

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1426

**SUSTAV ZA GENERIRANJE UPITA U JEZIKU SQL IZ
PRIRODNOG JEZIKA PRIMJENOM VELIKIH JEZIČNIH
MODELA**

Ivo Bralić

Zagreb, lipanj 2026.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1426

**SUSTAV ZA GENERIRANJE UPITA U JEZIKU SQL IZ
PRIRODNOG JEZIKA PRIMJENOM VELIKIH JEZIČNIH
MODELA**

Ivo Bralić

Zagreb, lipanj 2026.

DIPLOMSKI ZADATAK br. 1426

Pristupnik: **Ivo Bralić (0023133827)**

Studij: Računarstvo

Profil: Računarska znanost

Mentor: prof. dr. sc. Alan Jović

Zadatak: **Sustav za generiranje upita u jeziku SQL iz prirodnog jezika primjenom velikih jezičnih modela**

Opis zadatka:

Relacijske baze podataka pretražuju se korištenjem jezika SQL, što zahtijeva tehničko znanje korisnika. Razvojem velikih jezičnih modela (LLM) omogućeno je automatsko prevođenje upita iz prirodnog jezika u upite SQL-a. Takvi sustavi imaju potencijal pojednostaviti pristup podacima i povećati dostupnost informacijskih sustava širem krugu korisnika. U ovom diplomskom radu potrebno je razviti programski sustav koji korisniku omogućuje postavljanje upita nad relacijskom bazom podataka putem prirodnog jezika. Sustav treba dinamički dohvatiti shemu baze (tablice, attribute i relacije), konstruirati odgovarajući prompt te korištenjem LLM-a generirati upit u SQL-u. Generirani upit u SQL-u potrebno je validirati i sigurno izvršiti nad bazom, uz ograničavanje dozvoljenih operacija i zaštitu od sigurnosnih rizika (npr. SQL injection, neovlašteni pristup podacima). Programski sustav treba uključivati modul za analizu sheme baze, modul za komunikaciju s jezičnim modelom, mehanizam za validaciju SQL upita te korisničko sučelje za unos upita i prikaz rezultata. Sustav je potrebno ispitati na jednoj ili više relacijskih baza podataka te vrednovati njegovu točnost, robusnost i performanse na upitima različite složenosti (filtriranje, agregacije, operacije JOIN). Potrebno je proanalizirati i navesti praktične izazove implementacije i primjene te usporediti razvijeni sustav sa sustavima već predloženima u literaturi.

Rok za predaju rada: 26. lipnja 2026.

Zahvaljujem svome mentoru,
prof. dr. sc. Alanu Joviću,
na strpljivom vodstvu, korisnim savjetima i pomoći
u izradi ovog rada.

Sažetak

Naslov: Sustav za generiranje upita u jeziku SQL iz prirodnog jezika primjenom velikih jezičnih modela

Rad opisuje programski sustav koji korisničko pitanje prevodi u upit u SQL-u i pokreće ga nad relacijskom bazom. Shema baze dohvaća se dinamički, upit generira GPT-4o-mini, a prije izvršavanja prolazi kroz strukturalnu provjeru na razini apstraktnog sintaksnog stabla. Za slučajeve kad osnovni prolaz ne uspije, dodana je nadogradnja Cascade koja aktivira preciznije korake (fokusrano povezivanje sheme i glasovanje po više uzoraka). Upit se dodatno podešava iterativno: greške iz stvarnih pokretanja grupiraju se u kategorije, a za svaku se u upit dodaje kratko pravilo s primjerom. Sustav je evaluiran na skupu BIRD Mini-Dev, na uzorku od 200 pitanja koji pokriva svih jedanaest baza i sve tri razine težine. Postignuto je 47,0 % točnosti izvršavanja uz 100 % prolaznost validacije. Trošak jednog pokretanja iznosi oko dvadeset centi, što je približno pedeset puta manje od pristupa koji koriste GPT-4.

Ključne riječi: veliki jezični modeli, prirodni jezik, SQL, sigurnost baza podataka, apstraktno sintakšno stablo, BIRD, evaluacija, Cascade

Summary

Title: System for generating SQL queries from natural language using large language models

The thesis describes a software system that translates a natural-language question into an SQL query and runs it against a relational database. The database schema is fetched at runtime, the query is generated by GPT-4o-mini, and it goes through a structural check on its abstract syntax tree before execution. When the basic pass fails, a Cascade upgrade kicks in with a more targeted schema link and majority voting over several samples. The prompt is tuned iteratively: errors from actual runs are grouped into categories, and for each one a short rule and example are added to the prompt. Evaluation is done on BIRD Mini-Dev, on a 200-question sample covering all eleven databases and all three difficulty levels. The system reaches 47.0 % execution accuracy at 100 % validation. One 200-question run costs about twenty US cents, roughly fifty times less than GPT-4 based approaches.

Keywords: large language models, natural language, SQL, database security, abstract syntax tree, BIRD, evaluation, Cascade

Sadržaj

Sažetak.....	i
Summary.....	ii
Sadržaj	iii
Uvod	1
1. Teorijska podloga	3
1.1. Veliki jezični modeli	3
1.2. SQL i relacijske baze podataka	4
1.3. Tehnike upravljanja promptovima	5
2. Srodni radovi (Text-to-SQL).....	7
2.1. Rani pristupi	7
2.2. Pristupi bazirani na velikim jezičnim modelima	8
2.3. Referentni skupovi.....	10
2.4. Pristupi temeljeni na posebnome podešavanju modela	11
3. Arhitektura predloženog sustava	12
3.1. Pregled komponenti	12
3.2. Dinamičko dohvaćanje sheme	13
3.3. Konstrukcija prompta	13
3.4. Nadogradnja Cascade	16
3.5. Validator SQL upita	17
3.6. Izvršavanje upita i mehanizam ponavljanja.....	19
4. Implementacija	21
4.1. Korištene tehnologije.....	21
4.2. Modularna organizacija koda	21

4.3.	Kontejnerizacija i pokretanje.....	22
4.4.	Korisničko sučelje	22
5.	Evaluacija	25
5.1.	Mjerenje točnosti na skupu BIRD Mini-Dev	25
5.2.	Testiranje validatora	28
5.3.	Analiza latencije i troška	30
5.4.	Analiza pogrešaka.....	31
5.5.	Primjeri pogrešaka.....	32
6.	Rasprava	36
6.1.	Usporedba s literaturom.....	36
6.2.	Ograničenja rada.....	37
6.3.	Mogućnosti unaprjeđenja	38
	Zaključak	40
	Literatura	41
	Izjava o korištenju umjetne inteligencije.....	43
	Skraćenice.....	44
	Privitak	45

Uvod

Relacijske baze podataka desetljećima su dominantan način strukturiranog čuvanja podataka, a jezik SQL i dalje je glavni alat za njihovo pretraživanje. Problem je što SQL zahtijeva poznavanje sheme baze, sintakse i semantike upita, pa ga korisnici bez tehničkog znanja obično ne mogu izravno koristiti. Analitičari, novinari, poslovni korisnici i studenti drugih struka često točno znaju kakvu informaciju traže, ali do nje ne mogu doći bez posrednika, informatičkog osoblja ili unaprijed pripremljenih izvješća.

Razvoj velikih jezičnih modela (engl. *Large Language Model*, LLM) posljednjih je nekoliko godina otvorio mogućnost automatskog prevođenja pitanja iz prirodnog jezika u upit u SQL-u. Ideja je jednostavna: model dobije opis sheme i korisničko pitanje, a njegov je odgovor upit koji sustav potom izvršava nad bazom. Ipak, u praksi izazovi nisu trivijalni. Modeli izmišljaju nepostojeće stupce i tablice, ne pogađaju točan format vrijednosti u uvjetima filtriranja ili proizvedu sintaktički ispravan, ali logički pogrešan upit. Uz to, izvršavanje SQL-a koji je generirao model otvara sigurnosne rizike, sustav mora spriječiti nenamjerno brisanje ili izmjenu podataka, napade kao što je injektiranje SQL-a i pristup dijelovima baze koji nisu predviđeni za pretraživanje.

Cilj je ovog rada razviti programski sustav koji korisniku omogućuje postavljanje upita nad relacijskom bazom putem prirodnog jezika, uz sigurno izvršavanje i mjerljivu točnost. Sustav dinamički dohvaća shemu baze, konstruira upit (engl. (i dalje): *prompt*) s pitanjem, tablicama, stupcima i vezama među njima, koristi jezični model za generiranje SQL-a te dobiveni upit prije izvršavanja provjerava analizom apstraktnoga sintaksnog stabla. Točnost sustava izmjerena je na standardnome referentnom skupu BIRD Mini-Dev [\[11\]](#), koji sadrži pitanja različite složenosti nad jedanaest baza podataka.

Rad ima tri glavna doprinosa. Prvo, izgrađen je cjelovit sustav podijeljen na odvojene komponente, tako da se svaka od njih (dohvat sheme, konstrukcija prompta, validacija i izvršavanje) može zamijeniti neovisno o ostatku sustava. Drugo, sigurnost je riješena na više razina, umjesto oslanjanja na disciplinu jezičnog modela: prompt izričito traži samo `SELECT`, validator na razini apstraktnoga sintaksnog stabla odbija sve što nije čisto čitanje, a pristup bazi ide preko korisnika koji ima isključivo pravo `SELECT`. Treće, na standardni tijek generiranja upita nadovezana je nadogradnja Cascade koja aktivira preciznije korake

samo kad osnovni prolaz ne uspije, čime se točnost podiže bez razmjernog rasta troška izvođenja.

Rad je organiziran kako slijedi. Prvo poglavlje daje pregled teorijske podloge, koja obuhvaća velike jezične modele, jezik SQL i tehnike upravljanja promptovima. Drugo poglavlje analizira srodne radove iz područja pretvorbe prirodnog jezika u SQL i navodi standardne referentne skupove. Treće poglavlje detaljno opisuje arhitekturu razvijenoga sustava, a četvrto implementacijske odabire. Peto poglavlje iznosi rezultate evaluacije po točnosti, sigurnosti, latenciji (engl. *latency*) i pogreškama. Šesto poglavlje raspravlja o rezultatima i uspoređuje ih sa srodnim radovima, a zaključak iznosi sažetak doprinosa i mogućnosti daljnjeg razvoja.

1. Teorijska podloga

Ovo poglavlje daje osnove potrebne za razumijevanje ostatka rada. U 1.1 opisuju se veliki jezični modeli i način na koji generiraju tekst. U 1.2 kratko se ponavljaju ključni koncepti relacijskih baza podataka i jezika SQL. U 1.3 obrađuju se tehnike upravljanja promptovima na kojima se u kasnijim poglavljima temelji arhitektura razvijenoga sustava.

1.1. Veliki jezični modeli

Veliki jezični modeli su neuronske mreže naučene na velikim korpusima teksta koji predviđaju sljedeći znak ili riječ na temelju prethodnih. Većina suvremenih LLM-ova temelji se na transformerskoj arhitekturi [1], koja je zamijenila ranije rekurentne mreže mehanizmom pozornosti (engl. *attention*), čime se omogućuje paralelno obrađivanje sekvenci i bolje modeliranje dugih ovisnosti u tekstu.

Ulazni tekst dijeli se prije obrade na tokene, jedinice manje od riječi, obično podnizove. Suvremeni modeli koriste kodiranja poput kodiranja po parovima znakova (engl. *byte-pair encoding*) koja rječnik tokena uče iz podataka. Model potom pretvara tokene u vektorske reprezentacije i redom kroz slojeve pozornosti proračunava kontekstne veze. Zadnji sloj vraća distribuciju vjerojatnosti nad rječnikom, iz koje se odabire sljedeći token.

Izbor sljedećeg tokena kontrolira nekoliko parametara. Temperatura utječe na to koliko je odabir slučajan: niže vrijednosti daju predvidljiv, gotovo deterministički izlaz, dok više vrijednosti povećavaju raznolikost. Tehnike top-k i top-p uzorkovanja ograničavaju izbor na najvjerojatnije tokene. Za zadatke koji zahtijevaju točnost, poput generiranja SQL-a, koristi se niska temperatura ili čak pohlepno (engl. *greedy*) dekodiranje, kod kojega se uvijek bira najvjerojatniji token.

Ključno ograničenje LLM-ova je kontekstni prozor tj. najveći broj tokena koje model može odjednom obraditi. Suvremeni modeli kreću se u rasponu od nekoliko tisuća do stotina tisuća tokena, no čak i s velikim prozorima kvaliteta odgovora opada kad prompt postane predugačak [3]. Zbog toga izbor koje informacije o bazi pružiti modelu, samo relevantne kolone, veze i uzorke podataka, postaje netrivialan.

Za pretvorbu prirodnog jezika u SQL značajna je i sposobnost učenja s nekoliko primjera (engl. *few-shot learning*), koju su sustavno proučili Brown i sur. [2]: model iz nekoliko

primjera u promptu nauči obrazac odgovora bez potrebe za dodatnim učenjem. Ta osobina, uz razumijevanje prirodnog jezika, čini LLM-ove pogodnim alatom za promatrani zadatak, no ne uklanja rizike halucinacije i semantičkih pogrešaka opisane u uvodu.

1.2. SQL i relacijske baze podataka

Relacijski model podataka predložio je Codd 1970. godine [4]. Podaci se u modelu organiziraju u tablice sastavljene od redaka i stupaca. Svaka tablica predstavlja entitet ili odnos, a veze među tablicama izražavaju se stranim ključevima. Takva organizacija omogućuje formalnu manipulaciju podacima temeljenu na skupovnome računu i relacijskoj algebri.

Jezik *Structured Query Language*, poznatiji kao SQL, razvijen je sredinom sedamdesetih godina prošloga stoljeća kao deklarativan jezik za rad s relacijskim bazama [5]. SQL upit ne opisuje kako doći do rezultata, već što rezultat treba biti; sustav za upravljanje bazom sam pronalazi optimalan način izvođenja. SQL obuhvaća naredbe za dohvat podataka (`SELECT`), izmjenu sadržaja (`INSERT`, `UPDATE`, `DELETE`) i izmjenu strukture baze (`CREATE`, `ALTER`, `DROP`). U ovom radu razmatra se isključivo dohvat podataka.

