Financira Evropska unija
NextGenerationEU
REPUBLIKA SLOVENIJA
MINISTRSTVO ZA VISOKO ŠOLSTVO, ZNANOST IN INOVACIJE
ARIS Javna agencija za znanstvenoraziskovalno in inovacijsko dejavnost Republike Slovenije

D6.1

-

Dokumentacija in načrt rabe podatkov in pristopov

31.8.2024

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

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

Ključne besede
Izbor podatkov, Kriteriji za izbor podatkov, Izbor LLM metod, Metodologija

Kazalo

Kazalo Tabel

Financira Evropska unija NextGenerationEU
REPUBLIKA SLOVENIJA MINISTRSTVO ZA VISOKO ŠOLSTVO, ZNANOST IN INOVACIJE
ARIS Javna agencija za znanstvenoraziskovalno in inovacijsko dejavnost Republike Slovenije

Povzetek

Izbor podatkov je temeljni gradnik celotnega procesa izgradnje velikega jezikovnega modela. Glede na to, da gre za izjemno specializirano opravilo, je število učnih primerov izjemno omejeno. Nadalje želimo, da so zbrani podatki kvalitetni, zato smo se odločili, da jih bomo izbrali iz interno obstoječe množice dobrih primerov, preiskali tudi repozitorij Github in jih potem še ročno prečistili. Končni rezultat našega dela je relativno majhna množica kvalitetnih podatkov, ki jih uporabljamo za učenje (učna množica). S pomočjo orodja ChatGPT smo ustvarili tudi testno množico . Pri tem smo izkoristili obstoječe orodje Spotter, ki ocenjuje kvaliteto zbranih podatkov.

Ker vemo, da je kvaliteta končnega velikega jezikovnega modela odvisna tudi od izbora začetnega splošnega modela, smo ocenili tudi nekaj takšnih modelov. Pri tem smo bili pazljivi in določili ustrezne kriterije, ki so pomembni za končni produkt in so nam bili sporočeni s strani obstoječih uporabnikov orodja Spotter.

Na koncu smo izdelali tudi metodo, s katero bomo modele, naučene s tako zbranimi podatki ocenjevali. Trenutni rezultati so vzpodbudni.

1 Uvod

Kljub očitnemu porastu uporabe strojnega učenja in umetne inteligence, se je šele s prihodom velikih (ali pa izjemno velikih) jezikovnih modelov pričel novi razvojni cikel in razcvet področja umetne inteligence in povezanih aplikacij. Izkazalo se je, da so kljub izjemni uporabnosti, veliki jezikovni modeli izjemno dragi. Učenje namreč zahteva veliko podatkov in zelo redka podjetja imajo dovolj finančnih sredstev, da zberejo in hranijo tako veliko količino podatkov (velika prostorska zahtevnost). Samo učenje je računsko zelo zahtevno in zahteva izjemno drago strojno opremo, ter ogromno količino človeških naporov. Glede na velik finančni vložek, so podjetja v veliki večini svoje modele zaprla, kar pomeni, da je velika večina tako razvitih modelov zaprtih in ne omogočajo nobenega vpogleda v kvaliteto odgovorov in delovanje.

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. Ta korak se v jeziku umetne inteligence imenuje v angleščini »focusing the GPT model«. Leta 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 .

Naš cilj je zgraditi jezikovni model, ki je uporaben primarno tudi za produkt Spotter [1] 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**, za katerega skrbi eno večjih in starejših podjetij, ki se ukvarja s z odprto kodo – Red Hat.

Ansible je zmogljivo orodje za avtomatizacijo upravljanja z računalniško infrastrukturo. Kot vsako programsko orodje oz. jezik, je Ansible organiziran v hierarhično strukturo. Glavne komponente, napisane po velikosti, 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.

[1] https://steampunk.si/spotter/

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[2], 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 (https://www.python.org) ter podatkov, ki so zapisane v skriptah v jeziku YAML[3].

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 D6.1 je produkt eksperimentalnega razvoja v sklopu RRP6. Podjetje XLAB preko znanj, ki jih pridobiva predvsem v krovnem sklopu RRP2, postaviti temelje ter izpiliti kriterije za izbor primernih učnih podatkov, ki zadoščajo kriterijem, postavljenim s strani bodočih strank ter s strani uspešnosti samega učenja.

Namen je torej opisati pristop k zbiranju podatkov ter le-te primerno opredeliti. Ker so na področju velikih jezikovnih modelov podatki oz. njihova oblika tesno sklopljeni z izbiro metode, so opisane tudi lastnosti metod ter preliminarni eksperimenti, ki nas vodijo k izboru primernih metod. Končni cilj dokumenta je torej opisati izbor podatkov in metod (ta izroček D6.1 ter s tem določiti smer razvoja ter postaviti temelje za izroček D6.2, ki je naslednji izroček RRP6.

[2] https://www.redhat.com/en/technologies/management/ansible
[3] https://yaml.org

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 – horizontalni podprojekt RRP2 in je, v skladu z razvojnim načrtom projekta, povsem ločena od eksperimentalnega razvoja drugih podjetij oz. deležnikov v projektu.

1.3 Struktura dokumenta

Dokument je sestavljen iz treh gradnikov:

- **Izbor podatkov – poglavje 2 –** kjer opišemo kako smo zbirali podatke in predvsem kakšne kriterije smo pri tem izbrali in definirali.
- **Izbor metod – poglavje 3 –** ki je poglavje, ki smo ga zapisali v ta izroček, saj se izkaže, da brez upoštevanja ciljnih metod, težko zbiramo podatke. Same metode učenja smo tudi primerjali med sabo ter jih ocenili, glede na izbrane kriterije
- **Načrt uporabe tako zbranih podatkov** in seveda **preliminarna evalvacija**.

