1;
            int posHigh = i + j;

            int mul = digit1 * digit2 + result[posLow];
            result[posLow] = mul % 10;
            result[posHigh] += mul / 10;
        }
    }

    var sb = new System.Text.StringBuilder();
    bool leadingZero = true;
    foreach (var digit in result)
    {
        if (leadingZero && digit == 0)
            continue;

        leadingZero = false;
        sb.Append((char)(digit + '0'));
    }

    return sb.ToString();
}
}

```

Lisa 11 - ChatGPT ülesanne 3 - Täisarvude jagamine, lahendus

```

public class Solution
{
    /// <summary>
    /// Jagab kaks 32-bitist allkirjaga täisarvu ilma * / %-ita

```

```

    /// ja kärbib tulemuse Int32 piiridesse.
    /// </summary>
    public int Divide(int dividend, int divisor)    // enam pole
static
    {
        if (divisor == 0)
            throw new DivideByZeroException();

        // erand: Int32.MinValue / -1 ületaks vahemiku
        if (dividend == int.MinValue && divisor == -1)
            return int.MaxValue;

        bool negative = (dividend ^ divisor) < 0;

        long a = Math.Abs((long)dividend);
        long b = Math.Abs((long)divisor);

        int result = 0;

        for (int shift = 31; shift >= 0; shift--)
        {
            if ((a >> shift) >= b)
            {
                a -= b << shift;
                result += 1 << shift;
            }
        }

        return negative ? -result : result;
    }
}

```

Lisa 12 - ChatGPT ülesanne 4 - Neliku saumma, lahendus

```

public class Solution
{
    public IList<IList<int>> FourSum(int[] nums, int target)
    {
        var result = new List<IList<int>>();
        if (nums == null || nums.Length < 4)
            return result;

        Array.Sort(nums); // Sorteerib sisend massiivi
    }
}

```

```

int n = nums.Length;

for (int i = 0; i < n - 3; i++)
{
    // Väldi duplikaate esimese numbri puhul
    if (i > 0 && nums[i] == nums[i - 1])
        continue;

    for (int j = i + 1; j < n - 2; j++)
    {
        // Väldi duplikaate teise numbri puhul
        if (j > i + 1 && nums[j] == nums[j - 1])
            continue;

        int left = j + 1;
        int right = n - 1;

        while (left < right)
        {
            long sum = (long)nums[i] + nums[j] +
nums[left] + nums[right];

            if (sum == target)
            {
                result.Add(new List<int> { nums[i],
nums[j], nums[left], nums[right] });

                // Liigu vasakult paremale ja väldi duplikaate
                while (left < right && nums[left] ==
nums[left + 1]) left++;
                while (left < right && nums[right] ==
nums[right - 1]) right--;

                left++;
                right--;
            }
            else if (sum < target)
            {
                left++;
            }
            else // sum > target
            {
                right--;
            }
        }
    }
}

```

```

        return result;
    }
}

```

Lisa 13 - ChatGPT ülesanne 5 - Binaarpuu maksimaalne tee summa, lahendus

```

public class Solution {
    private int maxSum;

    public int MaxPathSum(TreeNode root) {
        maxSum = int.MinValue;
        MaxGain(root);
        return maxSum;
    }

    private int MaxGain(TreeNode node) {
        if (node == null) return 0;

        // Arvuta maksimaalne kasu vasakust ja paremast lapsest
        // (võttes 0 kui negatiivne)
        int leftGain = Math.Max(MaxGain(node.left), 0);
        int rightGain = Math.Max(MaxGain(node.right), 0);

        // Maksimaalne tee, mis läbib antud sõlme (võib olla
        // vasak + node + parem)
        int priceNewpath = node.val + leftGain + rightGain;

        // Uuenda globaalset maksimaalset väärtust
        maxSum = Math.Max(maxSum, priceNewpath);

        // Tagasta maksimaalne tee summeerimine ühelt poolt +
        // sõlme väärtus
        return node.val + Math.Max(leftGain, rightGain);
    }
}

```

Lisa 14 - DeepSeek ülesanne 1 - Elemendi eemaldamine massiivist, lahendus

```
public class Solution {  
    public int RemoveElement(int[] nums, int val) {  
        int left = 0;  
        int right = nums.Length - 1;  
        while (left <= right) {  
            if (nums[left] == val) {  
                nums[left] = nums[right--];  
            } else {  
                left++;  
            }  
        }  
        return right + 1;  
    }  
}
```

Lisa 15 - DeepSeek ülesanne 2 - Stringide korrutamise, lahendus

```
public class Solution {  
    public string Multiply(string num1, string num2) {  
        if (num1 == "0" || num2 == "0") return "0";  
  
        char[] arr1 = num1.ToCharArray();  
        char[] arr2 = num2.ToCharArray();  
        Array.Reverse(arr1);  
        Array.Reverse(arr2);  
  
        int[] product = new int[arr1.Length + arr2.Length];  
  
        for (int i = 0; i < arr1.Length; i++) {  
            int digit1 = arr1[i] - '0';  
            for (int j = 0; j < arr2.Length; j++) {
```

```

        int digit2 = arr2[j] - '0';
        product[i + j] += digit1 * digit2;
    }
}

int carry = 0;
for (int k = 0; k < product.Length; k++) {
    int total = product[k] + carry;
    product[k] = total % 10;
    carry = total / 10;
}

int start = product.Length - 1;
while (start >= 0 && product[start] == 0) start--;

StringBuilder result = new StringBuilder();
for (int i = start; i >= 0; i--) {
    result.Append(product[i]);
}

return result.ToString();
}
}

```

Lisa 16 - DeepSeek ülesanne 3 - Täisarvude jagamine, lahendus

```

public class Solution {
    public int Divide(int dividend, int divisor) {
        if (dividend == int.MinValue && divisor == -1) {
            return int.MaxValue;
        }

        int sign = (dividend < 0) == (divisor < 0) ? 1 : -1;

        if (dividend > 0) dividend = -dividend;
        if (divisor > 0) divisor = -divisor;

        int quotient = 0;

        while (dividend <= divisor) {
            int value = divisor;
            int power = 0;

```

```

        while (value >= (int.MinValue >> 1) && (value << 1)
        >= dividend) {
            value <<= 1;
            power++;
        }

        quotient += (1 << power);
        dividend -= value;
    }

    return sign == 1 ? quotient : -quotient;
}

```

Lisa 17 - DeepSeek ülesanne 4 - Neliku summa, lahendus

```

public class Solution {
    public IList<IList<int>> FourSum(int[] nums, int target) {
        IList<IList<int>> result = new List<IList<int>>();
        if (nums.Length < 4) return result;

        Array.Sort(nums);
        int n = nums.Length;

        for (int i = 0; i < n - 3; i++) {
            if (i > 0 && nums[i] == nums[i - 1]) continue;

            for (int j = i + 1; j < n - 2; j++) {
                if (j > i + 1 && nums[j] == nums[j - 1])
continue;

                int left = j + 1;
                int right = n - 1;

                while (left < right) {
                    long sum = (long)nums[i] + nums[j] +
nums[left] + nums[right];

                    if (sum == target) {
                        result.Add(new List<int> { nums[i],
nums[j], nums[left], nums[right] });

```

