Financira Evropska unija
NextGenerationEU

REPUBLIKA SLOVENIJA
MINISTRSTVO ZA VISOKO ŠOLSTVO, ZNANOST IN INOVACIJE

ARIS Javna agencija za znanstvenoraziskovalno in inovacijsko dejavnost Republike Slovenije

D6.2 - VeMo-IaC - v1

-

Prva različica programske opreme

31.8.2024

Podatki o dokumentu	
Različica	1.0
Pričakovan datum oddaje	31.8.2024
Datum oddaje	31.8.2024

RRP	6	**Oznaka izročka**	D6.2
Povezani izročki	D6.1	**Dostop**	Javno
Glavni urednik	Daniel Vladušič	**Glavni avtor**	Aljaž Potočnik
Ostali	Tomaž Martinčič		

Ključne besede
Programska oprema, Veliki jezikovni modeli (LLM), IaC

Kazalo

Kazalo Tabel

Kazalo Slik

Povzetek

Izroček poroča o štirih izbranih modelih (pristopih), izbranih hiperparametrih ter rezultatih v sklopu dela na RRP6 – VeMO-IaC. Pri slednjih se osredotočimo tudi na iskanje najbolj primernega modela za problem, ki ga rešujemo. Izbrali smo kompromis med velikostjo, hitrostjo in natančnostjo modela – skladno z uporabniškimi zahtevami.

Pri eksperimentih smo uporabili podatke ter znanje, ki smo ga pridobili pri sestavljanju učne množice, opisane v izročku D6.1. Za testiranje generirane kode smo uporabili tako produkt Spotter, ChatGPT 4 ter eksperte s področja infrastrukturne kode in jezika Ansible.

Z opravljenimi eksperimenti smo implementirali komponento produkta Spotter, ki omogoča generiranje kode. Ta komponenta je del produkta in kot taka del bodočih posodobitev, ki jih bomo razvili v okviru podprojekta RRP6 projekta Povejmo.

1 Uvod

S prihodom velikih (ali izjemno velikih) jezikovnih modelov (Large Language Models – LLM) se pričel nov razvojni cikel in ponovni razcvet področja umetne inteligence in povezanih tehnologij in aplikacij. Ti so usmerili pozornost na pristop v globokem učenju (ang. Deep Learning), poimenovan Transformer[1]. Izkazalo se je, da pristop ni samo uporaben samo pri velikih jezikovnih modelih, temveč tudi na drugih področjih (npr. računalniški vid), kar v pomeni, da se bo pristop razvijal še naprej v prihodnosti. Če se osredočimo samo na LLM, vidimo, da za gradnjo takšnih modelov potrebujemo veliko podatkov. Upravljanje tako velike količine podatkov (velika prostorska zahtevnost) je v domeni redkih podjetij, ki imajo dovolj finančnih sredstev, da zberejo in hranijo takšno količino podatkov.

Ker je kvaliteta splošnih velikih jezikovnih modelov v večini primerov direktno korelirana s količino podatkov, je posledično tudi učenje s takšno količino podatkov izjemno drago. Učenje je računsko zelo zahtevno in zahteva izjemno drago strojno opremo, ter v drugi fazi tudi ogromno količino človeških naporov.

Za manjša podjetja se uporablja pristop prednaučenih modelov (ang.: GPT - Generative pre-trained transformer), ki se zgolj »doučijo« znanj na domeni, jeziku, ki jo želimo uporabljati. Takšen pristop priporočajo tudi velika svetovalna podjetja (npr. Gartner). Ta korak se v jeziku umetne inteligence imenuje v angleščini »focusing the GPT model«. Le-ta nas, kot malo in srednje veliko podjetje (MSP), zanima. Za dosego tega uporabljamo znanje, pristope, ki se v okviru projekta izvajajo v horizontalnem sklopu RRP2 – SloLLaMai – Odprtodostopni računsko učinkoviti modeli za slovenščino. Poudarjamo, da lastnosti kot so majhnost učne množice, omejene zmogljivosti hranjenja podatkov in omejene računske zmogljivost vodijo v podobnost v delu in raziskavah v RRP2 ter RRP6 – Prilagoditev velikih jezikovnih modelov za generiranje infrastrukturnih opisov v kodi (VeMo-IaC).

Naš cilj je zgraditi jezikovni model, ki je uporaben primarno tudi za produkt Spotter[2] pri učinkovitem **generiranju infrastrukturne kode**, kjer bomo tehnologije za računsko učinkovite velike jezikovne modele in razvite cevovode za izgradnjo ukaznih in dialoških učnih množic uporabili za generiranje opisa računalniške infrastrukture v programski kodi. **Trenutno se kot glavno ciljno orodje uporablja jezik Ansible**[3], za katerega skrbi eno večjih podjetij, ki se ukvarja s z odprto kodo – Red Hat.

[1] https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
[2] https://steampunk.si/spotter/
[3] https://www.redhat.com/en/technologies/management/ansible

Ansible je zmogljivo orodje za avtomatizacijo upravljanja z računalniško infrastrukturo. Kot vsako programsko orodje oz. jezik, je Ansible organiziran v hierarhično strukturo. Glavne komponente, nevedene po velikosti od največje do najmanjše, so kolekcije (ang. collection), modul (ang. module) in naloga (ang. task). To pomeni da npr., ena kolekcija združuje več modulov, en modul pa več nalog. Pomembno je tudi poudariti, da so v modulih definicije in izvedba nalog. Slednje izvajamo s pomočjo t.i. playbook-ov. Slednji so idempotentni, kar pomeni, da večkratna izvedba ne povzroči vsakokratnih sprememb, temveč samo prva.

Če organizacijo povzamemo:

- Kolekcije: združujejo povezano vsebino, kot so moduli in vloge.
- Moduli so posamezne enote dela, ki izvajajo specifične naloge.
- Naloge so akcije, ki jih želimo izvesti na svojih upravljanih sistemih, opredeljene v "playbookih" ali vlogah, in kličejo module s specifičnimi parametri.

Vlog in detajlov, ki se tičejo programskega jezika Ansible, na tem mestu ne opisujemo. Za izroček je pomembno dejstvo, da Ansible vsebuje kolekcije, ki upravljajo z zelo raznorodnimi sistemi (npr. požarnimi zidovi, upravljanjem virtualnih računalnikov, omrežji, itd.). Iz tega sledi, da je potrebno pri izboru podatkov paziti tudi na to, da so upravljani sistemi podobni, oz. da se ukvarjajo s približno enakimi nalogami. Sam programski jezik je kombinacija programskih jezikov Python[4] ter podatkov, ki so zapisane v skriptah v jeziku YAML[5].

