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

Sistem de monitorizare a interacțiunilor voicebotilor folosind modele lingvistice mari (LLM)

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
Dr. ing. Lucian GEORGESCU

Absolvent:
Teodora MÎRZAN

2026

Anexa 2

Universitatea Națională de Știință și Tehnologie POLITEHNICA București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației
Programul de masterat **BIOSINF**

TEMA LUCRĂRII DE DISERTAȚIE
a masterandului MÎRZAN A. Teodora, 421-BIOSINF

1. Titlul temei: Sistem de monitorizare a interacțiunilor voicebotilor folosind modele lingvistice mari (LLM)

2. Descrierea temei:
Context
Sistemele conversaționale automate bazate pe modele lingvistice mari (LLM) sunt din ce în ce mai utilizate în domenii precum banking, asistență medicală sau suport clienți. Pe lângă identificarea corectă a intenției utilizatorului, este necesară și monitorizarea calității interacțiunilor: dacă solicitarea este rezolvată, dacă dialogul este coerent și dacă utilizatorul răspunde adecvat la întrebările voicebotului.
Prin utilizarea LLM-urilor în limba română, se pot detecta automat abateri, incoerențe sau rezultate nedorite ale conversației.
Descriere proiect
Proiectul propune dezvoltarea unui sistem de monitorizare a conversațiilor între un voicebot și utilizatori umani, având două componente principale:
C1. Extragerea intențiilor utilizatorului în limba română și C2. Evaluarea conversației folosind LLM-uri pentru: (i)
clasificarea rezultatului final al interacțiunii (soluționat / nesoluționat / întrerupt) și (ii)
analiza corelației perechilor "întrebare-răspuns" și identificarea neconcordanțelor (ex.: voicebotul cere "anul nașterii", iar utilizatorul răspunde cu vârsta sau cu informații irelevante).
Proiectul va integra modulele într-o aplicație vocală demonstrativă construită pe infrastructură
Twilio Zevo STT LLM Zevo TTS, care va permite testarea în timp real a interacțiunilor și colectarea de date conversaționale pentru monitorizare.

3. Contribuția originală:
Activități principale
A1. Sumarizarea principalelor LLM-uri care pot procesa texte în limba română. A2. Colectarea sau generarea unui set de conversații voicebot-utilizator.
A3. Crearea și rafinarea de prompt-uri pentru extragerea automată a intenției.
A4. Crearea și rafinarea de prompt-uri pentru clasificare a rezultatului conversației: soluționat / nesoluționat / întrerupt.
A5. Crearea și rafinarea de prompt-uri pentru identificarea neconcordanțelor între replicile voicebotului și răspunsurile utilizatorului.
A6. Evaluarea mai multor LLM-uri (API și locale) pe setul de conversatii.
A7. Integrarea Twilio Zevo STT Zevo TTS LLM într-o aplicație demo funcțională.
A8. Evaluarea acurateței vs. resurse vs. timp de răspuns și documentarea rezultatelor.
*Cerințe: *
Programare Python,
Cunoștințe de NLP și LLM prompting,
Familiaritate cu API-uri LLM și cu tehnologii STT/TTS,
Noțiuni de analiza conversatiilor și prelucrare text,
Experiență în lucrul cu containere Docker și medii Linux,
Familiaritate cu platforma Twilio și cu serviciile Zevo Tech.

4. Resurse și bibliografie:
Liu, P. et al. – Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in NLP, ACM Computing Surveys, 2023.
Bhandari, Prabin et al. A Survey on Prompting Techniques in LLMs. 2023.
Vatsal, Shubham, Dubey, Harsh. A Survey of Prompt Engineering Methods in Large Language Models for Different NLP Tasks. 2024
Lee, Jemin, et al. "Exploring the Trade-Offs: Quantization Methods, Task Difficulty, and Model Size in Large Language Models From Edge to Giant." arXiv preprint arXiv:2409.11055 (2024).
Deriu, Jan, et al. "Survey on evaluation methods for dialogue systems." Artificial Intelligence Review 54.1 (2021): 755-810.
ICVSI, Laborator 6, Biosinf, UNSTPB.
Twilio – Programmable Voice API Documentation.

5. Data înregistrării temei: 2026-01-15 18:54:03

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

Dr. Lucian Georgescu

Student,
MÎRZAN A. Teodora

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

Decan,
Prof. dr. ing. Mihnea UDREA

Cod Validare: **2edae78c57**

Declaraţie de onestitate academică

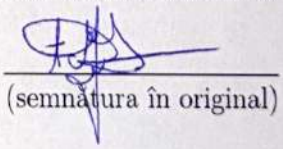
Prin prezenta declar că lucrarea cu titlul *Sistem de monitorizare a interacțiunilor voice-botilor folosind modele lingvistice mari (LLM)*, prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității Naționale de Știință și Tehnologie POLITEHNICA București 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, 15 iunie 2026

Absolvent: Teodora MÎRZAN



(semnătura în original)

CUPRINS

LISTA TABELELOR

LISTA FIGURILOR

LISTA ABREVIERILOR

AI	Artificial Intelligence (lb. en.; inteligență artificială)
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
BIOSINF	Tehnologii multimedia în aplicații de biometrie și securitatea informației
CPU	Central Processing Unit (lb. en.; unitate centrală de procesare)
EN	English (lb. en.; limba engleză)
F1	Scorul F1, media armonică dintre precizie și recall
FN	False Negative (lb. en.; rezultat fals negativ)
FP	False Positive (lb. en.; rezultat fals pozitiv)
GPU	Graphics Processing Unit (lb. en.; unitate de procesare grafică)
GPT	Generative Pre-trained Transformer
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IA	Inteligență artificială
JSON	JavaScript Object Notation
KPI	Key Performance Indicator (lb. en.; indicator-cheie de performanță)
LLM	Large Language Model (lb. en.; model lingvistic mare)
M-F1	Macro-averaged F1 score (lb. en.; scor F1 macro-mediat)
MGSM	Multilingual Grade School Math
MLM	Masked Language Modeling
MMLU	Massive Multitask Language Understanding
NF4	NormalFloat 4-bit
NLI	Natural Language Inference (lb. en.; inferență în limbaj natural)
NLP	Natural Language Processing (lb. en.; procesarea limbajului natural)
RAG	Retrieval-Augmented Generation
RAM	Random Access Memory (lb. en.; memorie cu acces aleatoriu)
RO	Romanian (lb. en.; limba română)
SDK	Software Development Kit
STT	Speech-to-Text (lb. en.; conversia vorbirii în text)
TTFB	Time to First Byte
TTFT	Time to First Token
TTS	Text-to-Speech (lb. en.; sinteză vocală din text)
VRAM	Video Random Access Memory
W-F1	Weighted-averaged F1 score (lb. en.; scor F1 mediat ponderat)
XLM-RoBERTa	Cross-lingual Language Model - Optimized BERT Pretraining Approach
XNLI	Cross-lingual Natural Language Inference

1. INTRODUCERE

1.1. Motivație și context

Asistenții vocali automatizați - numiți în continuare voiceboti - sunt din ce în ce mai prezenți în interacțiunile cu utilizatorii de aplicații (sau site-uri web) din sectorul bancar. Băncile implementează aceste sisteme pentru a gestiona volume mari de apeluri, pentru a reduce timpii de așteptare și pentru a oferi asistență 24/7 pe operații de rutină: verificarea soldului, blocarea sau deblocarea unui card, programarea unei întâlniri la sucursală. Adopția a fost accelerată de progresele în recunoașterea vorbirii și în procesarea limbajului natural, care permit voicebotilor să poarte conversații din ce în ce mai complexe cu utilizatorul final - clientul.

Alegerea acestei teme a fost motivată și de activitatea profesională desfășurată în sectorul bancar, în zona de asistenți conversaționali. Experiența directă cu aceste sisteme a evidențiat provocări concrete ale monitorizării în mediul de producție: volumul de conversații depășește capacitatea de verificare manuală, erorile sistematice pot persista nedetectate perioade îndelungate, iar echipele de dezvoltare au nevoie de feedback structurat și cuantificabil pentru a prioritiza îmbunătățirile. Lucrarea de față se situează la intersecția dintre aceste două domenii, aplicând metode de procesare a limbajului natural pe o problemă identificată în practica profesională.

Această creștere a utilizării ridică însă o întrebare naturală: cum ne asigurăm că voicebotul funcționează corect? O conversație în care voicebotul nu înțelege intenția clientului, furnizează informații greșite sau abandonează dialogul fără rezolvare poate avea consecințe reale - de la frustrarea clientului la pierderi financiare. Monitorizarea calității acestor interacțiuni se face în prezent predominant manual, prin ascultarea eșantioanelor de conversații de către echipe specializate de asigurare a calității. Procesul este lent, costisitor și acoperă inevitabil doar o fracțiune din volumul total de interacțiuni. Problemele sistematice pot trece nedetectate săptămâni întregi, iar feedback-ul ajunge cu întârziere la echipele de dezvoltare.

Progresele recente în domeniul modelelor lingvistice mari (LLM) oferă o alternativă: un model suficient de capabil ar putea prelua o parte semnificativă din acest proces, citind transcrierea unei conversații și identificând automat intenția clientului, statusul final al interacțiunii și eventualele neconcordanțe din răspunsurile voicebotului. Rămâne însă deschisă întrebarea dacă modelele disponibile în prezent ating nivelul de fiabilitate necesar pentru un astfel de sistem, în special pe conversații în limba română - o limbă mai puțin reprezentată în datele de antrenament ale majorității modelelor.

Această lucrare explorează posibilitatea utilizării modelelor lingvistice mari pentru automatizarea monitorizării voiceboților, evaluând experimental ce modele, configurații și strategii de prompting oferă rezultate suficient de fiabile pentru a fi aplicate în practică.

1.2. Obiective

Obiectivul general al lucrării este testarea fezabilității unui astfel de sistem de monitorizare automată, evaluând ce nivel de performanță poate fi atins pe fiecare sarcină, ce compromisuri implică diferitele configurații în termeni de acuratețe, latență, confidențialitate și resurse hardware, și în ce condiții un astfel de sistem devine viabil pentru utilizare practică. Acest obiectiv se concretizează prin trei direcții complementare, ordonate de la componenta de cercetare la cea de implementare.

Primul obiectiv este evaluarea comparativă a modelelor lingvistice pe sarcini de monitorizare a voicebotilor. Sunt testate opt modele cu arhitecturi, dimensiuni și modalități de acces diferite - de la modele comerciale accesate prin API (OpenAI o3, Gemini 2.5 Flash) la modele open-source rulate local cu cuantizare la 4 biți (Aya Expanse 8B, Mistral 7B, Qwen2.5 3B, RoLLaMA 2 7B) și modele de tip encoder-only fără capacitate generativă (XLM-RoBERTa, RoBERT-base). Evaluarea acoperă trei sarcini complementare de clasificare: extragerea intenției clientului dintr-o conversație (12 clase), clasificarea statusului final al conversației (5 clase: rezolvată, parțial rezolvată, nerezolvată, întreruptă, redirecționată) și detecția neconcordanțelor din răspunsurile voicebotului (6 clase: absența neconcordanțelor, halucinație, răspuns incomplet, răspuns irelevant, contradicție și nealiniere cu contextul conversației). Toate experimentele sunt rulate pe un dataset de 180 de conversații bancare simulate în limba română, adnotate automat și verificate manual pe cele trei sarcini.

Al doilea obiectiv este analiza impactului designului instrucțiunii (promptului). Fiecare model generativ este testat cu patru versiuni succesive de prompt (v1-v4), în limba română și engleză, reflectând un proces iterativ de rafinare: de la instrucțiuni minimale (v1) la prompturi cu definiții detaliate ale claselor, restricții de format și exemple (few-shot, v4). Întrebările pe care le adresăm sunt: cât de mult contează calitatea promptului pentru performanța finală, care este efectul limbii promptului asupra conversațiilor în limba română și cum interacționează complexitatea promptului și capacitatea modelului.

Al treilea obiectiv este construirea unui pipeline funcțional de monitorizare. Pornind de la rezultatele evaluării, lucrarea implementează un sistem complet care primește o conversație ca input și produce clasificările pe cele trei sarcini ca output, folosind cele mai bune configurații model-prompt identificate experimental. Pipeline-ul este demonstrat printr-o aplicație web interactivă care simulează interacțiunea cu un voicebot bancar și afișează rezultatele monitorizării în timp real.

1.3. Contribuții

Principalele contribuții ale lucrării sunt: o evaluare comparativă a opt modele lingvistice pe trei sarcini de clasificare a conversațiilor voicebo̧ților, pe un dataset de conversații bancare în limba română creat și adnotat pe multiple dimensiuni; o metodologie iterativă de proiectare a prompturilor (v1-v4), cu analiza câștigurilor și limitelor pe modele având capacități diferite; identificarea unui plafon de performanță al modelelor locale cuantizate pe sarcini complexe; și o analiză a compromisurilor practice dintre modelele API și cele locale, cu recomandări concrete pentru scenarii de implementare având constrângeri diferite de performanță, confidențialitate și infrastructură.

1.4. Structura lucrării

Lucrarea este organizată după cum urmează.

Capitolul 1 (prezentul capitol) introduce contextul, motivația și obiectivele lucrării.

Capitolul 2 analizează modelele lingvistice mari disponibile pentru limba română, prezintă criteriile de selecție utilizate (capabilități lingvistice, latență, cost, cerințe hardware) și documentează deciziile de proiectare privind definirea sarcinilor de clasificare.

Capitolul 3 descrie metodologia de construire și validare a datasetului conversațional: generarea automată a celor 180 de conversații bancare, procesul de adnotare, revizuirea manuală și distribuția finală a claselor pe cele trei sarcini.

Capitolul 4 detaliază procesul iterativ de inginerie a prompturilor: strategia de design, definirea etichetelor pentru fiecare sarcină și evoluția prin cele patru versiuni succesive (v1-v4), cu motivațiile fiecărei iterații.

Capitolul 5 constituie partea centrală a lucrării. Prezintă rezultatele experimentale pe cele trei sarcini, analiza per clasă, evoluția prompturilor, comparația dintre modele API și locale, impactul limbii, analiza latenței și discuția constatărilor, limitărilor și implicațiilor practice.

Capitolul 6 prezintă implementarea pipeline-ului de monitorizare care integrează cele trei sarcini într-un sistem unificat, demonstrat printr-o aplicație interactivă.

Capitolul 7 prezintă cele mai importante concluzii raportate la obiectivul lucrării, precum și principalele contribuții personale aduse acestei lucrări.

2. ANALIZA ȘI SELECȚIA MODELELOR LINGVISTICE PENTRU LIMBA ROMÂNĂ

2.1. Evoluția modelelor lingvistice mari și fundamente arhitecturale

Tehnicile de procesare a limbajului natural (NLP) au evoluat de-a lungul timpului, de la modele statistice bazate pe n-grame și lanțuri Markov, la rețele neuronale (RNN, LSTM), iar ulterior la arhitecturi de tip Transformer, modelele lingvistice mari (LLM) reprezentând rezultatul anilor de studii în acest domeniu. Introducerea mecanismului de *self-attention* în lucrarea fundamentală a lui Vaswani et al. [1], *Attention Is All You Need*, Secțiunea 3 („Model Architecture") a permis modelarea eficientă a dependențelor pe distanțe lungi (relații gramaticale sau semantice între cuvinte care nu sunt apropiate în propoziție, ci sunt separate de multe alte cuvinte) și reprezentarea contextuală bidirecțională (un cuvânt este interpretat folosind atât contextul din stânga, cât și din dreapta), aspect esențial pentru înțelegerea structurilor sintactice complexe și a relațiilor semantice din limbajul natural.

Pe baza acestei arhitecturi, modelele de tip *pretrained language models* (de exemplu BERT [2], Secțiunea 3 - „Pre-training Tasks", și GPT [3], Secțiunea 2 - „Model and Architecture") au demonstrat că antrenarea pe corpusuri masive permite învățarea unor reprezentări lingvistice generalizabile, reutilizabile ulterior prin fine-tuning sau prompturi pentru sarcini specifice, precum clasificarea intențiilor, analiza actelor de vorbire și urmărirea stării dialogului.

O etapă decisivă în apariția LLM-urilor moderne o constituie trecerea la *foundation models*, antrenate pe date multilingve la scară largă. Raportul tehnic GPT-4 [4] pune în evidență competențele de raționament dezvoltate de modele, precum și înțelegerea contextuală în limbi multiple în absența unui antrenament dedicat pentru fiecare limbă în parte. În mod similar, fișele de model (*system cards*) pentru GPT-4o și modelele Gemini [5, 6], secțiunile „Multilingual performance" și „Evaluation", subliniază fiabilitatea arhitecturilor Transformer în scenarii multilingve și capacitatea acestora de a menține performanța în sarcini de clasificare semantică și gestionare a dialogului .

Din perspectivă conceptuală, aceste proprietăți arhitecturale sunt direct relevante pentru limbile cu morfologie bogată și topică relativ liberă, precum limba română. Mecanismele de atenție permit captarea relațiilor de acord gramatical, a dependențelor sintactice neadiacente și a variațiilor flexionare, iar antrenarea multilingvă favorizează transferul de cunoștințe între limbi diferite. Astfel, înainte de a analiza comparativ modelele concrete selectate, este justificată plasarea acestora în cadrul evoluției generale a LLM-urilor și a principiilor arhitecturale care le conferă capacitatea de a susține sarcini precum extragerea intențiilor și clasificarea deznodământului conversației în contexte multilingve.

Performanța unui sistem de monitorizare a interacțiunilor unui voicebot în timp real este direct legată de capacitatea modelului lingvistic ales (LLM) de a procesa particularitățile

semantice și sintactice ale limbii române, fiind evaluată prin trei piloni fundamentali:

– Eficiența tokenizării și reprezentarea diacriticelor: sunt cunoscute provocările impuse de limba română la nivel de segmentare a textului, o problemă particulară fiind aceea a sub-reprezentării diacriticelor, ceea ce duce la o fragmentare excesivă în unități sub-lexicale (tokeni). Astfel, ne confruntăm cu o latență mărită și o înțelegere semantică alterată. RoLLaMA are un vocabular optimizat pentru a păstra integritatea morfologică a cuvintelor românești, iar XLM-RoBERTa utilizează un vocabular multilingv de 250.000 de token-uri care acoperă adecvat structurile morfologice ale limbii române.
– Sintaxa flexionară și acordul gramatical: limba română este o limbă cu o flexiune bogată (cazuri gramaticale, genuri, conjugări complexe), spre deosebire de limba engleză, de exemplu. Sistemul de monitorizare trebuie să identifice corect intenția chiar și atunci când utilizatorul folosește forme flexionare variate. Totodată, limba română permite alegerea unei topici libere a frazei, caracteristică limbajului vorbit, non-academic sau necontrolat, provenit din naturalețea limbajului vorbit, captat prin STT.
– Ambiguitatea semantică și pragmatica dialogului: în analiza corelației „întrebare-răspuns", modelul trebuie să descifreze nuanțe semantice subtile. De exemplu, un răspuns precum „Nu am încă 18 ani" la întrebarea voicebotului „Vă rog să îmi spuneți anul nașterii" reprezintă o neconcordanță informațională (utilizatorul oferă vârsta în loc de an), dar este corect din punct de vedere sintactic. Capacitatea de a detecta aceste abateri necesită un model cu un raționament semantic avansat, capabil să facă distincția între o informație relevantă și una redundantă sau eronată.

2.2. Criterii de selecție și metodologia de evaluare

Alegerea modelelor lingvistice pentru acest proiect a fost condiționată de necesitatea unui suport consistent pentru limba română, esențial pentru identificarea corectă a intențiilor utilizatorilor și a nuanțelor semantice din dialogurile vocale. Întrucât obiectivul principal este analiza conceptuală a modului în care arhitecturi LLM diferite gestionează sarcini de tip *intent extraction* și *conversation outcome classification*, selecția a fost realizată la nivel de familii de modele, considerate reprezentative pentru clase distincte de soluții comerciale, multilingve și specializate.

Evaluarea acoperă șapte modele din arhitecturi și categorii de acces distincte (plus RoBERT care a fost eliminat după primul experiment), fiecare ales pentru a reprezenta o clasă diferită de soluții:

- **Modelele OpenAI (familia GPT)** - considerate *benchmark-ul comercial de performanță maximă*, reprezentând stadiul actual al modelelor de frontieră în ceea ce privește acuratețea semantică, capacitatea de raționament și performanța multilingvă. Aceste modele sunt utilizate ca etalon pentru evaluarea teoretică a calității extragerii intențiilor și a interpretării stării conversației.
- **Modelele Google Gemini** - selectate ca *benchmark comercial orientat spre viteză și fereastră mare de context*, relevante pentru scenarii de producție în timp real, unde latența scăzută și capacitatea de a procesa conversații extinse sunt critice pentru monitorizarea dialogurilor și clasificarea corectă a deznodământului acestora.
- **Aya Expanse (familia Aya)** - aleasă ca *model open-source de referință pentru multilingvism*, antrenată explicit pe un număr mare de limbi și optimizată pentru generalizare cross-lingvistică. Această familie de modele permite evaluarea capacității LLM-urilor open-source de talie medie de a gestiona structuri morfosintactice diverse, inclusiv cele specifice limbii române.
- **RoLLaMA** - familie de modele generative de talie medie (aproximativ 7–8 miliarde de

parametri), special adaptate pentru limba română, considerate *pilonul lingvistic local* al studiului. Acestea permit analiza beneficiilor specializării pe corpusuri românești comparativ cu modelele multilingve generale.

- **RoBERT** - model de tip Encoder bazat pe arhitectura BERT, inclus inițial ca referință pentru paradigma clasică de clasificare discriminativă, cu ipoteza că specializarea pe text românesc poate compensa dimensiunea redusă. Această ipoteză a fost infirmată experimental, modelul fiind ulterior înlocuit în setul principal de experimente cu **XLM-RoBERTa XNLI** - un encoder multilingv antrenat explicit pe sarcini de inferență textuală naturală în 15 limbi, inclusiv română, care permite clasificarea zero-shot fără fine-tuning suplimentar.
- **Mistral 7B Instruct** și **Qwen2.5 3B Instruct** - incluse ca punct de comparație suplimentar față de nucleul principal al evaluării. Mistral 7B reprezintă o generație mai recentă de modele open-source de talie medie, cu îmbunătățiri semnificative de eficiență față de modelele echivalente ca dimensiune din generația anterioară. Qwen2.5 3B permite testarea limitei inferioare a dimensiunii modelului - un aspect relevant pentru scenariile cu resurse hardware limitate, unde costul de inferență trebuie minimizat. Dat fiind costul ridicat (în timp) al rulării complete, ambele modele sunt evaluate exclusiv pe configurația identificată ca optimă în experimentele anterioare.

Metodologia de evaluare vizează compararea celor cinci + două modele pe baza următorilor indicatori tehnici:

- **Capabilități lingvistice teoretice:** Analiza performanței modelelor pe baza rezultatelor raportate în benchmark-urile multilingve (precum MMLU sau MGSM) pentru limbile non-engleze.
- **Profilul de latență (estimare):** Evaluarea timpului de răspuns teoretic declarat pentru API-uri (OpenAI, Gemini) și a latenței de inferență estimate pentru modelele locale.
- **Analiza costurilor operaționale:** Studiu comparativ între structura de tarifare *per-token* și costurile de infrastructură.
- **Corelația între complexitate și arhitectură:** Influența dimensiunii și profunzimii arhitecturii asupra densității informaționale.
- **Cerințe tehnice de implementare și operare:** Constrângeri hardware și software pentru rulare stabilă.

2.2.1. Completare

Este important de menționat că setul final de modele evaluate diferă parțial față de selecția inițială prezentată în acest capitol. RoBERT-base a fost exclus după taskul de extragere a intenției, ca urmare a performanței la nivel de clasificare aleatorie ($\kappa = -0{,}073$), cauzată de incompatibilitatea arhitecturii MLM cu clasificarea zero-shot în absența fine-tuning-ului. În locul său, XLM-RoBERTa XNLI – un encoder multilingv antrenat explicit pe inferență textuală - a fost inclus ca reprezentant al paradigmei encoder-only. Mistral 7B Instruct și Qwen2.5 3B Instruct au fost adăugate ulterior ca punct de comparație suplimentar, evaluate exclusiv pe configurația optimă identificată experimental.

2.2.2. Capabilități lingvistice teoretice

Tabelul 2.1: Capabilități lingvistice teoretice: performanță pe benchmark-ul multilingv MMLU (non-EN) și repere raportate pentru limba română.

Model	**MMLU multilingv (non-EN)**	**Repere pentru limba română**
OpenAI (*o3-high*) [5]	88,8% (media pe 13 limbi, 0-shot)	–
Gemini (*Gemini 2.5 Flash*) [6]	83,4% (Global MMLU Lite, 15 limbi)	–
Aya Expanse 8B [7]	53,7% (Global-MMLU, 5-shot)	–
Mistral 7B Instruct [8]	60,1% (MMLU, 5-shot)	–
Qwen2.5 3B Instruct [9]	74,2% (MMLU, 5-shot, baza 3B)	–
RoLLaMA (*RoLlama2-7b-Instruct*) [10]	–	MMLU-RO: 37,41%
XLM-RoBERTa XNLI [11]	83,6% (XNLI mediu, 15 limbi)	–
RoBERT-base [12]	–	Diacritice (WordAll): 99,78

Tabelul 2.1 sintetizează capabilitățile lingvistice teoretice ale modelelor analizate, prin raportarea performanței pe benchmark-ul multilingv MMLU pentru limbile non-engleze și prin includerea unor repere specifice pentru limba română, acolo unde există evaluări dedicate. Valorile din coloana MMLU multilingv indică acuratețea medie obținută pe seturi traduse în mai multe limbi și reflectă capacitatea de raționament și înțelegere semantică în afara limbii principale de antrenament. Celulele nepopulate corespund situațiilor în care dezvoltatorii nu raportează rezultate oficiale comparabile pentru acele modele sau limbi. Benchmark-ul MMLU a fost ales deoarece evaluează în mod unitar cunoștințele și raționamentul pe domenii diverse, fiind un standard recunoscut pentru analiza generalizării cross-lingvistice. Reperele pentru limba română provin din seturi și sarcini specializate (de exemplu, MMLU-RO sau restaurarea diacriticelor) și oferă un punct de ancorare local, permițând evaluarea modului în care fiecare arhitectură gestionează particularitățile morfologice și semantice ale limbii române.

Valorile procentuale din tabel reprezintă acuratețea (accuracy) obținută pe seturile de evaluare, adică proporția de exemple pentru care modelul produce răspunsul corect. Un scor de 88.8% indică faptul că modelul rezolvă corect aproximativ 89 din 100 de itemi de test, în timp ce un scor de 53.7% arată un nivel de performanță semnificativ mai scăzut, corespunzător unui grad mai redus de generalizare semantică și de raționament. Diferențele dintre aceste valori reflectă, prin urmare, niveluri distincte de robustețe lingvistică și cognitivă ale arhitecturilor analizate.

Metrica WordAll în cadrul sarcinii de restaurare a diacriticelor măsoară procentul de cuvinte pentru care toate caracterele au fost reconstruite corect. Astfel, valoarea de 99.78 indică faptul că aproximativ 99.78% dintre cuvinte sunt complet corecte din punct de vedere ortografic,

evidențiind o capacitate aproape perfectă a modelului RoBERT de a captura regularitățile morfologice și fonetice specifice limbii române.

2.2.3. Profilul de latență și considerații pentru sisteme voice-to-voice

Profilul de latență reprezintă un criteriu esențial pentru utilizarea modelelor lingvistice în sisteme conversaționale în timp real, întrucât întârzierea cumulată dintre inițierea cererii și obținerea răspunsului influențează direct naturalețea interacțiunii. Din punct de vedere tehnic, latența poate fi descompusă în două componente principale: *time-to-first-token* (TTFT), care indică momentul în care modelul începe să genereze răspunsul, și timpul total de finalizare, care depinde de debitul de generare (token throughput) și de lungimea output-ului.

Pentru modelele lingvistice accesate prin API-uri comerciale, OpenAI descrie latența ca fiind influențată de mai mulți factori, printre care dimensiunea modelului utilizat, numărul de tokeni generați și modul de livrare a răspunsului. Documentația oficială subliniază că latența totală este corelată atât cu timpul necesar pentru generarea fiecărui token, cât și cu lungimea secvenței de ieșire, motiv pentru care reducerea numărului de tokeni generați și utilizarea mecanismelor de tip *streaming* pot contribui la îmbunătățirea timpului de răspuns perceput de utilizator. Astfel, optimizarea latenței este tratată ca o problemă end-to-end, care nu depinde exclusiv de modelul utilizat, ci și de configurarea cererilor și de modul de interacțiune cu infrastructura API-ului [13].

În cazul modelului Gemini 1.5 Flash, documentația tehnică oficială descrie o arhitectură de tip Transformer de tip decoder, optimizată pentru rularea eficientă pe infrastructură bazată pe TPU-uri. Autorii raportează un debit ridicat de generare, de ordinul sutelor de caractere pe secundă pentru interogările în limba engleză, evidențiind un comportament orientat spre latență redusă în scenarii de servire. În plus, modelul este evaluat cu ferestre de context extrem de mari, ajungând până la milioane de token-uri, ceea ce demonstrează capacitatea sa de a procesa eficient secvențe foarte lungi. Aceste caracteristici conturează un profil orientat spre performanță și eficiență în aplicații care necesită procesarea rapidă a unor volume mari de date [14].

Pentru modelele open-source de talie medie, precum Aya Expanse 8B și RoLLaMA, rapoartele tehnice se concentrează asupra arhitecturii și performanței pe benchmark-uri multilingve, fără a furniza metrici standardizate de latență. Totuși, evaluări de tip benchmarking realizate pentru arhitecturi Decoder-only de dimensiuni similare indică timpi de răspuns de ordinul secundelor pe GPU și o dependență puternică de lungimea contextului și a secvenței generate, ceea ce le plasează, din punct de vedere al latenței, între modelele comerciale de frontieră și soluțiile strict optimizate pentru viteză [7, 10].