```

        while (left < right && nums[left] ==
nums[left + 1]) left++;
        while (left < right && nums[right] ==
nums[right - 1]) right--;
        left++;
        right--;
    }
    else if (sum < target) left++;
    else right--;
}
}
}
return result;
}
}
}

```

Lisa 18 - DeepSeek ülesanne 5 - Binaarpuu maksimaalne tee summa, lahendus

```

public class Solution {
    private int _maxSum;

    public int MaxPathSum(TreeNode root) {
        _maxSum = int.MinValue;
        CalculateMaxGain(root);
        return _maxSum;
    }

    private int CalculateMaxGain(TreeNode node) {
        if (node == null) return 0;

        int leftGain = Math.Max(CalculateMaxGain(node.left), 0);
        int rightGain = Math.Max(CalculateMaxGain(node.right),
0);

        int currentPathSum = node.val + leftGain + rightGain;
        _maxSum = Math.Max(_maxSum, currentPathSum);

        return node.val + Math.Max(leftGain, rightGain);
    }
}

```

```
}  
}
```

Lisa 19 - ChatGPT ülesanne 6 - IBAN kalkulaator, lahendus üksikasjaliku prompti abil

```
// C# Code  
// File: IIBANCalculator.cs  
using System;  
  
namespace IBANCalculator.Core  
{  
    public interface IIBANCalculator  
    {  
        string GenerateAccountNr();  
        bool ValidateIBAN(string iban);  
        string? FindCountry(string iban);  
        void PrintRecentIbans();  
    }  
}  
  
// File: IBANCalculator.cs  
using System;  
using System.Collections.Generic;  
using System.IO;  
using System.Linq;  
  
namespace IBANCalculator.Core  
{  
    public class IBANCalculator : IIBANCalculator  
    {  
        private readonly Queue<string> _history = new(5);  
        private static readonly Random _rng = new();  
  
        private static readonly Lazy<Dictionary<string, string>>  
_countryMap = new(() =>  
    {  
        var map = new Dictionary<string,  
string>(StringComparer.OrdinalIgnoreCase);  
        var file = Path.Combine(AppContext.BaseDirectory,  
"countryCodes.txt");  
        if (!File.Exists(file))
```

```

        return map;

foreach (var line in File.ReadAllLines(file))
{
    if (string.IsNullOrEmpty(line))
        continue;

    var parts = line.Split(' ', 2,
StringSplitOptions.RemoveEmptyEntries);
    if (parts.Length == 2)
        map[parts[0]] = parts[1].Trim();
}

return map;
});

public string GenerateAccountNr()
{
    var length = _rng.Next(10, 18);           // 10-17
digits
    Span<char> digits = stackalloc char[length];
    digits[0] = (char)('1' + _rng.Next(9));   // first
digit ≠ 0
    for (int i = 1; i < length; i++)
        digits[i] = (char)('0' + _rng.Next(10));

    return new string(digits);
}

public virtual bool ValidateIBAN(string iban)
{
    AddToHistory(iban);

    if (string.IsNullOrEmpty(iban))
        return false;

    try
    {
        var compact = iban.Replace(" ",
"".ToUpperInvariant());
        if (compact.Length < 5 || compact.Length > 34)
            return false;

        if (compact.Any(c => !char.IsLetterOrDigit(c)))
            return false;

        var rearranged = compact[4..] + compact[..4];

```

```

        int remainder = 0;
        foreach (char c in rearranged)
        {
            if (char.IsDigit(c))
            {
                remainder = (remainder * 10 + (c - '0'))
% 97;

            }
            else // A-Z → 10-35
            {
                int value = c - 'A' + 10;
                remainder = (remainder * 100 + value) %
97;

            }
        }

        return remainder == 1;
    }
    catch
    {
        return false;
    }
}

public string? FindCountry(string iban)
{
    AddToHistory(iban);

    if (string.IsNullOrEmpty(iban) || iban.Length < 2)
        return null;

    var code = iban[..2].ToUpperInvariant();
    return _countryMap.Value.TryGetValue(code, out var
country) ? country : null;
}

public void PrintRecentIbans()
{
    foreach (var item in _history.Reverse())
        Console.WriteLine(item);
}

private void AddToHistory(string? iban)
{
    if (string.IsNullOrEmpty(iban))
        return;

    if (_history.Count == 5)

```

```

        _history.Dequeue();

        _history.Enqueue(iban);
    }
}

// File: EstonianIBANCalculator.cs
using System;

namespace IBANCalculator.Core
{
    public sealed class EstonianIBANCalculator : IBANCalculator
    {
        public override bool ValidateIBAN(string iban)
        {
            if (!base.ValidateIBAN(iban))
                return false;

            var compact = iban.Replace(" ",
            "").ToUpperInvariant();
            if (!compact.StartsWith("EE",
            StringComparison.Ordinal) || compact.Length != 20)
                return false;

            // Bank-code must be digits at positions 5-6 (1-
            based)
            if (!char.IsDigit(compact[4]) ||
            !char.IsDigit(compact[5]))
                return false;

            return true;
        }

        public string GenerateIBAN(string checkAndBankNr, string
        accountNr)
        {
            if (string.IsNullOrEmpty(checkAndBankNr) ||
            string.IsNullOrEmpty(accountNr) ||
            checkAndBankNr.Length != 4 ||
            !IsAllDigits(checkAndBankNr) ||
            accountNr.Length is < 10 or > 14 ||
            !IsAllDigits(accountNr))
            {
                throw new ArgumentException("Invalid arguments
                for IBAN generation.");
            }
        }
    }
}

```

```

        var iban = "EE" + checkAndBankNr +
accountNr.PadLeft(14, '0');

        if (!ValidateIBAN(iban))    // defensive – should
always pass
            throw new InvalidOperationException("Generated
IBAN failed validation.");

        return iban;
    }

    private static bool IsAllDigits(string value)
    {
        foreach (char c in value)
        {
            if (!char.IsDigit(c))
                return false;
        }
        return true;
    }
}

```

Lisa 20 - ChatGPT ülesanne 6 - IBAN kalkulaator, lahendus tavalise prompti abil

```

// -----
// File: IIBANService.cs
// -----
namespace IbanCalculator
{
    /// <summary>
    /// Contract for basic IBAN functionality.
    /// </summary>
    public interface IIBANService
    {
        string GenerateAccountNr();           // 10-
17 digits, never starts with 0
        bool ValidateIBAN(string iban);      //
full IBAN validation

```

```

        string FindCountry(string iban);                                //
two-letter code → country name
        IReadOnlyList<string> GetRecentIBANs();                        //
last max 5 IBANs (most recent first)
    }
}

