SVEUČILIŠTE U SPLITU

FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

POSLIJEDIPLOMSKI DOKTORSKI STUDIJ ELEKTROTEHNIKE I INFORMACIJSKIH TEHNOLOGIJA

KVALIFIKACIJSKI DOKTORSKI ISPIT

Generativne tehnike strojnog učenja za automatsko generiranje programskog koda

Marko Jevtić

Split, veljače 2026.

SADRŽAJ

1. UVOD ... 1
2. AUTOMATSKO GENERIRANJE PROGRAMSKOG KODA ... 3
2.1. Automatsko programiranje ... 4
2.2. Generiranje programskog koda ... 7
2.2.1. Formalna definicija ... 7
2.3. Razlika u procesima ... 8
2.4. Očekivanja od sustava za automatsko programiranje ... 9
2.5. Arhitektura AI sustava nadahnuta sustavima za učenje ... 10
3. GENERATIVNI SUSTAVI UMJETNE INTELIGENCIJE ... 12
3.1. Okvir generativnog modeliranja ... 14
3.1.1. Učenje reprezentacije ... 14
3.1.2. Ključni vjerojatnosni pojmovi ... 15
3.2. Klasifikacija modela strojnog učenja za generiranje programskog koda ... 16
3.3. Drugi biologijom nadahnuti modeli ... 17
3.3.1. Genetsko (evolucijsko) programiranje ... 18
3.3.2. Programiranje jatom ... 20
4. PREGLED LITERATURE ... 23
4.1. Metodologija ... 23
4.2. Rezultati ... 26
4.2.1. Vrednovanje uspješnosti modela ... 32
4.2.2. Klasifikacija radova ... 36
5. ANALIZA POSTOJEĆIH SUSTAVA ZA AUTOMATSKO GENERIRANJE KODA 38
5.1. Taksonomija sustava vođenih prirodnim jezikom ... 38
5.1.1. Oblik programa ... 38
5.1.2. Oblik ulaza ... 39
5.1.3. Oblik izlaza ... 39
5.2. Klase pristupa automatskom generiranju programskog koda vođene prirodnim jezikom ... 39
5.3. Slučajevi korištenja – potrebe zajednice razvojnih programera ... 41
5.3.1. Utjecaj sustava umjetne inteligencije u različitim domenama ... 42
5.3.2. Korištenje sustava vođenih LLM-om za odgovaranje na programerske upite. 43
5.4. Analiza problema i slučajeva korištenja ... 44
5.4.1. Obučavanje programera početnika ... 45

5.4.2. Natjecateljsko računalno programiranje 46
5.4.3. Dovršavanje programskog koda 47
5.4.4. Automatsko generiranje testova 48
5.5. Prikupljanje podataka 48
5.6. Korištenje podataka 49
5.7. Primjer AI modela za automatsko generiranje programskog koda 50
5.8. Oblikovanje korisničkih interakcija 51
5.8.1. Asistentski sustavi za automatsko generiranje programskog koda 51
5.8.2. Vizualizacija podataka 52
6. VREDNOVANJE UČINKOVITOSTI SUSTAVA ZA AUTOMATSKO GENERIRANJE KODA 54
6.1. Vrednovanje učinkovitosti kod natjecateljskog računalnog programiranja 54
6.2. Vrednovanje učinkovitosti kod dovršavanja koda 54
6.3. Vrednovanje višejezičnih modela za dovršavanje koda 54
6.4. Vrednovanje količine resursa potrebnih za automatsko generiranje koda 55
6.5. Vrednovanje učinkovitosti automatski generiranih testova 56
6.6. Utjecaj na formalno učenje programiranja kod programera početnika 56
6.7. Vrednovanje generiranog sadržaja u svrhu obrazovanja programera početnika 57
6.8. Transparentnost AI sustava 58
7. BUDUĆI IZAZOVI I PRILIKE 59
8. ZAKLJUČAK 60
LITERATURA 62
POPIS OZNAKA I KRATICA 69
SAŽETAK 70

Popis slika

Slika 2.1. Odnos između sudionika i domena znanja pri razvoju programske podrške 6
Slika 2.2. Vodopadni model razvoja programske podrške 8
Slika 2.3. Iterativni proces razvoja programske podrške 9
Slika 2.4. Troslojna arhitektura računalnog sustava za učenje 10
Slika 2.5. Pregled gustoće SOM-a 11
Slika 3.1. Kronološki prikaz razvoja sustava za automatsko generiranje programskog koda . 13
Slika 3.2. Primjer algoritma za genetsko programiranje 19
Slika 4.1. Metodologija pregleda literature (PRISMA) 27
Slika 4.2. Mentalna mapa znanstvenih radova prikupljenih tijekom pregleda literature 34
Slika 5.1. Proces vrednovanja AI modela pomoću generiranja jediničnih testova 48
Slika 5.2. Arhitektura AlphaCode modela 51
Slika 5.3. Primjer interakcije programera s asistentskim sustavom 52
Slika 5.4. Stablo odluka pri dovršavanju koda 53

Popis tablica

Tablica 3.1. Vrste modela strojnog učenja za automatsko generiranje programskog koda 17
Tablica 3.2. Klasifikacija jata agenata za sintezu programskog koda 21
Tablica 4.1. Rezultati vrednovanja pretraživača znanstvenih radova 24
Tablica 4.2. Popis relevantnih radova iz područja 28
Tablica 4.3. Rezultati preciznosti modela 32
Tablica 4.4. Tumač vizualizacije 35
Tablica 4.5. Klasifikacija pristupa automatskog generiranju programskog koda 36
Tablica 5.1. Klasifikacija sustava za automatsko generiranje programskog koda vođenih prirodnim jezikom 40
Tablica 5.2. Rezultati klasifikacije uzoraka sa stranice Stack Overflow 41
Tablica 5.3. Primjena AI sustava kao asistencija ekspertima 45
Tablica 5.4. Primjena AI sustava kao asistencija programerima početnicima 46
Tablica 5.5. Prikaz jezične agnostičnosti AI asistenta 47
Tablica 6.1. Rezultati vrednovanja višejezičnih modela za dovršavanje koda 55
Tablica 6.2. Rezultati analize ekonomičnosti AI modela 55
Tablica 6.3. Mjere za vrednovanje generiranog sadržaja 57

1. UVOD

Ideja o automatskom programiranju gotovo je stara koliko i računalno programiranje [1]. Od samih početaka programiranja, potreba za automatizacijom dijelova procesa razvoja programske podrške motivirala je inženjere i znanstvenike za pronalaskom novih, boljih tehnika i metoda za razvoj programske podrške, ne bi li se olakšao proces njenog razvoja, ali i povećala kvaliteta i pouzdanost krajnjeg proizvoda. No, u međuvremenu su ideje o automatizaciji razvojnog procesa prizemljene zbog tehnoloških ograničenja, a fokus se stavio na automatizaciju procesa pisanja tj. generiranja programskog koda [1]. Time se čovjeka (programera, razvijača / dizajnera programske podrške) zadržalo kao aktivnog sudionika u procesu razvoja programske podrške. Postavljanje realnih očekivanja prema sustavu za automatsko generiranje programskog kôda proizlazi iz shvaćanja kako trenutno postoji previše domena u kojima se javlja potreba za programiranjem kao inženjerskim, obrazovnim ili znanstvenim alatom.

Paralelno s evolucijom programerskih tehnika i paradigmi, mijenjala se i ideja o automatskom programiranju. U samim početcima, asembleri i kompajleri ispunjavali su očekivanja programera po pitanju automatizacije razvojnog procesa. S današnjim arhitekturama računala sačinjenim od stotina grafičkih i procesorskih jedinica, u mogućnosti smo obraditi velike količine podataka [2]. S druge strane, razvojem umjetne inteligencije (*eng. Artificial Intelligence, kr. AI*) i tehnika poput genetskih algoritama, višeagentskih i ekspertnih sustava te neuronskih mreža, javila se mogućnost za ambicioznijim pogledom na područje automatskog generiranja programskog koda. Osim što računala mogu obraditi velike količine programskog koda, implementacijom i primjenom algoritama strojnog (dubokog) učenja, kod računalnih sustava se razvila sposobnost učenja programiranja temeljenog na velikim količinama podataka.

Veliku popularnost trenutno uživaju tzv. veliki jezični modeli [3]. Postojanjem velikih repozitorija programskog koda poput *Stack Overflow*-a i *GitHub*-a, moguće je oblikovati velike podatkovne skupove za uvježbavanje modela u svrhu automatskog generiranja programskog koda. Takav oblik sustava umjetne inteligencije nazivamo generativnim sustavima jer iz postojećeg podatkovnog skupa može generirati nove reprezentativne podatke koji nisu bili dio tog skupa [4].

Vođeni taksonomijom sustava za automatsko programiranje postavljene 80-ih godina prošlog stoljeća te troslojnom arhitekturom sustava za učenje, ovim radom predstavljamo hibridnu taksonomiju AI agenata koja povezuje sustave uzlaznog pristupa, sustave unutar uske domene i asistentske sustave sa sustavima za upravljanje podacima, sustave za upravljanje

učenjem te sustave za upravljanje znanjem. Stvaranjem takve hibridne taksonomije i primjene troslojne arhitekture sustava za učenje kod strojnog učenja u svrhu generiranja programskog koda bit ćemo u mogućnosti bolje analizirati arhitekture AI sustava koje budemo koristili u budućim istraživanjima, ali i vrednovati njihove rezultate jer će i vrednovanje biti odvojeno u tri nezavisna koraka.

Kako bismo prikupili potrebne podatke i saznanja o aktualnim trendovima primjene AI sustava za automatsko generiranje programskog kôda, proveli smo pregled literature. Ovim radom predstavljamo različite tehnike izgradnje takvih AI sustava, njihove prednosti, ali i nedostatke. Fokus smo stavili na neuronske mreže, trenutno najkorišteniju tehniku za oblikovanje generativnih AI sustava [5]. Time smo ispunili svrhu ovog rada, a to bi bilo postavljanje teoretskih temelja za buduća istraživanja u području automatskog generiranja programskog koda.

Stoga smo ovaj rad podijelili na šest cjelina, a one su sljedeće:

1. cjelina u kojoj se fokusiramo na automatsko generiranje programskog koda kao problem istraživanja
2. cjelina u kojoj općenito govorimo o generativnim sustavima umjetne inteligencije
3. cjelina u kojoj opisujemo metodologiju te analiziramo rezultate pregleda literature provedenog u sklopu ovog rada
4. cjelina u kojoj prikazujemo rezultate analize postojećih sustava za automatsko generiranje programskog koda pronađenih tijekom pregleda literature
5. cjelina u kojoj vrednujemo prethodno analizirane sustave za automatsko generiranje programskog koda
6. cjelina u kojoj navodimo očekivani doprinos području, kao i tekuće probleme područja te buduće izazove i prilike

2. AUTOMATSKO GENERIRANJE PROGRAMSKOG KODA

Računalno programiranje je metoda za rješavanje problema opće svrhe, kako svakodnevnih, tako industrijskih i znanstvenih [6]. Iz tog razloga razvili su se brojni programski alati kako bi programerima omogućili što učinkovitiji razvoj programske podrške [7], a s druge strane učinili programiranje pristupačnijim tehničkim područjem [8].

Suptilna razlika između programiranja i kodiranja stvorila je jaz u literaturi te se javlja nekoliko naočigled sinonimnih izraza:

- *automatsko programiranje*
- *(automatsko) generiranje (programskog) kôda*
- *sinteza programa*

Razlika među navedenim pojmovima proizlazi iz širine pogleda na problematiku. Kada se govori o automatiziranom rješavanju problema, ispravno bi bilo koristiti pojam *automatsko programiranje* jer programiranje predstavlja metodu rješavanja problema te je širi pojam od samog kodiranja. Kodiranje je jedna od aktivnosti u programiranju te se odnosi na korak pisanja kôda u jednoj ili više odabranih tehnologija (npr. programskih jezika, okvira, biblioteka, itd.).

S druge strane, sinteza programa razlikuje se od automatskog programiranja u načinu predstavljanja specifikacije problema [9], [10]. Sinteza programa zahtjeva specifikaciju koja po prirodni nije algoritamska, poput specifikacije pisane jezikom formalne logike, dok automatsko programiranje koristi prirodni jezik, UML dijagrame ili reprezentativne parove ulaznih i izlaznih podataka kako bi se opisala ciljana funkcionalnost algoritamskim putem. Ako se prirodni jezik uzme kao programski, možemo poistovjetiti automatsko programiranje i generiranje kôda, pošto se time uvodi nova paradigma programiranja, koja se u literaturi naziva **programiranje prirodnim jezikom** [11].

Potreba za automatskim programiranjem, generiranjem kôda i sintezom programa javlja se zbog sljedećih problema [11]:

- cijena učenja novih programskih jezika kod programera početnika je visoka
- složenost programske podrške postaje veća što dovodi i programere eksperte do poteškoća razumijevanja programskog kôda kojeg je pisala druga osoba
- programski jezici ograničavaju slobodu ljudskog izražaja

2.1. Automatsko programiranje

Automatsko programiranje jedan je od najstarijih problema u području računalstva jer vuče korijene iz ranih problema u područjima programiranja, umjetne inteligencije, kibernetike i automatike [1].

Pojam "*automatsko programiranje*" javlja se 50-ih godina prošlog stoljeća [12], a tadašnje sustave za automatsko programiranje danas se naziva asemblerima. Primjer je obitelj asemblera koji se nazivaju „*autocoder*", a koji su bili dostupni u IBM-ovim računalima 1950-ih i 1960-ih godina prošlog stoljeća.

S druge strane, istim pojmom su se obilježavali i neki programski prevodioci tzv. kompajleri poput Fortrana. Razvojem računalstva paralelno se razvijao i koncept automatskog programiranja, a rastom računalne moći rastao je i tehnološki apetit čovječanstva.

Evolucijom programiranja kao znanstveno-inženjerskog područja, pronađena su rješenja na ukazane probleme, što je potaklo na evoluciju i samog pojma automatskog programiranja te podizanja očekivanja. S vremenom se javio i strah ili zabrinutost kako će automatsko programiranje izbaciti potrebu za ručnim programiranjem. Sustavi za automatsko programiranje bi se u tom slučaju sastojali od sljedeće tri ključne značajke [1]:

- orijentiranosti ka krajnjim korisnicima te sposobnosti komunikacije s krajnjim korisnicima
- opće svrhovitosti tj. sposobnosti za rad u bilo kojoj poslovnoj domeni
- potpune automatiziranosti u kojem slučaju ne bi zahtijevali ljudsku intervenciju

No, stvarna slika bila je nešto drugačija. Tehnološka ograničenja tog doba zahtijevala su od sustava orijentiranih ka krajnjim korisnicima posjedovanje velike količine domenskog znanja, što je značilo da su ti sustavi trebali biti domenski eksperti ne bi li kratko i jasno mogli prenijeti složene ideje iz specifične domene. Ovim radom pokazat ćemo kako je takva slika ostala vjeran prikaz sustava za automatsko programiranje.

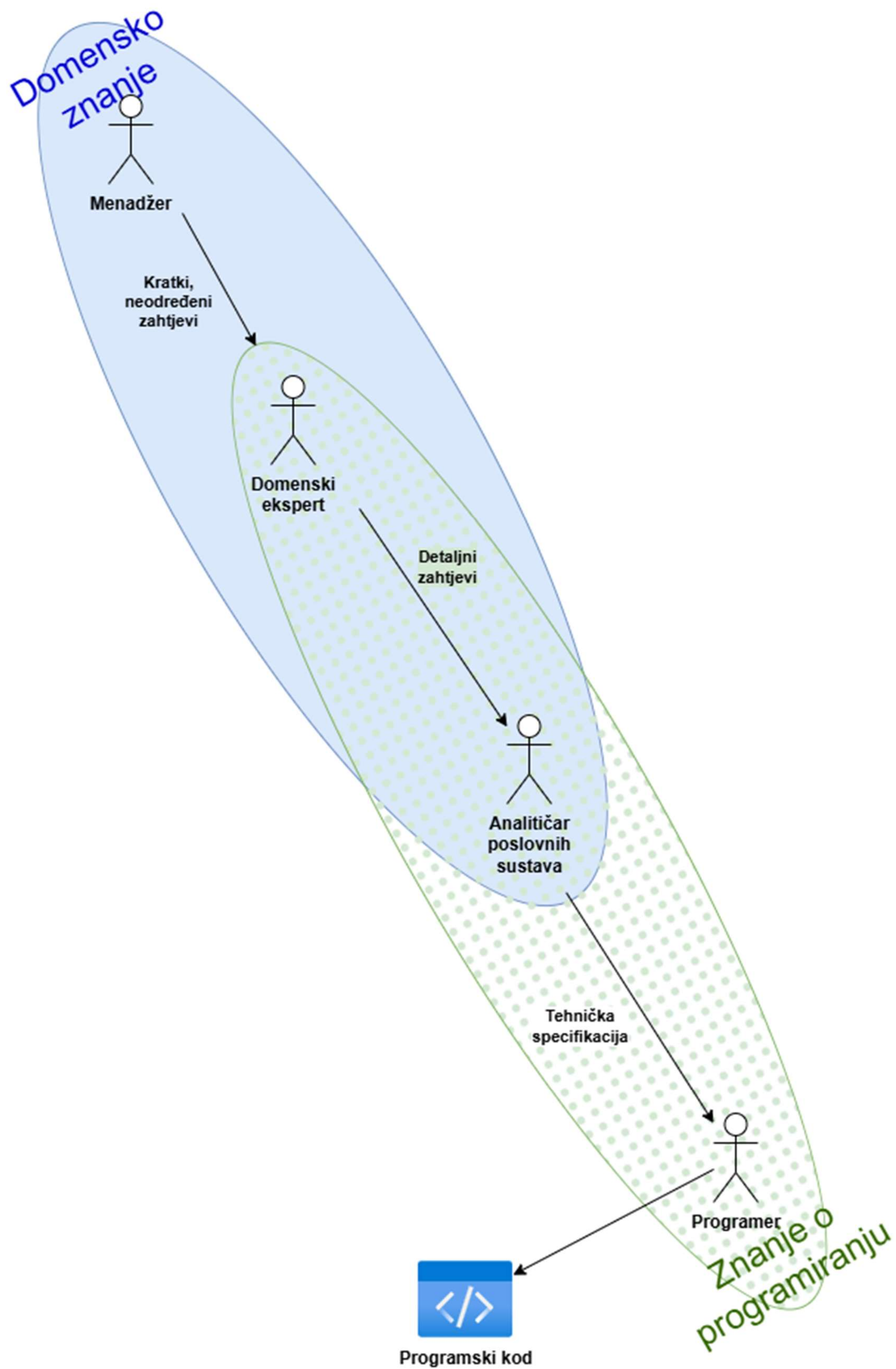
80-ih godina prošlog stoljeća, autori Charles Rich i Richard C. Waters sa sveučilišta MIT, ukazali su na to kako je riječ o senzacionalističkom pogledu na automatsko programiranje [1]. Na temelju svojih zapažanja, zaključili su kako je nemoguće imati sustav koji posjeduje sve tri prethodno navedene značajke jer bi to značilo kako je sustav ekspert u svim postojećim područjima. Stoga se razvila taksonomija sustava za automatsko programiranje kojom se pokušao prizemljiti pogled na samu problematiku.

Slijedi klasifikacija AI sustava za automatsko programiranje [1]:

- **sustavi uzlaznog pristupa** (*eng. bottom-up)* – riječ je o sustavima kod kojih se žrtvuje orijentiranost ka krajnjim korisnicima, a najbolji primjer ovakvog sustava je sustavni razvoj programskih jezika od strojnog jezika prema jezicima vrlo visoke razine
- **sustavi unutar uske domene** (*eng. narrow domain*) – ovakvi sustavi posjeduju sposobnost potpuno automatizirano generirati kôd te komunicirati s krajnjim korisnicima
- **asistentski sustavi** – cilj ovakvih sustava leži u bivanju integralnim dijelom razvojnog procesa vođenog čovjekom, što je u literaturi poznato i kao *čovjek u petlji* [13], [14]

Osim problematike znanja tj. ekspertize sustava za automatsko programiranje, nailazimo i na problematiku dinamičnosti radne okoline tog sustava. Zahtjevi korisnika nisu konstanti skup podataka već su skloni promjenama. Ne može se očekivati potpunost specifikacije i korisničkih zahtjeva [1], što problem automatskog programiranja čini izazovnijim.

Na razliku između automatskog programiranja i generiranja programskog kôda najbolje ukazuje činjenica kako je u proces razvoja programske podrške uključeno više ljudi od samog programera, drugim riječima, više domena znanja od isključivo domene programerskog znanja (Slika 2.1.).



Slika 2.1. Odnos između sudionika i domena znanja pri razvoju programske podrške, izvedeno iz [1]

2.2. Generiranje programskog kôda

Generiranje programskog kôda je polje unutar područja umjetne inteligencije gdje se na temelju izvora višeobličnih podataka poput nedovršenog kôda, programa napisanih u drugom programskom jeziku, prirodno-jezičnih opisa ili primjera izvršavanja pokušava predvidjeti eksplicitan programski kôd ili programska struktura [15].

Generiranje programskog kôda svodi se na pretraživanje opširnog strukturiranog prostora sačinjenog od mogućih programa, no s jako oskudnim signalom nagrađivanja [6]. Čak i mala izmjena u generiranom sadržaju čini razliku između ispravnog i neispravnog programa, jer takva izmjena može izazvati sintaktičku, logičku ili funkcionalnu pogrešku. Iz tog razloga su se prethodna istraživanja često ograničavala na domenske programske jezike [16] ili na generiranje malih isječaka programskog kôda [17].

