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

D6.3

-

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**	D6.3
Povezani izročki	D6.3	**Dostop**	Javno
Glavni urednik	Daniel Vladušič	**Glavni avtor**	Nikola Sekulovski
Ostali	Nejc Bat, Jure Medvešek		

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

Kazalo

Kazalo Tabel

Kazalo Slik

Povzetek

Izroček D6.3 nadaljuje delo iz D6.1 in D6.2 ter se osredotoča na izbor, pripravo in obogatitev podatkovnih množic za učenje velikih jezikovnih modelov (LLM) na področju Infrastructure-as-Code (IaC), z osrednjim poudarkom na jeziku Ansible.

Glavni cilj dokumenta je oblikovati konsistenten proces zbiranja, čiščenja in obdelave podatkov, ki omogoča gradnjo modelov, sposobnih natančnega sledenja navodilom in zanesljivega generiranja infrastrukturne kode.

V ta namen smo razširili kriterije za izbor podatkov, opredeljene v D6.1. Poleg kakovosti kode, priljubljenosti, velikosti in licenc smo dodali še kriterij raznolikosti (število unikatnih nalog, kombinacije parametrov znotraj modulov) ter kriterij kakovosti opisov nalog, saj dobro oblikovani opisi neposredno vplivajo na uspešnost modela.

Dokument opisuje iterativno oblikovanje več podatkovnih množic, ki so se med seboj razlikovale tako po velikosti kot po strukturi in kakovosti:

- osnovna množica Steampunk,
- različica z nadvzorčenjem,
- kombinacija Steampunka, sintetičnih podatkov in enotnih testov,
- množice z obogatenimi opisi (description augmentation) in naprednim maskiranjem parametrov,
- manjša, fokusirana množica, zasnovana z več ročnega pregleda in čiščenja,
- končna različica, ki vključuje še prečiščene GitHub podatke.

Vsaka od teh različic je bila skrbno evaluirana s trostopenjskim postopkom:

1. avtomatska analiza s pomočjo orodja Spotter,
2. ocenjevanje kakovosti z modelom GPT-4o-mini,
3. presoja s strani Ansible strokovnjakov.

Rezultati jasno potrjujejo, da zgolj povečevanje količine podatkov ne prinaša napredka, temveč je ključen pristop, ki združuje zadostno velikost učne množice, raznolikost primerov in kakovostno oblikovane opise. Najboljši dosežen rezultat – indeks resnosti (severity score) = 1 je bil dosežen pri zadnji različici podatkovne množice, kjer ni bilo ustvarjenih napak in le eno opozorilo.

S tem izročkom smo utemeljili pomen iterativne gradnje učnih množic in pokazali, da kombinacija širine in globine podatkov ter natančno zasnovanih opisov omogoča razvoj robustnega modela, ki je hkrati natančen in uporaben v praksi.

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.

Po drugi strani, se pojavlja pristop uporabe uporabe majhnih in 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 »fine-tuning«. Le-ta nas, kot malo in srednje veliko podjetje (MSP), zanima. Izkaže se, da so majhni modeli, kjer izvedemo takšno učenje, zelo uporabni, še posebej, ko jim dodamo znanje v obliki dokumentov, ki jih lahko pregledujejo pri svojem pristopu (ang. RAG: Retrieval Augmented Generation).

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. Samo orodje, jezik Ansible, smo opisali v izročku D6.1 in jo na tem mestu izpuščamo.

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] https://steampunk.si/spotter/

1.1 Namen dokumenta in povezava z drugimi deli projekta

Izroček D6.3 je produkt eksperimentalnega razvoja v sklopu RRP6 in logično nadaljevanje izročka D6.1. Podjetje XLAB preko znanj, ki jih pridobiva predvsem v krovnem sklopu RRP2, postavlja temelje ter pili 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 (izboljšave pristopov, opisani v D6.1) 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.

Poročilo D6.3 opisuje izbiro naših podatkovnih množic in sledi strukturi ter ciljem, podobnim kot v D6.1. Bistveno je, da pri tej izbiri sledimo tudi rezultatom in spoznanjem, ki jih dobimo s testi z modeli. Izroček D6.4 predstavlja pristope k učenju, ki so bili uporabljeni na podatkovnih množicah, opisanih v tem poročilu. To pomeni, da je izroček D6.4 neposredno povezan in odvisen od izročka D6.3 (ta izroček).

Sledimo enakemu načinu dela, kot pri prvem dvojcu izročkov (D6.1 in D6.2). Izročka D6.3 in D6.4 sta komplementarna na enak način kot D6.1 in D6.2. D6.3 in D6.4 se naslanjata na ugotovitve in eksperimente, dokumentirane v prejšnjih poročilih in sta njuno logično nadaljevanje.

Čeprav delovni sklop RRP6 ni neposredno povezan z RRP2 v smislu primerov uporabe, so njegovi pristopi, metode, utemeljitve in predvsem znanje v pomoč pri razvoju tega delovnega sklopa – so zelo prenosljivi.

1.2 Struktura dokumenta

Dokument je sestavljen iz treh gradnikov:

- Izbor metod – poglavje 1 – Tukaj opišemo sklepanje in razloge za izbor modela in podatkov;
- Evaluacijski postopek – poglavje 2 – Opišemo strukturo evaluacij – kako evaluiramo zbrane podatke;
- Izbor podatkov – poglavje 3 – Opis postopka izbire podatkov in opravljenih transformacij na podatkih.

2 Izbor metod za izbor in obogatitev podatkov

To poročilo gradi na raziskavah in ugotovitvah, predstavljenih v D6.1 in D6.2. Kot je bilo navedeno v D6.2, je model CodeLlama-7b-Instruct-hf dosegel najboljše rezultate z vidika natančnosti. Nasprotno pa je StarCoder2-3b-Instruct pokazal nižjo natančnost v primerjavi s CodeLlama, vendar je njegova manjša velikost prinesla prednost pri hitrosti inferenc.

V izročku D6.2 smo izbrali StarCoder2-3b-Instruct, vendar smo ugotovili, da z napredkom na področju strojnega učenja in predvsem pričakovanjih zmogljivosti računalniške opreme, lahko uporabimo natančnejši, večji model.

V tem poročilu se zato osredotočamo na predstavitev naših raziskav in rezultatov z modelom CodeLlama-7b-Instruct, saj smo prednost dali natančnosti in sposobnosti sledenja navodilom pred hitrostjo inference.