// -----
// File: CountryCodeProvider.cs
// -----
-----
using System;
using System.Collections.Generic;
using System.IO;

namespace IbanCalculator
{
    /// <summary>
    /// Loads country codes from <c>countryCodes.txt</c>
    (format: "DE;Germany").
    /// Falls back to a minimal in-memory set when the file is
    missing.
    /// </summary>
    internal static class CountryCodeProvider
    {
        private static readonly Lazy<Dictionary<string,string>>
_codes =
        new(() => Load());

        public static bool TryGetName(string code, out string
name) =>
            _codes.Value.TryGetValue(code, out name);

        private static Dictionary<string,string> Load()
        {
            var dict = new
Dictionary<string,string>(StringComparer.OrdinalIgnoreCase);

            try
            {
                foreach (var line in
File.ReadAllLines("countryCodes.txt"))
                {
                    var parts = line.Split('; ',
StringSplitOptions.RemoveEmptyEntries);
                    if (parts.Length == 2)
                        dict[parts[0].Trim()] = parts[1].Trim();
                }
            }
            catch { }
        }
    }
}

```

```

        }
    }
    catch
    {
        // minimal fallback – extend as needed
        dict["EE"] = "Estonia";
        dict["DE"] = "Germany";
        dict["FR"] = "France";
        dict["ES"] = "Spain";
        dict["IT"] = "Italy";
    }

    return dict;
}
}
}

// -----
// File: IBANService.cs
// -----
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Text.RegularExpressions;

namespace IbanCalculator
{
    /// <summary>
    /// Base implementation that covers account-number
generation, generic IBAN
    /// validation, country lookup and basic history.
    /// </summary>
    public class IBANService : IIBANService
    {
        private static readonly Regex _ibanSyntax =
            new(@"^[A-Z]{2}[0-9]{2}[A-Z0-9]{9,30}$",
                RegexOptions.Compiled);

        private static readonly Dictionary<string,int>
_ibanLengths =
            new(StringComparer.OrdinalIgnoreCase)
            {
                // The most common lengths; extend if needed.
                ["EE"] = 20, ["DE"] = 22, ["FR"] = 27,

```

```

        ["ES"] = 24, ["IT"] = 27
    };

    private readonly LinkedList<string> _history = new();

    private static readonly RandomNumberGenerator _rng =
        RandomNumberGenerator.Create();

    public string GenerateAccountNr()
    {
        int length = RandomInt(10, 17);           //
inclusive upper bound
        var sb = new StringBuilder(length);
        sb.Append(RandomInt(1, 9));               // first
digit ≠ 0
        for (int i = 1; i < length; i++)
            sb.Append(RandomInt(0, 9));
        return sb.ToString();
    }

    public bool ValidateIBAN(string iban)
    {
        if (iban is null)
            return false;

        iban = iban.Replace(" ",
string.Empty).ToUpperInvariant();

        if (!_ibanSyntax.IsMatch(iban))
            return false;

        // country-specific length test (when known)
        if (_ibanLengths.TryGetValue(iban[..2], out int
cLen) &&
            iban.Length != cLen)
            return false;

        // standard MOD-97 check
        var rearranged = iban[4..] + iban[..4];
        var numeric = new StringBuilder(rearranged.Length *
2);

        foreach (char ch in rearranged)
        {
            if (char.IsLetter(ch))
                numeric.Append((ch - 'A' + 10).ToString());
            else
                numeric.Append(ch);
        }
    }

```

```

    }

    int remainder = 0;
    foreach (char digit in numeric.ToString())
        remainder = (remainder * 10 + (digit - '0')) %
97;

    bool ok = remainder == 1;
    Track(iban, ok);
    return ok;
}

public string FindCountry(string iban)
{
    if (iban is null || iban.Length < 2)
        return string.Empty;

    var code = iban[..2].ToUpperInvariant();
    CountryCodeProvider.TryGetName(code, out var name);
    Track(iban, true);
    return name ?? string.Empty;
}

public IReadOnlyList<string> GetRecentIBANs() =>
    _history.ToList();

// -----
-----

protected void Track(string iban, bool accepted)
{
    if (!accepted) return;

    _history.Remove(iban);           // keep list unique
    _history.AddFirst(iban);
    if (_history.Count > 5)
        _history.RemoveLast();
}

protected static int RandomInt(int minInclusive, int
maxInclusive)
{
    Span<byte> buffer = stackalloc byte[4];
    _rng.GetBytes(buffer);
    int num = BitConverter.ToInt32(buffer);
    return (Math.Abs(num) % (maxInclusive - minInclusive
+ 1)) + minInclusive;
}

```

```

    }
}

// -----
// File: EstonianIBANService.cs
// -----
-----
using System;
using System.Text;
using System.Text.RegularExpressions;

namespace IbanCalculator
{
    /// <summary>
    /// Adds Estonia-specific behaviour to <see
    cref="IBANService"/>.
    /// </summary>
    public sealed class EstonianIBANService : IBANService
    {
        private static readonly Regex _eePattern =
            new(@"^EE[0-9]{2}[0-9]{2}[0-9]{14}$",
                RegexOptions.Compiled);

        /// <summary>
        /// Full EE-specific validation (length 20, bank-code +
        account part).
        /// </summary>
        public bool ValidateEstonianIBAN(string iban) =>
            _eePattern.IsMatch(iban?.Replace(" ", string.Empty)
            ?? string.Empty) &&
            ValidateIBAN(iban); // run
generic check as well

        /// <summary>
        /// Generates an EE IBAN from the constituent parts.
        Account number is
        /// padded to 14 digits.
        /// </summary>
        public string GenerateIBAN(string checkNr,
                                   string bankCode,
                                   string accountNumber)
        {
            if (checkNr?.Length != 2 || bankCode?.Length != 2)
                throw new ArgumentException("Check-number and
                bank-code must be 2 digits.");

            accountNumber ??= string.Empty;

```

```

        accountNumber = accountNumber.PadLeft(14, '0');

        if (!Regex.IsMatch(checkNr + bankCode +
accountNumber, @"^[0-9]+$"))
            throw new ArgumentException("All parts must be
numeric.");

        string iban =
$"EE{checkNr}{bankCode}{accountNumber}";
        if (!ValidateIBAN(iban))
            throw new InvalidOperationException("Generated
IBAN is illegal.");
        return iban;
    }

    /// <summary>
    /// Produces a random EE account number 10–14 digits
long (for convenience
    /// when composing IBANs).
    /// </summary>
    public string GenerateRandomEstonianAccountNumber()
    {
        int length = RandomInt(10, 14);
        var sb = new StringBuilder(length);
        sb.Append(RandomInt(1, 9)); // no
leading zero
        for (int i = 1; i < length; i++)
            sb.Append(RandomInt(0, 9));
        return sb.ToString();
    }
}

```

Lisa 21 - DeepSeek ülesanne 6 - IBAN kalkulaator, lahendus üksikasjaliku prompti abil