XLAB bo velike jezikovne modele uporabil za doseganje večje podatkovne učinkovitosti ter predvsem večje pravilnosti in robustnejšega delovanja pri generiranju izvorne kode in dokumentacije. V projektu se osredotočamo na zagotavljanje zanesljivosti, robustnosti in posledično varnosti generirane kode preko treh področij: (a) izboljšava jezikov Infrastructure-as-Code (IaC), ki že vsebujejo orodja z generiranje kode (npr. Ansible Wisdom oz. Ansible Lightspeed), vendar dopuščajo preveč napak in (b) zagotavljanje robustnosti, varnosti kode, kar trenutno zagotavljajo le s skeniranjem kode.

1.1 Namen dokumenta

Izroček D6.2 je produkt eksperimentalnega razvoja v sklopu RRP6, kjer podjetje skuša na podlagi podatkov, opisanih v izročku D6.1[6] ter z znanjem, ki ga pridobiva v celotnem projektu (podjetje XLAB sodeluje v dveh podprojektih – RRP2 ter RRP6), izbrati prednaučene modele (GPT), ki ustrezajo tako zbranim podatkom ter ostalim želenim lastnostim (majhna poraba računskih zmogljivosti, hitrost delovanja, itn.). Na tem mestu omenimo, da je podprojekt

[4] https://www.python.org
[5] https://yaml.org
[6] Izroček projekta Povejmo: D6.1 - Dokumentacija in načrt rabe podatkov in pristopov

RRP2, eden krovnih, horizontalnih podprojektov, ki ga vodijo vodilni strokovnjaki za umetno inteligenco in skozi nova spoznanja učijo podjetja v projektu, o tem, kateri so primerni pristopi za reševanje njihovega problema.

Namen izročka je opisati izbor in predvsem razloge za izbor modelov, naša pričakovanja ter tudi rezultate, ki smo jih z uporabo različnih množic in modelov z eksperimenti dosegli. Tako izbran model ovrednotimo tudi skozi kontekst produkta Spotter, ki ga podjetje razvija. Najboljši rezultat je tisti, ki je interno že uporabljen v produktu Spotter, v raziskovalnem in razvijalskem smislu ga bomo pa dodelovali naprej.

1.2 Povezava z drugimi deli projekta

Eksperimentalni razvoj projekta RRP6 je tesno povezan s spoznanji, ki jih skupaj s partnerjem FRI pridobivamo v sklopu RRP2. Le-ta se tiče postopkov, kriterijev za izbor podatkov, izborov samih prednaučenih modelov ter postopkov, kako zmanjšati model, da je obvladljiv in da lahko teče na manj zmogljivi strojni opremi.

Povezava sklopa RRP6 z ostalimi deli projekta je omejena le na zgoraj omenjeno povezavo – podprojekt RRP2 je horizontalen in je, v skladu z razvojnim načrtom projekta, povsem ločen od eksperimentalnega razvoja drugih podjetij oz. deležnikov v projektu.

1.3 Struktura dokumenta

Dokument je sestavljen iz treh gradnikov:

- **Opis izbranih metod – poglavje 2 –** kjer opišemo izbrane pristope, njihove relevantne značilnosti in tudi pričakovanja, ki jih od izbranih metod imamo. Opišemo tudi pridobljene rezultate.
- **Vrednotenje rezultatov – poglavje 3 –** je poglavje, kjer opišemo rezultate, ki smo jih pridobili z eksperimenti.
- **Vrednotenje rezultatov skozi produkt Spotter – poglavje 4 –** se osredotoči na to, kako je vrednotenje rezultatov odvisno od konteksta, ki je v našem primeru produkt Spotter.
- **Najboljši rezultat - prva različica programske opreme – poglavje 5 –** opiše prvo različico programske opreme, rezultat podprojekta RRP6 v projektu Povejmo.

Na tem mestu je potrebno poudariti, da se področje izjemno hitro razvija, kar pomeni, da je korak izbor metod potrebno ponavljati in sproti ocenjevati ali so nove metode uspešnejše pri učenju.

2 Izbrani pristopi in eksperimenti

2.1 Opis izbranih pristopov

V sklopu naših raziskav smo preizkusili različne pristope k reševanju zastavljenega problema. V nadaljevanju so opisane metode, ki smo jih uporabili za razvoj modela, ki je sposoben generirati naloge Ansible iz naravnega jezika.

Naš prvi pristop je temeljil na modelu CodeGen-350M-multi[7] in Steampunk učni množici, ki smo jo opisali v poglavju 5.2 v izročku D6.1. Implementirali smo nadzorovano učenje (ang. supervised learning). Prednost tega pristopa je hitra implementacija in posledično hitro pridobivanje prvih rezultatov. Zaradi manjšega števila parametrov je model hitro naučen, kar nam je omogočilo hitro oceno kompleksnosti našega problema.

Ta model nudi dober vpogled v naravo problema, vendar moramo biti realistični glede pričakovanj. Ker je model eden manj kompleksnih v našem naboru, nismo pričakovali vrhunskih rezultatov; eksperimente s tem modelom vidimo kot začetni korak, ki nam služi kot odskočna deska za nadaljnje, bolj sofisticirane pristope, ki jih raziskujemo v nadaljevanju našega projekta. Še vedno velja, da se MSPjem najbolj izplača preverjati prednaučene modele (GPT), hkrati pa se področje tako hitro razvija, da so novi modeli na voljo praktično vsak teden.

S takšnim pristopom pridobimo vpogled v problem, ki ga rešujemo, in usmerjamo naše odločitve pri izbiri in optimizaciji metod za prihodnje eksperimente. Analiza rezultatov pristopa je bistvena za razumevanje, kako različne tehnike učenja vplivajo na zmogljivost modela pri našem specifičnem problemu generiranja Ansible nalog.