V tem poročilu sistematično gradimo različne različice nabora podatkov, na katerih model zaporedoma prilagajamo (učimo - fine-tuning) in nato testiramo. Prve eksperimente smo izvedli na naboru podatkov Steampunk, ki nam je služil kot izhodiščna osnova. Z učenjem na teh podatkih smo dobili nezadovoljive rezultate, zato smo v naslednjih iteracijah smo poskušali povečati količino podatkov z nadvzorčenjem (oversampling) ter po testiranju izboljšati ne le količino, ampak tudi raznolikost podatkov. To smo dosegli z vključevanjem podatkov iz enotnih testov orodja Spotter (unit test) ter sintetično generiranih podatkov. Ti eksperimenti so raziskave usmerili v pravo smer in postavili temelje za nadaljnje delo.

Kasnejše iteracije so se osredotočile na izboljšanje kakovosti Ansible nalog, uporabljenih pri učenju, ter na izboljšanje navodil oziroma opisov nalog, ki služijo kot vhodni pozivi (prompts) za model. Ti pristopi so se izkazali kot zelo koristni in so prinesli opazne izboljšave. Naš primarni cilj je bil pridobiti konsistentno množico učnih podatkov, ki omogoča učenje sledenja ukaznim pozivom (promptom) – instruction following.

Zadnja iteracija nabora podatkov je združila vse do sedaj uporabljene pristope: povečanje nabora podatkov (sintetični podatki, podatki iz enotnih testov in iz GitHuba), spremembe Ansible nalog (ponovno maskiranje, čiščenje, ročni popravki) ter obogatitev pozivov z daljšimi in bolj podrobnimi opisi.

3 Evaluacijski postopek

Evaluacijsko metodologijo smo zasnovali tako, da smo s podatki ki smo jih spreminjali in dopolnjevali, na enak način gradili različice modela, ter potem ocenjevali njegove zmogljivosti.

Različice modela smo ocenjevali na ločeni, sintetično ustvarjeni, podatkovni množici ter spremljali različne metrike, ki jih je poda naše orodje Spotter (število napak, število opozoril, ter *indeks resnosti* (utežena vsota števila napak in opozoril)).

Za vsak primer, ki smo ga zgenerirali iz teh podatkov, smo uporabili tudi oceno gpt-4o-mini. Za oceno podatkov smo uporabili povprečje ocen.

Poleg tega smo k testiranju različnih različic modela povabili tudi naše Ansible strokovnjake, ki so zagotovili dodatne povratne informacije.

Če povzamemo celotno evaluacijo podatkov, vidimo, da smo z njimi gradili enake modele (zelo pomembno je, da pri ocenjevanju kvalitete podatkov uporabimo enake nastavitve pri gradnji modelov in predvsem enak model), potem pa kvaliteto modela preverili z uporabo naše standardne evaluacijske množice ter ocenjevali s pomočjo orodja Spotter, gpt-4o-mini in predvsem, s strokovnjakom za Ansible. Slednje jemljemo kot pomembno razširitev postopka evaluacije.

4 Izbor podatkov

4.1 Kriteriji za izbiro podatkov

Pri zbiranju dodatnih podatkov smo se držali kriterijev, ki smo jih defiinirali v izročku D6.1 in so prikazani v Tabela 1.

Tabela 1: Kriteriji za izbor programske kode v učni množici - definirani v izročku D6.1.

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).

V tem razvojnem obdobju smo prišli do nekaterih pomembnih ugotovitev.

Revidirali smo tretji kriterij (Velikost). Velikost podatkovne množice nedvomno pomembna, vendar je enako pomembno upoštevati tudi raznolikost nalog. Z raznolikostjo označujemo število unikatnih Ansible nalog znotraj posameznega modula, število učnih primerov z različnimi opisi (tudi če predstavljajo isto Ansible nalogo), ter število pojavitev parametrov v nalogah znotraj določenega modula.

To lahko razumemo kot širino in globino zajetega znanja v učnih podatkih: podatkovna množica je lahko široka, če vsebuje veliko modulov z mnogimi primeri, ali pa globoka, če vsebuje večje število raznolikih primerov, ki raziskujejo različne kombinacije parametrov znotraj manjšega števila modulov. Naš cilj je seveda doseči oboje – tako širino kot globino.

Dodali smo tudi nov kriterij (Tabela 2), ki se ukvarja s kvaliteto opisov samih nalog. To je pomebno (kritično) pri izboljšavah modela, ki naj sledi navodilom (instruction-following). Vemo, da zelo dobro sestavljeni opisi zmanjšujejo nedoločenost same naloge – pomemben je jasen zapis naloge in eksplicitno podajanje imen in parametrov. Tako se lahko model učinkovito in dobro nauči kateri parametri in njihove vrednosti morajo biti zgenerirani ob določenem navodilu.

Tabela 2: 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 in raznolikost	Zadostna količina podatkov za prepoznavanje vzorcev in generalizacijo. Zadostno število unikatnih Ansible nalog na modul ter kombinacije parametrov.
Licence	Koda izdana pod odprtokodnimi licencami (MIT, Apache 2.0).
Kakovost opisa/poizvedbe oz. prompta	Jasni in natančni opisi nalog ter parametrov, ki zmanjšujejo nejasnost in izboljšajo zmožnost modela za sledenje navodilom.

4.2 Zbrani podatki za učenje

Za izboljšanje delovanja modela smo večkrat (iterativno) in postopoma zasnovali ter izpopolnili več podatkovnih množic, pri čemer smo vsako posebej evaluirali, da bi ugotovili, kaj je potrebno in katere lastnosti podatkov vodijo do boljših rezultatov. Vsaka nova različica podatkovne množice je bila oblikovana posebej z namenom odpraviti pomanjkljivosti prejšnje, kar je vodilo k nenehnemu izboljševanju kakovosti in učinkovitosti. Ta proces ter posamezne podatkovne množice so podrobneje opisani v naslednjih poglavjih.

4.2.1 Množica podatkov: Steampunk

Kot je navedeno v D6.1, sta bila začetna vira primerov množica podatkov, ki smo jo poimenovali Steampunk in množica podatkov, ki smo jih legalno pridobili iz GitHub-a. Preizkusili smo dva pristopa – učenje s podatki, ki so sestavljeni izključno iz Ansible nalog, pridobljenih iz Steampunka (množica podatkov Steampunk), druga pa je združevala primere iz obeh virov – Steampunka in GitHub-a. Model, ki je bil naučen zgolj na podatkih iz Steampunka, je med ocenjevanjem dosegel boljše rezultate kot model, treniran na kombinirani zbirki.