Složenost naredbe `SELECT` raste od jednostavnog filtriranja jedne tablice do agregiranja preko više spojenih tablica. Osnovni gradbeni blokovi su:

- filtriranje uvjetom u klauzuli `WHERE`,
- spajanje tablica preko naredbe `JOIN` uz uvjet `ON`,
- agregacija (`COUNT`, `SUM`, `AVG`, `MIN`, `MAX`) uz grupiranje kroz `GROUP BY`,
- ograničavanje agregiranih grupa u `HAVING` klauzuli,
- podupiti (engl. *subqueries*), koji omogućuju ugniježđivanje jednog `SELECT`-a unutar drugoga.

Sustav za upravljanje bazom brine se o očuvanju integriteta podataka kroz ograničenja. Primarni ključ osigurava jedinstvenost redaka, strani ključ referencijalnu cjelovitost među tablicama, a ograničenja `NOT NULL` i `UNIQUE` pojedinačno na razini stupaca. Shema baze, popis tablica, njihovih stupaca, tipova i ograničenja, javno je dohvatljiva preko sistemskih tablica, primjerice sheme `information_schema` u PostgreSQL-u ili tablice `sqlite_master` u SQLite-u. Bez toga LLM ne može pisati ispravan SQL jer ne poznaje imena tablica ni stupaca.

Različite implementacije SQL-a razlikuju se u dijalektima. PostgreSQL koristi funkcije poput `NOW()`, operator `ILIKE` i `EXTRACT` za rad s datumima, dok SQLite koristi `date('now')`, obični `LIKE` (koji je već neosjetljiv na velika i mala slova) i funkciju `STRFTIME`. Zbog te razlike sustav za pretvorbu pitanja u SQL mora znati kojemu se dijalektu obraća, jer se isti opis pitanja u dva dijalekta prevodi različito.

1.3. Tehnike upravljanja promptovima

Prompt je tekstni ulaz koji LLM dobije i na temelju kojeg generira odgovor. Način sastavljanja prompta značajno utječe na kvalitetu izlaza, zbog čega se područje inženjerstva promptova razvilo u zasebnu poddisciplinu.[\[2\]](#)

Najjednostavnija tehnika je promptanje bez primjera (engl. *zero-shot*): modelu se daje samo pitanje, bez ijednoga primjera odgovora. Kod jednostavnih zadataka to je dovoljno, ali za složenije upite točnost pada. Promptanje s nekoliko primjera (engl. *few-shot*), koje su detaljno opisali Brown i sur. [\[2\]](#), poboljšava rezultat tako što se u prompt uključi nekoliko parova ulaz–izlaz koji ilustriraju očekivani format i stil odgovora. Kod pretvorbe prirodnog jezika u SQL primjeri sadrže parove pitanje–SQL nad srodnim shemama.

Lanac razmišljanja (engl. *Chain-of-Thought*) [\[6\]](#) potiče model da prije konačnoga odgovora eksplicitno ispiše korake razmišljanja. Kod matematičkih i logičkih zadataka takav pristup značajno povećava točnost, a kod generiranja SQL-a omogućuje modelu da najprije razjasni koje tablice treba spojiti i po kojem ključu, prije nego napiše konačan upit.

Samo-konzistentnost (engl. *self-consistency*) [\[7\]](#) nadograđuje lanac razmišljanja pozivanjem modela više puta uz nešto višu temperaturu (parametar koji kontrolira slučajnost izbora tokena, opisan u odjeljku 1.1), čime se dobiva niz međusobno neovisnih pokušaja. Konačan odgovor bira se glasanjem, a najčešći rezultat pobjeđuje. Za pretvorbu prirodnog jezika u SQL glasanje se provodi nad rezultatom izvršavanja SQL-a, a ne nad samim tekstom upita, jer semantički istovjetni upiti mogu biti različitih tekstualnih oblika.

Za područje pretvorbe prirodnog jezika u SQL razvijene su i specijalizirane tehnike. Povezivanje sa shemom (engl. *schema linking*) [\[8\]](#) ograničava opis sheme u promptu samo na tablice i stupce relevantne za dano pitanje, čime se smanjuje šum i olakšava modelu izbor pravih identifikatora. Dekompozicija pitanja [\[9\]](#) razlaže složeno pitanje u niz jednostavnijih koraka koje model rješava redom, što se pokazalo korisnim za pitanja s više spajanja i agregacija.

Sve navedene tehnike međusobno se ne isključuju; naprotiv, u ovom radu one se kombiniraju kroz cjelovit tijek i pametnu nadogradnju opisanu u kasnijim poglavljima.

2. Srodni radovi (Text-to-SQL)

Pretvorba prirodnog jezika u SQL područje je s dugom istraživačkom poviješću. U ovom se poglavlju sažeto opisuju pristupi koji su prethodili eri LLM-ova, potom se detaljnije razmatraju četiri utjecajna rada temeljena na velikim jezičnim modelima, te se navode standardni referentni skupovi na kojima se novi sustavi uspoređuju. Naglasak potpoglavlja 2.2 je na usporedbi s pristupom razvijenim u ovom radu.

2.1. Rani pristupi

Prvi pokušaji automatske pretvorbe prirodnog jezika u SQL sežu u sedamdesete godine prošloga stoljeća, kad su se koristili sustavi temeljeni na pravilima [\[14\]](#) (engl. *rule-based systems*). Takvi su sustavi radili na uskim domenama i zahtijevali su ručno oblikovan skup pravila za svaku bazu. Njihova su ograničenja bila slaba prilagodljivost novim shemama i rječniku.

Pojavom dubokoga učenja pojavljuju se pristupi temeljeni na neuronskim mrežama. Zhong i sur. [\[14\]](#) predstavili su Seq2SQL, model sekvenca-u-sekvencu koji je pitanje pretvarao izravno u SQL. Xu i sur. [\[15\]](#) razvili su SQLNet, arhitekturu s više specijaliziranih pod-modela koji su zajedno predviđali dijelove `SELECT` naredbe (agregaciju, uvjete u `WHERE`, izbor stupaca). Ovi su modeli postizali oko 60–70 % točnosti na skupu WikiSQL, ali samo za jednostavne upite nad jednom tablicom.[\[17\]](#)

Sljedeći korak bili su pristupi s izričitim modeliranjem sheme. IRNet [\[16\]](#) uvodi međurazinski jezik SemQL kao apstrakciju SQL-a, čime se skraćuje razmak između pitanja i konačnog upita. RAT-SQL [\[17\]](#) primjenjuje relacijski transformer koji istodobno kodira pitanje i shemu, gdje su veze među stupcima ugrađene kao induktivna prednost. [\[16\]\[17\]](#)

Iako su ovi pristupi činili značajan napredak, svi su zahtijevali skupo označene skupove za učenje i teško su se generalizirali na nove domene. Dolazak velikih jezičnih modela promijenio je paradigmu: umjesto specijaliziranog modela učenog isključivo za SQL, iskorišten je općeniti jezični model koji rješenje izvodi iz kombinacije prompta i unaprijed naučenoga jezičnog i tehničkog znanja.

2.2. pristupi bazirani na velikim jezičnim modelima

Prelaskom na pristupe temeljene na LLM-ovima područje je preskočilo za više desetaka postotnih bodova točnosti. U nastavku se detaljno razmatraju četiri utjecajna rada koji predstavljaju različite strategije poboljšanja rezultata na skupu BIRD.

Pristup DIN-SQL: Pourreza i Rafiei [9] predstavili su pristup temeljen na dekompoziciji upita (engl. *Decomposed In-Context Learning*). Cijeli postupak dijeli se u više koraka: klasifikaciju složenosti pitanja, izdvajanje relevantnih tablica, izgradnju SQL-a korak po korak i završnu samo-korekciju. Radeći s modelom GPT-4, DIN-SQL postiže oko 50 % točnosti izvršavanja (engl. *execution accuracy*, EX) na skupu BIRD Mini-Dev.

Pristup DAIL-SQL: Gao i sur. [8] usredotočili su se na dvije komponente. Prva je povezivanje sa shemom, kojim se u prompt uključuju samo tablice i stupci relevantni za dano pitanje. Druga je dinamičko biranje *few-shot* primjera pomoću sličnosti vektorske reprezentacije: model dobije nekoliko sličnih primjera pitanje–SQL koji su preuzeti iz skupa za učenje. Kombinacija s GPT-4 postiže oko 55 % EX, čime je pristup neko vrijeme držao vrh ljestvice.

Pristup MAC-SQL: Wang i sur. [12] predlažu višeagentsku arhitekturu u kojoj više specijaliziranih agenata surađuje iterativno: jedan bira relevantne tablice (selektor), drugi razlaže pitanje u korake (dekompozer), treći evaluira i po potrebi popravljajući upit (evaluator). Umjesto jednog LLM poziva, sustav pokreće nekoliko koraka gdje svaki agent doprinosi svojoj razini apstrakcije. Rezultat je oko 60 % EX na BIRD-u uz GPT-4.

Pristup CHESS: Talaei i sur. [13] predstavljaju trenutačni vrhunski pristup. Njihov pristup uključuje zasebne module za sintezu sheme (izdvajanje relevantnih dijelova baze), planiranje upita, generiranje SQL-a i reviziju temeljenu na testovima, svaki predloženi upit izvršava se na uzorku podataka i uspoređuje s očekivanim izlazom. CHESS s GPT-4 dostiže oko 70 % EX na skupu BIRD Mini-Dev.

Tablica 2.1 sažima ključne karakteristike navedenih pristupa i uspoređuje ih s pristupom razvijenim u ovom radu.

Tablica 2.1. Usporedba srodnih pristupa na skupu BIRD Mini-Dev

Pristup	Model	Ključna ideja	EX (%)
DIN-SQL	GPT-4	Dekompozicija + samo-korekcija	~50
DAIL-SQL	GPT-4	Schema linking + retrieval <i>few-shot</i>	~55
MAC-SQL	GPT-4	Multi-agent arhitektura	~60
CHESS	GPT-4	Sinteza sheme + revizija temeljena na testovima	~70
Ovaj rad	GPT-4o-mini	Cascade + iterativno podešen prompt	47

Iz tablice se vidi da svi navedeni radovi koriste model GPT-4, koji je znatno skuplji od modela korištenog u ovom radu. U trenutku izrade rada, cijena GPT-4 varijante iznosi oko 30 USD po milijun izlaznih tokena, dok GPT-4o-mini košta oko 0,60 USD po milijun izlaznih tokena, oko pedeset puta manje. U tom kontekstu, rezultat od 47 % EX s modelom koji je 50× jeftiniji ne treba vrednovati u smislu apsolutnog broja, nego omjerom točnosti prema trošku. Rezultat je usporediv s DIN-SQL pristupom (oko 50 %), uz značajno nižu cijenu izvođenja, dok je zaostatak za CHESS-om (~70 %) očekivan s obzirom na razliku u sposobnostima modela.

Također valja napomenuti da se ovaj rad odvaja od navedenih po dvije dodatne dimenzije. Prvo, veći dio srodnih radova ne uključuje formalnu provjeru sigurnosti generiranog SQL-

a, pretpostavlja se povjerenje u izlaz modela. U ovom se radu na tu dimenziju posebno stavlja naglasak kroz višeslojan model ograničavanja opsega SQL operacija (poglavlje [3.5](#)). Drugo, u odjeljku 3.4 predstavlja se pristup nazvan Cascade koji naprednije tehnike aktivira samo pri neuspjehu osnovnog prolaza, čime se čuva trošak izvođenja bez proporcionalnog gubitka točnosti.

2.3. Referentni skupovi

Za mjerenje kvalitete sustava *Text-to-SQL* u zajednici se koriste standardizirani referentni skupovi (engl. *benchmarks*). Dva najutjecajnija su Spider i BIRD.

Spider [\[10\]](#) sadrži 10 181 pitanje nad 200 baza podataka koje pokrivaju 138 različitih domena. Skup je podijeljen na podskupove za učenje, razvoj i testiranje, s naglaskom na složenim upitima koji uključuju više spajanja i agregacija. Karakteristika Spidera je da se razvojni i testni podskup ne preklapaju s dijelom za učenje po domenama, dakle sustav mora znati generalizirati na potpuno nove sheme. Osnovna metrika je egzaktna podudarnost (engl. *exact match*) nakon sintaksne normalizacije SQL-a.

BIRD [\[11\]](#) (engl. *Big Bench for Large-scale Database Grounded Text-to-SQL Evaluation*) noviji je i zahtjevniji skup. Sadrži 12 751 pitanje nad 95 baza podataka temeljenih na realnim izvorima, od kojih pojedinačna baza može biti veličine do desetak gigabajta. Za razliku od Spidera, BIRD ne mjeri egzaktnu podudarnost SQL-a već točnost izvršavanja: dva različita SQL upita koji daju istovjetan rezultat oba se broje kao točna. Ova metrika bolje odražava stvarnu korisnost sustava, jer korisnika zanima točan podatak, a ne konkretan oblik upita.

BIRD Mini-Dev podskup je BIRD-a namijenjen bržoj evaluaciji: 500 pitanja nad 11 baza, s poznatim referentnim SQL upitima. Sustav se procjenjuje tako što se svaki generirani upit izvrši nad bazom i rezultat se uspoređi s rezultatom referentnoga upita. Zbog svoje veličine postao je de facto standard za usporedbu novih pristupa. U ovom se radu koristi upravo taj podskup, kako bi rezultati bili izravno usporedivi s referentnim istraživanjima navedenima u odjeljku [2.2](#).

Uz Spider i BIRD spominju se i drugi referentni skupovi (WikiSQL, SPaC, CoSQL), no oni su ili premali za značajne zaključke ili se bave specifičnim aspektima kao što su više-koračni razgovorni upiti. Evaluacija u ovom radu ograničena je na BIRD Mini-Dev.