Dovršavanje programskog kôda jedna je primjer slučajeva korištenja generatora programskog kôda. Postojeći sustavi za dovršavanje programskog kôda služe kao pomoć programerima pri kodiranju te su najčešće dio integriranog razvojnog okruženja (*eng. Integrated Development Environment, kr. IDE*), a fokusirani su na specifične vrste tokena ili značajki programskog jezika. S druge strane, AI sustavi imaju širi fokus te u obzir uzimaju kontekst u kojem se javlja potreba za dovršavanjem kôda. Zbog toga se tvrdi kako AI sustavi imaju holistički pogled na kôd [18].

2.2.1. Formalna definicija

Generiranje programskog koda može se formalno definirati kao *zadatak rješavanja programerskog problema tj. generiranja rješenja **r** na temelju konteksta **C*** [19].

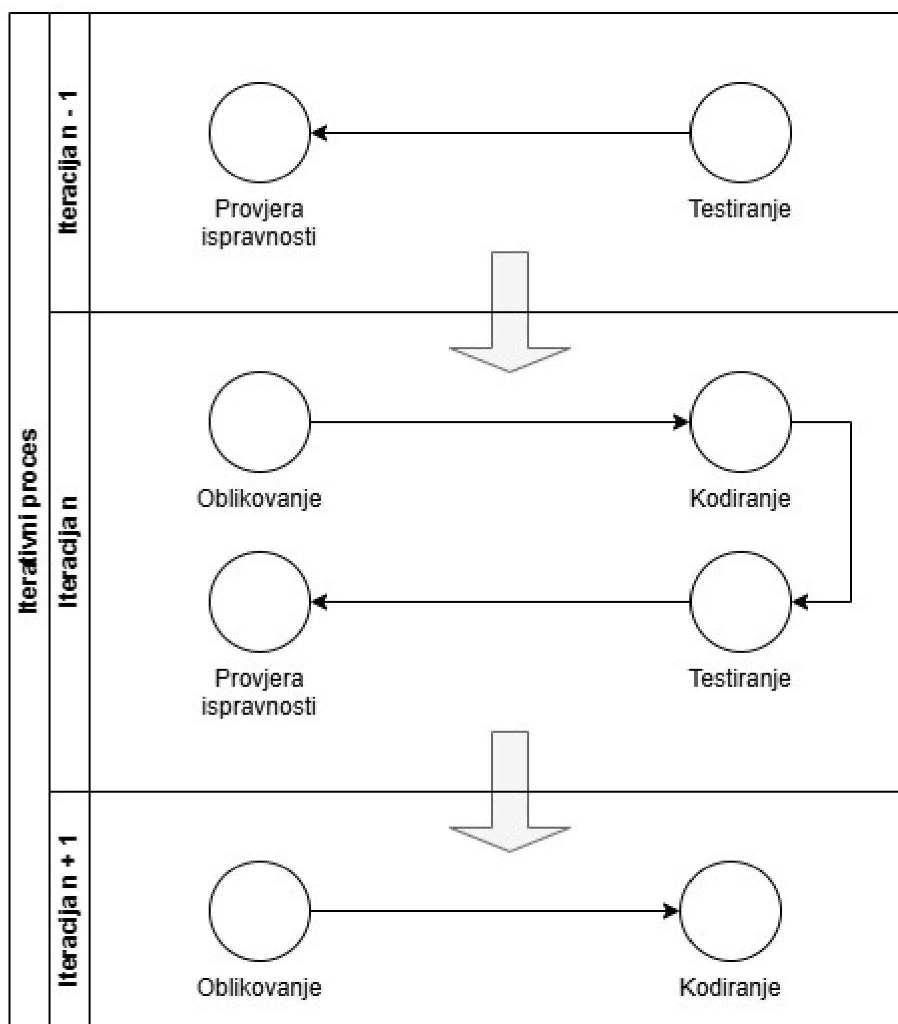
Iz pregleda literature, čiji su rezultati detaljno analizirani u četvrtom poglavlju ovog rada, vidi se kako kontekst *C* može imati više oblika, a čest oblik je opis programerskog problema pisan prirodnim jezikom.

2.3. Razlika u procesima

Automatsko programiranje iterativan je proces poput računalnog programiranja, dok je generiranje programskog kôda proces koji možemo prikazati vodopadnim modelom razvoja programske podrške (Slika 2.2.). Postoji miskoncepcija o tome kako je programiranje serijski proces, no u praksi je riječ o iterativnom procesu (Slika 2.3.) [1]. Pitanje je samo kada će se javiti potreba za iteracijom u procesu razvoja programske podrške.

Slika 2.2. Vodopadni model razvoja programske podrške, izvedeno iz [20]

Vodopadni model razvoja programske podrške iz tog su razloga zasjenile agilne metodologije razvoja [21].



Slika 2.3. Iterativni proces razvoja programske podrške, izvedeno iz [20]

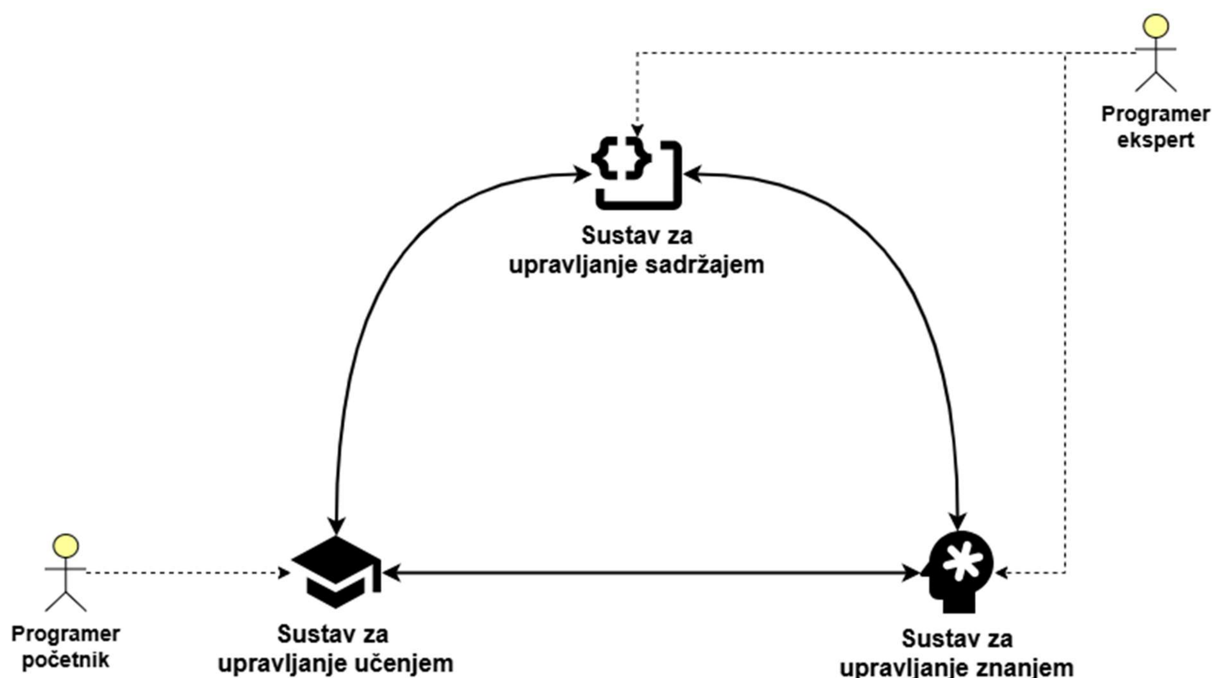
2.4. Očekivanja od sustava za automatsko programiranje

Sustavi za automatsko programiranje, sustavi za generiranje programskog kôda, kao i sustavi za sintezu programa danas su suočeni s istom miskoncepcijom koja ih prati još od 80-ih godina prošlog stoljeća, ali i sa strahovima od sustava za automatizaciju. Ovi sustavi preuzimaju zadaće unutar procesa razvoja programske podrške ne bi li olakšali cjelokupni proces, ubrzali proizvodnju, ali i povećali kvalitetu krajnjeg proizvoda, no nikada neće dosegnuti razinu autonomije čitavog razvojnog tima [1]. Stoga je važno postaviti stvarna očekivanja vođena sljedećim pitanjima:

- što krajnji korisnik vidi?
- kako sustav djeluje (uči)?
- što sustav zna?

2.5. Arhitektura AI sustava nadahnuta sustavima za učenje

U srži prethodno postavljenih pitanja nalaze se tri entiteta – podatci, proces učenja i znanje. Iz razloga što kod strojnog učenja možemo povući interdisciplinarnu paralelu sa računalnim sustavima za učenje koji su čest alat za prenošenja znanja o programiranju, u mogućnosti smo razmotriti stvaranje arhitekture AI sustava koja je strukturalno izomorfna arhitekturama sustava za učenje (Slika 2.4) [22].



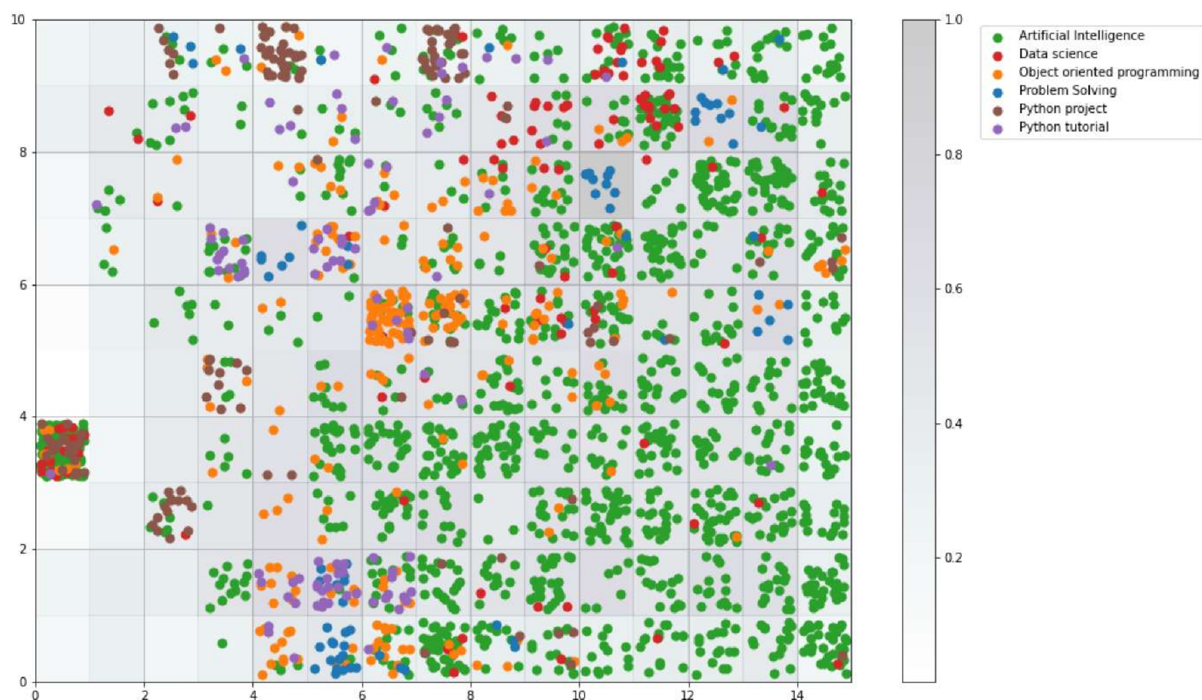
Slika 2.4. Troslojna arhitektura računalnog sustava za učenje, izvedeno iz [22]

Arhitektura modernih sustava za učenje najčešće se sastoji od sljedeća tri ključna dijela [22]:

- sustava za upravljanje sadržajem (*eng. Content Management Systems, kr. CMS)* koji bi u kontekstu strojnog učenja dali odgovor na pitanje *„što krajnji korisnik vidi?“*. U takvom sustavu upravljalo bi se podacima za uvježbavanje i vrednovanje AI sustava, npr. programski kôd.
- sustava za upravljanje učenjem (*eng. Learning Management Systems, kr. LMS)* koji bi nudili odgovor na pitanje *„kako sustav djeluje (uči)?“*. Ovaj sustav služio bi za upravljanje i održavanje procesa strojnog učenja te generiranja podataka. Sustav bi uz to pohranjivao i metapodatke o aktivnostima AI sustava tijekom različitih faza strojnog učenja kako bi se apriori povećala transparentnost AI sustava. O transparentnosti više govorimo u posljednjem poglavlju ovog rada.

- sustava za upravljanje znanjem (*eng. Knowledge Management Systems, kr. KMS)* koji pružaju uslugu pohranjivanja, organiziranja i predstavljanja znanja, a u kontekstu AI sustava bi mogli dati odgovor na pitanje „*što taj sustav zna?*“. Ovakvi sustavi su puno češći kod sustava cjeloživotnog učenja te poslovnog učenja.

Korištenjem tehnika strojnog učenja moguće je premostiti neke jazove koji postoje među ova tri sustava kako bi proces bio što uglađeniji, a tu mogućnost pokazali smo istraživačkim radom gdje smo korištenjem samo-organizirajućih mapa *(eng. Self-organizing maps*, *kr. SOM*) postigli razinu organiziranosti među podacima koja podsjeća na obrazovni program u sustavima učenja programiranja [23], ali daje i odgovor na pitanje *oko kojeg programskog koncepta su se prikupljeni podaci grupirali tj. organizirali*. Uz to pruža i presliku podatkovnog skupa u latentnom prostoru gdje se lako mogu vidjeti nedostatno predstavljeni programski koncepti (Slika 2.5). Drugim riječima, u nekoj mjeri smo premostili jaz između podataka i znanja. Postojanje mjere zaslužuje daljnje proučavanje kako bismo došli do kvalitativnih podataka primjenom egzaktnih metoda.



Slika 2.5. Pregled gustoće SOM-a [23]

3. GENERATIVNI SUSTAVI UMJETNE INTELIGENCIJE

Generativna umjetna inteligencija za cilj ima dati odgovor na sljedeće pitanje [4]:

“Možemo li stvoriti nešto što je samo po sebi kreativno?”

Dolazak do odgovora na postavljeno pitanje sastoji se od mnogo manjih ciljeva koje ćemo prikazati u ovom radu, kao i tehnike čijom se primjenom ti ciljevi mogu ostvariti. No, valja napomenuti kako među njima ne postoji objektivno apsolutno najbolja ili najgora tehnika. Moderni pristupi povezuju pozadinske ideje postojećih tehnika ne bi li stvorili hibridne tehnike kombiniranjem koncepata iz različitih područja generativnog modeliranja [23].

Generativno modeliranje definira se kao „*grana strojnog učenja gdje se uvježbavanjem modela dolazi do mogućnosti stvaranja novih podataka sličnih onima iz podatkovnog skupa za uvježbavanje*“ [24]. Riječ je o interdisciplinarnom području unutar kojeg se zahtjeva predznanje iz mnogih znanstvenih grana poput računalnog programiranja, linearne algebre te opće teorije vjerojatnosti. Razlog tome je potreba da generativni model bude *vjerojatnostan* umjesto *determinističan*, s obzirom na očekivanje da iz identičnog zahtjeva na izlazu trebaju vrlo vjerojatno proizaći dvije različite varijacije očekivanog odgovora. To zahtjeva uključivanje komponente koja uvodi nasumičnost u sustav.

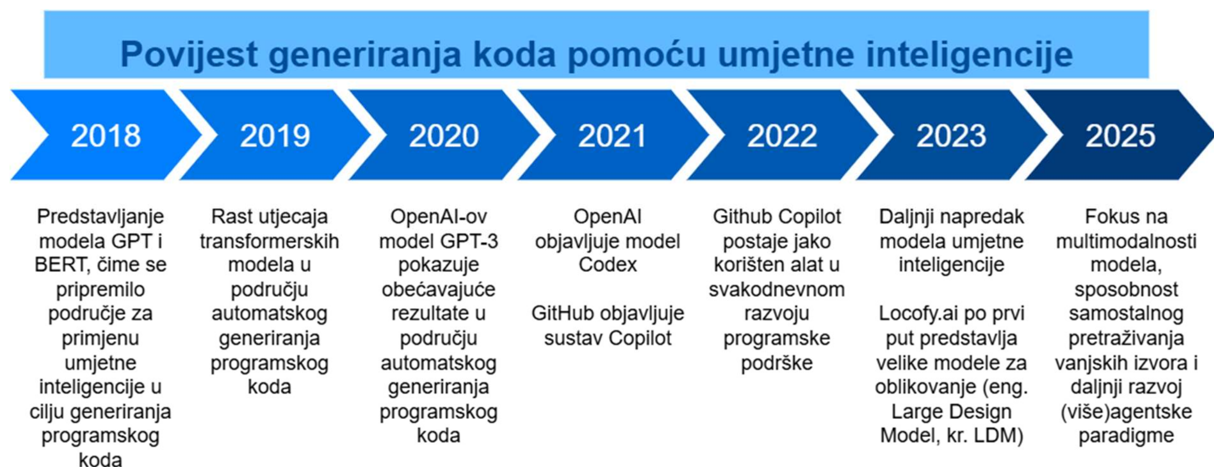
Kako bismo izgradili generativni model, potreban nam je podatkovni skup koji predstavlja entitete očekivane kao rezultat generativnog procesa. Takav skup podataka nazivamo *podatci za uvježbavanje*, a svaki element tog skupa nazivamo *opservacijom*.

Pandan generativnom modeliranju je diskriminativno modeliranje, a poznavanje obaju vrsta te razlika među njima ključno je kako bismo razumjeli ciljeve generativnog modeliranja. Razlike možemo prikazati sljedećim matematičkim zapisom:

- diskriminativno modeliranje procjenjuje vjerojatnost $p(y \mid \mathbf{x})$ te za cilj ima modelirati vjerojatnost oznake y za danu opservaciju $\mathbf{x}$
- generativno modeliranje procjenjuje vjerojatnost $p(\mathbf{x})$ te za cilj ima modelirati vjerojatnost susreta s opservacijom $\mathbf{x}$, gdje uzorkovanje iz dane distribucije omogućava generiranje novih opservacija
    - uvjetovano generativno modeliranje procjenjuje vjerojatnost $p(\mathbf{x} \mid y)$ - za cilj ima modelirati vjerojatnost susretanja s opservacijom $\mathbf{x}$ označenu oznakom y. Primjer ovakvog modeliranja je generiranje programskog koda iz podatkovnog skupa koji se sastoji od programskih kodova pisanih različitim programskim jezicima, a za koji želimo dobiti rješenje problema pisano u jednom od ponuđenih programskih jezika.

Diskriminativno modeliranje u početku je bilo zastupljenija vrsta modeliranja u području strojnog učenja iz razloga što su problemi diskriminativne prirode jednostavniji. Puno je lakše odrediti u kojem programskom jeziku je pisan kôd nego li riješiti problem u tom programskom jeziku. No, zbog sazrijevanja tehnologija vezanih uz strojno učenje, rješavanje problema generativnim modeliranjem postalo je dostižno i relativno pouzdano. 2018. godine pojavljuju se konkretni modeli sa značajnim rezultatima u području generiranja programskog kôda, a u periodu od 2018. do 2023. godine obilježen je rast u korištenju takvih sustava i u industrijskom kontekstu (Slika 3.1.) [24]. 2025. godine dolazi do revolucije u području zbog pojavljivanja:

- **multimodalnih modela** koji su sposobni učiti iz podataka raznih oblika (npr. teksta, slika, stablastih struktura, itd.)
- **tehnika generiranja proširenog pretraživanjem** (*eng. Retrieval-Augmented Generation, kr. RAG*) – tehnika kod koje se koriste vanjski izvori podataka kao nadomjestak podacima korištenim tijekom uvježbavanja i/ili uglađivanja modela
- **(više)agentskih modela** koji su suradničkim djelovanjem sposobni generirati podatke, ali i izvršavati zadaće poput proširenog pretraživanja, planiranja ili obrade podataka različitih oblika



Slika 3.1. Kronološki prikaz razvoja sustava za automatsko generiranje programskog koda, izvedeno iz [24], [25]

3.1. Okvir generativnog modeliranja

Motivacija i ciljevi stvaranja generativnog modela mogu se prikazati sljedećim okvirom [4]:

- postoji podatkovni skup opservacija X
- pretpostavlja se kako je podatkovni skup generiran u skladu s nepoznatom distribucijom $\boldsymbol{p_{podaci}}$
- potrebno je izgraditi generativni model $\boldsymbol{p_{model}}$ koji oponaša distribuciju $\boldsymbol{p_{podaci}}$. Postizanjem ovog cilja, podaci generirani modelom $\boldsymbol{p_{model}}$ izgledat će kao da su dobiveni iz distribucije $\boldsymbol{p_{podaci}}$
- Poželjna svojstva modela $\boldsymbol{p_{model}}$ su:
  - **Preciznost** - indikativna vrijednost koja je visoka za podatke koji izgledaju kao da su dobiveni iz p_{podaci}, u suprotnom niska
  - **Generativnost** - lakoća dobivanja nove opservacije
  - **Reprezentativnost** - mogućnost shvaćanja kako model predstavlja različite višedimenzionalne značajke

3.1.1. Učenje reprezentacije

U literaturi se često nailazi na pojam reprezentacije i reprezentativnih uzoraka. U stvarnom svijetu, većina oblikovanih opservacija su elementi višedimenzionalnog prostora. No, zbog vremenskih i prostornih ograničenja računala, potrebno je pronaći minimalan skup podataka koji može pouzdano prikazati višedimenzionalnu opservaciju. Prostor u kojem žive nisko-dimenzionalne reprezentacije naziva se latentnim prostorom [4].

