Universitatea Națională de Știință și Tehnologie POLITEHNICA București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Chatbot pentru gestionarea completă a interacțiunilor cu clienții

Lucrare de disertație

prezentată ca cerință parțială pentru obținerea titlului de *Master* în domeniul *Inginerie electronică, telecomunicații și tehnologii informaționale*
programul de studii de masterat *Tehnologii multimedia în aplicații de biometrie și securitatea informației (BIOSINF)*

Conducători științifici:
Conf. dr. ing. Horia CUCU
As. drd. ing. Marius-Răzvan MIHAI

Absolvent:
Rareș-Nicolae RODEAN

2026

Universitatea Națională de Știință și Tehnologie POLITEHNICA București
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Programul de masterat **BIOSINF**

Anexa 2

TEMA LUCRĂRII DE DISERTAȚIE

a masterandului **RODEAN V.C. Rareș-Nicolae, 421-BIOSINF**

1. Titlul temei: Chatbot pentru gestionarea completa a interactiunilor cu clientii

2. Descrierea temei:
Proiectul presupune realizarea unui prototip de chatbot conversational rulat local, capabil sa raspunda coerent la intrebari si sa duca la capat cereri simple, folosind documentatie interna ca sursa de adevar. Scopul este obtinerea unor raspunsuri corecte si verificabile, cu timp de raspuns bun, fara dependenta de servicii cloud pentru modelul de limbaj. Proiectul explica modul in care poate fi construit un flux conversational local care combina modele de tip LLM cu un mecanism RAG pentru a reduce numarul raspunsurilor gresite. De asemenea, se urmareste implementarea unei logici de rutare intre doua modele, unul rapid si unul mai capabil, pentru a optimiza viteza de raspuns fara a pierde din calitate in cazurile dificile. In final, prototipul este evaluat prin teste si metrici pe un set de scenarii prestabilite.

3. Contribuția originală:
- Orchestrarea locala a fluxurilor de conversatie in n8n (sau echivalent), incluzand integrarea si coordonarea tuturor componentelor. - LLM local: doua modele selectate si testate comparativ (model rapid, apoi fallback la un model mai capabil). - Implementarea mecanismului RAG pentru documentatie (ingestie, indexare, cautare). - Dezvoltarea unui modul de analiza a performantei, bazat pe un agent separat, pentru evaluarea automata a conversatiilor si calculul indicatorilor AHT, FCR, E2E Rate si Transfer Rate. - Implementarea rutarii intre cele doua modele si definirea conditiilor de fallback. - Testarea si evaluarea prototipului pe un set de scenarii, comparand variantele: cu sau fara RAG si model rapid versus fallback.

4. Resurse și bibliografie:
1. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks", 2020.

2. Jurafsky, D., Martin, J. H., Speech and Language Processing.

3. Documentatia si ghidurile pentru STT/TTS si pentru servirea locala a modelelor alese.

4. Documentatia platformei de orchestrare utilizate (ex. n8n).

5. Documentatia bibliotecilor pentru embeddings si baze de date vectoriale (ex. FAISS, Chroma).

6. Documentatia pentru rularea locala a modelelor de limbaj (ex. framework-uri de servire LLM).

5. Data înregistrării temei: 2026-01-08 17:33:54

Conducător(i) lucrare,
Conf. dr. ing. Horia CUCU

Student,
RODEAN V.C. Rareș-Nicolae

Ing. Razvan MIHAI, Orange Romania

Responsabil program master,
Prof. dr. ing. Dragoş BURILEANU

Decan,
Prof. dr. ing. Mihnea UDREA

Cod Validare: **e2af2acf1c**

Declarație de onestitate academică

Prin prezenta declar că lucrarea cu titlul *Chatbot pentru gestionarea completă a interacțiunilor cu clienții*, prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității Naționale de Știință și Tehnologie POLITEHNICA București drept cerință parțială pentru obținerea titlului de *Master* în domeniul *Inginerie electronică, telecomunicații și tehnologii informaționale*, programul de studii de masterat *Tehnologii multimedia în aplicații de biometrie și securitatea informației (BIOSINF)*, este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare ca referințe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și fac referire la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referire la sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor și măsurătorilor pe care le prezint ca fiind făcute de mine, precum și metodele prin care au fost obținute, sunt reale și provin din respectivele simulări, experimente și măsurători. Înțeleg că falsificarea datelor și rezultatelor constituie fraudă și se sancționează conform regulamentelor în vigoare.

București, 17.06.2026

Absolvent: Rareș-Nicolae RODEAN

(semnătura în original)

CUPRINS

LISTA TABELELOR

LISTA FIGURILOR

LISTA ABREVIERILOR

AHT	*Average Handle Time*
AI	*Artificial Intelligence* (Inteligență Artificială)
API	*Application Programming Interface*
ChromaDB	*Chroma Database* (bază de date vectorială)
CPU	*Central Processing Unit*
CSAT	*Customer Satisfaction Score*
DAG	*Directed Acyclic Graph*
E2E	*End-to-End*
eSIM	*embedded SIM* (cartelă SIM încorporată)
FAISS	*Facebook AI Similarity Search*
FAQ	*Frequently Asked Questions* (Întrebări Frecvente)
FCR	*First Contact Resolution*
GDPR	*General Data Protection Regulation*
GPU	*Graphics Processing Unit*
HNSW	*Hierarchical Navigable Small World*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
KPI	*Key Performance Indicator*
LLM	*Large Language Model*
MoE	*Mixture of Experts*
MRR	*Mean Reciprocal Rank*
n8n	*Nodemation* (instrument de automatizare a fluxurilor)
PDF	*Portable Document Format*
PIN	*Personal Identification Number*
RAG	*Retrieval-Augmented Generation*
SEE	Spațiul Economic European
SIM	*Subscriber Identity Module* (modul de identitate al abonatului)
SLM	*Small Language Model*
SMS	*Short Message Service*
SSD	*Solid State Drive*
TLS	*Transport Layer Security*
TTFT	*Time To First Token*
UUID	*Universally Unique Identifier*

1. INTRODUCERE

Contextul general

În contextul actual al transformării digitale, departamentele de suport și de asistență tehnică se confruntă cu provocări majore: volume mari de întrebări repetitive, timpi de răspuns crescuți și dificultăți în accesarea rapidă a informațiilor din documentația internă. Companiile dețin adesea baze de cunoștințe extinse (manuale de utilizare, proceduri operaționale, politici interne, secțiuni de întrebări frecvente), însă această informație este fragmentată în numeroase documente, ceea ce face dificilă identificarea rapidă a răspunsului corect de către agenții umani sau de sistemele automate.

Modelele de limbaj de mari dimensiuni (LLM) au demonstrat capacități impresionante în generarea de răspunsuri coerente și contextuale, devenind o soluție promițătoare pentru automatizarea interacțiunilor cu clienții. Totuși, majoritatea implementărilor actuale prezintă trei limitări majore care împiedică adoptarea lor pe scară largă în organizații:

Dependența de servicii cloud externe. Soluțiile comerciale precum ChatGPT, Claude sau Gemini necesită transmiterea întrebărilor și contextului către servere externe. Pentru organizații care gestionează date sensibile, cum sunt datele personale și datele de trafic ale clienților din telecomunicații, această dependență ridică probleme serioase de confidențialitate, de conformitate cu GDPR și de control al fluxului de informații. În plus, costurile operaționale recurente pot deveni nesustenabile la scale mari, iar latența depinde de calitatea conexiunii la internet.

Halucinațiile și lipsa de veridicitate. Modelele de limbaj generice, deși capabile să genereze text convingător, tind să producă afirmații inexacte sau complet inventate atunci când nu dețin informația necesară, fenomen cunoscut sub numele de "halucinații". În contextul suportului clienți, un răspuns incorect poate duce la transferuri inutile către operatori umani, la insatisfacția clientului și chiar la implicații legale. Absența unei legături explicite cu documentația internă face imposibilă verificarea surselor și validarea răspunsurilor.

Rigiditatea și lipsa de optimizare a resurselor. Sistemele actuale folosesc de obicei un singur model de limbaj pentru toate tipurile de întrebări, indiferent de complexitate. Acest lucru duce fie la costuri mari de procesare pentru întrebări simple, fie la calitate scăzută dacă se folosește un model mai mic pentru a reduce costurile. Nu există o strategie inteligentă de rutare care să aleagă automat modelul potrivit în funcție de dificultatea întrebării.

Prezenta lucrare abordează aceste trei probleme prin dezvoltarea unui sistem chatbot conversațional **rulat complet local (*on-premise*)**, care combină trei componente inovatoare:

a) **Mecanismul RAG (Retrieval-Augmented Generation)** pentru ancorarea răspunsurilor în documentația internă, reducând riscul de halucinații și asigurând trasabilitatea informației;
b) **Rutare inteligentă între modele**: un sistem de decizie care comută dinamic între un model rapid (pentru întrebări simple) și un model mai capabil (pentru scenarii complexe), optimizând raportul dintre viteză, calitate și cost computațional;
c) **Orchestrare flexibilă prin n8n**: o platformă de automatizare a fluxurilor de lucru care permite integrarea tuturor componentelor, gestionarea fluxurilor conversaționale și colectarea indicatorilor de performanță la nivel de conversație.

Sistemul propus elimină dependența funcțională de servicii cloud pentru generare, recuperare și orchestrare, rulând componentele principale (modelele de limbaj, indexarea vectorială, orchestrarea conversațiilor) pe infrastructura locală a organizației. Această abordare reduce expunerea datelor către servicii externe, oferă costuri operaționale mai previzibile și permite un control mai bun asupra fluxului informațional.

Documentele interne sunt indexate într-un sistem RAG local, astfel încât întrebările utilizatorilor să fie asociate cu fragmente relevante din documentație înainte de generarea răspunsului. Detaliile de fragmentare, reprezentare vectorială, stocare și recuperare sunt descrise în capitolele dedicate arhitecturii și implementării.

Un element suplimentar al lucrării este **evaluarea automată a performanței** prin metrici specifice domeniului de suport clienți, calculate la nivel de conversație. Sunt urmărite atât KPI-uri *obiective* măsurate direct din fluxul de execuție (Average Handle Time, număr de runde, traseul de rutare SLM/LLM, surse RAG consultate), cât și KPI-uri *subiective* estimate de un model evaluator separat (First Contact Resolution și Transfer Rate), din care se derivă indicatorul E2E Rate. Această împărțire permite atât validarea concretă a comportamentului sistemului, cât și optimizarea iterativă a configurațiilor.

Obiectivele proiectului

Scopul principal al lucrării este realizarea unui prototip de chatbot conversațional rulat local, care să elimine dependența funcțională de cloud pentru generarea răspunsurilor și accesarea documentației. În această lucrare, expresia "gestionarea completă a interacțiunilor" se referă la gestionarea fluxului conversațional: primirea întrebării, recuperarea contextului relevant, generarea răspunsului, activarea mecanismului de fallback, jurnalizarea conversației și evaluarea acesteia. Prototipul nu execută operațiuni reale asupra conturilor clienților, precum modificarea unui abonament, suspendarea unei cartele sau inițierea unei portări. Obiectivele specifice includ:

- **Implementarea mecanismului RAG (Retrieval-Augmented Generation):** Pentru a asigura veridicitatea răspunsurilor și a reduce halucinațiile modelului, folosind documentația internă ca sursă de adevăr [14].
- **Rutarea inteligentă:** Dezvoltarea unei logici care să comute dinamic între un model rapid și un model capabil (LLM), optimizând raportul viteză/calitate.

– **Evaluarea automată:** Crearea unui pipeline de analiză care combină KPI-uri obiective măsurate direct din fluxul de execuție (AHT, număr de runde, traseul de rutare SLM/LLM, surse RAG consultate) cu KPI-uri subiective estimate de un agent evaluator (FCR, Transfer Rate, E2E Rate) [7, 20].

Contribuția studentului

Contribuția personală constă în:

– proiectarea și implementarea arhitecturii de orchestrare în platforma *n8n*, incluzând fluxurile conversaționale complete;
– selecția și testarea comparativă a modelelor locale (latență, calitate, suport pentru limba română);
– implementarea pipeline-ului RAG: ingestia documentelor, fragmentarea, indexarea vectorială și mecanismul de retrieval;
– definirea logicii de rutare inteligentă și a mecanismului de *fallback* între modelul rapid și cel capabil, cu memorie conversațională partajată între agenți pentru asigurarea continuității atunci când se activează fallback-ul;
– proiectarea unui mecanism de **selecție dinamică a scenariilor few-shot** prin potrivire de cuvinte cheie, fără cost suplimentar de inferență;
– definirea cadrului de **evaluare automată** cu KPI-uri obiective (calculate din *staticData*) și subiective (estimate de un agent evaluator);
– configurarea infrastructurii *on-premise*: containerizare cu Docker, proxy invers cu Caddy și publicare securizată prin HTTPS.

2. STATE-OF-THE-ART

2.1. Context și tendințe actuale

În ultimii ani, modelele de limbaj au devenit un instrument important pentru automatizarea comunicării cu utilizatorii, în special în scenarii precum suport tehnic, consultanță, căutare în documentație și asistență operațională. Totuși, în multe organizații apare o limitare majoră: informația utilă este stocată în documente interne, iar expunerea acestor date către servicii externe poate ridica probleme de confidențialitate, conformitate și control.

În acest context, soluțiile moderne de tip chatbot tind să combine două direcții:

- folosirea unui model de limbaj pentru generarea unui răspuns coerent și ușor de înțeles;
- integrarea unor mecanisme care asigură că răspunsul este susținut de o sursă (de exemplu documentația internă).

Prin urmare, accentul se mută de la un simplu „chatbot generalist" către un asistent conversațional capabil să ofere răspunsuri corecte, controlabile și verificabile.

2.2. Arhitectura Transformer și modelele de limbaj

Toate modelele folosite în această lucrare au la bază arhitectura Transformer, introdusă în 2017 prin lucrarea *Attention Is All You Need* [21]. Spre deosebire de rețelele recurente folosite anterior, care procesau textul cuvânt cu cuvânt în ordine, Transformer-ul analizează întreaga secvență deodată și se bazează pe un mecanism de atenție (*attention*) prin care fiecare cuvânt poate cântări cât de relevante îi sunt celelalte cuvinte din context. Tocmai această capacitate de a lega informații aflate la distanță în text explică, în bună parte, calitatea răspunsurilor pe care le dau modelele de astăzi și faptul că pot fi antrenate pe corpusuri foarte mari.

Un model de limbaj este, în esență, un sistem antrenat să prezică următorul fragment de text (numit *token*) pe baza celor de dinaintea lui. Văzând volume uriașe de text în timpul antrenării, modelul ajunge să prindă tiparele limbii, o parte din cunoștințele factuale și anumite forme de raționament. Numărul de parametri, adică valorile ajustabile din rețea, este măsura obișnuită a dimensiunii unui model și se exprimă de regulă în miliarde (notat `B`, de la englezescul *billion*). Un model cu mai mulți parametri tinde să se descurce mai bine la întrebări complicate, însă cere mai multă memorie și mai mult timp de calcul.

Această diferență de scară stă la baza distincției, folosită pe tot parcursul lucrării, dintre un

model mic (SLM) și unul mare (LLM). Termenii sunt relativi: un model considerat mic răspunde repede și ocupă puțină memorie, fiind potrivit pentru întrebări directe, în timp ce un model mare oferă răspunsuri mai bune la cereri dificile, dar cu un cost mai mare de timp și de resurse. Pentru rularea locală, unde memoria și placa grafică sunt limitate, se folosesc adesea variante cuantizate ale modelelor: precizia numerică a parametrilor este redusă (de exemplu de la 16 biți la 4 biți), ceea ce micșorează memoria necesară și grăbește inferența, cu o pierdere de calitate de obicei mică. Combinarea unui model mic cu unul mare prin rutare, ideea centrală a sistemului propus, pornește exact de la acest echilibru între viteză și capacitate.

2.3. Modele de limbaj: soluții cloud și rulare locală

În practică, modelele de limbaj pot fi accesate în două moduri principale. Prima variantă este utilizarea serviciilor cloud, care oferă integrare rapidă și performanță ridicată, cu efort redus de configurare. Dezavantajele tipice sunt costurile recurente, dependența de conectivitate și control limitat asupra datelor și jurnalizării.

A doua variantă este rularea locală (*on-premise*). Aceasta oferă un control mai bun asupra datelor și a infrastructurii și permite integrarea în sisteme interne fără a trimite informații sensibile în afara organizației. În schimb, apar constrângeri tehnice reale: resursele hardware necesare, timpul de inferență și necesitatea optimizărilor (de exemplu alegerea unui model mai mic pentru latență sau folosirea unor variante cuantizate).

2.3.1. Strategia de rutare pentru modelele locale

În contextul rulării locale, unde resursele hardware sunt limitate și costul computațional este direct, o abordare folosită frecvent în aplicații reale este utilizarea a două niveluri de modele:

- un model rapid (mai mic), folosit ca primă linie de răspuns pentru toate întrebările;
- un model mai capabil (mai mare), activat doar când modelul rapid semnalează incertitudine sau incapacitate de a răspunde.

Există două paradigme principale pentru o astfel de arhitectură. În *rutarea la intrare*, o componentă separată clasifică întrebarea *înainte* de generare și o trimite direct modelului potrivit; dezavantajul este că necesită un clasificator antrenat sau reguli de complexitate, iar o clasificare greșită trimite întrebarea la modelul nepotrivit. În *rutarea de tip fallback* (în cascadă), modelul rapid generează întâi un răspuns, iar o etapă de evaluare decide dacă acesta este suficient sau dacă cererea trebuie reluată de modelul capabil.

Prezenta lucrare adoptă paradigma de tip fallback. Spre deosebire de o regulă binară simplă (de exemplu, fallback doar la apariția unui cuvânt-cheie), decizia se bazează pe un **scor agregat din mai multe semnale de risc**: incertitudinea exprimată de model, încrederea auto-raportată, calitatea contextului recuperat prin RAG, precum și lungimea și forma răspunsului. Atunci când scorul cumulat depășește un prag, cererea este preluată de modelul capabil, care regenerează răspunsul folosind același context. Mecanismul concret, semnalele luate în calcul și ponderile lor sunt detaliate în secțiunea 3.3.2.

Această strategie are două avantaje practice: (1) nu necesită un clasificator separat de complexitate, iar (2) modelul rapid rezolvă majoritatea întrebărilor fără overhead suplimentar, eliberând rapid resursele GPU. Spre deosebire de soluțiile cloud, unde rutarea este gestionată de furnizor și costurile sunt per-token, în cazul rulării locale costul principal este timpul de ocupare a GPU-ului. Un model capabil poate bloca GPU-ul pentru 10-15 secunde, timp în care nu poate procesa alte cereri. Utilizarea unui model rapid pentru întrebările care pot fi rezolvate direct îmbunătățește astfel throughput-ul sistemului, proporția exactă a cererilor preluate de modelul capabil fiind măsurată experimental (vezi capitolul 6).

Această strategie este specifică rulării locale și este abordarea adoptată în sistemul descris în capitolele următoare.

2.4. Soluții conversaționale existente

Automatizarea conversațiilor cu clienții nu este o idee nouă. De-a lungul timpului s-au folosit mai multe generații de soluții, fiecare cu avantajele și limitele ei, iar o privire asupra lor ajută la situarea sistemului propus și la explicarea alegerilor făcute.

Prima generație a fost cea a sistemelor bazate pe reguli, în care răspunsurile erau scrise de mână și declanșate de cuvinte cheie sau de arbori de decizie ficși. Astfel de soluții sunt previzibile și ușor de controlat, dar devin greu de întreținut pe măsură ce numărul de situații crește, iar utilizatorul trebuie adesea să își formuleze întrebarea fix așa cum a anticipat proiectantul.

A doua generație a adus cadrele bazate pe intenții, precum Rasa sau Dialogflow, în care un clasificator recunoaște intenția din spatele mesajului (de exemplu "vreau să îmi verific factura") și extrage din el datele relevante (sume, date calendaristice, nume de servicii). Răspunsul rămâne însă, în cele mai multe cazuri, pregătit dinainte pentru fiecare intenție. Aceste sisteme se descurcă bine pe domenii bine delimitate, dar cer un efort considerabil pentru etichetarea datelor de antrenare și pentru definirea intențiilor, iar adăugarea unui subiect nou înseamnă de obicei reantrenarea clasificatorului.