```

// File: IIBANCalculator.cs
namespace IBANCalculator.Core;

public interface IIBANCalculator
{
    string GenerateAccountNr();
}

```

```

        bool ValidateIBAN(string iban);
        string? FindCountry(string iban);
        void PrintRecentIbans();
    }
    // File: IBANCalculator.cs
    using System.Text;

    namespace IBANCalculator.Core;

    public class IBANCalculator : IIBANCalculator
    {
        private readonly Queue<string> _history = new(5);
        private static readonly Random _rng = new();
        private static readonly Dictionary<string, string>
        _countryCodes = new();

        static IBANCalculator()
        {
            try
            {
                var path = Path.Combine(AppContext.BaseDirectory,
"countryCodes.txt");
                foreach (var line in File.ReadAllLines(path))
                {
                    var parts = line.Split(' ', 2,
StringSplitOptions.RemoveEmptyEntries);
                    if (parts.Length == 2 && parts[0].Length == 2)
                        _countryCodes[parts[0].ToUpperInvariant()] =
parts[1];
                }
            }
            catch { }
        }

        public string GenerateAccountNr()
        {
            var length = _rng.Next(10, 18);
            var digits = new char[length];
            digits[0] = (char)('1' + _rng.Next(9));

            for (int i = 1; i < length; i++)
                digits[i] = (char)('0' + _rng.Next(10));

            return new string(digits);
        }

        public virtual bool ValidateIBAN(string iban)
        {

```

```

        EnqueueIfNotNull(iban);
        try
        {
            var clean = iban.Replace(" ",
"").ToUpperInvariant();
            if (clean.Length < 5 || clean.Length > 34 ||
!clean.All(char.IsLetterOrDigit))
                return false;

            var rearranged = clean[4..] + clean[..4];
            var numeric = ConvertToNumericString(rearranged);
            return ComputeMod97(numeric) == 1;
        }
        catch { return false; }
    }

    public string? FindCountry(string iban)
    {
        EnqueueIfNotNull(iban);
        if (iban?.Length < 2) return null;
        var code = iban![..2].ToUpperInvariant();
        return _countryCodes.GetValueOrDefault(code);
    }

    public void PrintRecentIbans()
    {
        foreach (var iban in _history.Reverse())
            Console.WriteLine(iban);
    }

    private void EnqueueIfNotNull(string? iban)
    {
        if (iban == null) return;
        if (_history.Count == 5) _history.Dequeue();
        _history.Enqueue(iban);
    }

    private static string ConvertToNumericString(string s)
    {
        var sb = new StringBuilder();
        foreach (var c in s)
        {
            if (char.IsDigit(c)) sb.Append(c);
            else sb.Append($"{c - 'A' + 10}");
        }
        return sb.ToString();
    }
}

```

```

        private static int ComputeMod97(string numeric)
        {
            int mod = 0;
            foreach (var c in numeric)
                mod = (mod * 10 + (c - '0')) % 97;
            return mod;
        }
    }
    // File: EstonianIBANCalculator.cs
    namespace IBANCalculator.Core;

    public sealed class EstonianIBANCalculator : IBANCalculator
    {
        public override bool ValidateIBAN(string iban)
        {
            if (!base.ValidateIBAN(iban)) return false;
            return iban.StartsWith("EE")
                && iban.Length == 20
                && iban[2..4].All(char.IsDigit)
                && iban[4..6].All(char.IsDigit);
        }

        public string GenerateIBAN(string checkAndBankNr, string
accountNr)
        {
            if (checkAndBankNr?.Length != 4 ||
!checkAndBankNr.All(char.IsDigit))
                throw new ArgumentException("Invalid check+bank");
            if (accountNr?.Length < 10 || accountNr.Length > 14 ||
!accountNr.All(char.IsDigit))
                throw new ArgumentException("Invalid account");

            var iban = $"EE{checkAndBankNr}{accountNr.PadLeft(14,
'0')}}";
            if (!ValidateIBAN(iban))
                throw new InvalidOperationException("Generated
invalid IBAN");

            return iban;
        }
    }
    // File: Program.cs
    namespace IBANCalculator.Core;

    internal static class Program
    {
        public static void Main()
        {

```

```

        IIBANCalculator calc = new IBANCalculator();
        Console.WriteLine($"Account:
{calc.GenerateAccountNr()}");
        Console.WriteLine($"Valid IBAN:
{calc.ValidateIBAN("EE871234567890123456")}");
        Console.WriteLine($"Country:
{calc.FindCountry("EE871234567890123456")}");

        var estCalc = new EstonianIBANCalculator();
        Console.WriteLine(estCalc.GenerateIBAN("1234",
"12345678901234"));
    }
}

```

Lisa 22 - DeepSeek ülesanne 6 - IBAN kalkulaator, lahendus tavalise prompti abil

```

// C# Code

// IIbanService.cs
public interface IIbanService
{
    string GenerateAccountNr();
    bool ValidateIBAN(string iban);
    string FindCountry(string iban);
    IEnumerable<string> GetHistory();
}

// IbanService.cs
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;

public class IbanService : IIbanService
{
    protected static readonly Random _random = new Random();
    private readonly Dictionary<string, (string Name, int Length)> _countryData;
    private List<string> _history = new List<string>();

    public IbanService()

```

```

{
    _countryData = new Dictionary<string, (string, int)>();
    var lines = File.ReadAllLines("countryCodes.txt");
    foreach (var line in lines)
    {
        var parts = line.Split(',');
        if (parts.Length >= 3)
        {
            string code = parts[0].Trim().ToUpper();
            string name = parts[1].Trim();
            if (int.TryParse(parts[2].Trim(), out int
length))
            {
                _countryData[code] = (name, length);
            }
        }
    }
}

public virtual string GenerateAccountNr()
{
    int length = _random.Next(10, 18);
    char[] digits = new char[length];
    digits[0] = _random.Next(1, 10).ToString()[0];
    for (int i = 1; i < length; i++)
        digits[i] = _random.Next(0, 10).ToString()[0];
    return new string(digits);
}

public virtual bool ValidateIBAN(string iban)
{
    AddToHistory(iban);

    if (string.IsNullOrEmpty(iban) || iban.Length < 4)
        return false;

    string countryCode = iban.Substring(0, 2).ToUpper();
    if (!_countryData.TryGetValue(countryCode, out var
countryInfo))
        return false;

    if (iban.Length != countryInfo.Length)
        return false;

    foreach (char c in iban)
        if (!char.IsLetterOrDigit(c) || (char.IsLetter(c) &&
!char.IsUpper(c)))
            return false;
}

```

```

        return true;
    }

    public virtual string FindCountry(string iban)
    {
        AddToHistory(iban);

        if (string.IsNullOrEmpty(iban) || iban.Length < 2)
            return null;

        string countryCode = iban.Substring(0, 2).ToUpper();
        return _countryData.TryGetValue(countryCode, out var
countryInfo) ? countryInfo.Name : null;
    }

    public IEnumerable<string> GetHistory() => _history.Take(5);

    private void AddToHistory(string iban)
    {
        _history.Insert(0, iban);
        if (_history.Count > 5)
            _history = _history.Take(5).ToList();
    }
}