Un profil distinct îl au modelele de tip encoder, precum RoBERT., bazate pe arhitectura BERT. Pentru aceste modele, inferența constă într-o singură trecere prin rețea, fără generare autoregresivă, întrucât întreaga secvență de intrare este procesată simultan. Studii dedicate evaluării latenței pentru modele Transformer de tip encoder, utilizate în sarcini de clasificare text, raportează timpi de procesare reduși, de ordinul sutelor de milisecunde pentru secvențe scurte, atât pe CPU, cât și pe GPU [15]. Această latență scăzută, coroborată cu arhitectura non-autoregresivă, recomandă aceste modele pentru aplicații care necesită răspuns rapid, precum identificarea intențiilor sau evaluarea stării conversației [16].

În scenarii de tip *voice-to-voice*, latența percepută de utilizator nu este determinată exclusiv de modelul lingvistic, ci de întregul lanț de procesare. Conform unei analize realizate pe peste 4 milioane de apeluri vocale în producție [17], latența end-to-end a unui agent vocal este suma a mai multor componente secvențiale: recunoașterea vorbirii (STT), cu un interval tipic de

200-400ms, inferența modelului lingvistic, care reprezintă circa 70% din latența totală, și sinteza vocală (TTS), cu un TTFB tipic de 150-500ms. Aceeași sursă raportează o latență mediană în industrie de 1,4-1,7 secunde, de aproximativ 5 ori mai mare decât pragul de 200-300ms considerat natural în conversațiile umane. Pentru un voicebot bancar, unde interacțiunea trebuie să fie fluidă, optimizarea fiecărei componente din acest lanț devine critică.

2.2.4. Analiza costurilor operaționale

Costurile operaționale reprezintă un criteriu definitoriu în alegerea arhitecturii unui sistem bazat pe modele lingvistice, în special pentru aplicații care funcționează continuu și procesează un volum mare de interacțiuni, precum monitorizarea în timp real a conversațiilor unui voicebot. Din punct de vedere economic, se disting două paradigme principale: utilizarea modelelor comerciale accesate prin API, cu tarifare *per-token*, și rularea locală a modelelor, care implică investiții în infrastructură hardware și costuri recurente de operare.

În cazul serviciilor comerciale, structura de tarifare este proporțională cu numărul de tokeni procesați la intrare și la ieșire. Costul total este determinat de volumul de text analizat, de dimensiunea ferestrei de context și de complexitatea modelului utilizat, fiind exprimat, de regulă, ca preț per milion de tokeni. Această abordare permite o estimare clară a cheltuielilor în funcție de trafic și o scalare previzibilă; însă, poate conduce la costuri cumulative ridicate în scenarii cu utilizare intensivă și flux continuu de conversații.

Prin contrast, rularea locală a modelelor presupune o structură de cost bazată pe investiții inițiale în hardware (GPU-uri, memorie, stocare, rețea), urmate de costuri operaționale recurente, precum consumul de energie, răcirea, mentenanța și administrarea infrastructurii. Aceste cheltuieli pot fi descrise prin combinația dintre costuri de capital (CAPEX) și costuri operaționale (OPEX). Deși investiția inițială este semnificativă, costul marginal per inferență poate scădea considerabil la volume mari de procesare, prin amortizarea infrastructurii pe termen lung.

Diferențele fundamentale dintre cele două paradigme sunt sintetizate în Tabelul 2.2.

În contextul unui sistem *voice-to-voice*, costurile operaționale includ nu doar inferența LLM, ci și componentele de recunoaștere automată a vorbirii (STT) și de sinteză vocală (TTS), fiecare dintre acestea putând fi tarifate fie per minut de audio, fie prin costuri de infrastructură locală. Astfel, costul total per conversație rezultă din suma costurilor asociate STT, procesării lingvistice și TTS, iar alegerea între soluții comerciale și soluții locale implică un compromis între flexibilitate, cost pe termen scurt și control economic pe termen lung.

În cazul modelelor comerciale, costurile API sunt stabilite, de regulă, în funcție de numărul de tokeni procesați, iar tarifele diferă în funcție de model și de complexitatea acestuia. Modelele performante au costuri mai ridicate, în special pentru tokenii de ieșire, în timp ce variantele optimizate pentru eficiență sunt considerabil mai accesibile. Totuși, este posibil ca prețurile actuale să reflecte încă o etapă de adopție în care serviciile au fost oferite la costuri relativ reduse, iar tarifele per token să crească în viitor. În acest context, tehnicile de reducere a consumului de tokeni, precum simplificarea prompturilor, limitarea răspunsurilor și utilizarea cache-ului, devin tot mai importante pentru controlul costurilor [18].

Aceste cifre ilustrează faptul că, în aplicații cu trafic mare, costul total al rulării prin API poate deveni semnificativ, ceea ce justifică analiza economică comparativă cu soluțiile locale, unde costurile nu mai depind direct de numărul de tokeni procesați la fiecare interacțiune.

Tabelul 2.2: Comparație conceptuală a costurilor operaționale: modele comerciale prin API vs. rulare locală

Categorie	**Modele comerciale (API, per-token)**	**Modele rulate local (infrastructură proprie)**
Structura costurilor	Plată variabilă în funcție de numărul de tokeni de intrare și ieșire; cost direct proporțional cu volumul de utilizare	Investiție inițială ridicată în hardware (GPU, stocare), urmată de costuri recurente de energie și mentenanță
Scalabilitate	Scalare elastică, instantanee, gestionată de furnizorul cloud	Scalare limitată de capacitatea hardware; extinderea necesită achiziții suplimentare
Cost marginal per inferență	Crește liniar cu volumul de trafic	Scade odată cu creșterea volumului, prin amortizarea infrastructurii
Mentenanță și operare	Gestionate integral de furnizorul de servicii	Necesită echipă tehnică pentru administrare, actualizări și monitorizare
Control asupra datelor	Datele sunt procesate extern, conform politicilor furnizorului	Datele rămân în infrastructura proprie, avantaj pentru confidențialitate și conformitate
Adecvare pentru voicebot la scară mare	Simplu de integrat, dar potențial costisitor pe termen lung	Eficient pentru volume mari și latență controlată, cu investiție inițială semnificativă

2.2.5. Corelația între complexitate și arhitectură

În contextul modelelor lingvistice generative, „complexitatea" unei soluții este adesea aproximată prin scara arhitecturii (număr de parametri, adâncime, lățime) și prin bugetul de antrenare (date și compute). Literatura de specialitate arată că, pentru modele autoregresive de tip Transformer, reducerea funcției de pierdere urmează legi de scalare netede în raport cu dimensiunea modelului, volumul de date și compute-ul utilizat, în timp ce variații precum raportul adâncime -lățime tind să aibă un efect secundar în limite rezonabile [19]. Din această perspectivă, comparația dintre un model „mediu" (de exemplu, RoLLaMA ∼8B parametri) și modele de scară mult mai mare (familia GPT) trebuie citită nu ca o diferență discretă de „calitate semantică", ci ca o diferență de regim: capacitate, cost de inferență și constrângeri de utilizare în timp real.

Un rezultat relevant pentru interpretarea densității informaționale a reprezentărilor este faptul că îmbunătățirile de performanță apar predominant prin creșterea scalei globale (parametri + date + compute), nu prin ajustarea fină a „formei" rețelei [19]. Practic, atunci când bugetul este fix, alegerea arhitecturii devine o problemă de alocare optimă a resurselor: cât de mare este modelul și cât de mult text vede în antrenare. În această linie, concluziile asupra antrenării compute-optime (de tip Chinchilla) indică faptul că, pentru un buget dat, un model mai mic, dar antrenat pe mai multe date (mai multe token-uri), poate depăși modele mai mari, dar sub-antrenate [20]. Această observație este importantă în scenarii aplicate, unde „mărimea" nu garantează performanța, iar eficiența devine criteriu de proiectare.

Raportat la limba română, există și argumente pragmatice care leagă scara modelului de „densitatea informațională” utilă: un model specializat pe română poate valorifica mai bine aceeași capacitate parametrică, deoarece își distribuie reprezentările pe regularități morfo-sintactice și lexicale relevante pentru domeniul țintă. De exemplu, rapoarte tehnice recente pentru familia RoLLaMA arată că antrenarea/finisarea pe date curate și evaluări traduse pentru română aduce câștiguri consistente față de modele generale, inclusiv printr-o conformitate lingvistică mai bună în generare [21, 22].

Prin urmare, pentru un sistem de monitorizare a dialogurilor în timp real, „modelul mai mare” nu este automat alegerea dominantă. Modelele de talie medie, bine aliniate pe română, pot oferi un compromis mai bun între (i) calitatea semantică efectiv utilizabilă, (ii) latență și cost de inferență și (iii) stabilitatea comportamentului lingvistic (respectarea limbii țintă în răspunsuri), în timp ce modelele de scară foarte mare rămân repere de capabilitate generală și raționament, chiar dacă detaliile interne ale arhitecturii nu sunt publicate complet [23].

2.2.6. Cerințe tehnice de implementare și operare

Succesul integrării modelelor lingvistice nu este determinat doar de performanța lor semantică, ci și de îndeplinirea unor condiții tehnice fundamentale pentru viabilitatea sistemului. Prin definirea unor standarde de infrastructură și stabilitate independente de o platformă anume, se urmărește protejarea flexibilității arhitecturale. Astfel, sistemul rămâne deschis ajustărilor ulterioare, fără a fi constrâns de limitările unei implementări specifice.

Din perspectiva infrastructurii de calcul, literatura de specialitate subliniază faptul că modelele lingvistice de mari dimensiuni sunt caracterizate printr-o dependență accentuată de resurse hardware specializate, în special acceleratoare GPU, atât pentru antrenare, cât și pentru inferență, dependență care nu poate doar să se extindă [4].

Un factor major în creșterea eficienței servirii modelelor lingvistice generative este reprezentat de evoluția acceleratoarelor hardware specializate, în strânsă corelare cu optimizări software realizate printr-o abordare de tip co-design hardware -software. Literatura de specialitate evidențiază faptul că adaptarea arhitecturii hardware la fluxul de date specific algoritmilor LLM, precum integrarea memoriei mai aproape de unitățile de procesare sau optimizarea organizării interne a cipurilor, poate conduce la reduceri semnificative ale latenței și consumului energetic. Aceste direcții sunt ilustrate de progresele recente în arhitectura GPU, cum este cazul generației NVIDIA Hopper, care introduce îmbunătățiri la nivelul capacității memoriei HBM și SRAM, al lățimii de bandă, al unităților de calcul și al interconectării interne, cu impact direct asupra procesării modelelor lingvistice de mari dimensiuni. În perspectivă, proiectarea unor acceleratoare hardware adaptate explicit tiparelor de calcul dominante în modelele generative, precum mecanismele de atenție și operațiile tensoriale, este de așteptat să influențeze semnificativ arhitectura și implementarea sistemelor de servire a LLM-urilor [24].

În plus, pentru scenarii de utilizare aplicată, precum monitorizarea interacțiunilor conversaționale, cerințele nu se limitează la posibilitatea de a rula modelul, ci includ și considerente de operare continuă: gestionarea simultană a mai multor cereri, controlul latenței și utilizarea eficientă a memoriei în regim de inferență. Lucrări recente dedicate servirii modelelor lingvistice arată că aceste aspecte pot deveni factori limitativi chiar și în absența unor constrângeri stricte de performanță teoretică, motiv pentru care ele trebuie tratate ca cerințe de proiectare încă din faza de selecție a modelului [25].

2.3. Decizii de proiectare bazate pe indicatorii-cheie de performanță

În vederea proiectării și evaluării unui sistem de monitorizare a dialogurilor bazat pe modele lingvistice mari, este necesară definirea unui set clar de indicatori-cheie de performanță (KPI), care să permită o analiză coerentă și reproductibilă a comportamentului conversațional. Deciziile de proiectare vor fi ghidate de patru categorii principale de KPI: definirea intențiilor utilizatorului în conformitate cu scenariul voicebotului, definirea stării conversației, identificarea neconcordanțelor din răspunsuri și stabilirea unui format standardizat de evaluare. Adoptarea acestor indicatori cu rigurozitate asigură comparabilitatea rezultatelor și evidențiază diferențele de comportament dintre modelele lingvistice utilizate. În acest context, KPI-urile definite permit o evaluare unitară atât a modelelor comerciale de tip OpenAI și Gemini, cât și a modelelor open-source de talie medie, precum Aya Expanse 8B, RoLLaMA și XLM-RoBERTa XNLI, facilitând corelarea deciziilor arhitecturale cu cerințele funcționale ale aplicației. Fiecare indicator va fi detaliat și justificat într-un subsubcapitol dedicat.

2.3.1. Decizii privind definirea intențiilor

În cadrul acestui proiect, sistemul conversațional este conceput ca un voicebot destinat gestionării unor interacțiuni uzuale din domeniul bancar, unde claritatea scopului utilizatorului și predictibilitatea răspunsurilor sunt esențiale. În consecință, definirea intențiilor este realizată printr-un set finit și explicit de clase, aliniate scenariilor tipice de utilizare ale unui serviciu bancar automatizat. Fiecare intenție corespunde unei acțiuni sau unui obiectiv bine definit al utilizatorului și este asociată cu un flux conversațional distinct, care ghidează interacțiunea către o rezolvare clară sau către o redirecționare controlată. Printre intențiile avute în vedere se numără: blocarea cardului bancar, deblocarea cardului, deschiderea unui cont nou, închiderea unui cont existent, verificarea soldului, obținerea extrasului de cont, raportarea unei tranzacții suspecte, modificarea datelor personale, programarea unei întâlniri cu un consilier bancar, resetarea sau recuperarea datelor de autentificare, precum și solicitarea de informații generale despre produse și servicii bancare. Adoptarea unui set închis de intenții contribuie la asigurarea unui comportament predictibil al voicebotului și facilitează analiza automată a dialogurilor, permițând identificarea rapidă a erorilor de clasificare și reducerea ambiguității în interpretarea interacțiunilor, în special în situațiile în care utilizatorii reformulează sau combină cereri. În plus, această abordare oferă un cadru controlat pentru extinderea ulterioară a sistemului, prin adăugarea de noi intenții, fără a afecta structura conversațională existentă. Pentru situațiile în care utilizatorul formulează solicitări care nu se încadrează în setul de intenții definite, precum întrebări de tip conversațional general sau solicitări în afara domeniului bancar, sistemul va utiliza un mecanism de tip *fallback*. Acest mecanism are rolul de a gestiona explicit cazurile de nerecunoaștere a intenției, prin furnizarea unui răspuns neutru care semnalează imposibilitatea procesării cererii în contextul aplicației sau prin redirecționarea utilizatorului către tipuri de solicitări suportate. Introducerea unui astfel de comportament controlat contribuie la menținerea coerenței conversației, previne generarea de răspunsuri eronate sau irelevante și asigură delimitarea clară a domeniului de competență al voicebotului.

2.3.2. Decizii privind definirea stării conversației

Starea conversației este definită și evaluată la nivelul întregului dialog, fiind clasificată într-un set discret de categorii operaționale: rezolvată, parțial rezolvată, nerezolvată, întreruptă

și redirecționată. Fiecare dintre aceste stări este determinată pe baza unor criterii clare, raportate la obiectivul inițial al utilizatorului și la modul în care acesta a fost atins pe parcursul interacțiunii. Adoptarea acestei clasificări permite evaluarea performanței voicebotului dintr-o perspectivă orientată spre rezultat, nu exclusiv spre corectitudinea sau completitudinea răspunsurilor individuale. Analiza stării la nivel de conversație reflectă mai fidel experiența utilizatorului final și oferă un indicator direct al gradului de succes al sistemului în îndeplinirea scopurilor pentru care a fost proiectat, fiind deosebit de relevantă în contextul aplicațiilor bancare, unde rezolvarea eficientă a solicitării este criteriul principal de evaluare.

2.3.3. Decizii privind definirea neconcordanțelor

Pentru monitorizarea calității răspunsurilor generate de voicebot, proiectul introduce o categorie distinctă de neconcordanțe, care include situații în care răspunsurile sunt incomplete, irelevante, contradictorii sau nealiniate cu contextul conversațional. Identificarea acestor neconcordanțe se realizează independent de starea finală a conversației, permițând evidențierea erorilor locale de generare chiar și în cazurile în care dialogul este clasificat, la nivel global, ca fiind rezolvat sau parțial rezolvat. Această separare deliberată între neconcordanțe și statusul conversației este esențială pentru a surprinde cazurile în care chatbotul produce răspunsuri fluent formulate, dar incorecte din punct de vedere semantic, logic sau informațional, inclusiv manifestări de tip halucinație.

În mod particular, categoria neconcordanțelor vizează și răspunsurile incomplete din punct de vedere informațional, în care modelul oferă o valoare aparent corectă, dar omite alternative relevante sau detalii esențiale pentru luarea unei decizii informate de către utilizator. De exemplu, în cazul unei solicitări privind costul unui card bancar, furnizarea unui singur preț fix, fără menționarea existenței unor opțiuni alternative sau a condițiilor asociate, este considerată o informație incompletă, chiar dacă valoarea prezentată este factual corectă. În plus, sunt urmărite și situațiile în care răspunsurile sunt potențial învechite sau insuficient contextualizate, ceea ce poate conduce la interpretări eronate în scenarii operaționale reale. Prin introducerea acestei categorii de analiză, sistemul permite o evaluare mai fină a limitărilor modelelor lingvistice, facilitând distincția între eșecurile structurale ale dialogului și erorile punctuale de generare sau de raționament.

2.3.4. Decizii privind stabilirea formatului răspunsurilor

Formatul răspunsurilor reprezintă un element central pentru asigurarea coerenței conversaționale și pentru facilitarea evaluării automate a interacțiunilor. În cadrul acestui proiect, răspunsurile generate de voicebot sunt constrânse la un format bine definit, care îmbină limbajul natural cu o structură internă standardizată. Această decizie are ca scop reducerea variabilității inutile a răspunsurilor, menținerea unui comportament predictibil și simplificarea procesului de monitorizare și clasificare a dialogurilor.

Formatul ales presupune ca fiecare răspuns să fie aliniat unei intenții clare și să respecte un șablon logic, adaptat tipului de solicitare bancară, incluzând elemente precum confirmarea înțelegerii cererii, furnizarea informației relevante sau indicarea pașilor următori. În cazul în care cererea nu poate fi procesată, răspunsul urmează un format de tip fallback, care semnalează explicit limitările sistemului și ghidează utilizatorul către alternative valide. Prin adoptarea unui format controlat al răspunsurilor, se facilitează atât experiența utilizatorului final, cât și analiza automată a calității răspunsurilor, permițând corelarea directă între intenție, conținutul răspunsului și starea finală a conversației.

Pentru a ilustra această decizie de proiectare, poate fi considerat cazul unei solicitări de blocare a cardului bancar. În acest scenariu, voicebotul generează un răspuns structurat logic, care confirmă înțelegerea cererii și comunică explicit rezultatul acțiunii, de exemplu printr-o formulare de tipul: „Am înțeles că doriți blocarea cardului. Cardul a fost blocat cu succes. Dacă aveți nevoie de emiterea unui card nou, vă pot oferi informații suplimentare.” Un astfel de răspuns respectă formatul stabilit, este concis și orientat spre obiectivul utilizatorului, evitând informațiile redundante sau ambigue. În același timp, structura sa internă permite etichetarea clară a intenției, a tipului de răspuns și a stării conversației, facilitând evaluarea automată a dialogului și corelarea răspunsului cu indicatorii-cheie de performanță definiți în cadrul proiectului.

3. METODOLOGIA DE CONSTRUIRE ȘI VALIDARE A DATASETULUI CONVERSAȚIONAL

Acest capitol prezintă metodologia utilizată pentru construirea și validarea setului de date conversațional (master dataset) care stă la baza evaluării modelelor lingvistice analizate în cadrul lucrării. Întrucât performanța sistemului de monitorizare depinde în mod direct de calitatea și structura datelor utilizate pentru testare, procesul de creare a setului de date a fost proiectat riguros, urmărind atât coerența semantică a conversațiilor, cât și consistența formală a etichetării.

Construirea corpusului a fost realizată printr-o abordare iterativă, ce a combinat generarea automată asistată de model lingvistic, validarea și adnotarea structurală prin scripturi dedicate și revizuirea manuală a instanțelor generate. Capitolul detaliază etapele parcurse, deciziile metodologice adoptate și ajustările efectuate pentru obținerea unei distribuții finale adecvate analizei comparative realizate în capitolele următoare.

3.1. Strategia inițială de construire a datasetului

Strategia inițială de construire a datasetului a avut ca obiectiv definirea unui cadru experimental controlat, care să permită evaluarea diferențiată a performanței modelelor lingvistice în scenarii conversaționale relevante pentru domeniul bancar. Întrucât sistemul propus urmărește monitorizarea calității răspunsurilor generate de voiceboți, a fost necesară proiectarea unui corpus care să reflecte atât situații uzuale, cât și interacțiuni cu grad ridicat de complexitate.

În etapa de planificare, au fost stabilite criterii clare privind dimensiunea datasetului, distribuția pe niveluri de dificultate și caracteristicile structurale ale conversațiilor. Accentul a fost pus pe obținerea unui echilibru între varietatea scenariilor simulate și fezabilitatea procesului de validare manuală.

Strategia adoptată a fost una etapizată, incluzând definirea obiectivului cantitativ, delimitarea conversațiilor simple și complexe și stabilirea regulilor inițiale de construcție.

3.1.1. Obiectivul cantitativ inițial

În etapa de proiectare a datasetului, a fost stabilit un obiectiv inițial de 150 de conversații. Acest prag a fost considerat optim pentru a asigura un volum suficient de date care să permită o analiză comparativă relevantă între modelele evaluate, menținând în același timp fezabilitatea procesului de validare automată și revizuire manuală.

Alegerea acestui număr a urmărit obținerea unui echilibru între diversitatea scenariilor simulate și complexitatea procesului de verificare structurală a instanțelor generate. Un volum semnificativ mai mic ar fi limitat capacitatea de generalizare a concluziilor, în timp ce un volum considerabil mai mare ar fi îngreunat controlul calității și consistența adnotării.

În cadrul acestui obiectiv cantitativ, conversațiile au fost concepute pe două niveluri de dificultate: simple și complexe. Această separare a fost introdusă pentru a permite evaluarea diferențiată a performanței modelelor în funcție de gradul de solicitare cognitivă și de complexitatea fluxului conversațional.

3.1.2. Definirea conversațiilor simple și complexe

În cadrul strategiei de construire a datasetului, conversațiile au fost structurate pe două niveluri de dificultate: simple și complexe. Această delimitare a fost introdusă pentru a permite evaluarea diferențiată a performanței modelelor lingvistice în funcție de complexitatea interacțiunii și de capacitatea acestora de a gestiona contextul conversațional.

Conversațiile simple au fost definite ca interacțiuni scurte, cu un obiectiv clar și un flux conversațional liniar. Acestea au avut între 2 și 6 schimburi de replici, incluzând solicitarea utilizatorului și unul sau mai multe răspunsuri consecutive ale asistentului. Scenariile simple au vizat cerințe directe, fără ambiguități semnificative sau solicitări multiple, fiind concepute pentru a testa corectitudinea factuală, coerența răspunsului și capacitatea modelului de a îndeplini o intenție unică bine definită.

Conversațiile complexe au avut între 6 și 16 turnuri de dialog și au fost proiectate pentru a simula interacțiuni mai apropiate de situațiile reale de utilizare a voiceboților. Acestea au inclus clarificări succesive, reformulări, solicitări compuse sau modificări ale intenției pe parcursul dialogului. În unele cazuri, utilizatorul a furnizat informații incomplete sau ambigue, necesitând integrarea contextului distribuit pe mai multe turnuri.

Conversațiile simple permit verificarea comportamentului de bază al modelului în condiții controlate, în timp ce conversațiile complexe testează capacitatea acestuia de a menține coerența pe termen lung, de a urmări contextul și de a adapta răspunsurile în funcție de evoluția dialogului.

În procesul de generare automată implementat în notebook, nivelul de dificultate a fost specificat explicit în promptul transmis modelului extern, ceea ce a permis controlul numărului de turnuri și al structurii conversației încă din etapa de generare.

În etapa de proiectare a datasetului, s-a urmărit ca fiecare conversație generată să includă explicit informațiile necesare evaluării definite în capitolul anterior, respectiv: intenția principală, eventualele neconcordanțe, mecanismele de tip fallback și statusul final al conversației.

În implementarea din notebook, aceste cerințe au fost impuse direct la nivelul promptului transmis modelului extern, solicitând generarea conversațiilor într-un format JSON standardizat. Fiecare instanță a inclus câmpuri structurale precum `conversation_id`, `difficulty`, `intent`, `turns`, `status` și `status_turn_ids`. Această structură a permis alinierea directă a datasetului cu schema de evaluare stabilită anterior.

Câmpul `intent` a fost utilizat pentru a marca obiectivul principal al utilizatorului, conform scenariilor definite (de exemplu, blocare card, solicitare informații, actualizare date). În mod similar, `status` a reflectat starea finală a conversației (rezolvat, parțial rezolvat, nerezolvat sau redirecționat), permițând corelarea rezultatului dialogului cu performanța modelului evaluat.

3.2. Generarea automată a conversațiilor

Procesul de generare a conversațiilor a fost realizat prin intermediul unui model lingvistic extern, utilizat pentru a produce instanțe structurate conform cerințelor definite anterior.

Această etapă a avut ca obiectiv obținerea unui volum controlat de conversații, respectând atât distribuția stabilită, cât și schema formală a datasetului.

Generarea a fost implementată într-un notebook dedicat, prin apeluri directe către API-ul modelului selectat, iar rezultatele au fost obținute în format JSON standardizat. Structura cererilor, parametrii utilizați și mecanismele de control al consistenței sunt detaliate în subcapitolele următoare.

3.2.1. Necesitatea utilizării unui model extern

Modelele analizate în capitolul anterior (OpenAI, Gemini, AyaExpanse 8B, RoLLaMa și RoBERT) sunt utilizate ulterior în etapa de evaluare comparativă a performanței sistemului de monitorizare propus. Din acest motiv, acestea nu puteau fi folosite în procesul de generare a datasetului. Utilizarea aceluiași model atât pentru construirea setului de test, cât și pentru evaluare ar fi compromis independența experimentală a studiului.

Într-un cadru experimental corect, datele de test trebuie să fie independente de sistemele evaluate, pentru a evita situațiile în care structura lingvistică, distribuția lexicală sau tiparele conversaționale reflectă implicit particularitățile modelului generator. Fiecare model lingvistic prezintă caracteristici proprii de formulare, preferințe stilistice și distribuții probabilistice specifice la nivel de token. Dacă unul dintre modelele evaluate ar fi fost utilizat pentru generarea datasetului, ar fi existat riscul ca acesta să performeze artificial mai bine pe un set de date care reflectă indirect propriile sale tipare de generare.

Prin urmare, utilizarea unui model extern pentru generarea conversațiilor a fost necesară pentru a menține separarea strictă între etapa de construire a datasetului și etapa de evaluare. Această separare contribuie la reducerea riscului de bias metodologic și la asigurarea unei comparații echitabile între modelele analizate ulterior.

Pentru a menține separarea clară între etapa de generare și cea de evaluare, a fost selectat un model extern din familia Claude, dezvoltată de Anthropic. Conform documentației oficiale, Claude reprezintă o familie de modele lingvistice mari disponibile în mai multe variante, între care Haiku și Sonnet [26].

În selecția modelului utilizat pentru generarea datasetului a fost urmărit un echilibru între costul de utilizare, performanța lingvistică și complexitatea reală a sarcinii de generare. Scopul acestei etape nu a fost obținerea celui mai performant model disponibil, ci generarea controlată a unor conversații coerente, structurate conform unui format impus și adecvate nivelului de complexitate definit anterior.

În cadrul familiei Claude sunt disponibile variante diferite, inclusiv Haiku, Sonnet și Opus [26]. Variantele superioare, precum Sonnet sau Opus, sunt descrise în documentația oficială ca fiind destinate sarcinilor cu grad ridicat de raționament și complexitate extinsă. În contextul prezentului proiect, generarea conversațiilor nu a implicat rezolvare de probleme matematice complexe sau raționament avansat, ci producerea de dialoguri controlate într-un domeniu bine delimitat.

Prin urmare, utilizarea unei variante mai mari ar fi implicat un cost suplimentar fără un beneficiu proporțional în raport cu cerințele task-ului. Varianta `claude-haiku-4-5-20251001` a fost selectată deoarece oferă capabilități suficiente pentru generarea conversațiilor coerente, menținând în același timp un consum redus de resurse. Această alegere a permis generarea întregului set inițial de conversații, adnotarea setului inițial precum și completarea ulterioară a setului de date, cu un cost total de 3.98 USD, menținând sustenabilitatea procesului experimental.

3.2.2. Integrarea API pentru generarea datasetului

Generarea automată a conversațiilor a fost realizată prin apeluri HTTP directe către API-ul Anthropic, utilizând biblioteca `requests`, conform implementării din notebook-ul `01_generate_conversations.ipynb`. Autentificarea a fost gestionată printr-o cheie de acces (`ANTHROPIC_API_KEY`) încărcată dintr-un fișier `.env` cu ajutorul bibliotecii `python-dotenv`. Această abordare a permis separarea credențialelor de codul sursă și o configurare reproducibilă a mediului de rulare.

API-ul este accesat prin endpoint-ul: `https://api.anthropic.com/v1/messages`, care primește cereri de tip `POST` cu un corp JSON ce descrie contextul conversațional și parametrii de generare. În implementare, cererea include următoarele headere:

- `x-api-key`: cheia API încărcată din variabilele de mediu;
- `anthropic-version`: versiunea API utilizată (setată explicit în notebook);
- `content-type`: `application/json`.

Corpul cererii (`payload`) este un obiect JSON care specifică modelul utilizat (în acest proiect `claude-haiku-4-5-20251001`), parametri de generare precum `max_tokens` și `temperature`, precum și lista `messages`. Lista `messages` conține mesajele de intrare în format rol–conținut (de exemplu, un mesaj de tip `user` cu instrucțiunile de generare). Notebook-ul include o etapă inițială de testare a cheii API, în care se trimite o cerere minimă către endpoint pentru a verifica funcționarea autentificării și a obține un răspuns valid.