Odată cu apariția modelelor de limbaj de mari dimensiuni, a treia generație de asistenți generează răspunsul direct, în limbaj natural, fără a-l alege dintr-o listă fixă. Asistenții comerciali din cloud (de tip ChatGPT, Gemini sau Claude) oferă o calitate ridicată a conversației și o configurare rapidă [8], însă presupun trimiterea datelor către servere externe și costuri care cresc odată cu volumul de cereri. Pentru a păstra această calitate fără a expune datele, multe organizații aleg varianta unui asistent care îmbină un model de limbaj rulat în interiorul organizației cu un mecanism de recuperare din documentația proprie (RAG), descris în secțiunea 2.6. Ecosistemul din jurul acestei abordări s-a maturizat rapid, prin biblioteci care leagă modelele de bazele de date vectoriale și de pașii de orchestrare [13].

Sistemul din această lucrare se încadrează în a treia categorie, dar adaugă două elemente care îl deosebesc de o implementare obișnuită cu un singur model în cloud: rularea complet locală a întregului lanț de inferență și rutarea între un model rapid și unul capabil, în funcție de dificultatea cererii. În felul acesta se păstrează calitatea răspunsurilor generative și controlul deplin asupra datelor, fără a ține tot timpul activ un model mare și costisitor.

2.5. Modele de limbaj utilizate în lucrare

Selecția modelelor de limbaj pentru acest proiect a urmărit identificarea unor candidați care să îndeplinească cerințele specifice: rulare locală eficientă, suport pentru limba română, și un raport favorabil între dimensiune (parametri) și calitate a răspunsurilor. În cadrul lucrării au fost analizate și testate preliminar mai multe familii de modele cu greutăți disponibile public, accesate prin intermediul platformei Ollama [18].

2.5.1. Criteriile de selecție

Modelele au fost evaluate în funcție de următoarele criterii:

- **Dimensiunea modelului:** numărul de parametri influențează direct memoria necesară și viteza de inferență. Pentru rutarea inteligentă, s-au căutat modele de dimensiuni diferite (7B-70B parametri);
- **Suport multilingv:** capacitatea de a genera răspunsuri coerente în limba română, nu doar în engleză;
- **Latența:** timpul necesar pentru generarea primului token (time-to-first-token) și viteza de generare (tokens/secundă);
- **Calitatea răspunsurilor:** coerență, acuratețe și capacitate de a respecta instrucțiunile furnizate în prompt;
- **Compatibilitate cu RAG:** abilitatea de a utiliza eficient contextul furnizat din documentație.

2.5.2. Modele testate

În etapa de explorare preliminară au fost evaluate următoarele modele:

DeepSeek (7B parametri). Un model compact, optimizat pentru inferență rapidă. DeepSeek-7B este un model antrenat de compania chineză DeepSeek AI [3], cu performanțe competitive pentru dimensiunea sa. În testele preliminare, modelul a demonstrat timpi de răspuns foarte buni, însă cu limitări în înțelegerea nuanțelor în limba română și în capacitatea de raționament complex.

Llama 3.1 (8B și 70B parametri). Familia Llama 3.1, dezvoltată de Meta [9], reprezintă una dintre cele mai populare serii de modele cu greutăți disponibile public. Versiunea de 8B parametri oferă un echilibru bun între viteză și calitate, fiind potrivită pentru rolul de "model rapid" în strategia de rutare. Versiunea de 70B parametri oferă performanțe semnificativ mai bune în sarcini complexe, dar necesită resurse hardware considerabile.

Qwen 2.5 (7B, 14B, 32B parametri). Modelele din familia Qwen (dezvoltate de Alibaba Cloud) [19] s-au remarcat prin suport multilingv îmbunătățit și performanțe competitive. Variantele de dimensiuni diferite permit testarea mai multor configurații, de la modele foarte rapide (7B) la modele cu capacități avansate (32B).

Gemma 3 (12B și 27B parametri). Familia Gemma, dezvoltată de Google DeepMind [6], oferă modele optimizate pentru eficiență și cu suport multilingv solid, inclusiv pentru limba

română. Varianta de 12B parametri oferă un echilibru bun între viteză și calitate, fiind potrivită pentru rolul de "model rapid" (SLM) în strategia de rutare, în timp ce varianta de 27B oferă performanțe superioare pe sarcini complexe, la costul unei latențe mai mari, ceea ce o recomandă pentru rolul de "model capabil" (LLM) activat prin fallback.

Mistral (7B parametri) și Mixtral (8x7B parametri). Mistral AI a dezvoltat modele remarcabile prin eficiență [10]. Mistral-7B este un model dense (toți parametrii activi), în timp ce Mixtral folosește arhitectura Mixture-of-Experts (MoE), care activează selective doar o parte din parametri pentru fiecare cerere. Această abordare oferă performanțe apropiate de modele mai mari, menținând latența la un nivel rezonabil.

Toate modelele sunt rulate local prin intermediul platformei Ollama [18], care oferă o interfață unificată pentru încărcarea și inferența modelelor, simplificând atât testarea diferitelor variante, cât și schimbarea configurației în producție.

2.6. Retrieval-Augmented Generation pentru documentație internă

Retrieval-Augmented Generation (RAG) [14] reprezintă o metodă eficientă de îmbunătățire a modelelor de limbaj prin ancorarea răspunsurilor în documente externe, fără re-antrenare. RAG adresează două probleme fundamentale: (1) lipsa de cunoștințe actualizate sau domeniu-specifice, și (2) tendința de a genera informații false (halucinații). Prin integrarea unui mecanism de recuperare din baze de cunoștințe externe, RAG permite generarea de răspunsuri fundamentate pe surse verificabile.

2.6.1. Arhitectura și componentele RAG

Un sistem RAG tipic cuprinde două module principale care lucrează secvențial:

Modulul de Recuperare (Retriever) identifică și extrage fragmentele de text cele mai relevante din baza de cunoștințe, pe baza unei interogări (query). Acesta utilizează reprezentări vectoriale (embeddings) ale documentelor pentru a calcula similaritatea semantică între query și fragmentele indexate [12]. Procesul se bazează pe principiul că texte cu semnificații similare vor avea reprezentări vectoriale apropiate în spațiul metric.

Modulul de Generare (Generator) primește atât interogarea utilizatorului, cât și contextul recuperat, și generează răspunsul final. Modelul de limbaj este instruit prin prompt engineering să utilizeze exclusiv informațiile din contextul furnizat, reducând halucinațiile și asigurând trasabilitatea.

2.6.2. Pipeline-ul RAG

Un pipeline RAG complet parcurge următoarele etape:

1. Ingestia documentelor: Documentele sursă (PDF, DOCX, HTML, text) sunt încărcate și convertite în format text procesabil, păstrând metadatele relevante (titlu, autor, secțiuni) pentru

citare ulterioară.

2. Fragmentarea (chunking): Documentele lungi sunt împărțite în fragmente optimizate pentru procesare. Principalele strategii includ:

- *Fragmentare la lungime fixă* cu suprapunere (overlap) de 10-20% pentru a păstra continuitatea
- *Fragmentare semantică* pe baza structurii documentului (paragrafe, secțiuni)
- *Fragmentare recursivă* pe separatori naturali (rânduri noi, punctuație)

În practică, dimensiunea fragmentelor se alege ca un compromis: fragmentele prea mici pierd contextul necesar, iar cele prea mari diluează relevanța și introduc informație nepotrivită în prompt [5].

3. Generarea embeddings: Fiecare fragment este transformat într-un vector numeric de dimensiune fixă printr-un model de embeddings. În implementarea curentă este utilizat modelul `qwen3-embedding:4b`, rulat local prin Ollama. Modelele de embeddings sunt antrenate prin *contrastive learning* pentru a maximiza similaritatea între texte cu sens similar.

4. Indexarea vectorială: Vectorii sunt stocați într-o bază de date vectorială (de exemplu FAISS [4, 11], ChromaDB sau Pinecone), care permite căutarea eficientă a celor mai apropiați vecini. Algoritmi de căutare aproximativă precum HNSW (*Hierarchical Navigable Small World*) reduc semnificativ timpul de căutare față de o parcurgere exhaustivă, cu o pierdere mică de precizie, controlabilă prin parametri. În prototipul de față este folosită ChromaDB, cu detaliile de configurare descrise în secțiunea 3.3.3.

5. Căutarea semantică: La primirea unei interogări, sistemul generează embedding-ul query-ului folosind același model utilizat pentru documente, calculează similaritatea cosinusoidală cu toți vectorii din index și returnează top-k fragmente cu similaritate maximă. Similaritatea cosinusoidală măsoară gradul de apropiere între doi vectori, cu valori între -1 (complet diferiți) și 1 (identici).

6. Generarea răspunsului: Fragmentele sunt integrate într-un prompt structurat trimis modelului de limbaj, cu instrucțiuni explicite să utilizeze doar contextul furnizat.

Figura 2.1 ilustrează fluxul datelor de la documentele sursă până la generarea răspunsului final.

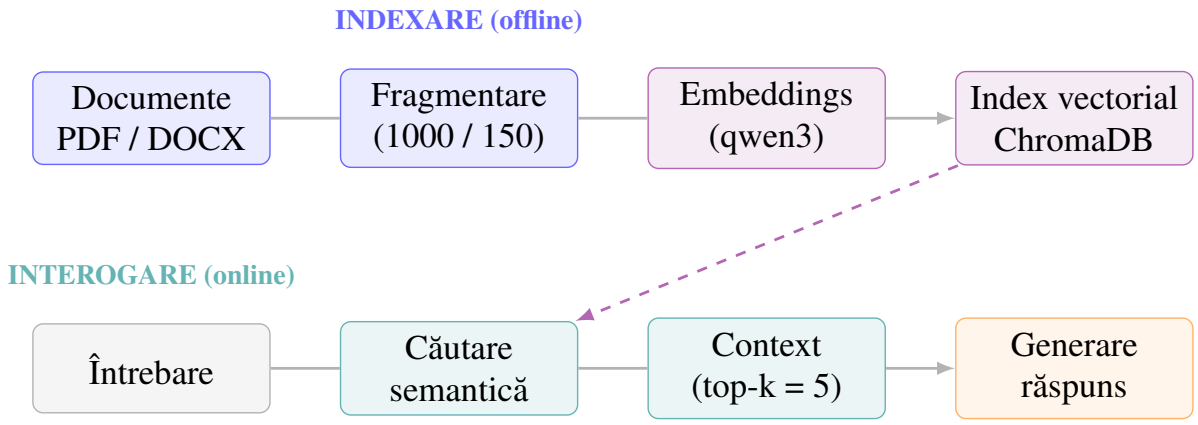

Figura 2.1. Fluxul de date în pipeline-ul RAG.

2.6.3. Tehnici avansate de recuperare

Pipeline-ul descris mai sus, cu o singură căutare semantică și un set fix de fragmente returnate, este varianta de bază a unui sistem RAG. În implementările mature și în literatura de specialitate se folosesc o serie de tehnici suplimentare care îmbunătățesc calitatea fragmentelor aduse, mai ales pe corpusuri mari sau pe întrebări formulate ambiguu [5]. Le prezentăm pe scurt aici, deoarece ele marchează distanța dintre prototipul de față și o soluție extinsă, iar unele revin în discuția limitărilor (secțiunea 3.6).

O primă direcție este căutarea hibridă, care combină căutarea semantică (pe bază de embeddings) cu o căutare lexicală clasică, după potrivirea exactă a cuvintelor. Cele două liste de rezultate sunt apoi îmbinate, o metodă simplă și eficientă fiind *Reciprocal Rank Fusion*, care favorizează fragmentele bine clasate de ambele căutări [2]. Avantajul este că termenii foarte specifici, cum ar fi coduri sau denumiri exacte de servicii, sunt prinși de căutarea lexicală chiar și atunci când embedding-ul nu îi separă bine.

O a doua direcție este reordonarea (*re-ranking*). După prima căutare, care aduce repede un set mai larg de candidați, un model separat reia fiecare fragment și îl evaluează în raport cu întrebarea, apoi rearanjează lista astfel încât fragmentele cu adevărat relevante să ajungă în față. Un astfel de model, antrenat să compare direct întrebarea cu fiecare fragment, dă o judecată mai fină decât simpla similaritate cosinus [17]. Prețul este o etapă de calcul în plus pentru fiecare candidat.

O a treia direcție privește diversitatea rezultatelor. Când mai multe fragmente spun aproape același lucru, contextul trimis modelului se umple de informație repetată. Metode precum *Maximal Marginal Relevance* aleg fragmentele astfel încât fiecare să fie relevant pentru întrebare, dar și diferit de cele deja selectate, acoperind mai multe fațete ale răspunsului [1].

În sfârșit, recuperarea poate fi făcută în mai mulți pași. Dacă prima căutare aduce un context slab, sistemul poate reformula întrebarea sau o poate sparge în întrebări mai mici și poate relua căutarea, în loc să se mulțumească cu primul rezultat. Această abordare iterativă crește șansele de a găsi informația potrivită, cu prețul unor căutări suplimentare și al unei logici de orchestrare mai complicate.

Prototipul de față folosește varianta de bază, cu o singură căutare semantică. Este o alegere potrivită pentru un corpus mic și bine delimitat, format din documentația Orange și YOXO, unde întrebarea și fragmentul căutat folosesc, în general, același vocabular. Tehnicile de mai sus rămân direcții firești de extindere atunci când corpusul crește sau când întrebările devin mai variate.

2.7. Orchestrarea fluxurilor conversaționale

Un sistem conversațional complet implică de regulă mai mult decât apelul unui singur model. În scenarii de suport sau operațiuni, apar cerințe precum:

- validarea întrebării și normalizarea intrării;
- decizii de rutare între modele sau între moduri de lucru (de exemplu cu sau fără RAG);

- acces la surse externe (baze de date, sisteme interne, servicii de autentificare);
- post-procesare: formatare, citare, structurare a răspunsului;
- jurnalizare și colectare de metrici.

Există două direcții principale de implementare. Orchestrarea prin cod oferă flexibilitate maximă, dar costul de dezvoltare și mentenanță poate crește rapid. Alternativ, platformele de automatizare a fluxurilor de lucru permit definirea logicii ca fluxuri de noduri, ceea ce accelerează prototiparea și clarifică dependențele dintre pași. În cadrul prezentei lucrări a fost aleasă platforma n8n [16], ale cărei caracteristici sunt detaliate în secțiunea 3.5.1.

2.8. Evaluarea sistemelor bazate pe LLM și RAG

Evaluarea unui chatbot cu LLM se face, în general, pe două axe: performanță și calitate. Performanța acoperă latența, viteza de generare și consumul de memorie, fiind importantă pentru experiența utilizatorului și pentru costul de rulare. Calitatea acoperă corectitudinea, relevanța, coerența și respectarea instrucțiunilor.

În cazul RAG este utilă o separare clară a evaluării:

- **calitatea recuperării** (retrieval), adică dacă fragmentele recuperate chiar conțin informația necesară;
- **calitatea generării** (generation), adică dacă răspunsul folosește corect fragmentele și evită afirmații nesusținute de context.

Într-un proiect aplicat, evaluarea combină de obicei teste standardizate (prompturi fixe, măsurători repetabile) cu validare pe scenarii realiste din domeniul țintă. Această abordare este importantă deoarece succesul practic al unui chatbot nu depinde doar de un scor numeric, ci și de claritatea răspunsului, de consecvență și de capacitatea de a indica sursa informației atunci când este posibil.

Pe latura de calitate, evaluarea făcută de oameni rămâne cea mai de încredere, dar este înceată și costisitoare, motiv pentru care s-au căutat metode automate. O practică tot mai răspândită este folosirea unui model de limbaj pe post de evaluator, abordare cunoscută drept "LLM ca judecător": un model citește conversația și acordă o apreciere după criterii date, în locul unui evaluator uman [22]. Studiile arată că, pe sarcini bine definite și cu instrucțiuni clare, aprecierile unui astfel de model se apropie rezonabil de cele ale oamenilor, ceea ce îl face util pentru iterare rapidă, deși rămâne supus variabilității și unor înclinații proprii (de exemplu tendința de a favoriza răspunsurile mai lungi).

Pentru sistemele RAG s-au propus și seturi de metrici dedicate, care separă cele două surse posibile de eroare. Pe latura de recuperare se măsoară dacă fragmentele aduse sunt relevante pentru întrebare, iar pe latura de generare se verifică dacă răspunsul chiar se sprijină pe aceste fragmente (fidelitate) și dacă răspunde efectiv la întrebare (relevanță). Aceste măsurători pot fi automatizate tot cu ajutorul unui model de limbaj, fără a cere un set de răspunsuri corecte scrise de mână.

Sistemul din această lucrare adoptă o variantă pragmatică a abordării "LLM ca judecător": un agent evaluator separat estimează indicatorii subiectivi (FCR și Transfer Rate), în timp

ce ceilalți indicatori sunt calculați determinist din datele de execuție. Modul concret în care funcționează acest agent și cum se îmbină cu indicatorii obiectivi este detaliat în capitolul de arhitectură.

3. ARHITECTURA SISTEMULUI ȘI TEHNOLOGII

3.1. Prezentare generală

Arhitectura sistemului a fost proiectată astfel încât componentele principale (orchestrare, rutare, RAG și generare) să poată fi dezvoltate și testate independent. În practică, acest lucru permite înlocuirea unui model, schimbarea strategiei de căutare sau modificarea fluxului conversațional fără a rescrie întregul sistem.

Figura 3.1 prezintă diagrama de ansamblu a sistemului, cu blocurile funcționale principale și legăturile dintre acestea.

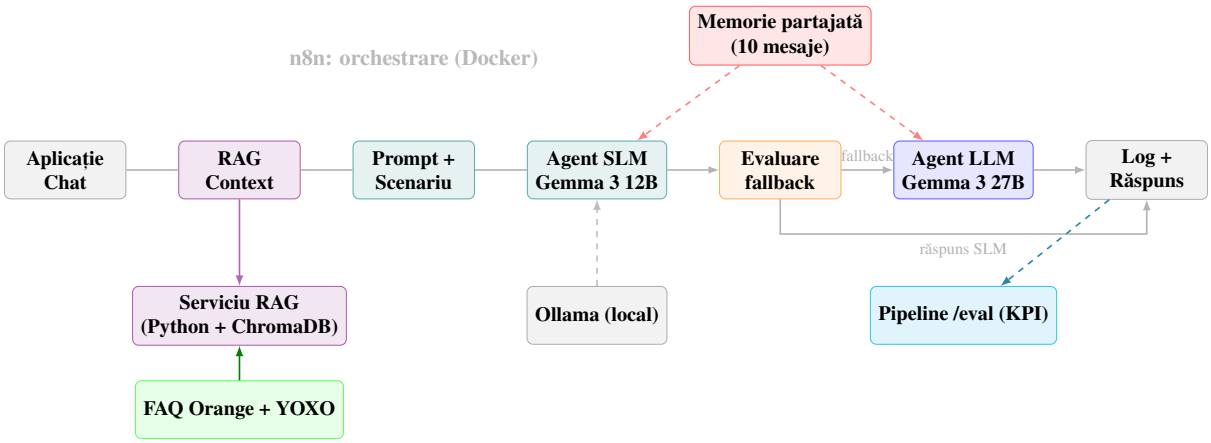

Figura 3.1. Arhitectura generală a sistemului.

3.2. Domeniul de aplicare și corpusul de date

Sistemul a fost construit și testat pe un domeniu concret: suportul pentru clienții a două branduri ale Orange România. Primul este Orange, operatorul de telecomunicații. Al doilea este YOXO, abonamentul de mobil complet digital al Orange, administrat în întregime dintr-o aplicație, care funcționează în rețeaua Orange și care permite activarea prin eSIM, fără perioadă contractuală. Cele două branduri se adresează aceluiași tip de client, dar acoperă subiecte diferite, motiv pentru care documentația lor a fost păstrată separată.

Acest domeniu se potrivește bine cu o abordare locală bazată pe LLM și RAG, din trei motive. În primul rând, întrebările clienților se repetă în jurul unui număr finit de teme (activare,

facturare, roaming, portare), pentru care există deja răspunsuri oficiale, deci recuperarea din documentație are o sursă de adevăr clară. În al doilea rând, interacțiunile cu clienții pot conține date personale, iar rularea locală a modelului evită trimiterea lor către servicii externe. În al treilea rând, răspunsurile trebuie să fie corecte și verificabile, ceea ce favorizează un sistem care își ancorează răspunsul în texte oficiale în loc să se bazeze doar pe cunoștințele învățate de model la antrenare.