Na tem mestu je potrebno povdariti, 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 Izbor podatkov

2.1 Kriteriji za izbiro podatkov

Da bi zagotovili kakovost podatkov, ki bodo uporabljeni za učenje globokih modelov, smo določili natančne kriterije za izbiro podatkov. Le-ti so določeni glede na pričakovano namembnost in na pričakovanja s strani uporabnikov.

Prvi kriterij je kakovost kode. Izbrali smo samo tiste datoteke in dele kode, ki izpolnjujejo standarde najboljših praks v Ansible programiranju. To vključuje jasno in berljivo strukturo kode, uporabo veljavnih in preverjenih Ansible modulov ter skladnost s priporočenimi smernicami za pisanje Ansible kode, kot jih navaja uradna dokumentacija in sprejema ter priporoča skupnost. Kvalitetna koda je ključnega pomena za grajenje robustnih modelov, saj tako izbrani vzorci zmanjšujejo možnost generiranja napak in omogočajo, da modeli pridobijo natančne in zanesljive vzorce pri učenju, kar se odraža tudi pri generiranju.

Drugi kriterij je priljubljenost kode. Koda, ki je pogosto uporabljena in dobro ocenjena s strani skupnosti, je izbrana prednostno. To vključuje zbiranje podatkov iz repozitorijev (odložišč kode), kot so GitHub[4], kjer lahko ocenimo popularnost in zanesljivost kode na podlagi števila zvezdic[5]. Priljubljena in visoko ocenjena koda je običajno večkrat in natančneje preizkušena in izpopolnjena, kar pomeni, da imajo modeli, zgrajeni na tej osnovi, večjo verjetnost za uspeh oz. večjo verjetnost generiranja dobre kode.

Tretji kriterij je velikost učne množice. Za učenje globokih modelov potrebujemo zadostno količino podatkov, da bi lahko modeli prepoznali vzorce in se naučili generalizirati. Večja učna množica pomeni, da modeli pridobijo večjo raznolikost podatkov, kar izboljša njihovo zmogljivost in zmožnost obravnavanja različnih scenarijev. Zato smo izbrali podatkovne množice, ki vsebujejo zadostno količino kode, da lahko modeli dosežejo visoko stopnjo natančnosti in učinkovitosti. Pri tem moramo poudariti, da je količina podatkov, ki je na voljo v tako specializirani domeni, še vedno zelo majhna v primerjavi s siceršnjimi količinami, ki jih za učenje uporabljajo velika podjetja.

Četrti kriterij se nanaša na dovoljenja licenc, pod katerimi je Ansible koda objavljena. Uporabljamo le tiste podatkovne množice, ki so izdane pod odprtokodnimi licencami, kot so MIT[6], Apache 2.0[7] in GPLv3[8], ki dovoljujejo prosto uporabo, spreminjanje in distribucijo kode.

[4] https://github.com
[5] https://docs.github.com/en/get-started/exploring-projects-on-github/saving-repositories-with-stars
[6] https://mit-license.org
[7] https://www.apache.org/licenses/LICENSE-2.0
[8] https://www.gnu.org/licenses/gpl-3.0.en.html

To je pomemben kriterij, ki nam omogoča pravno skladno uporabo podatkov pri gradnji modelov ter zagotavlja, da naši rezultati in morebitne prilagoditve ostanejo v skladu s pravili odprtokodnih skupnosti.

S izborom teh kriterijev ter njihovo definicijo zagotavljamo, da so v skladu z njimi izbrane podatkovne množice kakovostne in pravno skladne. S tako izbranimi kriteriji za izbor podatkov bomo predvidoma dosegali generiranje kakovostne in relevantne kode, ki bodo, tako kot izvorni podatki, predstavljali najboljše prakse in standarde v Ansible skupnosti.

S tem pristopom smo povečali verjetnost, da bodo naučeni modeli delovali učinkovito in zanesljivo v realnih scenarijih uporabe, kar je ključnega pomena za uspeh produkta.

Tabela 1: Kriteriji za izbor programske kode v učni množici

Kriterij	Opis
Kakovost kode	Jasna in berljiva struktura, uporaba preverjenih modulov, dokumentirani komentarji.
Priljubljenost	Pogosto uporabljena in dobro ocenjena koda s strani skupnosti, zbrana iz repozitorijev.
Velikost	Zadostna količina podatkov za prepoznavanje vzorcev in generalizacijo.
Licence	Koda izdana pod odprtokodnimi licencami (MIT, Apache 2.0).

2.2 Zbrani podatki

2.2.1 Steampunk

Množica Steampunk je bila ustvarjena z namenom zagotavljanja visokokakovostnih podatkov za nadaljnje analize in modeliranje. Začetna faza je vključevala zbiranje podatkov s tehniko zajemanja s platforme Ansible Galaxy[9]. Ansible Galaxy je spletna platforma, ki omogoča uporabnikom deljenje in dostop do ponovno uporabnih Ansible vlog (ang. role), knjižnic in drugih konfiguracij, s čimer pospešuje razvoj avtomatiziranih okolij in olajšuje upravljanje infrastrukture. Za pridobivanje zbirk s te platforme smo uporabili filtre, ki preprečujejo nalaganje zbirk, ki bi lahko bile škodljive ali so imele pretirano veliko število verzij zaradi pogostih posodobitev.