V drugem pristopu smo naredili korak naprej in uporabili večji, zmogljivejši model: StarCoder2-3b[8]. Tudi tu smo uporabili nadzorovano učenje in podatke iz Steampunk množice. Ker je model kompleksnejši, smo od njega upravičeno pričakovali boljše delovanje v primerjavi s prejšnjim, manjšim modelom, vendar se zavedamo, da je izboljšano delovanje posledica kompromisov: počasnejše učenje in počasnejša inferenca. Kljub temu menimo, da je ta pristop ključen za razumevanje razmerja med velikostjo modela in njegovo učinkovitostjo pri našem specifičnem problemu. Poleg tega smo v tem eksperimentu raziskali vpliv različnih oblik vhodnih navodil na kakovost generiranih Ansible nalog. Ta vidik

[7] https://huggingface.co/Salesforce/codegen-350M-multi
[8] https://huggingface.co/bigcode/starcoder2-3b

pomaga optimizirati interakcijo med uporabnikom in modelom ter izboljšati uporabniško izkušnjo.

Naš tretji pristop gradi na spoznanjih prejšnjih eksperimentov, vendar s pomembno razliko v podatkovni osnovi. Uporabili smo enake koncepte kot pri drugem pristopu, vendar smo tokrat združili obe učni podatkovni množici (Steampunk ter GitHub učna množica – opisani v izročku D6.1) v eno učno množico. Ta pristop prinaša potencialne prednosti in izzive. Po eni strani lahko večja in bolj raznolika podatkovna množica vodi do boljšega razumevanja Ansible kode in večje fleksibilnosti modela. Po drugi strani pa obstaja tveganje, da vključitev potencialno manj kakovostnih podatkov lahko negativno vpliva na natančnost in zanesljivost modela. Tudi ta eksperiment pripomore k razumevanju, kako obseg in kakovost podatkov vplivata na zmogljivost našega sistema.

Za zaključek našega niza eksperimentov smo uporabili najnaprednejši model CodeLlama-7b-Instruct-hf[9]. Ta pristop predstavlja ima potencial za doseganje najboljših rezultatov, saj je tudi najbolj kompleksen. Pri tem poudarjamo, da se zavedamo, da večja kompleksnost modela prinaša tudi večja tveganja. Obstaja možnost, da bo model zaradi svoje sofisticiranosti zajel poleg koristnih vzorcev tudi šum v podatkih, kar bi lahko vodilo do pretiranega prilagajanja (ang. overfitting). Eksperiment je časovno najzahtevnejši, tako v fazi učenja, kot v fazi inference. V povezavi z eksperimenti z enostavnejšimi modeli pridobivamo znanje o optimalnem ravnovesju med kompleksnostjo prednaučenega modela, velikostjo učne množice in kakovostjo končnih rezultatov za naš specifični problem generiranja Ansible nalog iz naravnega jezika.

Skozi vse te pristope smo spremljali in analizirali rezultate, pri čemer smo upoštevali tako kvantitativne metrike kot kvalitativno oceno generiranih Ansible nalog.

Naš cilj je ne le izbrati najboljši model, temveč tudi poglobiti naše razumevanje dinamike učenja in generiranja pri tovrstnih specializiranih jezikovnih nalogah in upoštevati omejitve ciljnih sistemov, na katerih teče produkt Spotter oz. kot poročajo stranke, ki uporabljajo produkt. To znanje predstavlja veliko dodano vrednost pri nadaljnjem razvoju in optimizaciji sistemov za avtomatizacijo IT infrastrukture.

2.2 Eksperimenti opravljeni z izbranimi pristopi

Pri vseh eksperimentih smo z namenom, da bi pospešili proces učenja implementirali LoRA (Low-Rank Adaptation) optimizacijo. Ta tehnika nam je omogočila hitrejše in učinkovitejše

[9] https://huggingface.co/meta-llama/CodeLlama-7b-Instruct-hf

fino prilagajanje modela, ob hkratnem ohranjanju visoke kakovosti rezultatov. V naslednjih podpoglavjih smo opisali konfiguracije posameznih eksperimentov, ki smo jih izvedli.

2.2.1 CodeGen-350-multi + Steampunk

V našem začetnem eksperimentu smo se lotili izziva z uporabo CodeGen-350M-multi modela, ki smo ga učili na učni množici Steampunk. Ta pristop je predstavljal naš prvi korak k reševanju zastavljenega problema.

Za optimalno učenje smo prilagodili format učne množice, da je ustrezal zahtevam nadzorovanega učenja. Ta prilagoditev je bila ključna za učinkovito izkoriščanje zmogljivosti CodeGen-350-multi modela in zagotavljanje, da lahko model ustrezno interpretira in se uči iz predstavljenih podatkov.

Po temeljitem preizkušanju smo določili optimalno kombinacijo hiperparametrov:

- Stopnja učenja: 3e-5
- Velikost gruče: 32
- Število epoh: 2
- Razporejevalnik učne stopnje: kosinusni

Ta konfiguracija je dala najboljše rezultate. Dosežena hitrost učenja je znašala 2.668 korakov na sekundo.

2.2.2 StarCoder2-3b + Steampunk

V tem eksperimentu smo se posvetili drugemu pristopu k reševanju problema. Ponovno smo prilagodili format učne množice, da bi ustrezal našemu novemu pristopu. Ta metoda se je izkazala za zelo obetavno, zato smo ji namenili več pozornosti in izvedli več podobnih poizkusov.

Spodnja tabela prikazuje različne hiperparametre, ki smo jih uporabili pri treh podpoizkusih:

Tabela 1: Iskanje ustreznih parametrov pri kombinaciji podatkov in metode: StarCoder2-3b + Steampunk

Poizkus	**1.**	**2.**	**3.**
Stopnja učenja(learning rate)	2e-5	3e-5	3e-5
Število epoh(epochs)	1	0.574	1
Elementov v gruči(batch size)	16	32	32
Testna množica	**7.233**	6.9	7.033

V teh treh preizkusih smo raziskovali učinke različnih nastavitev hiperparametrov:

1. Stopnja učenja: Preizkusili smo počasnejšo (2e-5) in hitrejšo (3e-5) stopnjo učenja, da bi ugotovili optimalno hitrost prilagajanja modela.

2. Število epoh: V vseh treh primerih smo uporabili eno epoho, kar nakazuje, da smo se osredotočili na učinkovitost učenja v enem prehodu skozi podatke.
3. Velikost gruče: Primerjali smo manjšo (16) in večjo (32) velikost gruče, da bi razumeli vpliv na hitrost in kakovost učenja.