Corpusul efectiv folosit este format din două documente, câte unul pentru fiecare brand, rezumate în Tabelul 3.1. Informațiile au fost adunate din secțiunile publice de întrebări frecvente de pe site-urile celor două branduri și organizate într-un format gândit pentru indexare. Documentul YOXO este cel mai cuprinzător și acoperă paisprezece categorii tematice: prezentare generală, activarea abonamentului, administrarea abonamentului și a opțiunilor, aplicația și contul, facturi, plăți și consum, portarea numărului, roaming, eSIM, închiderea și reactivarea abonamentului, magazinul YOXO, programul de recomandare, alte servicii (5G, Wi-Fi, mesagerie), servicii de urgență și securitate, precum și contactul cu serviciul clienți. Documentul Orange este mai restrâns și acoperă, în mare parte, aplicația Orange TV Go: autentificare și acces, control parental, asocierea abonamentului, înregistrarea conținutului și compatibilitatea dispozitivelor.

Tabelul 3.1. Compoziția corpusului de documentație indexat.

Document	**Domeniu acoperit**	**Pagini**	**Volum aprox.**
`orange_rag.pdf`	Aplicație Orange (FAQ)	6	$\sim$10 600 caractere
`yoxo_rag.pdf`	YOXO (abonament mobil digital)	48	$\sim$104 800 caractere
Total	ambele	54	$\sim$115 000 caractere

Ambele documente folosesc același format, gândit special pentru recuperare: informația este împărțită în intrări scurte și de sine stătătoare, fiecare cu o categorie, o întrebare, un set de cuvinte cheie și un răspuns complet. Acest mod de organizare se potrivește direct cu strategia de fragmentare descrisă în secțiunea 3.3.3, deoarece o intrare începe de obicei într-un singur fragment și rămâne inteligibilă scoasă din context, ceea ce reduce riscul ca o căutare să întoarcă o jumătate de răspuns. Tot acest format explică și de ce un corpus relativ mic acoperă bine întrebările uzuale: fiecare intrare este densă în informație utilă, fără text de legătură.

Cele paisprezece categorii ale documentului YOXO corespund îndeaproape situațiilor reproduse în scenariile conversaționale din capitolul 4 (activare și eSIM, portare, roaming, neplată și suspendare, închidere și reactivare), ceea ce asigură că, pentru fiecare scenariu de test, există în corpus informația oficială pe care răspunsul ar trebui să se sprijine. Corpusul reprezintă o fotografie a documentației la momentul colectării și este integral în limba română. El poate fi extins sau reîmprospătat prin reluarea procesului de indexare, fără modificări în restul sistemului, datorită separării dintre ingestie și recuperare.

3.2.1. Delimitarea funcțională a prototipului

În raport cu titlul lucrării, noțiunea de "gestionare completă" este folosită pentru a descrie gestionarea completă a fluxului conversațional și informațional, nu executarea directă a operațiunilor comerciale sau administrative în sistemele interne ale operatorului. Prototipul primește mesajul utilizatorului, recuperează contextul relevant din documentație, generează un răspuns, decide dacă este necesar fallback-ul către un model mai capabil, păstrează memoria conversațională și colectează indicatorii necesari evaluării.

Prototipul nu modifică abonamente, nu suspendă cartele SIM sau eSIM, nu inițiază portări, nu schimbă titulari și nu accesează date personale din sisteme interne Orange. Rolul său este exclusiv conversațional și informativ. Pentru operațiuni sensibile, acțiuni care necesită autentificare sau situații care nu pot fi rezolvate pe baza documentației, sistemul trebuie să îndrume utilizatorul către un agent uman.

3.3. Componentele arhitecturii

Sistemul este compus din câteva componente cu responsabilități distincte, vizibile și în Figura 3.1. Punctul de intrare este o aplicație de chat separată, prin care utilizatorul trimite mesajele și primește răspunsurile; aceasta mediază comunicarea cu orchestratorul n8n, unde se află întreaga logică conversațională (recuperare RAG, selecție de scenariu, rutare cu fallback și evaluare). Generarea propriu-zisă este asigurată de modele rulate local prin Ollama, iar contextul provine dintr-un serviciu RAG dedicat, cu index vectorial în ChromaDB. Subsecțiunile următoare descriu fiecare componentă, începând cu stratul de orchestrare și terminând cu aplicația de chat care leagă utilizatorul de restul sistemului.

3.3.1. Orchestrare cu n8n

n8n este folosit ca strat de orchestrare pentru definirea fluxurilor conversaționale [16]. Avantajul principal este separarea logicii de integrare (ramificații, apeluri către servicii, prelucrare de date, logging) de codul specific RAG sau inferenței modelului. În acest proiect, n8n este utilizat pentru:

- declanșarea fluxului la primirea unei cereri (Chat Trigger sau webhook);
- apelarea serviciului RAG prin HTTP Request pentru recuperarea contextului din documentație;
- **selecția dinamică a scenariului few-shot** prin potrivire de cuvinte cheie, înainte de a invoca modelele (vezi secțiunea 3.3.5);
- generarea răspunsului prin noduri de tip *AI Agent* conectate la modele Ollama locale;
- gestionarea memoriei conversaționale prin **un nod unic Window Buffer Memory**, partajat între ambii agenți (vezi secțiunea 3.3.6);
- evaluarea răspunsului și rutarea de tip fallback prin noduri Code și IF;
- colectarea datelor necesare evaluării automate într-un acumulator separat (*staticData*).

3.3.2. Modulul de rutare

Modulul de rutare implementează o strategie de tip *fallback*: modelul rapid (Gemma 3 12B) generează întâi un răspuns, iar un nod de evaluare (*Evaluare Răspuns*) analizează calitatea acestuia. Dacă răspunsul prezintă semnale de risc, cererea este redirecționată către modelul capabil (Gemma 3 27B), care regenerează răspunsul cu *același* context RAG, același exemplu few-shot și aceeași memorie partajată. Logica este implementată prin noduri native n8n (Code + IF), ceea ce o face transparentă și ușor de ajustat pe baza rezultatelor de testare.

Spre deosebire de o regulă binară simplă (de tipul "comută la modelul mare dacă apare cuvântul ESCALADARE"), decizia de fallback se bazează pe un sistem de scoring ponderat. Răspunsul produs de SLM este analizat din perspectiva mai multor indicatori de risc, fiecare adăugând un număr de puncte la un scor cumulat. Atunci când acest scor atinge sau depășește pragul de fallback (configurat la valoarea 50), răspunsul este considerat insuficient, iar sistemul activează modelul capabil. Avantajul acestei abordări este că mai mulți factori slabi pot declanșa împreună comutarea, chiar dacă niciunul dintre ei nu ar fi fost suficient luat separat: este cazul, de pildă, al unui răspuns scurt formulat pe un context RAG mediocru.

Indicatorii (numiți în continuare *trigger*-e) și ponderile lor sunt sintetizați în Tabelul 3.2. Ei se împart în trei categorii. Prima cuprinde semnalele explicite ale modelului, anume marcajul `ESCALADARE:` și indicatorul de auto-încredere `[INCREDERE: ...]` pe care SLM-ul îl atașează obligatoriu la finalul răspunsului. A doua cuprinde semnalele despre calitatea recuperării RAG, precum absența contextului sau o similaritate semantică slabă, citite din obiectul `rag_signals` returnat de serviciul RAG. A treia cuprinde semnalele lingvistice ale răspunsului în sine: lungimea insuficientă, formulările de incertitudine sau tratarea superficială a unui subiect sensibil.

Tabelul 3.2. Semnalele de fallback și ponderile lor (prag de fallback: 50).

Trigger	Condiție	Puncte
`EXPLICIT_ESCALATION`	Răspunsul conține marcajul `ESCALADARE:`	+100
`EMPTY_RESPONSE`	Răspuns gol după curățare	+100
`UNCERTAIN_RESPONSE`	Formulare de incertitudine ("nu am găsit...")	+45
`SELF_CONFIDENCE_LOW`	Auto-raport `[INCREDERE: SCAZUTA]`	+40
`NO_RAG_CONTEXT`	RAG nu a returnat context relevant	+40
`SENSITIVE_WITHOUT_ACTION`	Subiect sensibil fără pași de acțiune clari	+35
`TOO_SHORT_RESPONSE`	Răspuns scurt raportat la lungimea întrebării	+25
`WEAK_RAG_CONTEXT`	Context prezent, dar similaritate maximă $< 0{,}40$	+20
`GENERIC_SHORT_RESPONSE`	Răspuns scurt și sărac în cuvinte unice	+20
`SELF_CONFIDENCE_MEDIUM`	Auto-raport `[INCREDERE: MEDIE]`	+15
`CLARIFICATION_OVERRIDE`	Răspunsul este o întrebare de clarificare	-50

Tabelul evidențiază două aspecte importante. Marcajul `ESCALADARE:` și răspunsul gol au pondere decisivă (+100 de puncte), ceea ce garantează comutarea către modelul capabil indiferent de ceilalți factori. La polul opus, regula `CLARIFICATION_OVERRIDE` *scade* scorul cu 50 de puncte atunci când răspunsul SLM este, de fapt, o întrebare legitimă de clarificare

adresată utilizatorului. Fără acest corectiv, cererile de clarificare, scurte prin natura lor, ar fi penalizate greșit ca răspunsuri insuficiente și ar provoca un fallback inutil. Atât indicatorul de auto-încredere, cât și textul răspunsului final sunt curățate de orice marcaj intern (`[INCREDERE: ...]`) înainte de livrarea către utilizator, pentru ca aceste artefacte de rutare să rămână invizibile.

3.3.3. Modulul RAG

Modulul RAG conectează întrebările utilizatorului la documentația internă, urmând pipeline-ul descris în secțiunea 2.6 (ingestie, fragmentare, indexare vectorială, căutare semantică). Este implementat ca un serviciu Python separat (FastAPI), apelat de n8n prin HTTP Request pe portul 8100. Serviciul expune câteva endpoint-uri HTTP, dintre care cel principal, folosit în fluxul conversațional, este `POST /context` (returnează contextul recuperat pentru o întrebare); pe lângă el există endpoint-uri pentru ingestia documentelor (`POST /index/file`) și pentru verificarea stării serviciului (`GET /health`).

Baza de cunoștințe a fost construită prin colectarea sistematică a informațiilor publice de pe site-urile Orange și YOXO, în special secțiunile de FAQ (Întrebări Frecvente), care acoperă subiecte precum activarea abonamentelor, portarea numărului, roaming, facturare, eSIM și proceduri de închidere cont. Concret, în index sunt încărcate două fișiere PDF distincte (`orange_rag.pdf` și `yoxo_rag.pdf`), fragmentate și indexate în ChromaDB cu același pipeline. Prin separarea ingestiei de recuperare, componenta poate fi optimizată independent (dimensiune fragmente, overlap, strategie de căutare) fără a modifica restul arhitecturii.

Ingestia și fragmentarea. La ingestie, textul este extras în funcție de tipul fișierului (`pypdf` pentru PDF, `python-docx` pentru DOCX, citire directă pentru text simplu) și apoi fragmentat. Algoritmul de fragmentare nu folosește o tăiere oarbă la lungime fixă, ci alege **primul separator natural disponibil** dintr-o listă ordonată după granularitate descrescătoare (paragraf `\n\n`, rând nou, sfârșit de propoziție, spațiu). Textul este apoi acumulat secvențial: fiecare bucată este adăugată la fragmentul curent atât timp cât nu depășește dimensiunea maximă (`CHUNK_SIZE`= 1000 de caractere); la depășire, fragmentul curent este finalizat, iar următorul *moștenește* ultimele `CHUNK_OVERLAP`= 150 de caractere ale celui precedent, asigurând continuitatea contextului la granițe. Fragmentele sub un prag minim (`MIN_CHUNK_SIZE`= 50 de caractere) sunt eliminate ca nesemnificative.

Fiecare fragment primește un identificator derivat din numele sursei și indexul fragmentului (cu un sufix hash pentru unicitate) și un set de metadate: sursa, indexul, numărul total de fragmente, un hash al fișierului și data indexării. Procesul de ingestie poate fi repetat în siguranță: înainte de inserarea fragmentelor noi ale unei surse, fragmentele vechi ale aceleiași surse sunt șterse din colecție. Astfel, reindexarea unui document modificat nu lasă fragmente orfane în index.

Indexarea, căutarea și conversia de similaritate. Vectorii sunt stocați într-o colecție ChromaDB persistentă pe disc, configurată cu metrica de distanță cosinus (`hnsw:space = cosine`). La interogare, întrebarea este transformată în embedding cu *același* model ca documentele, iar ChromaDB returnează cele mai apropiate `TOP_K`= 5 fragmente. ChromaDB raportează însă o *distanță* cosinus în intervalul $[0, 2]$, nu o similaritate; aceasta este convertită în scor de

similaritate în intervalul $[0,1]$ prin relația:

$$similaritate = 1 - \frac{distanță}{2} \tag{3.1}$$

Fragmentele cu similaritate sub pragul `SIMILARITY_THRESHOLD`$= 0{,}3$ sunt eliminate, astfel încât contextul transmis modelului conține doar potriviri suficient de relevante. Generarea embeddings-urilor se face printr-un apel HTTP per fragment către endpoint-ul `/api/embeddings` al Ollama; pentru corpusul redus al proiectului (două documente FAQ), abordarea secvențială este suficientă, dar reprezintă un punct natural de optimizare prin procesare în loturi pentru corpusuri mari.

Asamblarea contextului și semnalele de calitate. Fragmentele filtrate sunt concatenate de funcția `build_context()`, punctul de intrare apelat de n8n pe endpoint-ul `/context`, într-un text gata de inserat în prompt. Fiecare fragment este prefixat cu un antet care indică sursa și scorul de relevanță (`[Sursa: ... | Relevanta: ...]`), oferind modelului trasabilitate, iar concatenarea se oprește la atingerea unei limite de lungime (4000 de caractere) pentru a evita diluarea prompt-ului. Lista surselor consultate este deduplicată și sortată.

Pe lângă context, funcția returnează un obiect `rag_signals` care cuantifică *calitatea* recuperării și care alimentează direct sistemul de scoring al fallback-ului (secțiunea 3.3.2). Acesta conține: similaritatea maximă și medie a fragmentelor reținute, numărul de potriviri "puternice" (similaritate $> 0{,}5$), un indicator boolean privind existența contextului și numărul de fragmente. Structura este **întotdeauna prezentă**, chiar și când nu se găsește niciun fragment relevant (caz în care câmpurile au valori nule și `has_context` este fals), astfel încât orchestratorul primește mereu un obiect bine formatat, niciodată o valoare lipsă. Separarea dintre contextul furnizat modelului și semnalele despre încrederea în recuperare este motivul pentru care decizia de fallback poate ține cont atât de răspunsul modelului, cât și de soliditatea informației pe care acesta s-a bazat. Fragmentele de cod corespunzătoare (fragmentarea documentelor, conversia distanței în similaritate și construirea obiectului `rag_signals`) sunt redate în Anexa A.

3.3.4. Modulul de generare (LLM/SLM)

Modulul de generare primește întrebarea și, când este cazul, contextul recuperat de RAG. Modelele sunt rulate local prin Ollama și integrate în n8n ca noduri de tip *AI Agent* cu sub-nod *Ollama Chat Model*. În urma testelor comparative descrise în secțiunea 2.5, configurația curentă folosește **Gemma 3 12B** ca model rapid (SLM) și **Gemma 3 27B** ca model capabil (LLM). Fiecare agent este configurat cu un system prompt care instruiește modelul să utilizeze exclusiv informațiile din contextul RAG furnizat și care include exemplul few-shot selectat dinamic (vezi secțiunea 3.3.5).

3.3.5. Modulul de selecție a scenariului (few-shot dinamic)

Imediat înainte de invocarea modelelor de limbaj, fluxul rulează un nod Code dedicat (*Selectare Scenariu*) care **selectează dinamic** cel mai relevant scenariu din cele 7 definite în capitolul 4 și îl injectează ca exemplu *few-shot* în system prompt-ul ambilor agenți (SLM și

LLM).

Mecanismul de selecție este intenționat simplu pentru a evita un cost suplimentar de inferență:

- fiecare scenariu (S01-S07) are asociată o listă scurtă de cuvinte cheie reprezentative (de exemplu pentru S01: `activare`, `abonament`, `esim`, `iphone`, `plan`; pentru S05: `furat`, `furt`, `telefon furat`, `pierdut` etc.);
- textul întrebării și cuvintele cheie sunt normalizate (lower case, eliminarea diacriticelor românești `ă/â/î/ș/ț`) pentru a gestiona variațiile de scriere;
- se calculează scorul de potrivire (numărul de cuvinte cheie comune); scenariul cu scorul cel mai mare este ales doar dacă pragul minim este atins (cel puțin 2 potriviri);
- dacă niciun scenariu nu trece pragul, sistemul nu adaugă exemplu și se bazează exclusiv pe contextul RAG.

Această abordare are mai multe avantaje practice. În primul rând, nu introduce o invocare suplimentară de model pentru clasificarea intenției (cost zero la latență). În al doilea rând, exemplul few-shot oferă modelului un *template* concret de format al răspunsului (lungime, ton, structură) pentru tipul de întrebare detectat, ceea ce îmbunătățește consistența. În al treilea rând, scenariile sunt versionabile (definite explicit în nod) și pot fi extinse fără retraining. Aceeași logică alimentează și cadrul de evaluare automată din capitolul de scenarii. Implementarea acestei potriviri (calculul scorului și pragul de minim două cuvinte cheie) este redată în Anexa B.

3.3.6. Memoria conversațională

Pentru menținerea contextului pe parcursul mai multor schimburi de mesaje, sistemul utilizează **un singur nod nativ *Window Buffer Memory*** din n8n, configurat cu o fereastră de **10 mesaje** (5 perechi întrebare/răspuns).

Decizia de design centrală în acest punct este **partajarea aceleiași instanțe de memorie între ambii agenți** (SLM și LLM), prin conectarea aceluiași nod la portul `ai_memory` al fiecărui agent. Această alegere are o motivație directă, derivată din strategia de fallback: atunci când răspunsul SLM este preluat de LLM, modelul capabil trebuie să aibă acces la întreg istoricul conversației, inclusiv la rundele răspunse anterior de SLM. O memorie separată pentru fiecare agent ar produce două istorii divergente, iar continuitatea conversației s-ar pierde la prima comutare. System prompt-urile celor doi agenți instruiesc explicit modelul să *ignore* mesajele de tip `ESCALADARE:` găsite în istoric (acestea sunt artefacte ale rutării și nu răspunsuri reale către utilizator).

Memoria este distinctă de mecanismul de stocare folosit pentru evaluarea automată (*staticData*): prima ține contextul lingvistic pe care îl primesc modelele, a doua acumulează metadate operaționale (rutare per rundă, surse RAG, timestamps) necesare KPI-urilor.

3.3.7. Colectarea de date și evaluarea automată

Pe lângă livrarea răspunsului, fluxul colectează la fiecare rundă informații operaționale: mesajele schimbate, modelul utilizat efectiv (SLM sau LLM, după rezultatul fallback-ului),

scorul și motivele de fallback per rundă, sursele RAG consultate și marcaje temporale. Aceste date sunt acumulate într-un dicționar indexat după `sessionId` în mecanismul *Static Data* (`$getWorkflowStaticData('global')`) al n8n, care supraviețuiește execuțiilor individuale ale fluxului. Acumularea se face în nodul *Log și Detecție Final*, parcurs la fiecare rundă a conversației normale.

Declanșarea evaluării este **inițiată de utilizator**, nu automată: interfața de chat expune un buton *Finalizează* care, la apăsare, trimite către orchestrator comanda specială `/eval` (detalii despre interfață în secțiunea 3.3.8). În flux, un nod IF dedicat (*Verifică Eval*) inspectează fiecare mesaj primit: dacă mesajul este comanda `/eval`, cererea este rutată direct către pipeline-ul de evaluare, ocolind complet recuperarea RAG și agenții conversaționali; altfel, mesajul urmează fluxul normal de răspuns. Această separare garantează că pasul de evaluare nu interferează cu conversația și nu consumă resurse de inferență decât atunci când este cerut explicit.

Pipeline-ul de evaluare este format din trei noduri:

- *Pregătire Evaluare* reconstruiește textul conversației din `staticData`, calculează KPI-urile **obiective** (numărul de runde, AHT în secunde ca diferență dintre primul și ultimul marcaj temporal, secvența de rutare per rundă cu scorurile de fallback, lista surselor RAG consultate) și șterge sesiunea pentru a elibera memoria;
- *Agent Evaluator*, o instanță separată de Gemma 3 27B cu `temperature=0.1` pentru determinism, primește întreaga conversație și estimează KPI-urile **subiective**, returnate într-un format strict prestabilit;
- *Format Rezultat Final* parsează ieșirea evaluatorului, derivă indicatorii calculați și asamblează raportul final, livrat atât ca text lizibil, cât și ca obiect structurat (`kpis`).

