Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004:007
DOI: https://doi.org/10.24867/22OI01Katic

PRIMENA GITHUB COPILOTA U RAZVOJU VEB APLIKACIJE

USE OF GITHUB COPILOT IN BUILDING A WEB APPLICATION

Marko Katić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – INŽENJERSTVO INFORMACIONIH SISTEMA

Kratak sadržaj – *Programi za rekomendaciju koda postaju sve popularniji, dok je automatsko generisanje programa već odavno velika želja u softverskom inženjerstvu. U poslednje vreme najveću pažnju je privuklo deep learning rešenje po imenu* Github *Copilot, napravljen od strane OpenAI i podršku Microsoft-a. U ovom radu analiziramo upotrebu softvera za rekomendaciju koda prilikom kreiranja višeslojne veb aplikacije.*

Ključne reči: *mašinsko učenje, duboko učenje, Github Copilot*

Abstract – *Code recommendation programs are becoming increasingly popular, while automatic program generation has long been a desire in software engineering. Recently, a deep learning solution called Github Copilot, created by OpenAI and supported by Microsoft, has attracted the most attention. In this paper, we analyze the use of code recommendation software when creating a web application.*

Keywords*: machine learning, deep learning, Github Copilot*

1. UVOD

Duboko učenje je već neko vreme izaziva veliku popularnost, osim što ima veliki obim primene u različitim naukama i obično je pružio neverovatno tačne rezultate istraživačima. Duboko učenje je deo mašinskog učenja, iz toga razloga će prvo biti predstavljene osnove mašinskog učenja, razlike između nadgledanog i nenadgledanog učenja i neke njihove primere. Nakon toga počinjemo da proučavamo različite vrste neuronskih mreža i počinjemo da pričamo o dubokom učenju. Glavna svrha teorijskog poglavlja je da pruži osnovne informacije o različitim procesima učenja i da pokaže teorije iza različitih metoda koje se kasnije primenjuju u nekim aplikacijama.

Modeli dubokog učenja su naširoko korišćeni prilikom razvoja softvera za rekomendaciju koda, kada su jednom pravilno obučeni, mogu uzeti kao ulaz nekompletnu komponentu koda (npr. nepotpunu funkciju) i predvideti nedostajući kod da bi je finalizovali. *GitHub Copilot* je primer preporuke koda napravljenog obučavanjem modela dubokog učenja (engl. *Deep Learning*, DL) kod deluje kao podaci za treniranje, omogućavajući modelu da nauči „kako da programira“. Upotreba takvog koda je obično regulisana licencama za besplatni softver i softver otvorenog koda (engl. *Free and Open Source Software*, FOSS), koje određuju pod kojim uslovima se kod može ponovo distribuirati ili modifikovati. Do danas je još nejasno je da li kod generisan od DL modela obučenih na otvorenom kodu treba smatrati „novim” ili kao rezultat dobijenim radom DL modela. Ako ovi modeli imaju tendenciju da generišu novi kod koji se ne vidi u skupu za treniranje, onda je malo verovatno da će se pojaviti problemi sa licenciranjem.

U praktičnom delu rada biće prikazana primena *Github Copilota* u razvoju višeslojne aplikacije koju će činiti *NestJS* na *backend*-u i *ReactJS* na *frontend*-u.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Andraš Anderla, vanr.prof.

2. VEŠTAČKA INTELIGENCIJA

Veliko interesovanje za veštačku inteligenciju (engl. *Artificial intelligence*, AI) nije se pojavilo sve donedavno, a glavni razlog za to je velika količina podataka koja je potrebna za obradu. Na primer, da bi neuronska mreža mogla da klasifikuje slike, potrebno je obezbediti hiljade slika za treniranje mreže, dok u slučaju teksta, za zadovoljavajući rezultat može biti potreban ekvivalent od hiljadu knjiga. Drugim rečima - ograničena količina računarske snage i memorije potrebne za obradu ogromnih količina podataka donedavno je sprečavala veliku primenu veštačke inteligencije.