Rezultati na testni množici kažejo, da je prvi poizkus z najnižjo stopnjo učenja in najmanjšo velikostjo gruče dosegel najboljši rezultat (7.233). To nakazuje, da je v tem primeru počasnejše in bolj postopno učenje privedlo do boljše generalizacije modela.

Ti preizkusi nam omogočajo boljše razumevanje vpliva različnih hiperparametrov na uspešnost modela in predstavljajo temelj za nadaljnjo optimizacijo našega pristopa.

2.2.3 StarCoder2-3b + razširjena učna množica

V tem eksperimentu smo model izpostavili obsežnejši učni množici Ansible kode, s ciljem izboljšanja njegovega razumevanja in zmogljivosti.

Za optimizacijo učnega procesa smo izbrali naslednje hiperparametre:

1. Začetna stopnja učenja: 2e-5
2. Razporejevalnik učne stopnje: kosinusni
3. Velikost gruče: 16 elementov
4. Število epoh: 1

Nizka začetna stopnja učenja (2e-5) omogoča stabilno prilagajanje modela. Kosinusni razporejevalnik dinamično prilagaja to stopnjo skozi čas, spodbujujoč hitrejšo začetno konvergenco in fino prilagajanje proti koncu. Velikost gruče 16 elementov predstavlja ravnotežje med učinkovitostjo in raznolikostjo podatkov v vsaki iteraciji. Ena epoha nakazuje, da smo predpostavili zadostno obsežnost in raznolikost učne množice za učinkovito učenje v enem prehodu.

Ta konfiguracija je bila izbrana z namenom optimalne izrabe razširjene učne množice, ob hkratnem ohranjanju stabilnosti učenja in preprečevanju prekomernega prilagajanja.

2.2.4 CodeLlama-7b + Steampunk

V tem eksperimentu smo uporabili najkompleksnejši model z namenom doseganja optimalnih rezultatov pri razumevanju in generiranju Ansible kode. Zaradi visoke računske zahtevnosti in večje porabe pomnilnika smo prilagodili naše pristope:

Ključni hiperparametri:

1. Razporejevalnik učne stopnje: kosinusni
2. Velikost gruče: 8 elementov

3. Število epoh: 1
4. Začetna stopnja učenja: 2e-5

Izbira manjše velikosti gruče (8 elementov) je bila nujna zaradi omejitev pomnilnika, kar je manj kot pri drugih konfiguracijah. Ta prilagoditev omogoča učinkovito delo s kompleksnim modelom kljub računskim omejitvam.

Kosinusni razporejevalnik učne stopnje omogoča dinamično prilagajanje hitrosti učenja, začenši z višjo vrednostjo in postopnim zmanjševanjem. Nizka začetna stopnja učenja (2e-5) in ena epoha kažeta na previdno in učinkovito strategijo učenja.

Ta konfiguracija je skrbno izbrana za optimizacijo učinkovitosti našega najnaprednejšega modela, upoštevajoč omejitve računskih virov in zahtevnost naloge razumevanja Ansible kode. Model je na naši testni množici dosegel rezultat 7,48.

2.3 Rezultati

V tem poglavju na enem mestu prikažemo ključne nastavitve (hipeparametre) naših različnih pristopov v povezavi z metrikami (rezultati), ki jih te metode dosegajo.

Za vrednotenje rezultatov smo uporabili več pristopov, ki vključujejo kvantitativne metrike in vizualne analize. Evalvacija je temeljila na naslednjih ključnih metrikah: minimum kriterijske funkcije (Eval Loss), ki smo jo uporabili za merjenje natančnosti modelov med učenjem; perpleksnost, s katero smo merili, kako dobro modeli napovedujejo verjetnost naslednje besede v sekvenci; hitrost učenja (primeri na sekundo), s katero smo ocenili računsko učinkovitost modelov; ter vrednosti gradientov, s katerimi smo analizirali stabilnost učnega procesa skozi čas.

Primerjava pristopov je prikazana v Tabela 2.

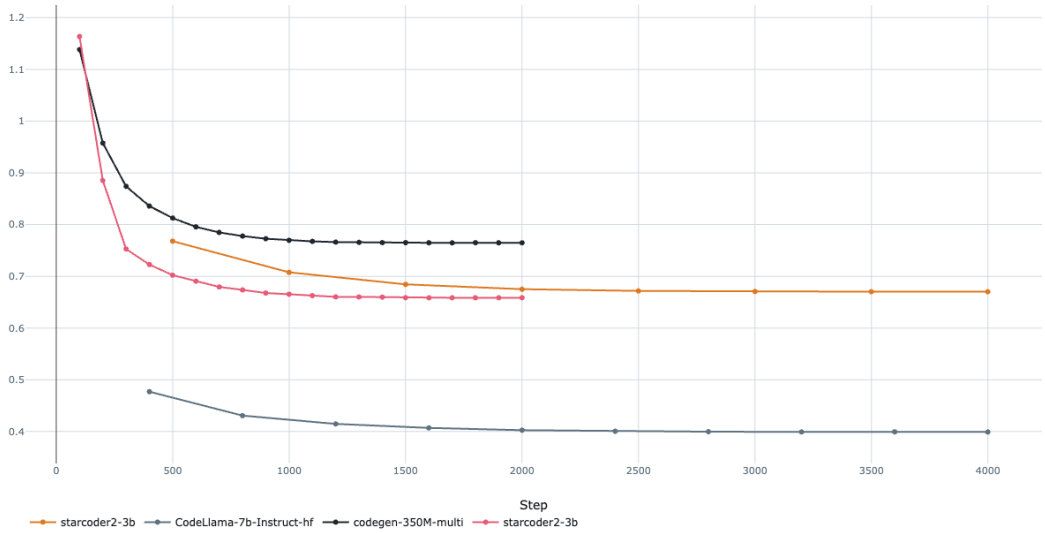
Tabela 2: Primerjava ključnih metrik vseh eksperimentov

Barva na grafu	Model	Učna množica	Stopnja učenja	Št. epoh	Velikost gruče	Eval loss (najnižji)	Perpleksnost (najnižja)	Hitrost učenja (primerov na	Rezultat na testni množici
Črna	CodeGen-350-multi	Steampunk	3e-5	2	32	0,76	2,14	**64,1**	4,9
Roza	StarCoder2-3b	Steampunk	2e-5	1	16	0,65	1,93	26,9	7,23

Rumena	StarCoder2-3b	Steampunk + GitHub	2e-5	1	16	0,67	1,95	41,6	6,8
Siva	CodeLlama-7b	Steampunk	2e-5	1	8	**0,39**	**1,49**	15,3	**7,48**

V nadaljevanju prikazujemo tudi rezultate treh ključnih metrik pri vseh izbranih metodah skozi čas (korake). Za vizualizacijo rezultatov smo uporabili grafe, ki prikazujejo evalvacijsko izgubo, perpleksnost in vrednosti gradientov skozi čas učenja. Te vizualizacije so nam omogočile boljše razumevanje trendov in stabilnosti modelov.