Nadaljnja analiza je pokazala, da je bila splošna kakovost množice podatkov GitHub slaba. Analiza je pokazala, da je bilo mnogo nalog Ansible takšnih, da so vsebovale samo ime modula, namesto celotnega oz. polnega imena (fully qualified collection name – FQCN), ki služi kot unikatna označba (unique identifier). Rezultat te slabosti je, da sta npr. nalogi `ansible.builtin.user` in `sensu.sensu_go.user` prikazani zgolj z imenom modula `user`, kar je povzročalo zmedo pri učenju. Dodatno so bila imena nalog podatkov zbranih z GitHub-a slabo opisana, ali pa res kratka. To je povzročilo, da smo pri učenju dobili izjemno malo

uporabnih informacij, kar je spet poslabšalo kvaliteto podatkov, ki smo jih nameravali uporabiti za učenje.

Zavedamo se, da je uporaba okrajšanih namesto polno kvalificiranih imen ter neustrezno opisanih imen nalog razširjena in pogosto sprejeta praksa. Vendar pa je pri programiranju IaC to potrebno izboljšati, saj se koda IaC redko pregleduje, hkrati pa predstavlja temelj za pravilno delovanje celotnega sistema. Spodaj sta prikazana dva primera – prvi z omenjenimi pomanjkljivostmi in drugi kot priporočena praksa

Primer slabega opisa in opisa imena modula:

```
- name: rabbitmq_clustering | copy erlang cookie
  template:
      src: erlang.cookie.j2
      dest: '{{ rabbitmq_erlang_cookie_file }}'
      owner: rabbitmq
      group: rabbitmq
      mode: 256
      backup: true
  become: true
  when: inventory_hostname != rabbitmq_master
```

Primer dobre prakse (razširjen opis in raba polno kvalificiranih imen):

```
- name: Update sshd configuration safely, avoid locking yourself out
  ansible.builtin.template:
    src: etc/ssh/sshd_config.j2
    dest: /etc/ssh/sshd_config
    owner: root
    group: root
    mode: '{{ value }}'
    validate: '{{ value }}'
    backup: true
```

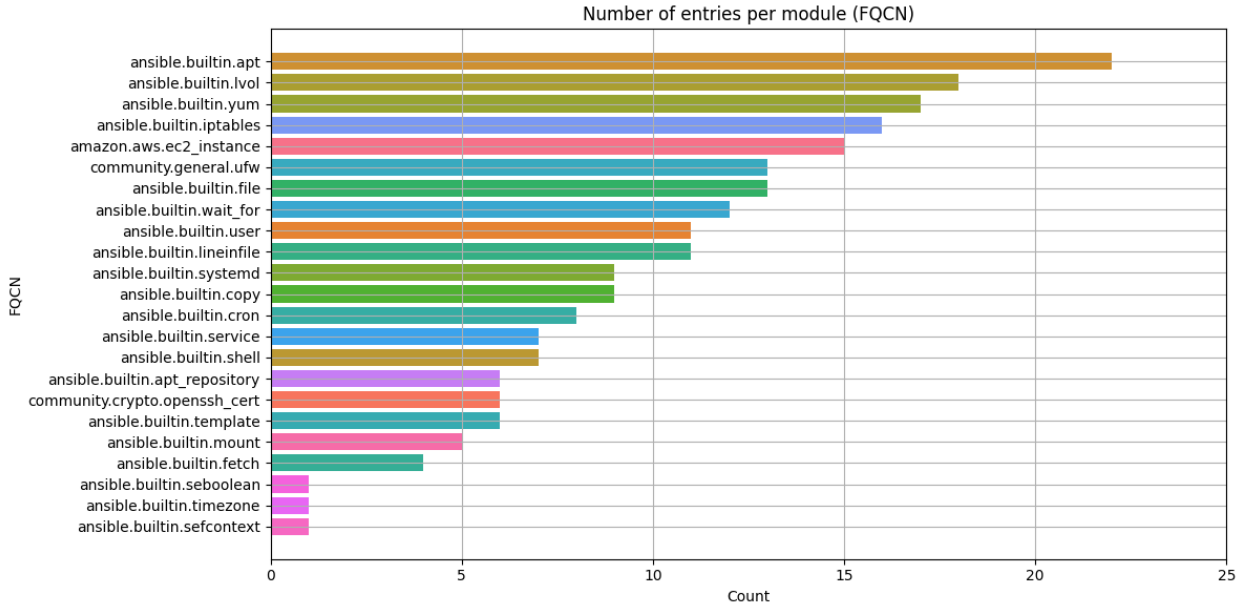
Zato smo se odločili, da se osredotočimo izključno na podatkovno zbirko Steampunk. Začetni postopek čiščenja je ostal enak kot je opisan v D6.1 (filtriranje Ansible nalog iz starejših različic modulov, zakrivanje parametrov z visoko entropijo ipd.).

Statistiko nastale podatkovne zbirke prikazuje Tabela 3 , iz katere je razvidno, da je povprečno 2,16 primera na modul. Ker naš model vrednotimo na ločeni (evaluacijski) množici, ki vsebuje 32 primerov iz 24 različnih modulov, je koristno pregledati in spremljati prisotnost primerov iz teh modulov.

Slika 1 prikazuje Ansible naloge iz modulov, na katerih smo evaluirali naš model. Vidimo, da 22 Ansible nalog za module ansible.builtin.apt, medtem ko imajo moduli ansible.builtin.seboolean, ansible.builtin.timezone in ansible.builtin.sefcontext zgolj en primer, kar ni dovolj. Pomankljivost se pokaže pri uporabi (ko zgradimo model), saj pri učenju nismo videli dovolj (možnih) kombinacij parametrov in njihovih vrednosti ali pa celo parametrov, ki so možni, veljavni, vendar v teh primerih niso bili uporabljeni.

Rezultati naučenega modela na podatkovni zbirki Steampunk so prikazani v

Tabela 4. Rezultati kažejo, da so Ansible naloge, ki jih je generiral naš model, na evaluacijski množici povzročile 2 napaki in 2 opozorili. Napake se obravnavajo kot bistveno resnejše in zato imajo utež 10, medtem ko imajo opozorila utež 1. Tako je končni indeks resnosti (severity score) enak 22.



Slika 1: Število Ansible nalog na modul izbrane podmnožice množice podatkov Steampunk

Tabela 3: Statistika množice podatkov Steampunk

Število primerov	57137
Število različnih modulov	26462
Število različnih zbirk	810

Tabela 4: Rezultati vrednotenja doučenega (fine-tuned) modela na množici podatkov Steampunk

Indeks resnosti (severity score)	22
Število napak	2
Število opozoril	2

Ta podatkovna množica in različica modela, ki smo ga na njej zgradili, sta služila kot naša osnovna referenca za nadaljnje eksperimente. Pomembno je poudariti, da so bili GitHub podatki ponovno vključeni v kasnejši fazi razvojnega cikla kot del enega od naših eksperimentov, z namenom povečanja učne množice.