KPI-urile produse de pipeline sunt:

- **FCR** (*First Contact Resolution*, valori DA/NU/PARȚIAL): a fost problema clientului rezolvată complet la primul contact? Indicator estimat de evaluator.
- **Transfer Rate** (DA/NU): a fost necesar transferul către un agent uman? Indicator estimat de evaluator. El vizează *exclusiv* predarea conversației către un operator uman și este distinct de fallback-ul intern SLM $\rightarrow$ LLM, care rămâne o decizie de rutare invizibilă pentru utilizator.
- **E2E Rate** (*End-to-End*, DA/NU): rezolvare integrală fără intervenție umană. Este un indicator *derivat* prin regulă, evaluat ca DA doar atunci când FCR=DA și Transfer Rate=NU.
- **AHT**, **numărul de runde**, **traseul de rutare** și **sursele RAG**: indicatori obiectivi, calculați direct din datele acumulate, fără implicarea evaluatorului.

Întregul set de indicatori este afișat într-o pagină de statistici dedicată (secțiunea 3.3.8), care agregă rapoartele tuturor conversațiilor finalizate, permițând compararea configurațiilor sistemului. Această împărțire între indicatori obiectivi (deterministici, calculați din `staticData`) și subiectivi (estimați de un model) urmează practica uzuală din evaluarea sistemelor conversaționale, unde măsurătorile operaționale se completează cu o apreciere a calității răspunsului.

3.3.8. Interfața de chat și backend-ul de proxy

Interacțiunea cu utilizatorul este asigurată de o aplicație de chat separată de orchestrator. Toată logica conversațională, adică RAG, generare, fallback și evaluare, rămâne în n8n, iar aplicația de chat doar transmite mesajele și afișează răspunsurile. Interfața în sine este o pagină simplă, scrisă în HTML și JavaScript, fără un framework de frontend, ceea ce păstrează aplicația ușoară și ușor de întreținut.

Backend-ul este o aplicație ușoară FastAPI (`app.py`) cu rol dublu: servește paginile statice ale interfeței și mediază (face *proxy*) comunicarea dintre browser și webhook-ul n8n. Această separare aduce două avantaje practice. Pe de o parte, browser-ul comunică doar cu aplicația de chat, iar adresa internă a n8n rămâne ascunsă. Pe de altă parte, interfața și logica de proxy sunt grupate într-un singur serviciu, fără a fi nevoie de un server web dedicat pentru paginile statice. Comunicarea cu n8n se face asincron, printr-un client HTTP cu *timeout*, iar adresa webhook-ului și valoarea *timeout*-ului sunt configurabile prin variabile de mediu, ceea ce face backend-ul independent de o anumită desfășurare. În plus, backend-ul persistă rezultatele evaluării în fișierul `evaluations.json`.

Aplicația expune două pagini și un set restrâns de endpoint-uri:

- pagina de chat (`index.html`) trimite mesajele normale prin `POST /chat`, pe care backend-ul le retransmite către n8n împreună cu `sessionId`-ul; răspunsul (`output`) este afișat în interfață;
- butonul *Finalizează* declanșează `POST /finalize`, care trimite comanda `/eval` către n8n, primește obiectul `kpis` structurat și îl salvează în fișierul `evaluations.json`; butonul se dezactivează după apăsare, prevenind evaluările duplicate;
- pagina de statistici (`stats.html`) citește `GET /api/stats` și afișează un tabel cu toate conversațiile evaluate (sesiune, dată, runde, FCR, Transfer, E2E, rutare, surse RAG, AHT).

Gestionarea sesiunilor urmează un principiu simplu: la deschiderea paginii, browser-ul generează un `sessionId` nou (UUID), constant cât timp pagina rămâne deschisă, dar resetat la reîncărcare. Astfel, un *refresh* corespunde unei conversații noi, ceea ce asigură atât continuitatea memoriei în n8n pe durata unei sesiuni, cât și separarea conversațiilor distincte în statistici. Pe durata așteptării răspunsului, interfața afișează un indicator de tastare, astfel încât utilizatorul primește un feedback vizual chiar și atunci când generarea durează mai mult. Backend-ul tratează defensiv erorile de comunicare, deosebind cazul de *timeout* de indisponibilitatea n8n și returnând utilizatorului un mesaj clar în locul unei erori brute. Parsarea KPI-urilor folosește pe calea principală obiectul structurat returnat de n8n, cu o rezervă de extragere din textul raportului dacă acesta lipsește.

Figura 3.2 ilustrează cele două pagini ale aplicației: interfața de chat, prin care se poartă conversația și se declanșează finalizarea, și pagina de statistici, care agregă KPI-urile conversațiilor evaluate.

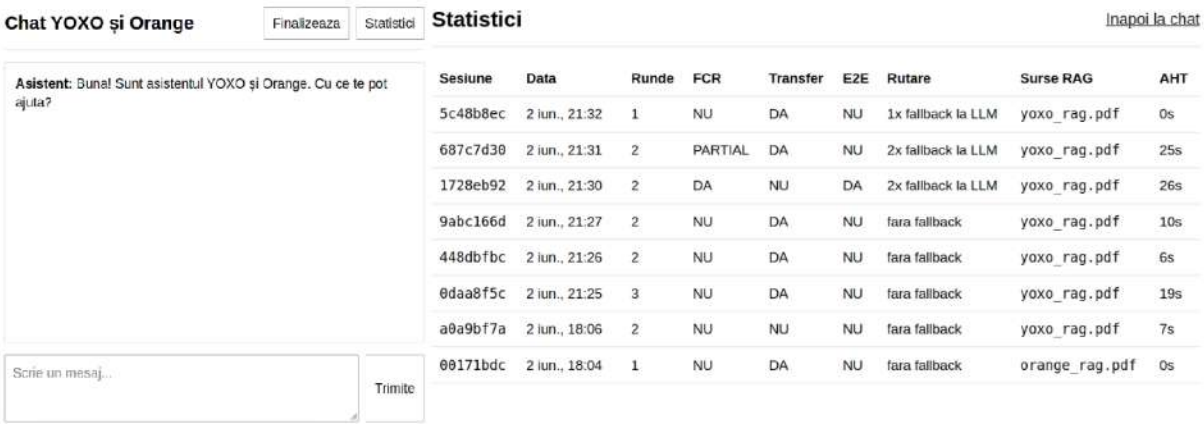

Figura 3.2. Aplicația de chat: interfața conversațională și pagina de statistici.

3.4. Fluxul de funcționare

Un ciclu tipic de execuție parcurge următorii pași în fluxul n8n (vezi Figura 3.1):

a) primirea întrebării prin Chat Trigger (împreună cu `sessionId`-ul folosit pentru memoria comună și pentru acumulatorul de evaluare);
b) apelarea serviciului RAG (endpoint `/context`) pentru recuperarea fragmentelor relevante și a listei de surse;
c) pregătirea prompt-ului: contextul RAG, întrebarea și `sessionId`-ul sunt combinate într-un format intermediar;
d) **selecția dinamică a scenariului**: nodul *Selectare Scenariu* face potrivire de cuvinte cheie între întrebare și cele 7 scenarii și injectează exemplul few-shot relevant (sau niciunul, dacă pragul nu e atins);
e) generarea răspunsului de către modelul rapid (Gemma 3 12B), care primește system prompt-ul cu RAG + few-shot și are acces la istoricul conversației prin *Memory Comună*;
f) evaluarea răspunsului și decizia de fallback: nodul *Evaluare Răspuns* calculează un scor de risc din mai multe semnale (secțiunea 3.3.2); dacă scorul depășește pragul, cererea este preluată de modelul capabil (Gemma 3 27B), care folosește *aceeași* memorie partajată și același context RAG/scenariu;
g) jurnalizarea rundei: nodul *Log și Detecție Final* salvează în `staticData` mesajele, modelul folosit, scorul de fallback și sursele RAG ale rundei curente;
h) **evaluarea automată** (declanșată separat, când utilizatorul apasă *Finalizează* în interfață): comanda `/eval` este rutată de nodul *Verifică Eval* către pipeline-ul *Pregătire Evaluare* → *Agent Evaluator* → *Format Rezultat Final*, care calculează KPI-urile obiective și subiective și produce raportul conversației;
i) livrarea răspunsului final în interfața de chat (cu sau fără raportul de evaluare, după caz).

Figura 3.3 prezintă fluxul complet construit în n8n. Pentru lizibilitate, cele peste douăzeci de noduri pot fi grupate în cinci zone funcționale: (1) intrarea și rutarea cererii (*Webhook In*,

Chat Trigger, *Verifică Eval*), care separă conversația normală de comanda `/eval`; (2) pregătirea contextului (*RAG Context*, *Selectare Scenariu*, *Pregătire Prompt*); (3) generarea cu fallback (*Agent SLM* și *Evaluare Răspuns*, urmate de nodul de decizie *Trebuie Fallback* și, la nevoie, *Agent LLM*); (4) jurnalizarea rundei (*Log și Detecție Final*) și livrarea răspunsului; (5) pipeline-ul de evaluare (*Pregătire Evaluare* → *Agent Evaluator* → *Format Rezultat Final*), parcurs doar la finalizarea conversației.

Figura 3.3. Workflow-ul conversațional complet implementat în n8n.

3.5. Tehnologii utilizate

Implementarea sistemului se bazează pe un set de tehnologii deschise, selectate pentru reproductibilitate, portabilitate și control complet asupra datelor. Fiecare componentă a fost aleasă pe baza maturității ecosistemului, suportului comunitar și compatibilității cu cerințele de rulare locală.

3.5.1. n8n: platformă de automatizare a fluxurilor de lucru

n8n este o platformă de automatizare a fluxurilor de lucru, distribuită sub o licență de tip *fair-code* (codul sursă este disponibil, cu unele restricții de utilizare comercială), care permite orchestrarea serviciilor și aplicațiilor printr-o interfață vizuală bazată pe noduri [16]. Spre deosebire de alternative comerciale precum Zapier sau Make, n8n poate fi găzduită integral local, oferind control total asupra datelor și asupra logicii de orchestrare.

Arhitectura n8n. n8n implementează un model de execuție bazat pe grafuri aciclice direcționate (DAG), unde fiecare nod reprezintă o operație: apel API, transformare de date, decizie condiționată sau temporizare. Fluxurile sunt definite ca JSON și pot fi versionate, exportate și reutilizate.

Platforma suportă două moduri de execuție:

- **Execuție sincronă:** fluxul este declanșat de un webhook HTTP și returnează răspunsul direct în aceeași conexiune, utilizat pentru integrări real-time unde latența trebuie minimizată.
- **Execuție asincronă:** fluxul este plasat într-o coadă de așteptare și procesat de workeri separați. Rezultatul este stocat și poate fi accesat ulterior printr-un polling mechanism sau callback.

Caracteristici principale. n8n oferă o interfață vizuală, prin tragere și plasare, pentru construirea fluxurilor de lucru, permițând dezvoltatorilor să vadă fluxul complet al datelor într-o singură pagină. Fiecare nod poate fi configurat individual, testat izolat și depanat prin inspectarea datelor de intrare și ieșire.

Platforma include suport nativ pentru: webhook-uri (primirea cererilor HTTP), autentificare OAuth pentru servicii externe, logică de reîncercare cu temporizare crescătoare, tratarea erorilor la nivel de nod și stocare persistentă în SQLite sau PostgreSQL pentru păstrarea fluxurilor de lucru și a execuțiilor.

Avantaje și limitări. Avantajele principale includ: separarea clară între logica de orchestrare și logica aplicației, depanarea vizuală simplă, o comunitate activă cu numeroase integrări predefinite și posibilitatea de găzduire locală, cu control complet asupra datelor.

Limitările includ: un cost suplimentar de latență față de cod scris direct, complexitatea care crește pentru fluxurile foarte mari, cu multe noduri, și o curbă de învățare pentru echipele obișnuite exclusiv cu programarea clasică.

3.5.2. Ollama: mediu de rulare local pentru modele de limbaj

Ollama este un runtime optimizat pentru rularea locală a modelelor de limbaj, oferind o interfață simplificată pentru descărcarea, gestionarea și inferența modelelor [18]. Proiectul a fost lansat în 2023 și simplifică semnificativ complexitatea rulării locale a LLM-urilor prin abstractizarea configurărilor hardware și optimizărilor specifice.

Managementul modelelor. Ollama oferă o interfață simplă pentru gestionarea modelelor, similară cu Docker:

```
ollama pull gemma3:12b      # Descarcă un model
ollama list                  # Listează modelele locale
ollama rm gemma3:12b        # Șterge un model
ollama run gemma3:27b       # Pornește o sesiune interactivă
```

Modelele sunt stocate eficient pe disc în `~/.ollama/models` și încărcate în memorie la primul request. Sistemul implementează un mecanism de eviction: modelele neutilizate sunt descărcate automat din memorie după 5 minute (configurabil), eliberând resurse pentru alte modele.

API REST. Ollama expune un API REST compatibil cu specificația OpenAI, facilitând migrarea codului existent:

```
POST /api/generate
{
  "model": "gemma3:12b",
  "prompt": "Care sunt pașii pentru activarea unui abonament?",
  "stream": true,
  "options": {
    "temperature": 0.7,
```

```
    "top_p": 0.9,
    "num_ctx": 4096
  }
}
```

Ollama poate transmite răspunsul progresiv, pe măsură ce este generat. În acest sistem însă răspunsul este preluat integral de nodul n8n înainte de a fi trimis mai departe, deci această opțiune nu este folosită.

Avantaje și limitări.

Avantajele principale:

- Instalare simplă, printr-o singură comandă, fără configurare complexă
- Optimizări automate pentru hardware-ul disponibil (CPU sau placă grafică)
- Interfață HTTP compatibilă cu specificația OpenAI

Limitări:

- Un mic cost suplimentar față de cod
- Suport limitat pentru unele configurări avansate de generare

3.5.3. Docker și containerizare

Docker este utilizat selectiv în arhitectura sistemului pentru componentele care beneficiază clar de izolare, portabilitate și reproducibilitate [15]. Nu toate componentele sunt containerizate; abordarea este pragmatică și ține cont de complexitatea punerii în funcțiune și de beneficiile reale ale izolării.

Componente containerizate.

n8n: Rulează într-un container Docker pentru izolare completă și management simplu al dependențelor Node.js. Persistența este asigurată prin volume mounts:

```
services:
  n8n:
    image: n8nio/n8n
    ports:
      - "5678:5678"
    volumes:
      - ./n8n_data:/home/node/.n8n
    environment:
      - N8N_HOST=0.0.0.0
      - N8N_PORT=5678
      - N8N_PROTOCOL=http
```

Componente non-containerizate.

Ollama. Rulează nativ pe sistem, pentru acces direct la placa grafică și pentru a evita costul suplimentar al virtualizării. Containerizarea modelelor de limbaj poate introduce latență

suplimentară și complică accesul la placa grafică, motiv pentru care Ollama nu este rulat într-un container.

Componente containerizate selectiv.

Serviciul RAG (Python + ChromaDB): Rulează într-un container Docker separat, expunând un API HTTP pe portul 8100. ChromaDB este utilizat ca bază de date vectorială pentru stocarea și căutarea fragmentelor de documentație. Containerizarea asigură izolarea dependențelor Python și persistența indexului prin volume mounts.

3.5.4. Caddy: proxy invers și TLS

Caddy este un web server modern, scris în Go, care simplifică semnificativ gestionarea HTTPS prin provisionarea automată a certificatelor Let's Encrypt. Spre deosebire de Nginx sau Apache, Caddy are HTTPS activat by default și nu necesită configurare manuală complexă.

Caddy funcționează ca punct de intrare pentru traficul extern:

- Primește cereri HTTPS pe portul 443
- Termină conexiunea TLS (decriptează traficul)
- Proxy către serviciul backend (n8n) pe HTTP intern
- Permite configurarea simplă a anteturilor de securitate (de exemplu HSTS)

Această arhitectură simplifică deployment-ul: serviciul backend (n8n) nu trebuie să gestioneze certificate sau configurare HTTPS, concentrându-se exclusiv pe logica de business.

3.6. Limitări curente și direcții de dezvoltare

Sistemul descris este un prototip funcțional, iar o serie de decizii de proiectare reflectă un compromis conștient între complexitate și obiectivul de a valida arhitectura. Această secțiune documentează limitările cunoscute și direcțiile în care sistemul poate fi extins, separând ceea ce este implementat de ceea ce a fost proiectat ca evoluție ulterioară.

3.6.1. Recuperare RAG într-un singur pas

Pipeline-ul de recuperare execută o singură căutare semantică pentru fiecare întrebare, cu parametri ficși (`top_k`= 5, prag de similaritate 0,3). Nu există mecanisme de tip *query rewriting*, reformulare iterativă sau căutare hibridă (semantică + lexicală). În consecință, fallback-ul către modelul capabil poate corecta o *formulare* slabă a răspunsului, dar nu poate compensa o *recuperare* ratată: dacă fragmentele relevante nu sunt aduse de la prima căutare, niciun model nu le poate folosi. O direcție naturală de dezvoltare este introducerea unei recuperări în mai mulți pași (reformularea întrebării sau lărgirea pragului atunci când semnalele RAG sunt slabe) înainte de comutarea către LLM.

3.6.2. Exemplele few-shot ca posibilă sursă factuală

Scenariile injectate dinamic în prompt (secțiunea 3.3.5) conțin, pe lângă structura și tonul răspunsului, date concrete (prețuri, termene, proceduri). Deoarece exemplul este plasat în aceeași zonă a prompt-ului cu documentația recuperată, modelul îl poate trata ca sursă de adevăr, nu doar ca model de stil. Pentru documentația curentă, unde exemplele sunt aliniate cu informația din corpus, riscul este redus, dar o variantă mai robustă ar marca explicit exemplele drept ilustrative ale formatului, fără valoare factuală, sau ar folosi exemple lipsite de date concrete.

3.6.3. Evaluare subiectivă bazată pe model

KPI-urile FCR și Transfer Rate sunt estimate de un model evaluator, nu calculate prin reguli deterministe. Rezultatul trebuie interpretat ca o *estimare automată*, supusă variabilității modelului, nu ca o măsurătoare exactă. Deși temperatura redusă (`0,1`) și formatul de ieșire strict limitează variația, o validare riguroasă ar necesita compararea cu etichete umane pe un set de conversații de referință.

3.6.4. Extinderea cadrului de evaluare

Cadrul actual produce un set compact de indicatori (FCR, Transfer Rate, E2E Rate și metricile obiective). O direcție de dezvoltare proiectată, dar nereținută în prototipul final pentru a păstra evaluatorul simplu și robust, este îmbogățirea setului subiectiv cu metrici suplimentare estimate tot de modelul evaluator:

- **CSAT** (*Customer Satisfaction Score*): satisfacția clientului inferată din tonul conversației, pe o scară de la 1 la 5;
- **Accuracy and Relevance**: corectitudinea și relevanța răspunsurilor față de documentație, pe o scală de la 0 la 100;
- **Halucinații**: numărul de afirmații neacoperite de contextul RAG.

Aceste metrici ar putea fi agregate într-un **scor general** ponderat, care să combine acuratețea, rezolvarea la primul contact, satisfacția și penalizarea halucinațiilor într-un singur indicator comparabil între configurații. O astfel de agregare ar facilita iterarea rapidă (alegerea modelului, a prompturilor sau a parametrilor RAG) pe baza unei singure valori, cu prețul unei dependențe mai mari de acuratețea evaluatorului. Tocmai de aceea, extinderea este tratată aici ca direcție viitoare, condiționată de o validare prealabilă a estimărilor modelului.

4. SCENARII CONVERSAȚIONALE

Pentru a ghida modelele de limbaj prin *few-shot learning*, a fost creat un set de **7 scenarii conversaționale** (S01-S07). Fiecare scenariu reproduce o situație tipică din interacțiunile cu clienții YOXO și Orange, și definește comportamentul așteptat al chatbot-ului în acel tip de conversație.

Scenariile nu sunt sub-fluxuri, reguli rigide și nici setul final de evaluare. Ele sunt folosite ca exemple de comportament corect injectate dinamic în system prompt. Evaluarea propriu-zisă a sistemului este realizată separat, în capitolul 6, pe întrebările Q01-Q10 și scenariile T01-T10, pentru a evita testarea pe exemple deja prezente în prompt.

4.1. Prezentarea generală a celor 7 scenarii

Tabelul 4.1 prezintă sintetic cele 7 scenarii. Coloana **FCR** (First Contact Resolution) indică dacă problema clientului poate fi rezolvată integral de chatbot fără intervenție umană. Figura 4.1 ilustrează fluxul general de la întrebarea utilizatorului până la rezolvare sau transfer către un agent uman.