2.4. pristupi temeljeni na posebnome podešavanju modela

Uz pristupe koji koriste općenite jezične modele bez dodatnog učenja (prompt inženjerstvo, uzorkovanje *few-shot*, dekompozicija upita), u posljednje se dvije godine razvija skupina pristupa temeljena na finom ugađanju (engl. *fine-tuning*) manjih otvorenih modela specifično za zadatak pretvorbe prirodnog jezika u SQL. Ovi radovi pokazuju da posebno podešen model može zamijeniti komercijalne modele klase GPT-4 uz osjetno niži trošak izvođenja.

CodeS [\[11\]](#) je serija otvorenih modela veličine od 1 do 15 milijardi parametara, posebno podešenih na kombiniranom skupu Spiderovih i BIRD-ovih podataka za učenje. Uz sam model, autori prilažu i modul za povezivanje sa shemom koji radi neovisno o LLM-u, sličan pristup našem opisu u odjeljku 3.3. Najveći model, CodeS-15B, postiže oko 60 % EX na BIRD-ovom razvojnem skupu, što je usporedivo s pristupom MAC-SQL uz značajno manje računsko opterećenje po pitanju.

SQL-PaLM [\[19\]](#) predstavlja Googleov pristup s posebno podešavanjem modela PaLM-2. Uz to koristi ansambl više posebno podešenih modela s glasanjem, ideja slična našoj samokonzistentnosti u Sloju 3, ali s više specijaliziranih modela umjesto jednog modela s višom temperaturom.

XiYan-SQL [\[20\]](#) kombinira posebno podešene modele za povezivanje sa shemom s velikim modelom za završnu generaciju SQL-a. Trenutačno drži jedno od najboljih mjesta na BIRD-ovoj ljestvici i pokazuje da se sinergijom manjih posebno podešenih komponenata i općenitog LLM-a može postići rezultat koji je konkurentan pristupima temeljenima isključivo na velikim komercijalnim modelima.

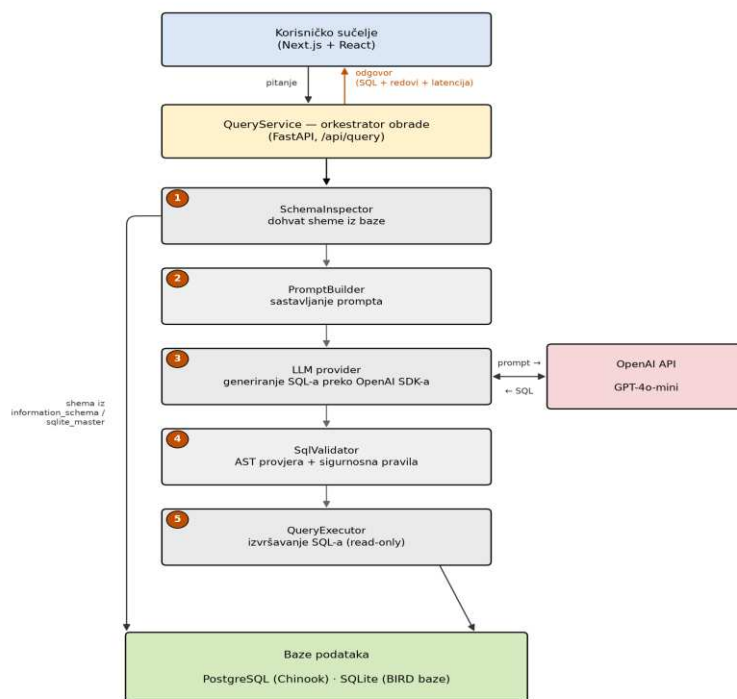
Zajednička točka svih pristupa temeljenih na posebnome podešavanju je izmjena težina same neuronske mreže na temelju podataka za učenje BIRD-a ili srodnih skupova. To čini pristup manje uopćivim (posebno podešen model može biti previše prilagođen specifičnom skupu baza), ali omogućuje visoku točnost uz osjetno manje veličine modela. U okviru ovoga rada posebno podešavanje nije provedeno jer je težište stavljeno na modularno korištenje općeg modela, smjer koji je otvoreniji za brzu prilagodbu novim domenama, uz teorijsko ograničenje gornjega praga točnosti. Poglavlje 6.3 detaljnije raspravlja o posebnome podešavanju kao smjeru daljnjega rada.

3. Arhitektura predloženog sustava

U ovom se poglavlju detaljno opisuje arhitektura razvijenog sustava. Nakon sažetog pregleda komponenti (3.1), redom se obrađuje dinamičko dohvaćanje sheme (3.2), konstrukcija prompta (3.3), nadogradnja Cascade (3.4), višeslojna validacija SQL upita (3.5) te izvršavanje upita s mehanizmom ponavljanja (3.6).

3.1. Pregled komponenti

Sustav je organiziran u dva glavna dijela: klijentsku aplikaciju (engl. front-end) i poslužiteljsku aplikaciju (engl. back-end). Klijentska aplikacija zadužena je za prihvatanje pitanja od korisnika i prikaz rezultata, dok se cijela logika obrade, dohvatanje sheme, konstrukcija prompta, komunikacija s LLM-om, validacija i izvršavanje SQL-a, odvija na poslužiteljskoj strani. Sl. 3.1 prikazuje glavne komponente i tokove podataka među njima.



Sl. 3.1. Arhitektura predloženog sustava

Ključne komponente su sljedeće. SchemaInspector dinamički dohvaća shemu odabrane baze i drži je u memoriji. PromptBuilder konstruira tekst prompta prema odabranoj strategiji (A

ili D). LLM provider komunicira s vanjskom OpenAI uslugom preko službenog SDK-a. SqlValidator provodi strukturalnu i sigurnosnu analizu generiranog upita. QueryExecutor izvršava validirani upit nad bazom uz ograničenja postavljena na razini korisnika baze.

Sve komponente povezane su kroz FastAPI orkestrator (QueryService) koji provodi ulazno pitanje kroz cijeli tijek i sastavlja odgovor za korisnika. Odabir modula na ovaj način omogućuje jasno razgraničenje odgovornosti i olakšava kasnija proširenja, primjerice, dodavanje novog LLM kao davatelja usluge zahtijeva samo implementaciju istog sučelja koje koristi postojeći.

3.2. Dinamičko dohvaćanje sheme

SchemaInspector čita metapodatke baze u trenutku prve upotrebe i sprema ih u memoriju. Za PostgreSQL koristi standardnu shemu `information_schema`, iz koje čita tablice, stupce, tipove podataka, primarne ključeve i strane ključeve. Za SQLite koristi sistemsku tablicu `sqlite_master` i pragma naredbe `PRAGMA table_info` te `PRAGMA foreign_key_list`, koje daju istu informaciju u drugačijem obliku.

Dohvaćena shema pretvara se u interni objekt DTO (engl. *Data Transfer Object*) koji sadrži popis tablica, za svaku tablicu popis stupaca s tipovima i oznakama, te popis stranih ključeva prema drugim tablicama. Ovaj oblik odabran je jer je jednostavan za pretvorbu u tekst prompta i za serijalizaciju u JSON-ov odgovor prema korisničkom sučelju.

Za povećanje kvalitete generiranog SQL-a, SchemaInspector za svaku tablicu dohvaća i uzorak podataka, do tri retka, samo za strategiju D. Uzorak pomaže LLM-u da nauči točan format vrijednosti u stupcima (na primjer, je li ime države zapisano kao „USA” ili „United States”) i tako izbjegne uobičajenu grešku pisanja uvjeta `WHERE` s nepostojećom vrijednošću.

Rezultati inspektora keširaju se u memoriji tijekom trajanja procesa, s vremenom valjanosti (engl. *Time-To-Live*). Time se izbjegava ponovno kontaktiranje baze pri svakom pitanju, što bi bilo nepotrebno usporavanje jer se shema tijekom rada mijenja rijetko.

3.3. Konstrukcija prompta

PromptBuilder sastavlja tekst koji se šalje jezičnom modelu. Struktura prompta razdvojena je na dva dijela: sistemski prompt, koji opisuje ulogu modela i pravila ponašanja, te korisnički prompt, koji sadrži pitanje i shemu baze. Sistemski prompt postavlja se jednom

po sesiji i uključuje eksplicitno pravilo da odgovor mora biti isključivo naredba `SELECT`, bez dodatnog teksta ili objašnjenja.

Za korisnički prompt sustav primjenjuje jednu od dvije strategije. Strategija A daje modelu samo pitanje i služi kao naivna osnova za usporedbu. Strategija D uključuje potpun kontekst: opis sheme s tablicama, stupcima i tipovima, veze između tablica (strane ključeve), uzorke redova (do tri po tablici), opis stupaca iz BIRD dataseta (kad je dostupan) te po potrebi tzv. *evidence hint*, tj. kratku uputu koja pojašnjava značenje pojmova iz pitanja u odnosu na stupce baze.

Uz obogaćen kontekst, strategija D koristi tehniku lanca razmišljanja: prompt sadrži uputu modelu da najprije unutar oznaka `<plan></plan>` kratko opiše koje tablice treba spojiti i po kojem ključu, a tek potom napiše SQL. Struktura rezultata modela olakšava kasniju dijagnostiku pogrešaka jer se plan i konačan upit mogu čitati zajedno.

Za slučajeve kad je pitanje složeno, dodatna dekompozicija [\[9\]](#) razlaže pitanje u niz podkoraka koji se zatim uključe u prompt kao dodatna smjernica. Ovaj korak nije uvijek aktivan; aktivira ga Cascade sloj opisan u 3.4.

Iterativno podešavanje sistemskog prompta. Osnovni sistemski prompt sadrži opća pravila (isključivo `SELECT`, jedan iskaz po odgovoru, korištenje samo tablica i stupaca iz sheme, oznake za spajanje). Ta pravila jamče sintaksnu ispravnost, ali ne pokrivaju semantičke sitnice po kojima automatsko ocjenjivanje razlikuje točno od netočnog odgovora. Manje odstupanje u broju stupaca, izboru identifikatora ili preciznosti računa dovoljno je da rezultat bude označen kao pogrešan, iako je smisao vrlo blizu očekivanome.

Cijeli postupak podešavanja tekao je u ciklusima. Nakon svakog pokretanja prošlo se po pogrešnim odgovorima i grupiralo ih se u kategorije prema tipu greške, primjerice: model vraća samo jedan stupac umjesto dva, model poziva funkciju kakva u SQLite-u ne postoji, model bira stupac iz krive tablice. Za svaku takvu kategoriju napisano je jedno konkretno pravilo u sistemskom promptu, u obliku kratke naredbe s jednim primjerom kako ga primijeniti. Ponovilo bi se pokretanje i pratilo koliko su iste kategorije još učestale. Kategorije koje su nakon nekoliko rundi pale ispod nekoliko postotaka smatrale su se riješenima; ostale su ili tražile još jednu iteraciju ili su prihvaćene kao trenutačna ograničenja pipeline-a.

Na taj su način prepoznati i uklonjeni sljedeći tipovi grešaka, uz odgovarajuće pravilo u sistemskom promptu:

Prvo pravilo pokrivalo je dopuštene funkcije dijalekta SQLite. U ranijim je verzijama model povremeno pozivao funkcije koje u SQLiteu ne postoje, poput `DIVIDE`, `DATEDIFF`, `IF` ili `TOP N`. Popis dopuštenih funkcija (`SUM`, `AVG`, `COUNT`, `MIN`, `MAX`, `ROUND`, `CAST`, `ABS`, `LENGTH`, `SUBSTR`, `IIF`, `CASE WHEN`, `STRFTIME`) unesen je izravno u sistemski prompt, uz kratki podsjetnik da se dijeljenje piše operatorom `/`, a decimalna vrijednost dobiva se `CAST`-om jednog operanda u `REAL`.

Drugo pravilo tiče se preciznosti: kad pitanje traži određen broj decimalnih mjesta ili postotak, u prompt je dodana uputa da model obavezno koristi `ROUND`. `CAST` na `REAL(P, S)` u SQLiteu ne obrezuje vrijednost, samo označava tip, pa sam `pass` tipa nije bio dovoljan.

Sljedeća skupina pravila pokrivala je izbor stupaca i tablica. Kada isti atribut poput naziva škole postoji u tablici koja opisuje entitet (`schools.School`) i u denormaliziranoj kopiji u drugoj tablici (`frpm."School Name"`), model je bio obavezan uzeti izvornik iz „izvorišne” tablice, jer denormalizirana kopija ne mora biti ažurna. Za formulacije poput „posts made by user X” ili „atoms in a bond”, gdje logičko spajanje prolazi kroz posrednu tablicu (`postHistory`, `connected`), pravilo je izričito tražilo uključivanje te posrednice u `JOIN`, iako je pitanje ne spominje. Zasebno pravilo odnosilo se na broj stupaca u ispisu: kad pitanje traži više atributa istovremeno, sustav ih mora vratiti sve u istom redoslijedu, a kad traži samo jedan, dodavanje „pomoćnog” identifikatora ili brojača kraj traženog stupca vodi u pogrešnu ocjenu.

Posljednje pravilo bavilo se sintaksnim spletom oko `HAVING`. Model je povremeno generirao `HAVING` bez odgovarajućeg `GROUP BY`, što u SQLiteu vodi izravnoj grešci. Uputa u promptu preusmjeruje takve filtere u `WHERE` s podupitom ili u CTE, ovisno o formi upita. Uz svako od navedenih pravila u prompt je dodan jedan konkretan primjer para pitanje-SQL koji ga oprimjeruje. Pristup je u literaturi opisan kao *few-shot in-context learning* [6], a njegova je snaga u tome što model istovremeno dobiva apstraktnu naredbu i njezinu primjenu na konkretnom obrascu.