// EstonianIbanService.cs
using System.Linq;
using System.Text;

public class EstonianIbanService : IbanService
{
    public bool ValidateEstonianIBAN(string iban)
    {
        if (!ValidateIBAN(iban) return false;
        string rest = iban.Substring(2);
        return rest.All(char.IsDigit) &&
IsValidIbanChecksum(iban);
    }

    public string GenerateIBAN(string checkNr, string bankCode,
string accountNumber)
    {
        string paddedAccount = accountNumber.PadLeft(14, '0');
        return $"EE{checkNr}{bankCode}{paddedAccount}";
    }

    public string GenerateEstonianAccountNr()

```

```

    {
        int length = _random.Next(10, 15);
        char[] digits = Enumerable.Range(0, length)
            .Select(_ => _random.Next(0,
10).ToString()[0]).ToArray();
        return new string(digits);
    }

    private bool IsValidIbanChecksum(string iban)
    {
        string rearranged = iban.Substring(4) +
iban.Substring(0, 4);
        var numericIban = new StringBuilder();
        foreach (char c in rearranged)
            numericIban.Append(char.IsLetter(c) ? (c - 'A' +
10).ToString() : c.ToString());

        int modResult = 0;
        foreach (char c in numericIban.ToString())
            modResult = (modResult * 10 + (c - '0')) % 97;

        return modResult == 1;
    }
}

```

Lisa 23 - Sofiia Hapon eneseanalüüs

Bakalaureusetööd teostasime kahekesi Dmitriy Zakharenkov'iga, kus panustasime mõlemad töö etappidesse võrdselt. Meie mõlemad tegelesime järgmiste ülesannetega:

- Mudelite (ChatGPT O3 ja DeepSeek R1) valimine ja nendega töötamise metoodika loomine;
- Viie erineva programmeerimisülesande kogumine ja nende testimiseks sobiva “ideaalse prompti” koostamine;
- Lihtsate algoritmiliste ülesannete lahenduste genereerimine mõlema mudeli abil;
- Ülesannete mõõdikute testimine LeetCode'i keskkonnas;
- Katsetulemuste struktureerimine ja nende põhjal kvantitatiivne analüüs (Accuracy, Precision, Recall, Efficiency ning koodi tsruktuurilised parameetreid);

- OOP-ülesande valimine, ümberkujundamine ja sobiva struktuuri leidmine testimise eesmärgil;
- OOP-ülesande põhjalik käsitsi analüüs ja hinnangud vastavalt ülesehitusele, disainile ja arusaadavusele;
- Ekspertrühma hindamiskriteeriumite koostamine ja koondtulemuste analüüs;
- Kogu dokumentatsiooni (sissejuhatus, metoodika, analüüs, järeldused) järkjärguline kirjutamine ja vormistamine.

Kõige keerulisemaks osutus töö jooksul oli sobiva OOP-ülesande leidmine ja testitava struktuuri kujundamine. Esialgsed katsed osutusid mudelite jaoks kas liiga lihtsaks või liiga abstraktseks ning tuli mitmel korral muuta ülesande sisu, et säilitada nii universaalsus kui ka testitavus. Lisaks tuli töö käigus mitmel korral kohandada uuringu metoodikat, kuna algselt kavandatud hindamismudelid ei olnud kõikide ülesannetüüpide jaoks rakendatavad. Sellised muudatused nõudsid korduvaid analüüse ja mitmeid uusi katsetusi.

Üldiselt oli diplomitöö arendamise protsess minu jaoks väga huvitav. Nii mina kui minu meeskonnakaaslane panustamise sellesse tööse maksimaalselt ning lõpptulemusena olen enda panusega rahul. See polnud esimene kord, kus oleme Dmitriga ühes meeskonnaprojektis tegutsenud, seega temaga koostöö tegemine edenes tõrkedeta ja sujuvalt, seejuures alati leidsime kompromisse, motiveerisime teineteist ja püüdsime jõuda ühise selge arusaamisele erinevate aspektide puhul.

Lõppkokkuvõttes usun, et nii minu kui ka mu meeskonnaliikme tehtud töö ja analüüs annab märkimisväärset väärtust koodi kvaliteedile hinnangu saamise ja koodi muutmise

lähenemisviiside osas ning arvan, et oleme püstitatud eesmärgiga edukalt hakkama saanud.

Lisa 24 - Dmitriy Zakharenkov eneseanalüüs

Diplomitööd tegime Sofia Hapon'iga kahekesi koos, kus kõik lõputöö sammud olid

teostatud ja jagatud meie mõlema poolt võrdselt. Nii mina kui ka Sofia oleme t

einud

järgmisi ülesandeid:

- Mudelite (ChatGPT O3 ja DeepSeek R1) valimine ja nendega töötamise metoodika loomine;
- Viie erineva programmeerimisülesande kogumine ja nende testimiseks sobiva “ideaalse prompti” koostamine;
- Lihtsate algoritmiliste ülesannete lahenduste genereerimine mõlema mudeli abil;
- Ülesannete mõõdikute testimine LeetCode'i keskkonnas;
- Katsetulemuste struktureerimine ja nende põhjal kvantitatiivne analüüs (Accuracy, Precision, Recall, Efficiency ning koodi tsruktuurilised parameetreid);
- OOP-ülesande valimine, ümberkujundamine ja sobiva struktuuri leidmine testimise eesmärgil;
- OOP-ülesande põhjalik käsitsi analüüs ja hinnangud vastavalt ülesehitusele, disainile ja arusaadavusele;
- Ekspertühma hindamiskriteeriumite koostamine ja koondtulemuste analüüs;
- Kogu dokumentatsiooni (sissejuhatus, metoodika, analüüs, järeldused) järkjärguline kirjutamine ja vormistamine.

Töö käigus ilmnes, et DeepSeek mudeli testimine oli aeganõudvam kui esialgu oodatud. Suureks takistuseks osutus mudeli märkimisväärselt pikem reageerimisaeg võrreldes ChatGPT-ga, mis aeglustas praktilist töövoogu ja andmete kogumist. Lisaks esines DeepSeeki genereeritud koodis sageli väljundstruktuuri probleeme — näiteks puudusid vajalikud plokid, kood oli katkestatud või vormindus ei vastanud ootustele. See tähendas, et iga probleemne lahendus tuli käsitsi üle vaadata, vajadusel parandada ja uuesti testida,

et kontrollida, kas see täidab ülesande nõudeid. Selline korduv ja manuaalne töö suurendas ajakulu märgatavalt ning nõudis põhjalikku tähelepanu igale juhtumile eraldi.

Samuti osutus keeruliseks OOP-ülesande analüüsi praktiline teostamine. Esialgu oli keeruline täpselt mõista, millist ülesannet valida ja kuidas seda testitavalt struktureerida, et see sobituks nii mudelite võrdlemiseks kui ka selgeks analüüsimiseks. Mitmeid kordi tuli muuta nii ülesande sisu kui ka selle sõnastust, enne kui jõudsime vormini, mis võimaldas usaldusväärselt hinnata mudelite käitumist. Alles mitmete katsete ja vigade kaudu kujunes arusaam sellest, milliseid komponente peame säilitama ja millist lähenemist kasutama, et tulemused oleksid võrreldavad ja sisukad.

Kogu protsessi jooksul kujunes arusaam, kui oluline on täpne promptide koostamine ning kuidas erinevad LLM-mudelid võivad samale ülesandele reageerida täiesti erinevalt — eriti just objektorienteeritud programmeerimise puhul. See muutis töö sisukaks ja õpetlikuks ning avas võimalused edasiseks uurimiseks.

Kokkuvõttes olen rahul oma panuse ja tiimitöö Sofiiaga ja ise saavutasin selge arusaama LLM-tehnoloogia tugevustest ja piirangutest. Arvan, et see töö annab praktilist väärtust arendajatele, kes soovivad mõista, kuidas kasutada keelemudeleid mitte ainult koodi genereerimiseks, vaid ka selle analüüsimiseks ja hindamiseks.

Lisa 25 – Küsimustik eksperdirühmale analüüsimiseks

4.1.1 Rakendus 1 - Elemendi eemaldamine massiivist

Eesmärk:

Kirjuta C# programm, mis võtab sisendiks täisarvude massiivi `nums` ja täisarvu `val`. Programm peab kohapeal (in-place) eemaldama kõik elemendid, mis on võrdsed väärtusega `val`. Pärast töötlemist peab massiivi `nums` esimesed `k` elementi sisaldama üksnes neid arve, mis ei võrdu `val`. Ülejäänud elemendid ja massiivi kogusuurus pole olulised. Funktsioon peab tagastama arvu `k` - ehk mitu elementi ei võrdu väärtusega `val`.

Programmi käitumine:

- Funktsiooni sisendiks on massiiv `nums` ja täisarv `val`.

- Programm eemaldab kõik valiga võrdsed elemendid ilma lisamassiive kasutamata.
- Pärast eemaldamist:
 - Esimesed k positsiooni massiivis sisaldavad kõiki ülejäänud arve (järjekord pole oluline).
 - Ülejäänud positsioonide sisu on ebaoluline.
- Funktsioon tagastab k.

Näide töövoost:

- Sisend: `nums = [3, 2, 2, 3]`, `val = 3`
 Väljund: `2`, `nums = [2, 2, _, _]`
- Sisend: `nums = [0, 1, 2, 2, 3, 0, 4, 2]`, `val = 2`
 Väljund: `5`, `nums = [0, 1, 4, 0, 3, _, _, _]`

Lisatingimused:

- $0 \leq \text{nums.length} \leq 100$
- $0 \leq \text{nums}[i] \leq 50$
- $0 \leq \text{val} \leq 100$
- Eemaldamine peab toimuma in-place – lisamälu, mille suurus sõltub `nums` pikkusest, ei tohi kasutada.
- Lahenduse korrektsust kontrollib kohandatud kohtunik, mis:
- Kutsub sinu funktsiooni: `k = removeElement(nums, val)`.
- Veendub, et `k` võrdub eeldatava tulemuse pikkusega.
- Sorteerib massiivi esimesed `k` elementi ja võrdleb neid eeldatava väljundiga.

LLM Model A Lahendus