Nadaljevali smo z natančno analizo celotne zbirke podatkov, kjer smo najprej izvedli postopek zakrivanja vrednosti z visoko entropijo. To smo storili zato, ker so nekatere vrednosti vsebovale nize z visoko entropijo, kar običajno kaže na občutljive informacije, kot

[9] https://galaxy.ansible.com/ui/

so gesla ali ključi za šifriranje, ali pa tudi zapisi slik v različnih formatih. S tem korakom smo zmanjšali tveganje neželenega razkritja občutljivih podatkov in povečali varnost celotnega nabora podatkov, ter ohranili vrednosti iz katerih se bodo naši modeli lahko učili.

Nato smo naleteli na vzorce, ki so vsebovali celotne seznamov Ansible nalog (ang. playbooke) namesto posameznih Ansible nalog (angl. task). Ker so Ansible skripte sestavljale kompleksne in večplastne naloge, smo jih razdelili na manjše, obvladljive enote, saj je cilj našega projekta razviti model, ki bo sposoben generirati Ansible naloge iz naravnega jezika.

Vsak posamezni izsek kode smo preverili glede pravilne uporabe YAML sintakse, saj je pravilna struktura ključna za izvedbo kode brez napak. Prav tako smo naloge pregledali glede njihove dolžine, saj prekratke ali preveč jedrnate naloge pogosto ne vsebujejo dovolj informacij. Po drugi strani predolge naloge vsebujejo preveč nepotrebnih informacij potrebnih za kvalitetno učenje modelov. Po naši analizi smo ugotovili, da Ansible naloge, ki imajo več kot 600 znakov praktično vedno vsebujejo šum, prekratke naloge pa ne vsebujejo kvalitetne informacije. S tem smo zagotovili, da so vse naloge dovolj obsežne in vsebinsko bogate, kar omogoča boljše razumevanje in aplikacijo v praktičnih scenarijih.

V nadaljevanju smo odstranili naloge, ki so vsebovale vnaprej določene izraze v opisu izseka kode. To smo naredili zato, ker določeni opisi Ansible nalog niso nosili informacije, ki bi lahko modelu pomagala pri razumevanju in je kvečjemu oteževala razumevanje in generalizacijo.

Za zaključek smo izbirali samo naloge iz najnovejših zbirk, saj to zagotavlja, da so podatki aktualni in relevantni. S tem korakom smo zagotovili, da so podatki v skladu z najnovejšimi standardi in praksami, kar povečuje verodostojnost in uporabnost naučenih modelov.

Ta proces je bil ključnega pomena ne samo za zagotavljanje visoke kakovosti in varnosti podatkov, ampak tudi za vzpostavitev trdnih temeljev za nadaljnje faze projekta, kjer se uporabljajo ti podatki za razvoj in testiranje modelov.

Množica Steampunk je bila ustvarjena z namenom zagotavljanja visokokakovostnih podatkov za nadaljnje analize in modeliranje. Kvaliteta te množice je zagotovljena z natančno selekcijo in maskiranjem visoko entropičnih vrednosti, da bi ohranili občutljive informacije zaščitene. Priljubljenost nalog je zagotovljena z izborom in uporabo dobro dokumentiranih in široko uporabljanih metod v skupnosti, medtem ko je velikost množice obsežna, saj pokriva širok spekter scenarijev uporabe. Vsa dokumentacija je licencirana pod odprtokodnimi licencami, kar omogoča široko uporabo in nadaljnje prilagajanje.

Tabela 2: Statistika podatkovne zbirke zbranih učnih podatkov – izvor Ansible Galaxy in Steampunk

Število primerov	37.174
Število različnih modulov	16.028
Število različnih collectionov	561
Povprečno število znakov	243,6

2.2.2 Javna koda iz repozitorijev GitHub

Za pridobitev množice GitHub podatkov smo izvedli več korakov s pomočjo posebej razvitih skript. Najprej smo uporabili skripto, ki nam je omogočila pridobivanje seznama repozitorijev, ki vključujejo ključno besedo "Ansible" v imenu, opisu ali datoteki README. Po identifikaciji ustreznih repozitorijev smo z uporabo druge skripte iz teh repozitorijev pridobili vse YAML datoteke. Končno smo analizirali in prešteli vse licence povezane z zbranimi repozitoriji. Ko smo imeli za vsako Ansible nalogo licenco, smo ohranili samo taske, katerih licence dovolijo odprto uporabo. Ta proces nam je omogočil sistematično zbiranje in organizacijo podatkov za naš projekt, tako smo zagotovili natančnost in relevantnost pridobljenih informacij.

Podobno kot pri izboru podatkov v množici Steampunk, smo tudi za zbiranje podatkov iz Ansible kode na GitHubu uporabili metode, ki zagotavljajo visoko kakovost in varnost podatkov. Med analizo teh zbirk smo prav tako izvajali maskiranje visoko-entropičnih vrednosti, da bi preprečili razkritje občutljivih informacij, podobno kot smo to storili pri izboru podatkov množice Steampunk.

Pri nadaljnji obdelavi Ansible kode smo, tako kot pri Steampunk množici, identificirali in ločili celotne playbooke na posamezne, obvladljive Ansible naloge.

Vsak izsek kode smo preverili:

- ali je sintaksa YAML pravilna in
- ali je dolžina naloge ustrezna.

Vemo namreč, da dolžina nalog (prekratke ali predolge naloge), lahko negativno vplivala na kakovost učenja. Kot pri prvi množici, smo tudi tukaj določili optimalno dolžino kode, ki ne presega 600 znakov, da zagotovimo ustreznost in kvaliteto informacij, potrebnih za učenje.