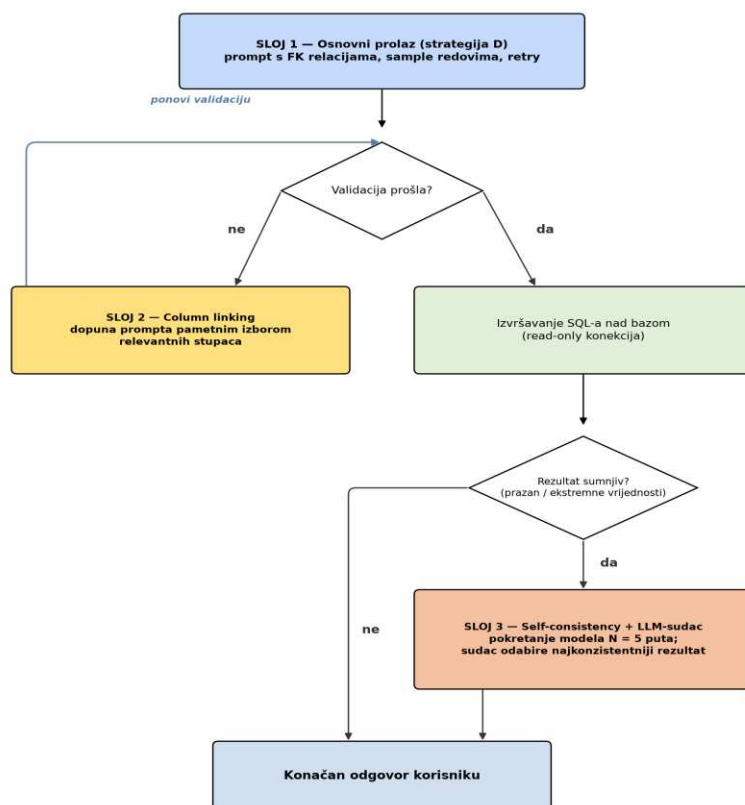
Uz pravila priložena su tri-četiri konkretna primjera (parovi pitanja i SQL-a) koji ilustriraju svaki obrazac. Ovaj se pristup u literaturi naziva *few-shot in-context learning* [6] i pokazao se učinkovitim jer model dobiva ne samo pravilo kao apstrakciju nego i njegovu primjenu kao konkretan sintaksnu predložak. Primjeri su pisani da pokrivaju SQLite-specifične konstrukcije (`STRFTIME`, `IIF`, `||` za konkatenciju, dvostruki navodnici za identifikatore s razmakom), tako da model paralelno uči i dijalekt.

Učinak ovog postupka je mjerljiv. Prije uvođenja opisanih pravila konfiguracija Cascade davala je 35,5 % EX na početnom podskupu od 200 pitanja. S dodanim pravilima, na uzorku od 200 pitanja koji ravnomjerno pokriva sve baze i sve težine, rezultat je 47,0 % EX. Dobitak od 11,5 postotnih bodova postignut je isključivo kroz prompt, bez ijedne promjene modela, arhitekture ili cjevovoda. Doprinos pojedinih pravila razlaže se u poglavlju [5.1](#).

3.4. Nadogradnja Cascade

Osnovni tijek (strategija D) proizvodi SQL, validira ga i izvršava. Za većinu pitanja ovaj je prolaz dovoljan. Za dio pitanja, međutim, prolaz može podbaciti, validator odbaci SQL zbog nepostojećeg stupca, ili izvršavanje vrati sumnjiv rezultat (prazan skup, ekstremne vrijednosti). Nadogradnja Cascade rješava takve slučajeve uvjetnim uključivanjem naprednijih tehnika, umjesto da se te tehnike uvijek aktiviraju.

Sustav radi u tri sloja koji se aktiviraju samo po potrebi. Sl. 3.2 prikazuje logiku odlučivanja.



Sl. 3.2. Odlučivanje nadogradnje Cascade po slojevima

Sloj 1 je osnovni prolaz opisan u prethodnom odjeljku. Sloj 2 aktivira se ako validacija ne uspije: prompt se dopunjava dodatnim korakom povezivanja stupaca sa shemom, u kojem

se izborom relevantnih stupaca dodatno usmjerava izbor tablica. Nakon toga se validacija pokušava ponovno. Sloj 3 aktivira se ako izvršavanje vrati sumnjiv rezultat: sustav poziva LLM pet puta (*self-consistency*, $N = 5$) i konačan izbor donosi LLM-sudac koji među pet kandidata bira najkonzistentniji na temelju usporedbe njihovih izvršenja.

Ovaj pristup razlikuje se od pristupa koji odjednom aktiviraju sve napredne tehnike (poput MAC-SQL-a). Uvjetna aktivacija čuva prosječan trošak izvođenja jer se za jednostavna pitanja i dalje izvodi samo jedan poziv modela, dok se za teška pitanja aktivira više koraka.

3.5. Validator SQL upita

Validator SQL upita najosjetljiviji je dio sustava jer određuje smije li generirani upit uopće doći do baze. Njegov je zadatak dvostruk: provjeriti da je upit sintaksno ispravan i da leži unutar dopuštenog opsega operacija (samo `SELECT` nad postojećim tablicama).

Validator ne koristi jezični model niti pretragu podniza (engl. *substring match*), nego apstraktno sintaksno stablo (engl. *abstract syntax tree*, AST) generirano bibliotekom `sqlglot`. Cijela naredba parsira se u stablo čvorova; svaki čvor je poznatoga tipa (npr. `SELECT`, `WHERE`, `JOIN`, `INSERT`, `DROP`). Provjera se svodi na obilazak stabla i inspekciju tipova čvorova, a ne na tekstnu analizu, čime se izbjegavaju uobičajene greške poput podudaranja podniza „`DELETE`” u imenu stupca „`delete_flag`”.

Provjera se odvija u nekoliko koraka:

- parsiranje: SQL se parsira u AST; ako to ne uspije, upit se odbija kao sintaksno neispravan,
- provjera broja naredbi: dopuštena je točno jedna naredba, čime se sprječava napad tipa injekcija SQL-a sa spajanjem više upita (npr. `SELECT ... ; DELETE ...`),
- provjera tipa naredbe: dopuštena je samo naredba `SELECT`; `INSERT`, `UPDATE`, `DELETE`, `DROP`, `ALTER`, `CREATE` i sve ostale odbijaju se odmah,
- semantička provjera: sve tablice i stupci u upitu moraju postojati u shemi; nepoznati identifikator vraća se kao greška LLM-u za ponovni pokušaj,

- prisiljavanje ograničenja: ako upit nema klauzulu `LIMIT`, sustav automatski dodaje `LIMIT 1000` kako bi izbjeglo vraćanje ogromnih skupova nad velikim tablicama.

Tablica 3.1 sažima glavne napadne vektore koji su bili razmatrani i način na koji ih pojedini sloj obrane presreće.

Tablica 3.1. Napadni vektori i odgovarajući sloj obrane

Napadni vektor	Primjer	Sloj obrane
Uništavanje tablice	<code>DROP TABLE customers</code>	Validator (samo <code>SELECT</code>)
Brisanje podataka	<code>DELETE FROM invoice</code>	Validator + <i>read-only user</i>
Injekcija putem više izjava	<code>SELECT 1; DROP TABLE x</code>	Validator (točno 1 naredba)
Modifikacija podataka	<code>UPDATE customer SET ...</code>	Validator + <i>read-only user</i>
Halucinacija sheme	<code>SELECT * FROM nepostojeća</code>	Semantička provjera
Preveliki rezultat	<code>SELECT * FROM big_table</code> (bez <code>LIMIT</code>)	Auto- <code>LIMIT 1000</code>
Sporo izvršavanje	kartezijski <code>JOIN</code> više velikih tablica	Timeout 60 s

Ako korisničko pitanje uzrokuje generiranje opasnog SQL-a, bilo namjerno (napad) bilo slučajno (pogrešna interpretacija pitanja), validator ga zaustavlja prije doticaja s bazom. Ako se, unatoč validaciji, u sustavu pojavi greška, kao dodatna razina obrane koristi se poseban korisnik baze koji ima jedino ovlast `GRANT SELECT`. Time se osigurava da čak i propuštena naredba ne može mijenjati podatke.

Sl. 3.3 prikazuje ovaj mehanizam u sučelju: pri unosu zahtjeva „Drop all the circuits” nad BIRD bazom Formula 1 model odbija generirati `DROP` naredbu i vraća unaprijed definiran siguran odgovor, a sučelje prikazuje crveni panel s objašnjenjem korisniku umjesto tablice rezultata. Iako u ovom slučaju validator nije morao intervenirati, on ostaje spreman ako

budući model ipak generira opasan upit, što je i pokazano u odjeljku 5.2 kroz izravno testiranje validatora.

The screenshot displays the NL2SQL interface. At the top, it says 'Convert natural-language questions into safely executable SQL queries against a relational database.' Below this, the 'Active database' is set to 'Formula 1'. The 'Question' input field contains 'Delete all circuits'. A button 'Generate and run SQL' is visible. The 'Status' section shows 'Refused - security layer'. A 'Latency by phase' table shows: Prompt build (8 ms), LLM (2.95 s), Validation (3 ms), and Execution (4 ms). A red box contains the message: 'Request refused (security layer). The question requests an operation outside the system's allowed scope. This system supports read-only queries (SELECT) — operations such as delete, update, insert or schema changes are not permitted. The model refused to generate SQL for this request and returned a safe fallback response. This system only allows read-only queries (SELECT). Statements that modify data or schema (DROP, DELETE, UPDATE, INSERT, ALTER, or multi-statement injection) are stopped before they reach the database.' Below this, the 'Generated SQL' section shows a normalized query: 'SELECT "unable to answer" AS error'. The 'Question history' section shows a list with 'Delete all circuits'.

Table	Columns
circuits (9 col)	circuitId (PK, integer), circuitRef (text), name (text), location (text), country (text), lat (real), lng (real), alt (integer), url (text)
constructorResults (5 col)	constructorResultsId (PK, integer), raceId (FK, integer), constructorId (FK, integer), points (real), status (text)
constructorStandings (7 col)	constructorStandingsId (PK, integer), raceId (FK, integer), constructorId (FK, integer), points (real), position (integer), positionText (text), wins (integer)
constructors (5 col)	constructorId (PK, integer), constructorRef (text), name (text), nationality (text), url (text)
driverStandings (7 col)	driverStandingsId (PK, integer), raceId (FK, integer), driverId (FK, integer), points (real), position (integer), positionText (text), wins (integer)
drivers (9 col)	driverId (PK, integer), driverRef (text), number (integer)

Sl. 3.3. Odbijanje opasnog zahtjeva na razini jezičnog modela (Sloj 1 obrane)

3.6. Izvršavanje upita i mehanizam ponavljanja

Nakon uspješne validacije upit se izvršava kroz komponentu QueryExecutor. Ova komponenta koristi zasebnu vezu prema bazi koja je postavljena kao *read-only*, a upit ima postavljeni *timeout* od 60 sekundi. Istek vremena štiti sustav od pogrešno napisanih upita koji bi mogli zaključati resurse ili trošiti procesorsko vrijeme na kartezijskom produktu velikih tablica.

Za slučajeve kad validacija ne prođe zbog greške u tekstu SQL-a, primjerice, „nepostojeći stupac“, sustav pokreće mehanizam ponavljanja (engl. *retry*). Retry Engine sastavlja novi prompt koji uz izvorno pitanje uključuje i tekst prethodnog SQL-a te opis greške, i traži od modela popravak. Broj pokušaja ograničen je: za tipografske greške u imenima tablica ili stupaca dopuštaju se dva pokušaja, dok se za tipske greške (npr. usporedba stringa s brojem) dopušta samo jedan.

Klasifikacija greške određuje se jednostavnim uparivanjem uzorka u tekstu koji baza vraća. Iako pristup nije potpun, u praksi pokriva glavninu slučajeva; empirijski rezultat pokazuje da druga i naredne iteracije brzo dosežu granicu korisnosti, model ili popravi grešku prvim pokušajem, ili je greška takva da dodatni pokušaji ne pomažu.

Rezultat izvršavanja, stupci, redovi, broj redaka i vrijeme izvođenja, pakira se u odgovor koji se prosljeđuje klijentskom sučelju. Uz podatke, odgovor uključuje i mjere vremena po fazama: trajanje dohvata sheme, trajanje LLM poziva, trajanje validacije i trajanje izvršavanja upita. Te se mjere prikazuju korisniku i koriste se za analizu latencije u poglavlju [5](#).

4. Implementacija

Prethodno poglavlje opisuje arhitekturu na razini komponenti; ovo se poglavlje kratko osvrće na tehnološki izbor, organizaciju izvornog koda, kontejnerizaciju i konačan izgled korisničkog sučelja.

4.1. Korištene tehnologije

Poslužiteljska strana napisana je u Pythonu 3.12 uz okvir FastAPI koji koristi Pydantic v2 za validaciju ulaza i izlaza te uvicorn kao ASGI poslužitelj. Za rad s bazom koristi se SQLAlchemy 2.x u asinkronom načinu rada, zajedno s pokretačima asyncpg za PostgreSQL i aiosqlite za SQLite. Za parsiranje i normalizaciju SQL-a odabrana je biblioteka sqlglot, čije apstraktno sintaksko stablo čini srž validatora. Komunikacija s vanjskim jezičnim modelom obavlja se službenim OpenAI SDK-om.

Klijentska strana temelji se na Next.js 14 uz React 19, TypeScript i TailwindCSS. Odabir Next.js-a motiviran je jasnim modelom usmjeravanja stranica kroz App Router te potpunom podrškom za TypeScript, koji smanjuje broj grešaka pri komunikaciji s poslužiteljem. Za oblikovanje se koristi Tailwind, pristup u kojem se stilovi ostavljaju uz oznake elemenata, čime se izbjegava odvojen sloj CSS-a i pojednostavljuje održavanje.

Kao demo baza koristi se javno dostupna Chinook baza glazbene trgovine (11 tablica, ~275 izvođača, ~350 albuma, ~3500 pjesama), a za evaluaciju točnosti služi BIRD Mini-Dev skup s 500 pitanja nad 11 baza različitih domena, od financijskih transakcija do sportske statistike.