U našem slučaju, programski kôd je opservacija koja dolazi iz višedimenzionalnog prostora jer se kôd sastoji od podataka vođenih sintaktičkim i semantičkim pravilima, kao i stilskim pravilima pisanja urednog kôda, pravilima programske paradigme, itd. Jedan od ciljeva našeg istraživanja je predstaviti programski kôd i korisničke zahtjeve nisko-dimenzionalnom strukturom podataka poput vektora, kako bismo mogli učinkovito uvježbavati naš AI sustav.

3.1.2. Ključni vjerojatnosni pojmovi

Razumijevanje teorijske pozadine generativnog dubokog učenja zahtjeva poznavanje sljedećih pojmova iz teorije vjerojatnosti.

Prostor uzorkovanja je potpuni skup svih vrijednosti koje neka opservacija x može poprimiti [4].

Funkcija gustoće vjerojatnosti je funkcija $p(x)$ koja preslikava programski kôd $\boldsymbol{x}$ iz prostora uzorkovanja u broj iz intervala [0, 1]. Integral funkcije gustoće nad svim točkama prostora uzorkovanja mora iznositi 1.

Parametrijsko modeliranje je tehnika za strukturiranje pristupa s ciljem pronalaska prikladnog modela $p_{model}(x)$. Parametrijski model je obitelj funkcija gustoće vjerojatnosti $p_{\theta}(x)$ predstavljena konačnim brojem parametara θ.

Funkcija izglednosti $L(\theta|x)$ može se izraziti relacijom 3.1. [4]:

$$\boldsymbol{L(\theta|x) = p_{\theta}(x)} \quad (\mathbf{3.1.})$$

gdje je:

θ	skup parametara
x	opservacija
$P_{\theta}(x)$	funkcija gustoće vjerojatnosti skupa parametara θ za opservaciju x

Drugim riječima, funkcija izglednosti skupa parametara θ nekog promatranog programskog koda x definira se kao vrijednost funkcije gustoće vjerojatnosti parametriziranu s θ za taj programski kod. Za čitavi podatkovni skup nezavisnih opservacija $\boldsymbol{X}$ vrijedi [4]:

$$L(\theta|x) = \prod_{x \in X} p_{\theta}(x) \quad (3.2.)$$

Računanje umnoška kod velikog broja opservacija može biti vremenski zahtjevno. Zbog velikog broja opservacija x čije se vrijednosti nalaze unutar segmenta [0, 1], iz praktičnih razloga koristi se tzv. *log-izglednost l,* a njenu vrijednost računamo pomoću formule 3.3. [4]

$$l(\theta|X) = \sum_{x \in X} log p_{\theta}(x) \quad (3.3.)$$

Procjena maksimuma izglednosti vrši se pronalaskom skupa parametara θ funkcije gustoće $p_{\theta}(x)$ koja je najizgledniji kandidat za objašnjavanje skupa opservacija X. Formalno se izražava preko formule 3.4. [24]

$$\hat{\theta} = arg_x max l(\theta|X) \quad (3.4.)$$

$\hat{\theta}$ se također naziva i procijenjenom vrijednošću maksimuma izglednosti (eng. *maximum likelihood estimate, kr. MLE*). Neuronske mreže rade na principu minimiziranja funkcije gubitka, stoga se također može reći kako minimiziraju negativnu log-izglednost. To formalno zapisujemo sljedećom relacijom 3.5. [25]

$$\hat{\theta} = arg_{\theta}min(-l(\theta|X)) = arg_{\theta}\min(-logp_{\theta}(X)) \quad (3.5.)$$

3.2. Klasifikacija modela strojnog učenja za generiranje programskog kôda

Problem generiranja programskog kôda veoma je složen. Zbog višeobličja specifikacija i korisničkih zahtjeva (ulaznih podataka), na temelju rezultata prethodnih pregleda literature, ustanovili smo kako se za generiranje programskog kôda koristi više različitih klasa modela strojnog učenja.

U pregledu literature [26] navode se sljedeće klase:

- rekurentne neuronske mreže (*eng. Recurrent Neural Networks, kr. RNNs*)
- transformeri
- konvolucijske neuronske mreže (*eng. Convolutional Neural Networks, kr. CNNs*)
- druge vrste neuronskih mreža
- ostale vrste modela

Područjem automatskog generiranja programskog kôda vođenog strojnim učenjem dominiraju transformeri zbog prirode problematike (radimo s programskim kôdom, specifikacijama te korisničkim zahtjevima koji dolaze u obliku teksta), ali i RNN mreže, specifično modeli koji koriste dugu kratkoročnu memoriju *(eng. Long Short Term Memory, kr. LSTM*) (Tablica 3.1.).

Tablica 3.1. Vrste modela strojnog učenja za automatsko generiranje programskog kôda

Klasa modela strojnog učenja	**Podklase**	**Broj pronađenih istraživanja**
RNN	Standardni RNN-ovi	2
	LSTM varijante	12
Transformeri	Transformativni modeli	16
CNN	CNN-ovi vezani uz slike	1
Druge vrste neuronskih mreža	Generativni modeli	2
	Ostale vrste neuronskih mreža	4
Ostale vrste modela	Tradicionalne tehnike strojnog učenja Simboličko učenje	1

3.3. Drugi biologijom nadahnuti modeli

Uspjeh aktualnih generativnih AI sustava leži u činjenici da su, za razliku od tradicionalnih sustava umjetne inteligencije, fokusirani na temeljne aspekte biološke inteligencije. Kod neuronskih mreža govorimo o njihovoj sposobnosti učenja prepoznavanja i generiranja teško predvidljivih opservacija. To ih čini otpornima na nepoznanice koje dolaze iz svijeta koji se cijelo vrijeme mijenja, što dolazi do izražaja i u procesu razvoja programske podrške [27].

Postojanje različitih obitelji biologijom nadahnutih algoritama rezultat je potrebe za imitacijom prirodnih fenomena koji spadaju u tzv. biološku inteligenciju, a neki od njih su:

- fizičko utjelovljenje
- bihevioralna autonomija
- samoizlječenje
- društvene interakcije
- evolucija
- učenje

Osim neuronskih mreža, postoje i sljedeći utjecajni biologijom nadahnuti pristupi za sintezu programa:

- genetski tj. evolucijski algoritmi – algoritmi kojima se oponaša prirodni evolucijski proces
- algoritmi jata - algoritmi kojima se oponašaju društvene interakcije pripadnika neke biološke vrste

3.3.1. Genetsko (evolucijsko) programiranje

Genetsko programiranje je evolucijski algoritam nastao na temelju Darwinove teorije evolucije, a u kojem se putem emuliranja prirodne selekcije odabire populacija koja može pronaći tj. generirati opservaciju, rješenje zadanog problema [28].

Evolucijski proces sastoji se od sljedećih koraka [27]:

1. odabira genetskog prikaza
2. izgradnje populacije
3. oblikovanja funkcije prikladnosti
4. odabira selekcijskog operatora
5. odabira rekombinacijskog operatora
6. odabira mutacijskog operatora
7. stvaranja procedure za analizu podataka

Formalno cijeli proces zapisujemo kao uređenu n-torku [29]

$$G = \langle P, S, F, T, \mu, \chi, \rho, \phi, \tau \rangle \quad (3.6.)$$

gdje su:

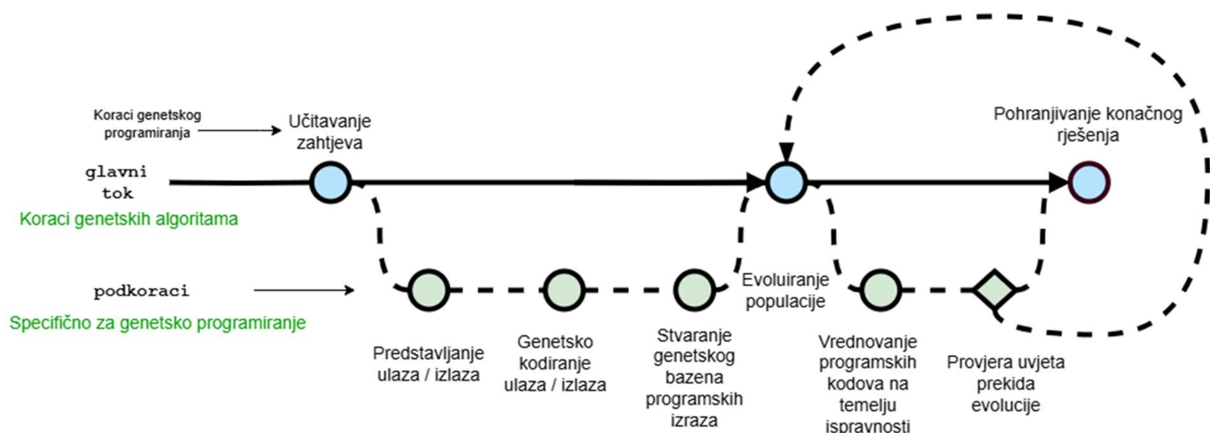
- *P:* **Prostor** pretrage programa, primjer: sva konačna stabla izgrađena na temelju skupa funkcija *F* i skupa završnih znakova *T*, koji su po utjecajem sintaktičkih ograničenja
- *S*: **Populacija** veličine *N*, učestalo je riječ o multiskupu $\{p_1, \dots, p_N\}$ gdje je $p_i \in P$
- *F*: **Skup funkcija**, konačni skup primitivnih operacija
- *T*: **Skup završnih znakova**, konačni skup varijabli i konstanti, listovi programskog stabla

- μ: **Operator mutiranja** $\mu : P \rightarrow P$ koji izvršava nasumičnu lokalnu mutaciju unutar računalnog programa
- χ: **Operator rekombiniranja** $\chi: P \times P \rightarrow P \times P$ koji razmjenjuje podstrukture između dva roditeljska računalna programa
- ρ: **Operator reproduciranja/kopiranja** koji kopira nepromijenjene individue u sljedeću generaciju, a implicitni je dio selekcijske i varijacijske faze.
- Φ: **Funkcija prikladnosti** $\phi: P \rightarrow \mathbb{R}$ ($\mathbb{R}^k$ kod višeciljnog genetskog programiranja) koja mjeri izvedbu programa s fokusom na određenu zadaću.
- τ: **Uvjet zaustavljanja**, primjer: dosegnut je maksimalni broj generacija G_{max}, dosegnuta je prikladnost iznad definiranog praga.

Genetski materijal od kojeg se sastoje pojedinci takve populacije naziva se genotipom. Fenotipom se naziva fizička manifestacija genotipa u organizmu. Genetski prikaz, u literaturi navođen i kao genetski kôd, opisuje elemente genotipa te način na koji se ti genotipovi preslikavaju u fenotipove [27].

Kod genetskog programiranja koristi se stablasti oblik genetskog prikaza [27]. Svaki pojedinac iz populacije predstavlja programski kôd. Pojedinac je prikazan kao ugniježđena lista koja se može direktno preslikati u stablo. Takvu stablastu strukturu sačinjavaju dva konačna skupa koja nazivamo *funkcijama* i *terminalima*. No, pri odabiru elemenata tih skupova, a u svrhu učinkovitog pronalaska rješenja problema, potrebno je prethodno poznavanje prostora pretrage.

Ovakav oblik programiranja vođen je gramatikom programskog jezika jer se iz gramatike jezika tvori genetski prikaz, stoga se u literaturi koristi i naziv gramatičko programiranje [30].



Slika 3.2. Primjer algoritma za genetsko programiranje, izvedeno iz [28]

Genetsko programiranje je iterativni proces (Slika 3.2.) u kojem dolazi do sinteze relativnog, no dovoljno dobrog rješenja. Veliki problem genetskog programiranja je mogućnost zaustavljanja u točki lokalnog maksimuma, što rezultira sintezom programskog kôda koji je vrednovan kao najbolje rješenje jer ostatak populacije:

- ne pruža dovoljno dobro rješenje
- ne može razmnožavanjem proizvesti bolje rješenje

Iz tog razloga, klasa ovakvih algoritama dobra je za pronalazak rješenja o čijem postojanju se tek teoretizira ili rješenja koja su složena, stoga ne postoji skup ustanovljenih pravila koja bi proces njihovog generiranja učinila kontinuiranom (neprekidnom) funkcijom ili bijekcijom gdje za svaki postojeći element iz skupa korisničkih zahtjeva postoji element iz skupa sintaktički i semantički ispravnih kôdova.

3.3.2. Programiranje jatom

Još jedan primjer biologijom nadahnutih relevantnih algoritama su algoritmi programiranja jatom – skupine računalnih agenata [27]. Računalni agenti su sustavi koji djeluju na autonoman način, percipiraju svoje okruženje, postoje duži vremenski period, prilagođavaju se promjenama u okolini, a osim toga stvaraju i slijede ciljeve [26]. Racionalni agent je agent koji pokušava ostvariti najbolji ishod, a u slučaju neizvjesnosti, najbolji očekivani ishod.

Agent se može klasificirati pomoću taksonomije inteligentnih agenata PEAS. PEAS je akronim iza kojeg stoje sljedeći pojmovi:

- izvedba *(eng.* ***p****erformance)*
- okruženje *(eng.* ***e****nvironment)*
- aktuatori *(eng.* ***a****ctuators)*
- senzori *(eng.* ***s****ensors)*

Stoga se tom taksonomijom mogu klasificirati i agenti koji djeluju u svrhu sinteze programskog kôda (Tablica 3.2.).

Tablica 3.2. Klasifikacija jata agenata za sintezu programskog kôda

Vrsta agenta	Mjera izvedbe	Okruženje	Aktuatori	Senzori
Sintetizator programskog kôda	ispravnost, učinkovitost, pouzdanost, stil programskog kôda, itd.	Korisnički zahtjevi, gramatika programskog jezika, stil programiranja, smjernice programske paradigme, itd.	Izražavanje pomoću gramatike programskog jezika, korisničko sučelje, krajnja točka aplikacijskog programskog sučelja *(eng. Application Programming Interface, kr. API)*	Virtualni senzori za konzumaciju podataka poput kanala *standardnog ulaza, izlaza, i greške*, ulazna točka za obradu vektorskog zapisa podataka, itd.

Agente se također može klasificirati u odnosu na okruženje u kojem djeluju te svojstva koja takvo djelovanje opisuju, a ta svojstva se dijele na sljedeće kategorije:

- percepcija agenta
- broj agenata, a u slučaju više agenata njihov međuodnos (kooperativan ili natjecateljski)
- određenost akcija u okruženju
- pozornost agenta
- prirodu okruženja
- stanje okruženja
- znanje agenta

Stoga se za agente za sintezu programskog kôda tvrdi da:

- djeluju u okruženju za razvoj programske podrške, a na koji utječe i prostor problema koji se rješava programiranjem
- mogu imati potpunu ili djelomičnu vidljivost, ovisno o potpunosti korisničkih zahtijeva i agentskim senzorima
- djeluju kao višeagentski sustav
- djeluju u sekvencijalnom okruženju jer svaki prethodni odabir ima utjecaj na buduće odabire
- većinom djeluju u nepoznatom okruženju te kroz proces promatranja i učenja stvaraju model okruženja u kojem djeluju

Formalno, ovakav višeagentski sustav za sintezu programskog koda izražavamo kao uređenu n-torku [31]

$$S = \langle A, E, X, U, f, g, \pi, \Phi, \tau \rangle \quad (3.7.)$$

gdje su:

- $A = \{1, \dots, N\}$: Konačni skup **agenata**
- E: Prostor stanja **okruženja**, koje može biti kontinuirano ili diskretno
- $X \subseteq \prod_{i \in A} X_i$: **spoj prostora stanja** agenata, gdje je X_i lokalno stanje agenta i (pozicija, unutrašnje varijable, lokalna memorija)
- $U \subseteq \prod_{i \in A} U_i$: **spoj prostora radnji** agenata, gdje je U_i skup radnji dostupnih agentu i
- $f: X \times E \times U \rightarrow X$: **funkcija prijelaza** koja vrši preslikavanje trenutnog jata i stanja okruženja te radnji u sljedeće stanje jata, a može biti deterministička ili stohastička
- $g_i: X_i \times E \rightarrow O_i$: **senzorna funkcija** svakog agenta, preslikava njegovo lokalno stanje i stanje okruženja u lokalno opažanje O_i
- $\pi = \{\pi_i\}_{i \in A}$: skup **lokalnih kontrolnih programa**, gdje svaki program preslikava lokalna opažanja (i unutrašnje stanje memorije, ako postoji) u akcije; *ovo je program kojeg jato piše.*
- $\Phi: X \times E \rightarrow \mathbb{R}^k$: **globalni cilj ili specifikacija**
- τ: **uvjet zaustavljanja ili vrednovanja**, može biti vremenski horizont ili trenutak kada se globalno svojstvo u Φ obistini

Kada govorimo o agentskom učenju, ono može, ali i ne mora biti vođeno algoritmima strojnog učenja. Riječ je o općenitom shvaćanju radnje učenja, gdje je cilj pronaći vrijednosti koje pružaju najbolji ishod tj. ishod koji je najbliži očekivanom.

Za okruženje djelovanja agenta može se reći da je:

- determinističko ako je problem koji se rješava deterministički, a stohastičko u slučaju kada su jasne vjerojatnosti ishoda na temelju kojeg se vrši procjenu djelovanja. Rijetko se nailazi na probleme koji su nedeterminističke prirode. Kod programiranja jatom i genetskog programiranja potrebno je predznanje o prostoru pretrage.
- statično ili dinamično, ovisno o metodologiji razvoja
- sastavljeno od diskretnih stanja iz razloga što je proces razvoja programske podrške sastavljen od diskretnih stanja

Poput genetskog programiranja, programiranje jatom spada u klasu algoritama pretrage prostora [31]. No, za razliku od pasivne populacije prisutne kod genetskog programiranja, jata su aktivna. U višeagentskim sustavima agenti mogu surađivati, natjecati se ili djelovati na koordinirani način.

4. PREGLED LITERATURE

Kako bismo usmjerili daljnja istraživanja te utvrdili njihovu opravdanost, proveli smo pregled literature. U tu svrhu postavili smo sljedeća istraživačka pitanja:

- *Kako su uspjesi u strojnom učenju, obradi prirodnog jezika i tehnikama dubokog učenja utjecali na razvoj alata za automatsko generiranje programskog kôda?*
- *Koji izazovi i prilike stoje pred nama pri integriranju tih tehnologija u proces razvoja stvarne programske podrške?*

Važno je napomenuti kako smo trebali provesti pregled literature čak tri puta jer je područje veoma aktivno. Unutar perioda od 6 mjeseci izađe više od 20 novih relevantnih radova. Daljnja istraživanja zahtijevat će dodatne etape pregleda literature kako budemo mijenjali fokus istraživanja.

4.1. Metodologija

Tijekom pregleda literature provedenog u 2024. godini koristili smo sljedeće kriterije uključivosti:

- *radovi objavljeni u posljednjih pet godina (2019. - 2024.), kako bismo obuhvatili nedavne radove koji prikazuju trenutne uspjehe u istraživačkom području*
- *radovi koji se fokusiraju na strojno učenje, duboko učenje i obradu prirodnog jezika u svrhu automatskog generiranja programskog kôda*
- *radovi koji govore o praktičnim implikacijama, izazovima te ishodima u stvarnom procesu razvoja programske podrške*

Morali smo uključiti i neke nerecenzirane radove iz razloga što se znanstveni doprinosi često prikažu prvo na otvorenim repozitorijima radova poput arXiv-a [32]. Svrha pregleda takvih radova je osiguravanje autentičnosti naših znanstvenih doprinosa, uz svjesnost o riziku nepouzdanosti, ali i nedostatku akademske strogosti.

Uz to, neke radove smo morali isključiti iz rezultata zbog toga što su:

- *teoretski radovi ili studije u početnoj fazi istraživanja pa ne nude bitne empirijske dokaze ili upotrebu alata za automatsko generiranje kôda*
- *radovi koji ne govore o primijeni tehnika umjetne inteligencije u svrhu generiranja programskog kôda*
- *radovi koji govore o generiranju kôda u drugom kontekstu (npr. generiranje QR kôdova)*
- *radovi koji nisu pisani na engleskom jeziku ili jeziku srodnom hrvatskom*

Pošto je kod provođenja pregleda literature odabir baze podataka ključan korak, kako bismo osigurali sveobuhvatnost i pouzdanost rezultata, oslonili smo se na prethodnu analizu trenutno dostupnih pretraživača (Tablica 4.1.) [33].