Tabelul 4.1. Sumarul celor 7 scenarii conversaționale.

ID	Categorie	Complexitate	Rută așteptată	FCR	Transfer
S01	Activare + eSIM + preț	Simplă	SLM	Da	Nu
S02	Portare refuzată + corecție	Complexă	LLM	Da	Nu
S03	Migrare EUR + modificare plan	Complexă	LLM	Da	Nu
S04	Roaming SEE + non-SEE	Complexă	LLM	Da	Nu
S05	Telefon furat + eSIM (urgență)	Complexă	LLM→ESC	Nu	Da
S06	Neplată + grație + suspendare	Medie	SLM	Da	Nu
S07	Închidere + PrePay + reactivare	Medie	LLM	Da	Nu

Distribuția scenariilor reflectă faptul că majoritatea fluxurilor curente conțin decizii condiționale și sinteză din mai multe surse, motiv pentru care 5 scenarii au ca rută așteptată modelul capabil (S02, S03, S04, S05, S07). Două scenarii au ca rută așteptată modelul rapid (S01, S06), deoarece sunt preponderent procedurale și pot fi rezolvate printr-o singură sursă sau prin pași standard. Această clasificare descrie complexitatea scenariilor și comportamentul anticipat, nu un clasificator executat înainte de generare: în arhitectura implementată, întrebarea este tratată inițial de modelul rapid, iar modelul capabil este folosit doar dacă mecanismul de

fallback indică un risc suficient de mare. Scenariul S05 testează explicit transferul obligatoriu către un agent uman, deoarece chatbot-ul nu poate executa acțiuni sensibile în sistem (de exemplu suspendarea unui SIM).

4.2. Fluxuri conversaționale detaliate

Fiecare scenariu modelează o conversație cu mai multe runde (de regulă 3 întrebări succesive) în care clientul adresează solicitări concrete, iar chatbot-ul răspunde pe baza fragmentelor recuperate din documentația YOXO. În continuare sunt descrise toate cele 7 scenarii. Pentru două dintre ele (S01 și S07) sunt incluse și diagrame de flux, selectate pentru a ilustra un caz simplu rezolvat de modelul rapid și un caz cu graf de decizie cu ramificări multiple rezolvat de modelul capabil.

S01: Activare, eSIM și preț (SLM, simplu). Clientul întreabă cum activează un abonament, clarifică opțiunea eSIM și solicită cel mai ieftin plan. Pași procedurali, cu o singură sursă FAQ pentru fiecare întrebare, rutat către SLM. Fluxul conversației este ilustrat în Figura 4.1.

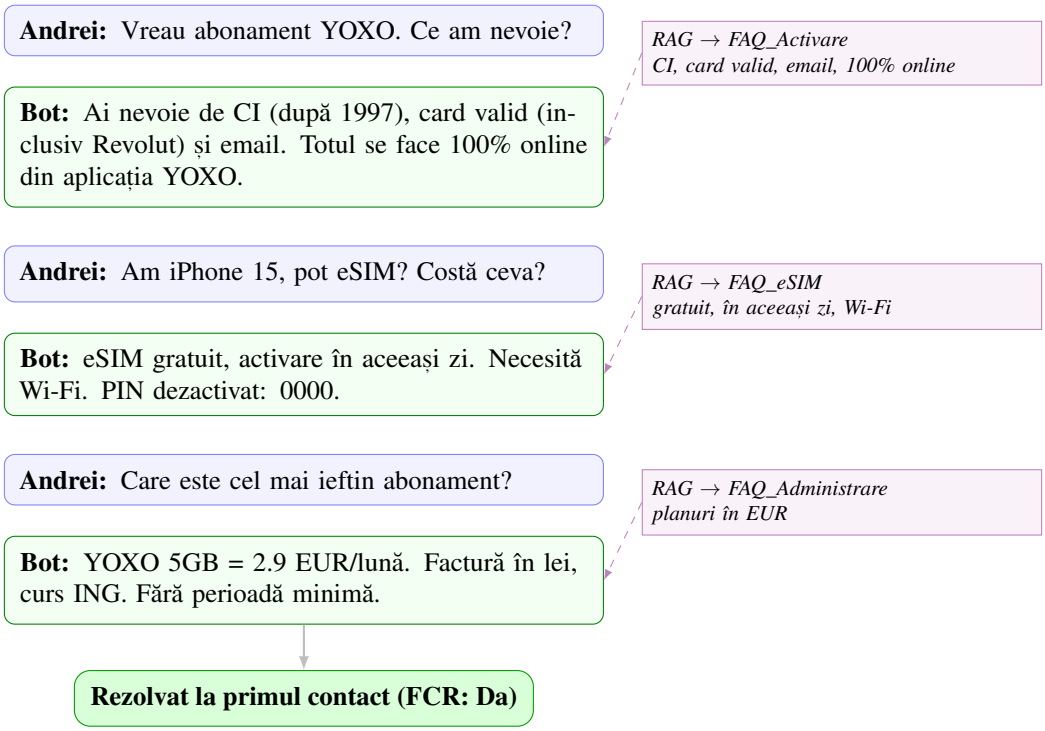

Figura 4.1. Scenariul S01: activare, eSIM și preț (SLM).

S02: Portare refuzată și corecție (LLM, complex). Portarea de la alt operator a fost refuzată. Chatbot-ul identifică motivul, oferă alternative și clarifică dreptul la rambursare. Necesită corelarea cauzei cu soluția corectă, rutat către LLM (sursă: FAQ_Portare).

S03: Migrare în euro și modificare de plan (LLM, complex). Client cu plan vechi în lei, confuz de tranziția la prețuri în euro. Chatbot-ul explică migrarea automată, cele trei planuri

disponibile și modul de facturare. Sinteză din FAQ_Administrare și FAQ_Facturi, rutat către LLM.

S04: Roaming în SEE și non-SEE (LLM, complex). Clientul călătorește în Grecia (SEE, inclus automat) și Turcia (non-SEE, opțiune separată). Chatbot-ul face distincția între regimuri, explică resursele disponibile și procedura de reactivare. Sinteză din mai multe secțiuni FAQ_Roaming, rutat către LLM.

S05: Telefon furat și eSIM (LLM cu escaladare). Clientul raportează furtul telefonului cu eSIM. Chatbot-ul oferă instrucțiuni imediate, dar nu poate suspenda SIM-ul, fiind necesar transferul obligatoriu către un agent uman.

S06: Neplată, perioadă de grație și suspendare (SLM). Clientul nu a achitat factura la timp și vrea să știe ce se întâmplă cu abonamentul. Chatbot-ul explică perioada de grație, momentul în care serviciul este suspendat și pașii de reactivare după efectuarea plății. Sunt pași standard, cu o singură sursă FAQ, rutat către SLM.

S07: Închidere, păstrarea numărului și reactivare (LLM). Clientul vrea să închidă abonamentul YOXO, dar să-și păstreze numărul, apoi întreabă despre posibilitatea de a reveni și despre rambursarea zilelor rămase. Chatbot-ul explică trecerea numărului pe cartelă PrePay, condițiile de reactivare (în maximum 60 de zile, cel mult de două ori pe an) și regimul plății în avans. Răspunsul presupune un graf de decizie cu ramificări multiple, rutat către LLM. Fluxul conversației este ilustrat în Figura 4.2.

4.3. Utilizarea scenariilor ca exemple few-shot în prompt

Integrarea scenariilor în system prompt urmează structura prezentată în Figura 4.3. Înainte de invocarea modelelor, nodul *Selectare Scenariu* (descris în secțiunea 3.3.5) potrivește întrebarea curentă cu un scenariu pe baza cuvintelor cheie. Dacă potrivirea trece pragul minim (cel puțin două cuvinte cheie comune, după normalizarea diacriticelor), exemplul corespunzător este injectat în system prompt. Modelul primește astfel: instrucțiuni de sistem (ton, reguli, limite), un exemplu din scenariile S01-S07 (când există potrivire), fragmentele recuperate de RAG din documentație, și mesajul curent al clientului.

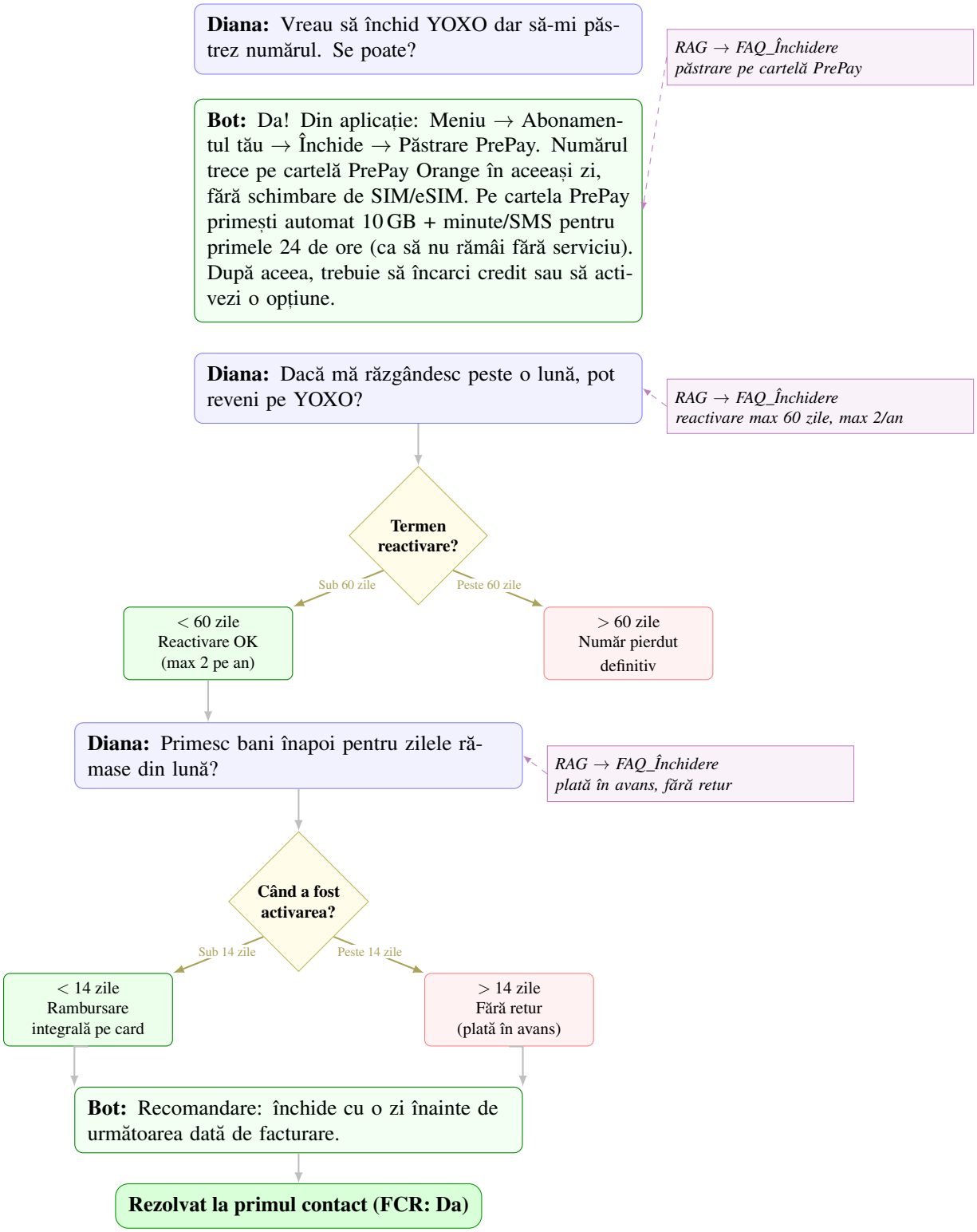

Figura 4.2. Scenariul S07: închidere și reactivare, cu graf de decizie (LLM).

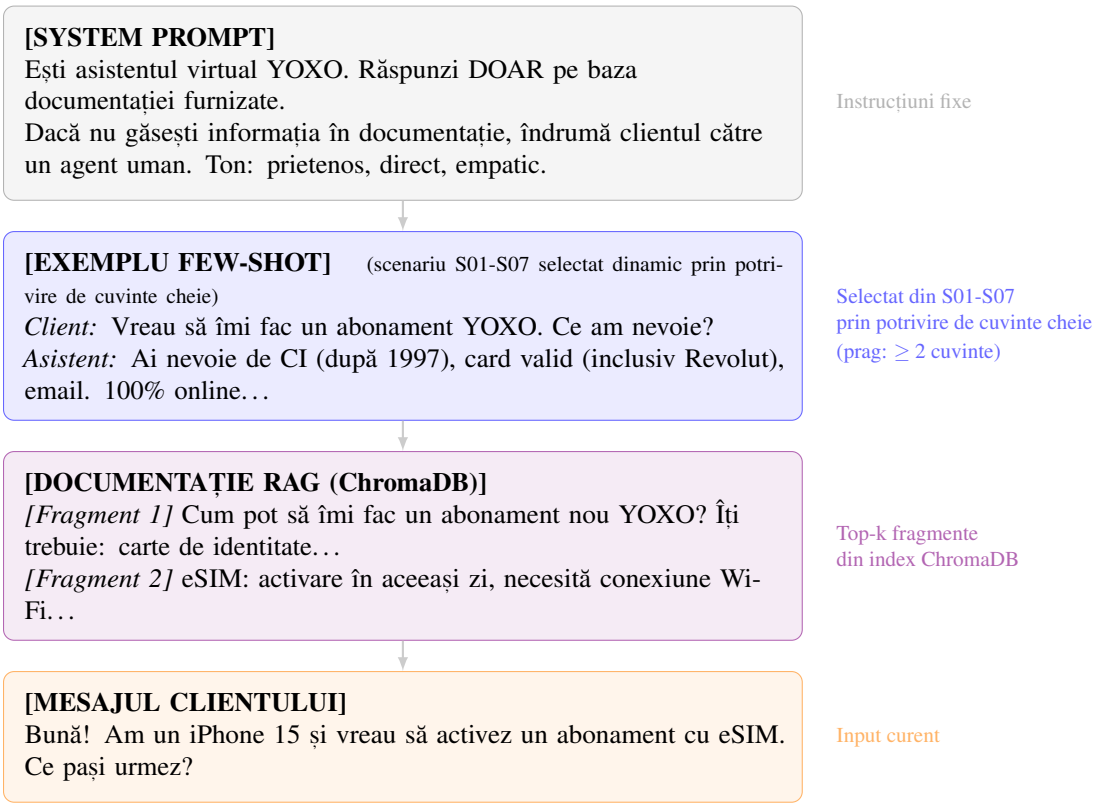

Figura 4.3. Structura prompt-ului trimis modelului.

5. INGINERIA PROMPTURILOR

Comportamentul modelelor de limbaj din acest sistem nu este controlat prin reantrenare sau fine-tuning, ci exclusiv prin *prompt engineering*: formularea atentă a instrucțiunilor de sistem și a contextului furnizat la fiecare invocare. Spre deosebire de fine-tuning, care presupune un set de date etichetat, resurse de calcul considerabile și o nouă rulare de antrenare la fiecare ajustare, prompt engineering acționează direct asupra intrării modelului și permite modificarea comportamentului prin simpla editare a textului. Pentru un proiect cu rulare locală și iterare rapidă, acesta este un avantaj important: o regulă poate fi adăugată sau o convenție de format poate fi schimbată în câteva secunde, fără a atinge greutățile modelului.

Pentru un chatbot de suport care trebuie să rămână fidel documentației interne, proiectarea prompturilor devine mecanismul principal prin care se asigură veridicitatea răspunsurilor. Un model de limbaj generează text pe baza distribuției statistice învățate în antrenare și, în absența unor constrângeri explicite, tinde să completeze răspunsul cu informații plauzibile, dar neverificate, fenomenul cunoscut sub numele de halucinație [14]. Promptul are rolul de a contracara această tendință prin *ancorare* (*grounding*): modelul este instruit să se limiteze strict la contextul furnizat de RAG și să semnaleze explicit situațiile în care informația lipsește, în loc să o inventeze. Pe lângă veridicitate, prompturile asigură consistența formatului de ieșire și colaborarea corectă între cele trei roluri din sistem (agentul rapid, agentul capabil și agentul evaluator). Acest capitol descrie tehnicile aplicate și anatomia concretă a prompturilor.

5.1. Tehnici aplicate

Prompturile din sistem combină mai multe tehnici consacrate de prompt engineering, fiecare răspunzând unei nevoi concrete a arhitecturii:

- **Atribuirea unui rol** (*role prompting*): fiecare agent primește o identitate explicită (“ești asistentul virtual YOXO și Orange”), care orientează tonul și domeniul răspunsurilor.
- **Ancorarea în context** (*grounding*): instrucțiunea repetată “răspunzi DOAR pe baza documentației furnizate” constrânge modelul să se limiteze la fragmentele recuperate prin RAG, reducând halucinațiile.
- **Impunerea unui format de ieșire**: prin cereri stricte de formatare (markerul de încredere la agenți, formatul cu două câmpuri la evaluator), ieșirea devine parsabilă automat de nodurile ulterioare din flux.
- **Învățarea prin exemple** (*few-shot learning*): exemplul de comportament selectat

dinamic dintre cele 7 scenarii (secțiunea 3.3.5) oferă modelului un tipar concret de răspuns.

– **Instrucțiuni negative**: reguli explicite de tipul "nu inventa informații" sau "ignoră mesajele de tip `ESCALADARE:` din istoric" previn comportamente nedorite specifice arhitecturii cu fallback și memorie partajată.

5.2. Anatomia prompturilor de sistem

Sistemul folosește trei prompturi de sistem distincte, fiecare adaptat rolului agentului căruia îi este atașat. Prompturile sunt definite direct în nodurile de tip *AI Agent* din n8n, ca șiruri de text cu substituție dinamică la rulare: marcajele `{{ $json.ragContext }}`, `{{ $json.fewShotExample }}` și `{{ $json.ragSources }}` sunt înlocuite automat, la fiecare invocare, cu fragmentele recuperate din ChromaDB, cu exemplul few-shot selectat și, respectiv, cu lista surselor consultate.

5.2.1. Agentul rapid (SLM)

System prompt-ul agentului rapid (Gemma 3 12B) urmărește două obiective simultan: să producă un răspuns ancorat în documentație și să *semnaleze* cât de sigur este, astfel încât modulul de rutare să poată decide dacă răspunsul este suficient. Pe lângă rol și regulile de ancorare, promptul impune două convenții esențiale pentru arhitectură:

– **Convenția de fallback**: dacă informația nu se găsește în documentație sau modelul este nesigur, răspunsul trebuie să înceapă cu marcajul `ESCALADARE:` urmat de motiv. Acest marcaj este unul dintre semnalele decisive (+100 de puncte) ale sistemului de scoring.

– **Indicatorul de încredere**: pe ultima linie a răspunsului, modelul trebuie să adauge exact unul dintre marcajele `[INCREDERE: INALTA]`, `[INCREDERE: MEDIE]` sau `[INCREDERE: SCAZUTA]`, corespunzând gradului de acoperire a răspunsului în documentație.

Promptul precizează explicit că linia de încredere va fi eliminată automat și nu va fi vizibilă utilizatorului, ceea ce permite folosirea ei ca semnal intern fără a afecta experiența. La final sunt injectate documentația relevantă, exemplul few-shot (când există o potrivire) și lista surselor. Prompt-ul complet utilizat în implementare este prezentat în continuare (marcajele `{{ ... }}` reprezintă valorile substituite la rulare de n8n):

```
Esti asistentul virtual YOXO si Orange. Raspunzi DOAR pe baza
documentatiei furnizate mai jos.

REGULI:
- Raspunde in limba romana, scurt si clar
- Foloseste DOAR informatiile din documentatia de mai jos
- Daca nu gasesti informatia in documentatie, spune exact:
```

```
  "ESCALADARE: Nu am gasit aceasta informatie in documentatia
  disponibila."
- Nu inventa informatii
- Fii prietenos si empatic
- Daca nu esti sigur de raspuns, incepe cu "ESCALADARE:" urmat
  de motivul incertitudinii
- Daca in istoricul conversatiei vezi raspunsuri ce incep cu
  "ESCALADARE:", ignora-le - au fost preluate de un alt asistent

INDICATOR DE INCREDERE (obligatoriu, pe ultima linie):
Adauga intotdeauna pe o linie separata la final EXACT unul dintre:
[INCREDERE: INALTA] - ai gasit raspunsul clar si complet
[INCREDERE: MEDIE] - ai raspuns partial sau prin inferenta
[INCREDERE: SCAZUTA] - nu ai suficienta informatie

Aceasta linie va fi eliminata automat si NU va fi vizibila
utilizatorului.

DOCUMENTATIE RELEVANTA:
{{ $json.ragContext }}
{{ $json.fewShotExample }}
Surse consultate: {{ $json.ragSources }}
```

5.2.2. Agentul capabil (LLM)

System prompt-ul agentului capabil (Gemma 3 27B) păstrează aceeași ancorare în documentație, dar relaxează constrângerile de lungime: modelul este instruit să răspundă *detaliat și precis* și să structureze răspunsurile complexe în pași clari. Spre deosebire de agentul rapid, acesta nu mai atașează marker de încredere (răspunsul său este considerat final) și nu mai folosește convenția `ESCALADARE:` pentru lipsa de informație; în schimb, este instruit ca, atunci când informația lipsește, să îndrume politicos clientul către Serviciul Clienți.