4.2. Modularna organizacija koda

Poslužiteljski dio organiziran je u direktoriju backend/app s osam glavnih podmodula. Modul *api* sadrži HTTP točke za /query, /schema, /health i /databases, modul *core* sadrži logiranje, mjerenje vremena, iznimke i audit trag, modul *db* sadrži bazičnu vezu (engine) i SchemaInspector, modul *llm* sadrži sučelje i implementaciju LLM providera, gradnju prompta i predloške, modul *validation* sadrži sqlglot AST provjere, sigurnosne i semantičke provjere te mehanizme prisile (auto-LIMIT), modul *services* sadrži orkestrator QueryService, RetryEngine, ExecutionService i BenchmarkQueryService, modul

evaluation sadrži benchmark runner, metrike i analizu pogrešaka, a modul *models* sadrži Pydantic DTO objekte za API.

Klijentski dio (direktorij `frontend/src`) prati konvenciju App Routera od Next.js-a i sadrži komponente organizirane oko funkcionalnosti, izbor baze, prikaz sheme, forma za pitanje, prikaz SQL-a i rezultata, graf latencije po fazama. Za dohvat podataka koristi se jednostavan sloj oko funkcije `fetch`, bez dodatnih biblioteka podatkovnog sloja.

4.3. Kontejnerizacija i pokretanje

Cijeli sustav pokreće se jednom naredbom: `docker compose up`. Datoteka `docker-compose.yml` opisuje tri usluge: *postgres* (PostgreSQL 16 s automatskim učitavanjem baze Chinook pri prvom pokretanju), *backend* (aplikacija FastAPI s automatskim osvježavanjem koda putem dijeljenih direktorija) i *frontend* (Next.js-ov razvojni poslužitelj). Pristupi na `localhost:3000` (korisničko sučelje) i `localhost:8000/docs` (Swagger UI za API) dostupni su odmah nakon pokretanja.

U okruženju se osim vlasnika baze (`nl2sql_owner`) automatski kreira i poseban korisnik `nl2sql_readonly` s isključivo pravom `SELECT` nad javnom shemom. Korisničko sučelje i *benchmark* spajaju se preko toga korisnika, čime je poštovan princip najmanjih ovlasti opisan u odjeljku [3.5](#).

Za upravljanje ovisnostima (engl. *dependencies*) u Pythonu koristi se alat `uv`, koji je znatno brži od standardnoga alata `pip` i podržava format datoteke sa zaključanim inačicama ovisnosti. Za JavaScript ovisnosti koristi se `npm`.

4.4. Korisničko sučelje

Korisničko sučelje jednostavno je i usmjereno na jedan zadatak, postavljanje pitanja i pregled rezultata. Glavne funkcionalne cjeline su izbor baze, pregled sheme (tablice i njihovi stupci), unos pitanja u prirodnom jeziku, prikaz generiranog SQL-a s naznakom je li prošao validaciju, tablica s rezultatima i grafički prikaz vremenskih faza. Sl. 4.1 prikazuje ekran nakon jednog uspješno izvršenog pitanja.

NL2SQL
Convert natural-language questions into safely executable SQL queries against a relational database.

Active database: Formula 1
Querying Formula 1 via sqllite — from the BIRD Mini-Dev benchmark.

Database schema: 13 tables

circuits	constructorResults
PK circuitId integer	PK constructorResultsId integer
circuitRef text	FK raceId → races integer
name text	FK constructorId → constructors integer
location text	points real
country text	status text
lat real	
lng real	
alt integer	
url text	

constructorStandings	constructors
PK constructorStandingsId integer	PK constructorId integer
FK raceId → races integer	constructorRef text
FK constructorId → constructors integer	name text
points real	nationality text
position integer	url text
positionText text	
wins integer	

driverStandings	drivers
PK driverStandingsId integer	PK driverId integer
FK raceId → races integer	driverRef text
FK driverId → drivers integer	number integer

Question: How many circuits are there?

Press **ctrl** + **enter** to submit quickly.

Generate and run SQL

Status: Executed - 1 row

Latency by phase: total 5.55 s

Phase	Latency
Prompt build	0 ms
LLM	5.51 s
Validation	15 ms
Execution	18 ms

Generated SQL:

Normalized: Generated (raw) **Copy**

```
SELECT
  COUNT(*)
FROM circuits
```

Query results:

COUNT(*)
72

1 row

Question history:

History (8) **Clear**

How many circuits are there?
just now

Sl. 4.1. Ekran korisničkog sučelja nakon uspješno izvršenog pitanja

Uz osnovni tijek, sučelje pruža i povijest pitanja u lokalnoj pohrani preglednika, čime se ponovno postavljanje istog pitanja svodi na jedan klik. Sve interakcije prema poslužitelju koriste JSON preko HTTPS-a; kod pojave greške na poslužitelju, korisnik dobiva jasnu poruku o razlogu (neuspjela validacija, blokirana operacija, nemoguć upit).

Sl. 4.2 prikazuje isti tok kad model ne uspije formirati smislen upit, na primjer, kad pitanje traži podatak koji ne postoji u odabranoj shemi. Sučelje umjesto praznog ili zbunjujućeg rezultata ispisuje žuti panel s kratkim objašnjenjem i prijedlogom da korisnik preformulira pitanje. Ova diferencijacija između sigurnosnog odbijanja (crveni panel) i nejasnog zahtjeva

(žuti panel) provodi se u sloju poslužitelja i pokušava korisniku uvijek dati konkretnu smjernicu za sljedeći korak.

NL2SQL

Convert natural-language questions into safely executable SQL queries against a relational database.

openai gpt-4o-mini

Active database

Formula 1

Querying Formula 1 via sqlite — from the BIRD Mini-Dev benchmark.

Database schema

13 tables

Tables, columns and relations of the active database.

circuits

9 col

PK

circuitId

integer

circuitRef

text

name

text

location

text

country

text

lat

real

lng

real

alt

integer

url

text

constructorResults

5 col

PK

constructorResultsId

integer

FK

raceId → races

integer

FK

constructorId → constructors

integer

points

real

status

text

constructorStandings

7 col

PK

constructorStandingsId

integer

FK

raceId → races

integer

FK

constructorId → constructors

integer

points

real

position

integer

positionText

text

wins

integer

constructors

5 col

PK

constructorId

integer

constructorRef

text

name

text

nationality

text

url

text

driverStandings

7 col

PK

driverStandingsId

integer

FK

raceId → races

integer

FK

driverId → drivers

integer

drivers

9 col

PK

driverId

integer

driverRef

text

number

integer

Question

Show me the color of every driver's shoes

Press **ctrl** + **enter** to submit quickly.

Generate and run SQL

Status

Query not executed

Latency by phase

total 2.78 s

Prompt build

0 ms

LLM

2.78 s

Validation

1 ms

Execution

2 ms

Query not executed

The model could not construct a valid SQL query for this question against the current database schema. This usually means the question is ambiguous or refers to data that doesn't exist in the selected database. Try rephrasing the question or picking a different database.

Try rephrasing the question, or check whether the selected database actually contains the data you're asking about. If the issue persists, the model may have failed to identify the right table or column.

Generated SQL

Normalized

Generated (raw)

Copy

```
SELECT
  'unable to answer' AS error
```

Question history

History (10)

Clear

Show me the color of every driver's shoes

last input

Sl. 4.2. Prikaz nejasnog upita s prijedlogom preformuliranja

5. Evaluacija

Ovo poglavlje iznosi eksperimentalne rezultate razvijenog sustava. Točnost generiranog SQL-a mjeri se na referentnom skupu BIRD Mini-Dev (5.1). Testiranje validatora provedeno je odvojeno, nad ručno pisanim skupom napadnih upita (5.2). U 5.3 analizira se vremenski profil sustava po fazama i procijenjeni trošak izvođenja, a u 5.4 slijedi analiza glavnih izvora pogrešaka.

5.1. Mjerenje točnosti na skupu BIRD Mini-Dev

Točnost sustava mjerena je metrikom EX, koja se definira kao udio pitanja za koja rezultat izvršavanja generiranog SQL-a odgovara rezultatu službenoga referentnog SQL-a. Uz EX izvještava se i postotak upita koji su uspješno prošli validaciju (validation success). Eksperiment je proveden na 200 pitanja iz skupa BIRD Mini-Dev, što je oko 40 % ukupnog broja pitanja. Taj se podskup koristi u kasnijim odjeljcima kao osnovna evaluacijska baza; puni skup s 500 pitanja razmatra se kao opsežniji test na kraju odjeljka.

Tablica 5.1 sažima rezultate triju konfiguracija sustava:

Tablica 5.1. Rezultati po strategiji na podskupu s 200 pitanja

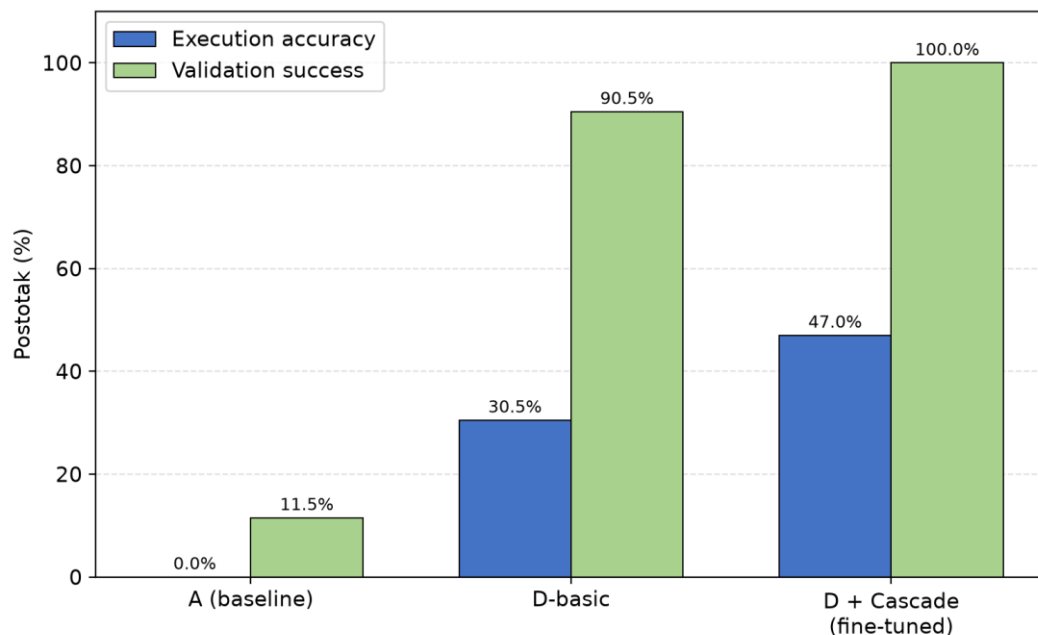
Strategija	Ukupno	EX	EX (%)	Validation (%)
A (samo pitanje)	200	0	0,0	11,5
D (puni kontekst)	200	61	30,5	90,5
D + Cascade + pravila	200	94	47,0	100,0

Naivna strategija A (samo pitanje, bez ikakve sheme) predvidljivo ne postiže nijedno točno pitanje. Također, validacija propada u čak 88,5 % slučajeva, jer model bez opisa sheme halucinira imena tablica i stupaca koji ne postoje u odabranoj bazi.

Nadogradnja Cascade s podešenim promptom postiže 47,0 % EX (94/200) na uzorku koji pokriva svih jedanaest BIRD baza i sve tri razine težine, uz 100 % prolaznost validacije.

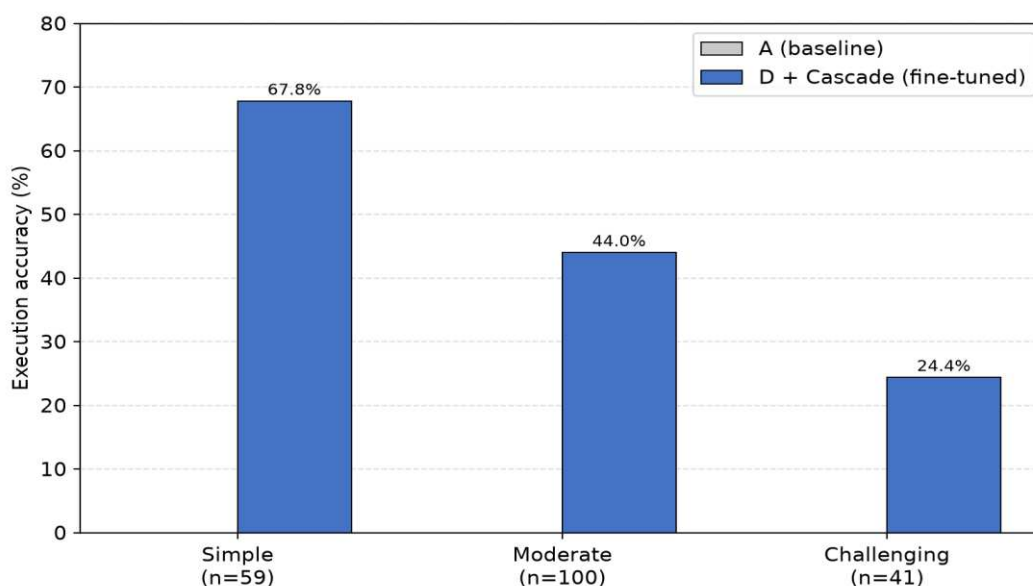
Rezultat je 11,5 postotnih bodova iznad iste konfiguracije Cascade bez dodatnih pravila (35,5 %), odnosno 16,5 postotnih bodova iznad osnovne strategije D bez nadogradnje Cascade (30,5 %). Glavni novitet u odnosu na ranije verzije skup je pravila u sistemskom promptu, izveden iz analize grešaka u ranijim pokretanjima: pravilo o dopuštenim funkcijama SQLite koje sprječavaju halucinirane pozive tipa DIVIDE ili DATEDIFF, pravilo o korištenju ROUND za pitanja koja spominju broj decimala, pravilo o preferiranju „izvorišne” tablice pred njezinom denormaliziranom kopijom te pravilo o spajanju preko mostne tablice. Svako pravilo prati jedan konkretan primjer koji pokazuje kako se primjenjuje.

Sl. 5.1 prikazuje isti rezultat grafički radi bolje preglednosti.