Tablica 4.1. Rezultati vrednovanja pretraživača znanstvenih radova [33]

Rang		**Naziv pretraživača**	**Znanstveno područje**
Kategorija	***Veličina (cca)***		
Neophodno	323,000,000	Web of Science	Multidisciplinarno
Neophodno	144,252,584	Bielefeld Academic Search Engine (BASE)	Multidisciplinarno
Neophodno	70,000,000	Scopus	Multidisciplinarno
Neophodno	15,000,000	ScienceDirect	Multidisciplinarno
Neophodno	8,000,000	Wiley Online Library	Multidisciplinarno
Neophodno	2,000,000	ACM Digital Library	Računalna znanost
Suplement	389,000,000	Google Scholar	Multidisciplinarno
Suplement	323,000,000	WorldWideScience	Multidisciplinarno
Suplement	233,127,915	AMiner	Multidisciplinarno
Suplement	213,850,455	Microsoft Academic	Multidisciplinarno
Suplement	72,366,665	Semantic Scholar	Multidisciplinarno
Suplement	12,731,539	Springer Link	Multidisciplinarno
Suplement	12,000,000	JSTOR	Multidisciplinarno
Suplement	8,401,126	CiteSeerX	Multidisciplinarno
Suplement	8,000,000	WorldCat	Multidisciplinarno
Suplement	4,831,568	IEEE Xplore	Računalna znanost, elektrotehnika, elektronika
Suplement	4,539,285	Digital Bibliography & Library Project (DBLP)	Računalna znanost
Suplement	3,902,698	Directory of Open Access Journals (DOAJ)	Multidisciplinarno
Suplement	1,518,677	arXiv	Multidisciplinarno

U ovom pregledu literature koristili smo akademske baze podataka *Web of Science*, *BASE*, *Scopus, ScienceDirect, Wiley Online Library* i *ACM Digital Library*, pošto je riječ o jako poznatim i opsežnim izvorima recenziranih članaka, konferencijskih radova i tehničkih izvješća iz područja računalne znanosti i programskog inženjerstva, što ih čini neophodnim instrumentima za pretragu [33]. Uz to smo koristili i dopunske baze podataka poput *arXiv* i *IEEE Explore* te pretraživača *Google Scholar* kako bismo pokrili i moderne radove iz područja generiranja programskog kôda. Dok je *IEEE Explore* specijalizirana baza radova fokusirana na područja računalne znanosti, elektrotehnike i elektronike, ostale korištene baze i pretraživači nemaju fokus već su multidisciplinarni po prirodi, stoga nam pružaju pristup širokom rasponu metodologija, algoritamskih pristupa te praktičnih primjena kada je riječ o generiranju kôda. Kada ih sagledamo kao cjelinu, ovi pretraživači predstavljaju pouzdan izvor aktualnih rezultata istraživanja i novonastalih trendova iz područja.

Koristili smo sljedeće pojmove pri pretrazi:

- "*code generation*"
- *"automatic programming"*

Sustavnom i strukturiranom metodologijom pregleda rezultata odabranih studija iz područja generiranja kôda osigurali smo opsežno razumijevanje predmeta istraživanja. Inicijalno smo probrali istraživačke radove na temelju prethodno navedenih kriterija uključenosti. Provođenjem kvalitativne analize utvrdili smo ključne informacije o radovima poput vrsta učestalo korištenih tehnika za generiranje kôda (npr. *pristupi vođeni modelom, pravilima ili neuronskom mrežom*), izazove te domene primjene. Zatim smo kategorizirali radove na temelju fokusa istraživanja kako bismo vrednovali njihov praktični značaj. Komparativnom analizom smo ukazali na trendove i jazove u literaturi s fokusom na učinkovitost i ograničenja različitih tehnika generiranja kôda. Naposlijetku smo povezali rezultate kako bismo istaknuli suvremene tehnike te područja koja iziskuju daljnje istraživanje. Ovakvim sustavnim pregledom literature osigurali smo postojanje jake teoretske osnove, ali i skupa praktičnih smjernica za buduća istraživanja.

Povezivanje prikupljenih informacija omogućilo nam je razumijevanje trenutne slike u istraživanjima AI tehnika za generiranje programskog kôda. Nakon što smo kategorizirali radove, dodatno smo ih klasificirali primjenom već utvrđene i prethodno iznesene taksonomije sustava za automatsko programiranje. Ovim procesom smo grupirali radove te istaknuli vezu između pristupa automatskom generiranju programskog kôda i prednosti, ali i ograničenja postojećih sustava. Prepoznavanjem ponavljajućih obrazaca, poput sve prisutnije primjene

dubokog učenja, uvažavajući pritom i kontrastne pristupe, ovakvom sintezom dobili smo uvid u postojeće jazove koje imamo priliku premostiti u budućim istraživanjima.

4.2. Rezultati

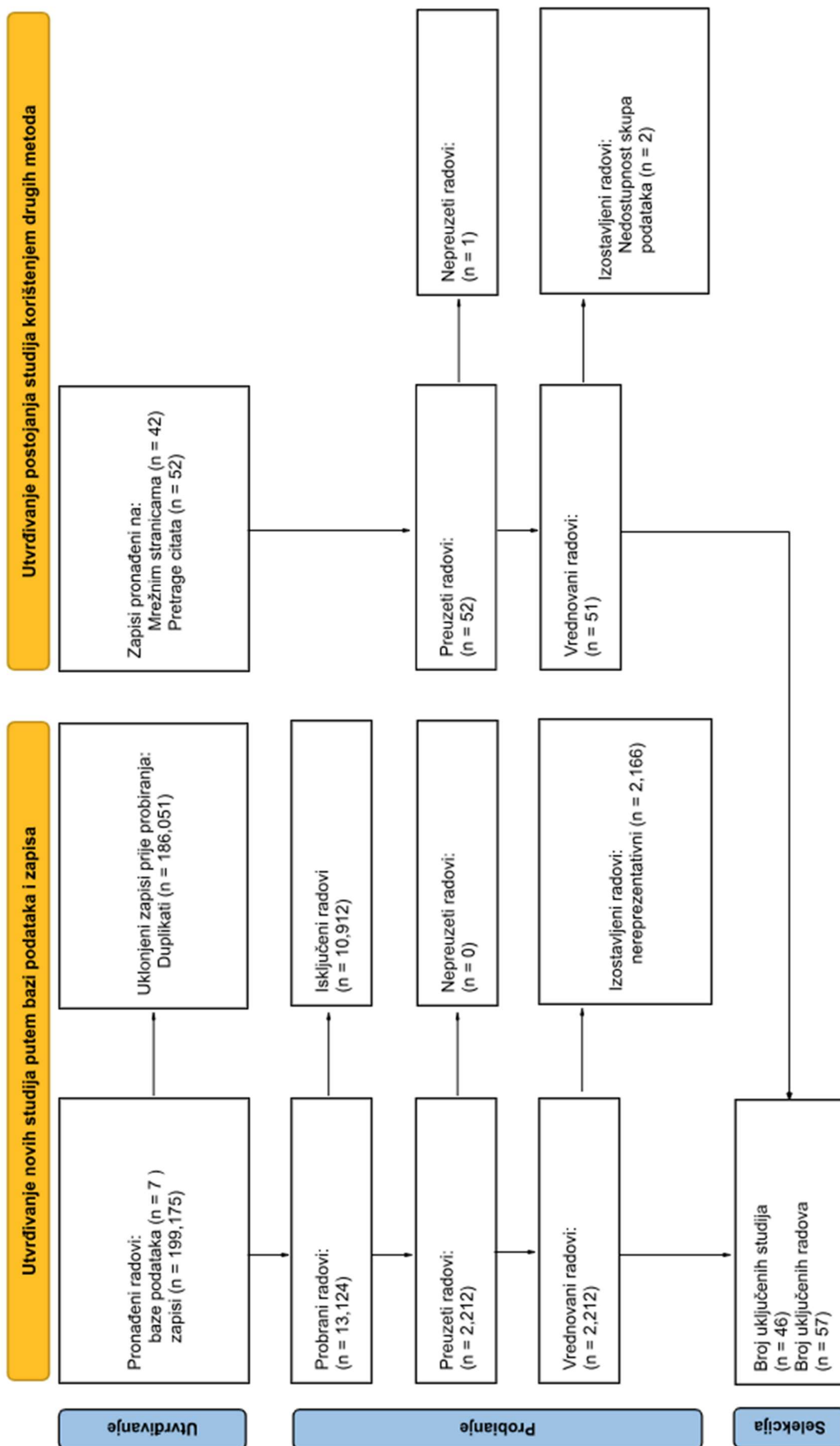
Automatskom pretragom relevantnih baza podataka došli smo do 199 175 radova. Prije svega smo uklonili radove koji su izvučeni iz bazi podataka bez pripadajućih autora. Nakon što smo uklonili i duplikate, broj radova smo sveli na 14 555 radova, no zbog razlika u formatiranju naslova, dodatnom ručnom analizom rezultata uklonili smo preostale duplikate te skup sveli na 13 088 radova.

Sljedeći korak bilo je ustanovljavanje relevantnosti, iz razloga što se pojam " *kôd*" i "*programiranje*" koriste u brojnim znanostima, ali i u različitim kontekstima stoga njihovo značenje nije jedinstveno. Eliminacijom radova koji nisu relevantni za problem automatskog generiranja programskog kôda, popis smo sveli na 2 174 rada. Količina prikupljenih radova ukazuje na popularnost problematike u istraživačkim krugovima, kao i na postojanje velike potrebe za znanstvenim i inženjerskim doprinosima u tom području.

Zatim smo pokušali utvrditi koji su podatkovni skupovi za uvježbavanje AI sustava utjecajni kada je riječ o automatskom generiranju programskog kôda. Te skupove iskoristili smo za daljnje grupiranje prikupljenih radova. Popis relevantnih skupova podataka pronašli smo na mrežnoj stranici *Papers with Code* [34] koja nudi informacije o utjecajnim rezultatima za razne zadaće iz područja strojnog učenja [34].

Na stranici *Papers with Code* pronašli smo 52 relevantna podatkovna skupa (datuma *25. 1. 2025.*), a zatim smo te podatkovne skupove poredali po broju srodnih radova, počevši sa skupom koji ima najviše srodnih radova. Neki od podatkovnih skupova predstavljeni su tek jednim sveobuhvatnim radom. Iako se nalazi na popisu skupova podataka povezanih s problematikom automatskog generiranja programskog kôda, podatkovni skup pod nazivom *EGSM (eng. Educational Grade School Math)* nije korišten u tom kontekstu ili za rješavanje tog problema, stoga je izostavljen iz pregleda literature.

Kako bismo generirali prikaz rezultata pregleda literature (Slika 4.1.), koristili smo alat *PRISMA Flow Diagram* [35] koji prati metodologiju PRISMA [36].



Slika 4.1. Metodologija pregleda literature (PRISMA)

Uz to smo proveli analizu i vrednovanje prikupljenih radova, iz čega su proizašli sljedeći podaci (Tablica 4.2.):

- naziv podatkovnog skupa pomoću kojeg se vršilo uvježbavanje, uglađivanje te vrednovanje
- naziv modela
- naziv znanstvenog rada (članka)
- godinu objave rezultata

Tablica 4.2. Popis relevantnih radova iz područja [34]

Skupovi podataka	**Najbolje rangirani model**	**Reprezentativni istraživački rad na stranici *Papers with Code*[1]**	**Godina objave[2]**
HumanEval	LLMDebugger	Llama 2: Open foundation and fine-tuned chat models	2023.
MBPP	QualityFlow (Sonnet-3.5)	Llama: Open and efficient foundation language models	2023.
WikiSQL	NL2SQL-RULE	FREB-TQA: A Fine-Grained Robustness Evaluation Benchmark for Table Question Answering	2024.
CodeXGLUE	Redcoder-ext	Starcoder: may the source be with you	2023.
APPS	LPW (GPT-4o)	Code llama: Open foundation models for code	2023.
CoNaLa	PanGu-Coder-FT-I	Ernie-code: Beyond English-centric cross-lingual pretraining for programming languages	2022.
CodeContests	LPW (GPT-4o)	Wizardcoder: Empowering code large language models with evol-instruct	2023.
DS-1000		DeepSeek-Coder: When the Large Language Model Meets Programming--The Rise of Code Intelligence	2024.
CONCODE	Redcoder-ext	CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation	2021.

[1] Reprezentativni rad nije nužno i rad koji predstavlja model s trenutno najvećom postignutom preciznošću već rad koji je zbog visoke citiranosti rangiran kao najpopularniji unutar znanstvene zajednice
[2] Godina objave odnosi se na reprezentativni istraživački rad, a ne na model

Skupovi podataka	Najbolje rangirani model	Reprezentativni istraživački rad na stranici *Papers with Code*[1]	Godina objave[2]
ClassEval		Selfcodealign: Self-alignment for code generation	2024.
Django	MarianGG	Can we generate shellcodes via natural language? An empirical study	2022.
Hearthstone		Unifying the perspectives of NLP and software engineering: A survey on language models for code	2023.
JulCe		A survey of neural code intelligence: Paradigms, advances and beyond	2024.
SecurityEval		When LLMs meet cybersecurity: A systematic literature review	2024.
BigCodeBench	GPT-4o-2024-05-13	TULU 3: Pushing Frontiers in Open Language Model Post-Training	2024.
HumanEval-X		CodeGeeX: A Pre-Trained Model for Code Generation with Multilingual Benchmarking on HumanEval-X	2023.
MCoNaLa		Prompt2model: Generating deployable models from natural language instructions	2023.
LLaMEA		Toward Automated Algorithm Design: A Survey and Practical Guide to Meta-Black-Box-Optimization	2024.
Data Science Problems		DS-1000: a natural and reliable benchmark for data science code generation	2023.
Lyra		CloudEval-YAML: A Practical Benchmark for Cloud Configuration Generation	2024.
Shellcode_IA32	CodeBERT	Security of Language Models for Code: A Systematic Literature Review	2024.
PyTorrent		Pytorrent: A Python library corpus for large-scale language models	2021.
CoNaLa-Ext	BART W/Mined	Incorporating external knowledge through pre-training for natural language to code generation	2020.
CriticBench		CriticBench: Benchmarking LLMs for Critique-Correct Reasoning	2024.

Skupovi podataka	**Najbolje rangirani model**	**Reprezentativni istraživački rad na stranici *Papers with Code*[1]**	**Godina objave[2]**
BioCoder		Openhands: An open platform for AI software developers as generalist agents	2024.
DSEval-LeetCode		Mlcopilot: Unleashing the power of large language models in solving machine learning tasks	2023.
SAFIM		Qwen2.5-coder technical report	2024.
Spectre-v1		Fastspec: Scalable generation and detection of spectre gadgets using neural embeddings	2021.
WebApp1K-React	o1-preview	Insights from Benchmarking Frontier Language Models on Web App Code Generation	2024.
EVIL EVIL-Decoders EVIL-Encoders		EVIL: Exploiting Software via Natural Language	2021.
HumanEval-XL		HumanEval-XL: A Multilingual Code Generation Benchmark for Cross-lingual Natural Language Generalization	2024.
SLTrans		Ircoder: Intermediate representations make language models robust multilingual code generators	2024.
CodeGen4Libs		CodeGen4Libs: A Two-Stage Approach for Library-Oriented Code Generation	2023.
DISL		DISL: Fueling Research with A Large Dataset of Solidity Smart Contracts	2024.
DSEval-Exercise DSEval-Kaggle DSEval-SO		Benchmarking Data Science Agents	2024.
Flat Real World Simulink Models		SLGPT: Using transfer learning to directly generate Simulink model files and find bugs in the Simulink toolchain	2021.

Skupovi podataka	**Najbolje rangirani model**	**Reprezentativni istraživački rad na stranici *Papers with Code*[1]**	**Godina objave[2]**
MMCode		MMCode: Benchmarking Multimodal Large Language Models for Code Generation with Visually Rich Programming Problems	2024.
OpenAPI completion refined		Optimizing Large Language Models for OpenAPI Code Completion	2024.
PECC	Claude 3 Haiku	Pecc: Problem extraction and coding challenges	2024.
RES-Q	QurrentOS-coder + Claude 3.5 Sonnet	RES-Q: Evaluating Code-Editing Large Language Model Systems at the Repository Scale	2024.
RMCBench		RMCBench: Benchmarking Large Language Models' Resistance to Malicious Code	2024.
SOEval		Franc: A Lightweight Framework for High-Quality Code Generation	2023.
TACO-Code	GPT-4	Taco: Topics in algorithmic code generation dataset	2023.
TFix's Code Patches Data		Tfix: Learning to fix coding errors with a text-to-text transformer	2021.
Turbulence	GPT-4	Turbulence: Systematically and automatically testing instruction-tuned large language models for code	2023.
Verified Smart Contract Code Comments Verified Smart Contracts	GTP-J 6B Smart Contract	Efficient avoidance of vulnerabilities in auto-completed smart contract code using vulnerability-constrained decoding	2023.
WebApp1K-Duo-React	claude-3-5-sonnet	A Case Study of Web App Coding with OpenAI Reasoning Models	2024.
Livecodebench	LPW (GPT-4o)	Planning-Driven Programming: A Large Language Model Programming Workflow	2024.
Android Repos	Entity Type Model	Building Language Models for Text with Named Entities	2018.

Iz pregleda smo izbacili rezultat navodno ostvaren modelom Jupyter-AI zbog toga što se ne navodi rad iz kojeg proizlazi. Rezultat je objavljen na mrežnoj stranici *Papers with Code* pod mjerom *Generiranje programskog kôda pomoću podatkovnog skupa DSEval-LeetCode* [37].

4.2.1. Vrednovanje uspješnosti modela

Preciznost modela jedna je od najkorištenijih mjera za vrednovanje uspješnosti AI modela, no usporedba grubih podataka nije veoma informativna iz razloga što ne potiče dublje razumijevanje računalne (umjetne) inteligencije [37]. Preciznost modela izražava se u postotcima (

Tablica 4.3.), a odnosi se na uspješnost modela vrednovanog korištenjem podatkovnog skupa za vrednovanje koji se sastoji od ulaznih podataka koji se ne nalaze u podatkovnom skupu za uvježbavanje te očekivanog rezultata.

Tablica 4.3. Rezultati preciznosti modela

Naziv modela	**Naziv podatkovnog skupa**	**Ostvarena preciznost**
LLMDebugger	HumanEval	99.4% [38]
QualityFlow (Sonnet-3.5)	MBPP	94.2% [39]
NL2SQL-RULE	WikiSQL	89.2% [40]
MarianGG	Django	81.83% [41]
PanGu-Coder-FT-I	CoNaLa	44.32% [42]

Neki od modela vrednovani su korištenjem jediničnih testova. Tako je za model LPW (GPT-4o) [43], uvježbavanog nad podatkovnim skupom APPS, zabilježena prolaznost jediničnih testova od čak 87.2%. No, broj radova koji koriste jedinične testove značajno je manji u odnosu na radove koji mjere isključivo preciznost. U daljnjim istraživanjima morat ćemo staviti fokus i na ovakvu vrstu vrednovanja te napraviti komparativnu analizu kako bismo utvrdili u kojoj mjeri prolaznost testova utječe na preciznost modela, kao i na druga svojstva modela pomoću kojih možemo utvrditi njegovu uspješnost u generiranju programskog kôda.

Postojanje velikog broja raznolikih pristupa, čak i nakon sužavanja područja interesa na AI sustave iz domene dubokog učenja, govori o tome kako ne postoji skup zlatnih pravila kada je riječ o automatskom generiranju programskog kôda pomoću AI sustava. Analizom

prikupljenih radova uvidjeli smo velik broj jazova koji nastaju pod utjecajem primjene velikog broja različitih pristupa (Slika 4.2.).

Slika 4.2. Mentalna mapa znanstvenih radova prikupljenih tijekom pregleda literature

Prethodna vizualizacija (Slika 4.2.) prikazuje postojanje nekoliko struja istraživanja u području primjene generativnih AI tehnika za automatsko generiranje programskog kôda, a sve zbog:

- Složenosti problema automatskog generiranja programskog kôda
- Složenosti procesa programiranja tj. razvoja programske podrške

U središtu vizualizacije nalazi se autor Niklas Muennighoff sa sveučilišta Stanford čiji su radovi imali značajan utjecaj na istraživanja u području u proteklih pet godina. Rezultat njegovih istraživanja su brojne mjere, podatkovni skupovi te veliki jezični modeli koji su i danas relevantni kada govorimo o automatskog generiranju kôda.

Tablica 4.4. Tumač vizualizacije (Slika 4.2.)

Istraživački fokus	**Klasifikacija**	**Boja u vizualizaciji**
Izvlačenje znanja iz podataka	Upravljanje znanjem	Crvena
Specifična domena	Sustavi unutar uske domene	Plavozelena
Hibridni pristupi strojnom učenju	Upravljanje učenjem	Ljubičasta
Oblikovanje sustava za strojno učenje	Upravljanje učenjem	Zelena
Podatkovni skupovi	Upravljanje podacima	Žuta
Različite tehnologije i pristupi kodiranju	Sustavi uzlaznog pristupa	Plava
Čovjek u petlji	Asistentski sustavi	Narančasta

Analizom vizualizacije potvrđujemo kako su uočeni obrasci tijekom pregleda literature doista postojeći jer ih je korištenjem automatizirane tehnike grupiranja uspio oblikovati i prikazati sustav *VosViewer* [44]. Time smo opravdali oslanjanje na dvije postojeće klasifikacije sustava (taksonomije sustava za automatsko programiranje i taksonomije sustava unutar računalnih arhitektura za e-učenje) koje nadalje koristimo u radu te planiramo koristiti u budućim radovima.

Utjecaj pristranosti minimizirali smo prikupljanjem znanstvenih radova automatiziranjem pretrage pomoću više različitih pretraživača, a filtriranje reprezentativnih radova temeljilo se na postojećim rezultatima vrednovanja modela, podatkovnih skupova te mjera prikazanih na stranici *Papers with Code* [44].

4.2.2. Klasifikacija radova

Slijedi prikaz klasifikacija pronađenih radova temeljene na postojećoj taksonomiji sustava za automatsko programiranje. Klase sustava za automatsko programiranje povezali smo s klasama korištenih AI modela, a uz to smo prikazali i ključne probleme u svakoj kategoriji (Tablica 4.5.).

