

# Upotreba velikih jezičkih modela

## Uvod

Industrija video-igara beleži izuzetan rast tokom poslednjih nekoliko decenija, koji je u velikoj meri podstaknut tehnološkim napretkom i sve većom dostupnošću igara široj publici. Sa porastom upotrebe pametnih telefona i drugih mobilnih uređaja, igranje igara postalo je praktičnije i pristupačnije nego ikada ranije, na šta je industrija odgovorila razvojem raznovrsnijih i sadržajnijih naslova koji zadovoljavaju širi spektar interesovanja, preferencija i tipova igrača.

U mnogim aspektima, savremene video-igre postale su interaktivni filmovi i snažan instrument za pripovedanje, u kojima narativ, dijalog i interakcija NPC-ja sa igračem imaju ključnu ulogu. Tradicionalno, dijalozima NPC-jeva upravlja grananje dijaloga, to je sistem u kome programeri unapred definišu odgovore na pitanja, koji najčešće zahtevaju veliki trud za izradu. Veliki jezički modeli mogu rešiti taj problem jer omogućavaju stvaranje bogatijih i fleksibilnijih narativnih struktura, doprinoseći dinamičnijem i jedinstvenijem igračkom iskustvu. Veliki jezički modeli mogu se primeniti za generisanje dodatnih sadržaja unutar igre, poput dijaloga sporednih neigračkih likova (NPC-jeva).

Potencijalne prednosti ovakvog pristupa su višestruke. Ne samo da bi NPC-jevi pokretani velikim jezičkim modelima mogli da nude jedinstvene i neponovljive dijaloge, već bi igračima omogućili i da sa njima vode razgovore koji u velikoj meri podsećaju na prirodnu ljudsku komunikaciju. Ovakav pristup mogao bi značajno da unapredi i ponovno igranje igre, kao i iskustvo nakon završetka glavne priče.

## Large Language Model

Large Language Model (LLM) predstavlja naprednu veštačku inteligenciju sposobnu za razumevanje i generisanje prirodnog jezika. U kontekstu video igara, LLM se može koristiti za omogućavanje dinamičkog dijaloga sa NPC-jevima, njihovo raznovrsno ponašanje koje zavisi od prethodnih interakcija igrača u svetu, umesto da se oslanja na fiksne skripte. Međutim, uprkos širokim perspektivama, primena LLM-a u igricama je još uvek u začetku i suočena je sa tri ključna tehnička izazova:

- Prvi je adaptacija modela - postoji nesklad između opšteg semantičkog znanja LLM-a i sveta i priče specifične za igricu, što lako može dovesti do nepovezanih dijaloga koji narušavaju osećaj igre.

- Drugi je optimizacija za realno vreme - kašnjenje u zaključivanju LLM-a često je jako dugo, dok igre zahtevaju da kašnjenje odgovora NPC-a bude minimalno.
- Treći je doslednost karaktera – neki LLM-ovi mogu generisati sadržaj koji prevazilazi znanje tog NPC-a. Dodatno, memorijska ograničenost tih modela može otežati praćenje konteksta u dužim interakcijama, što vodi ka generisanju manje relevantnih odgovora.

Neka od rešenja za prevazilaženje ovih ključnih izazova su:

- Fine-tuning - Dodatno treniranje modela na skupu podataka specifičnim za svet igre. Ovaj skup podataka može sadržati lore igre, karakteristike likova, istoriju sveta, dijaloge i druge relevantne informacije. Kroz fine-tuning, model uči da prioritizuje znanje o svetu igre nad svojim opštim znanjem, što rezultuje konzistentnijim i relevantnijim odgovorima.
- Retrieval-Augmented Generation (RAG) - Tehnika koja omogućava modelu da pristupa eksternoj bazi znanja tokom generisanja odgovora. Pre nego što model generiše odgovor, sistem pretražuje relevantne informacije iz baze podataka igre (wiki sveta igre, karakterizacije NPC-jeva, trenutno stanje quest-ova) i uključuje ih u prompt.
- Prompt sa jasnim ograničenjima - Pažljivo dizajniran system prompt koji eksplicitno definiše šta NPC zna, šta ne zna, i kako treba da se ponaša kada mu se postavi pitanje van njegove oblasti znanja.
- Strukturirani system prompt - Detaljno definisanje NPC-a kroz višeslojni prompt koji uključuje: osnovne informacije (ime, uzrast, zanimanje), ličnost i temperament, znanje i veštine, motivacije i ciljeve, stil govora i dijalekt, odnos prema igraču, trenutno emocionalno stanje i relevantne quest informacije. Ovaj prompt osigurava da model konzistentno tumači karakter kroz sve interakcije.
- Upravljanje memorijom razgovora - Implementacija sistema koji održava istoriju interakcija između igrača i NPC-a. Za kraće razgovore, cela istorija može biti uključena u svaki prompt. Za duže sesije, koriste se tehnike koje sumiraju istoriju komunikacije gde se raniji delovi razgovora kompresuju u sažetke, zadržavajući ključne informacije, dok se skorašnji deo razgovora čuva u potpunosti.
- Keširanje i memorija dijaloga - Čuvanje prethodno generisanih odgovora za identična ili vrlo slična pitanja. Ako igrač ponovo pita isto pitanje ili ako drugi igrač postavi istu vrstu pitanja NPC-u, sistem može vratiti keširani odgovor instantno umesto ponovnog generisanja.