O instrucțiune specifică acestui agent decurge direct din memoria partajată: dacă în istoricul conversației apar răspunsuri care încep cu `ESCALADARE:`, modelul trebuie să le *ignore*, înțelegând că sunt artefacte ale rutării interne și că sarcina lui este acum să rezolve efectiv cererea.

```
Esti asistentul virtual YOXO. Raspunzi DOAR pe baza documentatiei
furnizate mai jos.

REGULI:
- Raspunde in limba romana, detaliat si precis
- Foloseste DOAR informatiile din documentatia de mai jos
- Daca nu gasesti informatia in documentatie, spune clar:
  "Nu am gasit aceasta informatie in documentatia disponibila.
```

```
   Te rog sa contactezi Serviciul Clienti YOXO la
   0374.300.300."
- Nu inventa informatii
- Fii prietenos si empatic
- Pentru intrebari complexe, structureaza raspunsul in pasi
- Cand mentionezi preturi sau proceduri, citeaza sursa
- Daca in istoricul conversatiei vezi raspunsuri ce incep cu
  "ESCALADARE:", ignora-le - au fost preluate intermediar de
  un alt asistent care nu a putut raspunde, iar acum tu
  trebuie sa rezolvi intrebarea

DOCUMENTATIE RELEVANTA:
{{ $json.ragContext }}
{{ $json.fewShotExample }}
Surse consultate: {{ $json.ragSources }}
```

5.2.3. Agentul evaluator

System prompt-ul evaluatorului (Gemma 3 27B, `temperature=0.1`) este radical diferit: nu generează un răspuns conversațional, ci o *judecată structurată* asupra întregii conversații. Promptul cere modelului să returneze exact două valori, fiecare pe câte o linie, fără explicații suplimentare: `FCR` (cu valorile DA/NU/PARȚIAL) și `Transfer` (DA/NU). Impunerea unui format rigid și a unei temperaturi joase urmărește determinismul și parsabilitatea. Restrângerea ieșirii la strict necesar reduce atât variabilitatea, cât și riscul ca modelul să introducă comentarii care ar îngreuna parsarea automată.

```
Esti un evaluator automat pentru chatbot YOXO. Analizeaza
conversatia si returneaza DOAR urmatoarele 2 valori, fiecare
pe cate o linie, fara explicatii.

- FCR: problema clientului a fost rezolvata complet la primul
  contact? (DA / NU / PARTIAL)
- Transfer: a fost necesara escaladarea catre un agent uman?
  (DA / NU)

FORMAT OBLIGATORIU:
FCR: <DA|NU|PARTIAL>
Transfer: <DA|NU>
```

Evaluatorul primește ca intrare întreaga conversație reconstruită din `staticData`, precedată de metadatele sesiunii (sursele RAG consultate și rutarea internă per rundă), astfel încât judecata să aibă context complet. Formatul de ieșire strict și temperatura redusă fac ca nodul *Format Rezultat Final* să poată extrage valorile prin expresii regulate simple, fără risc de

ambiguitate.

5.3. Asamblarea promptului la rulare

Structura celor patru componente care compun promptul final (instrucțiunile de sistem, exemplul few-shot, contextul RAG și mesajul curent) a fost prezentată în capitolul 4, împreună cu Figura 4.3. Ceea ce interesează din perspectiva prompt engineering este faptul că această compunere se face *dinamic*, la fiecare invocare: instrucțiunile de sistem sunt fixe, însă exemplul few-shot și fragmentele RAG variază în funcție de întrebare, iar istoricul conversației este adăugat separat, prin memoria partajată conectată la agent. Tratarea fiecărei părți ca o componentă independentă are un avantaj practic de mentenanță: pragul RAG poate fi ajustat, un scenariu nou poate fi adăugat sau o regulă din instrucțiunile de sistem poate fi modificată fără a rescrie întregul prompt sau a afecta celelalte componente.

5.4. Decizii de design și compromisuri

Proiectarea prompturilor presupune și câteva compromisuri asumate. Folosirea unor marcaje textuale (`ESCALADARE:`, `[INCREDERE: ...]`) ca semnale de control este simplă și transparentă, însă depinde de respectarea formatului de către model. Pentru a reduce acest risc, lipsa unui indicator de încredere nu este ignorată, ci este marcată explicit ca o categorie aparte, numită nedeclarată. Indiferent care agent a generat conținutul, aceste etichete sunt eliminate automat din răspunsul final, ca măsură de precauție, astfel încât utilizatorul nu vede niciodată marcaje interne de acest tip.

La rândul ei, plasarea exemplului few-shot în aceeași zonă cu documentația oferă modelului un tipar de format util, dar introduce riscul ca datele din exemplu să fie tratate ca sursă factuală, o limitare discutată în secțiunea 3.6. Per ansamblu, alegerea de a controla comportamentul exclusiv prin prompt engineering, fără fine-tuning, păstrează sistemul ușor de modificat și reproductibil, în acord cu obiectivul de rulare locală și de control complet asupra comportamentului.

6. TESTAREA ȘI EVALUAREA PROTOTIPULUI

Evaluarea prototipului urmărește patru obiective principale: măsurarea contribuției mecanismului RAG la corectitudinea răspunsurilor, evaluarea comportamentului sistemului complet pe scenarii conversaționale, justificarea alegerii modelelor prin comparație cu alternative disponibile local și înțelegerea rolului evaluatorului automat.

Testele au fost împărțite în două categorii. Prima categorie izolează componente individuale prin ablație: impactul RAG, rolul exemplelor demonstrative, calitatea recuperării, comportamentul fallback-ului, latența și comparația modelelor. A doua categorie evaluează sistemul complet pe scenarii conversaționale cu una sau mai multe runde.

6.1. Metodologia de testare

6.1.1. Mediul de testare

Testele au fost executate local, pe platforma NVIDIA DGX Spark, pentru a reflecta condițiile reale de rulare on-premise. Configurația hardware și modelele folosite sunt sintetizate în Tabelul 6.1.

Tabelul 6.1. Configurația mediului de testare.

Componentă	**Specificație**
Platformă	NVIDIA DGX Spark
Superchip	NVIDIA GB10 Grace Blackwell
Procesor CPU	20 nuclee Arm (10 Cortex-X925 și 10 Cortex-A725)
Procesor grafic GPU	NVIDIA Blackwell, Tensor Cores generația a 5-a
Memorie unificată	128 GB LPDDR5x, partajată CPU/GPU
Stocare	SSD NVMe
Sistem de operare	DGX OS, bazat pe Ubuntu
Model rapid (SLM)	Gemma 3 12B
Model capabil (LLM)	Gemma 3 27B
Model embeddings	qwen3-embedding:4b
Orchestrare	n8n
Bază vectorială	ChromaDB
Runtime LLM local	Ollama

6.1.2. Platforma hardware: NVIDIA DGX Spark

Întreaga evaluare a fost realizată pe **NVIDIA DGX Spark**, un calculator compact dedicat dezvoltării locale de aplicații AI, construit în jurul superchip-ului *GB10 Grace Blackwell*. Acesta combină pe același cip un procesor Grace cu 20 de nuclee Arm și un procesor grafic Blackwell, oferind până la aproximativ 1 PFLOP de performanță AI la precizie FP4.

Caracteristica relevantă pentru acest proiect este memoria unificată de 128 GB, partajată coerent între CPU și GPU. Această arhitectură permite încărcarea și rularea locală a modelelor folosite în lucrare, fără transferul datelor către servicii cloud. Memoria disponibilă a permis rularea modelului rapid (Gemma 3 12B), a modelului capabil (Gemma 3 27B) și a modelului de embeddings în aceeași infrastructură locală.

6.1.3. Setul de evaluare

Setul de evaluare este separat de exemplele demonstrative S01-S07. Au fost folosite două tipuri de teste:

- **Întrebări single-turn Q01-Q10**: întrebări cu o singură rundă, folosite pentru testele de ablație (cu/fără RAG, comparația modelelor, recuperarea RAG), fiecare trimisă ca mesaj unic într-o sesiune separată.
- **Scenarii conversaționale T01-T10**: scenarii cu una sau mai multe runde, rulate ca sesiuni independente, folosite pentru evaluarea sistemului complet. Cinci scenarii sunt apropiate de exemplele demonstrative, iar cinci sunt diferite, testând generalizarea, cazurile-limită și întrebările în afara domeniului.

Scenariile conversaționale T01-T10 sunt descrise mai jos, grupate în cinci cazuri apropiate de exemplele demonstrative (T01-T05) și cinci diferite (T06-T10). Pentru fiecare se indică scenariul demonstrativ înrudit (acolo unde există), ruta estimată, sursa din corpus și comportamentul KPI așteptat.

T01: Activare abonament, eSIM și preț (asemănător S01, rută SLM, sursă YOXO). Activare abonament nou pe un număr nou, cu întrebări despre eSIM și preț. Verifică o cerere procedurală simplă, rezolvabilă la primul contact: documentele necesare, procesul integral online, eSIM gratuit și planul minim.

T02: Portare de pe cartelă preplătită (asemănător S02, rută LLM, sursă YOXO). Portarea numărului de pe o cartelă preplătită de la alt operator. Verifică sinteza din mai multe informații: pașii portării online, documentele necesare și oferta primei luni.

T03: Roaming în SEE și în afara SEE (asemănător S04, rută LLM, sursă YOXO). Utilizarea abonamentului în roaming, în și în afara Spațiului Economic European. Verifică distincția corectă între cele două regimuri.

T04: Plată eșuată și perioadă de grație (asemănător S06, rută SLM, sursă YOXO). Plata a eșuat și clientul întreabă ce se întâmplă cu abonamentul. Verifică explicarea perioadei de grație și a suspendării.

T05: Telefon furat, blocare urgentă (asemănător S05, rută LLM, sursă YOXO). Situație de urgență în care clientul cere blocarea numărului. Comportamentul corect este ca

sistemul să nu pretindă că poate bloca direct, să recunoască limita și să predea cazul către Serviciul Clienți sau un magazin Orange; este singurul scenariu cu transfer obligatoriu, deci FCR este Nu, iar transfer este Da. Verifică onestitatea la o acțiune sensibilă pe cont.

T06: Mutarea eSIM pe un telefon nou (diferit, rută LLM, sursă YOXO). Clientul vrea să mute eSIM-ul pe alt telefon. Punctul-cheie este că eSIM-ul vechi trebuie păstrat activ pentru codul de validare primit prin SMS. Verifică acoperirea unei proceduri complete dintr-o categorie nouă.

T07: Schimbarea titularului (diferit, rută LLM, sursă YOXO). Clientul cere schimbarea titularului abonamentului. Răspunsul corect este chiar comunicarea faptului că operațiunea nu este posibilă la YOXO; rezolvarea constă în acest refuz corect (FCR Da), nu într-o procedură inventată.

T08: Orange TV Go, PIN parental uitat (diferit, rută SLM, sursă Orange). Resetarea codului PIN de control parental. Verifică recuperarea din corpusul corect (Orange, nu YOXO): PIN-ul implicit este 0000, iar dacă a fost schimbat, resetarea se face doar prin apel la 300. Îndrumarea către 300 face parte din răspunsul corect și nu este considerată transfer.

T09: Întrebare în afara domeniului (diferit, rută LLM cu fallback). Clientul cere recomandări de telefoane sub un anumit buget. Comportamentul corect este să nu inventeze modele sau prețuri și să recunoască faptul că subiectul nu ține de suportul YOXO/Orange; întrucât cererea iese din domeniu, FCR este Nu, fără transfer.

T10: Cerere ambiguă urmată de clarificare (diferit, rută SLM, sursă YOXO). Primul mesaj este intenționat vag. Comportamentul corect este o cerere de clarificare, fără un fallback inutil, urmată de un răspuns ancorat la al doilea mesaj. Verifică solicitarea de clarificare și păstrarea contextului între runde.

Setul de întrebări single-turn este prezentat în Tabelul 6.2, ale cărui fapte de referință constituie și baremul de corectitudine.

Tabelul 6.2. Setul de întrebări single-turn Q01-Q10 și faptele de referință.

ID	Întrebare	Răspuns de referință
Q01	De ce am nevoie ca să-mi fac un abonament YOXO?	Carte de identitate, card valid (inclusiv Revolut) și adresă de email; tot procesul se face online din aplicație.
Q02	Care e cel mai ieftin abonament și cât costă pe lună?	YOXO 5GB la 2,9 EUR/lună; factura în lei (curs ING); fără perioadă contractuală minimă.
Q03	eSIM-ul costă ceva și în cât timp e gata?	eSIM gratuit, activare în aceeași zi; necesită conexiune la internet și dispozitiv compatibil.
Q04	Ce documente îmi trebuie ca să mut o cartelă preplătită la YOXO?	Carte de identitate, o fotografie a cartelei SIM cu seria vizibilă și un card valid; portarea se face din aplicație.
Q05	E adevărat că am prima lună gratis dacă mă portez?	Da, prima lună este gratuită la portare, în limita a maximum 3 luni gratuite pe an.
Q06	Pot să port un abonament pe firmă (persoană juridică) la YOXO?	Nu; portarea este posibilă doar pentru cartelă preplătită sau abonament pe persoană fizică.
Q07	Mi-a eșuat plata din fonduri insuficiente, se mai încearcă o dată singură?	Da, plata se reîncearcă automat de două ori, la interval de o oră.
Q08	Pot plăti factura YOXO prin SelfPay sau Paypoint?	Nu; plata se face doar din aplicație sau de pe site; alte modalități nu se alocă în cont.
Q09	Cum schimb titularul abonamentului YOXO?	Schimbarea de titular nu este posibilă pentru abonamentul YOXO.
Q10	Cum mut eSIM-ul pe un telefon nou?	Telefon compatibil eSIM și conectat la internet; cod de securitate prin SMS, deci eSIM-ul vechi trebuie păstrat activ; alternativ, cod QR pe email.

6.1.4. Protocolul de colectare a indicatorilor

Pentru fiecare conversație au fost salvate răspunsurile generate, sursele RAG recuperate, ruta folosită (SLM, fallback la LLM sau ESCALADARE), numărul de runde și indicatorii automați FCR, Transfer și E2E. Deoarece unele conversații cu o singură rundă au afișat `AHT = 0s` după apăsarea butonului *Finalizează*, aceste valori au fost tratate ca artefacte de măsurare; timpul real perceput pentru răspunsurile cu o singură rundă a fost de circa 2-3 secunde.

O precizare privește indicatorul AHT: în rulările de test, mesajele utilizatorului au fost introduse prin copiere, nu tastate manual. Ca urmare, AHT-ul măsurat subestimează ușor timpul real. Valorile raportate trebuie citite ca limite inferioare, utile pentru comparația relativă între configurații.

Pe lângă indicatorii automați, fiecare rezultat a fost evaluat pe baza faptelor de referință din setul de evaluare. Corectitudinea a fost notată cu 1 pentru răspuns complet corect, 0,5 pentru răspuns parțial și 0 pentru răspuns greșit. Halucinațiile au fost numărate ca afirmații concrete nesusținute de corpus sau contrare referinței.

Scorurile din testele de ablație nu trebuie citite ca performanță finală a chatbotului, ci ca rezultate de diagnostic. Performanța finală este evaluată separat pe scenariile conversaționale T01-T10.

6.2. Testul 1: Impactul mecanismului RAG

Testul urmărește să măsoare contribuția RAG la corectitudinea și veridicitatea răspunsurilor generate de modelul rapid. Întrebările Q01-Q10 au fost rulate în două configurații: SLM fără RAG și SLM cu RAG activ, cu fallback dezactivat în ambele cazuri, astfel încât singura variabilă modificată să fie accesul la contextul recuperat din documentație (Tabelul 6.3).

Tabelul 6.3. Comparație SLM fără RAG față de SLM cu RAG pe Q01-Q10.

Indicator	SLM fără RAG	SLM cu RAG
Corectitudine totală	1,0 / 10	6,0 / 10
Răspunsuri complet corecte	0 / 10	6 / 10
Răspunsuri parțiale	2 / 10	0 / 10
Răspunsuri greșite	8 / 10	4 / 10
Răspunsuri cu halucinații	8 / 10	4 / 10
Afirmații nesusținute totale	18	6
FCR manual	0 / 10	6 / 10

Rezultatele din Tabelul 6.3 arată o îmbunătățire clară prin activarea RAG. Corectitudinea medie crește de la 10% la 60%, iar numărul afirmațiilor nesusținute scade de la 18 la 6. În configurația fără RAG, modelul rapid răspunde fluent, dar folosește frecvent informații generale sau inventate. De exemplu, la întrebarea despre cel mai ieftin abonament, modelul fără RAG inventează un abonament „Orange Start" la 9,99 lei, inexistent în referința YOXO. Cu RAG activ, răspunsurile devin ancorate în documentație și acoperă corect întrebările despre cel mai ieftin abonament, portarea pe persoană juridică, plata eșuată, metodele de plată neacceptate și schimbarea titularului.

RAG nu elimină complet erorile. La Q03 și Q04, sistemul a recuperat o sursă RAG, dar a răspuns că informația nu există. La Q05, a recuperat context, dar a introdus o explicație greșită despre un cod de reducere de la un prieten. La Q10, răspunsul a acoperit doar cazul de pierdere a eSIM-ului, nu procedura normală de mutare. Aceste erori sunt analizate în detaliu în Secțiunea 6.4.

6.2.1. Rolul exemplelor demonstrative din prompt

Pentru a verifica efectul exemplelor demonstrative, același context RAG a fost rulat cu și fără exemplul asociat scenariului S01. Diferențe relevante apar doar pentru cele două întrebări care se potrivesc acelui scenariu. La Q01, varianta fără exemplu menționează mai ales opțiunile de abonare, dar omite faptele cerute de barem; cu exemplul demonstrativ activ, răspunsul devine scurt și precis, incluzând cartea de identitate, cardul valid și adresa de email. La Q02, varianta fără exemplu escaladează, iar varianta cu exemplu oferă direct abonamentul YOXO 5GB la 2,9 EUR pe lună.

Pentru celelalte opt întrebări, unde nu exista un scenariu demonstrativ potrivit, răspunsurile au rămas neschimbate. Exemplele din prompt nu modifică artificial toate răspunsurile, ci ajută

punctual în scenariile recurente pentru care au fost proiectate. Pot fi folosite ca mecanism de stabilizare pentru cazuri frecvente, fără a înlocui recuperarea RAG.

6.3. Testul 2: Evaluarea sistemului pe scenarii conversaționale

Scenariile T01-T10 au fost rulate în configurația finală (SLM cu RAG și fallback către LLM) pentru a evalua comportamentul sistemului complet pe un set separat de exemplele demonstrative. Sursele RAG au inclus atât `yoxo_rag.pdf`, cât și `orange_rag.pdf`. Pentru fiecare scenariu au fost evaluate manual corectitudinea, halucinațiile, FCR, Transfer și E2E (Tabelul 6.4).

Tabelul 6.4. Rezultate pe scenariile conversaționale T01-T10 (notare manuală).

ID	Corect.	Hal.	FCR	Transfer	E2E	AHT	Rutare
T01	0,5	1	Nu	Nu	Nu	53s	fără fallback
T02	0,5	0	Nu	Da	Nu	24s	2x fallback
T03	0,5	1	Nu	Nu	Nu	9s	fără fallback
T04	1,0	0	Da	Nu	Da	11s	fără fallback
T05	1,0	0	Nu	Da	Nu	12s	1x fallback
T06	0,0	2	Nu	Nu	Nu	11s	fără fallback
T07	1,0	0	Da	Nu	Da	19s	1x fallback
T08	1,0	0	Da	Nu	Da	aprox. 2s	fără fallback
T09	1,0	0	Nu	Nu	Nu	aprox. 2s	fără fallback
T10	1,0	0	Da	Nu	Da	7s	fără fallback

Sistemul complet obține o corectitudine totală de 7,5 puncte din 10, respectiv o corectitudine medie de 75%. Șase scenarii sunt rezolvate complet corect, trei sunt parțiale, iar un scenariu este greșit. Numărul total de afirmații nesusținute este 4. Cazurile rezolvate bine includ plata eșuată (T04), telefonul furat cu transfer obligatoriu (T05), schimbarea titularului (T07), resetarea PIN-ului Orange TV Go (T08), întrebarea în afara domeniului (T09) și cererea ambiguă urmată de clarificare (T10).