Danas je veštačka inteligencija svuda oko nas i svako ima svoje mišljenje o njoj. Kompanije i institucije se prilagođavaju za tranziciju i sve veću primenu veštačke inteligencije, zabrinuti glasovi su postali sve glasniji o tome kako će veštačka inteligencija uticati na tržište rada, takođe se raspravlja o složenim etičkim pitanjima, a sa dramatičnije strane da će veštačka inteligencija doprineti propasti čovečanstva. U suštini veštačka inteligencija je postala popularna reč koja ima tendenciju da postanu zaboravljene u određenom trenutku, zbog čega je važno naglasiti na oblasti i primenu u kojima se veštačka inteligencija ističe i gde doprinosi daljem razvoju.

Veštačka inteligencija ima nekoliko konotacija, a jedan od mnogih načina da se pristupi razumevanju veštačke inteligencije jeste da se podeli na usku veštačku inteligenciju (engl. *Artificial Narrow Intelligence*, ANI), opštu veštačku inteligenciju (engl. *Artificial General Intelligence*, AGI) i super veštačku inteligenciju (engl. *Artificial Super Intelligence*) [1].

2.1. Uska veštačka inteligencija

Uska veštačka inteligencija se takođe naziva slaba veštačka inteligencija (engl. *Weak AI*), ova vrsta veštačke inteligencije je ona koja se prvenstveno fokusira na jedan uski zadatak, sa ograničenim spektrom sposobnosti i jedina je vrsta veštačke inteligencije koja je do danas uspešno realizovana. Uska veštačka inteligencija je

orijentisana ka cilju, dizajnirana je za obavljanje pojedinačnih zadataka - npr. prepoznavanje lica, prepoznavanje govora/glasovni asistenti, vožnja automobila ili pretraživanje interneta - i veoma je inteligentana u ispunjavanju specifičnog zadatka za koji je programirana. Uska veštačka inteligencija ne oponaša niti pokušava da replicira ljudsku inteligenciju, već samo simulira ljudsko ponašanje zasnovano na uskom rasponu parametara i konteksta [6]. Uska veštačka inteligencija je doživela brojna otkrića u poslednjoj deceniji, zasnovana na dostignućima u mašinskom učenju i dubokom učenju. Na primer, sistemi veštačke inteligencije se danas koriste u medicini za dijagnostikovanje raka i drugih bolesti sa izuzetnom preciznošću.

2.2. Opšta veštačka inteligencija

Opšta veštačka inteligencija takođe poznata kao jaka veštačka inteligencija (engl. *strong* AI) ili duboka veštačka inteligencija (engl. *deep* AI) je koncept mašine sa opštom inteligencijom koja oponaša ljudsku inteligenciju i/ili ponašanje, sa sposobnošću da uči i primeni svoju inteligenciju za rešavanje bilo kog problema. Opšta veštačka inteligencija može misliti, razumeti i delovati na način koji se ne razlikuje od ljudskog u bilo kojoj situaciji. Istraživači i naučnici veštačke inteligencije još uvek nisu postigli AGI, da bi uspeli morali bi da pronađu način da naprave svesne mašine i programirajući čitav niz kognitivnih sposobnosti. Mašine bi morale da podignu iskustveno učenje na viši nivo i ne samo da poboljšaju efikasnost na pojedinačnim zadacima nego da steknu sposobnost primene iskustveno znanje na širi spektar različitih problema. Opšta veštačka inteligencija koristi teoriju AI okvira uma koja se odnosi na sposobnost da se rasaznaju potrebe, emocije, uverenja i misaoni procesi drugih inteligentnih bića [6].

2.3. Super veštačka inteligencija

Super veštačka inteligencija je hipotetička veštačka inteligencija koja niti oponaša niti razume ljudsku inteligenciju i ponašanje. Super veštačka inteligencija je mesto gde mašine postaju samosvesne i prevazilaze kapacitet ljudske inteligencije i sposobnosti. Koncept veštačke superinteligencije vidi da AI evoluira tako da bude toliko slična ljudskim emocijama i iskustvima, da ih ne samo razume već i izaziva sopstvene emocije, potrebe, uverenja i želje. Super veštačka inteligencija bi imao veću memoriju i bržu sposobnost obrade i analize podataka i stimulusa tako da sposobnosti superinteligentnih bića u donošenju odluka i rešavanju problema bile bi daleko superiornije od ljudskih bića.