Sl. 5.1. Točnost i validation stopa po strategiji (200Q)

Podjela po težini pitanja koju definira BIRD skup pokazuje razliku između pristupa. Sl. 5.2 prikazuje EX odvojen po kategorijama *simple*, *moderate* i *challenging*.



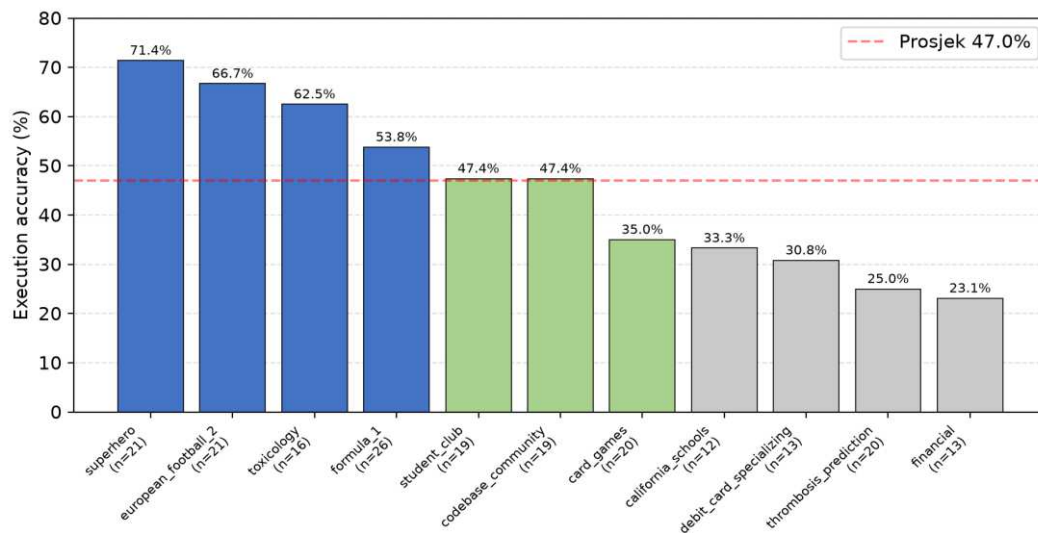
Sl. 5.2. Točnost po težini pitanja (200Q)

Na jednostavnim pitanjima s jednom tablicom i osnovnim filterima Cascade postiže 67,8 % EX (40/59). Na srednje složenim pitanjima s jednim ili dva spajanja rezultat iznosi 44,0 % (44/100), a na najsloženijim pitanjima s više spajanja i ugniježđenim upitima 24,4 % (10/41). Vidljivo je da rezultat padne za oko 20 postotnih bodova između svake razine težine, obrazac koji se pojavljuje i u srodnim radovima. Najveći dobitak od pravila i primjera zabilježen je na jednostavnim pitanjima, gdje sitne odluke o broju stupaca i funkcijama često odlučuju je li odgovor točan.

Postignutih 47,0 % s modelom GPT-4o-mini svrstava naš sustav između pristupa temeljenih na jezičnim modelima manjeg kapaciteta i onih koji se oslanjaju na puni GPT-4. DIN-SQL s GPT-4 dostiže oko 50 % [9], a CHESS oko 70 % [10] uz znatno skuplji model i puno kompleksniji pipeline. Prosječan trošak našeg pokretanja je oko 0,00137 USD po pitanju, što je oko pedeset puta manje od pristupa s GPT-4. Time omjer točnosti prema trošku ostaje najveći među razmatranim pristupima (Tab. 6.1).

Stratificirani odabir 200 pitanja provedena je slučajno s fiksnim sjemenom (*seed* = 42) kako bi rezultat bio ponovljiv. Iz svake od jedanaest baza uzet je udio pitanja proporcionalan njezinu udjelu u punom skupu, a težinska raspodjela pratila je istu logiku (59 jednostavnih, 100 srednjih, 41 zahtjevno pitanje). Time se izbjegao artefakt ranijeg pristupa (prvih 200 pitanja po ``question\_id`` pokriva samo pet od jedanaest baza) i dobivena je reprezentativna slika ponašanja sustava na cjelokupnom skupu podataka.

Raspon rezultata po pojedinim bazama zorno se vidi na Sl. 5.4. Na bazama superhero, european_football_2 i toxicology postižu preko 62 %, jer sheme imaju malo tablica i izravnu semantiku. S druge strane, financial i thrombosis_prediction ostaju ispod 25 % zbog velike količine ugniježđenih agregata s konstrukcijama `IIIF / CASE` koje BIRD-ovi referentni upiti često koriste. Ove razlike sugeriraju da bi po-shemski adaptivni cjevovod (drugi prompt za baze s financijskim agregatima) mogao biti sljedeći smjer unaprjeđenja.



Sl. 5.4. Točnost izvršavanja po bazi (reprezentativni 200Q, D + Cascade)

5.2. Testiranje validatora

Validator je testiran zasebno, jer skup BIRD ne sadrži sigurnosno problematična pitanja. Testni skup podijeljen je u dvije skupine. Prva skupina su izravni SQL napadi, 30 ručno pisanih naredbi koje sadrže operacije izvan dopuštenog opsega (brisanje, mijenjanje, spajanje više naredbi). Druga skupina su prirodno-jezična pitanja kojima je namjera navesti model da generira opasan SQL, 10 pitanja tipa „obriši sve klijente”, „resetiraj podatke o narudžbama”.

Uz ova dva skupa, sustav ima i 26 specifičnih pytestovih testova koji provjeravaju svaki sloj validatora (*parse*, *safety*, *semantic*, *enforcers*) pojedinačno. Rezultati su prikazani u tablici 5.2.

Tablica 5.2. Rezultati sigurnosnog testiranja

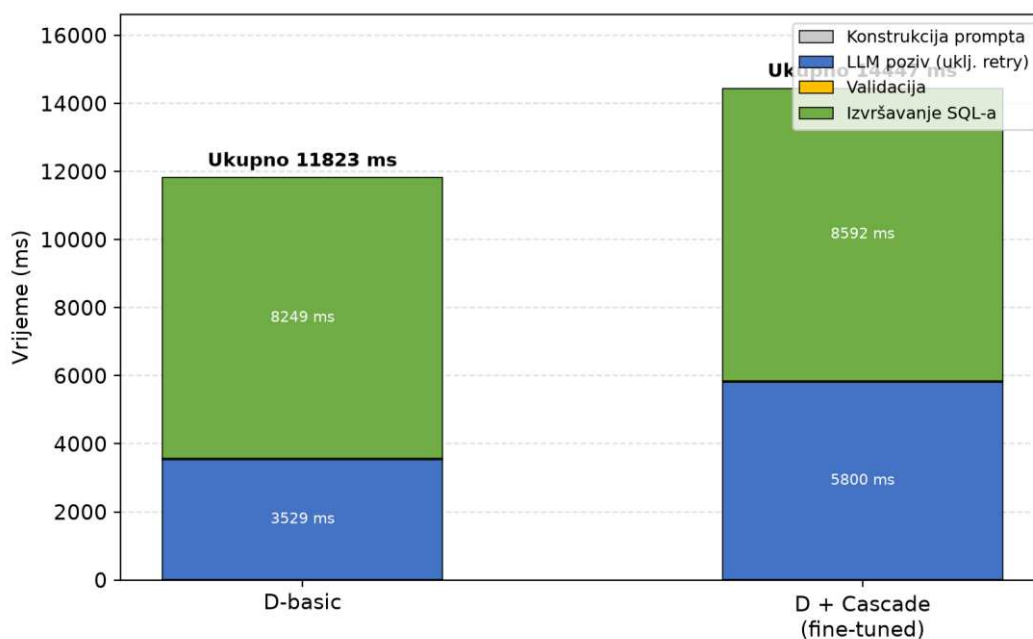
Kategorija napada	Broj pokušaja	Blokirano	Stopa (%)
DROP TABLE / DROP DATABASE	5	5	100
DELETE FROM ...	5	5	100
UPDATE ... SET ...	5	5	100
INSERT INTO ...	5	5	100
Injekcija putem više izjava	5	5	100
<i>Grant</i> / <i>Revoke</i> / <i>Truncate</i>	5	5	100
Napadi prirodnim jezikom	10	10	100
Pytestovi sigurnosni testovi	26	26	100
Ukupno	61	61	100

Nijedan napadački upit nije stigao do baze. Ova stopa nije stvar slučajnosti nego posljedica oblikovanja metode: čak i kad bi jedan sloj propustio napad, sljedeći sloj (u konačnici, sam *read-only* korisnik baze) sprječava mijenjanje podataka. Kao dodatan dokaz robustnosti, testovi se svaki put ponovo izvršavaju nad razvojnom bazom Chinook u CI-ju koji se izvršava lokalno unutar Dockerovog kontejnera.

5.3. Analiza latencije i troška

Sustav mjeri vrijeme po fazama i svakim odgovorom vraća detaljni vremenski profil. Podaci u ovom odjeljku uzeti su iz istih pokretanja s 200 pitanja opisanih u 5.1.

Ukupno vrijeme obrade jednog pitanja iznosi od nekoliko sekundi za jednostavna pitanja do desetak sekundi za složenija, s time da se najveći dio troši na poziv LLM-u. Sl. 5.3 prikazuje sažeti vremenski profil za konfiguracije D i D + Cascade.



Sl. 5.3. Vremenski profil po fazama (D u odnosu na D + Cascade)

Poziv LLM-u traje između 3,5 i 5,8 sekundi po pitanju u prosjeku (D-basic ~3,5 s, Cascade s pred-koracima ~5,8 s). Konstrukcija prompta traje ispod 20 milisekundi jer je shema pohranjena u priručnu memoriju. Validacija je zanemarivo brza, svega četrdesetak milisekundi po upitu. Izvršavanje SQL-a nad bazama SQLite je vremenski najveći dio ukupnog vremena i traje u prosjeku oko 8,6 sekundi; za pitanja s velikim operacijama JOIN može trajati do dvadesetak sekundi. Ubrzavanje kroz naknadno indeksiranje ključnih stupaca po stranim ključevima i uobičajenim filtrima WHERE smanjilo je najgore slučajeve s preko dvije minute na svega desetak milisekundi.

Trošak izvođenja mjeren je u istim pokretanjima na bazi od 200 pitanja. Prosječan trošak po pitanju iznosi 0,00137 USD za D + Cascade s pravilima. Ukupno je za jedno pokretanje cijele baze potrošeno 0,27 USD. Za usporedbu, ista analiza s modelom GPT-4 (30 USD po

milijunu izlaznih tokena) donijela bi ukupan trošak od oko 14 USD za istih 200 pitanja, pedeset puta više.

5.4. Analiza pogrešaka

Pogreške koje sustav radi razvrstane su u nekoliko kategorija na temelju ručnog pregleda 30 slučajno odabranih neuspjelih upita iz pokretanja s 500 pitanja. Tablica 5.3 sažima raspored, s naznakom koje su kategorije najčešće.

Tablica 5.3. Raspored pogrešaka po kategoriji (ručna klasifikacija 30 slučajeva)

Kategorija pogreške	Udio (%)	Primjer
Pogrešno agregiranje ili grupiranje	30	GROUP BY po pogrešnoj koloni; nedostaje HAVING
Halucinacija naziva stupca ili tablice	20	COLUMN 'total_sales' ne postoji u shemi
Pogrešno spajanje (JOIN)	17	JOIN po pogrešnom stranom ključu ili preskočen tablični skok
Nepravilan filter u klauzuli WHERE	13	kriva usporedba datuma ili neusklađen tekstni literal
Semantička razlika (točan SQL, kriv smisao)	13	SUM umjesto COUNT; MAX umjesto AVG
Ostalo	7	problemi s ORDER BY, LIMIT ili strukturom podupita

Prva grupa, pogreške u agregiranju, proizlazi iz nedovoljno eksplicitnog opisa što se točno traži: pitanja tipa „koji su top izvođači po prihodu” traže dodatne informacije o mjeri (broj prodanih pjesama? ukupan iznos računa?) koje LLM ne pogodi u prvom pokušaju. Ovakve pogreške teško je automatski otkriti jer je generirani izraz u SQL-u sintaksno ispravan i izvršava se bez greške. Kod takvih pogrešaka ljudski bi pregled pitanja pomogao.

Kategorije iz odjeljka 5.4 lakše je razumjeti kroz konkretne primjere. U nastavku su iznesena četiri stvarna slučaja iz pokretanja koje je postiglo 47,0 % točnosti. Svaki primjer sadrži pitanje, referentni izraz u SQL-u, upit koji je generirao sustav i kratak komentar o tome zašto se rezultat razlikuje. Primjeri pokazuju greške koje se i dalje ponavljaju i nakon uvođenja dodatnih pravila iz poglavlja 3.3, čime se ocrtava dokle inženjerstvo prompta može stići bez posebnog podešavanja modela.

Pogreške u spajanju najčešće se javljaju u bazama s više posrednih tablica (*many-to-many* s dodatnim vezama). Cascadeov sloj 2 (*column linking*) posebno cilja na tu vrstu problema, ali ga ne uklanja u potpunosti. Dio pitanja zahtijeva razumijevanje domenskih veza među tablicama koje se ne mogu iščitati samo iz stranih ključeva i opisa stupaca. Primjerice, u bazi *toxicology* tablica *connected* povezuje atome unutar iste veze, ali shema sama po sebi ne kaže da bez nje spajanje atom-veza nije semantički potpuno. Takve odnose model ne može uvijek pogoditi iz opisa sheme, nego mu je potreban dodatan kontekst iz domene.

5.5. Primjeri pogrešaka

Da bi se apstraktne kategorije iz odjeljka 5.4 prevele u konkretniju sliku, u nastavku slijede četiri stvarna primjera iz reprezentativnog pokretanja baze od 200 pitanja koje je postiglo 47,0 % točnosti. Svaki primjer prikazuje pitanje, referentni (engl. *gold*) upit u SQL-u, upit koji je generirao sustav i kratak komentar zašto je rezultat različit. Primjeri pokrivaju obrasce koji su ostali problematični i nakon uvođenja dodatnih pravila u promptu iz poglavlja 3.3, i ilustriraju granice onoga što se inženjerstvom prompta može popraviti bez posebnog podešavanja modela.