Prikaz kriterijske funkcije (Eval Loss) v odvisnosti od korakov učenja, perpleksnosti ter padanja gradientov je na Slika 1, Slika 2 ter Slika 3, po vrsti. Pod vsako sliko so zapisane tudi ključne ugotovitve, ki smo jih ob pregledu zaznali.

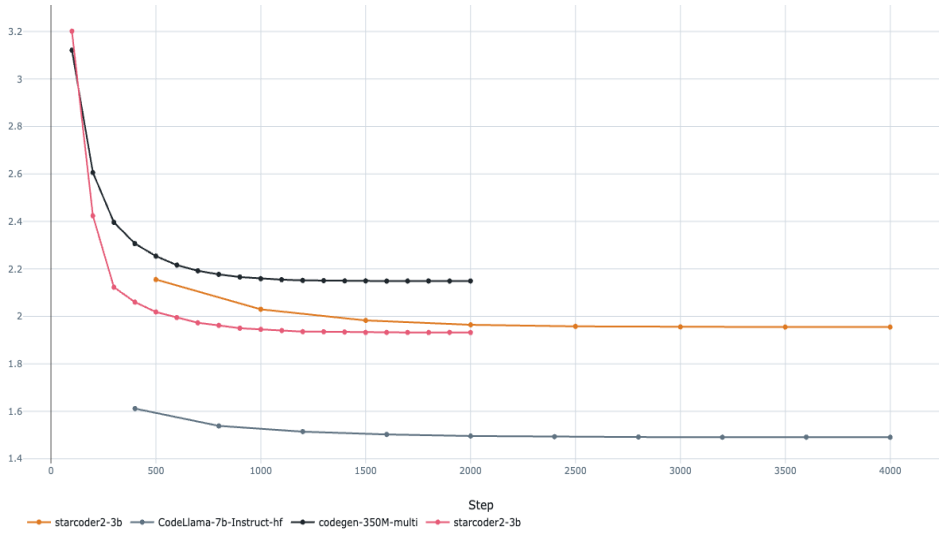


Slika 1: Potek kriterijske funkcije na evalvacijski množici za štiri različne modele skozi čas učenja.

Graf na sliki 1 prikazuje vrednosti eval loss na vertikalni osi in število korakov učenja na horizontalni osi.

Ključna opažanja:

- CodeLlama-7b (siva linija) dosega najnižji eval loss med vsemi modeli.
- StarCoder2-3b modeli (roza in rumena linija) kažejo podobno uspešnost.
- CodeGen-350-multi (črna linija) ima najvišji eval loss med prikazanimi modeli.
- Vsi modeli kažejo trend zmanjševanja eval loss skozi čas učenja.



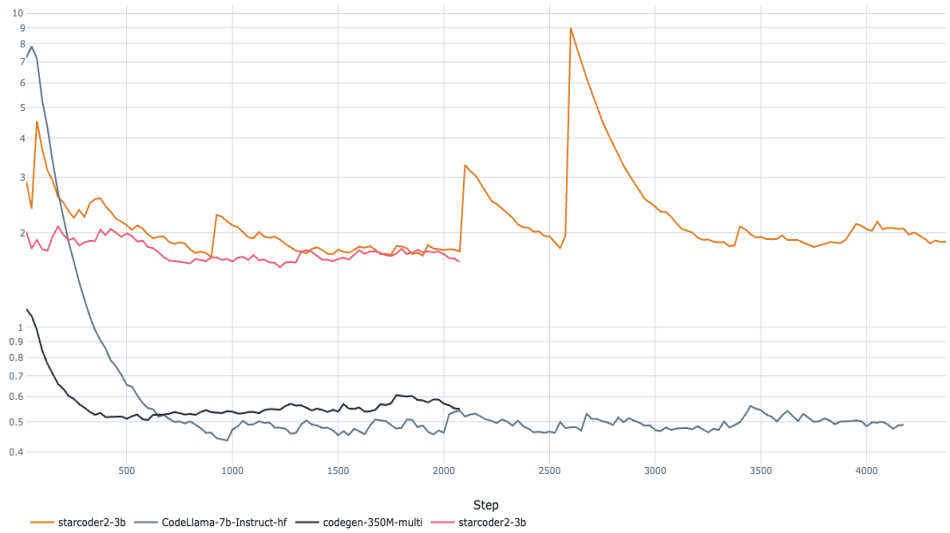
Slika 2: Perpleksnosti[10] za štiri različne modele skozi čas učenja.

Na grafu na sliki 2 so prikazane vrednosti perpleksnosti na vertikalni osi in število korakov učenja (do 4000) na horizontalni osi.

Ključna opažanja:

- CodeLlama-7b-Instruct-hf (siva linija) dosega najnižjo perpleksnost med vsemi modeli.
- Starcoder2-3b modeli (oranžna in roza linija) kažejo podobno uspešnost, z nekoliko nižjo perpleksnostjo za roza linijo.
- Codegen-350M-multi (črna linija) ima najvišjo perpleksnost med prikazanimi modeli.
- Vsi modeli kažejo trend zmanjševanja perpleksnosti skozi čas učenja.
- Končne vrednosti perpleksnosti se ujemajo s podatki v Tabeli 1, kjer je najnižja perpleksnost zabeležena za CodeLlama-7b (1,49).

[10] Perpleksnost je ključna mera, ki se uporablja za ocenjevanje, kako učinkovito jezikovni model napoveduje zaporedja besedila, pri čemer nižje vrednosti nakazujejo boljšo učinkovitost.



Slika 3: Vrednosti gradientov v odvisnosti od korakov učenja za različne modele.

Vrednosti gradientov na sliki 3 so prikazane na vertikalni osi, število korakov učenja pa na horizontalni osi.