3. MAŠINSKO UČENJE

Machine learning je jedan od pristupa dostizanju veštačke inteligencije korišćenjem metoda za automatsko učenje iz podataka bez eksplicitnog programiranja konkretne stvari. Dok je *deep learning* u stanju da predvidi ishode na osnovu sličnih podataka, a potom i da se poboljša iz sopstvenog iskustva.

Mašinsko učenje je potpolje unutar veštačke inteligencije koje se odnosi na kompjuterske programe koji mogu da uče, razmišljaju i deluju na osnovu podataka. Algoritam za mašinsko učenje je računarski program zasnovan na matematičkom modelu koji opisuje odnos između uključenih varijabli iz posmatranih podataka na kojima je algoritam obučen, takođe poznat kao podaci za treniranje. Na osnovu obrasca podataka o treniranju, algoritam mašinskog učenja može da predviđa i donosi odluke za nove nevidljive podatke na osnovu statistike [2].

Algoritmi mašinskog učebha se obično klasifikuju u tri osnovne kategorije: nadgledano učenje (eng. *Supervised learning*), nenadgledano učenje (engl. Unsupervised learning) i pojačano učenje učenje (engl. *Reinforcement learning*).

U slučaju nadgledanog učenja algoritmu ML izlaz je već poznat, tako što se radi mapiranje ulaza sa izlazom. Stoga, da bi se kreirao model mašina se napaja sa puno ulaznih podataka za treniranje (sa poznatim ulazom i odgovarajućim izlazom). Podaci za treniranje pomažu u postizanju nivoa tačnosti za kreirani model podataka. Izgrađeni model se zatim koristi za novi skup podataka za predviđanje ishoda. Neki od vrsta algoritama nadgledanog učenja su: linearna regresija (engl. *Linear regression*) i klasifikacija (engl. *Classification*).

Učenje bez nadzora odvija se bez pomoći nadzornika tj. to je nezavisan proces učenja. U ovom modelu, s obzirom da ne postoji izlaz mapiran sa ulazom, ciljne vrednosti su nepoznate ili neobeležene zato sistem mora sam da uči od unosa podataka na njega i detektuje skrivene obrasce.

U pojačanom učenju algoritam uči pomoću mehanizma povratnih informacija i prošlih iskustava. Dakle, kad god treba preduzeti sledeći korak, on dobija povratne informacije iz prethodnog koraka, zajedno sa iskustvom iz iskustva da predvidi šta bi mogao biti sledeći najbolji korak. Ovaj proces se naziva i postupak pokušaja i grešaka radi postizanja cilja. Učenje s pojačanjem je dugotrajni ponavljajući proces. Što je više povratnih informacija, sistem postaje tačniji [2].

3.1. Duboko učenje

Deep learning je moćan oblik mašinskog učenja koji omogućava računarima da rešavaju perceptivne probleme kao što su prepoznavanje slike i govora, sve više ulazi u biološke nauke, takođe se može reći da predstavlja kompoziciju jednostavnih koncepata za izgradnju složenih koncepata [5]. Ove metode dubokog učenja, kao što su duboke veštačke neuronske mreže, koriste više slojeva obrade za otkrivanje obrazaca i strukture u veoma velikim skupovima podataka. Svaki sloj uči koncept iz podataka na kojima se nadovezuju sledeći slojevi; što je nivo viši, to su pojmovi koji se uče apstraktniji. Deep learning ne zavisi od prethodne obrade podataka i automatski izdvaja karakteristike.

3.2. Neuronske mreže

Deep learning se generalno bavi neuronskim mrežama tj. tehnologija inspirisanim načinom na koji mozak radi sa svojom biološkom neuronskom mrežom, i kao takvi neuronske mreže imaju za cilj da oponašaju kako mozak pronalazi obrasce ili odnose među ogromnim količinama podataka [3]. Svaki neuron u mozgu je u stanju da izvrši neki jednostavan zadatak kao odgovor na neki unos, tako da povezivanjem više neurona omogućava realizaciju složenijih zadataka, kao što su na primer prepoznavanje govora ili prepoznavanje obrazaca na slikama. Za komunikaciju između neurona se koriste elektrohemijski signali, ako primljeni ukupni signal dostigne prag u neuronu, neuron počinje da se aktivira, što se odnosi na prenošenje signala susednom (koji se nazivaju povezani)

neuronima. Pretpostavlja se da promena jačine veze između dva neurona na osnovu prethodnog iskustva predstavlja osnovu pamćenja u mozgu [3].