```
public int RemoveElement(int[] nums, int val)
{
    int left = 0;
    int right = nums.Length - 1;

    while (left <= right)
    {
        if (nums[left] == val)
        {
```

```

        nums[left] = nums[right];
        right--;
    }
    else
    {
        left++;
    }
}

return left;
}

```

LLM Model B Lahendus

```

public class Solution {
    public int RemoveElement(int[] nums, int val) {
        int left = 0;
        int right = nums.Length - 1;
        while (left <= right) {
            if (nums[left] == val) {
                nums[left] = nums[right--];
            } else {
                left++;
            }
        }
        return right + 1;
    }
}

```

	Selged ja tähenduslikud nimed (Kas muutujate/meetodite nimed on selged ja eesmärgipärased?) (1–5)	Loogiline ülesehitus ja meetodite eraldus (Kas kood on jaotatud selgeteks funktsionaalseteks plokkideks / funktsioonideks?) (1–5)	Vorminduse ja stiili järjepidevus (1–5)	lakoonilisus (Kas ei ole üleliigset koodi, ei ole kordusi?) (1–5)	Kommentaaride eesmärgipärasus (1–5)
LLM Model A	5	5	5	5	Ei olnud kommentaare - 1
LLM Model B	5	5	5	5	Ei olnud kommentaare - 1

1. Kindlalt ei ole nõus
2. Ei ole nõus
3. Ei ole nõus ega ole eriarvamusel
4. Nõus
5. Kindlalt nõus

4.1.2 Rakendus 2 - Stringide korrutamine

Eesmärk:

Kirjuta C# programm, mis korrutab kaks mitte-negatiivset täisarvu, mis on esitatud stringidena (num1 ja num2). Tulemuseks peab samuti olema string, mis kujutab korrutise väärtust.

NB! Sisendeid ei tohi teisendada otse arvudeks (int.Parse, Convert.ToInt32 vms) ega kasutada sisseehitatud suurte arvude teeki (nt BigInteger).

Näide töövoost:

- Sisend: num1 = "2", num2 = "3"

Väljund: "6"

- Sisend: num1 = "123", num2 = "456"

Väljund: "56088"

Piirangud:

- $1 \leq \text{num1.Length}, \text{num2.Length} \leq 200$
- num1 ja num2 sisaldavad ainult numbreid 0–9.
- Kumbki string ei sisalda juhtivaid nulle, välja arvatud erandina number "0" ise.

LLM Model A Lahendus

```
public class Solution
{
    public string Multiply(string num1, string num2)
```

```

{
    if (num1 == "0" || num2 == "0")
        return "0";

    int n1 = num1.Length;
    int n2 = num2.Length;
    int[] result = new int[n1 + n2];

    for (int i = n1 - 1; i >= 0; i--)
    {
        int digit1 = num1[i] - '0';
        for (int j = n2 - 1; j >= 0; j--)
        {
            int digit2 = num2[j] - '0';

            int posLow = i + j + 1;
            int posHigh = i + j;

            int mul = digit1 * digit2 + result[posLow];
            result[posLow] = mul % 10;
            result[posHigh] += mul / 10;
        }
    }

    var sb = new System.Text.StringBuilder();
    bool leadingZero = true;
    foreach (var digit in result)
    {
        if (leadingZero && digit == 0)
            continue;

        leadingZero = false;
        sb.Append((char)(digit + '0'));
    }

    return sb.ToString();
}
}

```

LLM Model B Lahendus

```

public class Solution {
    public string Multiply(string num1, string num2) {
        if (num1 == "0" || num2 == "0") return "0";
    }
}

```