Ključna opažanja:

- Vsi modeli kažejo velike fluktuacije v vrednostih gradientov skozi čas učenja.
- Starcoder2-3b (oranžna linija) kaže najvišje vrhove in največje fluktuacije med vsemi modeli.
- CodeLlama-7b-Instruct-hf (siva linija) in codegen-350M-multi (črna linija) kažeta najbolj stabilne vrednosti gradientov z manjšimi fluktuacijami.
- Starcoder2-3b (roza linija) kaže vmesno stopnjo fluktuacij, manj izrazito kot oranžna linija istega modela.
- Proti koncu učnega procesa se vrednosti gradientov za vse modele nekoliko stabilizirajo, vendar še vedno kažejo določeno stopnjo variabilnosti.

3 Vrednotenje rezultatov eksperimentov

Analiza relativne uspešnosti različnih modelov je temelj za razumevanje njihovih prednosti in slabosti. Na podlagi kriterijske funkcije, perpleksnosti in rezultatov na naši testni množici lahko izpostavimo, da je CodeLlama-7b dosegel najboljše rezultate pri vseh metrikah, z najnižjo kriterijsko funkcijo (0,39) in perpleksnostjo (1,49), kar kaže na visoko natančnost modela. Po kvaliteti učenja so sledili StarCoder2-3b modeli, pri čemer je različica z dodatnimi GitHub podatki dosegla malce slabše rezultate kot različica brez. CodeGen-350-multi je imel najvišjo kriterijsko funkcijo (0,76) in perpleksnost (2,14), kar kaže na slabšo uspešnost v primerjavi z drugimi modeli. Oba ekstrema (najboljši in najslabši model) smo pričakovali, kar smo zapisali tudi pri opisu samega modela.

Razlogi za te razlike v uspešnosti lahko vključujejo velikost modela in arhitekturo. CodeLlama-7b je večji model z naprednejšo arhitekturo, kar mu omogoča boljše rezultate, vendar na račun daljšega časa učenja in inference. Pomembno je poudariti, da bolj kompleksni modeli, kot je CodeLlama-7b, zahtevajo daljši čas inference, kar vpliva na njihovo uporabnost v aplikacijah, kjer je hitrost odziva ključnega pomena.

Pri primerjavi rezultatov StarCoder2-3b na različnih učnih množicah lahko opazimo, da razširjena učna množica ni izboljšala uspešnosti modela, kar je razvidno iz rezultatov na testni množici.

Trendi v grafih kriterijske funkcije in perpleksnosti kažejo, da je CodeLlama-7b najhitreje konvergiral z najnižjo evalvacijsko kriterijsko funkcijo in perpleksnostjo skozi celoten učni proces. Modeli StarCoder2-3b so pokazali podobne trende, pri čemer je različica z GitHub podatki dosegla nekoliko slabše rezultate. CodeGen-350-multi je imel najpočasnejšo konvergenco, z najvišjimi vrednostmi kriterijske funkcije in perpleksnosti.

Graf vrednosti gradientov kaže, da je StarCoder2-3b imel najvišje vrhove in največje fluktuacije, kar nakazuje na potencialno nestabilnost v učnem procesu. CodeLlama-7b in CodeGen-350-multi sta pokazala bolj stabilne vrednosti gradientov z manjšimi fluktuacijami, kar kaže na bolj stabilen učni proces.

Primerjava hitrosti učenja z doseženo uspešnostjo pokaže, da je CodeGen-350-multi dosegel najvišjo hitrost učenja, vendar na račun slabših vrednosti (višjih) kriterijske funkcije in perpleksnosti. CodeLlama-7b je imel najnižjo hitrost učenja in najdaljši čas inference, vendar najboljše rezultate, kar kaže na kompromis med hitrostjo, časom inference in kakovostjo rezultatov. StarCoder2-3b modeli so dosegli zmerne hitrosti učenja in rezultate, z razumnim in uporabnim časom inference.

Analiza hiperparametrov in njihovega vpliva na uspešnost modelov kaže njihov pomen pri učenju, zlasti ob upoštevanju, da so bile za vsak model izbrane najboljše najdene nastavitve. Ta pristop nam omogoča boljši vpogled v optimalne konfiguracije za posamezne arhitekture modelov.

Za CodeLlama-7b se je izkazalo, da dosega najboljše rezultate z relativno konzervativnimi nastavitvami: nižja stopnja učenja (2e-5), ena epoha in manjša velikost gruče (8). Te nastavitve so omogočile modelu, da doseže najboljši rezultat na testni množici (7,48), kljub najnižji hitrosti učenja. To nakazuje, da CodeLlama-7b zaradi svoje napredne arhitekture in velikosti lahko učinkovito izkoristi podatke tudi z manj agresivnimi parametri učenja.

StarCoder2-3b modeli so pokazali optimalne rezultate s srednjimi nastavitvami: stopnja učenja 2e-5, ena epoha in srednja velikost gruče (16). Zanimivo je, da sta obe različici StarCoder2-3b (z in brez dodatnih GitHub podatkov) dosegli podobne rezultate s temi nastavitvami, kar kaže na robustnost teh parametrov za to arhitekturo.

CodeGen-350-multi je za doseganje svojih najboljših rezultatov zahteval bolj agresivne nastavitve: višjo stopnjo učenja (3e-5), dve epohi in večjo velikost gruče (32). Kljub temu je dosegel najnižji rezultat na testni množici, vendar najvišjo hitrost učenja. To nakazuje, da model morda zahteva intenzivnejše učenje za optimizacijo svoje zmogljivosti, vendar še vedno zaostaja za novejšimi arhitekturami.

Te ugotovitve poudarjajo, da optimalni hiperparametri niso univerzalni, ampak so specifični za posamezne arhitekture modelov. Novejši in večji modeli, kot je CodeLlama-7b, lahko dosežejo vrhunske rezultate z bolj konzervativnimi nastavitvami, medtem ko manjši ali starejši modeli morda potrebujejo bolj agresivne pristope k učenju za doseganje svojih najboljših rezultatov.

Pomembno je tudi upoštevati, da kljub optimizaciji hiperparametrov za vsak model obstajajo inherentne razlike v zmogljivosti, ki izhajajo iz arhitekture in velikosti modela. To poudarja pomen napredka v arhitekturi modelov, ne le optimizacije hiperparametrov.