## Komponente

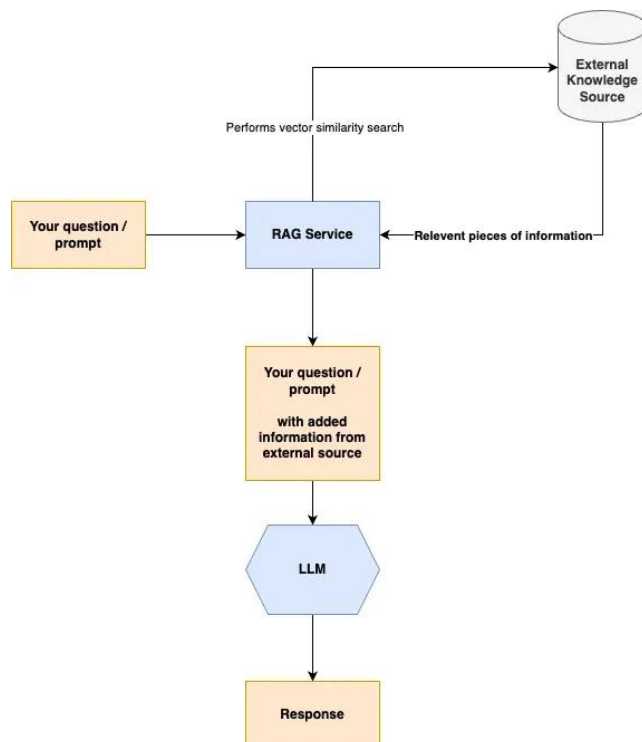
Većina modernih LLM-ova je zasnovana na transformer arhitekturi. Ona omogućava modelu da paralelno obrađuje ceo tekst i dinamički određuje kontekst svake reči pomoću attention mehanizma. Sastoji se od više slojeva koji kombinuju attention i MLP blokove, što modelu omogućava da razume složene jezičke obrasce. Arhitektura velikih jezičkih modela (LLM) se oslanja na specifične komponente koje omogućavaju obradu i razumevanje prirodnog jezika. Osnovne komponente su:

- **Tokeni i tokenizacija** - Osnova svakog velikog jezičkog modela (LLM) je njegova sposobnost da interpretira i upravlja tekstualnim podacima. Tokenizacija je proces pretvaranja ulaznog teksta u manje jedinice poznate kao tokeni. Ovaj korak omogućava modelu da tekst obrađuje na strukturisan način, olakšavajući njegovo pretvaranje u format koji model može razumeti.
- **Embedding sloj** - Nakon tokenizacije, tokeni se mapiraju u vektore - nizove brojeva u višedimenzionalnom matematičkom prostoru. U ovom prostoru, reči sa sličnim značenjem nalaze se blizu jedna druge. Na taj način se tekst prevodi u oblik koji model može da razume i obradi.
- **Attentional layer** - Ovo je srž transformer arhitekture. Model kroz self-attention proračunava važnost svakog tokena u odnosu na sve ostale u sekvenci. Na taj način dinamički određuje kontekst svake reči u rečenici. Na primer, ako imamo rečenicu "Treba da pronađeš drevni simbol", token "simbol" će putem self-attention mehanizma formirati jače veze sa tokenima "drevni" i "pronađeš", što će usmeriti model ka tumačenju "simbol" kao nekog artefakta u igri, dok bi u rečenici "Ne volim matematičke simbole", isti token formirao veze sa tokenom "matematičke" ukazujući na drugo značenje.
- **Transformer blokovi** - LLM se sastoji od više slojeva transformer blokova. Svaki sloj dodatno obrađuje ulazne podatke, usavršavajući modelovo razumevanje i predstavljanje jezika. Više slojeva generalno znači moćniji model, sposoban da obrađuje složene jezičke zadatke.
- **MLP (Multi-Layer Perceptron)** - Unutar svakog transformer bloka, MLP slojevi služe za dodatnu nelinearnu obradu informacija. Oni pomažu modelu da kombinuje i transformiše informacije iz attention sloja, čime se poboljšava kapacitet modela da razume i generiše kompleksan tekst.
- **Reinforcement Learning (RL)** - Neki moderni LLM sistemi koriste Reinforcement Learning kako bi dodatno usmerili model da generiše kvalitetan, bezbedan i korisnički prihvatljiv sadržaj. RL pomaže modelu da optimizuje svoje odgovore prema specifičnim kriterijumima kvaliteta.

- Normalization Layer - Ovi mehanizmi pomažu u stabilizaciji procesa treniranja, sprečavajući model da izgubi vredne informacije iz ranijih slojeva.
- Generisanje izlaza i dekodiranje - Poslednja komponenta se odnosi na generisanje tekstualnih izlaza iz modela. To se postiže kroz dekodiranje, koje konstruiše tekst čitljiv za ljude na osnovu predikcija modela.

## Retrieval-Augmented Generation - RAG

RAG je sistem koji kombinuje pretraživanje baze znanja sa generisanjem teksta, tako da model koristi stvarne informacije iz dokumenata da odgovori na pitanja.



*Funkcionisanje RAG-a*

U najširem smislu, komponente RAG sistema su:

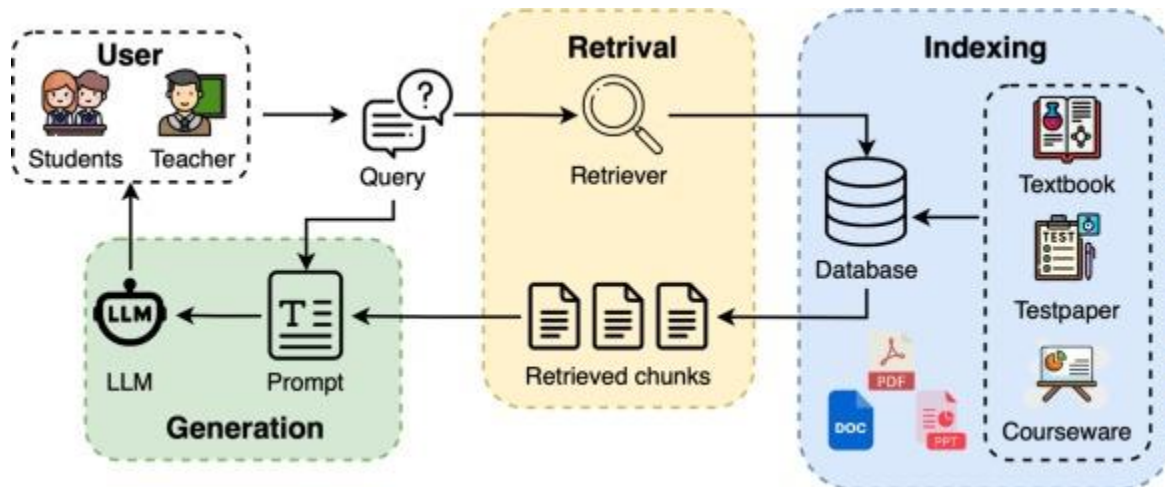
- Baza znanja - Ovo je mesto gde se čuvaju sve referentne informacije. Ona može sadržati sadržaje poput članaka, PDF dokumenata, veb sajtova, priručnika, viki stranica, izveštaja i slično. Ovaj sadržaj se obrađuje i čuva na način koji omogućava lako pretraživanje.
- Retriever (pretraživač) - Retrieval Engine je centralna komponenta RAG sistema koja pretražuje i rangira relevantne podatke na osnovu upita. Sastoji se od dve podkomponente:

- Procesor ulaznog upita (Input Query Processor): Ovaj deo tumači i unapređuje korisnički upit. Analizira upit kako bi razumeo nameru i kontekst.
- Pretraživač (Search Engine): Nakon obrade, upit se prosleđuje pretraživaču koji pretražuje indeksirane podatke, pronalazi podudaranja i rangira rezultate prema relevantnosti.
- Augmentation Engine (mehanizam za obogaćivanje) - On uzima najrelevantnije podatke iz mehanizma za pretragu i dodaje ih u prompt koji se prosleđuje velikom jezičkom modelu (LLM).
- Generator - Završnu komponentu čini mehanizam za generisanje, koji predstavlja napredni LLM. Ovaj deo kreira odgovor kombinujući svoje jezičke sposobnosti sa novopronađenim spoljnim podacima koji su dodati promptu. Integracija tog znanja omogućava generisanje relevantnog odgovora.

RAG funkcioniše na sledeći način:

- Korisnički unos - Proces započinje kada korisnik postavi upit ili unese zahtev u RAG model. Upit može biti pitanje, prompt ili zahtev za informacijama na koji model treba da odgovori.
- Encoding upita - Uneseni upit se transformiše u numeričku reprezentaciju pomoću jezičkog modela kao što su BERT ili GPT.
- Retrieval system - Sistem zatim pronalazi relevantne informacije iz spoljne baze znanja, kao što su skup dokumenata i baza podataka. Pretraga može koristiti tradicionalne metode zasnovane na ključnim rečima ili savremenije metode zasnovane na vektorima. Sistem pretražuje izvor i izdvaja najrelevantnije informacije.
- Rangiranje i filtriranje - Pronađene informacije se filtriraju i rangiraju prema relevantnosti.
- Generisanje kontekstualnih embeddinga - Svaki preuzeti dokument ili deo teksta se takođe pretvara u numerički embedding. Na taj način generativni model može efikasno da uključi pronađene informacije prilikom formiranja odgovora.
- Ugrađivanje konteksta (embedding context) - Preuzeti dokumenti ili njihovi relevantni delovi predstavljaju se kao vektori, što omogućava RAG modelu da razume i obradi njihov sadržaj u istom semantičkom prostoru kao i korisnički upit.
- Spajanje preuzetih informacija - Preuzeti dokumenti se spajaju sa korisničkim upitom i zajedno se prosleđuju modelu kako bi se generisao odgovor. RAG modeli kombinuju originalni kontekst pitanja sa eksternim informacijama kako bi poboljšali tačnost i relevantnost odgovora.
- Generisanje odgovora – Model koristi i korisnički upit i preuzeto znanje kako bi formirao odgovor.

- Isporuka odgovora – Generisani i obrađeni odgovor se šalje korisniku.



Funkcionisanje RAG-a

## Integracija LLM-a u NPC dijalogu

Integracija LLM-a u multiplayer igru zahteva dizajniranu arhitekturu koja balansira između performansi, skalabilnosti i fleksibilnosti u generisanju dijaloga.

### Tipovi modela koji se mogu koristiti

Implementacija LLM-a u video igrama može koristiti tri osnovna pristupa: isključivo lokalne modele, isključivo remote (cloud) modele, ili hibridnu arhitekturu koja kombinuje oba.

#### Lokalni modeli

Lokalni modeli se instaliraju direktno na uređaju igrača ili game serveru i izvršavaju se bez potrebe za internet konekcijom. Latencija odgovora zavisi od hardverskih mogućnosti uređaja. Prednosti su te što postoji veća kontrola nad podacima i privatnošću i nema potrebe za internet konekcijom. Mane su povećana instalacija igre, sistem zahteva značajane hardverke resurse, posebno GPU ili CPU procesorsku moć i RAM memoriju i kvalitet odgovora nije na nivou većih cloud modela.

#### Remote modeli

Remote modeli su hostovani u cloud-u i pristupa im se preko API poziva. Latencija odgovora zavisi od izabranog modela, trenutnog opterećenja servera i kvaliteta internet konekcije. Prednosti su kvalitet odgovora zahvaljujući ogromnim modelima treniranim na velikim skupovima podataka kao i to da sistem ne opterećuje igrač jer se sva obrada odvija u cloud-

u, što znači da nema specijalnih hardverskih zahteva. Ovaj tip modela je korisniji za multiplayer igre u odnosu na lokalni, jer one već zahtevaju internet konekciju.