Ansible koda, ki smo jo pridobili z GitHuba je natančno urejena oz. pregledana. Postopek je vključeval filtriranje repozitorijev po ključnih besedah, ekstrakcijo YAML datotek ter analizo modulov in taskov. Glede na to, da je ta množica pridobljena iz odprtih repozitorijev na GitHubu, se kakovost kode močno razlikuje. Pri tako majhni učni množici je potrebna pazljivost predvsem pri kvaliteti vhodnih podatkov, zato nadaljnji učni proces predvideva, da

je kvaliteta teh podatkov potencialno nižja v primerjavi s prvo zbirko. Z dodatnimi preverbami in ročnim čiščenjem podatkov smo zagotovili, da je množica uporabna za naše potrebe. Priljubljenost kode smo ocenili samo na podlagi števila zvezdic, kar odraža njeno popularnost, ne pa nujno kvalitete ali uporabnosti. Ta množica vsebuje bogat nabor taskov in modulov, ki so koristni za dodajanje dodatnih primerov v učni proces, s čimer lahko izboljšamo sposobnost modela za učenje. Zbirka je licencirana pod odprtokodnimi licencami, kar spodbuja izmenjavo in nadaljnje izboljšave, a zahteva skrbno upoštevanje licenčnih pogojev.

Tabela 3: Statistika podatkovne zbirke zbranih učnih podatkov – izvor GitHub

Število primerov	40.750
Povprečno število znakov	175.3

2.2.3 Testna množica

Za potrebe neodvisne evalvacije smo naredili povsem ločeno množico za evalvacijo. Bistvenega pomena pri sestavljanju te množice je bilo, da množica vsebuje samo nedvomno pravilno kodo in kodo, ki predstavlja najbolj pomembne naloge, ki se uporabljajo v Ansible-u.

Za sestavo te množice smo uporabili ChatGPT 4[10]. ChatGPT smo prosili, da naj sestavi 10 lahkih, 10 srednje težkih in 10 težkih Ansible nalog, na katerih bomo lahko ocenili delovanje naših metod. V prihodnosti se bo ta množica še razširila vendar je bila za začetne poizkuse dovolj reprezentativna. Trenutna evalvacijska množica je majhna, vendar zelo uporabna za spremljanje napredka eksperimentov.

[10] https://chatgpt.com/

3 Izbor metod

Pri izbiri metod smo uporabili določene kriterije, da bi zagotovili, da naše izbrane metode izstopajo v enem ali več ključnih vidikih. Specifični kriteriji, v katerih izbrane metode izstopajo, so opisani v naslednjem podpoglavju. Pomembno je poudariti, da se področje hitro spreminja, kar pomeni, da se tudi primerjava med metodami lahko hitro spremeni. Zaradi hitrega napredka na področju umetne inteligence in strojnega učenja je pomembno, da redno posodabljamo našo izbiro in primerjavo metod, da ostanemo na tekočem z najnovejšimi dosežki. Pri vseh metodah smo gledali, da so osnovane na generativnih velikih jezikovnih modelih, ki so osnovani na tehnologiji *Transformer*, zato to nismo obravnavali kot kriterij v spodnjih poglavjih.

3.1 Kriteriji za izbor metod

Računska zahtevnost modela se nanaša na hitrost in porabo računalniških virov med generiranjem kode. Pomembno je, da modeli hitro in učinkovito generirajo Ansible kodo, saj je to ključno pri uporabi v realnem času in integraciji v avtomatizirane delovne procese. Čas inference, torej hitrost, s katero model generira kodo, je bistvenega pomena, saj krajši čas inference pomeni hitrejše rezultate, kar je pomembno za uporabnike, ki potrebujejo takojšnje odgovore. Prav tako smo upoštevali porabo pomnilnika, ki je potreben za delovanje modela. Manjša poraba virov omogoča uporabo modela na manj zmogljivi opremi in zmanjšuje stroške obratovanja. Modeli, ki so optimizirani za hitrejše delovanje brez kompromisov pri natančnosti generiranja kode, so imeli prednost pri izboru.

Kompleksnost metode se nanaša na sposobnost modela za obdelavo in generiranje natančne kode, kar je pogosto povezano z velikostjo in arhitekturo modela. Bolj kompleksni modeli imajo več parametrov in naprednejše arhitekturne tehnike, ki jim omogočajo globlje razumevanje vzorcev in struktur v podatkih. Kompleksni modeli, kot so tisti z več nivoji mreže in večjim številom parametrov, so sposobni bolje obdelati kompleksne naloge, vključno z generiranjem natančne in funkcionalne kode. Vendar pa višja kompleksnost pogosto zahteva večje računske vire in lahko poveča čas učenja in inference.

Prednaučeni modeli, ki že vsebujejo osnovno razumevanje Ansible ali podobnih tehnologij, omogočajo boljše in hitrejše učenje (fokusiranje modela oz. ang. finetuning). Zato je bila ena izmed naših ključnih meril **sposobnost modela za generiranje Ansible kode** že po prvi fazi učenja. To daje vpogled v to, kako dobro model razume strukture Ansible kode in predstavlja zelo pozitiven signal o kvaliteti metode.