Răspunsul API este returnat tot în format JSON și poate fi procesat programatic pentru a extrage conținutul generat de model. Această integrare a constituit baza etapelor ulterioare de generare automată a conversațiilor în format structurat, detaliate în subcapitolele următoare.

3.2.3. Structura formală a instanțelor JSON

Fiecare conversație generată este reprezentată printr-o instanță JSON standardizată, care constituie unitatea fundamentală a datasetului. Alegerea formatului JSON a fost determinată de ușurința procesării programatice, compatibilitatea cu validările automate și integrarea directă în pipeline-ul de evaluare.

Structura unei instanțe include următoarele câmpuri principale:

- `conversation_id` – identificator unic al conversației;
- `difficulty` – nivelul de dificultate (simplu sau complex);
- `intent` – intenția principală asociată dialogului;
- `turns` – lista ordonată a intervențiilor conversaționale;
- `final_status` – statusul final al conversației (de exemplu: `rezolvata`, `redirectionata`, `nerezolvata`);
- `incongruities` – lista neconcordanțelor identificate (de exemplu: `halucinatie`);
- `annotation_confidence` – scor numeric care reflectă nivelul estimat de încredere al adnotării;
- `annotation_evidence` – structură auxiliară ce conține referințe către turnurile relevante.

Câmpul `turns` este o listă de obiecte JSON, fiecare având cel puțin două elemente: `role` (de exemplu, `user` sau `assistant`) și `text`, reprezentând conținutul intervenției.

Câmpul `annotation_evidence` include două liste de identificatori:

- `status_turn_ids` – turnuri asociate justificării statusului final;
- `incongruity_turn_ids` – turnuri asociate neconcordanțelor identificate.

În procesul de generare automată, identificatorii de turn au fost produși de modelul extern. S-a observat însă o inconsistență în indexare: în unele instanțe, numerotarea turnurilor pornea de la index 0, iar în altele de la index 1. Această variație a impus intervenții de corectare manuală pentru alinierea referințelor la structura efectivă a listei `turns`. Deși aceste ajustări au urmărit menținerea coerenței interne a fiecărei instanțe, este posibil ca unele referințe punctuale la turnuri să nu fie perfect aliniate.

Având în vedere acest aspect, câmpul `status_turn_ids` nu este utilizat în etapa principală de analiză, iar câmpul `incongruity_turn_ids` poate fi ignorat în evaluările viitoare. Elementul esențial pentru analiză rămâne adnotarea la nivel de conversație (status final și tipul de neconcordanță), nu identificarea exactă a poziției turnului în care apare problema.

Prin utilizarea acestei structuri formale, datasetul permite atât evaluarea globală a conversațiilor, cât și, opțional, analiza localizată a erorilor, în funcție de necesitățile experimentale.

3.3. Implementarea procesului în notebook

Procesul de generare și prelucrare a datasetului a fost implementat într-un notebook dedicat, care a permis automatizarea etapelor principale ale fluxului experimental. Utilizarea acestui mediu a facilitat integrarea apelurilor către API, gestionarea parametrilor de generare și procesarea structurată a răspunsurilor obținute.

În cadrul notebook-ului au fost implementate etapele necesare pentru generarea conversațiilor, adnotarea automată a acestora și verificarea consistenței structurale a instanțelor rezultate. Structura notebook-ului a fost organizată în celule distincte, fiecare corespunzând unei etape specifice din procesul de construire a setului de date.

3.3.1. Etapa de generare automată

Etapa de generare automată a conversațiilor a fost implementată într-un notebook dedicat, organizat în mai multe celule de cod care structurează progresiv procesul de construire a datasetului. Primele celule sunt dedicate configurării mediului de execuție și validării accesului la API-ul modelului utilizat pentru generare. În această etapă este încărcată cheia API din variabilele de mediu, iar conexiunea cu serviciul extern este verificată prin trimiterea unei cereri minimale către model. Scopul acestei etape este confirmarea faptului că autentificarea și accesul la endpoint-ul API funcționează corect înainte de inițierea procesului de generare la scară mai mare.

După validarea accesului la API, notebook-ul include o etapă de generare experimentală în care sunt produse câteva conversații simple. Această etapă are rolul de a testa promptul utilizat pentru generare și de a verifica manual coerența rezultatelor. Promptul specifică modelului structura exactă a conversațiilor și formatul JSON în care acestea trebuie returnate. Prin această etapă preliminară se validează faptul că modelul produce răspunsuri conforme cu schema datasetului și respectă restricțiile definite pentru numărul de turnuri și câmpurile structurale.

În etapa următoare este implementată generarea datasetului propriu-zis, conform distribuției stabilite în capitolele anterioare. Notebook-ul generează un total de 150 de conversații, împărțite în două categorii de dificultate: conversații simple și conversații complexe. Pentru fiecare categorie sunt definite limitele numărului de turnuri (2–6 pentru conversații simple și 6–16 pentru conversații complexe), precum și distribuția pe intenții și pe tipuri de status final.

Pentru a gestiona generarea unui număr mare de instanțe și pentru a reduce riscul de erori în răspunsurile modelului, procesul este organizat în batch-uri. Funcția `generate_batch()` este

responsabilă pentru generarea unui grup de conversații cu aceeași intenție și același nivel de dificultate. Această funcție include mecanisme de retry adaptiv în cazul în care modelul nu respectă formatul JSON sau restricțiile impuse de prompt.

Generarea datasetului complet este realizată printr-o funcție suplimentară care coordonează apelurile către `generate_batch()` pentru fiecare combinație de intenție și dificultate. Această funcție permite reluarea procesului de generare în cazul întreruperii execuției sau al apariției unor erori, fără a pierde conversațiile deja produse.

Conversațiile generate sunt salvate incremental în fișiere JSON în directorul dedicat datasetului. Această abordare permite inspectarea manuală a rezultatelor intermediare și facilitează reutilizarea datasetului în etapele ulterioare de adnotare și analiză.

În contextul generării datasetului, solicitarea unui număr mare de conversații într-o singură cerere ar putea conduce la creșterea lungimii ieșirii și la apariția unor erori de generare, precum nerespectarea formatului JSON sau apariția unor structuri incomplete. Pentru a limita aceste situații, procesul de generare a fost organizat în batch-uri de dimensiune redusă (3–5 conversații per apel API). Această abordare reduce lungimea secvenței generate la fiecare cerere și permite verificarea incrementală a rezultatelor, contribuind astfel la menținerea consistenței structurale a datasetului și la controlul calității datelor generate.

3.3.2. Etapa de adnotare automată

După generarea conversațiilor, urmează o etapă separată de adnotare automată, realizată tot cu ajutorul modelului Claude. În această etapă sunt încărcate conversațiile generate anterior, iar pentru fiecare instanță este produsă o etichetare la nivel de conversație. Rezultatul acestei etape este salvat într-un fișier distinct.

Separarea adnotării de procesul de generare a conversațiilor permite tratarea acestor două etape ca procese independente. Conversațiile brute pot fi generate o singură dată, iar adnotarea poate fi reluată ulterior fără a fi necesară regenerarea datasetului, de exemplu atunci când definițiile categoriilor de evaluare sunt ajustate sau rafinate.

Promptul utilizat pentru adnotare este construit din două componente. Prima este un `system prompt` care definește rolul modelului ca evaluator al dialogurilor generate pentru un voicebot bancar și impune returnarea unui obiect JSON valid. A doua componentă este un prompt generat dinamic pentru fiecare conversație, care include identificatorul acesteia, lista completă a turnurilor și definițiile explicite ale categoriilor utilizate în etichetare.

În implementarea curentă, modelul trebuie să determine statusul final al conversației și eventualele neconcordanțe identificate în răspunsurile generate de asistent. Răspunsul este returnat într-o structură JSON care conține câmpurile `conversation_id`, `final_status`, `incongruities`, `annotation_confidence` și o structură auxiliară `annotation_evidence`, care include identificatori ai turnurilor relevante pentru justificarea etichetării.

După generarea acestor adnotări, notebook-ul realizează un pas suplimentar în care rezultatele sunt reunite cu datasetul inițial de conversații. Prin această operație se obține un fișier final de tip `master_dataset_refined_date.json`, care conține atât conversațiile generate, cât și etichetele asociate fiecărei instanțe.

3.4. Revizuirea manuală și curățarea setului de date

Un alt aspect relevant în procesul de generare automată a datasetului este legat de limitările modelelor de limbaj atunci când sunt utilizate pentru generarea unor secvențe lungi sau a unor ieșiri foarte extinse într-un singur apel. Studiul realizat de cercetători de la Stanford și UC

Berkeley, intitulat *Lost in the Middle: How Language Models Use Long Contexts*, arată că performanța modelelor de tip Transformer poate scădea atunci când acestea trebuie să proceseze sau să utilizeze informații distribuite pe secvențe lungi de context, deoarece mecanismul de atenție tinde să acorde mai puțină importanță informațiilor aflate în zona centrală a ferestrei de context [27].

În contextul generării setului de date, aceste limitări pot conduce la apariția unor erori structurale sau semantice în conversațiile produse automat. Din acest motiv, a fost realizată și o etapă de validare manuală a conversațiilor generate. Această etapă a avut rolul de a verifica atât corectitudinea structurală a instanțelor JSON, cât și calitatea lingvistică și coerența conversațiilor.

3.4.1. Identificarea instanțelor neconforme

Validarea manuală a urmărit mai multe aspecte. În primul rând, a fost verificată corectitudinea gramaticală și adecvarea formulărilor în limba română. În timpul acestei analize au fost identificate mai multe tipuri de erori recurente, precum formulări nefirești de tipul „Voi vă” în loc de „Vă voi”, utilizarea incorectă a unor verbe (de exemplu „rugez” în loc de „sugerez”) sau structuri sintactice influențate de topica limbii engleze, cum ar fi utilizarea nenaturală a formei „dumneavoastră” în poziții neobișnuite în propoziție. De asemenea, în anumite conversații au apărut placeholder-e neînlocuite în câmpuri precum adresa de e-mail. Astfel de situații pot apărea deoarece modelele de limbaj sunt antrenate pe corpusuri multilingve și pot transfera tipare sintactice din alte limbi sau pot reproduce exemple incomplete întâlnite în datele de antrenare.

În al doilea rând, validarea a urmărit verificarea coerenței conversațiilor și evitarea generării unor dialoguri prea similare. Generarea automată a unui număr mare de conversații pe baza unor instrucțiuni similare poate conduce la repetarea anumitor scenarii sau formulări, motiv pentru care a fost necesară inspectarea manuală a datasetului pentru a asigura diversitatea situațiilor simulate.

Un alt obiectiv important al validării a fost verificarea corectitudinii etichetelor asociate fiecărei conversații, inclusiv intenția principală (`intent`), nivelul de dificultate (`simple` sau `complex`), statusul final al conversației și tipurile de neconcordanțe identificate. În timpul acestei analize s-a observat că anumite neconcordanțe erau etichetate incorect, probabil ca urmare a modului în care au fost definite inițial în promptul de generare. Aceste observații au condus la rafinarea definițiilor utilizate pentru categoriile de status și pentru tipurile de neconcordanțe, astfel încât acestea să poată fi utilizate ulterior într-un mod mai consistent în etapa de evaluare a modelelor.

În plus, a fost observată o inconsistență în numerotarea turnurilor conversaționale. În unele instanțe, indexarea turnurilor începea de la 0, în timp ce în altele începea de la 1. S-a încercat corectarea acestor referințe pentru identificarea neconcordanțelor, însă ajustarea nu a fost aplicată și în cazul referințelor utilizate pentru verificarea intenției. În etapa de analiză ulterioară, aceste identificatoare nu vor fi utilizate direct, evaluarea fiind realizată la nivel de conversație și nu la nivelul unuei replici de dialog individuale.

3.4.2. Reducerea setului de date la 146 de conversații

În urma procesului de validare structurală și lingvistică, s-a constatat că un număr redus de conversații prezenta probleme semnificative atât la nivelul textului generat, cât și la nivelul

etichetării automate. În aceste cazuri, erorile identificate ar fi necesitat intervenții extensive pentru corectarea formulărilor, refacerea structurii dialogului sau ajustarea adnotărilor asociate.

Pentru a menține consistența datasetului și pentru a evita introducerea unor instanțe modificate excesiv față de forma generată inițial de model, s-a decis eliminarea acestor conversații din setul final. În total, 4 conversații au fost eliminate, rezultând astfel un dataset final format din 146 de instanțe utilizate în etapele ulterioare de analiză și evaluare.

3.4.3. Analiza distribuției datasetului

După finalizarea etapelor de generare și validare a conversațiilor, a fost realizată o analiză a distribuției conversatiilor pentru a evalua modul în care instanțele generate acoperă diferitele categorii definite anterior. În acest scop, au fost calculate distribuțiile pentru câmpuri precum nivelul de dificultate, intenția conversației, statusul final și tipurile de neconcordanțe. Această analiză a avut rolul de a identifica eventuale dezechilibre în dataset și de a determina ce tipuri de conversații trebuie completate pentru a asigura o acoperire adecvată a tuturor scenariilor relevante pe fiecare palier de evaluare.

3.4.4. Distribuția inițială

Distribuția conversațiilor în funcție de intenția principală este prezentată în Tabelul 3.1. Se observă o repartizare relativ echilibrată între majoritatea intențiilor, fiecare fiind reprezentată prin aproximativ 8% din totalul datasetului. Categoria `fallback` apare într-o proporție ușor mai ridicată, deoarece include scenarii în care asistentul nu poate rezolva direct solicitarea utilizatorului și redirecționează conversația.

Tabelul 3.1: Distribuția conversațiilor în funcție de intenție

Intenție	Număr conversații	Procentaj (%)
fallback	16	10.96
update_personal_data	12	8.22
open_account	12	8.22
close_account	12	8.22
schedule_advisor_meeting	12	8.22
get_account_statement	12	8.22
reset_or_recover_auth	12	8.22
report_suspicious_transaction	12	8.22
check_balance	12	8.22
general_product_info	12	8.22
unblock_card	11	7.53
block_card	11	7.53

Distribuția statusurilor finale ale conversațiilor este prezentată în Tabelul 3.2. Majoritatea conversațiilor sunt clasificate ca `rezolvata`, ceea ce indică faptul că scenariile generate reflectă situații în care asistentul poate oferi o soluție completă. Restul instanțelor sunt distribuite între cazuri parțial rezolvate, redirecționate, întrerupte, sau nerezolvate, dar cu o distribuție inconsistentă, oferindu-se prea puține exemple din ultimele 2 categorii menționate.

Tabelul 3.2: Distribuția conversațiilor în funcție de statusul final

Status Final	Număr conversații	Procentaj (%)
rezolvata	85	58.22
partial_rezolvata	26	17.81
redirectionata	22	15.07
întreruptă	9	6.16
nerezolvata	4	2.74

Distribuția în funcție de nivelul de dificultate este prezentată în Tabelul 3.3.

Tabelul 3.3: Distribuția conversațiilor în funcție de dificultate

Dificultate	Număr conversații	Procentaj (%)
complex	76	52.05
simple	70	47.95

Tabelul 3.4 prezintă distribuția tipurilor de neconcordanțe identificate. Se poate observa numărul mic de exemple etichetate cu orice fel de neconcordanță, categoria "irelevant" lipsind cu totul.

Tabelul 3.4: Distribuția tipurilor de neconcordanțe

Neconcordanțe	Număr conversații	Procentaj (%)
halucinatie	6	54.55
incomplet	3	27.27
nealiniat_context	1	9.09
contradictoriu	1	9.09

3.4.5. Rafinarea definițiilor categoriilor de evaluare

Analiza setului de date generat inițial și procesul de validare manuală au evidențiat o serie de limitări legate de modul în care au fost definite categoriile de evaluare utilizate în adnotare. În special, anumite tipuri de neconcordanțe și unele statusuri finale au fost interpretate inconsistent de către modelul utilizat pentru adnotare, ceea ce a condus la etichetări imperfecte sau nealiniate cu intenția inițială a schemei de evaluare.

Pentru a îmbunătăți consistența acestor etichete și pentru a pregăti setul de date pentru etapa de evaluare a modelelor selectate în capitolele următoare, definițiile categoriilor de evaluare au fost rafinate. Acest proces a fost realizat pe baza observațiilor obținute în etapa de validare structurală și a analizei manuale a conversațiilor generate.

3.4.5.1 Regenerarea conversațiilor cu prompt îmbunătățit

Pe baza observațiilor rezultate în urma validării structurale și a analizei manuale a conversațiilor generate inițial, procesul de generare a datasetului a fost reluat într-o formă optimizată. În această etapă au fost rafinate definițiile utilizate pentru statusul final al conversației și pentru tipurile de neconcordanțe, astfel încât acestea să fie mai clare și mai ușor de interpretat de către model.

Spre deosebire de implementarea inițială, în care generarea conversațiilor și adnotarea acestora erau realizate în două etape distincte, în această fază cele două procese au fost integrate într-o singură celulă de execuție în notebook. Această decizie a fost posibilă deoarece structura datasetului, formatul instanțelor JSON și mecanismele de validare structurală erau deja stabilite în etapele anterioare. Prin combinarea generării și adnotării într-un singur pas s-a redus complexitatea pipeline-ului experimental și s-a accelerat procesul de creare a noilor instanțe.

Promptul utilizat pentru această etapă a fost modificat în mai multe moduri. În primul rând, au fost introduse definiții mai explicite pentru fiecare categorie de status final și pentru fiecare tip de neconcordanță, astfel încât modelul să poată diferenția mai clar situațiile în care o conversație este rezolvată complet, parțial rezolvată sau redirecționată. În al doilea rând, au fost adăugate constrângeri suplimentare privind structura JSON a răspunsului și coerența dialogului generat, pentru a reduce apariția unor erori observate în datasetul inițial.

În această etapă, au fost generate conversații suplimentare față de setul de date inițial, cu scopul de a completa anumite categorii insuficient reprezentate în distribuția inițială. Pentru fiecare instanță nouă, modelul a produs simultan atât conversația, cât și etichetele asociate acesteia (intenția, statusul final și eventualele neconcordanțe).

În implementarea utilizată, datasetul inițial conținea 146 de conversații, iar obiectivul stabilit a fost atingerea unui total de 180 de instanțe. Scriptul a calculat automat câte conversații lipsesc pentru fiecare categorie de status final și nivel de dificultate, construind un plan de generare care să completeze exact aceste diferențe. În acest mod, noile conversații generate au fost utilizate pentru a echilibra distribuția datasetului și pentru a asigura reprezentarea tuturor scenariilor relevante.

După generarea acestor instanțe suplimentare, conversațiile noi au fost salvate separat într-un fișier dedicat, iar ulterior au fost reunite cu datasetul existent pentru a forma versiunea finală utilizată în experimentele ulterioare. Distribuția datasetului rezultat a fost recalculată utilizând aceeași metodologie aplicată în analiza inițială, pentru a verifica dacă obiectivele de distribuție stabilite au fost atinse.

3.4.5.2 Distribuția finală a datasetului

După generarea conversațiilor suplimentare și integrarea acestora în datasetul existent, a fost realizată o analiză finală a distribuției instanțelor. Scopul acestei etape a fost verificarea modului în care datasetul rezultat acoperă categoriile relevante pentru evaluarea sistemului de monitorizare propus. Distribuția finală a fost analizată după aceleași dimensiuni utilizate în etapa inițială: intenția conversației, statusul final, nivelul de dificultate și tipurile de neconcordanțe identificate.

Este important de menționat că, la fel ca în cazul datasetului inițial, conversațiile generate automat au fost supuse unei etape de verificare manuală. În timpul acestei validări au fost corectate anumite formulări, ajustate unele etichete și eliminate instanțe problematice. Din acest motiv, distribuția finală nu reproduce perfect valorile țintă stabilite în etapa de generare. De exemplu, în cazul tipurilor de neconcordanțe pot apărea diferențe minore față de distribuția planificată inițial, deoarece anumite instanțe au fost modificate sau eliminate în urma verificării manuale.

Tabelul 3.5 prezintă distribuția conversațiilor în funcție de intenția principală. Se observă o distribuție relativ uniformă între majoritatea intențiilor, fiecare fiind reprezentată prin aproximativ 9% din totalul datasetului. Această distribuție a fost urmărită în mod intenționat pentru a evita supra-reprezentarea anumitor scenarii și pentru a asigura acoperirea tuturor tipurilor de interacțiuni relevante pentru un voicebot bancar.

Tabelul 3.5: Distribuția finală a conversațiilor în funcție de intenție

Intenție	Număr conversații	Procentaj (%)
update_personal_data	16	8.89
fallback	16	8.89
open_account	15	8.33
close_account	15	8.33
schedule_advisor_meeting	15	8.33
get_account_statement	15	8.33
reset_or_recover_auth	15	8.33
report_suspicious_transaction	15	8.33
check_balance	15	8.33
general_product_info	15	8.33
unblock_card	14	7.78
block_card	14	7.78

Distribuția statusurilor finale ale conversațiilor este prezentată în Tabelul 3.6. Majoritatea dialogurilor sunt clasificate ca `rezolvata`, reflectând scenarii în care utilizatorul obține informația sau acțiunea dorită.

Tabelul 3.6: Distribuția finală a conversațiilor în funcție de statusul final

Status Final	Număr conversații	Procentaj (%)
rezolvata	85	47.22
partial_rezolvata	34	18.89
redirectionata	28	15.56
întreruptă	20	11.11
nerezolvata	13	7.22

Tabelul 3.7 prezintă distribuția datasetului în funcție de nivelul de dificultate. Datasetul final este perfect echilibrat între conversații simple și conversații complexe, fiecare categorie reprezentând 50% din totalul instanțelor.

Tabelul 3.7: Distribuția finală a conversațiilor în funcție de dificultate

Dificultate	Număr conversații	Procentaj (%)
simple	90	50.00
complex	90	50.00

În final, Tabelul 3.8 prezintă distribuția tipurilor de neconcordanțe identificate în conversațiile generate. Cea mai frecventă categorie este `halucinatie`, deoarece poate apărea în situații reale mai des decât celelalte neconcordanțe. În total, 36 de conversații conțin cel puțin o neconcordanță, reprezentând 20% din totalul datasetului. Această proporție a fost considerată adecvată pentru a reflecta un scenariu realist de utilizare a unui voicebot, în care majoritatea interacțiunilor sunt rezolvate corect, iar erorile apar doar într-un procent limitat de cazuri.

Tabelul 3.8: Distribuția finală a tipurilor de neconcordanțe

Tipul neconcordanței	**Număr conversații**	**Procentaj (%)**
Fără neconcordanță	144	80.00
Halucinație	10	5.56
Incomplet	8	4.44
Irelevant	7	3.89
Contradictoriu	6	3.33
Nealiniat contextului	5	2.78
Total	**180**	**100.00**

În ansamblu, distribuția finală a datasetului reflectă un echilibru între diferitele tipuri de scenarii conversaționale și între nivelurile de dificultate ale dialogurilor. Structura rezultată oferă un set de date divers și reprezentativ pentru evaluarea sistemului de monitorizare propus și pentru compararea performanței modelelor lingvistice analizate în capitolele următoare.

4. INGINERIA PROMPTURILOR

4.1. Introducere și motivație

Proiectarea unui sistem de monitorizare a interacțiunilor conversaționale bazat pe modele lingvistice mari implică definirea modalității prin care acesta este direcționat să producă răspunsuri structurate, reproductibile și evaluabile automat. În acest context, ingineria prompturilor (*prompt engineering*) reprezintă domeniul care studiază proiectarea sistematică a instrucțiunilor transmise modelelor lingvistice mari, cu scopul de a extrage cunoștințe din acestea într-un mod structurat, fără a necesita reantrenarea sau ajustarea parametrilor interni ai modelului [28].

Această abordare se diferențiază de cele bazate pe *fine-tuning*, care presupun existența unor seturi de date adnotate, resurse computaționale semnificative și cicluri de antrenare repetate pentru fiecare sarcină nouă. Spre deosebire de aceste abordări, ingineria prompturilor operează exclusiv pe cunoștințele încorporate ale modelului, adaptând comportamentul acestuia prin modificarea instrucțiunii de intrare [28]. Mai mult, prin definirea unei funcții de promptare adecvate, modelul devine capabil să rezolve sarcini noi în regim *zero-shot* sau *few-shot*, fără a fi expus la date de antrenare suplimentare [29].

Aceste proprietăți sunt deosebit de valoroase în contextul prezentului proiect, unde cele trei sarcini de monitorizare - extragerea intenției utilizatorului, clasificarea statusului final al conversației și detecția neconcordanțelor din răspunsurile voicebotului - implică nuanțe semantice distincte și necesită iterații rapide de rafinare, fără costuri de reantrenare.

O motivație suplimentară pentru adoptarea acestei abordări este reprezentată de natura datasetului utilizat, prezentat în Capitolul 3. Cele 180 de conversații colectate sunt adnotate la nivel de conversație, nu la nivel de token, ceea ce face ca tehnicile de fine-tuning supervizat să fie mai puțin adecvate în absența unui volum mai mare de date. Abordarea zero-shot prin ingineria prompturilor valorifică direct cunoștințele lingvistice și de raționament acumulate de model în faza de pre-antrenare, permițând adaptarea la scenarii noi cu puține sau chiar fără date adnotate [29].

Din perspectiva cerințelor operaționale ale sistemului, abordarea prin inginerie a prompturilor prezintă avantaje suplimentare în raport cu alternativele bazate pe clasificatoare specializate. În primul rând, același model și același mecanism de inferență pot fi reutilizate pentru toate cele trei sarcini de monitorizare, reducând complexitatea arhitecturală a sistemului. În al doilea rând, prompturile pot fi actualizate sau extinse cu noi categorii de intenții sau tipuri de neconcordanțe fără a necesita reantrenarea modelului, asigurând flexibilitatea sistemului pe termen lung. În al treilea rând, structura explicită a instrucțiunilor permite controlul direct al formatului răspunsului generat, facilitând integrarea automată a predicțiilor în pipeline-ul de evaluare descris în capitolele următoare.

În cadrul acestui capitol sunt descrise strategia generală de proiectare a prompturilor, definirea formală a etichetelor pentru fiecare sarcină, structura detaliată a prompturilor utilizate,

infrastructura de experimentare construită pentru rularea și compararea iterațiilor de prompt, precum și procesul de rafinare iterativă desfășurat pe baza rezultatelor experimentale. Evaluarea comparativă finală a modelelor pe baza predicțiilor obținute este tratată separat, în capitolul următor.

4.2. Strategia de design a prompturilor

Proiectarea prompturilor utilizate în cadrul acestui proiect a urmat o strategie iterativă, structurată în jurul a patru axe principale: alegerea paradigmei de prompting, structura internă a promptului, limba instrucțiunii și rafinarea progresivă prin versiuni succesive.

4.2.1. Paradigma zero-shot ca punct de plecare

Primele prompturi au fost inițiate în regim *zero-shot*, fără includerea unor exemple de conversații etichetate în corpul instrucțiunii. În această paradigmă, modelul primește exclusiv o descriere în limbaj natural a sarcinii și etichetele posibile, fără nicio demonstrație anterioară. Brown et al. demonstrează că modelele lingvistice de mari dimensiuni sunt capabile să rezolve sarcini noi exclusiv pe baza descrierii acestora în limbaj natural, fără exemple de antrenare suplimentare, valorificând cunoștințele acumulate în faza de pre-antrenare [3]. Această proprietate justifică adoptarea regimului zero-shot ca punct de plecare al experimentelor, în condițiile unui dataset de dimensiuni reduse care nu permite alocarea unui subset reprezentativ de exemple per clasă fără a reduce semnificativ volumul disponibil pentru evaluare.

Versiunile finale ale prompturilor (v4) urmau să introducă exemple reprezentative (*few-shot prompting*), permițând astfel compararea directă a performanței zero-shot față de few-shot pentru fiecare model și limbă în parte. Liu et al. documentează că performanța în regim few-shot depășește consistent performanța zero-shot pe sarcini de clasificare, diferența fiind mai pronunțată la modelele de dimensiuni mai mari [29]. Această progresie deliberată de la simplu la complex constituie un protocol experimental reproductibil, permițând izolarea contribuției exemplelor în context față de performanța bazată exclusiv pe descrierea sarcinii.

4.2.2. Principii generale de design

Proiectarea prompturilor a urmat trei principii generale, menținute consistent pe parcursul tuturor versiunilor și pentru toate cele trei taskuri.

Claritate și completitudine a instrucțiunii. Fiecare prompt definește explicit sarcina modelului, clasele posibile și regulile de clasificare, fără a lăsa loc de interpretare. Întrucât modelul nu primește exemple care să îl ghideze, instrucțiunea trebuie să fie suficient de clară încât să nu genereze ambiguitate în alegerea etichetei.

Constrângerea formatului de ieșire. Toate prompturile solicită explicit returnarea unui obiect JSON valid cu câmpurile `intent` (sau echivalentul pentru celelalte taskuri), `confidence` și `reasoning`. Această constrângere urmărește două obiective: facilitarea parsării automate a răspunsurilor în pipeline-ul de evaluare și reducerea variabilității formatului de ieșire între modele.

Separarea instrucțiunii de date. Conversația clasificată este injectată dinamic în prompt, în timp ce instrucțiunile și definițiile etichetelor rămân fixe între rulări. Această separare menține reproductibilitatea experimentelor și permite modificarea independentă a celor două componente.

4.2.3. Structura internă a promptului

La baza fiecărui prompt a stat o structură comună, organizată în secțiuni distincte, care a evoluat ulterior prin rafinări iterative:

- **Rol și obiectiv** - definește identitatea și sarcina modelului în contextul unui sistem voicebot bancar.
- **Etichete** - enumeră clasele posibile cu descrieri explicite pentru fiecare, pentru a reduce ambiguitatea în clasificare.
- **Conversație** - blocul dinamic injectat prin templating Jinja2, care conține turns-urile conversației clasificate.
- **Instrucțiuni** - regulile de clasificare, ajustate iterativ pe baza analizei erorilor din versiunile anterioare.
- **Format de ieșire** - solicită returnarea unui obiect JSON valid cu câmpurile specifice taskului.