4.2.2 Množica podatkov: Steampunk, nadvzorčenje (oversampling)

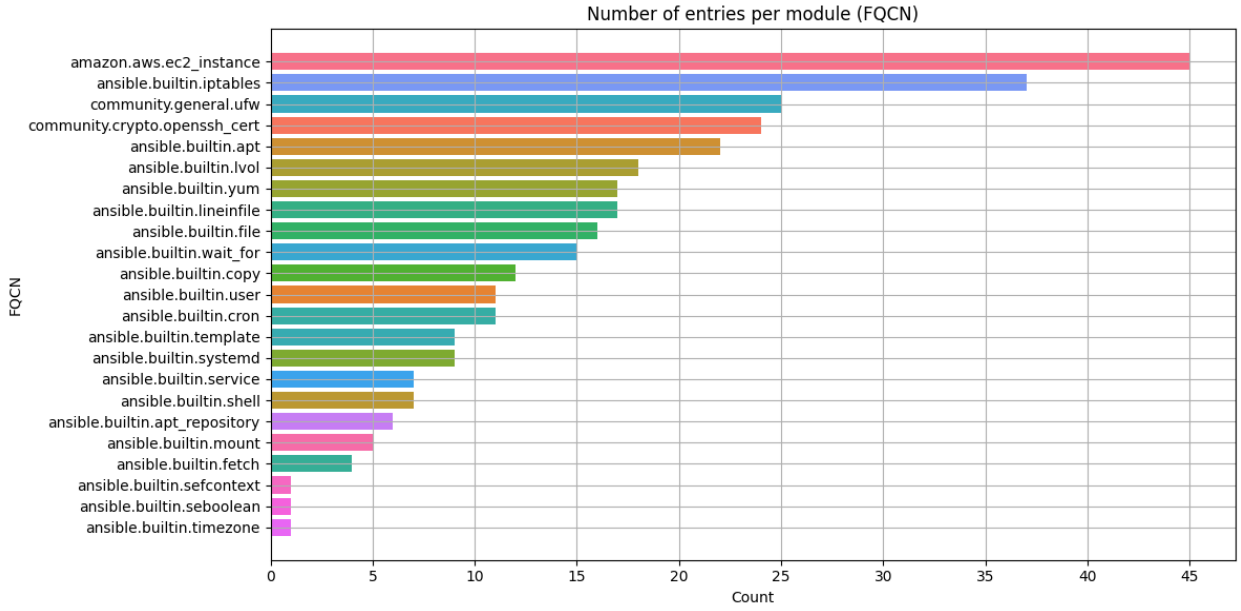
Nadvzorčenje (oversampling) je tehnika podvajanja podatkov. Tipično se ta tehnika uporablja tako, da se določi specifičen faktor s katerim se podvojijo (dejanska kopija) podatki in običajno se to uporablja pri podatkih, ki so namenjeni treningu. Zelo pomembno je, da se ohrani porazdelitev vhodnih podatkov – ločimo namreč takšne tehnike, kjer to porazdelitev lahko spremenimo.

Domneva, ki se upošteva pri uporabi tehnike nadvzorčenja podatkov je, da je samo število (količina) učnih podatkov premajhna za učinkovito učenje (Noor Fatima, 2024). S povečanjem števila učnih podatkov pomaga pri tem, da se izrazijo zakonitosti v učnih podatkih.

V našem eksperimentu smo se osredotočili na povečanje števila primerov, ki imajo več kot šest parametrov. Ta meja je bila izbrana, ker so primeri nalog, z več kot šestimi parametri redki in uporabni, medtem ko je število nalog z manj parametri veliko in običajno manj uporabno in izrazno v učenju. Število takšnih primerov, ki imajo več kot šest parametrov, smo povečali za tri krat. Takšen faktor povečanja smo izbrali s pregledom učnih podatkov – iskali smo ravnovesje med povečanjem učne množice in nevarnostjo prevelike prilagoditve učnim podatkom (overfitting).

Osnovna statistika te učne množice je prikazana na Slika 2 in v Tabela 5. Število primerov z več kot šest parametri je bilo 23233. Z nadvzorčenjem učnih podatkov (faktor 3) in z dodajanjem teh podatkov učni množici je končna velikost naše 126836 primerov.

Kot lahko vidimo v Tabela 6, nadvzorčenje podatkov ni bilo uspešno pri učenju. Nismo dosegli resnega napredka v primerjavi z modelom, ki smo ga učili na množici Steampunk, brez nadvzorčenja podatkov. Utežena vsota opozoril in napak (severity score) se je celo poslabšala.



Slika 2: Število Ansible nalog na modul izbrane podmnožice množice podatkov Steampunk z nadvzorčenjem

Tabela 5: Statistika množice podatkov Steampunk z nadvzorčenjem (oversampling).

Število primerov	126836
Število različnih modulov	26462
Število različnih zbirk	810

Tabela 6: Rezultati vrednotenja doučenega modela na množici podatkov Steampunk z nadvzorčenjem

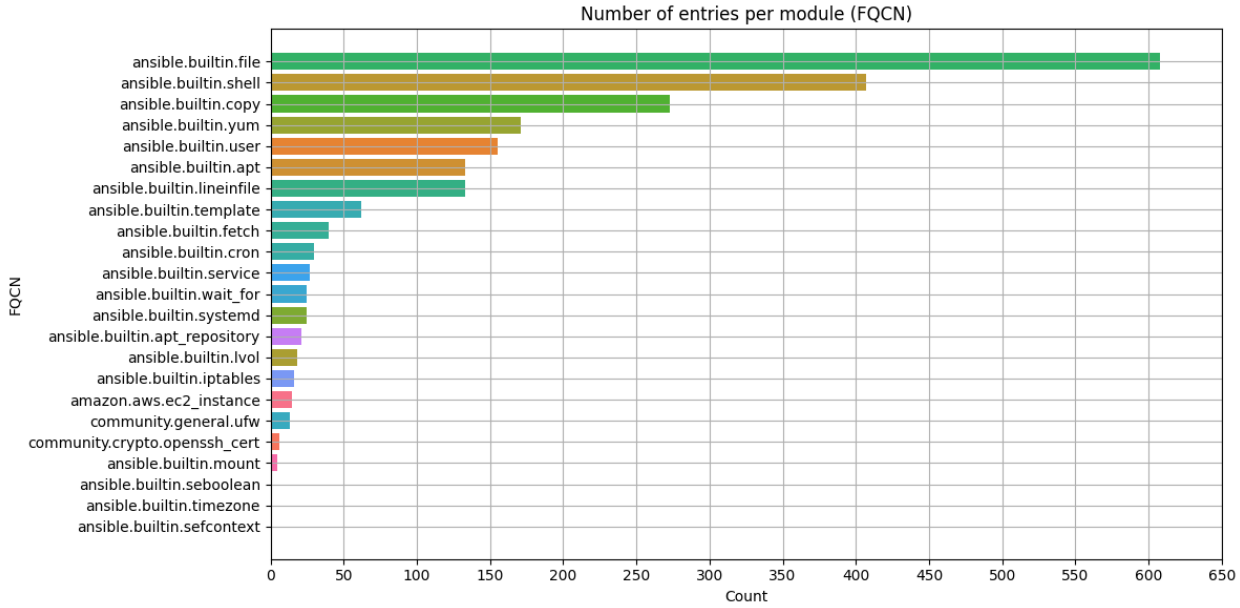
Indeks resnosti (severity score)	23
Število napak	2
Število opozoril	3