3.3. Github Copilot

Za rad *Copilot*-a je zaduzen transformer, vrsta modela *deep learning*-a koji se koristi u velikim jezičkim modelima kao što su GPT-3 i *LaMDA*. Transformer-i su posebno dobri u obradi sekvencijalnih podataka kao što su tekst ili softverski kod. Na osnovu ulaznih podataka model transformer-a može predvideti sledeće elemente niza, bilo da su to reči ili kompjuterski kod. *Copilot* je izgrađen na *OpenAI*-ovom *Codex*-u, transformer koji je obučen na desetinama miliona *Github* repozitorijuma koda. Kada se *Github Copilot* instalira unutar IDE, *Copilot* daje predloge na osnovu postojećeg koda u unutar lokalnog repozitorijuma ili na znakove, kao što su imena funkcija i klasa ili komentara kodu [4].

Veoma je bitno napomenuti da *Copilot* ne razmišlja i ne kodira kao programer, ali pošto je bio izložen ogromnim količinama izvornog koda, može da pruži veoma dobre predloge koda, posebno za standardne i ponavljajuće zadatke koji se često javljaju u pisanju softvera.

4. IMPLEMENTACIJA SISTEMA

Veb aplikacije biće zadužena da podrži funkcionalnosti vođenja jedne *rent a car* agencije, potrebno je da podržava CRUD operacije nad glavnim modulima koji su:

- Korisnik.
- Automobil.
- Rezervacija.

Uz pomoč *Github Copilot*-a biće prikazan celokupni razvoj *NextJs* aplikacije, koja ce služiti kao REST API, podržavajući CRUD operacije. Aplikacija ce takođe podržavati JWT autorizaciju i biće povezana sa bazom podataka.

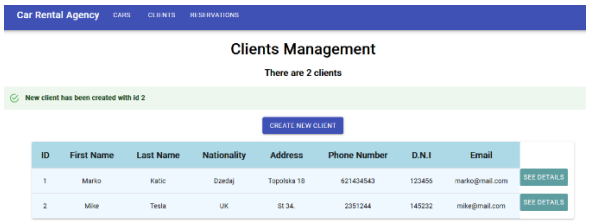
Svaka *NextJs* aplikacija mora imati jedan *root* modul, to je početni deo aplikacije iz kojeg se kasnije gradi aplikacija, svaki modul može importovati druge module i može sadržati kontrolere (engl. *controllers*) i servise (engl. *services*). Module klasa je označena anotacijom *@Module()* koji pomaže *NestJs* da bolje organizuje strukturu aplikacije, jer se uz pomoč modula aplikacija deli na manje komponente.

Kontroleri u *NestJs* su zaduženi za rukovanje dolazećih zahteva (engl. requests), validaciju parametara i vračanje odgovora klijentu. Kontroleri bi trebali da se održavaju jednostavnim i sa što manje koda unutar fajla. Servisi sadrže večinu funkcionalnosti aplikacije, svaka složena logika aplikacije je potrebno da prođe kroz servis.

Da kontroleri ne bi brinuli kako će nešta biti urađeno već šta, potrebno je kreirati servise koje će oni pozivati. Da bi se definisalo željeno ponašanje u *Nest*-u, prvo se definišu funkcije u repozitorijumu, oni govori kako bi nešta trebalo da radi tj. željeno ponašanje, i odvaja implementacionu logiku, dok se logika implementira u servisima koje korite repozitorijum.