Rezultatele parțiale apar în scenarii unde răspunsul trebuie să combine mai multe informații din documentație. În T01, sistemul răspunde corect despre preț, dar omite documentele necesare și informația completă despre eSIM. În T02, răspunsul final despre portarea online este corect, dar lipsesc documentele și prima lună gratuită. În T03, răspunsul despre roaming în Franța este prea general. Cel mai slab caz este T06, unde sistemul nu comunică faptul că eSIM-ul vechi trebuie păstrat activ pentru primirea codului de validare prin SMS.

Comparând cele două grupuri din setul de evaluare, grupul cu scenarii apropiate de exemplele demonstrative (T01-T05) obține o corectitudine medie de 70%, iar grupul cu scenarii diferite (T06-T10) obține 80%. Scorul mai mic al primului grup nu provine dintr-o slăbiciune de generalizare, ci din răspunsurile parțiale T01-T03. Sistemul gestionează bine inclusiv categoriile noi, în special cazurile-limită rezolvate corect prin refuz.

6.4. Testul 3: Calitatea recuperării RAG

Testul urmărește să identifice în ce măsură erorile rămase provin din modelul generator sau din etapa de recuperare a contextului. Calitatea recuperării a fost evaluată în două etape. În prima etapă, s-a verificat automat, pe baza potrivirii de cuvinte-cheie, dacă fragmentul relevant apare printre primele rezultate returnate de baza vectorială. Metrica folosită este Hit@k: Hit@1 înseamnă că fragmentul corect este primul rezultat returnat, Hit@3 că apare printre primele trei și Hit@5 că apare în primele cinci. MRR (Mean Reciprocal Rank) este media inversului poziției fragmentului corect, un indicator sintetic al calității recuperării. În a doua etapă, pentru întrebările care au rămas problematice a fost inspectat manual conținutul fragmentului efectiv recuperat.

Pe baza estimării automate, un fragment care conține cuvintele-cheie ale întrebării apare în primele cinci rezultate pentru șapte din zece întrebări (Hit@5 = 7/10), iar pentru patru dintre ele pe prima poziție (Hit@1 = 4/10: Q05, Q07, Q08, Q09); MRR estimat este aproximativ 0,52. Metrica verifică însă doar prezența cuvintelor-cheie, nu și dacă fragmentul conține efectiv răspunsul, deci produce fals-pozitive și trebuie tratată ca indicator brut. Cazul cel mai clar este Q05: fragmentul cu expresia „prima lună gratis" ajunge pe prima poziție, deci apare ca Hit@1, dar provine din contextul programului „Invită un prieten", nu din cel al portării, iar răspunsul iese greșit. Același efect apare la Q03 și Q10, care intră în primele cinci rezultate prin potrivire de cuvinte, dar primesc un fragment apropiat tematic și greșit ca precizie. În sens invers, Q01 și Q06 răspund corect deși fragmentul-țintă nu este pe primele poziții: Q01 prin exemplul demonstrativ potrivit, iar Q06 pentru că răspunsul corect este o limită de serviciu pe care modelul o comunică fără a depinde de un fragment exact.

Pentru a clarifica natura erorilor din configurația SLM cu RAG, cele patru întrebări la care răspunsul a fost greșit sau incomplet au fost analizate manual, prin inspectarea conținutului fragmentului efectiv recuperat (Tabelul 6.5).

Tabelul 6.5. Analiza manuală a recuperării pentru întrebările problematice.

Întrebare	Similaritate	Fragment corect?	Interpretare
Q03	0,790	Nu	Fragmentul recuperat este din zona eSIM, dar nu conține informațiile cerute: cost zero, activare în aceeași zi, conexiune internet și compatibilitate.
Q04	0,842	Nu	Fragmentul recuperat este irelevant pentru portarea unei cartele preplătite și se referă la recuperare sau reactivare număr.
Q05	0,740	Parțial	Fragmentul conține expresia „prima lună gratis", dar în contextul programului „Invită un prieten", nu în contextul portării.
Q10	0,830	Parțial	Fragmentul se referă la cazul în care eSIM-ul a fost șters, nu la procedura normală de mutare pe telefon nou.

Toate patru întrebările problematice au primit context cu similarități între 0,74 și 0,84, dar fragmentul returnat nu a fost cel corect, ceea ce arată că o similaritate semantică ridicată nu garantează că fragmentul conține răspunsul exact. Dintre celelalte șase întrebări, recuperarea a adus fragmentul potrivit pentru patru (Q02, Q07, Q08, Q09), iar Q01 și Q06 au fost rezolvate fără ca fragmentul-țintă să fie pe primele poziții.

6.4.1. Sensibilitatea la numărul de fragmente recuperate

Pentru a verifica dacă problema poate fi rezolvată prin returnarea mai multor fragmente, au fost comparate mai multe valori pentru numărul de fragmente recuperate (Tabelul 6.6).

Tabelul 6.6. Sensibilitatea recuperării la numărul de fragmente returnate.

Număr de fragmente	Întrebări cu fragment corect
3	6 / 10
5	7 / 10
8	7 / 10
10	7 / 10

Creșterea de la 3 la 5 fragmente aduce o întrebare în plus, dar peste 5 nu se observă o îmbunătățire pe acest corpus. Valoarea actuală de 5 fragmente este, deci, potrivită, iar simpla creștere a numărului de fragmente nu rezolvă problema.

6.4.2. Sensibilitatea la pragul de similaritate

A fost testată și influența pragului de similaritate folosit pentru a decide dacă există context relevant (Tabelul 6.7).

Tabelul 6.7. Sensibilitatea recuperării la pragul de similaritate.

Prag	Întrebări cu context	Întrebări cu fragment corect
0,2	10 / 10	7 / 10
0,3	10 / 10	7 / 10
0,4	10 / 10	7 / 10

La toate pragurile, toate cele zece întrebări primesc context, iar șapte au fragmentul corect. Niciuna dintre similarități nu coboară sub 0,4, ceea ce explică de ce semnalul intern de context slab (prag 0,4) nu se activează niciodată. Cu alte cuvinte, problema nu este dacă sistemul găsește context, ci dacă găsește contextul potrivit.

6.4.3. Analiza strategiei de chunking

Pentru a vedea dacă segmentarea documentelor influențează recuperarea, au fost testate mai multe strategii de chunking pe aceleași documente și aceleași întrebări, cu același criteriu relativ bazat pe potrivire de cuvinte-cheie (Tabelul 6.8).

Cele trei strategii cu dimensiune fixă produc același număr de fragmente (54) deoarece textul extras din PDF este organizat pe pagini, fiecare pagină depășind 1.500 de caractere; algoritmul de fragmentare nu împarte mai departe o bucată prea mare, ci o emite ca atare, cu suprapunere din bucata anterioară. În consecință, diferențele de MRR dintre fix 500/75 (0,60),

Tabelul 6.8. Comparație între strategii de chunking.

Strategie	Fragm.	Hit@1	Hit@3	Hit@5	MRR
fix 500/75	54	5	6	7	0,60
fix 1000/150 (actual)	54	4	6	7	0,52
fix 1500/200	54	5	7	7	0,58
paragraf	54	5	6	7	0,58
qa	150	5	6	7	0,57

fix 1000/150 (0,52) și fix 1500/200 (0,58) reflectă exclusiv efectul parametrului de suprapunere (*overlap*), nu al dimensiunii fragmentului.

Strategia *qa* este singura care segmentează la nivel de intrare semantică, producând 150 de fragmente mai granulare. Diferența de MRR față de strategia actuală este însă mică (0,57 față de 0,52), ceea ce sugerează că pentru acest corpus bine structurat segmentarea semantică nu aduce un câștig semnificativ față de segmentarea la nivel de pagină. Hit@5 rămâne 7/10 la toate cele cinci strategii, confirmând că plafonul de recuperare este robust și independent de modul de fragmentare.

Concluzia este că principala direcție de îmbunătățire nu este alegerea parametrilor de fragmentare, ci calitatea recuperării semantice în sine: introducerea unui pas de reranking, a metadatelor de categorie sau a unei căutări hibride ar putea ridica plafonul de 7/10 acolo unde parametrii de chunking nu au efect.

6.5. Testul 4: Comportamentul mecanismului de fallback

Testul urmărește să verifice în ce măsură fallback-ul către LLM contribuie la calitatea răspunsurilor față de rularea exclusivă pe SLM. Fallback-ul s-a activat în trei scenarii din T01-T10: T02, T05 și T07. În aceste cazuri, răspunsurile generate pe calea LLM sunt mai structurate, mai detaliate și mai bine formulate față de ce ar fi produs SLM-ul singur. În T05, LLM-ul articulează clar pașii de blocare a abonamentului și de înlocuire a SIM-ului într-o formă ușor de urmărit. În T07, confirmarea imposibilității schimbării titularului vine însoțită de o explicație mai completă. În T02, deși informațiile despre documente și luna gratuită lipsesc din contextul recuperat, fallback-ul reușește cel puțin să confirme că portarea se face online prin aplicație, reducând astfel numărul de runde escalate.

Limitele fallback-ului apar în scenariile unde contextul RAG recuperat este nepotrivit. Când fragmentul returnat conține informații incorecte sau parțiale, ambele modele produc răspunsuri incomplete, deoarece niciun model nu poate corecta o sursă greșită. Aceasta nu este o limitare a fallback-ului în sine, ci a pipeline-ului de recuperare, iar mecanismul de fallback rămâne util ca plasă de siguranță pentru scenariile cu complexitate ridicată.

Fallback-ul funcționează ca mecanism de scalare: rezervă modelul capabil pentru situațiile în care SLM-ul semnalează incertitudine sau lipsă de context, menținând latența scăzută pentru cazurile simple. Direcțiile de îmbunătățire vizează rafinarea criteriilor de declanșare, în special

detectarea mai precisă a răspunsurilor incomplete și a contextului RAG slab, astfel încât fallback-ul să se activeze mai consistent în scenariile unde aduce un câștig real.

6.6. Testul 5: Latență și performanță operațională

Testul evaluează timpii de răspuns atât la nivel de conversație (Tabelul 6.9), cât și la nivel de model (Tabelul 6.10). AHT-ul a fost preluat din pagina de statistici pentru conversațiile cu mai multe runde și estimat manual pentru cazurile cu o singură rundă. Latența la nivel de model a fost măsurată direct prin câmpurile de temporizare ale Ollama, după încălzirea modelului (TTFT exclude încărcarea în memorie).

Tabelul 6.9. Rezumatul performanței operaționale observate.

Configurație	Set	Timp/AHT observat
SLM fără RAG	Q01-Q10	aprox. 2-5s per răspuns
SLM cu RAG	Q01-Q10	aprox. 2-3s per răspuns
SLM + RAG + fallback	T01-T10	aprox. 15s medie pe scenariu

Tabelul 6.10. Latența la nivel de model

Model	Rol	TTFT	Viteză
Qwen 2.5 7B	rapid	0,14s	45 tok/s
DeepSeek 7B	rapid	0,14s	59 tok/s
Mistral 7B	rapid	0,06s	46 tok/s
Llama 3.1 8B	rapid	0,18s	44 tok/s
Gemma 3 12B	rapid, ales	0,32s	26 tok/s
Qwen 2.5 14B	rapid	0,13s	23 tok/s
Gemma 3 27B	capabil, ales	0,35s	12 tok/s
Qwen 2.5 32B	capabil	0,41s	9 tok/s
Llama 3.1 70B	capabil	0,51s	4 tok/s

Tabelul 6.10 arată că viteza de generare scade clar cu dimensiunea modelului. Diferența dintre calea rapidă (Gemma 3 12B, 26 tok/s) și calea de fallback (Gemma 3 27B, 12 tok/s) este de aproximativ două ori, ceea ce confirmă cantitativ motivația arhitecturii cu două niveluri. Timpul până la primul token rămâne mic pentru toate modelele după încălzire (sub 0,5 secunde) și crește moderat cu dimensiunea, astfel încât factorul determinant pentru durata răspunsului este viteza de generare, nu latența inițială. Modelele de 7B sunt cele mai rapide, dar, după cum arată secțiunea următoare, viteza brută nu compensează acoperirea factuală mai slabă.

6.7. Testul 6: Comparația modelelor candidate

Pentru a justifica alegerea Gemma 3 12B și Gemma 3 27B, toate modelele candidate au primit același context RAG și același prompt de grounding, fără exemplu few-shot și fără rutare,

astfel încât diferențele observate să provină din modelul generator. Setul a fost Q01-Q10, iar corectitudinea a fost notată manual față de barem. Deoarece few-shot și rutarea sunt dezactivate intenționat, procentele de corectitudine sunt mai mici decât în sistemul real (Testele 1 și 2, unde Gemma 3 obține 60-75%). Comparația rămâne validă deoarece toate modelele au fost evaluate în condiții identice, iar clasamentul relativ este rezultatul relevant (Tabelul 6.11).

Tabelul 6.11. Comparația modelelor rapide (rol SLM) pe Q01-Q10.

Model	Corectitudine	Halucinații	Viteză
Gemma 3 12B (ales)	**35%**	**1**	26 tok/s
Qwen 2.5 14B	35%	2	23 tok/s
Mistral 7B	20%	4	46 tok/s
DeepSeek 7B	20%	6	59 tok/s
Llama 3.1 8B	15%	0	44 tok/s
Qwen 2.5 7B	5%	0	45 tok/s

Gemma 3 12B și Qwen 2.5 14B conduc la egalitate (35%). Departajarea o face Gemma: o singură halucinație față de două, viteză mai bună și o dimensiune mai mică (12B față de 14B). DeepSeek 7B confirmă observația privind limba română: are cele mai multe halucinații (6) și produce text cu greșeli gramaticale. Qwen 2.5 7B escaladează la aproape toate întrebările (5% corectitudine, zero halucinații): este prudent, dar acoperă prea puțin. Mistral 7B este rapid, dar inventează frecvent, de exemplu un abonament inexistent.

Aceeași comparație, pentru rolul de model capabil (fallback), este prezentată în Tabelul 6.12.

Tabelul 6.12. Comparația modelelor capabile (rol LLM) pe Q01-Q10.

Model	Corectitudine	Halucinații	Viteză
Llama 3.1 70B	**50%**	1	4 tok/s
Gemma 3 27B (ales)	40%	**1**	12 tok/s
Qwen 2.5 32B	40%	2	9 tok/s

Pentru rolul de model capabil, Llama 3.1 70B este cel mai precis (50%), dar de aproximativ trei ori mai lent decât Gemma 3 27B (4 față de 12 tok/s) și mult mai mare. Gemma 3 27B egalează Qwen 2.5 32B la corectitudine (40%), cu mai puține halucinații și viteză ușor mai bună. Pentru rolul de fallback on-premise, Gemma 3 27B reprezintă compromisul practic între calitate, viteză și dimensiune, acceptând un decalaj mic de precizie față de modelul de 70B.

O constatare importantă este că dimensiunea modelului aproape că nu schimbă rezultatul: de la 12B la 70B, corectitudinea crește doar de la 35% la 50%. Mai mult, toate modelele, indiferent de dimensiune, greșesc pe aceleași întrebări (Q03, Q04, Q05, Q10), pentru că recuperarea a adus un fragment apropiat tematic, dar greșit sau incomplet ca conținut. Un model mai mare nu poate corecta un context greșit. Această constatare confirmă rezultatul din Secțiunea 6.4: principala zonă de optimizare este pipeline-ul de recuperare, nu modelul generator. Ea justifică atât

alegerea unui model rapid modest, cât și concentrarea eforturilor de îmbunătățire pe componenta RAG.

6.8. Testul 7: Robustețe și cazuri-limită

Testul verifică comportamentul sistemului în situații în care nu trebuie să inventeze răspunsuri: operațiuni imposibile, acțiuni sensibile pe cont, întrebări în afara domeniului și cereri ambigue (Tabelul 6.13).

Tabelul 6.13. Comportament pe cazuri-limită.

Caz	Scenariu	Rezultat observat
Acțiune sensibilă pe cont	T05	Sistemul nu pretinde că poate bloca direct numărul și îndrumă către Serviciul Clienți sau magazin Orange.
Operațiune imposibilă	T07	Sistemul comunică faptul că schimbarea titularului YOXO nu este posibilă.
Întrebare în afara domeniului	T09	Sistemul nu recomandă telefoane concrete și nu inventează prețuri.
Cerere ambiguă	T10	Sistemul cere clarificare la primul mesaj și păstrează contextul pentru răspunsul ulterior.

Comportamentul pe cazuri-limită este, în general, bun. Sistemul evită halucinațiile periculoase în cazurile T05, T07 și T09 și gestionează corect cererea ambiguă din T10. Limitarea principală este clasificarea automată a transferului: simpla menționare a unui canal oficial de suport este uneori interpretată ca transfer, deși răspunsul este corect. În suport clienți, există situații în care răspunsul corect este chiar refuzul unei acțiuni sau direcționarea către un canal oficial; sistemul trebuie să distingă între un transfer real și o îndrumare procedurală corectă.

6.9. Testul 8: Rolul și limitele evaluatorului automat

Indicatorii automați FCR, Transfer și E2E au fost comparați cu notarea manuală pe baza baremului factual, pentru a stabili în ce măsură pot fi folosiți pentru concluzii. Suplimentar, evaluatorul automat a fost rulat de cinci ori pe aceleași conversații, la temperatură mică, pentru a verifica stabilitatea sa.

Sistemul produce două tipuri de indicatori, cu fiabilitate diferită. Indicatorii calculați determinist din `staticData`, precum AHT, numărul de runde, traseul de rutare și sursele RAG, nu depind de o estimare a modelului și sunt de încredere. Indicatorii generați de agentul evaluator, precum FCR, Transfer și E2E, depind de o judecată a modelului și au o fiabilitate mai redusă.

Rulările repetate arată că evaluatorul automat este **consistent**: la temperatură mică, returnează aceleași valori pentru aceeași conversație în toate cele cinci rulări. Consistența este utilă pentru monitorizare, deoarece valorile nu fluctuează aleatoriu între rulări.

Totuși, evaluatorul prezintă un **bias sistematic și explicabil**, nu erori aleatorii. Sunt două tipare clare. Primul: evaluatorul supraestimează FCR atunci când răspunsul este fluent, dar factual incomplet. În configurația SLM fără RAG, de exemplu, evaluatorul marchează multe răspunsuri ca rezolvate, deși notarea manuală arată că niciunul nu acoperă complet cererea.

Aceste tipare sunt o consecință a faptului că agentul evaluator este o instanță din aceeași familie de modele folosită pentru generare (Gemma 3 27B). Evaluarea de tip model-ca-judecător moștenește înclinațiile modelului, în special tendința de a considera corect un răspuns fluent, și nu constituie o verificare independentă. Aceasta este o limitare recunoscută a abordării și este motivul pentru care concluziile lucrării se bazează pe notarea manuală.

6.10. Testul 9: Repetabilitatea răspunsurilor

Modelul rapid (Gemma 3 12B) a fost rulat de cinci ori pe fiecare întrebare din Q01-Q10, folosind același context RAG, pentru a verifica stabilitatea răspunsurilor. Scopul nu a fost refacerea scorului final, ci observarea variației dintre generări.

Patru din zece întrebări au primit răspunsuri identice în toate cele cinci rulări (Q02, Q03, Q08, Q09), fiind răspunsuri stabile de tip escaladare sau formulare fixă. Pentru restul de șase întrebări, variația a fost doar la nivel de formulare, nu la nivel de decizie factuală. De exemplu, Q05 iese greșit în toate rulările, din cauza contextului înșelător, iar Q04 variază în nivelul de detaliu, dar păstrează aceleași fapte. Modelul nu trece aleatoriu de la un răspuns corect la unul greșit.

Această observație susține folosirea unei singure rulări în testele principale, cu precizarea că rezultatele trebuie citite ca indicative. Pentru o evaluare de producție, raportarea mediei și a variației pe mai multe rulări rămâne o extindere firească a protocolului.