Tablica 4.5. Klasifikacija pristupa automatskog generiranju programskog kôda

Kategorija	Problemi	Pristup
Sustavi uzlaznog pristupa	podržanost isključivo tekstualnih tokena [45] ograničenost konteksta na repozitorij kôda [45] nedostatak holističkog i pouzdanog vrednovanja [45] kritičko promišljanje [43], [46] inženjerstvo sufliranja [47], [48] podatkovna skalabilnost [49] višejezičnost [50] ekonomičnost upravljanja podacima i znanjem [51] sigurnost generiranog kôda [52] toksičnost / pristranost modela [52], [53] razumijevanje prirodnog jezika [54] zatvorenost sustava [55]	LLM [43], [45], [46], [47], [48], [49], [50], [51], [52], [53], [54], [55]
Sustavi unutar uske domene	pristranost pozitivnom ishodu (pristranost automatizaciji) [56] neovlašteni pristup i rad s podacima [56] zahtijevanje precizno specificirane zadaće [56], [57] ekonomičnost [56] nedostatna reprezentativnost podatkovnog skupa [58] nedostatna reprezentativnost postojećih mjera [59] inženjerstvo sufliranja [60]	LLM [56], [57], [58], [59], [60]

Kategorija	Problemi	Pristup
Asistentski sustavi	podatkovna skalabilnost [61] negativni utjecaj šumova u specifikaciji [62] inženjerstvo sufliranja [63], [64], [65] razumijevanje prirodnog jezika [66] razumijevanje netekstualnih tipova ulaznih podataka [67] širina domenskog znanja [68], [69] složenost problema [68], [70], [71] učenje jednog programskog jezika ne pomaže u učenju drugog jezika[3] [72] pristranost pozitivnom ishodu [73] nespoznavanje grešaka na temelju iskustva [73] kvaliteta kôda [74] sigurnost generiranog kôda [75], [76], [77], [78], [79] višejezičnost [80], [81], [82] poznavanje novih tehnologija [82], [83], [84], [85] trivijalnost rješenja [83] jedinični testovi postaju usko grlo procesa [39], [79] održavanje kvalitete podatkovnog skupa [79], [86] ograničenost konteksta [85], [87] mala količina dostupnih podataka za uvježbavanje [88]	LLM [38], [63], [64], [65], [66], [67], [68], [73], [74], [75], [76], [77], [80], [83], [84], [85], [86], [87] LLM integriran s IDE-om [72] LLM vođen jediničnim testovima [61], [70], [79] Višeagentski sustav temeljeni na LLM-u vođenim jediničnim testovima [61] neuralni strojni prevoditelj potpomognut LSTM [62] GAN [69] kodni LM [82] transferno učenje [88] evolucijsko programiranje [71] podržano učenje *(eng. Reinforced learning)* [81]

[3] Stoga tvrdimo kako je riječ o sustavima koji upravljaju sadržajima, a ne o sustavima koji uče.

5. ANALIZA POSTOJEĆIH SUSTAVA ZA AUTOMATSKO GENERIRANJE KÔDA

Suvremeni LLM-ovi temeljeni na transformacijama [89], a prethodno korišteni u svrhu generiranja teksta [90], uspješno su upotrijebljeni u svrhu rješavanja jednostavnih programskih problema u Pythonu [9], [91]. Na temelju tih uspjeha, došlo se do spoznaje kako bi prirodni jezik mogao postati sljedeća programska paradigma [11]. Postoji, pak, zabrinutost kako će ovom razinom automatizacije procesa razvoja programske podrške biti istisnuta mnoga zanimanja. No, riječ je o mitu.

Postavljanjem prirodnog jezika kao programskog jezika i stvaranjem nove metodologije, krajnji korisnici postali bi programeri [89] te bi na temelju svojih potreba mogli dobiti programski kôd, preciznije: programsku podršku. Programski kôd bio bi upakiran u crnu kutiju. No, tek kroz iterativni proces mogli bi dobiti program koji zadovoljava njihove potrebe. S druge strane, smanjivanjem složenosti procesa programiranja kao i izgradnje programske podrške povećali bismo apetite krajnjih korisnika, a time doveli do potrebe za izradom još većih i složenijih sustava programske podrške.

5.1. Taksonomija sustava vođenih prirodnim jezikom

Pomoću oznaka za oblik ulaza, programa i izlaza, oblikovala se taksonomija kojom se omogućila kategorizacija pristupa automatskom generiranju kôda vođenog prirodnim jezikom. Oznake pisane velikim slovima označavaju korištenje prirodnog jezika, dok oznake malim slovima označavaju korištenje neprirodnog jezika npr. u izvornom kôdu [90]. Detalji taksonomije nalaze se u nastavku.

5.1.1. Oblik programa

Oblik programa sastoji se od oznaka na lijevoj i desnoj strani taksonomske strukture. Na lijevoj strani nalazi se oznaka opisa u prirodnom jeziku $P_{\{L, A\}}$, dok se s desne strane nalaze oznake izlaznog generiranog kôda, a to su *{c, s, m}*.

Slijede opisi svake oznake:

- P_L - opis koda (liniju po liniju)
    - *primjer:* “učitaj cijeli broj *i* te inkrementiraj *i* dok ne dosegne vrijednost 100. Tijekom inkrementacije, pribroji *i* cijelom broju *suma*. Ispiši cijeli broj *suma* na ekran.”
- P_A - opis kôda (apstraktni)
    - *primjer:* “program koji zbraja cijele brojeve od 0 do 100”

- *c* - stvarni programski kôd - kôd koji je potpun te se može izvršiti
- *s* - isječak kôda - dio koji se ne može direktno izvršiti, ali sadrži dio logike većeg programa
- *m* - posrednički kôd - prikaz znanja koji se nalazi na pola puta između prirodnog i programskog jezika, kako bi se povećala čitljivost kôda
    - *primjer*: regularni izrazi, apstraktna sintaktička stabla, CIL

5.1.2. Oblik ulaza

Oblik ulaza nije obvezni dio strukture, no ako ga se odluči prikazati, nalazi se na lijevoj strani izraza i to u obliku indeksa od *P*, npr. P_i.
Dostupni su sljedeći indeksi:

- *i* - stvarni ulaz koji se koristi pri radu programa, npr. datoteke u formatu *MS Excel*
- *t* - testni slučaj - ulaz kojem je pridijeljen i očekivani izlaz, a služi isključivo u testne svrhe

5.1.3. Oblik izlaza

Postoji samo jedan tip izlaznog zapisa, a to je *o* te ga koristimo kao indeks oznake *c* na desnoj strani taksonomskog izraza, npr. $c_{\{o\}}$. Ovaj zapis označava kako je program za generiranje kôda proizveo izlazni podatak.

5.2. Klase pristupa automatskom generiranju programskog kôda vođene prirodnim jezikom

Služeći se prethodno opisanom taksonomijom elemenata sustava za automatsko generiranje programskog kôda možemo klasificirati postojeće pristupe automatskom generiranju programskog kôda (Tablica 5.1.) [91].

Tablica 5.1. Klasifikacija sustava za automatsko generiranje programskog kôda vođenih prirodnim jezikom

Klasa	Ulaz	Izlaz	Primjer	Ograničenja
$P_{L\{i\}} \Rightarrow c_{\{o\}}$	opis liniju po liniju stvarne vrijednosti ulaznih podataka	kôd koji se može izvršiti, specifičan za domenu	izvorni kôd specifičan za jednu domenu npr. funkcije u Excelu	sposobnost sustava za shvaćanje apstrakcija napisanih u prirodnom jeziku
$P_L \Rightarrow c$	opisi liniju po liniju stvarne vrijednosti ulaznih podataka **nisu nužne za rad**	kôd u programskom jeziku opće namjene koji se može izvršiti	generiranje kôda opće namjene pomoću semantičkog parsiranja i izvlačenja informacija	ograničenje uputa pisanih prirodnim jezikom zbog ograničenja tehnologije poput parsera
$P_A \Rightarrow s$	opis programa koji je apstraktniji i prirodniji	isječak kôda	generiranje izraza u programskom jeziku opće namjene	zbog utjecaja funkcije gubitka i grešaka kod strojnog učenja, algoritmi imaju poteškoća s generiranjem složenijeg kôda
$P_A \Rightarrow m$	apstraktni opis programa pisan prirodnim jezikom	posrednički kôd koji zatim koristi paralelni proces kako bi obradom došli do izvršnog kôda	dobivanje kostura programa ili programskog kôda pisanog u strojnom jeziku koji je potrebno kompajlirati	potrebno je dodatno obraditi izlazni kôd kako bismo dobili izvršni kôd
$P_{A\{t\}} \Rightarrow c$	apstraktni opis pisan prirodnim jezikom skup jediničnih testova (primjeri ulaznih i izlaznih podataka)	kôd u programskom jeziku opće namjene koji se može izvršiti	programerska natjecanja	zahtijevaju jedinične testove kako bi provjerili ispravnost generiranog programskog kôda

5.3. Slučajevi korištenja – potrebe zajednice razvojnih programera

Mrežne stranice poput *Stack Overflow*-a, oblikovane u svrhu pohranjivanja odgovora na postavljena pitanja, pružaju nam uvid u trendove unutar zajednice razvojnih programera [92].

Prikupljanjem parova (pitanje, odgovor) sa stranice *Stack Overflow* te kategorizacijom istih predstavljamo širinu područja automatskog generiranja programskog kôda (Tablica 5.2.). U obzir smo uzeli odgovor koji je zajednica procijenila kao najbolji te mu dodijelila najviše pozitivnih glasova. Broj glasova smo time koristili kao indikator točnosti danog odgovora.

Tablica 5.2. Rezultati klasifikacije uzoraka sa stranice Stack Overflow

Klasa pitanja	***Pitanje***	***Slučaj korištenja***
Otklanjanje kompilacijskih grešaka	*Kako pretvoriti decimalni broj u realni broj dvostruke preciznosti u programskom jeziku C#?* [93]	Pružanje automatiziranih povratnih informacija za programere početnike [94]
Osiguravanje kompatibilnosti s različitim internetskim preglednicima	*Zašto se širina podelementa deklarirana u postotcima svela na 0 unutar apsolutno pozicioniranog nadelementa na Internet Exploreru 7?* [95]	Brži razvoj kvalitetnijih korisničkih sučelja [96]
Algoritamsko rješavanje problema	*Kako izračunati nečiju dob na temelju rođendana predstavljenog tipom podataka DateTime?* [97]	Dekompozicija problema korištenjem LLM-ova za unaprjeđenje genetskih algoritama [98]
Lokalizacija	*Kako izračunati nečiju dob na temelju rođendana predstavljenog tipom podataka DateTime?* [97]	Pregledom literature nismo pronašli radove koji se bave problematikom generiranja algoritama koji su lokalizirani.
Korištenje API-ja za rješavanje problema	*Računanje relativnog vremena u C#-u* [99] *Punjenje podatkovnih struktura DataSet ili DataTable iz rezultantnog podatkovnog skupa LINQ upita* [100] *Bacanje greške kako bismo spriječili ažuriranje tablice kroz MySQL okidač* [101] *Kako osloboditi memorijski prostor koji zauzima podatak tipa ByteArray u programskom jeziku ActionScript?* [102]	Sugeriranje korištenja API-ja [103]

Klasa pitanja	*Pitanje*	*Slučaj korištenja*
Pozivanje mrežnih API-ja za dohvaćanje podataka	*Određivanje vremenske zone korisnika* [104]	Sugeriranje korištenja mrežnih API-ja [105]
Objašnjenje rada programskog kôda	*Razlika između Math.Floor() i Math.Truncate()* [106] *Kako se koristi C-ov API za spojne točke u C++ programskom jeziku na z/OS operacijskom sustavu?* [107]	Generiranje dokumentacije o kôdu [108]
Korištenje ispravnog tipa podataka	*Binarni podaci u MySQL-u* [109]	Obrazovanje o bazama podataka [110]
Optimizacija	*Koji je najbrži način za doći do vrijednosti* $\boldsymbol{\pi}$*?* [111]	Optimizacija brzine izvršavanja [44]

5.3.1. Utjecaj sustava umjetne inteligencije u različitim domenama

AI sustavi za automatsko generiranje programskog kôda pronalaze svoju primjenu u mnogim domenama [45], kako znanstvenim [56], [59], [112], tako i inženjerskim [38], [68], [88], [113]. Obrazovanje je jedna od takvih domena, a u ovom radu fokus stavljamo na obrazovanje programera početnika. Korištenjem sustava TEGCER, pokazalo se kako programeri početnici u prosjeku 25% brže rješavaju greške u odnosu na programere početnike kojima je pomagao čovjek-tutor [94]. Uzmemo li u obzir potrebe obrazovnih sustava, bili oni dio formalnih sustava obrazovanja ili sustava za razvoj programske podrške koji su fokus našeg istraživanja, kao i probleme uzrokovane nepovoljnim omjerom programera početnika naspram programera eksperata [95], ovakvim pozitivnim rezultatima motivirani smo za daljnje istraživanje za korištenje sustava u ulozi asistenata pri učenju programiranja, novih tehnologija i pristupa programiranju. Razvoj programske podrške izazovan je posao. Primjer nalazimo u slučaju oblikovanja i razvoja korisničkih sučelja. Pregledom literature uočili smo kako su zabilježeni brojni benefiti korištenja sustava za generiranje kôda pri razvoju korisničkih sučelja. Primjeri benefita su [96]:

- Povećana brzina razvoja programske podrške
- Povećana kvaliteta i konzistentnost programskog kôda
- Smanjen utjecaj ljudske greške
- Jednostavnije održavanje i ažuriranje

No, problem osiguravanja kompatibilnosti korisničkog sučelja s aktualnim internetskim preglednicima i dalje je otvoren, te se u literaturi tek ukazuje na njegovo postojanje. Možemo pretpostaviti kako na otvorenost tog problema utječe dinamična priroda internetskih preglednika, ali i činjenica kako u literaturi nedostaju podatkovni skupovi fokusirani na taj problem ili skupovi čiji elementi predstavljaju rješenja tog problema.

Na sličan problem deficita podataka nailazimo i kod problema predlaganja API-ja u svrhu rješavanja konkretnog problema. LLM-ovi mogu vršiti dekompoziciju problema, no za sugeriranje API-ja koji bi riješio problem potrebno je na raspolaganju imati dostatnu količinu podataka kako bismo fino podesili model [103]. Osim toga se kao problem navodi i različita razina granulacije između korisničkih zahtjeva i API-ja pogotovo u slučajevima kada za jedan problem trebamo koristiti više od jednog API-ja. U tom slučaju model može proizvesti lošije rezultate [103].

Još jedan primjer otvorenog problema je lokalizacija, tj. potreba prilagođavanja rada programske podrške drugim tržištima tj. kulturama [114]. Često se pojam lokalizacije odnosi na prevođenje teksta i simbola kojima su izloženi korisnici, no ponekad kulturološki čimbenici utječu i na funkcionalne aspekte aplikacije. Primjer nalazimo u algoritmu za računanje dobi jer se računanje dobi razlikuje među nekim zemljama. Prema našoj trenutnoj spoznaji, ne postoji istraživanje koje se bavi problematikom lokalizacije pri automatskom generiranju programskog kôda te možemo reći kako je taj problem nedovoljno istražen.

5.3.2. Korištenje sustava vođenih LLM-om za odgovaranje na programerske upite

Prije svega, ograđujemo se od tvrdnje kako su rezultati naših opservacija znanstveni doprinos. Podaci ukazuju na to kako uočeni fenomen zaslužuje dodatno istraživanje. Prilikom prikupljanja podataka, koristili smo sustav ChatGPT iz razloga što je besplatan, lako dostupan te jedan od popularnijih, ako ne i najpopularniji, sustav današnjice za generiranje podataka.
Daljnje istraživanje zahtjeva komparativnu studiju u kojoj bismo uključili i druge utjecajne sustave vođene LLM-om, a pogotovo one koji su specijalizirani za automatsko generiranje programskog kôda poput sustava GitHub Copilot u vlasništvu tvrtke GitHub [115]. Razlog zbog kojeg sustavi velikih kompanija privlače istraživače leži u činjenici kako te tvrtke na raspolaganju imaju ogromne količine podataka i računalnih resursa potrebnih za uvježbavanje LLM-ova.

Korištenjem ChatGPT-a nad pitanjima izvučenim sa stranice Stack Overflow, uočili smo kako je ChatGPT dao slične, ako ne i identične odgovore koji su se mogli pronaći i na Stack Overflow stranicama. Generirani odgovori često su bili varijanta onih sa stranice Stack

Overflow, doduše stilski uređeni i drugačije sročeni od onih sa stranice Stack Overflow. No u suštini, ChatGPT nije ponudio novo rješenje tj. odgovor na pitanje.

Dolazimo do pretpostavki kako je sustav ChatGPT:

- došao u susret s podacima sa stranice Stack Overflow
- rijetko, ako uopće, u mogućnosti dati inovativne, a ispravne odgovore
  - Postavlja se pitanje iz kojeg izvora dolazi točan odgovor te postoji li razlog za zabrinutost kada je riječ o autorskim pravima
- elegantniji, korisniku prilagođen sustav za pretraživanje i generiranje odgovora na temelju postojeći podataka dostupnih putem Interneta
  - Mrežne tražilice nemaju tu sposobnost već nude statičke informacije koje zahtijevaju dodatan kognitivni proces kojeg korisnici često ne mogu priuštiti zbog nedostatka vremena ili predznanja

5.4. Analiza problema i slučajeva korištenja

Prethodna istraživanja fokus su stavila na rješavanje jednostavnih problema popraćenih jednostavnim opisom zadatka čija su rješenja kratka, što stvara jaz između potpunih složenih rješenja na koja nailazimo u svakodnevnom programerskom životu [116]. S obrazovnog aspekta, u sustavima formalnog obrazovanja gdje su jednostavni problemi svakodnevnica, došlo je do potrebe za preispitivanjem organizacije učenja i podučavanja, kao i arhitekture te načina korištenja sustava CMS i LMS [117].

Računalno programiranje dinamično je područje jer se konstantno mijenjaju programski jezici, ali i okviri te alati korišteni kao dodatak tijekom razvoja programske podrške korištenjem nekog programskog jezika. Time svako okruženje u kojem se razvija programska podrška komunicira „drugačijim" jezikom – stvaraju se dijalekti programskih jezika, a programer svakom promjenom radnog okruženja riskira da postane programer početnik.

Promjena okruženja sa sobom vuče i promjenu tehnologije, a time i potrebu za cjeloživotnim učenjem. Obrazovanje postaje dio ciklusa razvoja programske podrške što nas dodatno udaljava od vodopadnog modela razvoja programske podrške. Iako je programer možda bio ekspert u starom okruženju, u novom okruženju tu ulogu preuzima osoba s najviše domenskog znanja i iskustva.

Zbog toga što svaki proces obrazovanja zahtjeva uloženo vrijeme, a omjer eksperata naspram programera početnika ne ide u prilog ekspertima, na AI sustave gledamo kao na potencijalno rješenje ovog problema [116].

Primjeri korištenja AI sustava u programerskom obrazovanju su [117]:

- pružanje povratnih informacija u vidu primjera programskog kôda, alternativnih rješenja, analize i osvrta na učenički kôd, itd.
- generiranje programerskih zadataka
- objašnjavanje programskog kôda
- generiranje ilustrativnih primjera

S druge strane, programiranje bez korištenja IDE-a danas je skoro pa nezamislivo, a dovršavanje kôda jedna je od njihovih najčešće korištenih značajki [110].

Pisanje testova svakodnevni je dio faze testiranja i održavanja programske podrške, stoga je za očekivati kako postoji potreba za automatskim generiranjem i takvih programskih kôdova. No, javila se i potreba za generiranjem testova u svrhu generiranja programskog kôda iz razloga što je pisanje testova financijski i vremenski zahtjevno kako bi se radilo ručno, no i dalje potrebno kako bi se vršila evaluacija ispravnosti generiranog programskog kôda [111].

5.4.1. Obučavanje programera početnika

U sustavima za formalno i neformalno obučavanje programera početnika AI sustavi za automatsko generiranje programskog kôda mogu pomoći u velikom broju vertikala, kako ekspertima (Tablica 5.3.), pa tako i programerima početnicima (Tablica 5.4.) [112].

Tablica 5.3. Primjena AI sustava kao asistencija ekspertima

Aktivnost	**Detalji o aktivnosti**	**Broj radova po aktivnosti**	**Jedinstveni radovi**
Automatsko generiranje zadaća	Opisi zadataka, primjeri odgovora i objašnjenja	5 radova	7 radova
	Testni slučajevi za vježbe	3 rada	
	Personalizirani problemi	3 rada	
	Varijacije pitanja	2 rada	
Ocjenjivanje i vrednovanje	Ocjenjivanje zadataka	3 rada	5 radova
	Utvrđivanje područja u kojima studenti imaju poteškoća	1 rad	
	Generiranje povratnih informacija	4 rada	
UKUPNI BROJ JEDINSTVENIH RADOVA:			9 radova

Tablica 5.4. Primjena AI sustava kao asistencija programerima početnicima

Aktivnost	Broj radova
Generiranje zadataka za vježbu	5 radova
Generiranje primjera rješenja	5 radova
Generiranje alternativnih rješenja	4 rada
Unaprjeđivanje studentskog kôda	2 rada
Pojašnjavanje poruka o grešci te nuđenje savjeta	3 rada
Podrška kod shvaćanja programskih koncepata	6 radova
Nuđenje savjeta za ispravljanje sintaktičkih grešaka	2 rada
Objašnjenje programskog kôda	6 radova
BROJ JEDINSTVENIH RADOVA:	14 radova

5.4.2. Natjecateljsko računalno programiranje

Korak naprijed u povećavanju složenosti problema za koja generiramo rješenje korištenje je programskih problema iz domene programerskih natjecanja u podatkovnim skupovima za uvježbavanje i vrednovanje AI modela. Razlog leži u činjenici da je za rješavanje takvih problema potrebno razumijevanje složenih specifikacija pisanih netehničkim prirodnim jezikom, korištenje logike za rasuđivanje pri susretu s dosad neviđenim problemom, svladavanje širokog raspona algoritama i podatkovnih struktura, a naposljetku i implementacija rješenja koje doseže i do nekoliko stotina linija kôda pa čak i više projektnih datoteka [6], [18].