4.2.3 Množica podatkov: Steampunk, sintetični podatki in enotni testi (unit test)

Cilj tega eksperimenta je bil podoben kot prej, povečati količino učnih podatkov. Glede na rezultate prejšnjega eksperimenta, smo ugotovili, da je nujno potrebno povečati tudi raznolikost učnih podatkov (Wang et al., 2023).

Ustvarili smo novo podatkovno množico za učenje, kjer so njeni gradniki bili sestavljeni iz primerov Steampunk, enotnih testov (unit test), ki smo jih pridobili od razvojnega tima Steampunk Spotter ter sintetičnih primerov, ki smo jih generirali z rabo GPT-4o-mini. Kot vidimo v Tabela 6 in Slika 3, je velikost nove množice 62233 primerov - 8.92% povečanje.

Rezultati na evaluacijski množici kažejo, da smo s tako oblikovanimi učnimi podatki (tako konstruirano učno množico), dosegli izboljšavo v primerjavi z rezultati, ki jih dobimo s preprostim nadvzorčenjem učnih podatkov. Rezultate kaže Tabela 8, kjer vidimo, da se je t.i. indeks resnosti (severity score) zmanjšal – z rabo takšnih podatkov smo uspeli zmanjšati število napak, a povečati število opozoril.



Slika 3: Število Ansible nalog na modul izbrane podmnožice množice podatkov Steampunk + sintetični podatki + enotni testi

Tabela 7: Statistika množice podatkov Steampunk + sintetični podatki + enotni testi

Število primerov	62233
Število različnih modulov	26497
Število različnih zbirk	840

Tabela 8: Rezultati vrednotenja doučenega modela na množici podatkov Steampunk + sintetični podatki + enotni testi

Indeks resnosti (severity score)	15
Število napak	1
Število opozoril	5

4.2.4 Množica podatkov: Steampunk + obogateni opisi/bogatenje opisov + sintetični podatki + enotni testi + ponovno maskiranje parametrov

4.2.4.1 Začetni poskusi bogatenja opisov in ponovnega maskiranja vrednosti parametrov

V D6.1 smo zapisali, da je pomemben korak pri čiščenju učnih podatkov maskiranje dejanskih vrednosti parametrov. Ti so namreč zelo raznoliki (v jeziku strojnega učenja imajo visoko entropijo). Primeri so npr. API ključi, razni žetoni, ter druge izračunane vrednosti (npr. hash). Pri čiščenju poiščemo takšne vrednosti (postavimo mejo), ter jih zamenjamo z nekim univerzalnim zapisom. S tem želimo doseči, da se model ne nauči vrednosti, ki so de-facto brez pomena. V D6.1 smo opravili menjavo vsake takšne vrednosti z enovito označbo, ne glede na to, za kakšno vrednost je šlo.

V ponovitvi tega eksperimenta smo ubrali drugo pot. Za vsako vrednost, ki jo je bilo potrebno maskirati, smo uporabili specifično oznako (glede na tip, oz. mesto, kjer je takšna vrednost nastopala). Kot primer navajamo preprosto menjavo vrednosti gesla – `password` – njegovo vrednost smo maskirali z `{{password_value}}`.

Dodatno smo za vsako takšno nalogo (primer) pripravili tudi alternativna navodila. Pri vsakem učnem primeru smo generirali tri različna navodila (z uporabo GPT-4o-mini). Navodila so bila različnih dolžin in kompleknosti. Tabela 9 in Slika 4 prikazujeta končno statistiko tako zgrajene učne množice.

Za takšen pristop smo se odločili, ker smo pri pregledovanju učne množice ugotovili, da je velika večina opisov nalog oz. navodil bila zelo slabe kvalitete. Ker velja preprosto pravilo, da je končni model tako dober kot njegovi učni podatki, smo s tem pristopom pričakovali boljše rezultate. Navajamo tudi primere treh Ansible nalog, z različno kvaliteto opisa.

Primer 1:

```
- name: Set JBoss memory settings correctly
  ansible.builtin.lineinfile:
    path: /opt/jboss-as/bin/standalone.conf
    regexp: '{{ regexp_value }}'
    line: '{{ line_value }}'
    backrefs: true
```

Primer 2:

```
- name: Update the memory settings in JBoss configuration file.
  ansible.builtin.lineinfile:
    path: /opt/jboss-as/bin/standalone.conf
    regexp: '{{ regexp_value }}'
    line: '{{ line_value }}'
    backrefs: true
```

Primer 3:

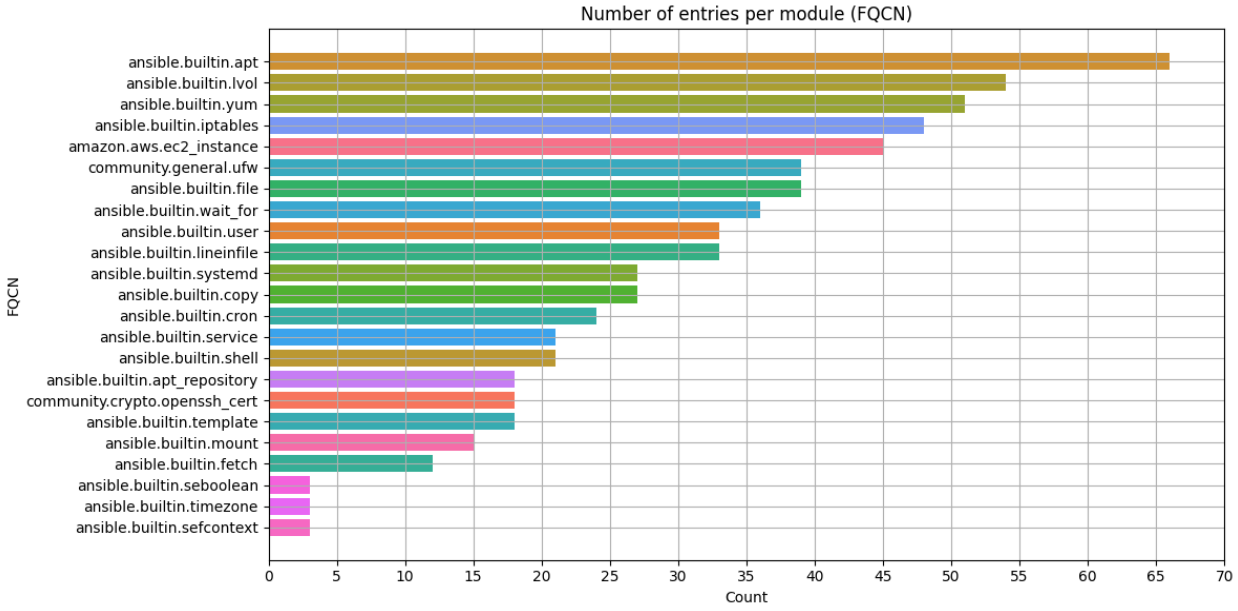
```
- name: Modify the JBoss configuration file to ensure memory settings
match the required values, using a regular expression to find the correct
line and replace it with the specified line.
  ansible.builtin.lineinfile:
    path: /opt/jboss-as/bin/standalone.conf
    regexp: '{{ regexp_value }}'
    line: '{{ line_value }}'
    backrefs: true
```

Izjemno pomembno je poudariti, da je bila gradnja te množice prvič usmerjena v kvaliteto sledenja navodilom (instruction following). S to učno množico smo želeli izboljšati zmožnost modela, da sledi navodilom, ki jih poda uporabnik, namesto da se slepo osredotoča samo na na točnost rezultata.

Rezultati so bili zelo slabi – pravzaprav najslabši doslej. Kaže jih Tabela 10.

Najbolj opazna sprememba je bila, da je model pričel generirati nadomestne vrednosti parametrov, ki so bile specifične za vrsto parametra, namesto da bi generiral generične. To je bilo najprej videti zelo vzpodbudno, vendar je podroben pregled pokazal, da je kvaliteta opisov navodil omejitveni faktor. V teh opisih so namreč manjkale dejanske reference na vrednosti parametrov. Izkazalo se je, da je pri generiranju bila bolj pomembna njihova dolžina kot kvaliteta opisa naloge in njenih parametrov. Brez strukturiranih informacij o pomenu naloge in njenih parametrih, se model ni zmogel naučiti nič relevantnega.

Če povzamemo, se je kljub temu, da je bila velikost učne množice trikrat večja kot originalna Steampunk učna množica, eksperiment izkazal za zelo slabega. Model, ki smo ga s temi podatki učili, ni zmogel slediti navodilom. Evaluacija Ansible ekspertov je pokazala, da model še vedno zelo slabo sledi navodilom, kjer ni pomembno, kako natančna in elaborirana ta navodila dejansko so.



Slika 4: Število Ansible nalog na modul izbrane podmnožice množice podatkov Steampunk z začetnim ponovnim maskiranjem in bogatenjem opisov

Tabela 9: Statistika množice podatkov Steampunk + sintetični podatki + enotni testi s ponovnim maskiranjem in bogatenjem opisov

Število primerov	171408
Število različnih modulov	26461
Število različnih zbirk	810

Tabela 10: Rezultati vrednotenja doučenega modela na množici podatkov Steampunk + sintetični podatki + enotni testi z ponovnim maskiranjem in bogatenjem opisov

Indeks resnosti (severity score)	33
Število napak	3
Število opozoril	3

4.2.4.2 Manjša in osredotočena učna množica, z avtomatskim in ročnim pregledom Ansible nalog in njihovih opisov oz. navodil.

Po posvetu z Ansible eksperti, smo se odločili, da zmanjšamo število Ansible modulov, ki jih upoštevamo pri učenju. To pomeni, da se osredotočimo samo na res pomembne module in zgradimo manjšo učno množico, ki je bolj obvladljiva.

V tako oblikovano novo učni množici, smo najprej vključili originalne Steampunk podatke. Nato smo te podatke obogatili s sintetično zgeneriranimi primeri in primeri enotnih testov, kjer smo sledili podobnemu pristopu kot v sekciji 4.2.4.1. Raje, kot da

bi sledili samo povečanju števila primerov, smo se osredotočili tudi na to, da smo povečali raznolikost primerov za vsak vključeni modul.

Veliko napora smo vložili tudi v razvoj hevristike za maskiranje ali menjavo vrednosti parametrov. V nekaterih primerih smo vrednosti menjali s krajšimi in bolj realističnimi alternativami, ki so bile odvisne od parametra in modula. Dodatno smo tudi restrukturirali Ansible naloge v pravilno obliko, da bi povečali njihovo jasnost in izrazno vrednost. Nadalnje preoblikovanje je bilo narejeno s pomočjo ekspertnega sistema, kjer smo običajne in splošnejše parametre zamenjali z bolj unikatnimi in tako povečali raznolikost v podatkih.

Razvili smo tudi metode, s katerimi smo preprečili pogostost opozoril, ki jih javlja Spotter. Vsak novi primer smo najprej preverili s Spotterjem; Primeri, ki so javili napake, so bili izločeni, medtem ko smo tiste, kjer je Spotter javil zgolj opozorilo popravili. To smo izvedli bodisi ročno, bodisi avtomatsko. S tem pristopom smo dosegli ravnovesje med številom primerov in njihovo kvaliteto – če bi izločili vse tiste, kjer je Spotter javil opozorilo, bi velikost podatkovne množice drastično upadla.