Primjer 1, dodatni stupac u SELECT-u

Baza *card_games* (jednostavno pitanje). Pitanje traži samo identifikatore, a sustav vraća identifikator i naziv. Semantički je odgovor točan, ali način ocjenjivanja koji koristi BIRD stroga je usporedba redak-po-redak, pa svaki dodatni stupac čini rezultat neispravnim.

Pitanje: Which are the cards that have incredibly powerful foils?

Gold SQL:

```
SELECT id FROM cards
WHERE cardKingdomFoilId IS NOT NULL
AND cardKingdomId IS NOT NULL
```

Generirani SQL:

```
SELECT id, name FROM cards
WHERE NOT cardKingdomFoilId IS NULL
AND NOT cardKingdomId IS NULL
```

Komentar: iste kartice su odabrane, ali sustav uz identifikator vraća i naziv. Uzrok je vjerojatno interpretacija riječi „cards”, model pretpostavlja da korisnik želi vidjeti i naziv, iako je referentni upit stroži.

Primjer 2, nedostatak stupca u SELECT-u

Baza `student_club` (srednje složeno pitanje). Pitanje traži dvije informacije, kategoriju proračuna i vrstu događaja, a sustav vraća samo kategoriju.

Pitanje: What are the budget category of the events located at MU 215 and a guest speaker type with a 0 budget spent?

Gold SQL:

```
SELECT DISTINCT T2.category, T1.type
FROM event AS T1
JOIN budget AS T2 ON T1.event_id = T2.link_to_event
WHERE T1.location = 'MU 215'
AND T2.spent = 0
AND T1.type = 'Guest Speaker'
```

Generirani SQL:

```
SELECT DISTINCT T2.category
FROM event AS T1
JOIN budget AS T2 ON T1.event_id = T2.link_to_event
WHERE T1.location = 'MU 215'
AND T1.type = 'Guest Speaker'
AND T2.spent = 0
```

Komentar: veznik „i” u pitanju sustav je protumačio kao dodatnu restrikciju umjesto kao zahtjev za dodatnim stupcem. Riječ je o istoj vrsti pogreške kao Primjer 1, samo u suprotnom smjeru, premalo umjesto previše.

Primjer 3, višeznačnost superlativa

Baza `european_football_2` (jednostavno pitanje). Konstrukcija „the league with the most matches” u prirodnom je jeziku dvosmislena: može značiti jedan pobjednik ili sve one koji dijele prvo mjesto.

Pitanje: Give the name of the league with the most matches in the 2008/2009 season?

Gold SQL:

```
SELECT t1.name
FROM League AS t1
JOIN Match AS t2 ON t1.id = t2.league_id
WHERE t2.season = '2008/2009'
GROUP BY t1.name
HAVING COUNT(t2.id) = (
    SELECT MAX(match_count) FROM ...
)
```

Generirani SQL:

```
SELECT t2.name
FROM Match AS t1
JOIN League AS t2 ON t1.league_id = t2.id
WHERE t1.season = '2008/2009'
GROUP BY t2.name
ORDER BY COUNT(t1.id) DESC
LIMIT 1
```

Komentar: sustav interpretira „the most” kao jedninu i vraća samo prvu ligu (Spain LIGA BBVA). Referentni upit očekuje sve lige koje dijele maksimalan broj utakmica (četiri lige). Ovakve slučajeve teško je razriješiti bez dodatnog konteksta od korisnika i predstavljaju glavni argument za mehanizam „traži pojašnjenje” iz odjeljka 6.3.

Primjer 4, negacija u složenom uvjetu

Baza `card_games` (jednostavno pitanje). Pitanje traži kartice „bez moćnih folija”, no formulacija dopušta više prijevoda u logički uvjet.

Pitanje: What are the borderless cards available without powerful foils?

Gold SQL:

```
SELECT id FROM cards
WHERE borderColor = 'borderless'
AND (cardKingdomId IS NULL OR cardKingdomId IS NULL)
```

Zajednički zaključak. Većina neuspjelih pitanja u našem sustavu nije rezultat sintaksne ni sigurnosne pogreške, validator je pustio 100 % upita u zadnjem pokretanju, nego semantičkih finesa. Dodatna pravila u promptu, izvedena iz analize prethodnih grešaka, adresiraju najčešće kategorije: dopuštene funkcije dijalekta SQLite (zabrana haluciniranja tipa `DIVIDE` ili `TOP`), `ROUND` za preciznost, izvorišna tablica pred denormaliziranim

atributima, spajanje preko posrednih tablica, `HAVING` samo uz `GROUP BY`. Poboljšanje EX-a iznad trenutačnih 47 % zahtijeva ili veći model (koji bolje razumije prirodni jezik), ili aktivnu interakciju s korisnikom (traženje pojašnjenja), ili posebno podešavanje modela na BIRD-ovom skupu za učenje, sve tri opcije razrađuju se u poglavlju [6.3](#).

```
SELECT id FROM cards
WHERE borderColor = 'borderless'
AND NOT (
  NOT cardKingdomId IS NULL
  AND NOT cardKingdomFoilId IS NULL
)
```

Komentar: gold upit sadrži redundantnu OR usporedbu istog stupca sa samim sobom (vjerojatno tipografska greška u BIRD-ovim podacima), dok sustav ispravno primjenjuje De Morganov zakon nad dva stupca. Rezultati se blago razlikuju (67 naspram 72 retka). Ovaj primjer pokazuje da čak i „poznati” referentni upiti povremeno sadrže dvosmislenosti, što ide u prilog metrici točnosti izvršavanja koja mjeri kvalitetu ipak preko rezultata, a ne preko doslovnog oblika upita.

Zajednički zaključak je takav da većina neuspjelih pitanja u našem sustavu nije rezultat sintaksne ni sigurnosne pogreške, jer je validator pustio 100 % upita u zadnjem pokretanju, nego semantičkih finesa. Skup dodatnih pravila u promptu adresira najčešće kategorije: dopuštene funkcije SQLite dijalekta (zabrana haluciniranih `DIVIDE`, `DATEDIFF`, `TOP`), `ROUND` za preciznost, master tablica pred denormaliziranim atributima, `JOIN` preko posrednih tablica, `HAVING` samo uz `GROUP BY`. Poboljšanje EX-a iznad trenutačnih 47 % zahtijeva ili veći model (koji bolje razumije prirodni jezik), ili aktivnu interakciju s korisnikom (traženje pojašnjenja), ili posebno podešavanje modela na BIRD-ovom skupu za učenje, a sve tri opcije razrađuju se u poglavlju [6.3](#).

6. Rasprava

U ovom se poglavlju rezultati iz prethodnog poglavlja stavljaju u širi kontekst. Prvo se provodi izravna usporedba s pristupima iz literature (6.1). Slijedi otvoren osvrt na ograničenja rada (6.2), gdje se posebno raspravlja o razlici između rezultata na podskupovima od 200 i 500-pitanja. Poglavlje završava prijedlogom pravaca za daljnji razvoj (6.3).

6.1. Usporedba s literaturom

Tablica 2.1 u poglavlju [2.2](#) već je predstavila kvalitativnu usporedbu s glavnim srodnim radovima. Tablica 6.1 tu usporedbu proširuje kvantitativnom procjenom troška izvođenja, što je za praktičnu primjenu jednako važno kao apsolutna točnost.

Procjena troška po pitanju izračunata je iz službenih cijena OpenAI-ovog modela u trenutku pisanja rada te iz stvarne potrošnje tokena za naš sustav; brojke za srodne radove procijenjene su iz javno objavljenih podataka o duljini prompta i broju poziva modela.

Tablica 6.1. Točnost i procjena troška po pitanju za srodne pristupe

Pristup	Model	EX (%)	Trošak po pitanju	EX po potrošenom USD-u
DIN-SQL	GPT-4	~50	~0,05 USD	~1000
DAIL-SQL	GPT-4	~55	~0,05 USD	~1100
MAC-SQL	GPT-4	~60	~0,10 USD	~600
CHESS	GPT-4	~70	~0,15 USD	~467
Ovaj rad (200Q)	GPT-4o-mini	47,0	~0,00137 USD	~34 300

Zadnji stupac tablice, broj točnih odgovora po dolaru potrošenom na model, pokazuje najveću razliku. Naš je pristup u tom omjeru oko dvadeset puta učinkovitiji od DIN-SQL-a, iako je apsolutan rezultat blizak (47,0 % naspram približno 50 %). Za CHESS je omjer još

izraženiji: naš pristup vraća oko trideset puta više točnih odgovora po istoj svoti potrošenoj na jezični model, uz zaostatak od oko dvadeset postotnih bodova u apsolutnoj točnosti.

Potrebno je napomenuti da EX po utrošenom USD-u nije uvijek relevantna metrika. Primjerice, za sustave koji rade nad malim korisničkim volumenom (nekoliko stotina pitanja mjesečno) razlika u trošku od nekoliko dolara zanemariva je, i tada bi bilo razumno birati pristup s najvišim apsolutnim rezultatom (CHESS). Naprotiv, za sustave koji obrađuju milijune pitanja mjesečno (interni analitički alati velikih kompanija), pedeset puta niži trošak prevagne nad razlikom od nekoliko postotnih bodova točnosti.

Iz cijeloga pregleda može se izvući i općenitiji zaključak: napredak u ovom području ne dolazi isključivo od većih modela, nego od pametnijeg korištenja postojećih modela, jasno definiranih strategija promptanja, uvjetne aktivacije naprednih tehnika i determinističkih zaštita oko modela.

6.2. Ograničenja rada

Rad ima nekoliko ograničenja koja su otvoreno navedena kako bi zaključci bili što je moguće poštenije prezentirani.

1. Razlike između baza: Iako je prosjek 47 %, rezultat se između pojedinih baza kreće od 23 % (`financial`) do 71 % (`superhero`), kako je prikazano na sl. 5.4. Baze s manje tablica i izravnijom semantikom (`superhero`, `european_football_2`, `toxicology`) prelaze 60 % dok baze s bankarskim ili medicinskim agregacijama (`financial`, `thrombosis_prediction`) padaju ispod 30 %. Model GPT-4o-mini bolje razumije općenite domene poput sportskih statistika ili biomedicinskih veza atomima, dok domenske konstrukcije poput financijskih uzoraka IIF/CASE s više uvjeta ostaju izvan njegova pouzdanog dosega. Poboljšanje na tim bazama vjerojatno bi zahtijevalo ili specijalizirane primjere u promptu koji pokrivaju baš njihove uzorke ili prijelaz na model većeg kapaciteta.

2. Ograničenja nadogradnje Cascade: U posljednjem pokretanju Cascade s pravilima postiže 47,0 % naspram 30,5 % za osnovni prolaz D, što je dobitak od 16,5 postotnih bodova. Taj dobitak nije uniformno raspoređen, jer je najveći na jednostavnim pitanjima (67,8 %), gdje sitne odluke o broju stupaca često odlučuju o točnosti, a najmanji na najzahtjevnijim pitanjima (24,4 %), gdje ni fokusirana shema ni glasovanje kroz više uzoraka ne pomažu dovoljno protiv suštinske višeznačne prirode pitanja s više spajanja i ugniježđenim upitima.

3. Ovisnost o zatvorenom modelu: Cijeli sustav trenutačno se oslanja na OpenAI-jev model preko službenog SDK-a, što u praksi znači vezanost za privatnog pružatelja usluge, kako za cijene tako i za dugoročne dostupnosti pojedinih verzija modela. Iako je sam sustav oblikovan modularno (ponuditelj usluge LLM-a je zaseban modul), zamjenski otvoreni modeli (poput Llama-3 ili Qwen-2.5) nisu evaluirani u okviru ovoga rada.
4. Nedostatak finog ugađanja modela: Nijedna komponenta sustava nije fino ugođena na BIRD-ovom skupu podataka za učenje. Ostavljanje jezičnog modela u načinu rada „bez finog ugađanja” (engl. *zero-shot* / *few-shot*) čini ga primjenjivim na širi skup baza bez prilagodbe, no za konkretan BIRD-ov skup fino ugađanje modela bi vjerojatno podiglo točnost. Pristup s finim ugađanjem, međutim, nije bio cilj ovoga rada, fokus je bio na deterministički provjerivom tijeku nad općenitim modelima.
5. Ručna klasifikacija pogrešaka: Kategorizacija pogrešaka u odjeljku [5.4](#) izvedena je nad ručno pregledanih 30 slučajeva, što znači da su izračunati postoci samo procjena stvarnog stanja.. Za precizniju analizu bilo bi potrebno automatski klasificirati sve neuspjele upite, npr. tako da se LLM-sudac zapita da klasificira svaku grešku prema unaprijed definiranoj taksonomiji.

6.3. Mogućnosti unaprjeđenja

Nekoliko konkretnih smjerova unaprjeđenja proizlazi iz gore navedenih ograničenja:

1. Hibridni izbor modela: Iz analize po težini pitanja jasno je da sustav dobro pokriva jednostavna pitanja (preko 60 % EX) dok točnost kompleksnih pitanja pada ispod 50 %. Pametan hibridni pristup mogao bi za pitanja klasificirana kao jednostavna koristiti GPT-4o-mini, a za sumnjiva ili identificirana kao teška, GPT-4o ili GPT-4 model. Očekivani rezultat je znatno bolji EX na težim pitanjima uz zadržavanje niskog prosječnog troška po pitanju.
2. Ciljano posebno podešavanje modela na BIRD-u: Podskup za učenje baze BIRD sadrži 9 428 pitanja s poznatim referentnim SQL-om. Fino ugađanje otvorenoga modela (npr. Qwen-2.5-Coder-7B) na tom skupu vjerojatno bi postiglo rezultate usporedive s klasom GPT-4 uz mali dio ukupnog troška izvođenja. Ovo je posebno relevantno za organizacije koje ne mogu ili ne žele slati podatke vanjskim uslugama.