6.11. Sinteza rezultatelor

Rezultatele experimentale confirmă utilitatea RAG pentru un chatbot local de suport clienți. Activarea RAG crește corectitudinea medie a modelului rapid de la 10% la 60% și reduce afirmațiile nesusținute de la 18 la 6. Modelul rapid, folosit fără documentație, nu este suficient pentru răspunsuri factuale într-un domeniu specializat. Exemplele demonstrative ajută punctual, pe scenariile recurente pentru care au fost proiectate, fără a înlocui recuperarea.

Pe scenariile conversaționale T01-T10, sistemul complet obține o corectitudine medie de 75%, cu șase scenarii complet corecte din zece. Sistemul gestionează bine cazurile de plată eșuată, telefon furat, schimbare titular, PIN Orange TV Go, întrebare în afara domeniului și cerere ambiguă. Limitările principale apar la întrebări care cer combinarea mai multor detalii procedurale, unde recuperarea RAG returnează un fragment corect ca topic, dar insuficient ca precizie.

Analiza recuperării arată că o parte importantă din erori provin din etapa RAG. Testele de sensibilitate arată că mărirea numărului de fragmente peste cinci nu aduce un câștig clar, iar pragul de similaritate nu separă contextul util de cel doar apropiat. Testul de chunking confirmă

un spațiu de optimizare la nivelul segmentării.

Comparația modelelor candidate susține alegerea modelelor Gemma. Gemma 3 12B oferă cel mai bun echilibru între corectitudine, număr de halucinații și viteză pentru rolul de model rapid, iar Gemma 3 27B reprezintă un compromis practic pentru rolul de fallback local. O constatare transversală importantă este că modelele mai mari nu rezolvă problemele cauzate de un context RAG nepotrivit: toate modelele greșesc pe aceleași întrebări, indiferent de dimensiune. Acest lucru confirmă faptul că zona principală de optimizare este recuperarea, nu modelul generator.

Evaluatorul automat este consistent și util pentru monitorizare, dar are un bias sistematic spre supraestimarea FCR și a Transferului. Indicatorii determiniști rămân de încredere, iar validarea finală a calității se bazează pe notarea manuală. Repetabilitatea modelului rapid este bună: variația dintre rulări apare doar la nivel de formulare, nu de corectitudine.

Per ansamblu, rezultatele susțin obiectivul lucrării: prototipul demonstrează că un chatbot rulat local, bazat pe RAG și orchestrat prin n8n, poate oferi răspunsuri utile și verificabile în scenarii de suport clienți. Limitările observate sunt clare și indică direcții concrete de îmbunătățire: calitatea fragmentării documentelor, introducerea unui pas de reranking, calibrarea criteriilor de fallback și separarea mai bună a transferului real de îndrumarea procedurală.

7. CONCLUZII

Lucrarea a pornit de la o problemă concretă a departamentelor de suport: volume mari de întrebări repetitive, informație fragmentată în documentația internă și nevoia de a oferi răspunsuri rapide, corecte și verificabile. Soluțiile bazate pe modele de limbaj pot susține interacțiuni naturale, dar introduc limitări importante atunci când sunt folosite în organizații care gestionează date sensibile: dependența de servicii cloud externe, riscul apariției afirmațiilor nesusținute și dificultatea de a controla costurile și resursele de calcul.

Prototipul realizat în această lucrare a abordat aceste probleme printr-o arhitectură locală, construită în jurul a trei componente principale: un mecanism RAG pentru ancorarea răspunsurilor în documentația disponibilă, un sistem de rutare adaptivă între un model rapid și un model mai capabil și un strat de orchestrare implementat în n8n. Rezultatul este un sistem conversațional care poate răspunde pe baza documentelor interne, poate păstra continuitatea dialogului și poate decide când este justificată activarea modelului mai puternic.

Raportarea rezultatelor la obiectivele lucrării

Rezultatele obținute confirmă realizarea obiectivului principal al lucrării: dezvoltarea unui prototip funcțional de chatbot conversațional rulat local, fără dependență de servicii cloud pentru generarea răspunsurilor și accesarea documentației. Sistemul primește întrebarea utilizatorului, recuperează fragmente relevante din corpus, generează un răspuns, evaluează dacă este necesară folosirea modelului mai capabil și păstrează contextul conversațional pe parcursul interacțiunii. În acord cu delimitarea funcțională asumată în secțiunea 3.2.1, prototipul are un rol conversațional și informativ, fără a executa operațiuni reale asupra conturilor clienților.

Primul obiectiv specific, implementarea unui mecanism RAG, a fost concretizat printr-un pipeline complet de pregătire, indexare și recuperare a documentației. Testul de ablație prezentat în Tabelul 6.3 arată că activarea RAG crește corectitudinea medie a modelului rapid de la 10% la 60% pe setul Q01-Q10 și reduce numărul afirmațiilor nesusținute de la 18 la 6. Acest rezultat arată că, într-un domeniu specializat, modelul rapid nu este suficient atunci când este folosit izolat, iar conectarea la documentație este esențială pentru răspunsuri factuale și verificabile.

Al doilea obiectiv specific, rutarea inteligentă între modele, a fost realizat printr-un mecanism de fallback bazat pe mai multe semnale de risc. Decizia de folosire a modelului mai capabil nu este luată pe baza unei reguli binare simple, ci pe baza unui scor agregat care ține cont de elemente precum nivelul de încredere, calitatea contextului, complexitatea cererii și semnalele de incertitudine din răspuns. În această arhitectură, fallback-ul are un rol practic important: permite

folosirea modelului rapid în cazurile simple, unde latența redusă este avantajoasă, și activează modelul mai capabil atunci când răspunsul necesită o formulare mai atentă, o sinteză mai bună a contextului sau o gestionare mai robustă a unei cereri complexe.

Prin urmare, fallback-ul nu trebuie privit doar ca o soluție de rezervă, ci ca un mecanism de rutare adaptivă și de control al calității. El contribuie la echilibrul dintre viteză și acuratețe, deoarece sistemul nu folosește permanent modelul mai mare, dar nici nu livrează automat răspunsul modelului rapid atunci când apar semnale că acesta ar putea fi incomplet sau insuficient fundamentat. În scenariile dificile, fallback-ul ajută sistemul să producă răspunsuri mai coerente, mai complete și mai bine aliniate cu istoricul conversației. Rolul său este complementar mecanismului RAG: RAG furnizează baza factuală, iar fallback-ul decide când este necesară o capacitate mai mare de raționare și formulare pentru valorificarea acelei baze factuale.

Al treilea obiectiv specific, evaluarea automată a conversațiilor, a fost realizat printr-un pipeline care combină indicatori determinați direct din datele de execuție cu indicatori estimați de un agent evaluator. Indicatorii precum AHT, numărul de runde, traseul de rutare și sursele RAG sunt calculați determinist, în timp ce indicatorii precum FCR, Transfer Rate și E2E Rate sunt estimați pe baza conținutului conversației. Testul 8 arată că evaluatorul automat este stabil la temperatură redusă și util pentru monitorizare, comparații între versiuni și iterații rapide. Totuși, indicatorii subiectivi trebuie interpretați împreună cu evaluarea manuală, deoarece pot apărea tendințe de supraestimare.

Pe setul de evaluare separat de exemplele demonstrative, sistemul complet, format din model rapid cu RAG și fallback către modelul mai capabil, obține o corectitudine medie de 75% pe scenariile conversaționale T01-T10, cu șase scenarii rezolvate complet corect din zece, conform Tabelului 6.4. Acest rezultat indică faptul că arhitectura propusă poate susține interacțiuni utile, verificabile și coerente în scenarii reprezentative de suport clienți, păstrând în același timp controlul asupra datelor și asupra costurilor de rulare.

Contribuții

Lucrarea aduce mai multe contribuții practice și metodologice. Prima contribuție este proiectarea și implementarea unui strat de orchestrare în n8n, care integrează fluxul conversațional, recuperarea documentației, rutarea adaptivă, memoria conversațională și colectarea indicatorilor de evaluare. Această abordare oferă o structură modulară, în care fiecare componentă poate fi analizată, testată și extinsă separat.

A doua contribuție este realizarea unei comparații între mai multe modele locale, folosind criterii relevante pentru un sistem de suport în limba română: corectitudine, număr de afirmații nesusținute, latență, comportament conversațional și capacitatea de a respecta instrucțiunile. Această comparație a fundamentat alegerea modelelor Gemma 3 12B și Gemma 3 27B pentru configurația finală.

A treia contribuție este implementarea unui pipeline RAG complet, de la pregătirea documentelor și fragmentarea corpusului până la indexarea vectorială și recuperarea fragmentelor relevante. Acest pipeline permite sistemului să își construiască răspunsurile pe baza documentației

disponibile, nu doar pe baza cunoștințelor interne ale modelului.

A patra contribuție este mecanismul de fallback bazat pe scoring ponderat. Spre deosebire de o comutare fixă între modele, mecanismul propus folosește mai multe semnale pentru a decide când este justificată activarea modelului mai capabil. Continuitatea conversației este păstrată prin memoria partajată între agenți, astfel încât trecerea de la modelul rapid la modelul mai capabil să nu întrerupă firul dialogului.

O altă contribuție este mecanismul de selecție dinamică a scenariilor few-shot prin potrivire de cuvinte cheie. Acesta permite includerea unor exemple relevante pentru contextul conversației fără a crește costul de inferență printr-o etapă suplimentară de căutare semantică. În final, lucrarea propune un cadru de evaluare care combină indicatori automați, indicatori determinați direct din execuție și notare manuală pe baza unui barem factual.

Principalele constatări

Testarea prototipului a condus la câteva constatări importante pentru proiectarea sistemelor conversaționale locale bazate pe RAG. Prima constatare este că recuperarea informației are o influență majoră asupra calității răspunsului final. Analiza recuperării arată că unele erori apar atunci când fragmentele aduse din corpus sunt apropiate tematic, dar nu conțin informația exactă cerută de utilizator. În aceste situații, similaritatea semantică ridicată nu garantează automat suficiența contextului. Prin urmare, optimizarea sistemului nu se reduce la alegerea unui model generator mai mare, ci presupune și îmbunătățirea modului în care informația este căutată, filtrată și ordonată înainte de generare.

A doua constatare privește utilitatea mecanismului de fallback. Rezultatele arată că fallback-ul este valoros în arhitectura propusă deoarece permite distribuirea sarcinii între două modele cu roluri diferite. Modelul rapid poate gestiona eficient întrebările simple și recurente, în timp ce modelul mai capabil este activat în situațiile în care conversația are nevoie de o analiză mai atentă, de o formulare mai bine structurată sau de integrarea mai coerentă a contextului. Această separare ajută sistemul să evite folosirea inutilă a modelului mare, dar oferă totuși o cale de creștere a calității atunci când este nevoie.

Fallback-ul contribuie și la robustețea conversațională. Atunci când apar semnale de incertitudine, răspuns incomplet sau complexitate ridicată, sistemul are o etapă suplimentară prin care poate reevalua formularea răspunsului înainte de a-l livra utilizatorului. Acest lucru este important într-un sistem de suport, unde nu este suficient ca răspunsul să fie rapid, ci trebuie să fie clar, complet și aliniat cu documentația disponibilă.

A treia constatare se referă la exemplele demonstrative few-shot. Acestea sunt utile pentru scenariile recurente și pentru stabilizarea stilului de răspuns, dar nu înlocuiesc documentația recuperată. Ele trebuie tratate ca ghidaj de comportament și formulare, nu ca sursă factuală principală.

Direcții de dezvoltare

Direcțiile de dezvoltare rezultă direct din limitările observate în testare. Prioritatea principală este îmbunătățirea recuperării informației, deoarece aceasta determină baza factuală pe care se construiește răspunsul final. O primă direcție este folosirea unei segmentări mai adaptate la granularitatea întrebărilor reale, însoțită de metadate de categorie și de un pas de reordonare (*re-ranking*) după căutarea inițială. O a doua direcție este introducerea căutării hibride, care să combine similaritatea semantică cu potrivirea lexicală, utilă mai ales pentru termeni specifici, coduri, denumiri comerciale sau formulări exacte din documentație. O a treia direcție este recuperarea în mai mulți pași, prin reformularea întrebării sau extinderea căutării atunci când semnalele RAG indică un context insuficient.

Mecanismul de fallback poate fi dezvoltat în paralel prin calibrarea mai fină a semnalelor de risc și prin introducerea unor criterii suplimentare pentru detectarea răspunsurilor incomplete. În loc să fie tratat doar ca o comutare între două modele, fallback-ul poate deveni o etapă mai amplă de control al calității, în care sistemul verifică dacă răspunsul rapid este suficient, dacă trebuie regenerat cu un model mai capabil sau dacă este necesară o recuperare suplimentară de context. Această direcție păstrează avantajul arhitecturii actuale, dar permite o folosire mai precisă a resurselor locale.

Evaluatorul automat poate fi îmbunătățit prin separarea mai clară a unui transfer real de o îndrumare procedurală corectă și prin validarea estimărilor sale față de notare umană, eventual cu un model evaluator dintr-o familie diferită de cea folosită pentru generare. După această validare, cadrul de evaluare poate fi extins cu indicatori suplimentari, precum satisfacția estimată a utilizatorului, relevanța răspunsului și numărul afirmațiilor nesusținute.

Pe termen mai lung, o direcție firească este integrarea cu sisteme interne autentificate, astfel încât prototipul să poată trece de la un rol strict informativ la unul tranzacțional. Această extindere ar trebui realizată gradual, cu măsuri adecvate de securitate, audit, control al accesului și validare umană pentru operațiunile sensibile.

Considerații finale

Prototipul demonstrează că un chatbot rulat complet local, cu răspunsuri ancorate în documentație prin RAG și orchestrare în n8n, poate susține interacțiuni utile și verificabile în scenarii de suport clienți. Arhitectura propusă oferă un compromis practic între calitatea răspunsului, latență, controlul datelor și costurile de rulare. În această arhitectură, fallback-ul are un rol important: nu este doar o soluție de rezervă, ci un mecanism de rutare adaptivă care permite folosirea eficientă a două modele cu roluri diferite.

Rezultatele obținute nu indică un sistem final lipsit de limitări, ci o bază funcțională și extensibilă. Cele mai importante direcții de îmbunătățire sunt creșterea calității recuperării, calibrarea mecanismului de fallback și validarea mai riguroasă a evaluării automate. Prin aceste extinderi, sistemul poate evolua de la un prototip conversațional local la o componentă matură de suport digital, capabilă să ofere răspunsuri precise, trasabile și adaptate nevoilor utilizatorilor.

BIBLIOGRAFIE

[1] Jaime Carbonell, Jade Goldstein, „The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries", *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 335–336 1998.

[2] Gordon V. Cormack, Charles L. A. Clarke, Stefan Buettcher, „Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods", *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 758–759 2009.

[3] DeepSeek-AI, „DeepSeek LLM: Scaling Open-Source Language Models with Longtermism", *arXiv preprint arXiv:2401.02954*, vol. nr. 2024.

[4] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, Hervé Jégou, „The Faiss library", *arXiv preprint arXiv:2401.08281*, vol. nr. 2024.

[5] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, Haofen Wang, „Retrieval-Augmented Generation for Large Language Models: A Survey", *arXiv preprint arXiv:2312.10997*, vol. nr. 2023.

[6] Gemma Team, „Gemma 3 Technical Report", *arXiv preprint arXiv:2503.19786*, vol. nr. 2025.

[7] Genesys, *What is Average Handling Time (AHT)?*, 2026, Disponibil: `https://www.genesys.com/definitions/what-is-average-handling-time-aht` (accesat în 22.1.2026).

[8] Google, *Gemini API Documentation*, 2026, Disponibil: `https://ai.google.dev/gemini-api/docs` (accesat în 22.1.2026).

[9] Aaron Grattafiori et al., „The Llama 3 Herd of Models", *arXiv preprint arXiv:2407.21783*, vol. nr. 2024.

[10] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, William El Sayed, „Mistral 7B", *arXiv preprint arXiv:2310.06825*, vol. nr. 2023.

[11] Jeff Johnson, Matthijs Douze, Hervé Jégou, „Billion-scale similarity search with GPUs”, *IEEE Transactions on Big Data*, vol. 7, nr. 3, pp. 535–547 2021.

[12] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, Wen-tau Yih, „Dense Passage Retrieval for Open-Domain Question Answering”, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* 2020.

[13] LangChain, *LangChain Documentation*, 2026, Disponibil: `https://python.langchain.com/docs/` (accesat în 22.1.2026).

[14] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, Douwe Kiela, „Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”, *Advances in Neural Information Processing Systems (NeurIPS)* 2020.

[15] Dirk Merkel, „Docker: Lightweight Linux Containers for Consistent Development and Deployment”, *Linux Journal*, vol. 2014, nr. 239 2014.

[16] n8n, *n8n Documentation*, 2026, Disponibil: `https://docs.n8n.io/` (accesat în 22.1.2026).

[17] Rodrigo Nogueira, Kyunghyun Cho, „Passage Re-ranking with BERT”, *arXiv preprint arXiv:1901.04085*, vol. nr. 2019.

[18] Ollama, *Ollama GitHub Repository*, 2026, Disponibil: `https://github.com/ollama/ollama` (accesat în 22.1.2026).

[19] Qwen Team, „Qwen2.5 Technical Report”, *arXiv preprint arXiv:2412.15115*, vol. nr. 2024.

[20] SQM Group, *First Call Resolution (FCR): A Comprehensive Guide*, 2026, Disponibil: `https://www.sqmgroup.com/resources/library/blog/fcr-metric-operating-philosophy` (accesat în 8.6.2026).

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, „Attention Is All You Need”, *Advances in Neural Information Processing Systems (NeurIPS)* 2017.

[22] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, Ion Stoica, „Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena”, *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track* 2023.

A. FRAGMENTE DIN SERVICIUL RAG

```
EMBEDDING_MODEL = "qwen3-embedding:4b"
CHUNK_SIZE = 1000
CHUNK_OVERLAP = 150
MIN_CHUNK_SIZE = 50
TOP_K = 5
SIMILARITY_THRESHOLD = 0.3
COLLECTION_NAME = "rag_documents"
max_context_chars = 4000
```

```
def chunk_text(text):
    if len(text) <= CHUNK_SIZE:
        return [text] if len(text.strip()) >= MIN_CHUNK_SIZE else []

    for sep in ["\n\n", "\n", ". ", "? ", "! ", " "]:
        if sep in text:
            break

    chunks, current = [], ""
    for part in text.split(sep):
        piece = part + sep
        if len(current) + len(piece) <= CHUNK_SIZE:
            current += piece
        else:
            if len(current.strip()) >= MIN_CHUNK_SIZE:
                chunks.append(current.strip())
            current = (current[-CHUNK_OVERLAP:] + piece) if CHUNK_OVERLAP else piece

    if len(current.strip()) >= MIN_CHUNK_SIZE:
        chunks.append(current.strip())
    return chunks
```

```
for i in range(len(results["ids"][0])):
    distance = results["distances"][0][i]
    similarity = 1 - (distance / 2)  # din distanta cosinus in similaritate

    if similarity < SIMILARITY_THRESHOLD:
        continue

    meta = results["metadatas"][0][i]
```

```
    chunks.append({
        "text": results["documents"][0][i],
        "source": meta.get("source", "unknown"),
        "category": meta.get("category"),
        "chunk_index": meta.get("chunk_index", 0),
        "similarity": round(similarity, 4),
    })

sims = [c["similarity"] for c in used_chunks]
max_sim = max(sims) if sims else 0.0
avg_sim = sum(sims) / len(sims) if sims else 0.0
strong_matches = sum(1 for s in sims if s > 0.5)

return {
    "context": "\n\n---\n\n".join(parts),
    "sources": sorted(sources),
    "chunk_count": len(parts),
    "has_context": True,
    "chunks": used_chunks,
    "rag_signals": {
        "max_similarity": round(max_sim, 4),
        "avg_similarity": round(avg_sim, 4),
        "strong_matches": strong_matches,
        "has_context": True,
        "chunk_count": len(parts),
    },
}
```

B. SELECȚIA DINAMICĂ A EXEMPLELOR FEW-SHOT

```
const question = $json.chatInput || '';
const qNorm = normalize(question);

const MIN_MATCHES = 2;
let bestMatch = null;
let bestScore = 0;

for (const s of scenarios) {
  const score = s.keywords.filter(k => qNorm.includes(normalize(k))).length;
  if (score > bestScore && score >= MIN_MATCHES) {
    bestScore = score;
    bestMatch = s;
  }
}

let fewShotExample = '';
if (bestMatch) {
  fewShotExample = '\nEXEMPLU DE COMPORTAMENT (intrebari similare):\n' + bestMatch.
    example + '\n';
}
```