Kot smo omenili v sekciji 4.2.4.1 so bila novo zgrajena navodila napačna. Vsebovala so opise, kaj posamezna naloga počne, korakoma pojasnevala tudi korake v kodi, logične odločitve in njihove pogoje, itd. Takšen pristop bi bil smotrnejši pri dokumentiranju teh nalog. Ubrali smo drugačen pristop ter za vsako Ansible nalogo ustvarili šest opisov oz. navodil, ki so bila različnih dolžin in imela različno izrazno moč. Skupno vsem pa je, da so navodila osredotočena na akcije, ki naj jih model izvede – skorajda tako natančno, kot da bi sestavljali skripto v ukazni vrstici. Osredotočala so se na operacije, uporabljala minimalno število besed, osredotočena na specifične objekte, parametre. Dodatno smo izvedli tudi delitev med opisom in imenom nalogo. V eksperimentih, ki smo jih naredili s to metodo, smo uporabili zgolj opis kot navodilo modelu. Za ta pristop smo se odločili, ker so lahko navodila oz. opisi zelo dolgi in napisani v obliki strojnih navodil in kot taki neprimerni za uporabo v imenu. Spodaj navajamo takšen primer.

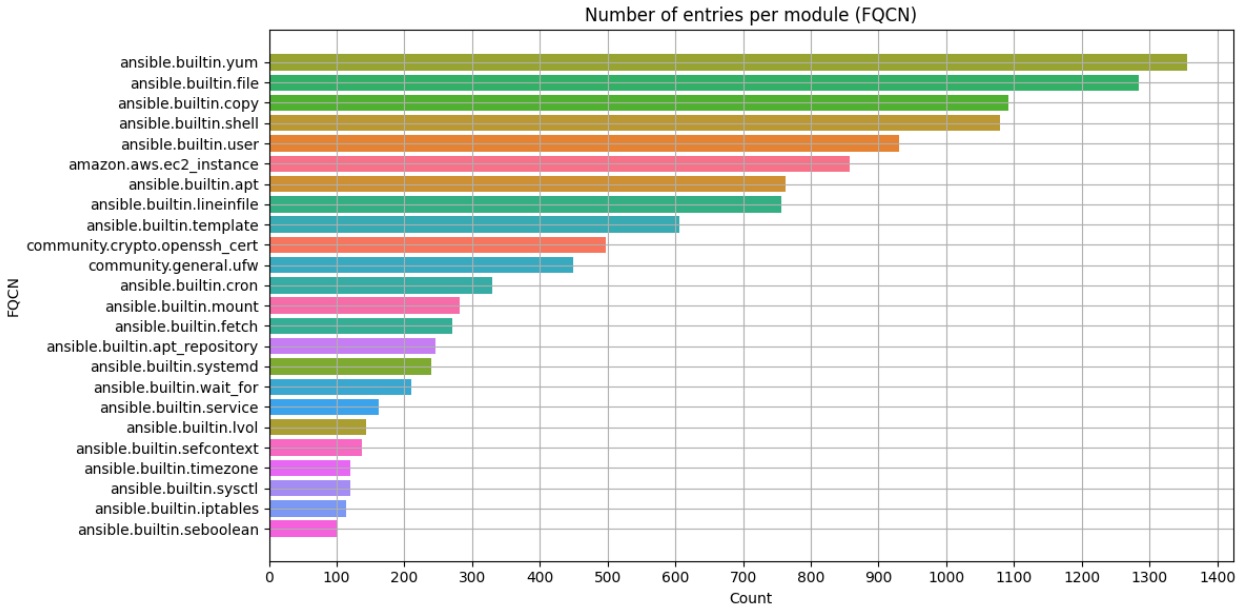
Primer:

```
Input: Create a symbolic link at /usr/local/bin/myapp to
/opt/myapp/myapp, ensuring SELinux level is s0 and state is link.

Ansible task:
- name: task name
  ansible.builtin.file:
    path: /usr/local/bin/myapp
    src: /opt/myapp/myapp
    state: link
    selevel: s0
```

Naš cilj je bil zgraditi množico podatkov, ki omogoča učenje modela, ki se ne osredotoča zgolj na pravilne tipe vrednosti za vsak parameter temveč je sposoben generirati realistične naloge, ki imajo pravilne vrednosti parametrov.

S tako zgrajeno učno množico smo uspeli naučiti model, ki je imel doslej najboljši rezultat na evaluacijski množici. T.i. indeks resnosti (severity score) je bil najnižji, vendar smo vseeno zgenerirali tudi primer, ki je vseboval napako. Kljub vsemu, je pregled in ocena Ansible eksperta potrdila, da je tako zgrajena učna množica dala dobre rezultate, še posebej če jo primerjamo z uporabo modelov, zgrajenih prejšnjih učnih množicah.



Slika 5: Število Ansible nalog na modul izbrane podmnožice iz osredotočene množice podatkov Steampunk + sintetični podatki + enotni testi z ponovnim maskiranjem in bogatenjem opisov

Število primerov	37787
Število različnih modulov	803
Število različnih zbirk	6

Tabela 11: Rezultati vrednotenja doučenega modela na manjši in bolj osredotočeni množici podatkov Steampunk + sintetični podatki + enotni testi z ponovnim maskiranjem in bogatenjem opisov

Indeks resnosti (severity score)	12
Število napak	1
Število opozoril	2

4.2.5 Množica podatkov: Steampunk + obogateni opisi/bogatenje opisov + sintetični podatki + enotni testi + ponovno maskiranje + izpopolnjeni GitHub podatki

Ponovno smo obdelali izvorne GitHub podatke. S tem smo zagotovili, da so bili ti podatki bolj kvalitetni in relevantni. Prvi korak je bil, da smo na podatkih uporabili Spotter in s tem odstranili nepravilne primere. Ta korak je bil ključen, saj so bili podatki zajeti več kot 18 mesecev prej, v tem času pa je bilo več modulov posodobljenih. Z izločanjem zastarelih primerov smo zagotovili, da je bil model učen na najnovejših različicah modulov.

V naslednjem koraku smo preostale primere preoblikovali v ustrezno obliko in imena modulov nadomestili s polnimi imeni kolekcij (FQCN). Nato smo s pomočjo GPT-4o-mini ustvarili ustrezna navodila, oz. pozive (prompte) in opise. V tej iteraciji smo dovolili daljše opise (do 45 besed), a smo omejili vsak primer naloge na en sam poziv ali opis, da bi ohranili konsistentnost in osredotočenost. Na koncu smo tako obdelane podatke dodali k učni množici, opisani v sekciji 4.2.4.2.

Različica modela, ki smo jo s temi podatki zgradili, je dosegla najboljše rezultate do sedaj. To potrjuje tudi evaluacija na evaluacijski množici kot tudi test z Ansible ekspertom. Model je namreč precej bolje sledil navodilom, ki mu jih je dajal Ansible ekspert. Tabela 13 prikazuje najnižji do sedaj dosežen kazalnik indeksa resnosti (severity score), in sicer vrednost 1, kar pomeni, da model ni ustvaril naloge, ki bi jo Spotter označil kot napačno.