Kakovost modelov smo ocenili z uporabo relevantnih testov (ang. benchmark-ov), kot so HumanEval[11], MBPP[12] in MultiPL-E[13], ki merijo zmogljivost pri generiranju kode in naravnem jeziku. HumanEval je nabor testov, zasnovan za ocenjevanje funkcionalne pravilnosti modelov, ki generirajo kodo v Pythonu, in vsebuje 164 ročno napisanih programskih nalog, ki vključujejo podpise funkcij, docstringe, telo funkcij in več enotnih testov. MBPP je osredotočen predvsem na Python in vključuje približno 1.000 množično zbranih programskih problemov, ki pokrivajo osnove programiranja in uporabo standardne knjižnice, kar omogoča ocenjevanje sposobnosti modelov pri reševanju osnovnih programerskih nalog. MultiPL-E pa je obsežnejši test, ki razširja ocenjevanje na več programskih jezikov in preizkuša modele v različnih programskih okoljih ter nalogah. Čeprav naši modeli niso bili ocenjeni neposredno na Ansible kodi, visoke ocene na teh testih še vedno nudijo pomemben vpogled v splošno zmogljivost modelov pri generiranju programske kode. Takšne ocene nam omogočajo, da ocenimo potencial modelov za uspešno generiranje Ansible kode po nadaljnem učenju.

Licenčni pogoji modelov določajo, kako jih lahko uporabljamo, distribuiramo in prilagajamo. Izbrali smo modele z odprtokodnimi licencami, kot so MIT, Apache 2.0 ali GNU GPL, ki omogočajo prosto uporabo in prilagoditev za komercialne in nekomercialne namene. Poleg tega smo upoštevali tudi modele s prilagojenimi licencami, ki dovoljujejo uporabo v naših specifičnih kontekstih. Takšne licence zagotavljajo večjo fleksibilnost in pravno varnost pri uporabi modelov v različnih kontekstih.

Kar se tiče **varnosti** smo upoštevali modele, ki so bili razviti in objavljeni s strani zaupanja vrednih organizacij ter vključujejo varnostne protokole za zaščito pred generiranjem zlonamerne ali nevarne kode. Varnostni mehanizmi so ključnega pomena za zagotovitev varne uporabe modelov.

Tabela 4: Kriteriji za izbor primernega UI modela oz. metode

Kriterij	Opis
Računska zahtevnost	Hitrost in poraba računalniških virov med generiranjem kode. Pomembna je hitrost inferenca in manjša poraba virov.
Kompleksnost	Sposobnost modela za obdelavo in generiranje natančne kode, kar je pogosto povezano z velikostjo in arhitekturo modela.

[11] https://deepgram.com/learn/humaneval-llm-benchmark
[12] https://huggingface.co/datasets/google-research-datasets/mbpp
[13] https://nuprl.github.io/MultiPL-E/

Primarno znanje	Modeli, ki že vsebujejo osnovno razumevanje Ansible, omogočajo boljši in hitrejši finetuning. Poskusili bomo tudi z modeli brez eksplicitnega znanja Ansible.
Kakovost	Kakovost modelov ocenjena na relevantnih testih, ki merijo zmogljivost pri generiranju kode in naravnem jeziku.
Licenca	Licenca, ki dovoljuje odprto uporabo
Varnost	Metoda mora izpolnjevati varnostne in etične standarde, da so generirane naloge varne in skladne z etičnimi načeli.

3.2 Izbrane metode

3.2.1 StarCoder2

Starcoder2 3b[14], dostopen na platformi Hugging Face, je napreden model s 3 milijardami parametrov, specifično zasnovan za obdelavo programske kode. Naučen je bil na raznolikih podatkovnih zbirkah, ki vključujejo 17 različnih programskih jezikov. Tehnike, kot sta Grouped Query Attention in sliding window attention, modelu omogočajo, da učinkovito obdela velike količine podatkov z relativno majhnim časovnim zamikom. Kvantizacija pomaga zmanjšati pomnilniške zahteve, kar omogoča uporabo modela tudi na strojni opremi z omejenimi viri. Varnost in etičnost sta bila temeljito obravnavana med razvojem, kar zagotavlja, da so generirane skripte v skladu z najvišjimi standardi.

3.2.2 CodeLlama

CodeLlama [15] je serija močnih modelov z velikostjo od 7 do 34 milijard parametrov, namenjena splošni sintezi in razumevanju kode. Model je bil usposobljen na obsežnem naboru podatkov, ki zajemajo tako GitHub kodo kot dodatne vire, kar mu omogoča visoko stopnjo prilagodljivosti in natančnost pri generiranju Ansible nalog. Kljub veliki moči modela pa obstajajo omejitve glede njegove učinkovitosti, saj optimizacija za hitro procesiranje ni vedno dosledna, kar lahko vpliva na hitrost delovanja na različnih strojnih platformah. Varnostni in etični protokoli so sicer dobro vzpostavljeni, a je potrebna previdnost pri uporabi modela v skladu z veljavnimi zakoni in regulativami.

3.2.3 CodeGen

CodeGen-Multi 350M [16]je del družine jezikovnih modelov za sintezo programov, zmožen obdelati obsežne nize podatkov iz večjezičnih programskih okolij. Model združuje 350

[14] https://huggingface.co/bigcode/starcoder2-3b
[15] https://llama.meta.com/code-llama/
[16] https://www.codegen.com/

milijonov parametrov in je bil usposobljen na raznolikih programskih jezikih, kar zagotavlja njegovo sposobnost hitrega in učinkovitega generiranja kode. CodeGen je robusten pri obvladovanju različnih vhodnih variacij, vendar njegova zmogljivost ni brez omejitev, saj lahko določene kompleksne zahteve vplivajo na natančnost generirane kode. Varnostni in etični standardi so bili temeljito obravnavani med razvojem, kar zagotavlja, da so vse aktivnosti modela skladne z najvišjimi varnostnimi standardi.