Natjecanja iz programiranja poput ICPC-a i IOI-a veoma su popularna i prestižna natjecanja te privlače na tisuće natjecatelja iz cijelog svijeta, što ukazuje na izazovnost takvih natjecanja, no također i na njihovu robusnost kojom predstavljaju značaju referentnu točku za mjerenje uspješnosti AI sustava u zadaći automatskog generiranja programskog kôda.

Postoje tri ključna koraka kada je riječ o natjecateljskom računalnom programiranju, a to su:

1. analiza teksta zadatka napisanog prirodnim jezikom popraćenog specifikacijom oblika ulaznih i izlaznih podataka te reprezentativnim primjerima ulazno-izlaznih podataka
2. pristupanje problemu kroz smišljanje ispravnog, ali učinkovitog algoritma koji nudi rješenje problema
3. implementacija osmišljenog algoritma, kao i odabir programskog jezika, uz popratno testiranje, evaluaciju te otklanjanje grešaka – čime se ukazuje na iterativnost procesa razvoja programske podrške

5.4.3. Dovršavanje programskog kôda

Kod automatskog dovršavanja programskog kôda, AI sustav fokusira se na podatkovni, ali i funkcionalni dio; drugim riječima, na jedan ili više programski jezik, ali i na funkcionalnost specifičnog IDE-a.

Tablica 5.5. Prikaz jezične agnostičnosti AI asistenta

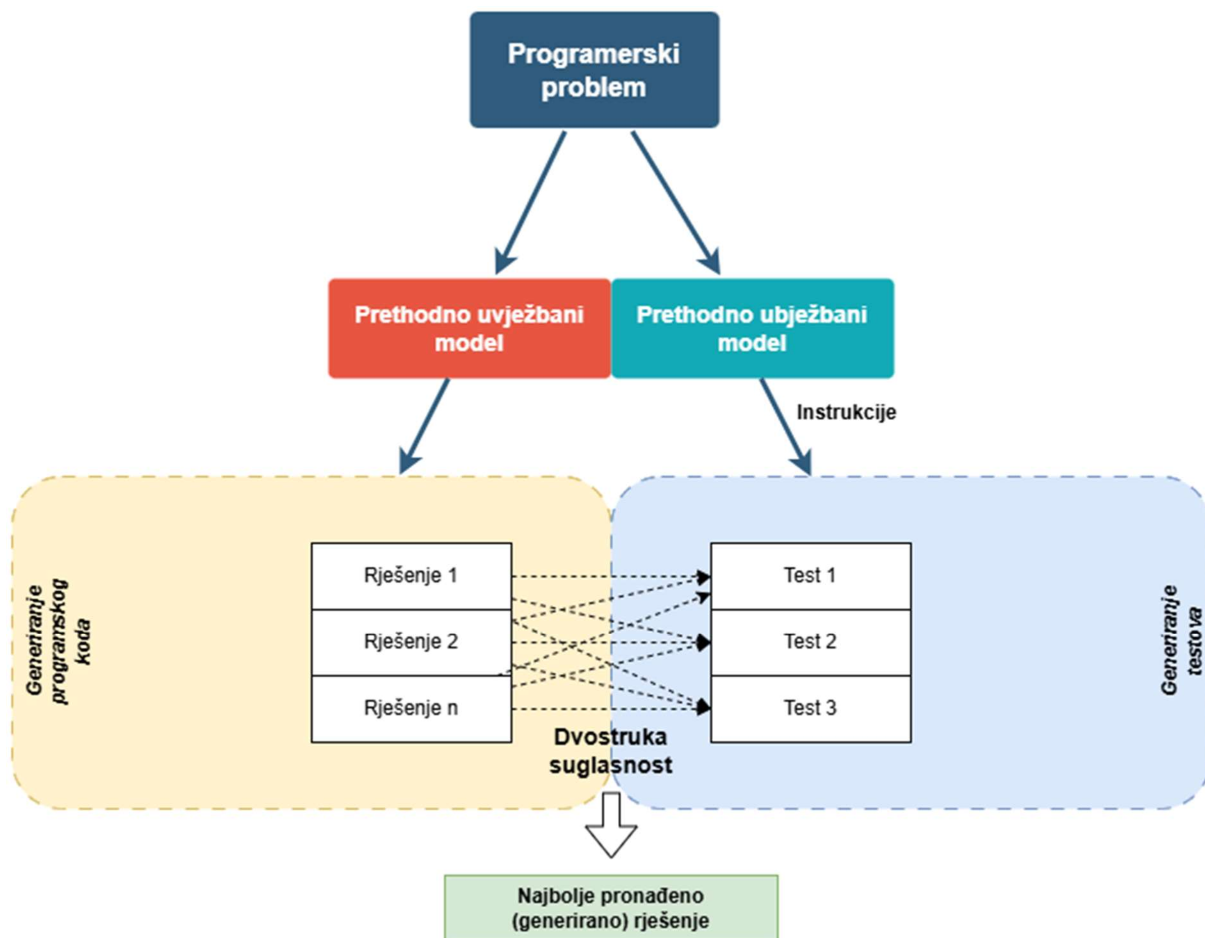
Programski jezici	IDE / uređivač koda	Naziv AI asistenta
C# Python JavaScript TypeScript	Visual Studio Code Azure Notebook	*IntelliCode Compose*

Korištenje jednog AI asistentskog sustava u više okruženja te u kontekstu više programskih jezika postala je industrijska norma (Tablica 5.5.). Valja napomenuti kako su programski jezici i IDE-i koji su bili fokus veoma popularni, što znači kako iza njih stoje ogromne količine relevantnih podataka reda veličina nekoliko stotina terabajta ili petabajta. S vremenom će količina dostupnih podataka postati još veća, a sustavi za generiranje programskog kôda dodatno će doprinijeti rastu količine dostupnih podataka.

Postavlja se pitanje mogu li asistentski sustavi pratiti trendove i novonastale tehnologije poželjnom razinom okretnosti te hoće li razvoj AI sustava ubrzati ili usporiti razvoj novih tehnologija.

5.4.4. Automatsko generiranje testova

Automatsko generiranje testova služi programerima kako bi osigurali te održavali pouzdanost i funkcionalnu ispravnost računalnog programa [116]. Uz generiranje programskog kôda možemo generirati i jedinične testove za vrednovanje serijski ili paralelno generiranih kôdova. Generirani jedinični testovi pomažu AI modelu pri vrednovanju generiranih rješenja kako bi se odabralo najbolje rješenje (Slika 5.1.).



Slika 5.1. Proces vrednovanja AI modela pomoću generiranja jediničnih testova

5.5. Prikupljanje podataka

Za uvježbavanje i vrednovanje AI modela potrebno je na raspolaganju imati velike količine podataka [117]. Princip otvorenih podataka omogućuje nam jednostavan i otvoren pristup podacima koje zatim možemo koristiti, uređivati i dijeliti s drugima u bilo koju svrhu [118]. U pozadini takvih podataka može stajati potreba za dijeljenjem znanja stoga nerijetko govorimo i o otvorenom znanju. Konkretno u našem slučaju govorimo o otvorenosti izvornog koda.

Razvojem internetskih usluga, na mrežu su se postavile brojne aplikacije čija je zadaća upravljanje podacima, a nerijetko i znanjem te procesom učenja [117]. Povećanjem računalne

pismenosti i okretnosti, raste broj aktivnih korisnika (u našem slučaju programera). Posljedično raste i broj dostupnog sadržaja, a njime i implicitnog znanja kojeg korisnici dijele.

Jedna od aplikacija preko koje možemo doći do velike količine podataka relevantnih za naše istraživanje je GitHub. Riječ je o platformi koja dopušta razvijačima programske podrške stvaranje, pohranjivanje, upravljanje i dijeljenje vlastitog programskog kôda. Zbog lakog pristupa programskom kôdu, oblikuju se podatkovni skupovi prikupljenih sa sustava GitHub i njemu sličnih sustava kako bi se uvježbavali sustavi umjetne inteligencije za razne aktivnosti tijekom procesa razvoja programske podrške [117]. Svaki korisnik u sustavu GitHub može ostaviti svoj glas (zvijezdu) na repozitoriju kôda. Stoga broj zvijezda ukazuje na popularnost repozitorija te se taj broj može koristiti za usmjeravanje pri automatskom prikupljanju podataka [118]. Osim toga, valja pripaziti na to kako postoji i funkcionalnost račvanja repozitorija, stoga je od iznimne važnosti ne uzimati u obzir repozitorije koji su rezultat račvanja kako bi se izbjegli duplikati u skupu podataka za uvježbavanje [15].

Otvorenost podataka omogućila je lakše dijeljenje i ponovno korištenje gotovih skupova podataka u svrhu istraživanja. Stranice poput *Papers with Code* [119] nude skupove podataka namijenjene za istraživanje određenih problema, a uz podatke dolaze i rezultati postojećih istraživanja s kojima se mogu usporediti dobiveni rezultati.

5.6. Korištenje podataka

Prikupljeni podaci koriste su u nekoliko faza pri strojnom učenju. Faza prethodnog uvježbavanja koristi se kako bi model [6]:

- naučio što predstavlja dobar programski kôd
- kako generirati programski kôd bez poteškoća

Faza finog podešavanja koristi se kako bi model prilagodili domeni korištenja [18] npr. natjecateljskom programiranju.

AI model koji uvježbavamo može podržavati više programskih jezika pa u tom slučaju govorimo o višejezičnim modelima. Prednost korištenja takvih modela vidi se kod jezika oskudnih količinom dostupnih podataka jer uživaju benefit postojanja dostatne količine podataka iz drugih programskih jezika [6].

U slučaju korištenja višejezičnih modela, postoji nekoliko načina za njihovo uvježbavanje [19]:

- *korištenje jezično-agnostičke osnovice* - pristup kod kojeg se ne uzima u obzir informacija o programskom jeziku već se uvježbavanje vrši kao da je riječ o jednojezičnom modelu.

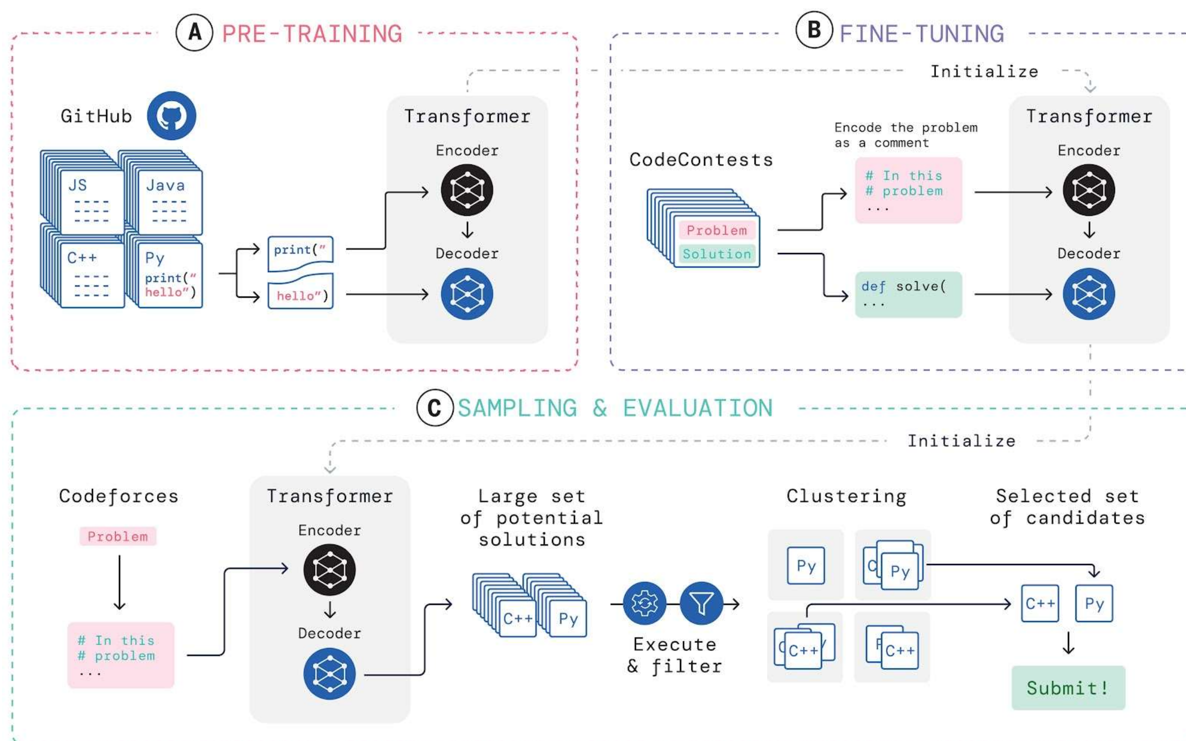
- *pridruživanje jezičnog tipa* - radi se o pridodavanju metapodatka o programskom jeziku svakoj oznaci u ulaznom podatkovnom skupu tako što se programski jezik predstavlja kao matrica koja se pribraja matrici oznaka.
- *Korištenje kontrolnih kôdova specifičnih za programski jezik* - metoda kod koje se kontrolni kôd za prikaz programskog jezika dodaje na početak sekvence ne bi li se neuronskoj mreži signaliziralo o kojem je programskom jeziku riječ
- *klasificiranje programskog jezika tijekom prethodnog uvježbavanja* - kod kojeg se model uvježbava s dva cilja: modeliranje programskog jezika i klasifikacije višestrukog odabira u svrhu prepoznavanja programskog jezika. Kažemo da u ovom slučaju model ima dvije glave.

5.7. Primjer AI modela za automatsko generiranje programskog kôda

Model AlphaCode implementira arhitekturu enkoder-dekoder te pomoću te dvije neuronske mreže uvježbava se nad podatkovnim skupom. Te dvije neuronske mreže tvore tzv. transformator (Slika 5.2.) [18].

Koraci u radu modela AlphaCode su sljedeći [6]:

1. **Prethodno uvježbavanje** - programski kôd prikupljen s GitHub-a nasumično se dijeli na dva dijela. Prvo dio služi kao ulaz u enkoder, dok se drugi dio koristi za uvježbavanje dekodera.
2. **Fino podešavanje** - problem se predstavlja u obliku komentara u kôdu te se šalje enkoderu na ulaz, dok se dekoder uvježbava u svrhu generiranja rješenja na temelju rješenja danog problema u odabranom programskom jeziku.
3. **Uzorkovanje i evaluacija** - AlphaCode generira više uzoraka rješenja danog problema, nakon čega ih izvršava ne bi li filtrirao loše uzorke te grupirao preostale uzorke kako bi ih predao sustavu na evaluaciju. Platforma za programerska natjecanja *Codeforces* koristila se kao instrument za vrednovanje.



Slika 5.2. Arhitektura AlphaCode modela, iz [18]. *Pretisak uz dopuštenje AAAS-a.*

5.8. Oblikovanje korisničkih interakcija

Slijedi nekoliko primjera korisničkih interakcija s asistentskim sustavima gdje se jasno vidi simbiotički odnos između programera i AI sustava.

5.8.1. Asistentski sustavi za automatsko generiranje programskog kôda

Visual Studio Code primjer je uređivača programskog kôda s kojim se može integrirati velika većina trenutno postojećih AI sustava kako bi ponudili prijedlog za dovršavanje kôda (Slika 5.3.). Iako ovakva vrsta interakcije između programera i AI sustava značajno povećava produktivnost programera, važno je istaknuti kako postoji imperativna potreba programera za kontrolom nad sugestijama [18].

```
# From https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

import torch
import torchvision
import torchvision.transforms as transforms

transform = transforms.Compose([transforms.ToTensor(),transforms.Normalize((0.5, 0.5, 0.

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset,batch_size=1,shuffle=False,num_workers
```

Slika 5.3. Primjer interakcije programera s asistentskim sustavom

5.8.2. Vizualizacija podataka

Zbog velike količine podataka koju AI sustavi obrađuju u različitim fazama, javlja se potreba za vizualizacijom podataka kako bi se pokazali apstraktni odnosi među podatkovnim atomima. Primjer može biti vizualizacija stabla odluka koje je temelj za generiranje prijedloga u svrhu automatskog dovršavanja kôda (Slika 5.4.). Crvenom bojom označena je odluka AI sustava tj. kôd koji je AI sustav odlučio predstaviti korisniku kao najvjerojatnije ispravno rješenje.

Slika 5.4. Stablo odluka pri dovršavanju kôda

6. VREDNOVANJE UČINKOVITOSTI SUSTAVA ZA AUTOMATSKO GENERIRANJE KÔDA

Generiranje potpunih rješenja u programskom jeziku opće svrhe poput C++-a, Jave, Pythona ili JavaScripta ostaje otvoren problem [6]. Osim zadovoljavanja mjera poput ispravnosti i učinkovitosti rješenja te pouzdanog generiranja programskog kôda, AI sustavi trebaju generirati programski kôd koji zadovoljava potrebe programera, a ne samo krajnjih korisnika generirane programske podrške. Neki od zahtjeva usmjerenih prema AI sustavima za automatsko generiranje kôda su:

- funkcionalna ispravnost - svojstvo programa da za svaki mogući ulaz na izlazu daje očekivani rezultat [19]
- *prirodnost* - za generirani programski kôd kažemo da je prirodan kada je konvencionalan, idiomatski i prepoznatljiv jer pomaže u razumijevanju i održavanju programske podrške [6]

6.1. Vrednovanje učinkovitosti kod natjecateljskog računalnog programiranja

U slučaju modela AlphaCode, platforma za programerska natjecanja *CodeForce* koristila se za evaluaciju uspješnosti modela. Pokazalo se kako je model riješio probleme s kojima se dotad nije susreo. AlphaCode je zauzeo prosječni rang među 54.3% najbolje rangiranih natjecatelja kada se ograničio na samo 10 predanih kôdova po problemu. Čak 66% zadataka riješio je već s prvim predanim kôdom. Smatra se kako taj rang odgovara rangu programera početnika, osobe s nekoliko mjeseci do godine dana iskustva u računalnom programiranju [50].

6.2. Vrednovanje učinkovitosti kod dovršavanja kôda

S obzirom kako čovjek i računalo kodiraju u tandemu, prevelika latencija nije prihvatljiva kod ovakvih simbiotičkih sustava. Zahtjev za maksimalnom latencijom od 100ms po pozivu stvorio je potrebu za uvođenjem predmemorije (*eng. cache*) na klijentskoj strani kako bi se uklonila potreba za pozivanjem poslužitelja [51].

6.3. Vrednovanje višejezičnih modela za dovršavanje kôda

Aktivnosti dovršavanja kôda može se pristupiti korištenjem više različitih arhitektura AI modela. Model s dvije glave parira modelu koji koristi kontrolne kôdove bez povećavanja veličine modela (Tablica 6.1.) [71].

Tablica 6.1. Rezultati vrednovanja višejezičnih modela za dovršavanje kôda

Model	PPL	ROUGE-L		Udaljenost uređivanja (%)	Veličina modela
		Preciznost	*Odziv*		
Jezično-agnostička osnovica	2.15	0.25	0.24	56.3	374M
Pridruživanje jezičnog tipa	1.94	0.52	0.66	71.7	379M
Kontrolni kôdovi	1.73	0.64	0.75	81.5	374M
Model MultiGPT-C (dvije glave)	1.65	0.66	0.76	82.1	374M

6.4. Vrednovanje količine resursa potrebnih za automatsko generiranje kôda

Uspješnost modela dolazi po cijeni velike potrošnje električne energije. Za model AlphaCode zabilježila se potrošnja energije otprilike 16 puta veća od prosječnog američkog kućanstva [52]. U slučaju kada model osim kôda treba generirati i testove potrebno je još više računalne moći obrade podataka, što implicira na veću potrošnju energije [51]. Nažalost, većina radova u literaturi ne navodi konkretne brojeve, čak i ako govore o problemu ekonomičnosti AI modela. Iz toga razloga nismo bili u mogućnosti provesti valjanu komparativnu analizu.

Tablica 6.2. Rezultati analize ekonomičnosti AI modela [52], [53]

Model	Broj parametara	Resursi	Rad
AlphaCode	41 milijarda	2,149 petaflop/s-dan	175 MWh
OPT-175B	175 milijardi	809,472 GPU-sati	356 MWh
BLOOM-175B	175 milijardi	1,082,880 GPU-sati	475 MWh
LLaMA-7B	7 milijardi	82,432 GPU-sati	36 MWh
LLaMA-13B	13 milijardi	135,168 GPU-sati	59 MWh
LLaMA-33B	33 milijardi	530,432 GPU-sati	233 MWh
LLaMA-65B	65 milijardi	1,022,362 GPU-sati	449 MWh

Tek dva znanstvena rada prikupljena pregledom literature transparentno su iznijela podatke o ekonomičnosti korištenih AI modela (Tablica 6.2.). To ukazuje na nedostatak fokusa u znanstvenim istraživanjima na očigledne nedostatke tih modela.

U budućim istraživanjima trebat ćemo staviti fokus i na ekonomičnost pristupa, a ne isključivo na mjerenje uspješnosti. Od iznimne je važnosti ukazati na omjer cijene i uspješnosti modela kako bi se moglo usmjeriti daljnje istraživanje, slično kao što nam je bitno da generirani kôd ne bude samo ispravan već i učinkovit.