Najpomembnejše ugotovitve iz analize vključujejo, da je CodeLlama-7b pokazal najboljše rezultate v smislu natančnosti in stabilnosti, vendar z najnižjo hitrostjo učenja in najdaljšim časom inference. Dodajanje dodatnih podatkov je poslabšalo uspešnost StarCoder2-3b modela, kar nakazuje na pomembnost kvalitetnih učnih množic. Različne konfiguracije hiperparametrov imajo pomemben vpliv na rezultate.

Implikacije za nadaljnji razvoj in uporabo modelov vključujejo potrebo po iskanju ravnotežja med hitrostjo učenja, časom inference in natančnostjo rezultatov. Pri izbiri modela za specifično aplikacijo je treba upoštevati zahteve glede hitrosti odziva in natančnosti.

Poudarek ostaja na uporabi obsežnih in raznovrstnih podatkov za izboljšanje uspešnosti modelov ter na nadaljnjih raziskavah na področju optimizacije hiperparametrov in arhitektur za doseganje boljšega razmerja med uspešnostjo in hitrostjo inference.

4 Vrednotenje rezultatov v kontekstu produkta Spotter

Poleg teh standardnih metrik, ki jih opisujemo v prejšnjih poglavjih, smo uporabili tudi naš lastni evalvacijski postopek z našo testno množico, ki je podrobneje predstavljen v poglavju 7.2 v izročku D6.1. Ta postopek nam je omogočil bolj specifično oceno uspešnosti modelov glede na naše posebne zahteve in pričakovanja.

Rezultati te študije imajo neposreden vpliv na razvoj in izboljšanje našega produkta na več načinov. Najboljši rezultati, doseženi s CodeLlama-7b-Instruct-hf, kažejo na potencial za izboljšanje natančnosti našega produkta, kar vodi do večje zanesljivosti in boljše uporabniške izkušnje. Razumevanje kompromisov med hitrostjo učenja in natančnostjo omogoča boljše načrtovanje učnega procesa, kar zmanjšuje stroške (in čas). Stabilnost modelov (prikazano skozi analizo gradientov), je ključna za dolgotrajno delovanje produkta brez pogostih ponovnih učenj – ena od bistvenih uporabniških zahtev pri produktu Spotter.

Pri tej študiji smo se soočali z več omejitvami, ki so vplivale na obseg in globino naših raziskav. Ena ključnih omejitev je bila potreba po uporabi manjših modelov zaradi zahteve po hitri inferenci v našem produktu. To je pomenilo, da smo se morali osredotočiti na modele, ki so sposobni hitrega procesiranja in generiranja kode v realnem času, kar je sicer omogočilo praktično uporabnost, vendar je morda omejilo potencial za doseganje še boljših rezultatov z večjimi, kompleksnejšimi modeli. Poleg tega smo bili omejeni tudi s količino in raznolikostjo razpoložljivih podatkov. Čeprav smo uporabili raznolike vire, vključno z GitHub podatki, je bila naša podatkovna množica še vedno omejena v primerjavi z obsežnimi podatkovnimi zbirkami, ki jih uporabljajo večji raziskovalni projekti. Ta omejitev je lahko vplivala na zmožnost naših modelov za generalizacijo in obvladovanje širšega spektra Ansible nalog. Kljub tem omejitvam smo uspeli pridobiti dragocene vpoglede, ki bodo usmerjali naše nadaljnje raziskave in razvoj produkta, s poudarkom na iskanju ravnovesja med hitrostjo inference in kakovostjo generiranih rezultatov.

Za prihodnje raziskave predlagamo več ključnih usmeritev. Smiselno bi bilo preizkusiti nove modele, ki se nenehno razvijajo in prihajajo na trg, saj ti lahko prinesejo izboljšave v zmogljivosti in učinkovitosti. Poleg tega bi bilo koristno razširiti našo učno množico, ne le v smislu količine, ampak tudi raznolikosti in kakovosti podatkov. To bi lahko izboljšalo splošno zmogljivost in robustnost naših modelov. Nazadnje, posebno pozornost bi bilo treba nameniti razvoju in implementaciji pristopov, ki bi preprečili modelom generiranje nepravilnih ali neželenih vsebin. To bi lahko vključevalo naprednejše tehnike nadzorovanega učenja, izboljšane mehanizme filtriranja ali nove arhitekture modelov, ki bi bile bolj odporne

na napake in bolje prilagojene našim specifičnim potrebam. S kombinacijo teh pristopov bi lahko dosegli pomemben napredek v kakovosti in zanesljivosti našega produkta.

5 Najboljši rezultat - prva različica programske opreme

Na podlagi podrobne analize rezultatov, predstavljenih v prejšnjih poglavjih, smo za prvo različico programske opreme izbrali model StarCoder2-3b. Čeprav model CodeLlama-7b dosega boljše rezultate glede evalvacijske izgube in perpleksnosti, smo se odločili za StarCoder2-3b zaradi več ključnih prednosti:

- StarCoder2-3b je s 3 milijardami parametrov znatno manjši od 7-milijardskega modela, kar se odraža v bistveno hitrejši inferenci. Ta hitrost je izjemno pomembna za naše aplikacije, kjer je odzivnost ključnega pomena za uporabniško izkušnjo.
- Kljub manjši velikosti StarCoder2-3b dosega zelo konkurenčno uspešnost, z evalvacijsko izgubo 0,65 in perpleksnostjo 1,93, kar je le marginalno slabše od večjega modela. Ta kombinacija hitrosti in učinkovitosti pomeni, da je model StarCoder2-3b optimalna izbira za naše potrebe, saj omogoča hitro delovanje v realnem času brez znatnega kompromisa glede kakovosti rezultatov.

Po izboru modela StarCoder2-3b smo nadaljevali z implementacijo prve različice programske opreme. Proces implementacije je vključeval integracijo modela v programsko opremo za zagotavljanje natančnih napovedi in analiz ter obsežna testiranja z različnimi scenariji, da bi zagotovili, da model deluje skladno s pričakovanji in da so napovedi natančne.

Ključni del našega evalvacijskega procesa je bilo testiranje s strani internih XLAB strokovnjakov za Ansible. Ti strokovnjaki so temeljito pregledali rezultate našega modela in podali svojo strokovno oceno. Njihovo mnenje je bilo izjemno dragoceno, saj so potrdili, da so rezultati modela visoke kakovosti in uporabni v praksi. Strokovnjaki so izrazili zadovoljstvo z natančnostjo in relevantnostjo generiranih Ansible skript, kar dodatno potrjuje uspešnost našega pristopa.