```

char[] arr1 = num1.ToCharArray();
char[] arr2 = num2.ToCharArray();
Array.Reverse(arr1);
Array.Reverse(arr2);

int[] product = new int[arr1.Length + arr2.Length];

for (int i = 0; i < arr1.Length; i++) {
    int digit1 = arr1[i] - '0';
    for (int j = 0; j < arr2.Length; j++) {
        int digit2 = arr2[j] - '0';
        product[i + j] += digit1 * digit2;
    }
}

int carry = 0;
for (int k = 0; k < product.Length; k++) {
    int total = product[k] + carry;
    product[k] = total % 10;
    carry = total / 10;
}

int start = product.Length - 1;
while (start >= 0 && product[start] == 0) start--;

StringBuilder result = new StringBuilder();
for (int i = start; i >= 0; i--) {
    result.Append(product[i]);
}

return result.ToString();
}
}

```

	Selged ja tähenduslikud nimed (1–5)	Loogiline ülesehitus ja meetodite eraldus (1–5)	Vorminduse ja stiili järgjepidevus (1–5)	lakoonilisus (1–5)	Kommentaaride eesmärgipärasus (1–5)
LLM Model A	5	5	5	5	Ei olnud komentaare - 1
LLM Model B	5	5	5	5	Ei olnud komentaare - 1

1. Kindlalt ei ole nõus
2. Ei ole nõus
3. Ei ole nõus ega ole eriarvamusel
4. Nõus
5. Kindlalt nõus

4.1.3 Rakendus 3 - Täisarvude jagamine

Eesmärk:

Kirjuta C# programm, mis jagab kaks täisarvu — dividend ja divisor — ilma kasutamata korrutamist, jagamist ega modulo-operaatorit. Tulemuseks peab olema täisarvuline jagatise väärtus, mis on trunkreeritud nulli suunas (ehk ilma murdosata). Ülesannes on vaja testida oskust lahendada probleeme piiratud tingimustes, kasutades madalama taseme loogikat nagu tsüklid, bitinihkeid ja loogilisi võtteid.

Näide töövoost:

- Sisend: dividend = 10, divisor = 3

Väljund: 3

Selgitus: $10 / 3 = 3.333\dots$, mis trunc'itakse 3-ks

- Sisend: dividend = 7, divisor = -3

Väljund: -2

Selgitus: $7 / -3 = -2.333\dots$, mis trunc'itakse -2-ks

Piirangud:

- $-2^{31} \leq \text{dividend}, \text{divisor} \leq 2^{31} - 1$ (ehk 32-bitiste allkiri arvude vahemik)
- $\text{divisor} \neq 0$ (jagamine nulliga on keelatud)
- Kui tulemus on suurem kui $2^{31} - 1$, tagasta $2^{31} - 1$
- Kui tulemus on väiksem kui -2^{31} , tagasta -2^{31}

LLM Model A Lahendus

```
public class Solution
{
    /// <summary>
    /// Jagab kaks 32-bitist allkirjaga täisarvu ilma * /
    %-ita
    /// ja kärbib tulemuse Int32 piiridesse.
    /// </summary>
    public int Divide(int dividend, int divisor)    // enam
    pole static
    {
        if (divisor == 0)
            throw new DivideByZeroException();

        // erand: Int32.MinValue / -1 ületaks vahemiku
        if (dividend == int.MinValue && divisor == -1)
            return int.MaxValue;

        bool negative = (dividend ^ divisor) < 0;

        long a = Math.Abs((long)dividend);
        long b = Math.Abs((long)divisor);

        int result = 0;

        for (int shift = 31; shift >= 0; shift--)
        {
            if ((a >> shift) >= b)
            {
                a -= b << shift;
                result += 1 << shift;
            }
        }

        return negative ? -result : result;
    }
}
```

LLM Model B Lahendus

```
public class Solution {
    public int Divide(int dividend, int divisor) {
        if (dividend == int.MinValue && divisor == -1) {
            return int.MaxValue;
        }
    }
}
```

```

    }

    int sign = (dividend < 0) == (divisor < 0) ? 1 : -1;

    if (dividend > 0) dividend = -dividend;
    if (divisor > 0) divisor = -divisor;

    int quotient = 0;

    while (dividend <= divisor) {
        int value = divisor;
        int power = 0;

        while (value >= (int.MinValue >> 1) && (value << 1)
>= dividend) {
            value <<= 1;
            power++;
        }

        quotient += (1 << power);
        dividend -= value;
    }

    return sign == 1 ? quotient : -quotient;
}
}

```

	Selged ja täenduslikud nimed (1–5)	Loogiline ülesehitus ja meetodite eraldis (1–5)	Vorminduse ja stiili järgipidevus (1–5)	lakoonilisus (1–5)	Kommentaaside eesmärgipärasus (1–5)
LLM Model A	5	5	5	5	5
LLM Model B	5	5	5	5	Ei olnud komentaare - 1

4.1.4 Rakendus 4 - Neljiku summa (4Sum)

Eesmärk:

Kirjuta C# programm, mis leiab kõik unikaalsed nelikute kombinatsioonid massiivist nums, mille elementide summa võrdub etteantud väärtusega target. Igas leitud nelikus

[nums[a], nums[b], nums[c], nums[d]] peavad kõik indeksid olema erinevad ($a \neq b \neq c \neq d$). Lahenduses tuleb vältida duplikaate — st üks ja sama nelik ei tohi korduda erinevates järjekordades. Ülesanne testib võimet kasutada sortimist, mitmetasandilisi tsükleid ja kahte osutit (two pointers) efektiivse lahenduse loomiseks.

Näide töövoost:

- Sisend: nums = [1, 0, -1, 0, -2, 2], target = 0
Väljund: [[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]
- Sisend: nums = [2, 2, 2, 2, 2], target = 8
Väljund: [[2, 2, 2, 2]]

Piirangud:

- $1 \leq \text{nums.Length} \leq 200$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- $-10^9 \leq \text{target} \leq 10^9$

LLM Model A Lahendus

```
public class Solution
{
    public IList<IList<int>> FourSum(int[] nums, int target)
    {
        var result = new List<IList<int>>();
        if (nums == null || nums.Length < 4)
            return result;

        Array.Sort(nums); // Sorteeri sisend massiiv

        int n = nums.Length;

        for (int i = 0; i < n - 3; i++)
```

```

{
    // Väldi duplikaate esimese numbri puhul
    if (i > 0 && nums[i] == nums[i - 1])
        continue;

    for (int j = i + 1; j < n - 2; j++)
    {
        // Väldi duplikaate teise numbri puhul
        if (j > i + 1 && nums[j] == nums[j - 1])
            continue;

        int left = j + 1;
        int right = n - 1;

        while (left < right)
        {
            long sum = (long)nums[i] + nums[j] +
nums[left] + nums[right];

            if (sum == target)
            {
                result.Add(new List<int> { nums[i],
nums[j], nums[left], nums[right] });

                // Liigu vasakult paremale ja väldi
duplikaate
                while (left < right && nums[left]
== nums[left + 1]) left++;
                while (left < right && nums[right]
== nums[right - 1]) right--;

                left++;
                right--;
            }
            else if (sum < target)
            {
                left++;
            }
            else // sum > target
            {
                right--;
            }
        }
    }
}

```