Neki radovi pronađeni pregledom literature navode kako je ekonomičnost korištenja ovakvih sustava problematična [73], ali ne iznose konkretne podatke o potrošnji energije.

6.5. Vrednovanje učinkovitosti automatski generiranih testova

Kako bi se pouzdano mjerila funkcionalna ispravnost generiranih programskih kôdova, potrebno je izmjeriti preciznost, ali i toksičnost generiranih testova. Generirani test smatramo preciznim ako je kanonsko rješenje iz podatkovnog skupa (za testiranje ili evaluaciju) uspješno zadovoljilo testne slučajeve tog testa [115].

Toksičnost testa smatramo svojstvom kod kojeg slučaj testa zadovoljava generirani programski kôd, ali ne i kanonsko rješenje iz podatkovnog skupa [116]. Pritom pretpostavljamo kako se podatkovni skup sastoji od funkcionalno ispravnih i reprezentativnih programskih kôdova.

6.6. Utjecaj na formalno učenje programiranja kod programera početnika

Utjecaj AI sustava poput ChatGPT-a može se osjetiti i nad formalnim obrazovnim sustavima. Generiranje sadržaja postalo je lakše no ikad, ali dolazi sa skupom problema. Pregledom literature naišli smo na navodi sljedećih značajnih problema [117]:

- kršenje akademskih pravila
- kršenje autorskih prava
- licenciranje ponovnog korištenja programskog kôda
- smanjena ekološka i ekonomska održivost - ovakvi sustavi troše enormne količine energije i računalnih resursa pri uvježbavanju i radu
- neprikladnost programerima početnicima jer je generirani kôd konceptualno zahtjevniji od početničkog kôda te je moguće svjesno ili nesvjesno korištenje pristupa i stilova koji nisu dio kurikuluma
- štetne pristranosti - zabilježeno je kako neki modeli bivaju skloni štetnim utjecajima poput rasizma

- manjak sigurnost - mnogi trenutni modeli skloni su generiranju programskog kôda koji krije nesigurnosti
- gubitak povjerenja programera u AI sustave
- razvoj ovisnosti o ovakvim sustavima kod programera početnika

6.7. Vrednovanje generiranog sadržaja u svrhu obrazovanja programera početnika

Kako bismo utvrdili upotrebljivost AI sustava pri interakciji s programerima početnicima, možemo se poslužiti postojećim kvalitativnim i kvantitativnim mjerama (Tablica 6.3.) za vrednovanje automatskog generiranja sadržaja [112].

Tablica 6.3. Mjere za vrednovanje generiranog sadržaja

Kvalitativne mjere programskih vježbi	**Opis**
smislenost	opisuje li tekst zadatka praktični problem koji se može dati programerima početnicima kao zadatak
inovativnost	je li taj isti ili sličan zadatak već dostupan na mreži
kontekstualnost	odgovara li zadatak tematski nastavnom sadržaju
spremnost za korištenje	količina ručnog rada kojeg učitelj treba odraditi nad vježbom, rješenjem ili testovima
Kvantitativni indikatori programskih vježbi	**Opis**
izvršivost generiranih rješenja	mogu li se izvršiti generirana rješenja
pouzdanost	prolaze li generirana rješenja automatske testove
pokrivenost	postotak linija kôda pokrivenih automatskim testovima u trenutku pokretanja kôda
Mjere za objašnjivost koda	**Opis**
prisutnost i frekvencija grešaka	vrste prisutnih grešaka, učestalost grešaka
potpunost objašnjenja kôda	jesu li svi dijelovi programskog kôda objašnjeni generiranim objašnjenjima
preciznost objašnjenja	omjer točnih objašnjenja i ukupno generiranih objašnjenja po linijama kôda

6.8. Transparentnost AI sustava

Ljudi trebaju moći vjerovati AI sustavu koji im stoji na raspolaganju [119]. Kako bi se postiglo povjerenje, svaki sustav mora proći kroz proces provjere ispravnosti i valjanosti. Ispravnost se odnosi na provjeru zadovoljavanja specifikacije, dok se valjanost odnosi na zadovoljavanje potreba korisnika. Certifikati su jedan od popularnih instrumenata za stjecanje povjerenja u sustav. Dodatni aspekt povjerenja je transparentnost sustava, a odnosi se na znanje korisnika o unutrašnjim djelovanjima u AI sustavu - preciznije: shvaćanje radi li sustav protiv njih iz bilo kojeg razloga (npr. zbog greške u sustavu, zlonamjernog korištenja, pristranosti, itd.).

Kako bi se osiguralo etičko korištenje AI sustava, brojne vladine i istraživačke skupine zahtijevaju transparentnost takvih sustava kako bi se omogućilo kritičko promišljanje o njima [120]. Kada sustav za automatsko generiranje programskog kôda proizvede kôd koji nije ispravan ili valjan, dužan krajnjim korisnicima je objašnjenje. Takav AI sustav nazivamo objašnjivim AI sustavom (*eng. eXplainable AI, kr. XAI*) [116].

Transparentnost AI sustava nadovezuje se na mjeru objašnjivosti kôda iz prethodnog primjera (Tablica 6.3.).

7. BUDUĆI IZAZOVI I PRILIKE

Rastom industrije generativnih modela osjeća se pritisak na industriju razvoja programske podrške da u što kraćem vremenu proizvede programsku podršku koja zadovoljava korisničke zahtjeve i specifikaciju. No, utjecaj generativnih AI modela ne utječe samo na domenu industrijskog programiranja. Njihov utjecaj širi se i na domene obrazovanja, druge industrije, ali i mnoge znanstvene domene gdje se programiranje koristi kao jedna od metoda obrade podataka. Razlog iza tako širokog utjecaja generativnih AI modela leži u činjenici kako smo u mogućnosti u vrlo kratkom vremenskom periodu doći do velike količine vrijednog sadržaja. Taj sadržaj može imati monetarnu, dokimološku ili karijernu vrijednost.

Stoga smo suočeni s brojnim etičkim dilemama koje nesmotreno korištenje ovakvih modela povlači sa sobom [117]. Pitanje autorskih prava, korištenje i zlouporaba kôda koji nije pisan tj. generiran po utvrđenim standardima i smjernicama, kao i odgovornost za kvalitetu generiranog sadržaja samo su neki od problema s kojim smo suočeni u industriji razvoja programske podrške. Zbog postojanja takvih problema uviđamo kako, u znanstvenom smislu, problematici moramo pristupati interdisciplinarno, a ne isključivo tehnički.

Osim etičkih problema, još 80-ih godina prošlog stoljeća uočene su tri komunikacijske barijere: *kognitivna, jezična tj. sadržajna* i *interaktivna* barijera [1]. Jezičnu tj. sadržajnu barijeru probili smo uspješnim uvježbavanjem LLM-ova u svrhu komunikacije prirodnim jezikom. No, problemi *kognitivne* i *interaktivne* prirode i dalje su prisutni iz razloga što je postupak uvježbavanja takvih modela vođen podacima tj. sadržajem te se *kognitivna* i *interaktivna* barijera implicitno pokušavaju probiti putem modeliranja vođenog podatcima. U svrhu daljnjih istraživanja postavljamo sljedeća istraživačka pitanja:

- kako postići razinu transparentnosti koja bi nam dala uvid u to što model zna te komunicira li na svim razinama znanja jednako; preciznije: ukazuju li komunikacijske sposobnosti modela na posjedovanje sposobnosti razumijevanja i obavljanja srodnih aktivnosti poput traženja grešaka u programskom kôdu?
- kako postići sustav visoke upotrebljivosti i korisnosti, s obzirom na to kako je jedan od ciljeva AI sustava za automatsko programiranje ostvarivanje što veće razine automatizacije? Iz ovoga proizlaze sljedeća dva potpitanja:
    - o kako u tom slučaju postižemo prirodnu interakciju između čovjeka i AI sustava?
    - o Što je neophodno ostvariti kako ni čovjek niti AI sustav ne bi ispali iz petlje (*čovjek u petlji – no, je li i AI sustav*)?

8. ZAKLJUČAK

S obzirom na rastući broj dostupnih, komercijalnih, ali zatvorenih AI sustava za automatsko generiranje programskog kôda, postaje sve teže evaluirati njihovu učinkovitost u kvalitativnom i kvantitativnom smislu jer zatvorenost AI sustava i/ili modela negativno utječe na ponovljivost istraživanja, kao i na razvoj transparentnosti kada je riječ o tim sustavima.

Ovim radom postavljen je teorijski temelj za buduća istraživanja u području korištenja AI sustava za automatsko generiranje programskog kôda s naglaskom na otvorenost i transparentnost tih sustava, kao i rezultata istraživanja provedenih nad njima, a sastoji se od:

- izdvojenih i jasno definiranih temeljnih koncepata iz područja automatskog generiranja programskog kôda, automatskog programiranja te automatske sinteze programa
- potrebnih kompetencija iz područja AI modeliranja u svrhu generiranja programskog kôda, što uključuje:
    - generativno (duboko / strojno) učenje
    - genetsko programiranje
    - programiranje jatom
- primjera primjene AI modeliranja u svrhu analize podatkovnog skupa za uvježbavanje i vrednovanje podatkovnih skupova za strojno učenje koji se sastoje od programskih kôdova [23]
- rezultata analize relevantnih istraživačkih radova iz područja automatskog generiranja programskog kôda korištenjem AI modela dubokog učenja pomoću kojih smo ustanovili kako postoje jazovi između procesa strojnog učenja, upravljanja znanjem te upravljanja podacima
- vrednovanja postojećih pristupa te prijedloga arhitekture AI sustava za buduća istraživanja, a koja bi se sastojala od:
    - sustava za upravljanje podacima (podatkovnim skupovima i generiranim sadržajem)
    - sustava za upravljanje strojnim učenjem
    - sustava za upravljanje znanjem što uključuje i predstavljanje znanja te njegov utjecaj na proces strojnog učenja

Na temelju provedene analize, planiramo provesti istraživanje koje će se fokusirati na premošćivanje jednog od utvrđenih jazova primjenom predložene troslojne arhitekture AI sustava. Fokus doktorata bit će na:

- analizi utjecaja višeobličnosti korisničkih zahtjeva i tehničkih specifikacija na pouzdanost generiranog programskog koda.
- analizi i evaluaciji tehnika za podizanje razine transparentnosti kada je riječ o unutarnjim stanjima AI modela u svrhu stjecanja spoznaje o znanju koje model posjeduje nakon faze uvježbavanja te faze finog podešavanja.
- analizi učinkovitosti AI sustava tj. modela u rješavanju programskim problema van skupa problema na kojima su uvježbani.

LITERATURA

[1] C. Rich i R. C. Waters, „Automatic programming: myths and prospects“, *Computer (Long Beach Calif.)*, sv. 21, izd. 8, str. 40–51, kol. 1988, doi: 10.1109/2.75.

[2] S. Russell i P. Norvig, *Artificial Intelligence: A modern approach*. Pearson Higher Education, 2019.

[3] C. E. A. Coello, M. N. Alimam, i R. Kouatly, „Effectiveness of ChatGPT in coding: A comparative analysis of popular large language models“, *Digit.*, sv. 4, str. 114–125, sij. 2024, doi: 10.3390/digital4010005.

[4] D. Foster, *Generative deep learning: Teaching machines to paint, write, compose, and play*. 2023.

[5] M. Katenova i K. Turmaganbetova, „Economic consequences of artificial intelligence“, u *Advances in Finance, Accounting, and Economics*, IGI Global, 2025, str. 21–44. doi: 10.4018/979-8-3693-7036-0.ch002.

[6] Y. Li *i sur.*, „Competition-level code generation with AlphaCode“, *Science*, sv. 378, izd. 6624, str. 1092–1097, pros. 2022, doi: 10.1126/science.abq1158.

[7] N. D. Matsakis i F. S. Klock II, „The rust language“, *ACM SIGAda Ada Lett.*, sv. 34, izd. 3, str. 103–104, stu. 2014, doi: 10.1145/2692956.2663188.

[8] M. Resnick *i sur.*, „Scratch“, *Commun. ACM*, sv. 52, izd. 11, str. 60–67, stu. 2009, doi: 10.1145/1592761.1592779.

[9] M. Chen *i sur.*, „Evaluating large language models trained on code“, *arXiv [cs.LG]*, 07. srpanj 2021. [Na internetu]. Dostupno na: http://arxiv.org/abs/2107.03374

[10] X. Chen, C. Liu, i D. Song, „Execution-guided neural program synthesis“, *Int Conf Learn Represent*, ruj. 2018, [Na internetu]. Dostupno na: https://openreview.net/forum?id=H1gfOiAqYm

[11] Nam, „A survey of automatic code generation from natural language“, *J. Inf. Process. Syst.*, sv. 17, izd. 3, str. 537–555, lip. 2021, doi: 10.3745/JIPS.04.0216.

[12] W. Selden, „Need for an algorithm“, *Commun. ACM*, sv. 1, izd. 4, str. 7–9, tra. 1958, doi: 10.1145/368796.368805.

[13] W. Takerngsaksiri *i sur.*, „Human-in-the-loop software development agents“, *arXiv [cs.SE]*, 19. studeni 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2411.12924

[14] M. Santolucito, „Human-in-the-loop program synthesis for live coding“, u *Proceedings of the 9th ACM SIGPLAN International Workshop on Functional Art, Music, Modelling, and Design*, New York, NY, USA: ACM, kol. 2021. doi: 10.1145/3471872.3472972.

[15] T. H. M. Le, H. Chen, i M. A. Babar, „Deep learning for source code modeling and generation“, *ACM Comput. Surv.*, sv. 53, izd. 3, str. 1–38, svi. 2021, doi: 10.1145/3383458.

[16] S. Gulwani, „Automating string processing in spreadsheets using input-output examples“, u *Proceedings of the 38th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, New York, NY, USA: ACM, sij. 2011. doi: 10.1145/1926385.1926423.

[17] M. Bruch, M. Monperrus, i M. Mezini, „Learning from examples to improve code completion systems“, u *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, New York, NY, USA: ACM, kol. 2009. doi: 10.1145/1595696.1595728.

[18] A. Svyatkovskiy, S. K. Deng, S. Fu, i N. Sundaresan, „IntelliCode Compose: Code Generation Using Transformer“, *arXiv [cs.CL]*, 16. svibanj 2020. doi: 10.1145/3368089.3417058.

[19] B. Chen *i sur.*, „CodeT: Code Generation with Generated Tests“, *arXiv [cs.CL]*, 21. srpanj 2022. [Na internetu]. Dostupno na: http://arxiv.org/abs/2207.10397

[20] M. S. Gharajeh, „Waterative model: An integration of the waterfall and iterative software development paradigms“, *Database Syst. J*, 2019, [Na internetu]. Dostupno na: https://www.researchgate.net/profile/Mohammad-Samadi-8/publication/335842857_Waterative_Model_an_Integration_of_the_Waterfall_and_Iterative_Software_Development_Paradigms/links/5d7ff65a92851c22d5dd2879/Waterative-Model-an-Integration-of-the-Waterfall-and-Iterative-Software-Development-Paradigms.pdf
[21] T. Dyba i T. Dingsoyr, „What do we know about agile software development?“, *IEEE Softw.*, sv. 26, izd. 5, str. 6–9, ruj. 2009, doi: 10.1109/ms.2009.145.
[22] M. Jevtić, S. Mladenović, i G. Zaharija, „The Prospect of Using Automatic Programming Assistant for Providing Direct Feedback in an Online Learning Environment“, u *International Conference proceedings: New Perspectives in Science Education*, B. Kummert, S. Kapelari, T. Trencheva, i T. Todorova, Ur., Bolonja: Filodiritto Publisher, 2021, str. 461–465.
[23] M. Jevtić, S. Mladenović, i A. Granić, „Source Code Analysis in Programming Education: Evaluating Learning Content with Self-Organizing Maps“, *Applied Sciences*, sv. 13, izd. 9, 2023.
[24] „State of AI Code Generation in 2023“, Locofy Blogs. Pristupljeno: 12. travanj 2025. [Na internetu]. Dostupno na: https://www.locofy.ai/blog/state-of-ai-code-generation-in-2023
[25] , „The 2025 state of AI in code generation“, Djimit van data naar doen. Pristupljeno: 19. veljača 2026. [Na internetu]. Dostupno na: https://djimit.nl/the-2025-state-of-ai-in-code-generation/
[26] M. Siswo Utomo, E. Utami, i A. Kusrini, „Machine learning innovations in code generation: A systematic literature review of methods, challenges and directions“, u *2024 International Conference on Information Technology and Computing (ICITCOM)*, IEEE, 2024, str. 24–29.
[27] D. Floreano i C. Mattiussi, *Bio-inspired artificial intelligence. in Intelligent Robotics and Autonomous Agents series*. London, England: MIT Press, 2025.
[28] P. Kodytek, A. Bodzas, i J. Zidek, „Correction: Automated code development based on genetic programming in graphical programming language: A pilot study“, *PLoS One*, sv. 19, izd. 4, str. e0302986, tra. 2024, doi: 10.1371/journal.pone.0302986.
[29] J. Koza, „Genetic programming as a means for programming computers by natural selection“, *Stat. Comput.*, sv. 4, izd. 2, lip. 1994, doi: 10.1007/bf00175355.
[30] N. Tao, A. Ventresque, i T. Saber, „Program synthesis with generative pre-trained transformers and grammar-guided genetic programming grammar“, u *2023 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*, IEEE, lis. 2023, str. 1–6. doi: 10.1109/la-cci58595.2023.10409384.
[31] M. Brambilla, E. Ferrante, M. Birattari, i M. Dorigo, „Swarm robotics: a review from the swarm engineering perspective“, *Swarm Intell.*, sv. 7, izd. 1, str. 1–41, ožu. 2013, doi: 10.1007/s11721-012-0075-2.
[32] W. W. Are, „arxiv.org e-Print archive“, *arXiv*. Pristupljeno: 13. travanj 2025. [Na internetu]. Dostupno na: https://arxiv.org/
[33] M. Gusenbauer i N. R. Haddaway, „Which academic search systems are suitable for systematic reviews or meta-analyses? Evaluating retrieval qualities of Google Scholar, PubMed, and 26 other resources“, *Res. Synth. Methods*, sv. 11, izd. 2, str. 181–217, ožu. 2020, doi: 10.1002/jrsm.1378.
[34] „Papers with Code - The latest in Machine Learning“. Pristupljeno: 13. travanj 2025. [Na internetu]. Dostupno na: https://paperswithcode.com/
[35] „PRISMA 2020 flow diagram —“, PRISMA statement. Pristupljeno: 13. travanj 2025. [Na internetu]. Dostupno na: https://www.prisma-statement.org/prisma-2020-flow-diagram

[36] N. R. Haddaway, M. J. Page, C. C. Pritchard, i L. A. Mcguinness, „“: An R package and Shiny app for producing PRISMA 2020-compliant flow diagrams, with interactivity for optimised digital transparency and Open Synthesis“, *Campbell Syst Rev*, sv. 18, izd. 2, 2022.

[37] „Papers with code - DSEval-LeetCode benchmark (code generation)“. Pristupljeno: 14. travanj 2025. [Na internetu]. Dostupno na: https://paperswithcode.com/sota/code-generation-on-dseval-leetcode

[38] L. Zhong, Z. Wang, i J. Shang, „Debug like a human: A Large Language Model Debugger via verifying runtime execution step-by-step“, *arXiv [cs.SE]*, 24. veljača 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2402.16906

[39] Y. Hu *i sur.*, „QualityFlow: An agentic workflow for program synthesis controlled by LLM Quality Checks“, *arXiv [cs.SE]*, 2025. doi: 10.48550/ARXIV.2501.17167.

[40] T. Guo i H. Gao, „Content enhanced BERT-based text-to-SQL generation“, *arXiv [cs.CL]*, 2019. doi: 10.48550/ARXIV.1910.07179.

[41] A. S. Soliman, M. M. Hadhoud, i S. I. Shaheen, „MarianCG: a code generation transformer model inspired by machine translation“, *J. Eng. Appl. Sci.*, sv. 69, izd. 1, pros. 2022, doi: 10.1186/s44147-022-00159-4.

[42] V. Lomshakov, S. Kovalchuk, M. Omelchenko, S. Nikolenko, i A. Aliev, „Fine-Tuning Large Language Models for Answering Programming Questions with Code Snippets“, str. 171–179, 2023.