3.3 Primerjava metod po kriterijih

3.3.1 Računska zahtevnost

Model Starcoder2 s tremi milijardami omogoča učinkovito obdelavo kompleksnih vzorcev brez pretiranih računalniških zahtev. Kvantizacija modela pomaga zmanjšati pomnilniške zahteve, kar omogoča njegovo uporabo tudi na strojni opremi z omejenimi viri. CodeLlama-7b je optimiziran za generiranje kode z visoko učinkovitostjo, vendar zaradi svoje velikosti (7 milijard parametrov) zahteva znatne računske vire za optimalno delovanje. Čas inference je relativno kratek, kar omogoča hitro generiranje kode, vendar so lahko zahteve po procesorski moči in pomnilniku visoke. CodeGen uporablja manj parametrov, kar omogoča hitrejše delovanje in manjše računalniške zahteve, vendar je lahko manj natančen pri obvladovanju zelo kompleksnih vhodov v primerjavi z večjimi modeli.

V spodnji tabeli vidimo primerjave naših metodo glede na računsko zahtevnost.

Tabela 5: Primerjava modelov po računski kompleksnosti

Metoda	Poraba GPU pomnilnika (GB)	Žetonov na sekundo (s) (token per second)
CodeGen (350 milijonov)	1,17	46,5
Starcoder2 (3 milarde)	6,46	40, 1
CodeLlama2 (7 miljard)	13,5	22,1

3.3.2 Kompleksnost

Starcoder2 vsebuje 3 miljarde parametrov kar mu omogoča visoko zmogljivost in natančnost pri generiranju kode. Model CodeLlama s sedmimi milijardami parametrov je zelo kompleksen, kar mu omogoča globoko razumevanje in natančno generiranje kode. Nasprotno pa model CodeGen uporablja manj parametrov in manj kompleksno arhitekturo v primerjavi z ostalima modeloma, kar omogoča hitrejše delovanje in manjše računalniške zahteve, vendar je lahko zaradi manjše kompleksnosti nekoliko manj natančen pri obvladovanju zelo kompleksnih vhodov.

3.3.3 Predhodno znanje

Model Starcoder2 naj ne bi vključeval Ansible kode v svojih podatkih, čeprav je imel model že na začetku spodobno znanje Ansible. To nakazuje, da je verjetno prednaučen model videl to kodo nekje v svojem učenju, kar omogoča generiranje strukturno pravilne Ansible kode brez obsežnega prilagajanja nastavitev (ang. finetuning). To pomeni, da Starcoder2 razume strukture Ansible kode in lahko generira ustrezno kodo že iz začetka. Za model CodeLlama-7b specifični podatki o podatkovnih zbirkah niso na voljo, tako da ne moremo natančno vedeti, kaj je bilo vključeno v podatkovni niz. Vemo le, da je bilo vključenih 500 milijard tokenov javno dostopne kode, kar verjetno vključuje tudi Ansible kodo. To kaže, da CodeLlama-7b razume strukture Ansible kode in je sposoben generirati strukturno pravilno kodo. Model CodeGen-Multi 350M je bil najprej inicializiran z CodeGen-NL 350M in nato prednaučen na BigQuery, velikem naboru podatkov več programskih jezikov iz GitHub repozitorijev. Podatki vključujejo 119.2 milijard žetonov (ang. token-ov) in vsebujejo jezike, kot so C, C++, Go, Java, JavaScript in Python. Ker podatki ne vključujejo Ansible kode, model CodeGen ne razume struktur Ansible kode tako dobro kot Starcoder2 in CodeLlama-7b, kar pomeni, da ni tako učinkovit pri generiranju strukturno pravilne Ansible kode.

Tabela 6: Primerjava modelov glede na predhodno znanje

Metoda	Učni podatki vsebujejo Ansible/YAML?	Razume Ansible strukturo?	Število žetonov v učnih podatkih (milijarde)	Temelji na modelu za splošni jezik?
Starcoder2 (3 milarde)	Ne	Da	3000	Ne
CodeLlama2 (7 miljard)	Ni podatkov	Da	500	Da
CodeGen (350 milijonov)	Ne	Ne	119.2	Da

3.3.4 Rezultati na testih

Na podlagi HumanEval benchmarka je model CodeLlama-7b dosegel višjo oceno (33,5) v primerjavi s Starcoder2-3b (31,7). Na MBPP benchmarku pa je Starcoder2-3b dosegel višjo oceno (57,4) kot CodeLlama-7b (41,4). Glede na MultiPL-E benchmark, je CodeLlama-7b dosegel višjo oceno (26,3) v primerjavi s Starcoder2-3b (24,9). Model CodeGen je dosegel nižje ocene na HumanEval (6,67) in MBPP (7,46) ter ni bil testiran na MultiPL-E. Njegova glavna prednost ostaja računska nezahtevnost v primerjavi z obema sodobnejšima modeloma.

Tabela 7: Primerjava modelov glede na Humaneval, MBPP ter MultiPL-E teste.