```

    }

    return result;
}
}

```

LLM Model B Lahendus

```

public class Solution {
    public IList<IList<int>> FourSum(int[] nums, int target)
    {
        IList<IList<int>> result = new List<IList<int>>();
        if (nums.Length < 4) return result;

        Array.Sort(nums);
        int n = nums.Length;

        for (int i = 0; i < n - 3; i++) {
            if (i > 0 && nums[i] == nums[i - 1]) continue;

            for (int j = i + 1; j < n - 2; j++) {
                if (j > i + 1 && nums[j] == nums[j - 1])
continue;

                int left = j + 1;
                int right = n - 1;

                while (left < right) {
                    long sum = (long)nums[i] + nums[j] +
nums[left] + nums[right];

                    if (sum == target) {
                        result.Add(new List<int> { nums[i],
nums[j], nums[left], nums[right] });

                        while (left < right && nums[left] ==
nums[left + 1]) left++;
                        while (left < right && nums[right]
== nums[right - 1]) right--;
                        left++;
                        right--;
                    }
                }
            }
        }
    }
}

```

```

        else if (sum < target) left++;
        else right--;
    }
}
}
return result;
}
}

```

	Selged ja tähenduslikud nimed (1–5)	Loogiline ülesehitus ja meetodite eraldus (1–5)	Vorminduse ja stiili järjepidevus (1–5)	lakoonilisus (1–5)	Kommentaaride eesmärgipärasus (1–5)
LLM Model A	5	5	5	5	5
LLM Model B	5	5	5	5	Ei olnud kommentaare - 1

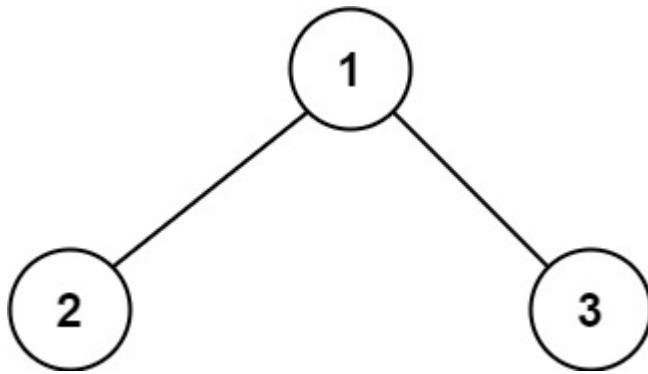
1. Kindlalt ei ole nõus
2. Ei ole nõus
3. Ei ole nõus ega ole eriarvamusel
4. Nõus
5. Kindlalt nõus

4.2 Algoritmilise ülesanne lahendamine - Binaarpuu maksimaalne tee summa

Eesmärk:

Kirjuta C# programm, mis leiab binaarse puu maksimaalse tee summa. Tee all mõeldakse järjestust tippudest, kus iga külgnevate tippude paar on omavahel ühendatud servaga. Tipp võib tee sees esineda ainult üks kord ning tee ei pea läbima juurt. Tee summa on kõigi selle tippude väärtuste summa.

Näide töövoost:



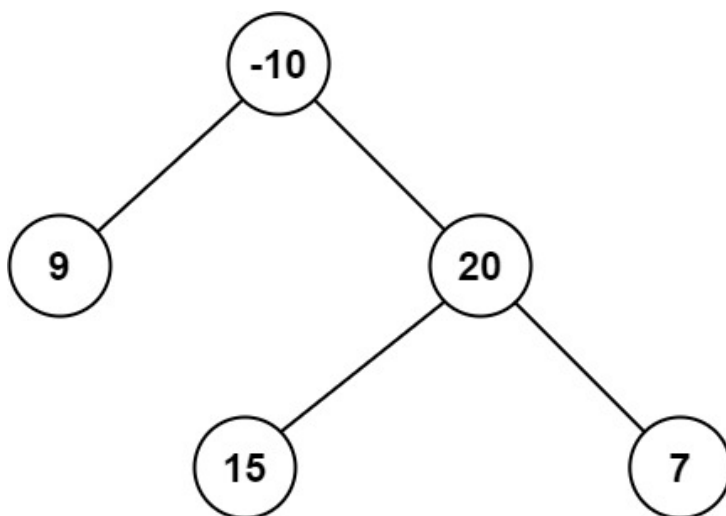
1. Sisend:

root = [1,2,3]

Väljund:

6

Seletus: optimaalne tee on 2 -> 1 -> 3, mille teesumma on $2 + 1 + 3 = 6$.



2. Sisend:

root = [-10, 9, 20, null, null, 15, 7]

Väljund:

42

Seletus: optimaalne tee on 15 -> 20 -> 7, mille teesumma on $15 + 20 + 7 = 42$.

Piirangud:

- Puu sõlmede arv on vahemikus $[1, 3 \cdot 10^4]$
- $-1000 \leq \text{Node.val} \leq 1000$

LLM Model A Lahendus

```
public class Solution {
    private int maxSum;

    public int MaxPathSum(TreeNode root) {
        maxSum = int.MinValue;
        MaxGain(root);
        return maxSum;
    }

    private int MaxGain(TreeNode node) {
        if (node == null) return 0;

        // Arvuta maksimaalne kasu vasakust ja paremast
        // lapsest (võttes 0 kui negatiivne)
        int leftGain = Math.Max(MaxGain(node.left), 0);
        int rightGain = Math.Max(MaxGain(node.right), 0);

        // Maksimaalne tee, mis läbib antud sõlme (võib olla
        // vasak + node + parem)
        int priceNewpath = node.val + leftGain + rightGain;

        // Uuenda globaalset maksimaalset väärtust
        maxSum = Math.Max(maxSum, priceNewpath);

        // Tagasta maksimaalne tee summeerimine ühelt poolt
        // + sõlme väärtus
        return node.val + Math.Max(leftGain, rightGain);
    }
}
```

LLM Model B Lahendus

```
public class Solution {
    private int _maxSum;
```

```

public int MaxPathSum(TreeNode root) {
    _maxSum = int.MinValue;
    CalculateMaxGain(root);
    return _maxSum;
}

private int CalculateMaxGain(TreeNode node) {
    if (node == null) return 0;

    int leftGain = Math.Max(CalculateMaxGain(node.left),
0);
    int rightGain =
Math.Max(CalculateMaxGain(node.right), 0);

    int currentPathSum = node.val + leftGain +
rightGain;
    _maxSum = Math.Max(_maxSum, currentPathSum);

    return node.val + Math.Max(leftGain, rightGain);
}
}

```

	Selged ja tähenduslikud nimed (1–5)	Loogiline ülesehitus ja meetodite eraldus (1–5)	Vorminduse ja stiili järgepidevus (1–5)	lakoonilisus (1–5)	Kommentaaride eesmärgipärasus (1–5)
LLM Model A	5	5	5	5	5
LLM Model B	5	5	5	5	Ei olnud komentaare - 1

1. Kindlalt ei ole nõus
2. Ei ole nõus
3. Ei ole nõus ega ole eriarvamusel
4. Nõus
5. Kindlalt nõus