[43] B. Rozière *i sur.*, „Code Llama: Open foundation models for code“, *arXiv [cs.CL]*, 24. kolovoz 2023. [Na internetu]. Dostupno na: http://arxiv.org/abs/2308.12950

[44] „VOSviewer - Visualizing scientific landscapes“, VOSviewer. Pristupljeno: 22. lipanj 2025. [Na internetu]. Dostupno na: https://www.vosviewer.com/

[45] Q. Sun *i sur.*, „A survey of neural Code Intelligence: Paradigms, advances and beyond“, *arXiv [cs.SE]*, 21. ožujak 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2403.14734

[46] Z. Lin, Z. Gou, T. Liang, R. Luo, H. Liu, i Y. Yang, „CriticBench: Benchmarking LLMs for Critique-Correct Reasoning“, *arXiv [cs.CL]*, 22. veljača 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2402.14809

[47] R. Li *i sur.*, „StarCoder: may the source be with you!“, *arXiv [cs.CL]*, 09. svibanj 2023. [Na internetu]. Dostupno na: http://arxiv.org/abs/2305.06161

[48] B. LaBash, A. Rosedale, A. Reents, L. Negritto, i C. Wiel, „RES-Q: Evaluating code-editing large language model systems at the repository scale“, *arXiv [cs.CL]*, 24. lipanj 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2406.16801

[49] W. Zhou, M. Mesgar, H. Adel, i A. Friedrich, „FREB-TQA: A fine-grained robustness evaluation benchmark for table Question Answering“, *arXiv [cs.CL]*, 29. travanj 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2404.18585

[50] Q. Peng, Y. Chai, i X. Li, „HumanEval-XL: A multilingual code generation benchmark for cross-lingual natural language generalization“, *arXiv [cs.CL]*, 26. veljača 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2402.16694

[51] F. F. Xu, Z. Jiang, P. Yin, B. Vasilescu, i G. Neubig, „Incorporating external knowledge through pre-training for natural language to code generation“, *arXiv [cs.CL]*, 19. travanj 2020. [Na internetu]. Dostupno na: http://arxiv.org/abs/2004.09015

[52] H. Touvron *i sur.*, „Llama 2: Open foundation and fine-tuned chat models“, *arXiv [cs.CL]*, 18. srpanj 2023. [Na internetu]. Dostupno na: http://arxiv.org/abs/2307.09288

[53] H. Touvron *i sur.*, „LLaMA: Open and efficient foundation language models“, *arXiv [cs.CL]*, 27. veljača 2023. [Na internetu]. Dostupno na: http://arxiv.org/abs/2302.13971

[54] P. Haller, J. Golde, i A. Akbik, „PECC: Problem Extraction and Coding Challenges“, *arXiv [cs.AI]*, 29. travanj 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2404.18766

[55] V. Viswanathan, C. Zhao, A. Bertsch, T. Wu, i G. Neubig, „Prompt2Model: Generating deployable models from natural language instructions“, *arXiv [cs.CL]*, 23. kolovoz 2023. [Na internetu]. Dostupno na: http://arxiv.org/abs/2308.12261
[56] Y. Zhang, Q. Jiang, X. Han, N. Chen, Y. Yang, i K. Ren, „Benchmarking Data Science Agents“, *arXiv [cs.AI]*, 26. veljača 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2402.17168
[57] R. Li *i sur.*, „TACO: Topics in Algorithmic COde generation dataset“, *arXiv [cs.AI]*, 22. prosinac 2023. [Na internetu]. Dostupno na: http://arxiv.org/abs/2312.14852
[58] G. Morello, M. Eshghie, S. Bobadilla, i M. Monperrus, „DISL: Fueling research with A large dataset of Solidity smart contracts“, *arXiv [cs.SE]*, 25. ožujak 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2403.16861
[59] Y. Lai *i sur.*, „DS-1000: A natural and reliable benchmark for data science code generation“, *ICML*, sv. 202, str. 18319–18345, stu. 2022, doi: 10.48550/arXiv.2211.11501.
[60] L. Zhang, Y. Zhang, K. Ren, D. Li, i Y. Yang, „MLCopilot: Unleashing the power of large language models in solving machine learning tasks“, *arXiv [cs.LG]*, 28. travanj 2023. [Na internetu]. Dostupno na: http://arxiv.org/abs/2304.14979
[61] Y. Cui, „A case study of web app coding with OpenAI reasoning models“, *arXiv [cs.SE]*, 19. rujan 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2409.13773
[62] P. Liguori, E. Al-Hossami, D. Cotroneo, R. Natella, B. Cukic, i S. Shaikh, „Can we generate shellcodes via natural language? An empirical study“, *Autom. Softw. Eng.*, sv. 29, izd. 1, svi. 2022, doi: 10.1007/s10515-022-00331-3.
[63] Z. Luo *i sur.*, „WizardCoder: Empowering code Large Language Models with Evol-Instruct“, *arXiv [cs.CL]*, 14. lipanj 2023. [Na internetu]. Dostupno na: http://arxiv.org/abs/2306.08568
[64] Y. Cui, „Insights from benchmarking frontier language models on web app code generation“, *arXiv [cs.SE]*, 08. rujan 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2409.05177
[65] S. Honarvar, M. van der Wilk, i A. Donaldson, „Turbulence: Systematically and automatically testing instruction-tuned large language models for code“, *arXiv [cs.SE]*, 22. prosinac 2023. [Na internetu]. Dostupno na: http://arxiv.org/abs/2312.14856
[66] D. Guo *i sur.*, „DeepSeek-Coder: When the large language model meets programming -- the rise of code intelligence“, *arXiv [cs.SE]*, 25. siječanj 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2401.14196
[67] K. Li, Y. Tian, Q. Hu, Z. Luo, Z. Huang, i J. Ma, „MMCode: Benchmarking multimodal large language models for code generation with visually rich programming problems“, *arXiv [cs.CL]*, 15. travanj 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2404.09486
[68] Y. Xu *i sur.*, „CloudEval-YAML: A Practical Benchmark for Cloud Configuration Generation“, *Proceedings of Machine Learning and Systems*, sv. 6, str. 173–195, 2024, [Na internetu]. Dostupno na: https://proceedings.mlsys.org/paper_files/paper/2024/hash/554e056fe2b6d9fd27ffcd3367ae1267-Abstract-Conference.html
[69] M. C. Tol, B. Gulmezoglu, K. Yurtseven, i B. Sunar, „FastSpec: Scalable generation and detection of spectre gadgets using neural embeddings“, u *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, ruj. 2021, str. 616–632. doi: 10.1109/eurosp51992.2021.00047.
[70] Y. Wei *i sur.*, „SelfCodeAlign: Self-Alignment for Code Generation“, *arXiv [cs.CL]*, 31. listopad 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2410.24198
[71] Z. Ma, H. Guo, Y.-J. Gong, J. Zhang, i K. C. Tan, „Toward automated algorithm design: A survey and practical guide to Meta-Black-Box-optimization“, *arXiv [cs.NE]*, 01. studeni 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2411.00625

[72] Q. Zheng *i sur.*, „CodeGeeX: A pre-trained model for code generation with multilingual benchmarking on HumanEval-X“, u *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, New York, NY, USA: ACM, kol. 2023. doi: 10.1145/3580305.3599790.
[73] Y. Wang, W. Wang, S. Joty, i S. C. H. Hoi, „CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation“, *arXiv [cs.CL]*, 02. rujan 2021. [Na internetu]. Dostupno na: http://arxiv.org/abs/2109.00859
[74] M. L. Siddiq, B. K. Casey, i J. C. S. Santos, „Franc: A lightweight framework for high-quality code generation“, *IEEE Work Conf Source Code Anal Manip*, str. 106–117, srp. 2023, doi: 10.1109/SCAM63643.2024.00020.
[75] J. Zhang *i sur.*, „When LLMs meet cybersecurity: A systematic literature review“, *arXiv [cs.CR]*, 06. svibanj 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2405.03644
[76] Y. Chen *i sur.*, „Security of language models for code: A systematic literature review“, *arXiv [cs.SE]*, 21. listopad 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2410.15631
[77] J. Chen *i sur.*, „RMCBench: Benchmarking Large Language Models’ resistance to malicious code“, *arXiv [cs.SE]*, 23. rujan 2024. doi: 10.1145/3691620.3695480.
[78] A. Storhaug, J. Li, i T. Hu, „Efficient avoidance of vulnerabilities in auto-completed smart contract code using vulnerability-constrained decoding“, u *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, IEEE, lis. 2023, str. 683–693. doi: 10.1109/issre59848.2023.00035.
[79] Z. Zhang *i sur.*, „Unifying the perspectives of NLP and software engineering: A survey on language models for code“, *arXiv [cs.CL]*, 14. studeni 2023. [Na internetu]. Dostupno na: https://simg.baai.ac.cn/paperfile/e3d3c624-dc64-4fa1-9bdd-f67f2964781b.pdf
[80] Y. Chai, S. Wang, C. Pang, Y. Sun, H. Tian, i H. Wu, „ERNIE-code: Beyond English-centric cross-lingual pretraining for programming languages“, *arXiv [cs.CL]*, 13. prosinac 2022. [Na internetu]. Dostupno na: http://arxiv.org/abs/2212.06742
[81] N. Lambert *i sur.*, „TÜLU 3: Pushing frontiers in open language model post-training“, *arXiv [cs.CL]*, 22. studeni 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2411.15124
[82] I. Paul, G. Glavaš, i I. Gurevych, „IRCoder: Intermediate representations make language models robust multilingual code generators“, *arXiv [cs.AI]*, 06. ožujak 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2403.03894
[83] M. Bahrami *i sur.*, „PyTorrent: A Python library corpus for large-scale language models“, *arXiv [cs.SE]*, 04. listopad 2021. [Na internetu]. Dostupno na: http://arxiv.org/abs/2110.01710
[84] B. Petryshyn i M. Lukoševičius, „Optimizing Large Language Models for OpenAPI code completion“, *arXiv [cs.SE]*, 24. svibanj 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2405.15729
[85] B. Berabi, J. He, V. Raychev, i M. Vechev, „TFix: Learning to Fix Coding Errors with a Text-to-Text Transformer“, u *Proceedings of the 38th International Conference on Machine Learning*, M. Meila i T. Zhang, Ur., u Proceedings of Machine Learning Research, vol. 139. PMLR, 18--24 Jul 2021, str. 780–791. [Na internetu]. Dostupno na: https://proceedings.mlr.press/v139/berabi21a.html
[86] B. Hui *i sur.*, „Qwen2.5-Coder Technical Report“, *arXiv [cs.CL]*, 18. rujan 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/2409.12186
[87] M. R. Parvez, W. Ahmad, S. Chakraborty, B. Ray, i K.-W. Chang, „Retrieval augmented code generation and summarization“, u *Findings of the Association for Computational Linguistics: EMNLP 2021*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.findings-emnlp.232.
[88] S. L. Shrestha i C. Csallner, „SLGPT: Using transfer learning to directly generate Simulink model files and find bugs in the Simulink toolchain“, u *Evaluation and Assessment in*

Software Engineering, New York, NY, USA: ACM, lip. 2021. doi: 10.1145/3463274.3463806.
[89] A. Vaswani *i sur.*, „Attention is All you Need“, *Neural Inf Process Syst*, str. 5998–6008, lip. 2017, [Na internetu]. Dostupno na: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html
[90] Z. Chen, M. M. Balan, i K. Brown, „Language models are few-shot learners for prognostic prediction“, *arXiv [cs.CL]*, 24. veljača 2023. [Na internetu]. Dostupno na: http://arxiv.org/abs/2302.12692
[91] J. Austin *i sur.*, „Program synthesis with large language models“, *arXiv [cs.PL]*, 15. kolovoz 2021. [Na internetu]. Dostupno na: https://www.research.google/pubs/pub50670/
[92] I. Moutidis i H. T. P. Williams, „Community evolution on Stack Overflow“, *PLoS One*, sv. 16, izd. 6, str. e0253010, lip. 2021, doi: 10.1371/journal.pone.0253010.
[93] „How to convert Decimal to Double in C#?“, Stack Overflow. Pristupljeno: 15. travanj 2025. [Na internetu]. Dostupno na: https://stackoverflow.com/questions/4/how-to-convert-decimal-to-double-in-c
[94] U. Z. Ahmed, R. Sindhgatta, N. Srivastava, i A. Karkare, „Targeted example generation for compilation errors“, u *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, stu. 2019, str. 327–338. doi: 10.1109/ase.2019.00039.
[95] „Why did the width collapse in the percentage width child element in an absolutely positioned parent on Internet Explorer 7?“, Stack Overflow. Pristupljeno: 15. travanj 2025. [Na internetu]. Dostupno na: https://stackoverflow.com/questions/6/why-did-the-width-collapse-in-the-percentage-width-child-element-in-an-absolutel
[96] N. A. Ikumapayi, „Automated front-end code generation using OpenAI: Empowering web development efficiency“, *SSRN Electron. J.*, 2023, doi: 10.2139/ssrn.4590704.
[97] „How do I calculate someone’s age based on a DateTime type birthday?“, Stack Overflow. Pristupljeno: 15. travanj 2025. [Na internetu]. Dostupno na: https://stackoverflow.com/questions/9/how-do-i-calculate-someones-age-based-on-a-datetime-type-birthday
[98] A. Shachar, „Introduction to Algogens“, 04. ožujak 2024. doi: 10.31219/osf.io/r2e6c.
[99] „Calculate relative time in C#“, Stack Overflow. Pristupljeno: 15. travanj 2025. [Na internetu]. Dostupno na: https://stackoverflow.com/questions/11/calculate-relative-time-in-c-sharp
[100] „Filling a DataSet or a DataTable from a LINQ query result set“, Stack Overflow. Pristupljeno: 15. travanj 2025. [Na internetu]. Dostupno na: https://stackoverflow.com/questions/16/filling-a-dataset-or-a-datatable-from-a-linq-query-result-set
[101] „Throw an error preventing a table update in a MySQL trigger“, Stack Overflow. Pristupljeno: 15. travanj 2025. [Na internetu]. Dostupno na: https://stackoverflow.com/questions/24/throw-an-error-preventing-a-table-update-in-a-mysql-trigger
[102] „How to unload a ByteArray using Actionscript 3?“, Stack Overflow. Pristupljeno: 15. travanj 2025. [Na internetu]. Dostupno na: https://stackoverflow.com/questions/34/how-to-unload-a-bytearray-using-actionscript-3
[103] Z. Ma, S. An, B. Xie, i Z. Lin, „Compositional API recommendation for library-oriented code generation“, u *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*, New York, NY, USA: ACM, tra. 2024, str. 87–98. doi: 10.1145/3643916.3644403.
[104] „Determine a user’s timezone“, Stack Overflow. Pristupljeno: 15. travanj 2025. [Na internetu]. Dostupno na: https://stackoverflow.com/questions/13/determine-a-users-timezone

[105] M. Vaziri, L. Mandel, A. Shinnar, J. Siméon, i M. Hirzel, „Generating chat bots from web API specifications“, u *Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, New York, NY, USA: ACM, lis. 2017. doi: 10.1145/3133850.3133864.

[106] „Difference between math.Floor() and math.Truncate()“, Stack Overflow. Pristupljeno: 15. travanj 2025. [Na internetu]. Dostupno na: https://stackoverflow.com/questions/14/difference-between-math-floor-and-math-truncate

[107] „How to use the C socket API in C++ on z/OS“, Stack Overflow. Pristupljeno: 15. travanj 2025. [Na internetu]. Dostupno na: https://stackoverflow.com/questions/25/how-to-use-the-c-socket-api-in-c-on-z-os

[108] A. H. Mohammadkhani, C. Tantithamthavorn, i H. Hemmatif, „Explaining transformer-based code models: What do they learn? When they do not work?“, u *2023 IEEE 23rd International Working Conference on Source Code Analysis and Manipulation (SCAM)*, IEEE, 2023, str. 96–106.

[109] „Binary data in MySQL“, Stack Overflow. Pristupljeno: 15. travanj 2025. [Na internetu]. Dostupno na: https://stackoverflow.com/questions/17/binary-data-in-mysql

[110] K. Prakash, S. Rao, R. Hamza, J. Lukich, V. Chaudhari, i A. Nandi, „Integrating LLMs into database systems education“, u *Proceedings of the 3rd International Workshop on Data Systems Education: Bridging education practice with education research*, New York, NY, USA: ACM, lip. 2024, str. 33–39. doi: 10.1145/3663649.3664371.

[111] Pristupljeno: 15. travanj 2025. [Na internetu]. Dostupno na: https://stackoverflow.com/questions/19/what-is-the-fastest-way-to-get-the-value-of-π

[112] D. Cambaz i X. Zhang, „Use of AI-driven code generation models in teaching and learning programming: A systematic literature review“, u *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, New York, NY, USA: ACM, ožu. 2024, str. 172–178. doi: 10.1145/3626252.3630958.

[113] H. Mahmood, A. A. Jilani, i A. Rauf, „Code Swarm: A code generation tool based on the automatic derivation of transformation rule set“, *arXiv [cs.SE]*, 03. prosinac 2023. [Na internetu]. Dostupno na: http://arxiv.org/abs/2312.01524

[114] B. Esselink, „The evolution of localization“, *The Guide from Multilingual Computing & Technology: Localization*, sv. 14, izd. 5, str. 4–7, 2003, [Na internetu]. Dostupno na: https://www.intercultural.urv.cat/media/upload/domain_317/arxius/Technology/Esselink_Evolution.pdf

[115] *GitHub Copilot · Your AI pair programmer*. Github. Pristupljeno: 16. travanj 2025. [Na internetu]. Dostupno na: https://github.com/features/copilot

[116] „Before you continue“. Pristupljeno: 16. travanj 2025. [Na internetu]. Dostupno na: https://gemini.google.com/

[117] B. A. Becker, P. Denny, J. Finnie-Ansley, A. Luxton-Reilly, J. Prather, i E. A. Santos, „Programming is hard - or at least it used to be“, u *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, New York, NY, USA: ACM, ožu. 2023, str. 500–506. doi: 10.1145/3545945.3569759.

[118] „What is open?“ Pristupljeno: 16. travanj 2025. [Na internetu]. Dostupno na: https://okfn.org/en/library/what-is-open/

[119] „Papers with code - Code Generation“. Pristupljeno: 16. travanj 2025. [Na internetu]. Dostupno na: https://paperswithcode.com/task/code-generation

[120] T. Hollanek, „AI transparency: a matter of reconciling design with critique“, *AI Soc.*, sv. 38, izd. 5, str. 2071–2079, lis. 2023, doi: 10.1007/s00146-020-01110-y.

POPIS OZNAKA I KRATICA

AI	*Artificial Intelligence*
API	*Application Programming Interface*
APPS	*Automated Programming Progress Standard*
CIL	*Common Intermediate Language*
CMS	*Content Management System*
CoNaLa	*CMU CoNaLa, the Code/Natural Language Challenge*
CoNaLa-Ext	*CoNaLa Extended With Question Text*
GP	*Genetic Programming*
ICPC	*International Collegiate Programming Competition*
IDE	*Integrated Development Environment*
IOI	*International Olympiad in Informatics*
KMS	*Knowledge Management System*
LLM	*Large Language Model*
LM	*Language Model*
LMS	*Learning Management System*
LSTM	*Long Short Term Memory*
MBPP	*Mostly Basic Python Programming*
MCoNaLa	*Multilingual CoNaLa*
MLE	*Maximum Likelihood Estimate*
PECC	*Problem Extraction and Coding Challenge*
RAG	*Retrieval-Augmented Generation*
RNN	*Recurrent Neural Network*
SAFIM	*Syntax-Aware Fill-In-the-Middle*
TACO-Code	*Topics in Algorithmic COde generation dataset*
UML	*Unified Modeling Language*

SAŽETAK

U radu su analizirane i vrednovane tehnike za automatsko generiranje programskog kôda s fokusom na generativne tehnike strojnog učenja. Pregledom literature utvrdili smo kako su namjerno ili slučajno nastali jazovi između pristupa istraživanju i razvoju sustava i modela vođenih umjetnom inteligencijom te zbog toga ključan izazov predstavlja ograničena iskoristivost takvih sustava. Kao glavni uzrok ovog fenomena prepoznaje se postojanje tri smjera u kojima je odvedeno istraživanje ove problematike, a to grananje zamijećeno je još davnih 80-ih godina prošlog stoljeća, davno prije nego li su sustavi potpomognuti velikim jezičnim modelima počeli dominirati područjem umjetne inteligencije. Ovim radom obuhvaćene su brojne tehnike upravljanja znanjem poput vizualizacije stabla odluke, tehnike upravljanja strojnim učenjem poput finog podešavanja i vrednovanja korištenjem jediničnih testova, ali i tehnike upravljanja podacima poput samoorganizacije koje se koriste odvojeno ne bi li se poboljšali rezultati dobiveni vrednovanjem uvježbanih AI modela, no rijetko u tandemu. Skupnim korištenjem ovakvih tehnika te oblikovanjem hibridnog AI sustava na temelju troslojne arhitekture nadahnute postojećim arhitekturama sustava za učenje programiranja, smatramo kako će biti moguće kontrolirati simbiozu koja bi nastala u takvom višeagentskom sustavu. Osim tehnika strojnog učenja, prikazane su i druge alternativne tehnike iz područja umjetne inteligencije poput programiranja jatom i genetskog (evolucijskog) programiranja koje nude rješenja u slučajevima kada ne postoji dostatno velik podatkovni skup za uvježbavanje modela ili kada je postojanje rješenja neizvjesno, ali i dalje potrebno. Zbog višeoblične prirode podataka koji bivaju generirani u procesu razvoja programske podrške, ne postoji jednoliko rješenje u pogledu AI sustava i modela te je potrebno osim učenja oblikovati i druge karakteristike bioloških sustava poput samoorganizacije, samoizlječivosti te društvene interaktivnosti ne bi li se povećala primjenjivost i pouzdanost ovakvih sustava u različitim domenama, uvjetima te okruženjima. Time bi se postigla bolja usklađenost s korisničkim zahtjevima poput transparentnosti, opće svrhovitosti i prirodnosti AI sustava, a između čovjeka i računala uspostavilo bi se simbiotsko asistentsko suradničko okruženje. Daljnje istraživanje usmjereno je na razvoj AI arhitektura za upravljanje sadržajem kao istraživačkog instrumenta koji bi kôd tretirao kao semantički, izvršivi sadržaj s formalnim podrijetlom. Time bismo omogućili sintezu programskog kôda na temelju bihevioralnih specifikacija umjesto sintaktičkih uzoraka. Primjenu vidimo u sustavima gdje je potpuna revizibilnost od važnosti jer je sigurnost od kritične važnosti (npr. kod programiranja samovozećih automobila ili sustava za liječenje ljudi), ali i sustavima gdje je potrebna suglasnost s određenim regulativama poput mrežne pristupačnosti.