## Hibridni model

Hibridni pristup kombinuje lokalne i remote modele, birajući koji model koristiti na osnovu konteksta, dostupnosti resursa i prioriteta. Ovaj pristup je najfleksibilniji i omogućava balansiranje između kvaliteta, brzine i troškova. U ovom slučaju, sistem bi trebao da sadrži centralnu komponentu koja analizira svaki zahtev i donosi odluku o tome da li bi koristili lokalni ili remote model. Odluke o tome koji se model koristi može biti određen na različite načine, u zavisnosti od situacije (važnost NPC-ja, situacija u igri, broj reči ili slova, tip interakcije...).

## Komponente sistema

**Klijentska strana (Unity Client)** - Unity klijent služi kao interfejs između igrača i backend sistema. Kada igrač inicira interakciju sa NPC-em, Unity:

- Detektuje trigger event (klik na NPC-a, quest interakcija).
- Prikuplja kontekstualni podatke (ID igrača, ID NPC-a, lokacija, quest status...) i proverava da li je poruka prazna i dužinu same poruke.
- Formira zahtev i šalje ga ka Rust serveru preko WebSocket ili HTTP protokola.
- Prima odgovor i renderuje ga u game UI-u (voice synthesis, subtitle sistem).

**Rust Game Server** - Rust server predstavlja centralni deo koji prihvata sve zahteve i upravlja logikom igre:

- Request Handler - Prima zahteve od Unity klijenata i validira ih.
- Router Logic - Odlučuje da li će koristiti lokalni model, remote model ili keširani odgovor.
- Response Manager - Obraduje generisane odgovore, validira ih, i šalje nazad klijentu.

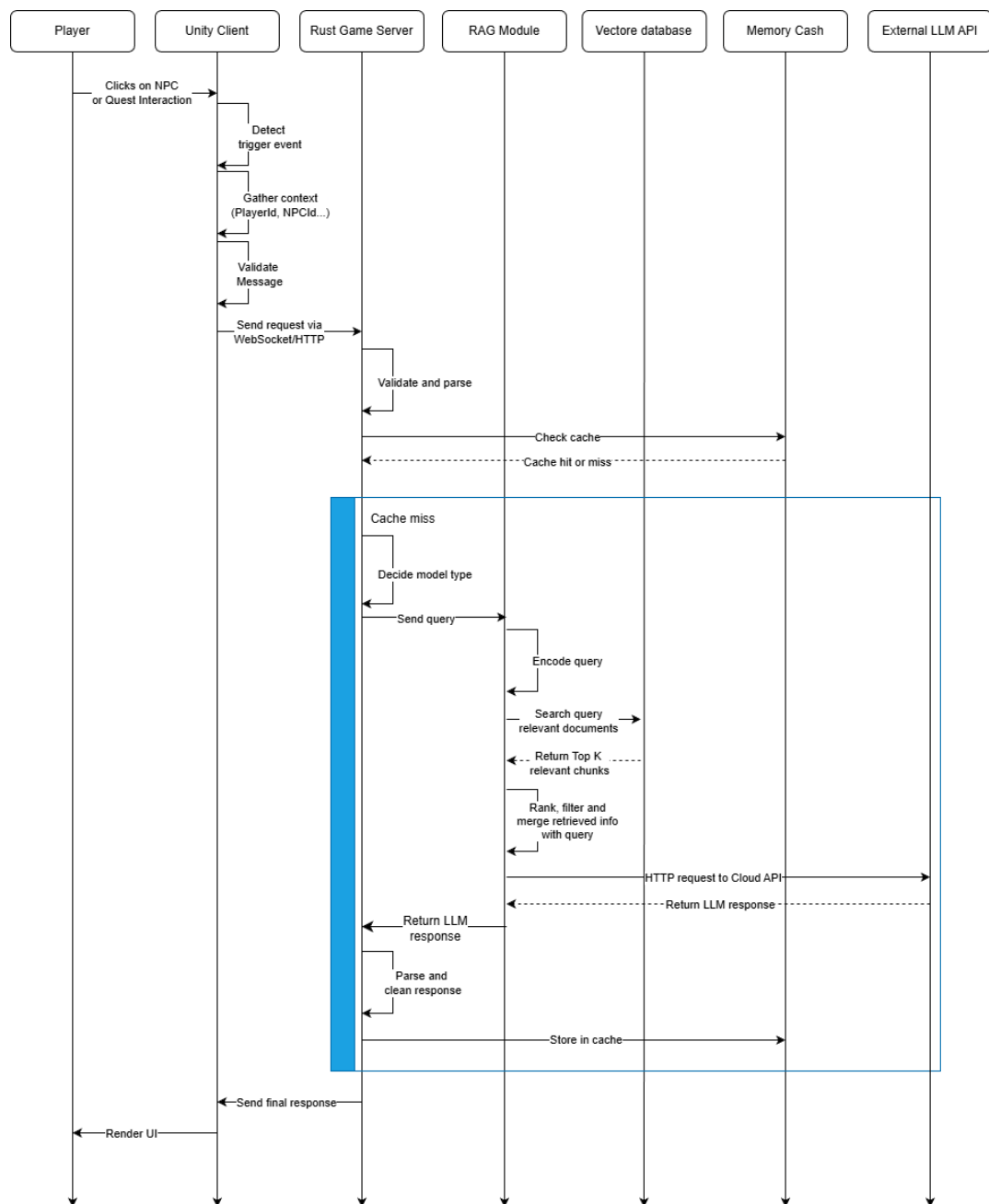
**RAG (Retrieval-Augmented Generation) Modul** - RAG sistem omogućava dinamičko obogaćivanje promptova relevantnim informacijama.