Conținutul și formularea fiecărei secțiuni au fost rafinate de-a lungul versiunilor, structura de bază rămânând însă constantă pentru a asigura comparabilitatea între experimente.

Această structură a fost menținută consistentă între toate cele trei taskuri - extragerea intenției, clasificarea statusului final și detecția neconcordanțelor - pentru a asigura comparabilitatea rezultatelor între taskuri și modele.

4.2.4. Sistemul de templating Jinja2 și separarea definițiilor

Toate prompturile sunt implementate ca fișiere `.jinja` încărcate și randate prin biblioteca Jinja2. Această abordare separă logic conținutul instrucțiunii de codul pipeline-ului și permite injectarea dinamică a conversației la momentul inferenței, prin iterarea peste câmpul `turns` al fiecărei instanțe din dataset.

Definițiile etichetelor pentru fiecare task sunt stocate separat în fișiere JSON dedicate - `intent_definitions.json`, `final_status_definitions.json` și `incongruities_definitions.json`. Aceste fișiere reprezintă sursa unică de adevăr pentru semantica etichetelor, asigurând consistența descrierilor între pipeline-ul de evaluare și documentația proiectului. Din fișierele JSON sunt extrase doar numele etichetelor. Descrierile efective ale etichetelor sunt incluse direct în corpul prompturilor `.jinja`, unde pot fi ajustate independent pentru fiecare versiune de prompt, fără a afecta logica pipeline-ului.

Versionarea prompturilor este realizată prin convenția de nume `{lang}_zero_shot_{version}.jinja` - pentru prompturile fără exemple, respectiv `{lang}_few_shot_{version}.jinja` pentru prompturile cu exemple, permițând selectarea combinației dorite - limbă și versiune - ca parametru de configurare în notebook-ul de experimente.

4.2.5. Bilingvismul ca variabilă experimentală

Pentru fiecare task au fost proiectate câte două variante ale fiecărui prompt: una în limba română (`ro`) și una în limba engleză (`en`). Motivația acestei decizii este dublă.

Pe de o parte, etichetele de clasificare sunt definite în limba engleză (`block_card`, `update_personal_data` etc.), ceea ce poate crea un avantaj pentru prompturile în engleză la modelele antrenate predominant pe date în limba de circulație internațională, engleza. Pe de altă parte, conversațiile din dataset sunt în limba română, ceea ce poate favoriza prompturile în română la modelele cu suport multilingv bun, prin reducerea decalajului lingvistic dintre instrucțiune și textul clasificat.

Evaluarea comparativă a celor două variante de limbă testează dacă promptul în română și cel în engleză produc performanțe echivalente - ceea ce ar indica un comportament agnostic față de limba instrucțiunii. Întrucât toate modelele din pipeline sunt multilingve, ambele variante sunt inteligibile pentru fiecare model evaluat.

Un caz particular îl reprezintă RoLLaMA 2 7B, fine-tunat preponderent pe română dar bazat pe LLaMA 2, pre-antrenat pe corpusuri multilingve. Pentru acest model a fost realizat și un experiment adițional cu etichetele traduse în română, aplicat pe versiunea de prompt cu cele mai bune rezultate, pentru a evalua dacă decalajul lingvistic dintre etichetele în engleză și specializarea modelului constituie un factor limitativ al performanței.

4.2.6. Progresie iterativă și rafinare

Fiecare task a beneficiat de patru versiuni de prompt, construite incremental după următorul protocol:

a) **v1 - prompt de bază (zero-shot)**: structura minimală, cu definiții simple ale etichetelor și instrucțiuni de bază.
b) **v2 - rafinare manuală**: pe baza analizei erorilor sistematice din v1 - inspecția predicțiilor greșite per etichetă - au fost ajustate definițiile și instrucțiunile vizând clasele cu rata cea mai mare de confuzie.
c) **v3 - optimizare asistată**: promptul a fost procesat printr-un optimizer de prompturi online, în cazul taskului de extragere a intențiilor utilizând optimizatorul dedicat al OpenAI, obținând o structură mai riguroasă și instrucțiuni mai precise.
d) **v4 - few-shot**: introducerea unor exemple reprezentative în corpul promptului, permițând compararea directă a performanței zero-shot față de few-shot pentru fiecare model.

Cele patru versiuni, în două limbi fiecare, au generat câte opt rulări per model. Suplimentar, pentru taskul de extragere a intențiilor a fost realizat un experiment adițional exclusiv pentru modelul RoLLaMA 2 7B, constând în rularea versiunii cu cele mai bune rezultate cu etichetele traduse în română, pentru a evalua dacă decalajul lingvistic dintre etichetele în engleză și specializarea modelului pe română constituie un factor limitativ al performanței.

4.3. Definirea etichetelor

Înainte de proiectarea prompturilor, a fost necesară definirea formală a etichetelor utilizate pentru fiecare din cele trei taskuri de monitorizare. Fiecare etichetă a fost descrisă printr-o definiție explicită în română și engleză, stocată în fișiere JSON dedicate - `intent_definitions.json`, `final_status_definitions.json` și `incongruities_definitions.json`. Aceste fișiere reprezintă sursa unică de adevăr pentru semantica etichetelor și sunt utilizate în pipeline-ul de evaluare pentru extragerea numelor claselor valide și validarea outputului modelului. Descrierile etichetelor sunt reproduse direct în corpul fișierelor `.jinja`, fără injecție dinamică, pentru a menține controlul explicit asupra formulării instrucțiunilor și a evita erorile de randare.

4.3.1. Etichete de intenție

Taskul de extragere a intenției utilizatorului este formulat ca o problemă de clasificare multi-clasă cu 12 etichete, acoperind scenariile tipice ale unui voicebot bancar. Etichetele acoperă acțiunile concrete pe care un utilizator le poate solicita în cadrul unui dialog bancar

automatizat, plus o clasă specială `fallback` pentru solicitările în afara domeniului bancar sau pentru conversațiile care conțin cel puțin o solicitare nebancar.

Tabelul 4.1: Etichetele de intenție

Etichetă	Descriere
`block_card`	Utilizatorul dorește să blocheze un card pierdut, furat, reținut sau compromis.
`unblock_card`	Utilizatorul dorește să deblocheze un card blocat anterior.
`open_account`	Utilizatorul dorește să deschidă un cont nou.
`close_account`	Utilizatorul dorește să închidă un cont existent.
`check_balance`	Utilizatorul dorește să afle soldul contului sau suma disponibilă.
`get_account_statement`	Utilizatorul dorește să obțină extrasul de cont sau istoricul tranzacțiilor.
`report_suspicious transaction`	Utilizatorul raportează o tranzacție suspectă, frauduloasă sau neautorizată.
`update_personal_data`	Utilizatorul dorește să actualizeze date personale sau de contact asociate contului.
`schedule_advisor meeting`	Utilizatorul dorește să programeze o întâlnire sau o discuție cu un consultant bancar.
`reset_or_recover_auth`	Utilizatorul dorește să reseteze sau să recupereze datele de autentificare: parolă, PIN sau acces în aplicație.
`general_product_info`	Utilizatorul solicită informații generale despre produse, servicii, comisioane sau condiții bancare.
`fallback`	Solicitarea utilizatorului este în afara domeniului bancar sau nu se încadrează în niciuna dintre intențiile suportate.

4.3.2. Etichete de status final

Taskul de clasificare a statusului final al conversației utilizează 5 etichete, definite pe baza modului în care s-a încheiat interacțiunea dintre utilizator și voicebot. Criteriul principal de diferențiere este dacă solicitarea utilizatorului a fost rezolvată și în ce măsură. O distincție importantă este cea dintre `partial_rezolvata` și `rezolvata`: dacă utilizatorul trebuie să efectueze orice acțiune după încheierea conversației, statusul este `partial_rezolvata`. De asemenea, `redirectionata` se aplică doar atunci când redirecționarea este menționată explicit în dialog, nu și în cazul simplei sugestii de a contacta un operator sau de a vizita o sucursală.

Tabelul 4.2: Etichetele de status final

Etichetă	Descriere
`rezolvata`	Cererea utilizatorului a fost rezolvată 100% de către voicebot, fără a fi necesari pași suplimentari din partea utilizatorului.
`nerezolvata`	Cererea utilizatorului nu a putut fi îndeplinită. Voicebotul nu a reușit să ducă acțiunea la final - fie din cauza unei erori, fie pentru că acțiunea solicitată depășește capacitățile voicebotului.
`redirectionata`	Utilizatorul a cerut explicit să fie redirecționat către un operator uman, sau voicebotul a inițiat redirecționarea în cadrul conversației. Redirecționarea este explicită și menționată direct în dialog.
`partial_rezolvata`	Cererea a fost rezolvată parțial. Cel mai frecvent: utilizatorul trebuie să efectueze pași suplimentari după conversație (să semneze un document, să acceseze o aplicație, să se prezinte fizic la bancă). Se aplică și când utilizatorul a avut mai multe cereri și doar o parte au fost rezolvate.
`intrerupta`	Conversația a fost întreruptă înainte ca voicebotul să aibă șansa de a rezolva cererea. Utilizatorul a închis brusc, a spus că revine mai târziu, s-a răzgândit, sau au apărut erori de conectare care au oprit dialogul.

4.3.3. Tipologia neconcordanțelor

Taskul de detecție a neconcordanțelor identifică situațiile în care răspunsurile voicebotului prezintă deficiențe calitative, independent de statusul final al conversației. Sunt definite 5 tipuri de neconcordanțe, acoperind principalele categorii de erori care pot apărea în răspunsurile unui sistem conversațional automatizat.

O distincție importantă este cea dintre `nealiniat_context` și `halucinatie`: primul tip se referă la ignorarea sau interpretarea greșită a informațiilor furnizate explicit de utilizator, în timp ce al doilea vizează inventarea de informații care nu au apărut niciodată în conversație. De exemplu, într-o conversație din dataset (`conv_complex_0056`), utilizatorul a precizat că dorește blocarea cardului de *credit* pentru afaceri, însă voicebotul a confirmat blocarea cardului de *debit* pentru afaceri - o neconcordanță de tip `nealiniat_context`. Prin contrast, într-o altă conversație (`conv_complex_0005`), voicebotul a descris retragerea anticipată dintr-un depozit ca având „un efect catastrofal" - o afirmație exagerată și nejustificată de contextul dialogului, clasificată drept `halucinatie`.

Neconcordanța de tip `irelevant` apare când voicebotul introduce subiecte fără legătură cu cererea utilizatorului. Un exemplu din dataset (`conv_complex_0082`) ilustrează acest tipar: în timp ce utilizatorul solicita închiderea unui cont, voicebotul a început să promoveze oferta de conturi de investiții, ignorând practic solicitarea inițială. Tipul `incomplet` acoperă situațiile în care voicebotul răspunde doar parțial la cerere - în `conv_complex_0016`, utilizatorul a întrebat despre condițiile unui cont de economii pentru afacere, iar voicebotul s-a limitat la a confirma că este posibil, fără a furniza nicio informație despre comisioane sau pași necesari. În final, `contradictoriu` vizează cazurile în care voicebotul se contrazice cu propriile afirmații anterioare - în `conv_complex_0059`, botul a estimat inițial că investigația va dura 24–48 de ore, pentru ca

mai târziu în aceeași conversație să afirme că durează 10–15 zile lucrătoare.

Tabelul 4.3: Tipologia neconcordanțelor

Tip	Descriere
`incomplet`	Răspunsul nu adresează toate cerințele din mesajul utilizatorului; botul a răspuns doar la o parte din întrebări sau cereri.
`irelevant`	Răspunsul nu corespunde cererii sau introduce subiecte fără legătură, ignorând practic solicitarea utilizatorului.
`contradictoriu`	Voicebotul se contrazice cu propriile răspunsuri anterioare din aceeași conversație.
`nealiniat_context`	Voicebotul ignoră informații deja furnizate de utilizator și acționează după propria interpretare, nu după ce s-a solicitat explicit.
`halucinatie`	Voicebotul afirmă fapte sau pași nejustificați de conversație, aparent inventați, identificabili prin afirmații exagerate sau implausibile.

4.4. Design-ul prompturilor și experimentarea

Această secțiune descrie în detaliu procesul de proiectare și experimentare pentru fiecare dintre cele trei taskuri de monitorizare. Pentru fiecare task sunt prezentate iterațiile de prompt de la versiunea de bază până la versiunea finală, împreună cu infrastructura tehnică utilizată pentru rularea experimentelor, configurația acestora și mecanismele de parsare și salvare a rezultatelor.

Cele trei taskuri - extragerea intenției utilizatorului, clasificarea statusului final al conversației și detecția neconcordanțelor - urmează același protocol de experimentare, dar diferă prin numărul de versiuni de prompt iterate și prin specificul outputului așteptat de la model. Taskul de extragere a intenției a beneficiat de patru versiuni de prompt plus un experiment adițional, în timp ce taskurile de clasificare a statusului final și de detecție a neconcordanțelor au fost iterate în trei versiuni. Infrastructura notebook-ului și modulul de evaluare sunt descrise separat pentru fiecare task, evidențiind adaptările specifice față de structura comună.

4.4.1. Extragerea intenției utilizatorului

Promptul de extragere a intenției are ca obiectiv identificarea scopului principal cu care utilizatorul a inițiat interacțiunea cu voicebotul bancar, pe baza întregului conținut al conversației. Sarcina este formulată ca o problemă de clasificare multi-clasă cu 12 etichete, acoperind atât intențiile bancare specifice cât și clasa `fallback`, rezervată conversațiilor care conțin cel puțin o solicitare în afara domeniului bancar.

Principala dificultate a acestui task rezidă în distincția dintre intenția inițială a utilizatorului și evoluția ulterioară a conversației - confirmări, furnizare de date, digresiuni sau cereri secundare - care nu trebuie să modifice clasificarea finală. O a doua sursă de ambiguitate o reprezintă conversațiile mixte, în care utilizatorul formulează atât cereri bancare cât și solicitări în afara domeniului bancar, pentru care clasificarea corectă este `fallback` indiferent de ordinea apariției

acestora în dialog. Definirea precisă a acestor reguli de clasificare și rafinarea lor iterativă pe baza erorilor observate constituie firul central al evoluției prompturilor de la v1 la v4.

Prompturi - v1 până la v4

v1 - Prompt de bază

Prima versiune constituie punctul de plecare zero-shot minimal, proiectată să testeze capacitatea de bază a modelelor de a clasifica intențiile fără instrucțiuni elaborate. Structura promptului urmărește o organizare simplă și directă, fără elemente suplimentare care ar putea introduce zgomot în instrucțiune.

Secțiunile sunt delimitate prin marcatori `##`, convenția standard Markdown pentru anteturi de nivel doi. Alegerea acestui format a fost motivată de faptul că structurile Markdown - inclusiv anteturile ierarhice - reprezintă gramatica organizațională a textelor de calitate înaltă din corpusurile de pre-antrenare ale modelelor lingvistice mari, fiind astfel recunoscute și interpretate corect de acestea fără instrucțiuni suplimentare [30].

Rolul atribuit modelului în blocul introductiv este cel de *asistent de clasificare a intențiilor pentru un sistem voicebot bancar în limba română* - o descriere care ancorează răspunsurile în domeniul bancar și stabilește contextul lingvistic al conversațiilor clasificate. Sarcina este definită explicit: identificarea intenției principale a utilizatorului pe baza întregii conversații, cu precizarea că intenția este determinată de ceea ce utilizatorul a dorit să realizeze la începutul interacțiunii. Rolul și sarcina nu sunt separate în secțiuni distincte în această versiune - ambele sunt exprimate în același bloc introductiv, fără o delimitare formală între identitatea modelului și obiectivul său.

Instrucțiunile din v1 sunt formulate ca trei reguli simple: intenția este stabilită de obicei în primul mesaj al utilizatorului; replicile ulterioare - confirmări, furnizare de date sau mulțumiri - nu schimbă intenția; dacă solicitarea nu se încadrează în niciuna dintre intențiile definite, se folosește eticheta `fallback`. Aceste reguli nu acoperă explicit cazurile mixte și nu constrâng lungimea sau structura raționamentului din câmpul `reasoning`.

Conversația de clasificat este injectată dinamic în prompt prin intermediul motorului de templating Jinja2, printr-un loop care iterează peste câmpul `turns` al fiecărei instanțe din dataset:

```
{% for turn in conversation %}
{{ turn.role | upper }}: {{ turn.text }}
{% endfor %}
```

Filtrului Jinja2 `| upper` transformă rolurile participanților în majuscule - `USER` și `ASSISTANT` - înainte de injectarea în prompt. Această alegere urmărește două obiective. În primul rând, majusculele creează o distincție vizuală clară între eticheta de rol și textul replicii, facilitând delimitarea turn-urilor în contextul conversațional al promptului. În al doilea rând, convențiile de formatare a dialogurilor în corpusurile de pre-antrenare ale modelelor lingvistice - în special în datele provenite din forumuri, documentații tehnice și seturi de date conversaționale - utilizează frecvent majuscule pentru a marca rolurile participanților, ceea ce face ca această convenție să fie ușor recunoscută de modele fără instrucțiuni suplimentare [30].

Formatul de ieșire solicitat este un obiect JSON valid cu trei câmpuri: `intent` - eticheta de intenție selectată din cele 12 clase definite; `confidence` - nivelul de încredere al modelului în clasificarea efectuată, cu trei valori posibile: `high`, `medium` sau `low`; și `reasoning` - o propoziție concisă în care explică alegerea făcută. Modelului i se solicită explicit să răspundă doar cu obiectul JSON, fără explicații suplimentare, fără formatare Markdown și fără preambul. Câmpul

`reasoning` servește unui dublu scop: pe de o parte facilitează depanarea erorilor de clasificare prin inspecția raționamentului modelului, iar pe de altă parte oferă o formă implicită de chain-of-thought care poate îmbunătăți acuratețea clasificării prin forțarea modelului să articuleze explicit motivația alegerii înainte de a o returna. Structura JSON a fost menținută identică în toate versiunile ulterioare ale promptului, asigurând compatibilitatea cu pipeline-ul de parsare și evaluare.

Structura generală a promptului v1 este:

- Introducere - rol și sarcină (nedistincte)
- Etichete de intenție - format bold-paragraph
- Conversație - bloc dinamic Jinja2 cu roluri în majuscule
- Instrucțiuni - 3 reguli simple
- Format de ieșire - obiect JSON

v2 - Rafinare manuală

Trecerea de la v1 la v2 a fost motivată de analiza erorilor sistematice din experimentele inițiale, care a evidențiat două probleme principale. Prima problemă era că modelele clasificau după intenția bancară apărută ulterior în conversație, ignorând solicitările inițiale în afara domeniului bancar - comportament incorect față de definițiile din dataset. A doua problemă era ambiguitatea formulării *„primul mesaj al utilizatorului”* din instrucțiunile v1, care părea calibrată pe un pattern specific din date și nu reflecta o regulă semantică generală.

Modificările din v2 vizează direct aceste două probleme. Definiția etichetei `fallback` a fost reformulată pentru a acoperi explicit cazurile mixte: *„conversația conține cel puțin o solicitare a utilizatorului în afara domeniului bancar, indiferent dacă există și cereri bancare valide în același dialog”* - față de formularea restrictivă din v1, care trata `fallback` ca o clasă de ultimă instanță. Instrucțiunile au fost reformulate pentru a ancora clasificarea în *„scopul cu care utilizatorul a inițiat conversația”*, eliminând referința explicită la primul mesaj. Regula pentru `fallback` a fost promovată din condiție de ultimă instanță în regulă cu prioritate explicită: *„dacă utilizatorul adresează orice solicitare în afara domeniului bancar în orice moment al conversației, folosește fallback”*. Formatul etichetelor rămâne neschimbat față de v1 - bold-paragraph - structura de ansamblu și anteturile `##` fiind păstrate identice.

v3 - Optimizare asistată

A treia versiune a fost obținută prin procesarea promptului v2 cu optimizatorul de prompturi dedicat al OpenAI, aplicat pe versiunea în română, urmată de o adaptare manuală a versiunii în engleză. Restructurarea a adus modificări semnificative atât de format cât și de conținut, fără a altera definițiile etichetelor sau regulile de clasificare stabilite în v2.

La nivel de format, secțiunile au fost reorganizate cu anteturi de nivel `#` în loc de `##`, iar etichetele au trecut din formatul bold-paragraph la liste cu liniuță și etichete în backtick-uri - `- ‘block_card‘: descriere` - mai compact și mai ușor de procesat vizual de către model. Promptul a primit un prefix `Developer:` care semnalează natura sa de instrucțiune de sistem, iar secțiunile **Rol și obiectiv** și **Conversație** au fost delimitate explicit prin anteturi proprii.

La nivel de conținut, au fost introduse trei secțiuni noi absente din v2. Secțiunea **Planificare și verificare** ghidează modelul printr-un raționament în pași înainte de a produce răspunsul: identificarea scopului principal, verificarea prezenței solicitărilor nebancare, ignorarea schimbărilor aparente de intenție produse de confirmări administrative și returnarea unei singure etichete cu justificare. Secțiunea **Verbosity** constrânge explicit lungimea câmpului `reasoning` la o singură propoziție scurtă. Secțiunea **Condiții de oprire** instruiește modelul să nu adauge

text după obiectul JSON - o constrângere relevantă în special pentru modelele care generează explicații suplimentare după răspuns. Secțiunea de output a fost redenumită din `## Output` în `# Format de ieșire`, aliniindu-se convenției de anteturi de nivel `#` adoptate în această versiune.

v4 - Few-shot

A patra versiune introduce o secțiune **Exemple** în corpul promptului, implementată ca un loop Jinja2 care injectează dinamic cele 6 exemple sintetice din fișierul `few_shot_examples.json`. Fiecare exemplu prezintă o conversație completă urmată de obiectul JSON corect de clasificare, conform paradigmei few-shot standard. ăpStructura, instrucțiunile, secțiunile de planificare, verbosity și condiții de oprire sunt identice cu v3 - singura diferență față de versiunea anterioară constând în adăugarea blocului de exemple.

Exemplele au fost construite pentru a acoperi cazuri reprezentative din dataset: o intenție bancară simplă rezolvată (`update_personal_data`), o intenție bancară complexă parțial rezolvată (`close_account`), un `fallback` pur, un `fallback` mixt cu cerere bancară ulterioară, o conversație întreruptă (`open_account`) și o conversație cu o neconcordanță de tip halucinație (`check_balance`). Această distribuție urmărește să exemplifice atât scenariile tipice cât și cazurile limită care au generat cele mai multe erori în versiunile zero-shot.

Fiecare exemplu include câmpul `reasoning` - o propoziție concisă care justifică alegerea etichetei - oferind modelului nu doar demonstrații ale outputului corect, ci și un tipar explicit de raționament pe care să îl reproducă.

Promptul în limba engleză pentru V4 este inclus în anexa finală.

Experiment adițional - etichete în română

Etichetele de intenție utilizate în toate versiunile de prompt sunt definite în limba engleză - `block_card`, `open_account`, `update_personal_data` etc. - reflectând convențiile de nomenclatură standard din industrie și asigurând consistența cu definițiile stocate în fișierul `intent_definitions.json`. Această alegere este naturală pentru modelele comerciale și pentru modelele multilingve din pipeline, care procesează etichetele în engleză fără dificultate indiferent de limba conversației sau a instrucțiunii. Cu toate acestea, ea ridică o potențială problemă specifică pentru RoLLaMA 2 7B - singurul model antrenat preponderent pe date în limba română prin fine-tuning - unde etichetele în engleză introduc un decalaj lingvistic față de distribuția datelor de antrenament. Ipoteza experimentului adițional este că înlocuirea etichetelor cu variante în română poate reduce acest decalaj și influența pozitiv performanța modelului, independent de calitatea instrucțiunilor din prompt.

Pentru a testa această ipoteză, a fost construită o variantă alternativă a promptului v3 - versiunea cu cele mai bune rezultate pe română - în care etichetele de intenție au fost traduse în limba română. Traducerile au urmat o convenție de nomenclatură similară cu cea originală, utilizând underscore ca separator: `blocare_card`, `deschidere_cont`, `actualizare_date_personale` etc. Maparea completă dintre etichetele în engleză și cele în română a fost stocată în fișierul `ro_labels_mapping.json`, iar la evaluare predicțiile au fost mapate înapoi la etichetele originale în engleză înainte de calculul metricilor, asigurând comparabilitatea directă cu celelalte experimente.

Structura promptului `ro_zero_shot_v3_ro_labels.jinja` este identică cu cea a variantei v3 în toate privințele - rol, instrucțiuni, format de ieșire, mecanismul de injecție Jinja2 - singura diferență constând în înlocuirea etichetelor din secțiunea **Etichete de intenție** cu variantele lor în română. Experimentul a fost rulat exclusiv pe RoLLaMA 2 7B cu prompt în română,

rezultatele urmând a fi comparate direct cu experimentul `rollama2_7b__ro__v3` pentru a izola impactul limbii etichetelor asupra performanței modelului.

Infrastructura notebook-ului

Experimentele de extragere a intenției au fost organizate într-un notebook Jupyter structurat în celule cu roluri bine definite, proiectat pentru a permite rularea oricărei combinații de model, limbă și versiune de prompt printr-o singură modificare de configurație. Modelele comerciale OpenAI o3 și Gemini 2.5 Flash au fost accesate prin API-urile oficiale ale furnizorilor, în timp ce modelele open-weight - Aya Expanse 8B și modelele encoder - au fost rulate local. Cu excepția modelului RoLLaMA 2 7B, pentru care resursele hardware locale s-au dovedit insuficiente din cauza duratei excesive de inferență, toate experimentele locale au fost rulate pe mașina locală. Rularea RoLLaMA a fost realizată într-un notebook separat pe Google Colaboratory, aspecte detaliate într-o secțiune ulterioară.

Un aspect metodologic relevant privește modelul Aya Expanse 8B: începând cu 4 aprilie 2026, Cohere a retras modelul `c4ai-aya-expanse-8b` din API-ul oficial. Deoarece modelul a fost rulat local prin Ollama, utilizând greutățile open-weight disponibile pe HuggingFace, această decizie nu a afectat experimentele - inferența locală funcționând independent de disponibilitatea API-ului comercial.

Configurația experimentului

Celula de configurație definește cei trei parametri principali ai fiecărui experiment: modelul evaluat (`MODEL`), limba promptului (`LANG` - `ro` sau `en`) și versiunea de prompt (`PROMPT_VERSION` - `v1` până la `v4`). Pe baza acestor parametri este generat automat numele experimentului după convenția {`model`}__{`lang`}__{`version`}, utilizat consistent pentru denumirea fișierelor de output. Fiecare experiment produce un fișier JSON în directorul `outputs/`, după același pattern de nomenclatură - de exemplu `exp_aya_expanse_8b__ro__v1.json`.

Deoarece fiecare combinație model-limbă este rulată în două variante lingvistice - română și engleză - pentru fiecare model și versiune de prompt există câte două fișiere de experiment, rezultând un total planificat de 40 de experimente pentru cele 5 modele principale $\times$ 2 limbi $\times$ 4 versiuni.

La acest total se adaugă două experimente suplimentare. Primul este experimentul cu RoBERT-base (`robert_encoder`), rulat exclusiv în română pe versiunea v1 pentru a documenta performanța unui model encoder monolingv pre-antrenat pe MLM în regim zero-shot. Rezultatele au confirmat că RoBERT-base nu este adecvat pentru zero-shot classification fără fine-tuning - înregistrând o acuratețe de 1.7% pe română - întrucât arhitectura MLM nu dispune de mecanismul de inferență logică necesar clasificării fără exemple de antrenament. Prin urmare, modelul a fost înlocuit în setul principal de experimente cu XLM-RoBERTa XNLI (`roberta_encoder`), un encoder multilingv de dimensiuni mai mari, antrenat explicit pe sarcini de Natural Language Inference în 15 limbi inclusiv română, care permite zero-shot classification prin compararea semantică a textului conversației cu fiecare etichetă posibilă. Al doilea experiment suplimentar este cel adițional cu etichete în română pentru RoLLaMA, descris în secțiunea anterioară. La finalul campaniei de experimentare, directorul `outputs/` conținea astfel 42 de fișiere JSON, acoperind toate combinațiile planificate plus cele două experimente suplimentare.

Inițializarea modelului

Celula de inițializare instanțiază clientul corespunzător modelului selectat și definește funcția `call_model(prompt)` cu o interfață uniformă pentru toate modelele, indiferent de modul de acces. Fiecare model a necesitat o procedură de inițializare specifică, descrisă în cele ce urmează.

OpenAI o3 a fost accesat prin SDK-ul oficial Python al OpenAI (`openai`), care oferă o interfață de nivel înalt peste API-ul REST al platformei. SDK-ul (*Software Development Kit*) este o bibliotecă software care abstractizează detaliile de comunicare HTTP, autentificare și gestionare a erorilor, permițând apelarea modelului printr-un simplu apel de funcție. Autentificarea se realizează prin cheia API citită automat din variabila de mediu `OPENAI_API_KEY`, configurată local în fișierul `.env`.

Gemini 2.5 Flash a fost accesat prin biblioteca `google-genai`, SDK-ul oficial Google pentru modelele Gemini, cu cheia API citită din variabila de mediu `GEMINI_API_KEY`. Apelurile sunt direcționate către Google AI Studio, iar răspunsurile sunt returnate ca obiecte Python cu câmpul `.text` conținând textul generat.

Aya Expanse 8B a fost rulat local prin Ollama - un runtime pentru modele lingvistice open-weight care gestionează descărcarea, cuantizarea și inferența modelelor local, fără conexiune la un API extern. Modelul a fost descărcat o singură dată prin comanda `ollama pull aya-expanse:8b` (5GB) și accesat din Python prin biblioteca `ollama`, care expune o interfață identică cu cea a API-ului OpenAI. Deși Cohere a oferit acces la Aya Expanse și prin API-ul comercial, accesul local prin Ollama a fost preferat din două motive. În primul rând, utilizând greutățile open-weight disponibile pe HuggingFace, rularea locală elimină costurile per token asociate API-ului comercial, relevante în contextul unui număr mare de apeluri generate de experimentele iterative. În al doilea rând, această decizie s-a dovedit înțeleaptă retrospectiv: începând cu 4 aprilie 2026, Cohere a retras modelul `c4ai-aya-expanse-8b` din API-ul oficial, accesul prin API devenind indisponibil - fără a afecta însă experimentele locale, care au continuat să ruleze independent de această decizie.