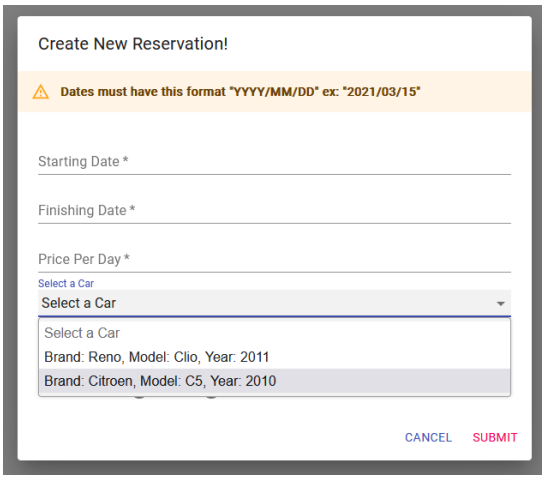
Taj mehanizam definisanja interfejsa odvojeno od klasa koje ih koriste naziva se umetanje zavisnosti (engl. *dependency injection*). Svaki kontroler će imati obaveznih prvih četri metoda za izvršavanje osnovnih CRUD funkcija i dobavljanje po jedinstvenom identifikatoru, dok će neki kontroleri zbog potreba razvijanja funkcionalnosti sistema imati dodatne metode.

Aplikacija podržava osnovne operacije upisa, izmene i brisanja. Ukoliko se želi uneti novi red u tabelu potrebno je kliknuti na plavo dugme iznad tabele, nakon toga će se pojaviti modalni dijalog sa formom za upisivanje podataka. Za tekstualne podatke koristiće se klasično polje za unos teksta, numeričke podatke pojaviće se polje koje prima samo numeričke vrednosti dok za datume pojaviće se kalendar preko kojeg biramo vrednosti.



Slika 1. *Prikaz uspešnog unosa novog klijenta i tabele*

Svako polje u zavisnosti od pravila vrši validaciju podataka, tako na primer ako se ne popune obavezna polja za obeležje koje mora da ima vrednosti ispisaće se odgovarajuća poruka kao što je prikazano na Slici 1. Kada se sva polja popune može se poslati zahtev za kreiranjem klikom na dugme podneti (engl. *submit*) koje se nalazi u donjem desnom uglu, takođe se može i prekinuti unos podataka klikom na dugme prekini (engl. *cancel*). U slučaju da se kreiraju nove rezervacije forma će automatski preuzeti aktuelne podatke iz baze i ubrzati proces kreiranja tako što će ponuditi trenutne slobodne automobile u opadajućem meniju.



Slika 2. *Primer automatskog povezivanja podataka prilikom kreiranja rezervacija*

4.1. Korišćenje Github Copilot-a

U toku izrade celokupne aplikacije unutar *Visual Studio Code* editora je bio omogučen *Github Copilo*t (potrebna prethodna autorizacija) koji je analizirao napisani kod i predlagao sugestije.

Najbitniji deo *Github Copliot*-a prilikom asistriranja pisanja koda jeste brzina, pogotovo za ponavljajuće delove koda.

Jedan od načina korišćenja *Copilot*-a jeste da se na opisan način definiše funkcionalnost koda koju želimo da *Copilot* izgeneriše unutar komentara, dok je drugi način tako što će *Copilot* automatski predlagati rešenja na osnovu delimično započetkog koda.

```
const handleDelete = () => {
  try{
    deleteClient(id);
    history.push('/clients');
  }
```

Slika 3. *Primer predloženog koda na osnovu prethodnog koda*

```
@Get('/:id')
async getCarById(@Param('id', ParseIntPipe) id: number)
  // return car by id

  return await this.carService.getCarById(id);

}
```