**Vektorska baza** - Vektorska baza se koristi kao deo RAG sistema kako bi NPC-jevi imali pristup relevantnom znanju bez potrebe da se ceo kontekst šalje direktno LLM-u. Umesto klasičnog pretraživanja po ključnim rečima, podaci se čuvaju kao vektori (embeddingi).

**Sistem za indeksiranje** - Indeksiranje je proces pripreme i organizovanja podataka kako bi mogli brzo i efikasno da se pretražuju. Tekstualni podaci se obrađuju, segmentiraju na

manje celine i pretvaraju u embeddinge, koji se zatim čuvaju u vektorskoj bazi zajedno sa odgovarajućim metapodacima (NPC id, quest status...). U MMO okruženju, indeksiranje omogućava da se pretraga ograniči samo na relevantan deo sveta igre, čime se smanjuje latencija i sprečava da NPC koristi informacije koje mu ne pripadaju. Sam proces indeksiranja je asinhron i odvija se van gameplay toka, tako da ne utiče na performanse.

**Keš sistem i memorija** - Sistem za upravljanje memorijom i keširanje obuhvata Conversation History Database (Redis ili PostgreSQL koji čuva istoriju razgovora), Response Cache (Brzi cache (Redis) za često postavljana pitanja) i Summary Engine (Periodično kompresuje duge razgovore u sažetke kako bi se sačuvao kontekst ali smanjio broj tokena).



Dijagram sekvenci

## Prompt

Prompt predstavlja strukturisan tekstualni ulaz (instrukciju, upit ili specifikaciju) koji se prosleđuje LLM-u radi generisanja sadržaja, narativa ili interaktivnih elemenata za NPC-jeve.

## Generisanje Promptova

Generisanje promptova predstavlja kritičan aspekt implementacije LLM-a u video igrima, jer kvalitet i struktura prompta direktno određuju konzistentnost i relevantnost NPC dijaloga.

Prompt se sastoji iz dva glavna dela: system prompt (definiše NPC-a) i user prompt (pitanje igrača).

## Struktura system prompta

System prompt služi kao temelj karaktera NPC-a i mora biti detaljno definisan kako bi model konzistentno tumačio ulogu kroz sve interakcije. Ovaj deo prompta ostaje nepromenjen tokom razgovora i sadrži sledeće elemente:

- Kontekst iz RAG sistema - Pre nego što model generiše odgovor, RAG modul pretražuje vector bazu i vraća relevantna dokumenta. On može da nam vrati sledeće bitne informacije:
  - Osnovne informacije - Ime, uzrast, zanimanje, lokacija i fizički opis NPC-a.
  - Ličnost i temperament NPC-ja, ponašanje, stavovi i slično. Eksplicitno, može se definisati šta NPC zna i šta ne zna. Na primer, srednjovekovni kovač zna o oružju, metalurgiji i lokalnim pričama, ali ne zna ništa o modernoj tehnologiji ili događajima van svog regiona. Uključuje se i instrukcija šta NPC treba da kaže kada mu se postavi pitanje van njegove oblasti znanja.
  - Odnos prema igraču - Dinamički deo koji se menja zavisno od prethodnih interakcija. NPC može biti neutralan prema novom igraču, prijateljski raspoložen prema onome ko mu je pomogao u quest-u, ili neprijateljski prema onome ko ga je prevario. Ovaj deo prompta se ažurira na osnovu reputation sistema igre.
- Player-specific kontekst - Informacije o trenutnom stanju igrača koje utiču na interakciju: nivo igrača, aktivni quest-ovi, items u inventory-ju, lokacija u svetu igre.
- Ograničenja odgovora - Definiše pravila formatiranja i strukture odgovora. Može sadržati i instrukcije šta NPC ne sme nikada da kaže ili radi, dužinu odgovora i slično...
- Prethodna konverzacija sa NPC-jem

## Struktura user prompta

User prompt se generiše za svaki pojedinačni zahtev i sadrži tekst koji je igrač uneo, prethodno očišćen od malicioznih pattern-a.

# Skaliranje

Skaliranje ovakvog sistema kako bi se podržao veliki broj korisnika zahteva specifičan pristup. Klientska strana šalje zahteve preko WebSocket ili HTTP-a ka Rust game serveru, koji deli opterećenje na više instanci. Za svaki zahtev, router logika određuje da li će se koristiti keširani odgovor, lokalni ili remote model (ako koristimo hibridni pristup). Keširanje često postavljanih pitanja značajno smanjuje latenciju i opterećenje modela.

Za obradu zahteva koji zahtevaju LLM, hibridni pristup omogućava dinamičko usmeravanje. Manje zahtevne interakcije (trgovina, jednostavni dijalozi) obrađuju se lokalnim modelima na edge serverima blizu korisnika. Kompleksniji zahtevi (glavni narativi, dinamički questovi) prosleđuju se skalabilnim cloud LLM servisima, koji automatski skaliraju resurse prema potražnji.

Load balancer predstavlja ulaznu tačku koja prima sve zahteve od Unity klijenata i distribuira ih ka pool-u backend servera. Može koristiti različite strategije:

- Round-robin - Zahtevi se distribuiraju ciklično svim dostupnim serverima
- Least connections - Novi zahtev se šalje serveru sa najmanje aktivnih konekcija
- Response time based - Preferira se server sa najbržim istorijskim vremenima odgovora
- Geographic routing - Zahtevi se usmeravaju ka najbližem serveru radi smanjenja latencije

Sistem implementira višeslojni rate limiting kako bi sprečio preopterećenje. Svaki igrač ima definisan maksimalan broj zahteva koji može iskoristiti u nekom periodu. Priority queue sistem kategorizuje zahteve prema važnosti - main story NPC-jevi i quest-critical dijalozi dobijaju visok prioritet i procesuiraju se prvi, dok generic NPC-jevi (seljani, gardovi) čekaju u redu sa nižim prioritetom.

## Reference

1. [https://medium.com/@sean\\_skimmer/npc-gpt-3f4cb5272773](https://medium.com/@sean_skimmer/npc-gpt-3f4cb5272773)
2. <https://jurnal.itscience.org/index.php/brilliance/article/view/6779/4964>
3. <https://arxiv.org/pdf/2404.19721>
4. <https://arxiv.org/html/2504.13928v1>
5. [https://lutpub.lut.fi/bitstream/handle/10024/167809/bachelorthesis\\_Huang\\_Junya\\_ng.pdf](https://lutpub.lut.fi/bitstream/handle/10024/167809/bachelorthesis_Huang_Junya_ng.pdf)
6. <https://www.weka.io/learn/guide/ai-ml/retrieval-augmented-generation/>

7. <https://www.appypieagents.ai/blog/architecture-and-components-of-llms>
8. <https://milvus.io/ai-quick-reference/what-are-the-key-components-of-an-llm>
9. <https://www.meilisearch.com/blog/what-is-rag>
10. <https://customgpt.ai/components-of-a-rag-system/>
11. <https://medium.com/@sandyeeep70/understanding-rag-evolution-components-implementation-and-applications-ecf72b778d15>