Tot pentru uz local, **XLM-RoBERTa XNLI** (`joeddav/xlm-roberta-large-xnli`) și **RoBERT-base** (`readerbench/RoBERT-base`) au fost descărcate automat din HuggingFace Hub la prima rulare și stocate în cache-ul local HuggingFace (1.1GB, respectiv 450MB). Ambele modele au fost instanțiate prin pipeline-ul de `zero-shot-classification` din biblioteca `transformers`, rulând pe CPU fără GPU. Funcția `call_model` pentru aceste modele nu trimite promptul Jinja2 complet la model - ci extrage textul conversației din prompt și îl pasează direct pipeline-ului NLI, care calculează scoruri de entailment între text și fiecare etichetă posibilă.

RoLLaMA 2 7B Instruct (`OpenLLM-Ro/RoLlama2-7b-Instruct`) a fost inițializat în Google Colaboratory, descărcat de pe HuggingFace Hub și încărcat cu cuantizare pe 4 biți prin biblioteca `bitsandbytes`, reducând memoria necesară de la 14GB la 4GB - compatibil cu GPU-ul T4 de 15GB al instanței Colab. Tokenizatorul și modelul au fost instanțiate prin `AutoTokenizer` și `AutoModelForCausalLM` din `transformers`, cu `device_map="auto"` pentru plasarea automată pe GPU.

Costurile experimentelor pe modele comerciale - sumar pentru extragerea intenției

Accesarea modelelor comerciale prin API a generat costuri financiare directe, spre deosebire de modelele locale care au rulat fără costuri de inferență. Pentru **OpenAI o3** au fost consumați aproximativ 1,59 milioane de tokens de input în perioada 11–26 aprilie 2026, incluzând toate versiunile de prompt (v1–v4), ambele limbi și rerulările necesare pentru corectarea erorilor de pipeline și validarea prompturilor. Pentru **Gemini 2.5 Flash** și resursele **Google Colaboratory**

utilizate pentru rularea RoLLaMA, costul total acumulat pe contul Google Cloud a fost de aproximativ $4,54 USD, în cadrul unui abonament pay-as-you-go. Aceste valori demonstrează că evaluarea unui pipeline de monitorizare bazat pe LLM-uri comerciale este fezabilă din punct de vedere economic chiar și în context academic, costul total al tuturor experimentelor pe modele comerciale încadrându-se sub $15.

Randarea promptului

Funcția `render_prompt(lang, turns, version)` încarcă templateul Jinja2 corespunzător și injectează dinamic conversația în prompt. Funcția acceptă un parametru opțional `examples` pentru versiunile few-shot (v4) și `ro_labels` pentru experimentul adițional cu etichete în română, returnând în toate cazurile un string gata de trimis modelului. Templateurile sunt încărcate din directorul `prompts/intent_extraction/` prin `FileSystemLoader`, ceea ce permite modificarea prompturilor fără a reporni kernel-ul.

Parsarea răspunsului

Funcția `parse_response(raw)` extrage câmpurile `intent`, `confidence` și `reasoning` din răspunsul brut al modelului. Parserul gestionează variațiile de format întâlnite în practică: JSON înfășurat în blocuri Markdown (```` ```json ````), ghilimele duble consecutive apărute din sufixul de forțare a formatului la RoLLaMA, text înainte sau după obiectul JSON, și JSON incomplet căruia îi lipsește acolada de închidere. În cazul modelelor encoder, funcția primește direct outputul pipeline-ului NLI, deja structurat ca dicționar Python, fără a necesita parsare JSON. Funcția returnează un tuplu de patru valori - `(intent, confidence, reasoning, parse_failed)` - unde `parse_failed` este `True` atât când parsarea eșuează cât și când eticheta returnată nu se regăsește în mulțimea etichetelor valide.

Bucla de rulare și salvarea rezultatelor

Bucla principală iterează peste toate conversațiile din dataset, randează promptul pentru fiecare conversație, apelează modelul, parsează răspunsul și acumulează rezultatele într-o listă de dicționare. Fiecare predicție este stocată cu toate câmpurile necesare evaluării: identificatorul conversației, eticheta din dataset, predicția modelului, nivelul de încredere, raționamentul, latența în milisecunde și un flag de parse failure. La finalul buclei, rezultatele sunt serializate într-un fișier JSON cu structura:

```
{
  "experiment": { "name", "model", "lang", "prompt_version", ... },
  "metrics":    { "accuracy", "macro_f1", "cohen_kappa", ... },
  "predictions": [ { "conversation_id", "dataset_label",
                     "predicted_intent", "confidence",
                     "reasoning", "latency_ms", ... }, ... ]
}
```

Metricile sunt calculate automat din predicții la momentul salvării, eliminând riscul de hardcodare manuală a valorilor. Fișierele sunt salvate în directorul `outputs/` după convenția `exp_{experiment_name}.json`, permițând încărcarea și compararea automată de către modulul de evaluare.

Modele rulate în Google Colab

RoLLaMA 2 7B Instruct a fost rulat într-un notebook separat găzduit pe Google Colaboratory, utilizând un accelerator GPU NVIDIA T4 pus la dispoziție de Google, din cauza limitărilor hardware ale mașinii locale. Modelul a fost încărcat cu cuantizare pe 4 biți prin biblioteca `bitsandbytes`, reducând memoria necesară de la aproximativ 14 GB la circa 4 GB. Datorită specificului RoLLaMA de a nu urma instrucțiunile de format JSON în mod fiabil, a fost adoptat un mecanism de suffix forțat: promptul este completat cu începutul obiectului JSON așteptat, iar modelul continuă generarea de la acel punct. Răspunsul generat este ulterior reconstruit și pasat aceluiași parser utilizat pentru celelalte modele. Fișierele de output au fost salvate în Google Drive și transferate ulterior în directorul local `outputs/` pentru procesare uniformă.

4.4.2. Detecția neconcordanțelor

Promptul de detecție a neconcordanțelor are ca obiectiv identificarea situațiilor în care răspunsurile voicebotului prezintă deficiențe calitative, independent de statusul final al conversației. Spre deosebire de celelalte două taskuri - care clasifică intenția utilizatorului și deznodământul dialogului - acest task evaluează exclusiv comportamentul voicebotului, analizând dacă răspunsurile sale sunt complete, relevante, coerente și aliniate cu informațiile furnizate de utilizator. Sarcina este formulată ca o problemă de detecție binară urmată de clasificare multi-clasă: modelul determină mai întâi dacă există o neconcordanță, iar în caz afirmativ identifică tipul cel mai proeminent dintre cele 5 definite.

Principala dificultate a acestui task constă în delimitarea precisă a granițelor dintre tipuri adiacente. Distincția dintre nealiniat context și halucinatie este cea mai subtilă - ambele reprezintă răspunsuri care ignorează sau deviază de la contextul furnizat de utilizator, însă mecanismul diferă fundamental: nealiniat context presupune ignorarea unei informații furnizate explicit (de exemplu, utilizatorul întreabă despre cardul de credit, iar botul răspunde despre cardul de debit), în timp ce halucinatie presupune inventarea unor fapte care nu au apărut niciodată în conversație (de exemplu, afirmarea unor sume, procente sau condiții neexistente, sau exagerarea în limbajul folosit fără suport factual). Confuzia poate apărea și cu contradictoriu, însă acest tip din urmă presupune ca botul să se contrazică pe sine în replici diferite, fără legătură cu contextul furnizat de utilizator - de exemplu, când afirmă că un proces durează 5 zile într-o replică și 10 zile într-o altă replică. Similar, incomplet și irelevant se disting prin ignorarea completă a cererii utilizatorului în cazul incomplet vs menționarea unui aspect total irelevant vis-a-vis de cererea utilizatorului (de exemplu începe să facă reclamă la un anumit produs). Definirea precisă a acestor granițe și reducerea erorilor sistematice - în special a false positive-urilor pe nealiniat context și contradictoriu și a false negative-urilor pe halucinatie - constituie firul central al evoluției prompturilor de la v1 la v4, ultimul fiind bazat pe exemplificări few-shot.

v1 - Prompt de bază

Promptul de detecție a incongruităților v1 urmează aceeași abordare zero-shot ca și cel de extracție a intențiilor, utilizând formatul Markdown cu headere '##' conform recomandărilor din [30]. Taskul este definit ca o clasificare binară cu tipizare secundară: modelul trebuie să determine dacă există o neconcordanță în răspunsurile voicebotului și, în caz afirmativ, să identifice tipul cel mai proeminent dintre cele cinci categorii: `incomplet`, `irelevant`, `contradictoriu`, `nealiniat_context` și `halucinatie`.

Secțiunea **Tipuri de neconcordanțe** definește fiecare categorie prin mecanismul care

o generează. `incomplet` acoperă răspunsuri care omit o parte din cererea multi-componentă a utilizatorului. `irelevant` identifică răspunsuri care schimbă complet subiectul sau ignoră cererea formulată. `contradictoriu` necesită cel puțin două replici ale botului aflate în conflict direct (de exemplu, afirmarea unor durate diferite pentru același proces). `nealiniat_context` presupune ignorarea unei informații furnizate explicit de utilizator - cum ar fi solicitarea despre cardul de credit când botul răspunde despre cardul de debit. `halucinatie` marchează inventarea unor fapte nefundamentate în conversație - sume, procente, condiții sau formulări exagerate fără suport contextual.

Dat fiind potențialul de confuzie între tipuri adiacente, promptul include o secțiune dedicată **Distincții cheie** care explică granițele: `incomplet` versus `irelevant` prin prezența versus absența oricărui răspuns la cerere; `nealiniat_context` versus `halucinatie` prin ignorarea contextului versus inventarea de fapte noi; `contradictoriu` prin necesitatea a două replici conflictuale ale botului, independent de contextul utilizatorului. O secțiune suplimentară, **Ce NU constituie o neconcordanță**, reduce riscul de false positive-uri prin excluderea explicită a a unor cazuri ce ar putea fi etichetate prost prin natura lor de graniță. Această secțiune clarifică trei surse frecvente de supraclasificare: (1) răspunsuri care satisfac complet cererea utilizatorului dar nu oferă informații suplimentare opționale - riscul fiind confuzia cu `incomplet` pe baza unei așteptări implicite de verbozitate; (2) escaladări sau redirecționări către un agent uman sau alt departament, care sunt comportamente corecte în limitele voicebotului dar ar putea fi percepute ca `irelevant` dacă modelul interpretează transferul ca evitare a răspunsului; și (3) variații stilistice sau de formulare între replici care nu schimbă sensul, susceptibile de clasificare greșită ca `contradictoriu` dacă modelul confundă reformularea cu contrazicerea.

Includerea acestor clarificări chiar din versiunea inițială - spre deosebire de promptul de extracție a intențiilor, unde astfel de distincții nu au fost necesare în v1 - reflectă natura mai ambiguă a task-ului de detecție a incongruităților. Dacă la extracția intențiilor categoriile semantice sunt relativ discrete și mutual exclusive (utilizatorul vrea informații despre card *sau* despre sold *sau* despre rate), la incongruități granițele sunt fundamental mai neclare: ce constituie exact „ignorarea contextului" versus „inventarea de fapte"? Când devine un răspuns suficient de incomplet pentru a fi clasificat ca atare, versus doar concis dar complet? Această ambiguitate structurală amplifică riscul de false positive-uri - un răspuns concis ar putea fi clasificat greșit ca `incomplet`, o redirecționare corectă ca `irelevant`, o formulare ușor diferită drept `contradictoriu`. Secțiunile de distincții și cazuri negative acționează astfel ca „gărzi de siguranță" împotriva supraclasificării, fiind integrate preventiv în arhitectura promptului încă de la prima versiune

Instrucțiunile de execuție structurează taskul ca o evaluare binară suprapusă peste o clasificare multi-clasă: modelul decide mai întâi dacă există vreo neconcordanță (`has_incongruity`: true/false), apoi - doar dacă răspunsul este afirmativ - identifică tipul cel mai proeminent dintre cele cinci categorii. Această arhitectură ierarhică previne clasificarea forțată în situații ambigue unde nu există nicio neconcordanță clară, permițând modelului să răspundă cu `null` pentru `incongruity_type` când conversația este corectă. Restricția către „un singur tip dominant" - chiar când multiple clasificări ar putea fi plauzibile - reduce complexitatea taskului și forțează modelul să prioritizeze neconcordanța cu cel mai mare impact asupra experienței utilizatorului.

Câmpul `reasoning` forțează modelul să explice decizia într-o propoziție concisă, activând un proces de raționament explicit care reduce erorile rapide bazate pe asocieri superficiale [31]. Cerința de a menționa turn-ul specific unde apare neconcordanța ancorează justificarea în segmente concrete ale conversației, prevenind clasificări bazate pe impresii vagi. Câmpul `confidence` (high/medium/low) permite modelului să semnaleze gradul de certitudine - informație utilă pentru identificarea cazurilor care necesită validare umană suplimentară.

Instrucțiunea explicită de a evalua doar răspunsurile voicebotului, nu mesajele utilizatorului, previne o sursă frecventă de confuzie: utilizatorii pot fi contradictorii, pot schimba subiectul sau pot furniza informații incomplete, dar aceste caracteristici nu constituie neconcordanțe ale sistemului. Această restricție nu împiedică modelul să consulte mesajele utilizatorului pentru context - dimpotrivă, detectarea tipurilor `nealiniat_context` și `halucinatie` necesită explicit analiza cererilor utilizatorului pentru a verifica dacă botul a ignorat informații furnizate sau a inventat fapte inexistente în conversație. Distincția crucială este că modelul citește întreaga conversație pentru a înțelege contextul, dar clasifică doar calitatea răspunsurilor botului, nu comportamentul utilizatorului. Formatul de ieșire este impus strict - „Răspunde doar cu un obiect JSON valid. Fără explicații, fără markdown, fără preambul" - pentru a facilita parsarea automată și integrarea în pipeline-ul de evaluare. Ca și în cazul extracției intențiilor, ambele limbaje - română și engleză - sunt testate pentru a evalua impactul idiomului asupra acurateței clasificării, cu ipoteza că limba maternă a conversațiilor (româna) ar putea oferi modelelor bilingve o înțelegere mai nuanțată a contextului cultural și a subtilităților semantice.

4.4.2.1 V2 - Rafinare manuală a criteriilor de clasificare

Evaluarea preliminară a performanței promptului v1 a evidențiat pattern-uri sistematice de eroare concentrate în jurul granițelor dintre tipuri adiacente. Deși analiza detaliată a rezultatelor experimentale va fi prezentată în capitolul de evaluare, aceste remarci sunt necesare aici pentru a contextualiza deciziile de rafinare a promptului. Fals-pozitivele apăreau frecvent pentru `incomplet` (răspunsuri concise dar complete clasificate greșit), `nealiniat_context` (acțiuni corecte interpretate ca ignorare a contextului), și `contradictoriu` (reformulări percepute ca contraziceri). Fals-negativele pentru `halucinatie` sugerau că modelele ezitau să clasifice afirmații implauzibile fără dovezi foarte clare de fabricare. Aceste observații au motivat o rafinare manuală a prompt-ului, concentrată pe operaționalizarea criteriilor de clasificare prin adăugarea unei secțiuni noi: **Precizări suplimentare per tip**.

Analiza manuală a erorilor a fost efectuată într-o secțiune dedicată din notebook, unde fiecare predicție greșită a fost clasificată ca false positive (FP), false negative (FN) sau tip greșit (confuzie între categorii adiacente). Examinarea acestor cazuri a scos în evidență un tipar clar, neacoperit de V1. `nealiniat_context` genera un număr mare de fals-pozitive, în special în scenarii unde utilizatorul solicita informații complet în afara domeniului bancar (vreme, sfaturi personale, recomandări de restaurante) iar botul răspundea corect oferind servicii bancare disponibile - modelul interpreta această redirecționare legitimă ca ignorare a contextului furnizat. Similar, cereri de autentificare sau escaladări corecte către agenți umani erau clasificate greșit ca `nealiniat_context`. `halucinatie` prezenta problema opusă: multiple fals-negative în cazuri unde botul inventa detalii plauzibile dar nefundamentate (sume specifice, termene, proceduri). Confuziile între tipuri adiacente apăreau sistematic: `contradictoriu` clasificat ca `nealiniat_context` când botul își contrazice propriile afirmații despre proceduri; `irelevant` confundat cu `nealiniat_context` când botul promovează produse nesolicitatate.

Noua secțiune **Precizări suplimentare per tip** adresează aceste tipare prin criterii mai stricte și mai operaționale, derivate din analiza erorilor dar formulate suficient de general pentru a evita overfitting-ul pe cazurile specifice examinate din dataset, pentru a preveni erori pe posibil ecazuri noi analizate. Pentru `incomplet`, criteriul devine „clasifică doar dacă o cerere explicită nu este adresată deloc - botul nu o menționează și nu oferă niciun răspuns la ea", eliminând astfel confuzia dintre răspunsuri incomplete și răspunsuri concise dar suficiente. Pentru `nealiniat_context`, criteriul devine „clasifică doar când botul acționează pe o informație diferită față de cea furnizată explicit de utilizator și relevantă pentru acțiunea întreprinsă", cu instrucțiunea explicită „nu clasifica drept nealiniat_context dacă botul a acționat corect pe baza

informațiilor disponibile" - această ultimă frază vizează direct cazurile FP unde botul răspunde corect la cereri în afara domeniului.

Rafinarea tipului `halucinatie` introduce trei criterii concrete - vocabular vădit neprofesionist sau exagerat pentru un sistem bancar automat, termene imposibile sau absurde, taxe și sume fără legătură cu contextul - urmate de instrucțiunea „Nu supraanaliza - dacă există o explicație rezonabilă în contextul bancar, nu este halucinație". Această formulare preventivă combate tendința modelelor de a suspecta halucinație în absența unor dovezi explicite de corectitudine, adresând direct tiparul de fals-negative observat. Pentru `contradictoriu`, criteriul devine „clasifică doar când botul afirmă în mod explicit aceeași informație cu valori diferite", cu precizarea că „exprimări diferite ale aceleiași durate aproximative nu constituie contradicție" - vizând cazurile în care botul spune „câteva zile" într-o replică și „3-5 zile" în alta, și implicit diferențiind de cazurile unde botul se contrazice cu sine însuși (de exemplu, afirmând că resetarea parolei se face prin aplicație mobilă, apoi prin email). Tipul `irelevant` primește o clarificare prin exemplu (promovarea de produse nesolicitatate) și o distincție explicită față de `nealiniat_context`: „la irelevant răspunsul nu are nicio legătură cu cererea, pe când la nealiniat_context răspunsul e similar dar pe o informație greșită" - această distincție adresează confuzia frecventă între cele două categorii observată în clasificările greșite.

Aceste rafinări nu modifică structura fundamentală a promptului v1 - secțiunile de definiții generale, distincții cheie și cazuri negative rămân neschimbate - ci adaugă un strat suplimentar de precizie operațională. Obiectivul este reducerea ambiguității interpretative prin ancorarea deciziilor de clasificare în criterii verificabile și în instrucțiuni explicite de evitare a supraclasificării.

4.4.2.2 Versiunea 3 - Optimizare asistată de LLM

După rafinarea manuală din v2, s-a urmărit îmbunătățirea promptului prin metode de optimizare automată similare celor aplicate la extragerea intențiilor. Procesul a început cu crearea unui meta-prompt - un set de instrucțiuni care descriu cum să fie construit un alt prompt pentru detecția neconcordanțelor. Acest meta-prompt a fost redactat inițial într-o formă rudimentară și apoi rafinat prin cererea către ChatGPT (modelul openai o3, același analizat în lucrare) de a îmbunătăți chiar meta-promptul însuși.

Meta-promptul final structura problema de optimizare în cinci componente: (1) descrierea task-ului (clasificare binară cu tipizare secundară pe 5 clase), (2) contextul datasetului (180 conversații, distribuția claselor, rangul de turnuri), (3) erorile sistematice cunoscute din v2 - fals-negative pentru `halucinatie` din cauza definiției vagi, fals-pozitive pentru `nealiniat_context` prin supraclasificare, fals-pozitive pentru `contradictoriu` prin confundarea replicilor aproximative cu contradicții explicite, (4) clarificări operaționale pentru `incomplet` (aplicare doar când cererea este complet ignorată) și `irelevant` (subiect complet diferit, nu informație greșită), și (5) constângeri stricte - păstrarea exactă a formatului JSON, a numelor de câmpuri (`has_incongruity`, `incongruity_type`, `confidence`, `reasoning`), a etichetelor claselor și a template-ului Jinja2 pentru conversații. Meta-promptul se încheia cu instrucțiunea „Please improve this prompt to reduce the systematic errors described above while maintaining the exact output format and label names", urmată de promptul v2 ca material de optimizat.

Încercarea de utilizare a optimizatorului OpenAI - același mecanism aplicat cu succes la promptul de extragere a intenției - s-a confruntat cu limitări tehnice neprevăzute. Combinația dintre promptul de îmbunătățit (v2 pentru incongruități, deja substanțial) și meta-promptul de optimizare depășea constant limita de context disponibilă, generând erori de overflow. Reducerea lungimii prin condensarea instrucțiunilor nu a produs rezultate satisfăcătoare: optimizatorul propunea modificări minore, nesemnificative sau chiar eronate, lipsind caracterul structural al

rafinărilor necesare - spre deosebire de succesul său la extragerea intențiilor, unde task-ul era mai simplu și promptul mai concis. O încercare ulterioară cu optimizatorul Claude a eșuat din cauze tehnice diferite: generarea promptului optimizat se bloca constant la jumătatea procesului, iar pagina nu mai răspundea.

Soluția adoptată a fost utilizarea directă a capabilităților de generare de text ale OpenAI o3, fără componentă de optimizare iterativă. Meta-promptul creat anterior și reiterat prin Claude Opus 4.6 a fost furnizat împreună cu promptul v2 și instrucțiunea explicită de a produce o versiune v3 care să adreseze erorile sistematice observate. Modelul a produs o restructurare completă a promptului, menținând invariantele semantice ale definițiilor dar modificând radical organizarea, formularea și nivelul de specificitate.

Comparația dintre v2 și v3 arată o schimbare importantă în filozofia promptului. V3 adaugă o secțiune **IMPORTANT** la început care schimbă modul de evaluare: în loc să judece dacă răspunsul botului e realist pentru o bancă reală, modelul trebuie să evalueze strict ce scrie în text față de regulile date. Instrucțiunea explicită - „Nu evalua conversația după reguli bancare reale, proceduri reale de securitate sau presupuneri despre ce ar putea face un voicebot în realitate. Evaluează doar textul conversației și regulile de etichetare de mai jos" - rezolvă o problemă majoră din v2: modelele clasificau greșit răspunsuri corecte ca fiind neconcordanțe pentru că aveau așteptări incorecte despre cum ar trebui să funcționeze o bancă în realitate.

Secțiunea **Reguli generale** consolidează 12 instrucțiuni negative explicite, fiecare adresând un tipar specific: „Nu marca neconcordanță dacă problema este doar o lipsă de detaliu", „Nu marca halucinație doar pentru că botul spune că a actualizat/trimis/confirmat ceva", „Nu marca incomplet doar pentru că botul nu oferă exact valoarea cerută, dacă oferă o cale relevantă de obținere".

Definițiile tipurilor de neconcordanțe adoptă o structură dual-componenta: **Condiții obligatorii** (ce trebuie să fie prezent pentru clasificare) și **NU este X dacă** (cazuri de excludere). Pentru `incomplet`, condițiile obligatorii devin „utilizatorul formulează o cerere clară; botul nu menționează deloc acea cerere; botul nu oferă nici răspuns, nici clarificare, nici redirecționare, nici pas următor relevant", iar excluderile enumeră șase scenarii specifice unde răspunsul este valid deși pare incomplet (autentificare, redirecționare, metodă alternativă). Similar, `nealiniat_context` primește condiția „botul folosește explicit o valoare diferită, alt produs, alt canal, alt card, alt cont" și opt excepții enumerate (nu repetă toate detaliile, cere clarificări, oferă cale alternativă). Structura este mult mai procedurală și verificabilă decât definițiile narative din v2.

O inovație majoră este secțiunea **Reguli speciale anti-erori frecvente**, care listează explicit opt scenarii problematice observate în evaluarea v2: „Cererile non-bancare precum vreme, glume, restaurante [...] pot fi refuzate sau redirecționate spre servicii bancare fără a fi neconcordanță", „Dacă utilizatorul cere un agent [...] iar botul transferă, programează, redirecționează [...] NU este nealiniat_context", „Dacă utilizatorul cere sold, extras, tranzacții [...] iar botul cere autentificare ori recomandă aplicația [...] NU este incomplet". Aceste reguli sunt cazuri-limită extrase direct din conversațiile clasificate greșit în v2, operaționalizate ca excepții explicite.

Promptul v3 introduce și o secțiune **Ordine de decizie** - o ierarhie procedurală în șase pași care dictează succesiunea verificărilor: (1) halucinație (taxe/sume/limbaj neprofesionist fără bază), (2) nealiniat context (valoare diferită explicit), (3) contradictoriu (valori incompatibile pentru aceeași informație), (4) incomplet (ignorare completă fără pas relevant), (5) irelevant (subiect complet diferit), (6) nicio neconcordanță dacă niciuna nu este îndeplinită clar. Această ordonare oferă un algoritm de decizie deterministic care reduce ambiguitatea în cazuri unde multiple tipuri par plauzibile.

Evoluția de la v1 la v3 ilustrează o traiectorie de la descriere către specificare operațională. V1 definește categoriile semantic și introducea distincții între tipuri, dar lăsa spațiu larg de interpretare. V2 adăuga criterii de clasificare mai stricte și precizări per tip, reducând ambiguitatea dar menținând o structură narativă. V3 restructurează complet promptul într-o formă procedurală, cu reguli negative explicite, condiții obligatorii enumerate, excepții detaliate și o ierarhie de decizie - transformând task-ul dintr-o evaluare interpretativă într-un proces de verificare algoritmică. Această evoluție reflectă învățarea din erorile succesive: fals-pozitivele sistematice din v1 au motivat rafinările din v2, iar persistența lor în v2 a declanșat reformularea defensivă din v3.

Deși nivelul ridicat de specificitate al regulilor din v3 - cu enumerări exhaustive de excepții și cazuri-limită - ar putea părea susceptibil de overfitting pe particularitățile datasetului, această îngrijorare este neîntemeiată în contextul task-ului de detecție a neconcordanțelor. Clasificarea răspunsurilor unui voicebot bancar reprezintă un task nișat cu un domeniu strict delimitat: botul operează într-un spațiu constrâns de acțiuni posibile (autentificare, redirecționare, interogare sold, blocare card), un set finit de proceduri bancare, și tipare repetitive de interacțiune. Regulile detaliate din v3 nu reprezintă overfitting pe exemple specifice, ci operaționalizarea limitărilor reale ale domeniului - ce constituie un răspuns valid versus o neconcordanță într-un context bancar automatizat. Liu et al. [29] arată în analiza lor sistematică a metodelor de prompting că task-urile specifice domeniului necesită adaptări ale prompturilor care capturează constrângerile și structura unică a taskului, implicit, pentru taskuri nișate, prompturile specializate nu sunt o slăbiciune ci o necesitate. În producție, un voicebot bancar real va avea comportamente predictibile și consistente; specificitatea regulilor v3 reflectă această realitate operațională, nu particularitățile arbitrare ale unui dataset de antrenament.

4.4.2.3 Versiunea 4 - Introducerea exemplelor

Promptul v4 introduce exemple demonstrative pentru consolidarea capacității de discriminare a modelului între tipurile de neconcordanțe și situațiile valide. Versiunea v4 se construiește pe fundamentul promptului v3, nu pe v1 sau v2, ca urmare a unei evaluări comparative pe toate cele 5 modele evaluate, care a arătat că v3 obține cea mai bună performanță dintre cele trei versiuni (rezultatele detaliate sunt prezentate în capitolul următor, dedicat evaluării modelelor). Performanța superioară a v3 derivă din structura sa procedurală - secțiunile **Reguli generale**, definițiile dual-componentă (**Condiții obligatorii** și **NU este X dacă**) pentru fiecare tip, și **Ordinea de decizie** ierarhică - care operaționalizează granițele conceptuale între categorii. Această arhitectură face v3 compatibilă cu few-shot learning: exemplele demonstrative pot ancora regulile procedurale existente fără a crea redundanță sau conflict, în timp ce v1 (definițional simplu, fără reguli anti-eroare) sau v2 (precizări ad-hoc fără structură sistematică) ar necesita rescrierea extensivă pentru a integra exemple fără inconsistențe. Cu toate acestea, evaluările preliminare au arătat că modelele întâmpină dificultăți în aplicarea consecventă a regulilor v3 pe cazuri-limită - tipologii sistematice de fals-pozitive pe `incomplet` și `nealiniat_context`, și fals-negative pe `halucinatie`. Învățarea cu exemple (few-shot learning) abordează această problemă prin furnizarea de exemple concrete care demonstrează aplicarea corectă a regulilor v3. Promptul V4 (en) este inclus în anexa finală.

Selecția și construcția exemplelor

Procesul de construcție a exemplelor few-shot a urmat o abordare sistematică, ghidată de analiza erorilor experimentelor v1–v3. Am creat celule dedicate de analiză în notebook care: (1) au extras metricile de performanță pe toate experimentele finalizate, identificând configurația

cu cel mai bun F1-Macro; (2) au clasificat greșelile best-performing-ului (openai_o3__en__v3) în false positive-uri, false negative-uri și confuzii de tip; și (3) au căutat conversații din dataset clasificate corect care ar putea servi drept candidate. Analiza a relevat tipologiile sistematice de erori: false negative-uri pe `halucinatie` (5), `incomplet` (4) și `irelevant` (3), false positive-uri pe `nealiniat_context` (3), și confuzii între tipuri adiacente.