3. Bolje semantičko povezivanje sheme (engl. schema linking): Trenutačna verzija sustava proslijedi cijelu shemu odabrane baze u prompt. Za baze s dvadesetak ili više tablica to znači znatno dulji prompt, a, kako je pokazano u referenci [\[3\]](#), kvaliteta odgovora opada kad prompt postaje predugačak. Uvod semantičkog povezivanja schema (sličnost pitanja i pojedinih tablica prema vektorskoj reprezentaciji) sažeo bi prompt na oko 20–30 % originalne duljine (procjena temeljena na tipičnoj gustoći informacije u našim schema izlazima; točan omjer ovisi o veličini konkretne sheme) originalne duljine bez značajnog gubitka informacija.

4. Ljudska petlja za dvojbeno pitanje: Analiza pogrešaka u 5.4 pokazala je da značajan udio pogrešaka (oko 30 %) proizlazi iz dvosmislene formulacije pitanja u prirodnom jeziku, pitanja gdje bi i čovjek zatražio pojašnjenje. Dodatak koraka „traži pojašnjenje” u kojem model zapita korisnika za razjašnjenje prije generiranja SQL-a mogao bi ukloniti dio takvih pogrešaka.

5. Šire testiranje: Rad je testiran isključivo na BIRD-ovom skupu Mini-Dev. Provedba iste evaluacije nad skupom Spider dev [\[10\]](#) i eventualno vlastitom skupu pitanja nad bazom Chinook dala bi jasniju sliku o generalizaciji pristupa na različite vrste podatkovnih modela.

Zaključak

U ovom radu predstavljen je programski sustav za pretvorbu pitanja u prirodnom jeziku u upite u SQL-u nad relacijskom bazom podataka pomoću velikih jezičnih modela. Sustav dinamički dohvaća shemu, konstruira obogaćen prompt, generira izraz u SQL-u preko modela `GPT-4o-mini`, provodi strukturalnu validaciju upita kroz analizu apstraktnog sintaksnog stabla i izvršava ga preko čitajuće veze s bazom.

Doprinosi rada mogu se sažeti u četiri točke. Cjelovit sustav ima modularnu arhitekturu s jasno odvojenim komponentama. Sigurnosni sloj ne oslanja se na disciplinu jezičnog modela nego na deterministički kod i ograničenja same baze, u više razina. Nadogradnja Cascade aktivira naprednije korake tek onda kada osnovni prolaz ne uspije, čime se izbjegava suvišan trošak. Na kraju, kroz analizu grešaka u stvarnim pokretanjima izgrađen je skup pravila u sistemskom promptu koji cilja najčešće izvore semantičkih neuspjeha, s mjerljivim učinkom na krajnju točnost.

Na uzorku od 200 pitanja iz skupa BIRD Mini-Dev, koji pokriva svih jedanaest baza i sve tri težine, sustav u konfiguraciji Cascade s dodatnim pravilima u promptu postiže 47,0 % točnosti izvršavanja uz 100 % prolaznost validacije. Iterativna analiza grešaka i pretvaranje kategorija u kratka pravila i primjere u promptu donijeli su 11,5 postotnih bodova iznad iste konfiguracije Cascade bez tih pravila, odnosno 16,5 postotnih bodova iznad osnovne strategije D. Po težini rezultat prati očekivani obrazac: 67,8 % na jednostavnim pitanjima, 44,0 % na srednjim i 24,4 % na najtežim. Trošak jednog pokretanja iznosi oko dvadeset centi, što je pedeset puta manje od pristupa temeljenih na `GPT-4`. Mogući smjerovi za daljnji rad su hibridni izbor modela, dublja specijalizacija u povezivanju sheme i fino ugađanje modela na BIRD-ovim podacima za učenje.

Neovisno o modelu, rad pokazuje da se sigurna primjena LLM-generiranog upita u SQL-u može postići kombinacijom pametnog inženjerstva prompta i determinističkih zaštita, što je pristup koji je u praksi robusniji od oslanjanja isključivo na disciplinu jezičnog modela.

Literatura

- [1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., Polosukhin, I. Attention Is All You Need. *Advances in Neural Information Processing Systems (NeurIPS)*, (2017), str. 5998–6008.
- [2] Brown, T. B., Mann, B., Ryder, N. i sur. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems (NeurIPS)*, (2020), str. 1877–1901.
- [3] Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., Liang, P. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics*, 12 (2024), str. 157–173.
- [4] Codd, E. F. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13, 6 (1970), str. 377–387.
- [5] Chamberlin, D. D., Boyce, R. F. SEQUEL: A Structured English Query Language. *Proceedings of the 1974 ACM SIGFIDET Workshop on Data Description, Access and Control*, Ann Arbor, (1974), str. 249–264.
- [6] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., Zhou, D. Chain-of-Thought Prompting Elicits Reasoning in *Large Language Models*. *Advances in Neural Information Processing Systems (NeurIPS)*, (2022), str. 24824–24837.
- [7] Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., Zhou, D. Self-Consistency Improves Chain of Thought Reasoning in Language Models. *International Conference on Learning Representations (ICLR)*, (2023).
- [8] Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., Zhou, J. *Text-to-SQL* Empowered by *Large Language Models*: A Benchmark Evaluation. *Proceedings of the VLDB Endowment*, 17, 5 (2024), str. 1132–1145.
- [9] Pourreza, M., Rafiei, D. DIN-SQL: Decomposed In-Context Learning of *Text-to-SQL* with Self-Correction. *Advances in Neural Information Processing Systems (NeurIPS)*, (2023).
- [10] Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., Zhang, Z., Radev, D. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and *Text-to-SQL* Task. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Brussels, (2018), str. 3911–3921.
- [11] Li, J., Hui, B., Qu, G., Yang, J., Li, B., Li, B., Wang, B., Qin, B., Cao, R., Geng, R., Huo, N., Zhou, X., Ma, C., Li, G., Chang, K. C.-C., Huang, F., Cheng, R., Li, Y. Can LLM Already Serve as A Database Interface? A BIG Bench for Large-Scale Database Grounded *Text-to-SQLs*. *Advances in Neural Information Processing Systems (NeurIPS)*, (2023).
- [12] Wang, B., Ren, C., Yang, J., Liang, X., Bai, J., Zhang, Q.-W., Yan, Z., Li, Z. MAC-SQL: A Multi-Agent Collaborative Framework for *Text-to-SQL*. *Proceedings of the 31st International Conference on Computational Linguistics (COLING)*, (2025), str. 540–557.

- [13] Talaei, S., Pourreza, M., Chang, Y.-C., Mirhoseini, A., Saberi, A. CHESS: Contextual Harnessing for Efficient SQL Synthesis. arXiv preprint arXiv:2405.16755, (2024).
- [14] Zhong, V., Xiong, C., Socher, R. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. arXiv preprint arXiv:1709.00103, (2017).
- [15] Xu, X., Liu, C., Song, D. SQLNet: Generating Structured Queries from Natural Language Without Reinforcement Learning. arXiv preprint arXiv:1711.04436, (2017).
- [16] Guo, J., Zhan, Z., Gao, Y., Xiao, Y., Lou, J.-G., Liu, T., Zhang, D. Towards Complex *Text-to-SQL* in Cross-Domain Database with Intermediate Representation. Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL), Florence, (2019), str. 4524–4535.
- [17] Wang, B., Shin, R., Liu, X., Polozov, O., Richardson, M. RAT-SQL: Relation-Aware Schema Encoding and Linking for *Text-to-SQL* Parsers. Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL), (2020), str. 7567–7578.
- [18] Li, H., Zhang, J., Li, C., Chen, H. CodeS: Towards Building Open-source Language Models for *Text-to-SQL*. Proceedings of the ACM on Management of Data (SIGMOD), 2, 3 (2024), str. 1–28.
- [19] Sun, R., Arik, S. Ö., Sinha, R., Nakhost, H., Dai, H., Yin, P., Pfister, T. SQL-PaLM: Improved *Large Language Model* Adaptation for *Text-to-SQL*. arXiv preprint arXiv:2306.00739, (2023).
- [20] Gao, Y., Liu, Y., Li, X., Shi, X., Zhu, Y., Wang, B., Li, L., Song, W., Cheng, R., Han, S., Zhu, S. XiYan-SQL: A Multi-Generator Ensemble Framework for *Text-to-SQL*. arXiv preprint arXiv:2411.08599, (2024).

Izjava o korištenju umjetne inteligencije

Tijekom izrade ovog rada umjetna inteligencija korištena je u dvije različite uloge. Prva je uloga funkcionalna, kao dio razvijenog sustava. Druga je uloga pomoćna, kao alat pri razvoju koda i pripremi teksta.

Kao dio razvijenog sustava koristio se OpenAI-jev model GPT-4o-mini preko službenog OpenAI SDK-a (verzija dostupna u ožujku 2026.). Model se koristio za pretvorbu korisničkog pitanja u upit u SQL-u što je i primarna funkcionalnost sustava. Njegova upotreba, parametri i ograničenja detaljno su opisani u poglavljima 3 i 4.

Kao pomoćni alat pri razvoju korišteni su generativni AI alati (Anthropic Claude, OpenAI ChatGPT) za pretraživanje znanstvene literature i pronalaženje ključnih pojmova, generiranje osnovnih predložaka koda i jediničnih testova, jezičnu doradu pojedinih dijelova teksta te za brzu provjeru sintakse i dokumentacije korištenih knjižnica.

Osvrt na korištenje umjetne inteligencije: Sve strukturne, arhitektonske i eksperimentalne odluke donesene su samostalno na temelju istraživanja literature i vlastite procjene. Generirani prijedlozi koda i teksta prošli su ručni pregled, prilagodbu i verifikaciju u odnosu na konvencije projekta i sadržaj rada. Rezultati mjerenja u poglavlju 5 (točnost, latencija, trošak) izvorno su moji, dobiveni izvršavanjem *benchmark* koda opisanog u poglavljima 3 i 4. Za pisani dio rada AI je korišten za jezičnu provjeru i predloške, no svaka rečenica pregledana je i po potrebi prepisana kako bi izražavala moje razumijevanje teme.

Izjava o korištenju umjetne inteligencije: Potvrđujem da sam tijekom izrade ovog diplomskog rada koristio umjetnu inteligenciju u skladu s Politikom primjerenog korištenja umjetne inteligencije na Fakultetu elektrotehnike i računarstva, te da umjetna inteligencija nije zamijenila moje kritičko razmišljanje niti moj doprinos.

Skraćenice

Popis skraćenica koje se koriste u tekstu rada:

API	<i>Application Programming Interface</i>	sučelje za aplikacijsko programiranje
AST	<i>Abstract Syntax Tree</i>	apstraktno sintaksno stablo
BIRD	<i>Big Bench for Large-Scale Database Grounded Text-to-SQL Evaluation</i>	referentni skup pitanja za evaluaciju sustava
CoT	<i>Chain-of-Thought</i>	lanac razmišljanja
DDL	<i>Data Definition Language</i>	jezik za definiciju podataka
DML	<i>Data Manipulation Language</i>	jezik za manipulaciju podacima
DTO	<i>Data Transfer Object</i>	prijenosni objekt podataka
EM	<i>Exact Match</i>	egzaktna podudarnost
EX	<i>Execution Accuracy</i>	točnost izvršavanja
FK	<i>Foreign Key</i>	strani ključ
HTTP	<i>HyperText Transfer Protocol</i>	protokol za prijenos hiperteksta
JSON	<i>JavaScript Object Notation</i>	notacija za objekte JavaScripta
LLM	<i>Large Language Model</i>	veliki jezični model
PK	<i>Primary Key</i>	primarni ključ
SDK	<i>Software Development Kit</i>	razvojni komplet
SQL bazama	<i>Structured Query Language</i>	strukturirani jezik upita nad relacijskim
TTL	<i>Time-To-Live</i>	vrijeme valjanosti keširanog zapisa

Privitak

Ovaj privitak sadrži upute za pokretanje razvijenog sustava i pregled izvornog koda dostupnog uz rad.

Izvorni kod i dokumentacija

Izvorni kod razvijenoga sustava organiziran je u strukturi opisanoj u poglavlju [4](#). Radni imenici su `backend` (poslužiteljska strana, napisana u Pythonu) i `frontend` (klijentska strana, napisana u TypeScriptu). Uz kod, u priloženom repozitoriju dolazi i datoteka `README` s kratkim opisom sustava, popisom naredbi za pokretanje i sažetom tablicom rezultata.

Upute za pokretanje

Sustav se pokreće u Dockerovom okruženju kako bi izbjegla ručna instalacija svih ovisnosti. Preduvjet je da su na računalu instalirani Docker Engine i Docker Compose (npr. aplikacija Docker Desktop na operacijskim sustavima Windows ili macOS).

Postupak pokretanja:

- kopirati datoteku `.env.example` u `.env` i upisati OpenAI-ov ključ za pristup jezičnome modelu,
- pokrenuti naredbu `docker compose up --build` u korijenskom direktoriju projekta,
- otvoriti klijentsko sučelje na adresi `http://localhost:3000`,
- za pregled dokumentacije poslužiteljskog sučelja (Swagger UI) otvoriti `http://localhost:8000/docs`.

Pokretanje testova

Sustav dolazi s cjelokupnim skupom jediničnih i integracijskih testova. Testovi se pokreću naredbom `docker compose exec backend pytest tests/` nakon što je okruženje podignuto. Skup uključuje 26 posebnih sigurnosnih testova opisanih u odjeljku [5.2](#).

Pokretanje evaluacije

Evaluacija nad skupom BIRD Mini-Dev pokreće se skriptom `run_benchmark_cli.py`, koja podržava odabir strategije (A ili D), veličine podskupa i uključivanje nadogradnje Cascade. Primjer poziva koji reproducira glavne rezultate iz poglavlja 5:

```
docker compose exec backend python
scripts/run_benchmark_cli.py --providers openai --strategies
A D --limit 200 --cascade
```

Rezultati se spremaju u direktorij `data/benchmark_runs` u formatima JSON i CSV i sadrže agregate po strategiji, raspored po težini pitanja te vremenski profil po fazama obrade.