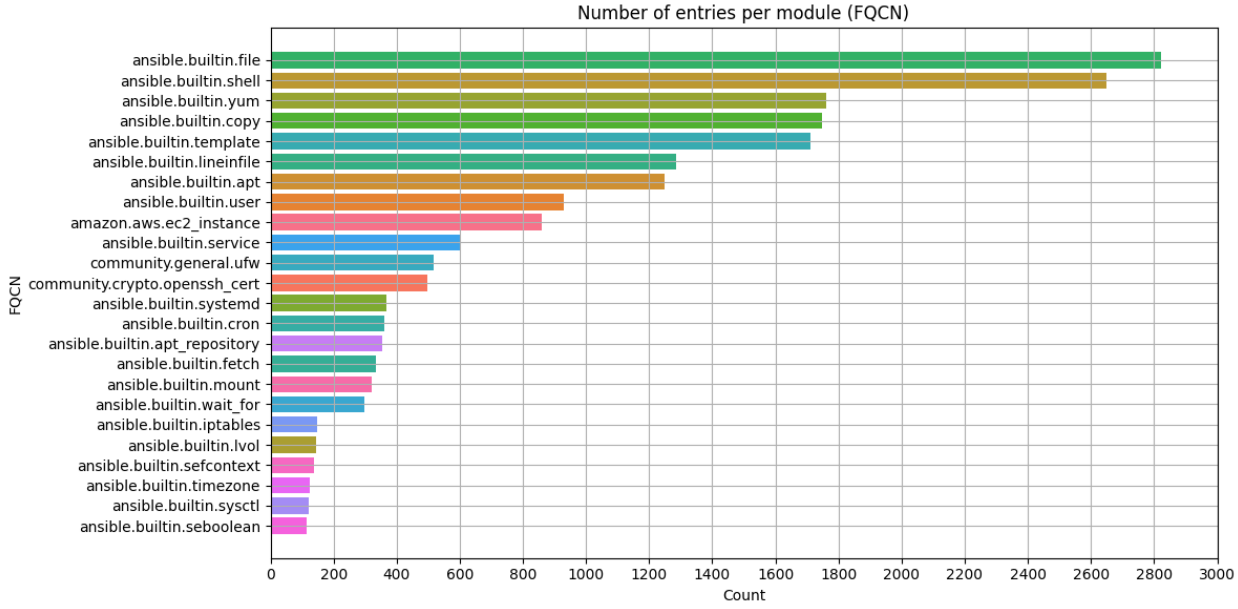
Pri testiranju so naši strokovnjaki za Ansible opazili, da je bila ta različica modela zmožna slediti tudi bolj kompleksnim pozivom, kot je pravilno umeščanje vrednosti več parametrov hkrati – tudi v primerih, ko je bilo parametrov več kot pri modelu iz sekcije 4.2.4.2. Poleg tega

je bil model sposoben ustvariti vrednosti parametrov, četudi ime parametra v navodilu ni bilo eksplicitno navedeno. Na primer, spodnji primer prikazuje, da način »read-only« pomeni vrednost parametra mode: 0444.

Primer:

```
Input: Create file my_dest form template my_source.j2. Using module
ansible.builtin.template. Also set mode to read-only.

Ansible task:
- name: task name
  ansible.builtin.template:
    src: my_source.j2
    dest: my_dest
    mode: '0444'
  register: file_result
```



Slika 6:. Število Ansible nalog na modul izbrane podmnožice iz osredotočene množice podatkov Steampunk + sintetični podatki + enotni testi z ponovnim maskiranjem in bogatenjem opisov + izpopolnjeni GitHub podatki

Tabela 12: Statistika manjše in bolj osredotočene množice podatkov Steampunk + sintetični podatki + enotni testi z ponovnim maskiranjem in bogatenjem opisov + izpopolnjeni GitHub podatki

Število primerov	49978
Število različnih modulov	803
Število različnih zbirk	6

Tabela 13: Rezultati vrednotenja doučenega modela na manjši in bolj osredotočeni množici podatkov Steampunk + sintetični podatki + enotni testi z ponovnim maskiranjem in bogatenjem opisov

Indeks resnosti (severity score)	1
Število napak	0
Število opozoril	1

5 Zaključek

Kakovost in količina podatkov sta ključnega pomena pri učenju vsakega modela strojnega učenja, še posebej pa pri velikih jezikovnih modelih. V tem poročilu smo podrobno predstavili naš iterativni proces ustvarjanja, čiščenja in razširjanja podatkovnih množic. Vsaka različica množice je bila oblikovana z jasnim ciljem: odpraviti pomanjkljivosti svoje predhodnice, tako na področju natančnosti kot tudi sposobnosti sledenja navodilom. S tem iterativnim ciklom smo dosegli merljive izboljšave, ki se odražajo tako v rezultatih evaluacije kot tudi v povratnih informacijah strokovnjakov.

Iz našega razvojnega procesa smo ugotovili, da so pri gradnji natančnega in uporabnega modela ključni trije elementi:

- velikost učne množice oziroma število učnih primerov,
- raznolikost primerov, kjer imajo osrednjo vlogo variacije med Ansible nalogami in parametri,
- ter kakovostno oblikovani in opisni pozivi (prompts), ki modelu omogočajo boljše razumevanje nalog.

Kombinacija številčnosti in raznolikosti učnih primerov na eni strani ter jasnih, nedvoumnih opisov nalog na drugi strani se je izkazala kot temeljna pri vzpostavitvi modela, ki je hkrati natančen in uporaben v praksi.

Na ta način smo v več zaporednih korakih uspeli izboljšati tako sposobnost modela za pravilno generiranje Ansible kode kot tudi njegovo robustnost pri sledenju zahtevnejšim in bolj kompleksnim navodilom.

6 Literatura

Noor Fatima. (2024, November 29). *Mastering LLM Fine-Tuning: A Comprehensive Guide to Data Preparation*. Medium. https://medium.com/%40noorfatimaafzalbutt/data-preparation-the-backbone-of-fine-tuning-large-language-models-1344a48f03fc

Wang, Y., Kordi, Y., Mishra, S., Liu, A., Smith, N. A., Khashabi, D., & Hajishirzi, H. (2023). *Self-Instruct: Aligning Language Models with Self-Generated Instructions*. http://arxiv.org/abs/2212.10560

Zhou, S., Alon, U., Xu, F. F., Wang, Z., Jiang, Z., & Neubig, G. (2023). *DocPrompting: Generating Code by Retrieving the Docs*. http://arxiv.org/abs/2207.05987