Pe baza acestor pattern-uri, am generat cu asistența Claude un set de 7 exemple sintetice - 5 cu neconcordanțe (câte unul pentru fiecare tip) și 2 fără neconcordanță. Am ales deliberat să nu folosim conversații existente din dataset pentru a preveni overfitting-ul: exemplele sintetice capturează mecanismele generale de eroare, nu particularitățile unor conversații specifice. Fiecare exemplu a fost construit urmând trei criterii:

Criteriul 1 - Claritate maximă. Fiecare exemplu demonstrează un singur pattern de eroare fără factori confuzi. Exemplul pentru `halucinatie` include o taxă specifică inventată („25 lei pentru verificare sold") fără alte elemente problematice; exemplul pentru `nealiniat_context` conține doar discrepanța zi-a săptămânii (luni → marți) fără alte abateri.

Criteriul 2 - Reprezentativitate pentru pattern-urile de eroare observate. Exemplele reflectă direct mecanismele de eroare identificate în analiza experimentelor v2 și v3:

- `incomplet`: cerere dublă (telefon + email) cu rezolvare parțială - botul actualizează telefonul dar ignoră complet email-ul
- `irelevant`: schimbare de subiect - utilizator raportează fraudă, botul promovează conturi de economii
- `contradictoriu`: informații incompatibile - termen inițial „3-5 zile" apoi „10 zile" pentru același proces
- `nealiniat_context`: valoare diferită de cea specificată - utilizator cere luni, botul confirmă marți
- `halucinatie`: informație inventată - taxă specifică (25 lei) fără bază în conversație

Criteriul 3 - Două exemple negative. Pentru a preveni tendința modelului de a supraclasifica (detecție excesivă de neconcordanțe - problem-a principală identificată prin cele 3 false positive-uri pe `nealiniat_context`), am inclus două exemple `none`: (1) o verificare de sold standard cu autentificare - demonstrează că cererea de verificare nu e `incomplet`, și (2) un refuz politicos al unei cereri non-bancare cu redirecționare - demonstrează că acest comportament nu e `irelevant` conform regulilor v3.

Exemplele au fost formalizate în `configs/few_shot_examples_incongruities.json` (română) și `configs/few_shot_examples_incongruities_en.json` (engleză) cu structura:

```
{
  "examples": [
    {
      "conversation": [...turns...],
      "has_incongruity": true|false,
      "incongruity_type": "<tip>|null",
      "reasoning": "<justificare>"
    }
  ]
}
```

Promptul v4 încarcă aceste exemple dinamic prin template-ul Jinja2 (`ro_incongruities_v4.jinja`, `en_incongruities_v4.jinja`) și le inserează în secțiunea **Exemple** după **Obiectiv**, înaintea regulilor generale. Fiecare exemplu e prezentat ca:

```
### Exemplul N
```

```
**Conversație:**
USER: <text>
ASSISTANT: <text>

**Clasificare corectă:**
{
  "has_incongruity": true,
  "incongruity_type": "halucinatie",
  "reasoning": "Asistentul inventează o taxă..."
}
```

Această formatare face explicită legătura dintre pattern-ul conversațional și decizia de clasificare, consolidând aplicarea regulilor v3 pe cazuri concrete.

Infrastructura notebook-ului

Experimentele de detecție a neconcordanțelor au fost organizate într-un notebook Jupyter separat (`incongruities_experiments.ipynb`), care reutilizează arhitectura modulară descrisă la extragerea intențiilor — celule dedicate configurării, inițializării modelelor, randării prompturilor, parsării răspunsurilor și salvării rezultatelor. Modelele evaluate, mecanismele de acces (API-uri comerciale, Ollama local, Google Colaboratory, HuggingFace) și procedura de inițializare sunt identice cu cele prezentate în secțiunea anterioară și nu vor fi reluate aici. În cele ce urmează sunt descrise exclusiv elementele care diferă față de notebook-ul de extragere a intențiilor.

Configurația experimentului

Celula de configurare definește aceiași trei parametri — `MODEL`, `LANG` și `PROMPT_VERSION` — iar numele experimentului este generat automat după convenția `{model}_{lang}_{version}`, la fel ca în notebook-ul de intenții. Fișierele de output sunt salvate în directorul `outputs/` cu prefixul `exp_inc_`, ceea ce permite identificarea imediată a task-ului. Fiecare combinație model–limbă–versiune produce un fișier JSON distinct, rezultând un total de 40 de experimente pentru cele 5 modele principale × 2 limbi × 4 versiuni. Spre deosebire de extragerea intențiilor, nu au fost necesare experimente suplimentare cu etichete traduse în limba română, întrucât etichetele de neconcordanțe (`incomplet`, `irelevant`, `contradictoriu`, `nealiniat_context`, `halucinatie`) sunt deja formulate în limba română.

Randarea promptului

Funcția `render_prompt(lang, turns, version)` operează identic cu cea din notebook-ul de intenții, încărcând templateurile Jinja2 din directorul `prompts/incongruities/` prin `FileSystemLoader`. O diferență relevantă față de notebook-ul de extragere a intențiilor este gestionarea exemplelor few-shot pentru versiunea v4. Funcția acceptă un al patrulea parametru opțional, `few_shot_examples`, utilizat exclusiv pentru versiunile few-shot. Exemplele sunt încărcate din fișiere JSON dedicate — `configs/few_shot_examples_incongruities.json` pentru română și `configs/few_shot_examples_incongruities_en.json` pentru engleză — conținând cele 7 exemple sintetice.

Construcția acestor exemple a urmat un proces sistematic, ghidat de analiza erorilor acumulate pe parcursul experimentelor v1–v3. În notebook au fost create celule dedicate de analiză care au extras metricile de performanță pe toate experimentele finalizate, au identificat configurația cu cel mai bun F1-Macro și au clasificat greșelile acesteia — predicțiile false positive, false negative și confuziile de tip. Analiza a relevat tipologii sistematice de erori: fals-negative concentrate pe `halucinatie`, `incomplet` și `irelevant`, fals-pozitive pe `nealiniat_context`, și

confuzii frecvente între tipuri adiacente. Aceste pattern-uri au determinat atât selecția tipurilor de exemple, cât și mecanismele demonstrate de fiecare exemplu.

Numărul de 7 exemple a fost stabilit pe baza a două constrângeri complementare. Pe de o parte, acoperirea completă a spațiului de clasificare impunea cel puțin un exemplu pozitiv pentru fiecare dintre cele 5 tipuri de neconcordanțe, astfel încât modelul să primească o demonstrație concretă a fiecărei granițe conceptuale. Pe de altă parte, analiza erorilor din v1–v3 a evidențiat ca problemă principală supraclasificarea — modelele tindeau să detecteze neconcordanțe acolo unde conversația era corectă — ceea ce a impus includerea a cel puțin două exemple negative care să demonstreze explicit ce *nu* constituie o neconcordanță. Un număr mai mare de exemple ar fi crescut lungimea promptului și implicit costul de inferență și riscul de depășire a ferestrei de context la modelele mai mici, fără un câștig proporțional de informație.

Exemplele au fost generate sintetic, cu asistența unui model Claude, și nu au fost preluate din conversațiile existente ale datasetului. Această alegere a fost deliberată: utilizarea unor conversații din dataset ar fi creat riscul ca modelul să memoreze particularitățile acelor instanțe specifice — formule stilistice, lungimi de dialog, combinații de intenție și status — și să performeze artificial mai bine pe ele, fără a generaliza regulile de clasificare. Exemplele sintetice captează mecanismele generale de eroare, nu particularitățile unor conversații specifice.

Fiecare exemplu a fost construit urmând criteriul de claritate maximă: demonstrarea unei singure tipologii de eroare, fără factori confuzi care ar putea ambiguiza decizia de clasificare. Cele 7 exemple acoperă următoarele cazuri:

a) `incomplet` — cerere dublă (actualizare telefon și email), la care voicebotul rezolvă doar prima componentă și ignoră complet a doua, fără a o menționa sau a oferi vreo alternativă;
b) `irelevant` — utilizatorul raportează o tranzacție suspectă, iar voicebotul schimbă complet subiectul și începe să promoveze conturi de economii, ignorând practic solicitarea inițială;
c) `contradictoriu` — voicebotul afirmă inițial că un proces durează 3–5 zile, iar ulterior, în aceeași conversație, menționează 10 zile pentru același proces, fără a justifica diferența;
d) `nealiniat_context` — utilizatorul solicită programarea unei întâlniri pentru ziua de luni, iar voicebotul confirmă programarea pentru marți, ignorând informația furnizată explicit;
e) `halucinatie` — voicebotul inventează o taxă specifică de 25 de lei pentru verificarea soldului, informație care nu are nicio bază în conversație și este implauzibilă în contextul unui serviciu bancar standard;
f) `none` (exemplu negativ 1) — o verificare de sold standard în care voicebotul solicită autentificarea înainte de a furniza informația, demonstrând că cererea de autentificare nu constituie o neconcordanță de tip `incomplet`;
g) `none` (exemplu negativ 2) — un refuz politicos al unei cereri non-bancare (recomandări de restaurante), în care voicebotul redirecționează utilizatorul către serviciile bancare disponibile, demonstrând că acest comportament nu constituie o neconcordanță de tip `irelevant`.

Cele două exemple negative vizează direct cele mai frecvente surse de fals-pozitive identificate în experimentele anterioare: confuzia dintre autentificarea legitimă și răspunsul incomplet, respectiv confuzia dintre redirecționarea corectă a cererilor non-bancare și răspunsul irelevant. Includerea lor ancorează explicit comportamentele corecte ale voicebotului ca fiind *non-neconcordanțe*, reducând tendința de supraclasificare a modelelor.

Fiecare exemplu este stocat ca un obiect JSON cu patru câmpuri: `conversation` (lista

completă de turnuri, cu rol și text), `has_incongruity` (boolean), `incongruity_type` (tipul identificat sau `null`) și `reasoning` (o propoziție concisă care justifică decizia). Câmpul `reasoning` din exemple servește un rol pedagogic: oferă modelului nu doar demonstrații ale outputului corect, ci și un tipar explicit de raționament pe care să îl reproducă — de exemplu, *"Asistentul inventează o taxă de 25 lei pentru verificare sold, informație care nu apare nicăieri în conversație și nu corespunde practicilor bancare standard."* Structura completă a fișierului este:

```
{
  "examples": [
    {
      "conversation": [
        {"role": "user", "text": "..."},
        {"role": "assistant", "text": "..."}
      ],
      "has_incongruity": true,
      "incongruity_type": "halucinatie",
      "reasoning": "Asistentul inventează o taxă..."
    },
    ...
  ]
}
```

Parsarea răspunsului

Funcția `parse_response(raw)` constituie principala diferență structurală față de notebook-ul de intenții, reflectând natura dual a task-ului — detecție binară urmată de clasificare multi-clasă. Parserul extrage patru câmpuri din răspunsul brut al modelului: `has_incongruity` (boolean), `incongruity_type` (unul dintre cele 5 tipuri sau `null`), `confidence` (`high`, `medium` sau `low`) și `reasoning`.

Logica de parsare gestionează mai multe cazuri specifice absente din parserul de intenții. Când `has_incongruity` este `false`, câmpul `incongruity_type` trebuie să fie `null` — dacă modelul returnează totuși un tip, parserul corectează automat valoarea la `null` și marchează inconsistența într-un log de avertizări. Invers, când `has_incongruity` este `true` dar `incongruity_type` este `null` sau absent, predicția este marcată ca `parse_failed`, întrucât o detecție pozitivă fără tipizare nu poate fi evaluată. Parserul gestionează și variațiile de format întâlnite în practică — aceleași ca la extragerea intențiilor (JSON în blocuri Markdown, ghilimele consecutive, text suplimentar) — plus un caz suplimentar specific: unele modele returnează valoarea boolean drept string (`"true"` / `"false"` în loc de `true` / `false`), ceea ce necesită o conversie explicită.

Funcția returnează un tuplu de patru valori — `(incongruity_type, confidence, reasoning, parse_failed)` — unde `incongruity_type` este `"none"` pentru conversațiile fără neconcordanțe (echivalentul funcțional al unei etichete de clasă, utilizabil direct în calculul metricilor), iar `parse_failed` este `True` atât când parsarea eșuează, cât și când tipul returnat nu se regăsește în mulțimea etichetelor valide.

Metricile de evaluare

Evaluarea performanței la detecția neconcordanțelor operează pe două niveluri, reflectând structura ierarhică a task-ului. La nivel de detecție binară, sunt calculate acuratețea, precizia, recall-ul și F1-scorul pentru decizia `has_incongruity` `true`/`false`, metrici care surprind capacitatea modelului de a distinge conversațiile problematice de cele corecte, independent de identificarea tipului specific. La nivel de clasificare multi-clasă, sunt raportate acuratețea, macro F1, weighted F1, Cohen's Kappa și raportul per clasă (precizie, recall, F1 pentru fiecare

dintre cele 6 etichete — cele 5 tipuri plus `none`). Cohen's Kappa este inclus pentru a evalua concordanța dincolo de nivelul aleatoriu, relevant în special datorită dezechilibrului dintre clasele cu și fără neconcordanțe (80% din conversații nu conțin neconcordanțe).

Această evaluare pe două niveluri — absentă din notebook-ul de intenții, unde task-ul este exclusiv de clasificare multi-clasă — permite diagnosticarea diferențiată a erorilor: un model poate avea detecție binară bună dar clasificare slabă (detectează că ceva nu este în regulă, dar confundă tipurile), sau poate rata complet anumite categorii de neconcordanțe (fals-negative pe `halucinatie`, de exemplu) în ciuda unei precizii ridicate pe tipurile pe care le identifică.

Bucla de rulare și salvarea rezultatelor

Bucla principală urmează același flux ca la extragerea intențiilor — iterează peste conversațiile din dataset, randează promptul, apelează modelul, parsează răspunsul și acumulează rezultatele. Structura fișierului JSON de output este adaptată task-ului:

```
{
  "experiment": { "name", "model", "lang", "prompt_version", ... },
  "metrics": {
    "binary_accuracy", "binary_precision", "binary_recall",
    "binary_f1",
    "accuracy", "macro_f1", "weighted_f1", "cohen_kappa", ...
  },
  "predictions": [
    { "conversation_id", "dataset_label",
      "predicted_has_incongruity", "predicted_type",
      "confidence", "reasoning", "latency_ms", ... }, ...
  ]
}
```

Față de structura de la extragerea intențiilor, blocul `metrics` include atât metricile binare cât și cele multi-clasă, iar fiecare predicție conține două câmpuri de output (`predicted_has_incongruity` și `predicted_type`) în loc de unul singur. Metricile sunt calculate automat din predicții la momentul salvării, iar fișierele sunt salvate în același director `outputs/`, permițând încărcarea uniformă de către modulul de evaluare comun celor trei taskuri.

4.4.3. Clasificarea statusului final al conversației

Promptul de clasificare a statusului final are ca obiectiv determinarea modului în care s-a încheiat interacțiunea dintre utilizator și voicebot, evaluată din perspectiva gradului de rezolvare a solicitării inițiale. Spre deosebire de celelalte două taskuri — care analizează fie intenția cu care utilizatorul a inițiat dialogul, fie calitatea punctuală a răspunsurilor voicebotului — acest task operează la nivel holistic, evaluând rezultatul de ansamblu al conversației ca proces. Sarcina este formulată ca o problemă de clasificare multi-clasă cu 5 etichete: `rezolvata`, `partial_rezolvata`, `redirectionata`, `intrerupta` și `nerezolvata`. Fiecare etichetă corespunde unui deznodământ distinct, definit pe baza corelației dintre obiectivul utilizatorului și acțiunile efective ale voicebotului pe parcursul întregului dialog.

Principala dificultate a acestui task rezidă în distincțiile fine dintre statusuri adiacente. Granița dintre `rezolvata` și `partial_rezolvata` este cea mai subtilă și cea mai frecventă sursă de confuzie: dacă utilizatorul trebuie să efectueze orice acțiune suplimentară după încheierea conversației — să semneze un document, să acceseze o aplicație, să se prezinte fizic la bancă — statusul corect este `partial_rezolvata`, chiar dacă voicebotul a furnizat toate

informațiile relevante și a confirmat inițierea procesului. Această distincție impune modelului să evalueze nu doar ce s-a întâmplat în dialog, ci și ce rămâne de făcut după acesta. O a doua sursă de ambiguitate privește eticheta `redirectionata`, care se aplică exclusiv atunci când redirecționarea către un operator uman este menționată explicit în dialog — fie prin solicitarea directă a utilizatorului, fie prin inițiativa voicebotului. Simpla sugestie de a contacta un consilier sau de a vizita o sucursală nu constituie o redirecționare în sensul acestei etichete. A treia distincție problematică este cea dintre `intrerupta` și `nerezolvata`: prima presupune că dialogul s-a încheiat prematur din cauze externe sau prin decizia utilizatorului (închidere bruscă, "revin mai târziu", erori de conectare), în timp ce a doua indică faptul că voicebotul nu a reușit să ducă acțiunea la final — fie din cauza unei erori, fie pentru că solicitarea depășea capacitățile sale. Diferența este în esență între un dialog care nu a avut șansa de a fi rezolvat și un dialog care a avut șansa, dar a eșuat.

Distribuția claselor în dataset reflectă proporțiile întâlnite în utilizarea reală a unui voicebot bancar: clasa `rezolvata` reprezintă 47% din instanțe, `nerezolvata` acoperă 7%, iar `intrerupta` 11%. Într-un sistem operațional funcțional, majoritatea conversațiilor sunt rezolvate cu succes, întreruperile și eșecurile complete sunt scenarii relativ rare, iar redirecționările și rezolvările parțiale ocupă o zonă intermediară. Datasetul a fost construit deliberat pentru a respecta această realitate, nu pentru a echilibra artificial clasele, tocmai pentru ca evaluarea modelelor să se desfășoare în condiții reprezentative pentru un mediu de producție. Această distribuție naturală introduce însă o provocare la nivel de evaluare: un model care ar prezice `rezolvata` pentru toate conversațiile ar obține o acuratețe aparent ridicată fără a demonstra nicio capacitate reală de discriminare. Din acest motiv, includerea metricii Cohen's Kappa alături de acuratețe și F1 este esențială, întrucât măsoară concordanța dincolo de nivelul aleatoriu și penalizează exact acest tip de comportament.

Taskul impune, de asemenea, un mod de raționament diferit față de celelalte două sarcini. La extragerea intenției, modelul trebuie să identifice obiectivul inițial al utilizatorului — o informație concentrată de obicei în primele turnuri. La detecția neconcordanțelor, modelul evaluează calitatea răspunsurilor individuale ale voicebotului, comparând fiecare replică cu contextul furnizat. La clasificarea statusului final, modelul trebuie să integreze întreaga traiectorie a conversației — de la solicitarea inițială, prin eventualele clarificări și acțiuni intermediare, până la ultima replică — și să judece dacă procesul, în ansamblul său, a atins obiectivul. Această evaluare globală necesită un raționament de tip sumarizare și inferență, nu doar potrivire locală de tipare.

Definirea precisă a granițelor dintre statusuri și reducerea confuziilor sistematice — în special între `rezolvata` și `partial_rezolvata`, dar și între `redirectionata` și celelalte statusuri — constituie firul central al evoluției prompturilor de la v1 la v4. Versiunile succesive introduc criterii din ce în ce mai operaționale pentru fiecare distincție, iar versiunea finală adaugă exemple demonstrative few-shot care ancorează regulile în scenarii concrete, urmând aceeași progresie iterativă aplicată și la celelalte două taskuri.

v1 — Prompt de bază

Prima versiune a promptului de clasificare a statusului final urmează aceeași abordare zero-shot utilizată la celelalte două taskuri, adoptând structura Markdown cu antete `##` conform convențiilor descrise anterior. Rolul atribuit modelului este cel de clasificator al rezultatului conversațiilor pentru un sistem voicebot bancar în limba română — o formulare care ancorează evaluarea în domeniul bancar și stabilește că obiectul clasificării este deznodământul interacțiunii, nu conținutul sau calitatea răspunsurilor individuale. Sarcina este definită explicit: clasificarea statusului final al conversației pe baza modului în care aceasta s-a încheiat și a gradului în care

solicitarea utilizatorului a fost îndeplinită.

Secțiunea `Etichete de status` prezintă cele cinci clase în format bold-paragraph, fiecare însoțită de o descriere și, în cazul etichetei `rezolvata`, de un exemplu concret (blocarea unui card efectuată complet în cadrul conversației). Descrierile urmăresc nu doar definirea fiecărei clase, ci și anticiparea confuziilor frecvente. Definiția etichetei `partial_rezolvata` menționează explicit cele două scenarii principale: pași suplimentari necesari după conversație (semnare document, accesare aplicație, prezentare fizică la bancă) și cereri multiple cu rezolvare incompletă. Definiția etichetei `redirectionata` include o constrângere negativă chiar din versiunea inițială — "nu se încadrează aici simpla sugestie de a vizita sucursala sau de a contacta un operator" — prevenind o sursă frecventă de confuzie cu `partial_rezolvata`. Definiția etichetei `intrerupta` enumeră cauzele tipice (închidere bruscă, "revin mai târziu", răzgândire, erori de conectare), delimitând explicit scenariile de întrerupere față de cele de eșec.

O decizie de proiectare relevantă este includerea unei secțiuni `Distincții cheie` chiar din versiunea inițială, similară celei utilizate la detecția neconcordanțelor și absentă din promptul de extragere a intențiilor. Această secțiune formulează explicit trei granițe critice. Prima — `rezolvata` versus `partial_rezolvata` — introduce regula operațională centrală a taskului: "dacă utilizatorul trebuie să facă ORICE acțiune după încheierea conversației, statusul este `partial_rezolvata`". Cuvântul "ORICE", scris cu majuscule în prompt, subliniază pragul minim al distincției: chiar și o singură acțiune suplimentară, oricât de minoră, determină clasificarea ca rezolvare parțială. A doua distincție — `nerezolvata` versus `intrerupta` — sintetizează diferența între un dialog în care voicebotul a încercat și a eșuat și un dialog care s-a încheiat înainte ca voicebotul să aibă șansa să încerce. A treia distincție — `redirectionata` versus `partial_rezolvata` — reiterează cerința de menționare explicită a transferului în dialog, nu doar sugestia unei vizite.

Includerea acestor distincții încă de la prima versiune reflectă experiența acumulată la celelalte două taskuri. La extragerea intențiilor, lipsa unor clarificări explicite în v1 a generat confuzii sistematice care au necesitat intervenții în v2 și v3. La detecția neconcordanțelor, secțiunea de distincții a fost introdusă preventiv din v1, reducând anumite tipuri de erori de la bun început. Promptul de status final beneficiază de ambele lecții: distincțiile sunt incluse de la bun început, iar formularea lor este mai operațională decât cea din v1 pentru neconcordanțe, utilizând criterii verificabile ("ORICE acțiune", "menționat explicit") în loc de descrieri narative.

Instrucțiunile din v1 orientează modelul către ultimele replici ale conversației — "citește cu atenție ultimele replici — ele conțin de obicei semnalul de rezoluție" — o directivă specifică acestui task și absentă din celelalte două. La extragerea intențiilor, clasificarea este ancorată în primele turnuri ale dialogului; la detecția neconcordanțelor, evaluarea vizează orice replică a voicebotului. La clasificarea statusului final, informația decisivă se concentrează de regulă în zona de încheiere a conversației: confirmarea efectuării unei acțiuni, anunțarea transferului către un operator, sau întreruperea bruscă a dialogului. Această directivă nu exclude analiza întregii conversații — modelul trebuie să înțeleagă solicitarea inițială pentru a evalua dacă a fost rezolvată — dar ghidează atenția către segmentul cu cea mai mare densitate informațională pentru decizia de clasificare. A doua instrucțiune — "bazează clasificarea pe modul în care conversația s-a încheiat, nu pe intenția inițială a utilizatorului" — previne o confuzie potențială între taskuri: modelul nu trebuie să clasifice ce a vrut utilizatorul, ci ce s-a întâmplat efectiv.

Conversia conversației este realizată prin același mecanism de injectare Jinja2 utilizat la celelalte două taskuri, cu filtrul `| upper` aplicat rolurilor participanților. Formatul de ieșire solicitat este un obiect JSON cu trei câmpuri — `final_status`, `confidence` și `reasoning` — identic structural cu outputul celorlalte taskuri, diferind doar prin denumirea câmpului principal (`final_status` în loc de `intent` sau `has_incongruity`/`incongruity_type`). Această

uniformitate a formatului de ieșire între cele trei taskuri facilitează reutilizarea infrastructurii de parsare și evaluare din notebook, diferențele fiind limitate la validarea etichetelor acceptate și la logica specifică fiecărui parser.

Structura generală a promptului v1 este:

- Introducere — rol și sarcină (formulate într-un bloc comun)
- Etichete de status — format bold-paragraph cu descrieri și constrângeri negative
- Distincții cheie — trei granițe critice cu criterii operaționale
- Conversație — bloc dinamic Jinja2 cu roluri în majuscule
- Instrucțiuni — 3 reguli orientate către deznodământul dialogului
- Format de ieșire — obiect JSON

v2 — Rafinare manuală

Trecerea de la v1 la v2 a fost motivată de analiza erorilor sistematice din experimentele inițiale. După rularea experimentelor v1 pe toate modelele, a fost executată în notebook o celulă dedicată de analiză a erorilor — similară celei descrise în secțiunea anterioară — care a extras predicțiile greșite grupate după tipul de confuzie, a calculat frecvența fiecărei perechi (etichetă corectă → etichetă prezisă) și a afișat pentru fiecare eroare identificatorul conversației, eticheta reală, eticheta prezisă și reasoning-ul modelului. Pe baza acestui output, conversațiile care generau erori au fost recitite manual una câte una, urmărind un singur obiectiv: identificarea unei reguli constante care să explice de ce modelul greșea în mod repetat pe același tip de caz. Acolo unde același pattern apărea în mai multe conversații cu modele diferite, regula a fost considerată robustă și a fost integrată în promptul v2. Analiza a evidențiat trei pattern-uri principale de confuzie, concentrate în jurul granițelor dintre statusurile adiacente.

Prima și cea mai frecventă sursă de erori a fost clasificarea greșită a conversațiilor în care voicebotul finalizase acțiunea solicitată, dar utilizatorul urma să aștepte un rezultat ulterior — o investigație pentru o tranzacție suspectă, livrarea unui card nou, procesarea unei cereri. Definiția `rezolvata` din v1 specifica că nu trebuie să fie necesari "pași suplimentari din partea utilizatorului", formulare care lăsa loc de interpretare: așteptarea unui rezultat putea fi percepută ca un pas implicit, determinând modelele să clasifice aceste conversații drept `partial_rezolvata`. În realitate, dacă voicebotul a executat tot ce putea face în cadrul conversației, statusul corect este `rezolvata` indiferent de durata de procesare ulterioară.

A doua sursă de confuzie privea statusul `redirectionata` în conversațiile cu intenția `schedule_advisor_meeting`. Întrucât programarea unei întâlniri cu un advisor reprezintă prin natura ei o redirecționare către un specialist, aceste conversații trebuiau clasificate `redirectionata` chiar dacă programarea fusese confirmată cu succes — nu `rezolvata`, deși obiectivul utilizatorului fusese îndeplinit 100%. Totodată, definiția `redirectionata` din v1 era restrictivă, acoperind doar redirecționările explicite menționate verbal în dialog, și nu includea scenariul în care voicebotul propune și confirmă o programare ca soluție pentru o cerere complexă.

A treia sursă de erori consta în confuzia dintre `intrerupta` și `partial_rezolvata` în conversațiile în care voicebotul furnizase informații dar utilizatorul nu confirmase și nu continuase dialogul. Lipsa unui semnal explicit de confirmare sau de închidere tindea să fie interpretată ca rezolvare parțială, în loc de întrerupere.

Modificările din v2 vizează direct aceste trei pattern-uri. Definiția `rezolvata` a fost rescrisă pentru a ancora criteriul de clasificare în acțiunile voicebotului, nu în pașii ulteriori ai utilizatorului: "voicebotul a finalizat tot ce putea face în cadrul conversației pentru cererea utilizatorului", urmată de o enumerare a acțiunilor care se califică (înregistrarea unui raport, blocarea unui card, trimiterea unui extras, programarea unei întâlniri). O distincție operațională

explicită a fost introdusă în definiția `partial_rezolvata`: "a aștepta nu este a acționa" — dacă utilizatorul doar așteaptă un rezultat (livrare card, rezultat investigație, confirmare SMS), statusul este `rezolvata`; dacă utilizatorul trebuie să facă ceva (să meargă la sucursală, să completeze documente, să sune din nou), statusul este `partial_rezolvata`. Această distincție apare atât în corpul definiției, cât și în secțiunea `Distincții cheie`, unde înlocuiește regula vagă din v1 bazată pe "ORICE acțiune".

Definiția `redirectionata` a fost extinsă pentru a acoperi explicit toate scenariile de programare cu un advisor, consultant sau specialist, indiferent de cine a inițiat programarea. A fost adăugată o regulă de prioritate — "dacă conversația include o programare cu un advisor, statusul este `redirectionata`, chiar dacă botul a oferit și informații suplimentare" — și o distincție nouă față de `rezolvata`, care nu exista în v1: chiar dacă intenția inițială era programarea unei întâlniri și a fost confirmată cu succes, statusul rămâne `redirectionata`.

Definiția `intrerupta` a fost completată cu semnale tipice de întrerupere absente din v1: utilizatorul nu mai răspunde după o întrebare a botului, ultimul mesaj aparține botului, utilizatorul spune că revine mai târziu. Secțiunea `Distincții cheie` a primit o graniță nouă — `intrerupta` versus `partial_rezolvata` — care clarifică că lipsa confirmării din partea utilizatorului indică întrerupere, nu rezolvare parțială.

Instrucțiunile au fost extinse de la trei la cinci reguli, adăugând două directive de verificare secvențială: "verifică dacă există o programare cu un advisor — dacă da, statusul este `redirectionata`" și "verifică dacă utilizatorul a confirmat și a încheiat conversația sau dacă a plecat fără confirmare". Aceste directive transformă instrucțiunile dintr-o listă de principii generale într-un protocol de decizie cu pași expliciți, reducând ambiguitatea în cazurile limită.