Po implementaciji in testiranju so bili doseženi pomembni praktični rezultati. Uporaba modela StarCoder2-3b je znatno izboljšala natančnost napovedi v primerjavi s prejšnjimi rešitvami. Povratne informacije uporabnikov, vključno s strokovnjaki za Ansible, so pokazale visoko zadovoljstvo zaradi izboljšane natančnosti in zanesljivosti programske opreme. Poleg tega smo pri uporabi nove različice programske opreme opazili zmanjšanje števila napak, kar je izboljšalo uporabniško izkušnjo in zaupanje v produkt.

Kljub doseženim uspehom smo identificirali več področij za nadaljnje izboljšave. Raziskali bomo možnosti za optimizacijo hitrosti učenja, ne da bi ogrozili natančnost. Načrtujemo tudi vključitev dodatnih podatkov za še boljšo natančnost in generalizacijo modela. Nadaljevali bomo z raziskavami na področju optimizacije hiperparametrov za doseganje še boljših

rezultatov, pri čemer bomo upoštevali tudi povratne informacije Ansible strokovnjakov za nadaljnje izboljšave. Spodaj vidimo 3 primere delovanja našega modela, kjer smo ocenili generirano kodo s produktom Spotter ter chatGPT 4.

Primer 1:

```
Vhod: Add a new user called 'John'
Izhod:
- name: Add a new user called 'John'
  ansible.builtin.user:
    name: '{{value}}'
    state: present

Spotter output:
Status:   success
Errors:   0
Warnings: 0
Hints:    0
Gpt ocena: 8.0
```

Ocena je nižja od 10, čeprav Spotter ni zaznal napak. To pokaže, da je ChatGPT precej strog ocenjevalec in morda naš model ni uporabil vseh najboljših praks za generiranje kode. Moramo pa seveda upoštevati, da tudi ChatGPT ni nezmotljiv.

Primer 2:

```
Vhod: Create a directory for custom nginx modules
Izhod:
- name: Create a directory for custom nginx modules
  ansible.builtin.file:
    path: /etc/nginx/modules
    state: directory
Spotter output:
Status:   success
Errors:   0
Warnings: 0
Hints:    0
GPT ocena: 9.0
```

Ponovno vidimo, da Spotter ne zazna napak na generiranem Ansible tasku, vendar mu ChatGPT ponovno ne da najvišje ocene. Ponovno vidimo, da ChatGPT ni nagnjen k dajanju visokih ocen.

Primer 3:

```
Vhod: Try to install package, retrying up to 3 times if it fails
Izhod:
- name: Try to install package, retrying up to 3 times if it fails
  ansible.builtin.apt:
    name: '{{value}}'
    state: present
    retries: 3
Spotter output:
Status:   error
Errors:   1
Warnings: 0
Hints:    1
----------
HINT: deprecated-parameter [H004] (line: 2, column:3)
Parameter 'name' is an alias for parameter 'package' in module
'ansible.builtin.apt'.
----------
ERROR: unknown-param-for-module-maybe-indent-error [E002] (line: 2, column:3)
'retries' is not a valid parameter in module 'ansible.builtin.apt', it looks
like a task keyword with incorrect indentation (too many spaces before the
keyword).
GPT ocena: 4.0
```

V tem primeru je Spotter opazil 1 napako in 1 namig. Napaka je seveda hujša in zato je tudi ChatGPT posledično precej spustil oceno kvalitete generirane Ansible naloge. Ta primer je spadal med težje v naši evalvacijski množici.

6 Zaključek

V tem dokumentu smo predstavili rezultate našega raziskovanja in primerjave štirih različnih modelov: CodeGen-350-multi, StarCoder2-3b, StarCoder2-3b z dodatnimi GitHub podatki in CodeLlama-7b. Naše glavne ugotovitve kažejo, da je model CodeLlama-7b pokazal najboljšo evalvacijsko izgubo in perpleksnost. Za naše potrebe, ki jih diktirajo uporabniške zahteve, smo izbrali kompromis med natančnostjo in hitrostjo – odločili smo se za model StarCoder2-3b, ki je dosegel zadovoljivo natančnost ob nižjih računalniških zahtevah, kar je bil ključni faktor pri izbiri za prvo različico programske opreme. Opazili smo tudi, da je dodajanje GitHub podatkov poslabšalo rezultate modela StarCoder2-3b, kar kaže na pomembnost kvalitete podatkov.

Izbrani model StarCoder2-3b je bil uspešno integriran v prvo različico naše programske opreme, kar je prineslo pomembne prednosti. Strokovnjaki so poročali o večjem zadovoljstvu zaradi boljše natančnosti in zanesljivosti. Poleg tega model omogoča učinkovitejše izvajanje nalog z nižjimi stroški in manjšo porabo virov.

Naše ugotovitve imajo več implikacij za prihodnji razvoj. Načrtujemo širitev učnih množic z dodatnimi kvalitetnimi podatki, da bi še dodatno izboljšali natančnost in generalizacijo modelov. Nadaljnje raziskave na področju optimizacije hiperparametrov so ključne za doseganje boljših rezultatov. Prav tako bomo raziskali načine za izboljšanje hitrosti učenja brez ogrožanja natančnosti, kar bo še dodatno povečalo učinkovitost našega produkta.

Za nadaljnje raziskave predlagamo poglobljeno analizo različnih modelov, vključno z novimi in izboljšanimi modeli, ki bi lahko ponudili še boljše rezultate. Pomembno bo tudi eksperimentiranje z različnimi učnimi množicami, da bi raziskali učinek različnih podatkovnih virov na uspešnost modelov. Razvoj bolj sofisticiranih metod za vrednotenje in primerjavo modelov, ki vključujejo širši nabor metrik in scenarijev, bo prav tako ključnega pomena za naše nadaljnje delo.

Dokončanje te raziskave in implementacija prve različice programske opreme predstavlja pomemben korak naprej pri izboljšanju naših produktov. Kljub doseženim uspehom se zavedamo, da je vedno prostor za izboljšave. Z zavezanostjo k nenehnemu učenju in inovacijam bomo še naprej stremeli k razvoju najboljših rešitev za naše uporabnike.