Metoda	Humaneval	MBPP	MultiPL-E
Starcoder2 (3 milarde)	31,7	**57,4**	24,9
CodeLlama2 (7 miljard)	**33,5**	41,4	**26,3**
CodeGen (350 milijonov)	6.67	7,46	Ni podatkov

3.3.5 Licence

Model Starcoder2 je dostopen pod odprtokodno licenco, kar omogoča prosto uporabo in prilagoditev za komercialne in nekomercialne namene, kar zagotavlja fleksibilnost pri integraciji modela v naše sisteme. Tudi model CodeLlama-7b je dostopen pod odprtokodno licenco, kar omogoča prosto uporabo in prilagoditev za različne namene, zagotavlja večjo fleksibilnost in pravno varnost pri uporabi modela v različnih kontekstih. Model CodeGen je licenciran pod pogoji OpenRAIL-M v1, kar določa uporabo modela skladno z etičnimi in varnostnimi standardi, kar zagotavlja visoko stopnjo varnosti in skladnosti.

3.3.6 Varnost

Model Starcoder2 je naučen na podatkih specifičnih za programsko kodo in ne vključuje podatkov iz spleta, kar zmanjšuje tveganje za problematične ali zlonamerne podatke. Varnostni protokoli so bili temeljito upoštevani med razvojem, kar zagotavlja varno uporabo modela. Model CodeLlama-7b je bil razvit s poudarkom na varnosti, saj so bili upoštevani strogi varnostni in etični protokoli. Model je naučen na podatkih, ki so specifični za programsko kodo, kar zmanjšuje tveganje za problematične podatke. Vrednost varnosti je ocenjena kot visoka. Model CodeGen je bil temeljito obravnavan z vidika varnostnih in etičnih vidikov med razvojem, kar zagotavlja, da so vse aktivnosti modela skladne z najvišjimi standardi. Model je licenciran pod pogoji OpenRAIL-M v1, kar določa uporabo modela skladno z etičnimi in varnostnimi standardi.

4 Načrt uporabe zbranih podatkov

V naših eksperimentih bomo uporabili različne množice, ki smo jih predstavili v poglavju 5.1. Eksperimenti bodo vsebovali različne kombinacije učnih.

4.1 Prilagajanje modela za sledenje navodilom

Prilagajanje modela za sledenje navodilom oziroma nadzorovano učenje je pristop v umetni inteligenci, kjer je model zasnovan tako, da sledi navodilom, ki jih dobi v naravnem jeziku. Namesto da bi zgolj obdeloval vnaprej določen nabor podatkov ali nalog, se model uči razumeti in izvesti specifične ukaze, ki jih vnese uporabnik. Ta pristop omogoča bolj intuitivno interakcijo med človekom in strojem, saj uporabnik poda navodila v naravnem jeziku, model pa jih interpretira in izvede ustrezne akcije.

V kontekstu generativnih jezikovnih modelov to pomeni, da lahko uporabnik vnese opis naloge ali vprašanje, model pa bo ustvaril ustrezen odgovor ali izvedel zahtevano dejanje, kot je generiranje besedila, ustvarjanje kode ali izvedba kompleksnih nalog na podlagi podanih navodil.

V našem primeru je to smiseln pristop, saj bi lahko model bolje razumel strukture Ansible taska in morda tudi zaradi bolj bogatega navodila bil bolj omejen na generiranje pravilnega Ansible taska.

Pri tem pristopu je potrebno pred učenjem narediti vhod in pravilen izhod modela za vsak primer v učni množici, iz česar izhaja ime nadzorovano učenje. Primer:

```
Vhod: Create Ansible task with following task name: 'create sensu user'

Izhod:
- name: create sensu user
  user:
    name: sensu
    state: present
    groups: sensu
    shell: /bin/false
```

Učna množica mora biti sestavljena iz takih parov za pristop nadzorovanega učenja. V naših eksperimentih bomo tudi eksperimentirali z različnimi navodili. V zgornjem primeru je en primer navodila: *Create Ansible task with following task name: {task_name}*". Polje *{task_name}* pridobimo iz učne množice.

Navodila v naših eksperimentih se bodo rahlo spreminjala, predvsem v smislu manjših sprememb v besedah ali črkah, vendar bodo vsa podobna osnovni strukturi.

Učinkovitost različnih oblik navodil bomo merili na podlagi testne množice. To nam bo omogočilo objektivno primerjavo med različnimi pristopi in izbiro najuspešnejšega formata navodil za naš končni model.

Priprava učne množice bo potekala tako, da bomo za vsak primer vzeli ime naloge (task name), zgradili ustrezno navodilo in uporabili pravilen izhod. Ta proces nam bo omogočil ustvarjanje obsežne množice parov vhodov in izhodov, potrebnih za učinkovito nadzorovano učenje.

Za fino prilagajanje (ang. finetuning) našega modela bomo uporabili tehniko prenosa znanja (transfer learning). Ta pristop izkorišča znanje, ki ga je model pridobil pri reševanju ene naloge, in ga prenese na novo, sorodno nalogo. V našem primeru bomo začeli z modelom, ki je že naučen razumevanja in generiranja naravnega jezika, ter ga nato prilagodili specifični nalogi generiranja Ansible taskov.

Prenos znanja nam omogoča, da:

1. Skrajšamo čas učenja, saj model že ima osnovno razumevanje jezika.
2. Izboljšamo učinkovitost, zlasti ko imamo omejeno količino specifičnih podatkov za našo nalogo.
3. Dosežemo boljše rezultate, saj model gradi na že pridobljenem znanju.

Z uporabo teh pristopov – prilagajanja modela za sledenje navodilom, skrbno pripravljene učne množice in tehnike prenosa znanja – pričakujemo, da bomo razvili model, ki bo sposoben natančno in zanesljivo generirati Ansible taske na podlagi podanih navodil.