Structura de ansamblu a promptului — antete `##`, format bold-paragraph pentru etichete, injectare Jinja2, format JSON de ieșire — rămâne identică cu v1.

v3 — Optimizare asistată

A treia versiune a promptului de clasificare a statusului final a fost obținută prin intermediul optimizatorului dedicat al OpenAI, același instrument utilizat la extragerea intențiilor. Procesul a debutat cu construirea unui meta-prompt care descria cele două tipologii de confuzie sistematică identificate în analiza erorilor din experimentele v2: confuzia dintre `partial_rezolvata` și `redirectionata` și confuzia dintre `intrerupta` și `nerezolvata`.

Prima tentativă de utilizare a optimizatorului a eșuat din cauza depășirii ferestrei de context. Meta-promptul inițial era formulat detaliat, enumerând pentru fiecare caz ambiguu regulile de prioritate, exemplele de scenarii și constrângerile de format, ceea ce împreună cu promptul v2 inclus ca material de îmbunătățit a depășit limita de tokeni acceptată. Soluția a fost condensarea meta-promptului la esențial: pentru fiecare dintre cele două confuzii, o regulă de prioritate clară formulată în engleză, însoțită de criteriul de departajare. Meta-promptul condensat a specificat:

a) `redirectionata` versus `partial_rezolvata`: dacă botul a programat, confirmat sau inițiat concret o întâlnire sau un apel cu un angajat al băncii, eticheta este `redirectionata`, indiferent dacă utilizatorul mai trebuie să participe; `partial_rezolvata` se aplică doar când programarea rămâne în sarcina utilizatorului după conversație.

b) `intrerupta` versus `nerezolvata`: dacă conversația se oprește din cauza lipsei de răspuns a utilizatorului sau a unui mesaj final al botului fără continuare, eticheta este `intrerupta`; `nerezolvata` se rezervă exclusiv cazurilor în care botul declară explicit că nu poate ajuta sau eșuează clar.

Meta-promptul condensat a impus și constrângerile invariante: păstrarea celor 5 etichete

existente, a schemei JSON de output și a template-ului Jinja2 pentru injectarea conversației.

Similar procesului aplicat la detecția neconcordanțelor, a fost realizat și un experiment alternativ în care promptul v2 a fost îmbunătățit direct prin ChatGPT, fără componenta de optimizare iterativă a optimizatorului OpenAI. Compararea rezultatelor celor două abordări a arătat că promptul produs de optimizator obținea metrici mai bune pe ansamblul modelelor evaluate, confirmând valoarea procesului de optimizare față de generarea directă de text pentru această sarcină.

Modificările introduse de optimizator față de v2 sunt mai puțin structurale decât cele observate la detecția neconcordanțelor — structura de ansamblu a promptului, secțiunile și ordinea lor rămân aceleași — și se concentrează pe adăugarea unor reguli operaționale explicite la finalul definițiilor etichetelor cele mai problematice. Promptul primește prefixul `Developer:` la începutul rolului, convenție prezentă și la v3 pentru extragerea intențiilor.

Definiția `partial_rezolvata` primește un paragraf suplimentar care operaționalizează granița față de `redirectionata`: "Folosește `partial_rezolvata` doar când utilizatorul trebuie să programeze, să sune, să confirme, să completeze formulare sau să meargă la sucursală fără ca o programare sau inițiere concretă a unei întâlniri sau a unui apel cu un reprezentant uman să fi fost efectiv realizată în conversație." Această formulare introduce criteriul decisiv — dacă programarea a fost realizată concret în dialog sau nu — și elimină ambiguitatea din v2 legată de scenariile în care botul facilitase parțial procesul fără a-l finaliza.

Definiția `nerezolvata` primește un paragraf de delimitare negativă față de `intrerupta`: "Folosește `nerezolvata` doar când botul spune explicit că nu poate ajuta, că solicitarea este în afara capacităților sau domeniului său, oferă răspunsuri irelevante sau eșuează clar după ce încearcă să rezolve cererea." Formularea "eșuează clar după ce încearcă" este esențială — ea exclude cazurile în care dialogul se oprește înainte ca botul să aibă șansa să încerce.

Definiția `redirectionata` primește un paragraf de confirmare pozitivă a criteriului de clasificare: "Dacă botul a programat deja, a confirmat sau a inițiat concret o întâlnire sau un apel cu un advisor, consultant, specialist, operator sau alt angajat al băncii, clasifică drept `redirectionata`. Faptul că utilizatorul mai trebuie să participe la întâlnire sau apel nu face cazul `partial_rezolvata`." Ultima propoziție adresează direct tiparul de eroare observat în v2, în care participarea viitoare a utilizatorului la o întâlnire deja programată era interpretată ca acțiune suplimentară necesară și declanșa o clasificare greșită ca rezolvare parțială.

Definiția `intrerupta` primește o regulă de decizie rezumativă: "Dacă conversația se oprește pentru că utilizatorul nu răspunde, nu oferă informațiile cerute, nu confirmă, sau ultimul mesaj relevant este o întrebare sau cerere din partea botului, clasifică drept `intrerupta`." Această regulă ancorează decizia în structura dialogului — cine are ultimul cuvânt și dacă acela este un mesaj de finalizare sau o cerere nerezolvată — eliminând necesitatea de a interpreta intențiile implicite ale utilizatorului.

Secțiunea `Distincții cheie` este extinsă cu o graniță suplimentară față de v2: `intrerupta` versus `nerezolvata`, cu regulile de prioritate formulate simetric față de celelalte distincții. Granița `redirectionata` versus `partial_rezolvata` este rescrisă cu criteriul operațional central din noile definiții, înlocuind formularea mai generică din v2.

v4 — Few-shot

A patra versiune a promptului de clasificare a statusului final introduce exemple demonstrative în corpul promptului, urmând aceeași progresie aplicată la celelalte două taskuri: fundamentul structural rămâne v3, iar exemplele few-shot ancorează regulile procedurale existente în scenarii concrete. Decizia de a construi v4 pe baza v3 - nu pe v1 sau v2 - derivă

din evaluarea comparativă a rezultatelor versiunilor, care a arătat că v3 obține cea mai bună performanță dintre cele trei versiuni zero-shot (rezultatele detaliate sunt prezentate în capitolul următor). Promptul V4 (RO) este inclus în anexa finală.

Selecția și construcția exemplelor.

Procesul de construcție a exemplelor few-shot a urmat aceeași metodologie aplicată la detecția neconcordanțelor: celule dedicate de analiză în notebook au extras predicțiile greșite ale configurației cu cel mai bun Macro-F1, le-au clasificat în fals-pozitive, fals-negative și confuzii de tip, iar pe baza tipologiilor identificate au fost construite exemple sintetice care demonstrează aplicarea corectă a regulilor v3.

Analiza erorilor din experimentele v1-v3 a evidențiat trei tipologii de confuzie, în concordanță cu cele identificate în etapele de rafinare a prompturilor:

- confuzia dintre `rezolvata` și `partial_rezolvata` - în special în conversațiile în care voicebotul a finalizat acțiunea dar utilizatorul urmează să aștepte un rezultat (livrare card, rezultat investigație), situații în care modelele clasificau greșit ca rezolvare parțială;
- confuzia dintre `redirectionata` și `partial_rezolvata` - în conversațiile în care voicebotul a programat concret o întâlnire cu un agent uman, dar modelele interpretau participarea viitoare a utilizatorului ca acțiune suplimentară necesară;
- confuzia dintre `intrerupta` și `nerezolvata` - în conversațiile care se opresc brusc fără confirmare din partea utilizatorului, unde modelele ezitau între cele două etichete.

Pe baza acestor tipologii, au fost generate cu asistența unui model Claude (ales intenționat diferit de modelele puse sub evaluare) un set de 7 exemple sintetice — câte unul pentru fiecare dintre cele 5 clase, plus 2 exemple suplimentare pentru cazurile-limită cele mai frecvente. Ca și la detecția neconcordanțelor, exemplele au fost construite sintetic, nu preluate din dataset, pentru a preveni supraantrenarea pe particularitățile conversațiilor existente. Fiecare exemplu a fost construit urmând criteriul de claritate maximă: demonstrarea unui singur pattern de clasificare, fără factori confuzi.

Cele 7 exemple acoperă următoarele cazuri:

a) `rezolvata` - voicebotul blochează cardul utilizatorului, confirmă acțiunea și oferă informații despre pașii următori automatizați (emitere card nou prin poștă). Demonstrează că așteptarea unui rezultat nu constituie acțiune suplimentară din partea utilizatorului - rezolvă confuzia cu eticheta `partial_rezolvata`

b) `partial_rezolvata` - utilizatorul solicită deschiderea unui cont, voicebotul oferă toate informațiile necesare dar menționează explicit că utilizatorul trebuie să se prezinte fizic la sucursală cu documentele, fără a-i programa o vizită în cadrul conversației. Demonstrează distincția centrală: dacă utilizatorul trebuie să *facă* ceva, statusul este parțial rezolvat;

c) `redirectionata` - utilizatorul solicită informații despre un credit, voicebotul oferă detalii generale și programează concret o întâlnire cu un consultant bancar pentru marți la ora 14. Demonstrează că programarea confirmată a unei întâlniri constituie redirecționare, indiferent de informațiile suplimentare furnizate;

d) `intrerupta` - utilizatorul solicită deblocarea cardului, voicebotul cere verificarea identității, utilizatorul răspunde că revine mai târziu și conversația se oprește. Demonstrează că lipsa confirmării și abandonul explicit constituie întrerupere, nu nerezolvare;

e) `nerezolvata` - utilizatorul solicită anularea unei tranzacții deja procesate, voicebotul explică că acțiunea nu este posibilă. Demonstrează că eșecul explicit al voicebotului - nu lipsa de răspuns, ci incapacitatea declarată - constituie nerezolvare;

f) `rezolvata` (exemplu suplimentar) - voicebotul înregistrează un raport de tranzacție suspectă, confirmă primirea și comunică termenul de investigație. Demonstrează că finalizarea acțiunii de către voicebot face conversația rezolvată, chiar dacă investigația în sine durează zile;

g) `partial_rezolvata` (exemplu suplimentar) - utilizatorul are două cereri (actualizare adresă și verificare sold), voicebotul rezolvă prima dar cere autentificare suplimentară pentru a doua, iar utilizatorul confirmă doar prima. Demonstrează că rezolvarea incompletă a cererilor multiple constituie rezolvare parțială.

Distribuția exemplelor reflectă prioritatea reducerii confuziilor cele mai frecvente: cele două exemple suplimentare vizează tocmai granițele `rezolvata-partial_rezolvata` (cazul așteptării pasive vs. acțiunii active) și `partial_rezolvata-rezolvata` (cereri multiple cu rezolvare incompletă), care au generat cele mai multe erori în versiunile anterioare.

Fiecare exemplu este stocat ca un obiect JSON cu patru câmpuri: `conversation` (lista completă de turnuri), `final_status` (eticheta corectă), `confidence` și `reasoning` (o propoziție concisă care justifică decizia). Câmpul `reasoning` oferă modelului un tipar explicit de raționament — de exemplu, *"Voicebotul a confirmat blocarea cardului și a inițiat emiterea unuia nou. Utilizatorul nu trebuie să facă nicio acțiune suplimentară - așteptarea livrării nu constituie o acțiune."* Exemplele sunt stocate în fișierele `configs/few_shot_examples_final_status.json` și sunt in limba română

Promptul v4 încarcă aceste exemple dinamic prin template-ul Jinja2 (`ro_final_status_v4.jinja`, `en_final_status_v4.jinja`) și le inserează în secțiunea **Exemple** după definițiile etichetelor, înaintea secțiunii de instrucțiuni. Fiecare exemplu este prezentat ca:

```
### Exemplul N
**Conversație:**
USER: <text>
ASSISTANT: <text>
**Clasificare corectă:**
{
  "final_status": "rezolvata",
  "confidence": "high",
  "reasoning": "Voicebotul a confirmat blocarea..."
}
```

Structura, instrucțiunile, secțiunea de distincții cheie și formatul de ieșire sunt identice cu v3 - singura diferență constând în adăugarea blocului de exemple. Această consistență permite izolarea contribuției exemplelor few-shot față de performanța bazată exclusiv pe instrucțiunile v3.

Un caz particular l-a reprezentat modelul RoLLaMA 2 7B, rulat în Google Colaboratory, având o cuantizare pe 4 biți. Fereastra de context limitată a modelului, combinată cu lungimea promptului v3 (deja substanțială datorită regulilor procedurale, definițiilor dual-componentă și secțiunii de distincții cheie) și cu lungimea conversațiilor din dataset, a făcut ca promptul v4 cu toate cele 7 exemple few-shot să depășească frecvent capacitatea de procesare a modelului, generând eșecuri de inferență. Pentru a permite rularea experimentului, numărul de exemple a fost redus la 4, păstrând câte un exemplu reprezentativ pentru fiecare dintre cele mai problematice granițe de clasificare și eliminând cele trei exemple care adresau confuzii mai puțin frecvente.

Această reducere reprezintă un compromis necesar între acoperirea spațiului de clasificare și fezabilitatea inferenței pe hardware cu resurse limitate - un aspect practic relevant pentru evaluarea modelelor locale de talie medie în regim few-shot, unde lungimea promptului devine ea însăși o constrângere experimentală.

Infrastructura notebook-ului

Experimentele de clasificare a statusului final au fost organizate într-un notebook Jupyter separat (`final_status_experiments.ipynb`), care reutilizează arhitectura modulară descrisă la extragerea intențiilor - celule dedicate configurării, inițializării modelelor, randării prompturilor, parsării răspunsurilor și salvării rezultatelor. Modelele evaluate, mecanismele de acces (API-uri comerciale, Ollama local, Google Colaboratory, HuggingFace) și procedura de inițializare sunt identice cu cele prezentate în secțiunile anterioare și nu vor fi reluate aici. În cele ce urmează sunt descrise exclusiv elementele care diferă față de notebook-urile anterioare.

Configurația experimentului.

Celula de configurare definește aceiași trei parametri - `MODEL`, `LANG` și `PROMPT_VERSION` - iar numele experimentului este generat automat după convenția `{model}__{lang}__{version}`, la fel ca în celelalte notebook-uri. Fișierele de output sunt salvate în directorul `outputs_final_status/` cu prefixul `exp_fst_`, ceea ce permite identificarea imediată a taskului și diferențierea față de fișierele de intent (`exp_`) și neconcordanțe (`exp_inc_`). Fiecare combinație model–limbă–versiune produce un fișier JSON distinct, rezultând un total de 40 de experimente pentru cele 5 modele principale $\times$ 2 limbi $\times$ 4 versiuni.

Randarea promptului.

Funcția `render_prompt(lang, turns, version)` operează identic cu cele din notebook-urile anterioare, încărcând templateurile Jinja2 din directorul `prompts/final_status/` prin `FileSystemLoader`. Ca și la detecția neconcordanțelor, funcția acceptă un al patrulea parametru opțional, `few_shot_examples`, utilizat exclusiv pentru versiunea v4. Exemplele sunt încărcate din fișierele JSON dedicate - `configs/few_shot_examples_final_status.json`.

Parsarea răspunsului.

Funcția `parse_response(raw)` extrage câmpurile `final_status`, `confidence` și `reasoning` din răspunsul brut al modelului. Parserul gestionează aceleași variații de format întâlnite la celelalte taskuri - JSON în blocuri Markdown, ghilimele consecutive, text suplimentar - plus o normalizare specifică acestui task: etichetele cu diacritice sau variații de formulare (de exemplu, "rezolvată" în loc de "rezolvata", "partial_rezolvată" în loc de "partial_rezolvata", "întreruptă" sau "intreruptă") sunt mapate automat la formele canonice definite în fișierul de definiții. Funcția returnează un tuplu de patru valori - `(status, confidence, reasoning, parse_failed)` - unde `parse_failed` este `True` atât când parsarea eșuează, cât și când eticheta returnată nu se regăsește în mulțimea etichetelor valide după normalizare.

Bucla de rulare și salvarea rezultatelor.

Bucla principală urmează același flux ca la celelalte taskuri - iterează peste conversațiile din dataset, randează promptul, apelează modelul, parsează răspunsul și acumulează rezultatele. Structura fișierului JSON de output este adaptată taskului:

```
{
  "experiment": { "name", "model", "lang",
                  "prompt_version", ... },
  "metrics": { "accuracy", "macro_f1",
               "weighted_f1", "cohen_kappa", ... },
  "predictions": [
    { "conversation_id", "dataset_status",
      "predicted_status", "confidence",
      "reasoning", "latency_ms",
      "parse_failed", ... }, ...
  ]
}
```

Câmpul principal al fiecărei predicții este `predicted_status` (în loc de `predicted_intent` sau `predicted_has_incongruity / predicted_type`), reflectând natura taskului. Metricile sunt calculate automat din predicții la momentul salvării, iar fișierele sunt salvate în directorul `outputs_final_status/`, permițând încărcarea uniformă de către modulul de evaluare comun celor trei taskuri.

Verificarea și validarea experimentelor

Fiecare notebook include, pe lângă celulele de rulare, un set de celule de verificare executate la două niveluri. La nivel de experiment individual, celula [4] testează promptul pe o singură conversație înainte de rularea completă, iar celula [6] produce imediat după rulare un raport cu metricile sumare, raportul de clasificare per clasă și tabelul celor mai frecvente confuzii. La nivel batch de experimente, celula [8] încarcă toate fișierele de output și generează un tabel comparativ al tuturor experimentelor ordonate după Macro-F1, iar celulele de agregare și analiză a erorilor (celulele [9] și următoarele) extrag predicțiile greșite per versiune, le grupează după tipul de confuzie și afișează reasoning-ul modelului pentru fiecare caz - output care a ghidat direct rafinarea prompturilor și construcția exemplelor few-shot.

Acest mecanism de verificare a condus la identificarea și corectarea mai multor anomalii. Analiza rezultatelor pentru OpenAI o3 a evidențiat o rată de parse failure ridicată pe o configurație unde experimentele anterioare aveau zero eșecuri; investigarea a arătat că depășirea bugetului API generase erori înregistrate ca eșecuri de parsare, iar experimentul a fost rerulat după reîncărcarea bugetului. Pe taskul de detecție a neconcordanțelor, tabelul comparativ din celula [8] a arătat că v4 obținea rezultate mai slabe decât v2 la toate modelele — semnal clar de anomalie, nu de limitare a exemplelor few-shot. Verificarea codului a confirmat că funcția `render_prompt()` era apelată fără argumentul `few_shot_examples`, iar exemplele nu erau randate în prompt. Fără aceste celule de verificare, ambele probleme ar fi fost interpretate greșit ca rezultate experimentale.

4.5. Concluzii

Capitolul de față a descris procesul complet de proiectare, rafinare și experimentare a prompturilor pentru cele trei taskuri ale sistemului de monitorizare: extragerea intenției utilizatorului, clasificarea statusului final al conversației și detecția neconcordanțelor din răspunsurile voicebotului.

Toate cele trei taskuri au urmat același protocol de experimentare - patru versiuni de prompt, două limbi, aceeași arhitectură de notebook - dar au evidențiat provocări distincte.

Extragerea intenției s-a dovedit cel mai accesibil task, cu clase relativ discrete și mutual exclusive, unde rafinările au vizat în principal gestionarea cazurilor mixte (fallback). Clasificarea statusului final a impus un raționament holistic asupra întregii conversații, cu granițe subtile între statusuri adiacente - în special distincția `rezolvata` / `partial_rezolvata`, unde criteriul operațional "a aștepta nu este a acționa" s-a dovedit decisiv. Detecția neconcordanțelor a fost cel mai dificil task, cu granițe fundamental ambigue între tipuri adiacente (nealiniat context versus halucinație, incomplet versus irelevant), necesitând cele mai elaborate structuri de prompt - reguli negative explicite, condiții obligatorii enumerate, ierarhie de decizie și exemple sintetice calibrate pe tipologiile de erori.

Progresul de la v1 la v4 ilustrează o traiectorie consecventă de la descriere către specificare operațională. Versiunile inițiale defineau categoriile prin descrieri narative; versiunile intermediare au introdus criterii verificabile și reguli anti-eroare derivate din analiza predicțiilor greșite; versiunile finale au ancorat regulile în exemple concrete prin paradigma few-shot. Această progresie nu a fost planificată a priori, ci a rezultat organic din analiza iterativă a erorilor - fiecare versiune a rezolvat probleme identificate în versiunea anterioară, iar unele modificări au generat erori noi care au motivat iterația următoare.

Un element transversal al capitolului este separarea consecventă între infrastructura de experimentare și conținutul prompturilor. Sistemul de templating Jinja2, definițiile etichetelor în fișiere JSON dedicate, convențiile de nomenclatură pentru fișierele de output și parsarea adaptivă a răspunsurilor au permis rularea a peste 120 de experimente - 42 pentru extragerea intenției, 40 pentru statusul final și 40 pentru neconcordanțe - cu o infrastructură reutilizabilă și rezultate comparabile între taskuri. Această infrastructură constituie o contribuție practică a lucrării, fiind disponibilă integral în repository-ul proiectului.

Evaluarea comparativă a rezultatelor obținute de fiecare model pe fiecare combinație de limbă și versiune de prompt face obiectul capitolului următor, unde sunt prezentate metricile detaliate, analizele per clasă și comparațiile transversale între modele, taskuri și configurații.

5. EVALUAREA MODELELOR

Într-un sistem de monitorizare a voicebotilor, modelul lingvistic nu este doar o componentă tehnică - este componenta de care depinde întreaga utilitate a sistemului. Dacă modelul nu identifică corect intenția clientului, dacă clasifică greșit statusul unei conversații sau dacă nu detectează o neconcordanță din răspunsul voicebotului, pipeline-ul de monitorizare produce rezultate care nu ajută sau, mai rău, pot induce în eroare echipele de asigurare a calității. Evaluarea riguroasă nu este, prin urmare, o etapă opțională, ci condiția de validitate a întregii lucrări.

Provocarea centrală a acestei evaluări este distincția dintre a genera un răspuns și a genera răspunsul corect, într-un format parsabil, pe date reale în limba română. Un model poate produce text fluent și convingător, dar dacă outputul nu respectă schema JSON așteptată de parser, dacă confundă două clase semantice apropiate sau dacă returnează o etichetă inexistentă în taxonomia definită, rezultatul este inutilizabil pentru un pipeline automat. Această evaluare măsoară tocmai această capacitate: nu doar înțelegerea limbajului, ci și disciplina structurală a răspunsului și acuratețea clasificării.

Experimentele acoperă cele trei taskuri ale sistemului. Nucleul evaluării constă în cinci modele testate complet pe toate cele trei taskuri, în toate versiunile de prompt și în ambele limbi: două modele comerciale accesate prin API — OpenAI o3 și Gemini 2.5 Flash — două modele generative open-source rulate local (Aya Expanse 8B și RoLLaMA 2 7B) și un model encoder-only (XLM-RoBERTa). Un al doilea model encoder-only, RoBERT-base, a fost inclus inițial în evaluare, dar exclus după taskul de extragere a intenției din cauza performanței scăzute, discutată în Secțiunea 5.2.2. Pe lângă acestea, Mistral 7B Instruct v0.3 și Qwen2.5 3B Instruct au fost incluse ca punct de comparație suplimentar. Ambele modele sunt mai recente decât cele din nucleul evaluării și reprezintă o generație diferită de modele open-source: Mistral aduce îmbunătățiri semnificative de eficiență față de modelele de generație anterioară la parametri echivalenți, iar Qwen2.5 3B oferă posibilitatea de a testa limita inferioară a dimensiunii modelului - un aspect relevant pentru scenariile cu resurse hardware limitate. Dat fiind costul ridicat al rulării complete pe GPU la distanță, Mistral și Qwen sunt evaluate exclusiv pe configurația identificată ca optimă în experimentele anterioare: EN v4 pentru extragerea intenției, RO v4 pentru statusul final și EN v2 pentru detecția neconcordanțelor.

Pe lângă metricile standard de clasificare, evaluarea analizează aspecte cu relevanță directă pentru implementarea practică: latența de răspuns, rata de parse failure, eficiența raportată la dimensiunea modelului și impactul limbii promptului. O atenție specială este acordată comparației dintre modelele API și cele locale - distincție relevantă pentru orice organizație care trebuie să aleagă între performanța maximă a modelelor comerciale și nevoia de a procesa date sensibile ale clienților exclusiv în infrastructura proprie.

5.1. Metodologia de evaluare

5.1.1. Designul experimental

Spațiul experimental este definit de patru factori: modelul evaluat, limba promptului (română sau engleză), versiunea promptului (v1–v4) și taskul de clasificare. În practică, fiecare combinație validă dintre acești factori constituie un experiment - o rulare completă a unui model pe cele 180 de conversații din dataset, cu un prompt specific, urmată de evaluarea rezultatelor folosind metricile definite in subcapitolul următor.

Mistral 7B și Qwen2.5 3B sunt evaluate doar pe configurația identificată ca optimă în experimentele anterioare: EN v4 pentru extragerea intenției, RO v4 pentru statusul final și EN v2 pentru detecția neconcordanțelor. Această decizie are o motivație practică directă: ambele modele rulează pe Google Colab cu o cuantizare 4-bit NF4 (Normal Float 4), unde fiecare rulare completă pe 180 de conversații durează între 30 de minute și câteva ore în funcție de task, iar accesul la GPU este limitat ca timp și disponibilitate. Rularea tuturor combinațiilor de limbă și versiune ar fi multiplicat costul fără a aduce informație suplimentară relevantă, dat fiind că scopul includerii lor este comparația pe configurația de vârf, nu profilarea completă.

În total, evaluarea cuprinde 44 de experimente pe taskul de extragere a intenției, 42 pe clasificarea statusului final și 42 pe detecția neconcordanțelor.

Datasetul utilizat constă în 180 de conversații bancare simulate în limba română, adnotate automat și verificate manual pe toate cele trei taskuri. Distribuția claselor este dezechilibrată, dar intenționat: dezechilibrul reflectă proporțiile din conversațiile reale de call center, nu o limitare a colectării. În practică, majoritatea interacțiunilor cu un voicebot bancar se finalizează cu succes, iar majoritatea răspunsurilor voicebotului sunt corecte - un dataset perfect echilibrat ar fi artificial și ar supra-reprezenta cazurile problematice. Concret, pe taskul de status final clasa *rezolvata* reprezintă 47,2% din exemple, iar pe detecția neconcordanțelor 80% din conversații nu conțin nicio neconcordanță. Dezechilibrul contează pentru evaluare pentru că influențează direct ce poate ascunde o metrică simplistă: un model care prezice mereu clasa majoritară obține acurateță ridicată fără nicio capacitate reală de discriminare. Alegerea metricii principale ține cont de acest aspect și este detaliată mai jos.

Configurația optimă per model este selectată după criteriul Macro-F1 maxim, cu acuratețea ca departajare în caz de egalitate. Macro-F1 este metrica principală deoarece tratează toate clasele echivalent, indiferent de câte exemple are fiecare: un F1 slab pe o clasă minoritară scade media la fel de mult ca un F1 slab pe clasa majoritară. Aceasta este proprietatea dorită într-un sistem de monitorizare, unde ratarea unui tip rar de problemă este la fel de relevantă ca ratarea unuia frecvent.

5.1.2. Metrici de evaluare

Evaluarea folosește mai multe metrici complementare, fiecare surprinzând un aspect diferit al calității clasificării.

Acuratețea este proporția predicțiilor corecte din total. Este ușor de interpretat, dar înșelătoare pe date dezechilibrate: pe taskul de status final, un model care prezice mereu *rezolvata* obține 47% acurateță fără să fi învățat nimic din datele de intrare.

Macro-F1 este media aritmetică a scorurilor F1 calculate per clasă, unde F1 combină precizia și recall-ul. Spre deosebire de acurateță, Macro-F1 penalizează explicit performanța slabă pe clasele minoritare și nu poate fi mărit artificial prin predicția clasei dominante.

Weighted F1 calculează aceeași medie, dar ponderată cu numărul de exemple per clasă.

Clasele frecvente contribuie mai mult la scor, ceea ce îl face mai aproape de perspectiva operațională - dacă clasa *rezolvata* apare în jumătate din conversații, performanța pe ea contează mai mult în practică.

Cohen's κ măsoară cât de mult depășește acordul dintre model și adnotator acordul așteptat întâmplător. Un model care ghicește aleatoriu obține $\kappa \approx 0$; un model perfect obține $\kappa = 1$. Interpretarea urmează scala propusă de Landis și Koch [32]: $\kappa < 0{,}20$ - *Slight*, 0,21–0,40 - *Fair*, 0,41–0,60 - *Moderate*, 0,61–0,80 - *Substantial*, $> 0{,}80$ - *Almost Perfect*.

Pe lângă metricile agregate, sunt raportate precision, recall și F1 per clasă. Acestea sunt utile pentru a identifica exact unde greșește un model: o metrică agregată bună poate ascunde faptul că o anumită clasă este sistematic ignorată sau confundată cu alta. Sunt raportate și statistici de latență - medie, mediană, deviație standard și percentila 95. Latența medie poate fi distorsionată de câteva răspunsuri foarte lente, dând o imagine prea optimistă a vitezei reale. Percentila 95 este mai relevantă în practică: înseamnă că 95% din răspunsuri se încadrează sub acea valoare, ceea ce o face un indicator mai potrivit pentru estimarea unui nivel de serviciu acceptabil într-un sistem de producție.

Predicțiile invalide - rezultate care nu pot fi parsate ca JSON sau care conțin etichete inexistente în taxonomia definită - sunt mapate la clasa `__invalid__` și penalizate explicit în calculul tuturor metricilor. Excluderea lor din evaluare ar umfla artificial scorurile modelelor cu rate mari de parse failure și ar ascunde o limitare practică reală.

5.1.3. Modulul de evaluare automată

Toate experimentele sunt procesate de scriptul `evaluate_models.py`, un modul unificat care citește outputurile JSON ale modelelor, normalizează etichetele și calculează toate metricile descrise anterior. Normalizarea acoperă variante comune de scriere: diacritice lipsă, sinonime în engleză pentru etichete românești, spații suplimentare, ghilimele diferite și variații de majuscule. Detecția câmpului de predicție din JSON folosește o listă ordonată de chei candidate (`predicted_intent`, `predicted_label`, `prediction` etc.), astfel încât outputurile cu structuri ușor diferite sunt procesate corect fără modificarea codului de evaluare.