Slika 4. *Primer predloženog koda na osnovu komentara*

Github Copilot će uobičajno prikazivati predlog samo za narednu liniju ali ukoliko želimo da vidimo celokupnu funkciju kao i listu svih mogućih predloga, tu opciju uključujemo sa posebnom prečicom. Nakon toga će se u tabu pored otvoriti novi prozor sa filtiranim mogućim sugestijama kao što je prikazano na Slici 5.

```
GitHub Copilot 4
Synthesizing 10/10 solutions (Duplicates hidden)

=======

Accept Solution
async deleteClientById(id: number): Promise<void> {
    const result = await this.clientRepository.delete({ id });

    if (result.affected === 0) {
      throw new NotFoundException('Client not found');
    }
  }

=======

Accept Solution
async deleteClientById(id: number): Promise<void> {
    const result = await this.clientRepository.delete(id);

    if (result.affected === 0) {
      throw new NotFoundException('Client not found');
    }
  }

=======

Accept Solution
async deleteClientById(id: number): Promise<void> {
    const client = await this.clientRepository.findOne({ where: { id } });

    if (!client) {
      throw new NotFoundException('Client not found');
    }

    await this.clientRepository.delete({ id });
  }

=======

Accept Solution
async deleteClientById(id: number): Promise<void> {
    const client = await this.getClientById(id);
    await this.clientRepository.delete(client.id);
  }

=======

Accept Solution
async deleteClientById(id: number): Promise<void> {
    const client = await this.getClientById(id);
    await this.clientRepository.delete(client);
  }
```

Slika 5. *Prikaz liste sa celokupnim predlozima*

Zatim nakon što se izabere prihvatljivo rešenje potrebno je kliknuti na prihvati rešenje (engl. Accept Solution) dugme i nakon toga će se odabrani deo koda automatski pojaviti unutar fajla.

5. ZAKLJUČAK

Github Copilot je primer softvera za predikciju koda izgrađen treniranjem DL modela na milionima otvorenim repozitorijuma, kao takav ima mogučnost da predvidi nedostajuće delove koda i predloži kao rešenja. Korišćenje takvih setova podataka je uglavnom regulisano putem FOSS licence za otvoreni kod, ali se i dalje postavlja pitanje da li se kod generisan preko DL modela može smatrati u potpunosti novim ili kao "derivat" nečijeg tuđeg koda. Dok *Copilot* pokazuje svoje limite u predlozima za napredniji nivo programiranja tj. u zadacima koji su usko specijalizovani za određeni problem i projekat, u većini slučajeva je sposoban da predloži upotrebljiva rešenja za osnovne programerske zadatke. Takođe kvalitet generisanog koda od strane *Copilot*-a u velikoj meri zavisi od konciznosti i kvaliteta upita koji je prosledio programer. Često se dešava da *Copilot* ne uspeva da izgeneriše kod koji u potpunosti ispunjava sve zadate kriterijume, ali i takav kod može poslužiti programeru da uz male ručne modifikacije postane u potpunosti funkcionalan. Mora se upamtiti da je *Copilot* alat čija je glavna funkcionalnost da poveća efikasnost kod programera a ne da ga skroz zameni. Korisnici *Copilot*-a ne smeju u potpunosti da se oslone na predloženi kod, jer je on treniran na kodu koji bez sumnje ima dosta grešaka ili je prosto zastareo, kao takav je sklon različtim sigurnosnim rizicima. Opšti je zaključak da je *Github Copilot* nova tehnologija koja zahteva još dosta podataka za treniranje da bi se u potpunosti mogao da popuni mesto asistenta prilikom programiranja. Uzevši u obzir da je *Copilot* nedavno izašao iz faze zatvorenog testiranja i postao komercijalni proizvod koji će biti dostupan većem broju programera, to znači da će veličina podataka za treniranja biti značajno uvećana.

6. LITERATURA

[1] A. Ng, "Coursera," 2022. [Online]. Available: https://www.coursera.org/lecture/ai-for-everyone/week-1-introduction-SRwLN.

[2] T. O. Ayodele, "Types of Machine Learning Algorithms," 2010.

[3] R. Rojas, Neural networks: a systematic introduction, Springer Science & Business Media, 2013.

[4] Mark Chen, Jerry Tworek, "Evaluating Large Language Models Trained on Code," 2021

[5] Ian Goodfellow and Yoshua Bengio and Aaron Courville, Deep Learning, MIT Press, 2016.

[6] E. Escott, "CodeBots," 10 2017. [Online]. Available: https://codebots.com/artificial-intelligence/the-3-types-of-ai-is-the-third-even-possible.

Kratka biografija:

Marko Katić rođen je 1997. god. Diplomski rad na Fakultetu tehničkih nauka iz oblasti Inženjerstva informacionih sistema odbranio je 2020. godine, kada upisuje i master studije.

kontakt: markokatic97@gmail.com