4.2 Evalvacija

Tu bomo predstavili naš pristop k evalvaciji generiranih Ansible nalog z našo testno množico. Za evalvacijo smo uporabili testno množico, ki je podrobneje opisana v poglavju 5.2.3. Naš proces evalvacije je sestavljen iz treh ključnih korakov: generiranje Ansible nalog, analiza z orodjem Steampunk Spotter in ocenjevanje s pomočjo ChatGPT API-ja.

Za vsak primer iz testne množice smo sestavili navodilo, ki je vsebovalo ime Ansible naloge, na podlagi katere smo želeli generirati Ansible nalogo (task). To navodilo smo nato uporabili kot vhod za našo metodo generiranja Ansible nalog. S tem pristopom smo zagotovili, da smo lahko preizkusili našo metodo na širokem naboru različnih Ansible nalog.

Generirane Ansible naloge smo nato analizirali z orodjem Steampunk Spotter. To orodje nam je omogočilo podrobno analizo generiranih nalog ter nam zagotovilo število napak, opozoril in namigov, kako bi izboljšali generirano Ansible nalogo in tudi kratek opisa problema. Spodaj vidimo primer izhoda iz orodja Steampunk Spotter.

```
Spotter Evaluation:
Status:   error
Errors:   1
Warnings: 0
Hints:    0

----------
ERROR: required-parameter-missing [E005] (line: 2, column:3)
'path' is a required parameter in module
'community.crypto.openssh_cert'.
```

Vidimo, da orodje v tem primeru vrne da je pri generiranju nalog iz te zbirke potrebno uporabiti parameter *path*. Spotter je pomemben korak v našem evalvacijskem procesu, saj nam omogoča avtomatizirano preverjanje skladnosti in kakovosti generiranih Ansible nalog.

V zadnjem koraku smo rezultate analize Steampunk Spotter orodja uporabili kot vhod za ChatGPT API. Modelu smo naročili, naj oceni generirano Ansible nalogo na lestvici od 1 do 10, pri čemer smo mu omogočili, da pri ocenjevanju upošteva tudi rezultate in razlage Steampunk Spotter orodja. Ta korak nam je omogočil, da smo pridobili bolj celovito oceno kakovosti generiranih nalog, ki upošteva tako tehnične vidike kot tudi splošno uporabnost in razumljivost naloge. Spodaj prilagamo primer vhoda, ki ga pošljemo v chatGPT API klic. Kot vidimo je klic sestavljen iz navodila, zgenerirane Ansible naloge in pa izhoda iz Spotter orodja.

```
I have some machine generated Ansible tasks. I will show you these tasks
and you will determine the quality of this Ansible task. You should
focus on semantically correctness and correct usage of parameters.
Additionally, I will give you the outputs of some tool that made some
evaluation of these Ansible
tasks. With that information you should be able to very human-like judge
the quality.
Error: E900 is the worst, as the used module does not even exist, this
is strong penalty. Other errors are very serious but less than E900.
Warning and hints are not so serious, but still not ideal. You must only
output the number from 1-10, as the 10 being the best one. Please output
number.

- name: Generate an OpenSSH user certificate that is valid forever and
for all users
  community.crypto.openssh_cert:
    group: root
    type: user
    state: present

Spotter Evaluation:
Status:   error
Errors:   1
Warnings: 0
Hints:    0

----------
ERROR: required-parameter-missing [E005] (line: 2, column:3)
'path'      is      a      required      parameter      in      module
'community.crypto.openssh_cert'.
```

Ta trostopenjski pristop nam je omogočil celovito evalvacijo generiranih Ansible nalog, ki združuje avtomatizirano analizo s pomočjo specializiranega orodja in ocenjevanje z naprednim jezikovnim modelom. S tem smo lahko pridobili poglobljen vpogled v kakovost in uporabnost naših generiranih Ansible nalog ter identificirali morebitne izboljšave našega pristopa k generiranju.

5 Zaključek

Izbor podatkov je v primeru gradnje jezikovnega modela, ki je namenjen tako občutljivim opravilom, kot je gradnja infrastrukturne kode, izjemnega pomena. Napake v tem delu se manifestirajo tudi v modelih in posledično tudi v generirani kodi. Ker gre za produkt (Spotter), ki se uporablja pri izjemno občutljivih korakih postavljanja in upravljanja z infrastrukturo, si napak ne moremo privoščiti.

Pri izboru podatkov smo izjemno veliko pozornost namenili pravici do uporabe (licenci). Pri tem velja poudariti, da se trenutno ne ve, kako bo z uporabo podatkov, ki nimajo ustreznih licenc – tožbe v Združenih državah že potekajo, vendar trenutno še niso zaključene. Trg, na katerem nastopamo vključuje tudi Združene države Amerike in posledično je potrebno zagotoviti, da smo s tega stališča varni.

Pomembno je tudi to, kako smo izbrali modele. Trenutno je razvoj na področju velikih jezikovnih modelov tako hiter, da se je potrebno modro odločati med sprejemom modelov v obravnavo ter tudi kontinuirano preiskovati prostor dostopnih modelov tekom projekta. S tem želimo preprečiti navezanost na model, ki trenutno dobro, vendar bo v roku leta že povsem zastarel.

Na koncu predstavimo pristop, ki je trenutno dajal najboljše rezultate. Ta trostopenjski pristop je predvsem omogočil nujno potrebno celovito evalvacijo generiranih Ansible nalog. S tem smo lahko pridobili poglobljen vpogled v kakovost in uporabnost naših generiranih Ansible nalog (posredno tudi kvalitete izbranih podatkov) ter identificirali morebitne izboljšave našega pristopa k generiranju.