Scriptul generează automat tabele LaTeX gata de inclus în lucrare, rapoarte text detaliate prin comentarii interpretative și suportă filtre prin linia de comandă pentru model, limbă și versiune de prompt, asigurând reproductibilitatea completă a oricărui sub-experiment.

```
# Evaluare completa cu generare LaTeX
python evaluate_models.py --task all --save-latex

# Filtrare pentru un singur model si task
python evaluate_models.py --task intent --model openai_o3
```

Listing 5.1: Exemple de utilizare a modulului de evaluare

5.2. Modelele evaluate

5.2.1. Prezentarea modelelor

Modelele evaluate pot fi împărțite în trei categorii distincte din punct de vedere arhitectural și al modalității de acces.

Modele API (cloud). OpenAI o3 și Gemini 2.5 Flash sunt modele comerciale de mari dimensiuni, accesate prin API. Dimensiunea exactă a parametrilor nu este publicată de niciunul dintre furnizori; estimările plasează o3 în categoria modelelor de ordinul sutelor de miliarde de

parametri. Ambele modele au capacități extinse de urmărire a instrucțiunilor și sunt antrenate masiv pe date multilingve. Accesul prin API implică transmiterea datelor pe rețea și facturarea per token, ceea ce ridică întrebări legate de confidențialitate și costuri de operare.

Modele generative locale. Aya Expanse 8B (Cohere For AI), RoLLaMA 2 7B (OpenLLM-Ro, Universitatea Politehnica din București), Mistral 7B Instruct (Mistral AI) și Qwen2.5 3B Instruct (Alibaba Cloud) sunt modele open-source rulate fără acces la un serviciu extern. Aya Expanse 8B este rulat local prin Ollama cu cuantizare pe 4 biți, necesitând aproximativ 4-5 GB RAM. RoLLaMA 2 7B, Mistral 7B și Qwen2.5 3B sunt rulate pe Google Colab pe un GPU T4, cu cuantizare 4-bit NF4 prin biblioteca *bitsandbytes*, necesitând aproximativ 4-6 GB VRAM per model.[1] RoLLaMA 2 7B este singurul model din evaluare antrenat specific pe date în limba română, dezvoltat de echipa OpenLLM-Ro de la Universitatea Politehnica din București.

Modele encoder-only. XLM-RoBERTa (Meta/HuggingFace) și RoBERT-base (Readerbench Lab, UPB) nu generează text liber și funcționează fundamental diferit față de modelele generative. Clasificarea se realizează prin inferență textuală naturală (NLI - *Natural Language Inference*) în regim zero-shot: pentru fiecare etichetă candidată, modelul primește ca intrare textul conversației concatenat cu eticheta și calculează probabilitatea ca eticheta să fie o concluzie logică a textului. Eticheta cu probabilitatea maximă devine predicția finală.

5.2.2. Cazul RoBERT-base: limitele modelelor encoder-only mici

RoBERT-base a fost inclus în evaluare cu ipoteza că specializarea pe text românesc ar putea compensa dimensiunea redusă (125 milioane de parametri) față de alternativele multilingve. Ipoteza s-a dovedit incorectă. Pe sarcina de extragere a intenției, RoBERT-base obține o acuratețe de 1,7% și un $\kappa = -0{,}073$ - valoare negativă, indicând o performanță sub nivelul clasificării aleatorii. Analiza pe clase confirmă tabloul: pentru 10 din cele 12 clase, atât precizia cât și recallul sunt zero; singurele semnale de predicție provin din clasele *fallback* și *general_product_info*, care par cel mai ușor de asociat cu texte generice din etapa de preantrenare.

Cauza nu este lipsa cunoașterii limbii române, ci arhitectura modelului. Un model de tip encoder antrenat prin modelare a limbajului cu măști nu dispune de un mecanism nativ de clasificare în absența unui antrenament supervizat suplimentar. Clasificarea prin inferență textuală naturală necesită ca modelul să fi fost antrenat explicit pe sarcini de inferență; RoBERT-base nu include un astfel de antrenament, spre deosebire de XLM-RoBERTa, care a fost rafinat pe setul de date XNLI. Rezultatele sunt în acord cu observațiile din literatura privind clasificarea zero-shot, unde performanța nu depinde doar de acoperirea lingvistică a modelului, ci și de modul în care sarcina este formulată și de capacitatea modelului de a alinia textul de intrare cu descrierea etichetelor [33].

5.2.3. Experimentul cu etichete în limba română

Experimentul adițional cu etichete traduse în română a fost realizat exclusiv pentru modelul RoLLaMA 2 7B, aplicat pe versiunea RO v3 - configurația cu cele mai bune rezultate pe română în cadrul experimentelor standard. Ipoteza testată a fost că etichetele în engleză (`block_card`, `open_account` etc.) introduc un decalaj lingvistic față de specializarea modelului pe date românești, iar traducerea lor ar putea îmbunătăți performanța.

---

[1] Formatele exacte de cuantizare utilizate: Q4_K_M pentru Aya Expanse prin Ollama, respectiv NF4 pentru modelele rulate pe Colab.

Rezultatele infirmă această ipoteză. Varianta cu etichete traduse (RO v3 etichete RO) obține M-F1 = 0,0145 și o acuratețe de 9,5%, față de M-F1 = 0,0608 pe aceeași versiune de prompt cu etichete în engleză (RO v3), și față de M-F1 = 0,1201 al configurației optime a modelului (EN v1). Degradarea este însoțită de o rată ridicată de eșec la parsare: 54 din cele 180 de conversații (30%) nu au produs un răspuns parsabil, față de doar 3 eșecuri pe RO v3 standard. Coeficientul Cohen's κ coboară la 0,000, echivalent clasificării aleatorii.

Un tipar similar apare și pe versiunea v4, unde RO v4 înregistrează 64 de eșecuri la parsare și M-F1 = 0,0167, iar EN v4 obține M-F1 = 0,0143 cu $\kappa \approx 0$ - ambele semnificativ sub EN v1. Acest comportament sugerează că prompturile mai lungi și mai complexe (v3, v4) depășesc capacitatea efectivă a modelului cuantizat de a urma instrucțiuni structurate, indiferent de limba etichetelor. O ipoteză plauzibilă este că modelul a fost antrenat preponderent pe date de tip instrucțiune-răspuns în care etichetele de clasificare apar în engleză; înlocuirea lor cu variante românești perturbă tiparele de generare JSON fără a oferi un beneficiu compensator, deoarece problema fundamentală nu este recunoașterea semantică a etichetelor, ci disciplina structurală a răspunsului în regim zero-shot pe prompturi extinse.

5.3. Rezultate pe taskul de extragere a intenției

5.3.1. Rezultate generale

Taskul de extragere a intenției constă în clasificarea fiecărei conversații în una din 12 clase de intenție. Distribuția este relativ echilibrată (14-16 exemple per clasă), ceea ce face din Macro-F1 o metrică fiabilă și comparabilă cu acuratețea.

Tabelul 5.1: Cele mai bune configurații per model: Extragerea intenției

Model	Config	Acc	M-F1	W-F1	κ	Interp
OpenAI o3	EN v4	98,3%	0,983	0,983	0,982	Almost Perfect
Gemini 2.5 Flash	RO v4	97,8%	0,977	0,977	0,976	Almost Perfect
Aya Expanse 8B	EN v4	91,1%	0,909	0,910	0,903	Almost Perfect
Mistral 7B	EN v4	89,4%	0,887	0,885	0,885	Almost Perfect
Qwen2.5 3B	EN v4	75,0%	0,697	0,703	0,727	Substantial
XLM-RoBERTa	EN v1	53,9%	0,552	0,550	0,497	Moderate
RoLLaMA 2 7B	EN v1	15,6%	0,119	0,120	0,084	Slight
RoBERT-base	RO v1	1,7%	0,012	0,012	-0,073	Poor

Tabelul 5.1 prezintă configurația optimă per model. OpenAI o3 (EN v4) atinge Macro-F1 = 0,983 și $\kappa = 0,982$ - acord aproape perfect conform scalei Landis & Koch. Singurele clase care nu ating F1 = 1,00 sunt *block_card* (F1 = 0,966, datorat unui singur caz de confuzie cu *unblock_card*) și *report_suspicious_transaction* (F1 = 0,933). Gemini 2.5 Flash (RO v4) urmează cu 0,977, cu diferența principală la *general_product_info* (recall = 0,800), sugerând că Gemini tinde să asocieze unele cereri generice altor intenții.

Aya Expanse 8B (EN v4) obține 0,909 - cel mai bun rezultat local - la o latență de aproximativ 94 de secunde per conversație, un contrast marcat față de modelele API. Mistral 7B (EN v4, 0,887) se situează la doar 2,2 puncte sub Aya, dar cu o latență de circa 9 secunde, oferind un compromis net favorabil. Ambele modele ating $\kappa > 0,88$, în zona *Almost Perfect*. Qwen2.5 3B (EN v4, 0,697) produce rezultate acceptabile pentru un model de 3 miliarde de parametri, fără niciun parse failure pe acest task, deși prezintă confuzii sistematice pe perechile

semantice apropiate: F1 = 0,000 pe *unblock_card* (toate exemplele clasificate ca *block_card*) și F1 = 0,000 pe *get_account_statement* (toate exemplele clasificate ca *check_balance*), ambele perechi implicând acțiuni similare pe același obiect bancar.

XLM-RoBERTa produce M-F1 = 0,552 identic pe toate cele opt configurații testate (EN/RO × v1–v4), confirmând că modelul ignoră complet conținutul instructiv al promptului și compară pur semantic textul conversației cu etichetele. RoLLaMA 2 7B (M-F1 = 0,119) și RoBERT-base (M-F1 = 0,012) nu ating un nivel de performanță utilizabil. RoLLaMA prezice aproape exclusiv *block_card* (recall = 1,000, precision = 0,086), producând F1 = 0,000 pe 8 din cele 12 clase. RoBERT-base produce F1 = 0,000 pe 10 din cele 12 clase, cu semnale minime doar pe *general_product_info* (F1 = 0,029) și *fallback* (F1 = 0,111).

5.3.2. Analiza per clasă

RoLLaMA 2 7B și RoBERT-base nu sunt incluse în analiza per clasă deoarece rezultatele lor nu oferă informație interpretabilă.

Tabelul 5.2: Metrici per clasă - modele API (P = Precizie, R = Recall)

Clasă	**Nr.**	**OpenAI o3 (EN v4)**			**Gemini 2.5 Flash (RO v4)**		
		P	**R**	**F1**	**P**	**R**	**F1**
block_card	14	0,933	1,000	0,966	1,000	1,000	1,000
unblock_card	14	1,000	1,000	1,000	1,000	1,000	1,000
open_account	15	1,000	1,000	1,000	0,938	1,000	0,968
close_account	15	1,000	1,000	1,000	1,000	1,000	1,000
check_balance	15	1,000	1,000	1,000	1,000	1,000	1,000
get_account_statement	15	1,000	1,000	1,000	1,000	1,000	1,000
report_suspicious_transaction	15	0,933	0,933	0,933	0,938	1,000	0,968
update_personal_data	16	1,000	1,000	1,000	1,000	1,000	1,000
schedule_advisor_meeting	15	0,938	1,000	0,968	0,882	1,000	0,938
reset_or_recover_auth	15	1,000	1,000	1,000	1,000	1,000	1,000
general_product_info	15	1,000	0,933	0,966	1,000	0,800	0,889
fallback	16	1,000	0,938	0,968	1,000	0,938	0,968

Tabelul 5.3: Metrici per clasă - cele mai bune modele locale

Clasă	**Nr.**	**Aya Expanse 8B (EN v4)**			**Mistral 7B (EN v4)**		
		P	**R**	**F1**	**P**	**R**	**F1**
block_card	14	0,700	1,000	0,824	0,867	0,929	0,897
unblock_card	14	0,846	0,786	0,815	1,000	0,857	0,923
open_account	15	0,789	1,000	0,882	0,882	1,000	0,938
close_account	15	1,000	1,000	1,000	1,000	0,933	0,966
check_balance	15	0,933	0,933	0,933	0,778	0,933	0,848
get_account_statement	15	1,000	1,000	1,000	1,000	0,933	0,966
report_suspicious_transaction	15	0,833	1,000	0,909	0,750	1,000	0,857
update_personal_data	16	1,000	1,000	1,000	1,000	1,000	1,000
schedule_advisor_meeting	15	1,000	1,000	1,000	0,789	1,000	0,882
reset_or_recover_auth	15	1,000	0,733	0,846	0,938	1,000	0,968
general_product_info	15	1,000	0,800	0,889	1,000	0,867	0,929
fallback	16	1,000	0,688	0,815	1,000	0,312	0,476

Tabelul 5.4: Scorul F1 per clasă - Qwen2.5 3B și XLM-RoBERTa

Clasă	Nr.	Qwen2.5 3B (EN v4)	XLM-RoBERTa (EN v1)
block_card	14	0,560	0,625
unblock_card	14	0,000	0,714
open_account	15	0,938	0,688
close_account	15	1,000	0,500
check_balance	15	0,638	0,435
get_account_statement	15	0,000	0,432
report_suspicious_transaction	15	0,786	0,562
update_personal_data	16	1,000	0,667
schedule_advisor_meeting	15	1,000	0,615
reset_or_recover_auth	15	0,897	0,571
general_product_info	15	0,933	0,593
fallback	16	0,609	0,222

Tabelele 5.2–5.4 prezintă metricile per clasă pentru toate modelele cu performanță interpretabilă. Al treilea tabel include doar F1, deoarece Qwen și XLM-RoBERTa nu prezintă tipare de eroare care să necesite analiza separată a preciziei și recall-ului.

Modelele API obțin performanțe aproape perfecte pe majoritatea claselor. OpenAI o3 atinge F1 = 1,000 pe 7 din 12 clase; abaterile rămase sunt minore și distribuite uniform — *report_suspicious_transaction* (F1 = 0,933, datorat a două confuzii reciproce), *block_card* (F1 = 0,966, un singur fals pozitiv), și trei clase cu F1 între 0,966 și 0,968. Gemini 2.5 Flash prezintă o singură slăbiciune notabilă: *general_product_info* (F1 = 0,889, recall = 0,800), unde 3 din 15 conversații cu cereri generice sunt clasificate în alte intenții. Cauza probabilă este că aceste conversații conțin termeni bancari specifici care activează intenții mai precise decât *general_product_info*.

Aya Expanse 8B prezintă cel mai clar tipar de confuzie pe perechea *block_card*/*unblock_card*: precizie = 0,700 pe *block_card* indică false pozitive - conversații de *unblock_card* clasificate greșit - iar recall-ul = 0,786 pe *unblock_card* confirmă că modelul ratează o parte din cererile de deblocare. Ambele intenții implică acțiuni pe același obiect (cardul), iar distincția direcției acțiunii (blocare vs. deblocare) necesită înțelegerea contextului complet al conversației. Aya prezintă de asemeneaun recall scăzută pe *fallback* (0,688) și *reset_or_recover_auth* (0,733).

Mistral 7B este consistent pe 11 din 12 clase, cu F1 $\geq$ 0,848 peste tot cu excepția unui singur colaps: *fallback* (F1 = 0,476, recall = 0,312). Modelul identifică corect doar 5 din cele 16 conversații cu solicitări în afara domeniului bancar, mapând restul la intenții bancare existente. Comportamentul este problematic în cazuri reale unde *fallback* semnalează solicitări ce nu pot fi gestionate de voicebot.

Qwen2.5 3B prezintă confuzii sistematice pe două perechi: F1 = 0,000 pe *unblock_card* (toate cele 14 exemple clasificate ca *block_card*) și F1 = 0,000 pe *get_account_statement* (toate clasificate ca *check_balance*). Pe restul claselor, Qwen obține rezultate surprinzător de bune pentru dimensiunea sa - F1 = 1,000 pe *close_account*, *update_personal_data* și *schedule_advisor_meeting*. XLM-RoBERTa distribuie erorile uniform pe toate clasele, fără colapsuri complete dar fără clase stăpânite; cel mai slab scor este pe *fallback* (F1 = 0,222), unde mecanismul NLI nu reușește să identifice absența unei intenții bancare definite.

5.3.3. Analiza erorilor și evoluția prompturilor

Erorile sunt concentrate în perechi - *block_card*/*unblock_card* și *check_balance*/*get_account_statement* - și apar transversal la mai multe modele. Aceasta sugerează că distincția nu este doar o problemă de capacitate, ci și o ambiguitate reală în date: unele conversații scurte nu furnizează suficient context pentru a discrimina cele două intenții fără informații despre starea curentă a cardului sau contului.

Evoluția pe versiunile de prompt relevă câștiguri clare pentru modelele cu capacitate suficientă. Aya Expanse 8B progresează pe EN de la 0,842 (v1) la 0,845 (v2), 0,872 (v3) și 0,909 (v4), un câștig total de +6,7 puncte. OpenAI o3 câștigă +3,1 puncte pe EN (0,952 la v1, 0,983 la v3), atingând plafonul la v3 și menținând același scor la v4. Un tipar important este că o3 pe RO scade la v4 față de v3 (0,944 față de 0,972). XLM-RoBERTa produce M-F1 = 0,552 identic pe toate cele opt configurații testate, confirmând că modelul nu procesează conținutul promptului.

5.4. Rezultate pe taskul de clasificare a statusului final

5.4.1. Rezultate generale

Sarcina de clasificare a statusului final este mai dificilă decât extragerea intenției din două motive care se combină. În primul rând, distribuția claselor este dezechilibrată: clasa *rezolvata* reprezintă 47,2% din conversații. Un model care prezice *rezolvata* pentru orice conversație ar obține 47% acurateță fără să fi înțeles nimic - tocmai de aceea Macro-F1 este metrica principală, nu acuratețea. În al doilea rând, granițele dintre clase nu sunt definite de cuvinte cheie, ci de evoluția întregii conversații. Diferența dintre *nerezolvata* și *intrerupta*, de exemplu, nu ține de ce s-a spus, ci de cum s-a încheiat interacțiunea - dacă clientul a abandonat activ sau dacă voicebotul pur și simplu nu a reușit să rezolve cererea. Această distincție necesită înțelegerea contextului conversațional în ansamblu, nu doar identificarea unor expresii specifice.

Tabelul 5.5: Cele mai bune configurații per model - Clasificarea statusului final

Model	Config	Acc	M-F1	W-F1	κ	Interp
OpenAI o3	RO v4	90,0%	0,860	0,899	0,856	Almost Perfect
Gemini 2.5 Flash	EN v4	86,1%	0,834	0,863	0,804	Almost Perfect
Aya Expanse 8B	EN v3	72,2%	0,620	0,690	0,580	Moderate
Mistral 7B	RO v4	68,9%	0,548	0,667	0,537	Moderate
Qwen2.5 3B	RO v4	51,1%	0,493	0,542	0,365	Fair
XLM-RoBERTa	EN v1	22,8%	0,224	0,237	0,069	Slight
RoLLaMA 2 7B	RO v3	41,1%	0,155	0,316	0,003	Slight

Tabelul 5.5 prezintă configurația optimă per model. OpenAI o3 pe promptul RO v4 obține Macro-F1 = 0,860 și $\kappa = 0,856$, cel mai bun rezultat pe acest task. Remarcabil este faptul că o3 preferă promptul în limba română (+2,8 puncte acurateță față de EN v4, care obține 87,2%) - singura situație în care limba română produce o diferență clară pozitivă la un model API, discutată în Secțiunea 5.6.4. Gemini 2.5 Flash (EN v4, M-F1 = 0,834) urmează, cu o latență de aproape două ori mai mică decât o3 (4 769 ms față de 8 916 ms).

Aya Expanse 8B (EN v3, M-F1 = 0,620) prezintă un comportament contra-intuitiv: performanța scade de la v3 la v4 (de la 0,620 la 0,531 pe EN, de la 0,546 la 0,392 pe RO). Introducerea exemplelor few-shot în v4 nu îmbunătățește Aya pe acest task, ci degradează

rezultatele semnificativ - un indiciu că promptul v4 depășește fereastra de context efectivă a modelului cuantizat de 8B sau că exemplele generează confuzie în loc de clarificare. Aceeași tendință apare și pe detecția neconcordanțelor (discutată în Secțiunea 5.5.4).

Mistral 7B (RO v4, M-F1 = 0,548) și Qwen2.5 3B (RO v4, M-F1 = 0,493) se situează sub Aya, dar cu latențe semnificativ mai mici: 22,7s pentru Mistral și 13,9s pentru Qwen, față de 77,6s pentru Aya. Diferența de latență trebuie interpretată cu precauție, deoarece condițiile de inferență nu sunt identice: Mistral și Qwen sunt rulate pe Google Colab cu un GPU T4, în timp ce Aya este rulată local prin Ollama pe hardware propriu. Qwen cu 3 miliarde de parametri obține de peste trei ori performanța RoLLaMA cu 7 miliarde de parametri (M-F1 = 0,493 față de 0,155) - un model de mai mult decât dublul dimensiunii, ceea ce confirmă că arhitectura și calitatea antrenamentului contează mai mult decât numărul brut de parametri.

XLM-RoBERTa (EN v1, M-F1 = 0,224) prezintă variații neglijabile între versiuni (M-F1 între 0,216 și 0,224), similar comportamentului identic observat pe intenție. RoLLaMA 2 7B (RO v3, M-F1 = 0,155, $\kappa = 0,003$) - performanță la nivelul clasificării aleatorii.

5.4.2. Analiza per clasă

RoLLaMA 2 7B nu este inclus în analiza per clasă deoarece produce F1 = 0,000 pe 3 din cele 5 clase (*nerezolvata*, *redirectionata*, *intrerupta*), prezicând preponderent *rezolvata* (recall = 0,800, precizie = 0,479). Celelalte șase modele sunt prezentate în trei tabele grupate după nivelul de performanță.

Tabelul 5.6: Metrici per clasă - modele API (P = Precizie, R = Recall)

Clasă	**Nr.**	**OpenAI o3 (RO v4)**			**Gemini 2.5 Flash (EN v4)**		
		P	**R**	**F1**	**P**	**R**	**F1**
rezolvata	85	0,943	0,976	0,960	0,950	0,894	0,921
partial_rezolvata	34	0,848	0,824	0,836	0,730	0,794	0,761
redirectionata	28	0,926	0,893	0,909	0,862	0,893	0,877
intrerupta	20	0,882	0,750	0,811	0,882	0,750	0,811
nerezolvata	13	0,733	0,846	0,786	0,706	0,923	0,800

Tabelul 5.7: Metrici per clasă - cele mai bune modele locale

Clasă	**Nr.**	**Aya Expanse 8B (EN v3)**			**Mistral 7B (RO v4)**		
		P	**R**	**F1**	**P**	**R**	**F1**
rezolvata	85	0,790	0,976	0,874	0,825	0,941	0,879
partial_rezolvata	34	0,407	0,324	0,361	0,404	0,559	0,469
redirectionata	28	0,688	0,786	0,733	0,739	0,607	0,667
intrerupta	20	1,000	0,200	0,333	1,000	0,100	0,182
nerezolvata	13	0,833	0,769	0,800	0,667	0,462	0,545

Tabelul 5.8: Scorul F1 per clasă - Qwen2.5 3B și XLM-RoBERTa

Clasă	Nr.	Qwen2.5 3B (RO v4)			XLM-R (EN v1)
		P	R	F1	F1
rezolvata	85	1,000	0,447	0,618	0,257
partial_rezolvata	34	0,300	0,882	0,448	0,182
redirectionata	28	0,842	0,571	0,681	0,286
intrerupta	20	0,273	0,150	0,194	0,227
nerezolvata	13	0,833	0,385	0,526	0,167

Tabelele 5.6-5.8 prezintă metricile per clasă pentru toate modelele cu performanță interpretabilă. Clasele sunt ordonate descrescător după suport (numărul de exemple per clasă). Al treilea tabel include precizia și recall-ul doar pentru Qwen, deoarece prezintă un tipar de eroare specific discutat mai jos; pentru XLM-RoBERTa este raportat doar F1.

Clasa *rezolvata* (85 exemple, 47,2% din dataset) este identificată bine de toate modelele capabile: F1 = 0,960 la o3, 0,921 la Gemini, 0,874 la Aya și 0,879 la Mistral. Suportul ridicat și tiparele lingvistice clare de rezoluție (voicebotul confirmă efectuarea operației, clientul mulțumește) fac din aceasta clasa cea mai ușor de clasificat.

Clasa *redirectionata* (28 exemple) este de asemenea relativ bine separată: F1 = 0,909 la o3, 0,877 la Gemini. Conversațiile redirecționate implică un tipar distinct - voicebotul anunță explicit transferul apelului către un operator uman sau un alt departament, incluzând specific data și ora pentru programare - ceea ce oferă un semnal lexical clar.

Clasele cele mai problematice sunt *intrerupta* și *partial_rezolvata*. La Mistral, *intrerupta* obține F1 = 0,182 (recall = 0,100): modelul identifică corect doar 2 din cele 20 de conversații întrerupte, dar când prezice *intrerupta* o face corect (precizie = 1,000). La Aya, recall-ul pe *intrerupta* este 0,200 - doar 4 din 20 identificate corect, de asemenea cu precizie perfectă. Ambele modele locale preferă să clasifice conversațiile întrerupte ca *nerezolvata*, ceea ce este ușor de înțeles: în ambele cazuri conversația nu s-a finalizat pozitiv, diferența constând în dacă clientul a abandonat activ (*intrerupta*) sau voicebotul pur și simplu nu a reușit să rezolve cererea (*nerezolvata*) - o distincție care necesită înțelegerea modului în care se încheie conversația, nu doar a conținutului ei. Chiar și modelele API au dificultăți: o3 și Gemini obțin ambele recall = 0,750 pe *intrerupta*, ratând 5 din 20 de conversații.

Partial_rezolvata (34 exemple) este a doua clasă dificilă. Aya obține F1 = 0,361 (recall = 0,324), Mistral F1 = 0,469 (recall = 0,559). Clasa presupune că voicebotul a răspuns parțial la cererea clientului - de exemplu, a furnizat informații despre sold dar nu a reușit să proceseze un transfer solicitat în aceeași conversație. Granița cu *rezolvata* este subiectivă: depinde de ce considerăm "suficient" pentru o rezolvare completă.

Qwen2.5 3B prezintă un tipar de eroare specific și simetric: precizie = 1,000 pe *rezolvata* dar recall = 0,447, ceea ce înseamnă că modelul plasează peste jumătate din conversațiile rezolvate în alte clase. Simultan, *partial_rezolvata* are recall = 0,882 dar precizie = 0,300 - modelul atrage în această clasă atât conversațiile corect parțial rezolvate cât și multe conversații care sunt de fapt complet rezolvate. Qwen manifestă o tendință sistematică de prudență: preferă să clasifice o conversație ca parțial rezolvată decât să declare rezolvarea completă fără suficientă certitudine.

XLM-RoBERTa distribuie erorile uniform pe toate clasele, cu F1 între 0,167 și 0,286, fără colapsuri complete dar fără clase stăpânite - la fel ca pe taskul de intenție, mecanismul NLI nu reușește să discrimineze clase care necesită înțelegerea contextului conversațional.

5.4.3. Evoluția prompturilor

Evoluția prompturilor produce câștiguri consistente și substanțiale pentru modelele API pe clasificarea statusului final: OpenAI o3 câștigă +15,9 puncte pe RO (de la 0,701 la v1 la 0,860 la v4) și +15,1 pe EN (de la 0,688 la 0,839); Gemini câștigă +15,8 pe RO (de la 0,648 la 0,806) și +12,8 pe EN (de la 0,706 la 0,834). Aceste câștiguri reprezintă cel mai mare salt absolut dintre cele trei taskuri, reflectând că sarcina este mai sensibilă la calitatea promptului: definițiile clare ale claselor și exemplele structurate reduc ambiguitatea granițelor.

Prin contrast, Aya Expanse înregistrează declin pe versiunile finale: -22,8 puncte pe RO (de la 0,620 la v1 la 0,392 la v4) și -6,7 pe EN (de la 0,598 la 0,531). Declinul este și mai abrupt pe RO v4, unde latența crește la 515 secunde per conversație - peste 8 minute - sugerând că modelul cuantizat epuizează practic fereastra de context efectivă. RoLLaMA oscilează fără o tendință clară (M-F1 între 0,040 și 0,155), iar XLM-RoBERTa rămâne stabil (M-F1 între 0,216 și 0,224).

5.5. Rezultate pe taskul de detecție a neconcordanțelor

Detecția neconcordanțelor este cea mai dificilă sarcină din cele trei. Dificultatea provine din mai multe surse care se combină. Distribuția claselor este puternic dezechilibrată - 80% din conversații nu conțin nicio neconcordanță - dar dezechilibrul reflectă realitatea: într-un sistem funcțional, majoritatea răspunsurilor voicebotului sunt corecte, iar neconcordanțele sunt excepția, nu regula. Numărul de exemple per tip de neconcordanță este mic (5-10 per clasă), granițele dintre tipuri sunt adesea subiective, iar identificarea unei neconcordanțe presupune înțelegerea profundă a contextului conversațional - modelul trebuie să evalueze nu doar ce a spus voicebotul, ci dacă răspunsul este corect în raport cu cererea clientului. Sarcina este evaluată pe două niveluri: detecție binară (există sau nu o neconcordanță în conversație) și clasificare pe tipuri (cele 6 clase: *none* plus cele 5 tipuri de neconcordanțe definite în taxonomie).

5.5.1. Cele mai bune rezultate per model

Tabelul 5.9: Cele mai bune configurații per model - Clasificarea tipului de neconcordanță (6 clase, 180 conversații)

Model	Config	Acc	M-F1	W-F1	κ	Interp
Gemini 2.5 Flash	EN v4	89,4%	0,734	0,898	0,713	Substantial
OpenAI o3	EN v4	90,0%	0,705	0,892	0,682	Substantial
Aya Expanse 8B	EN v3	71,1%	0,291	0,722	0,285	Fair
Qwen2.5 3B	EN v4	80,0%	0,284	0,761	0,278	Fair
Mistral 7B	EN v4	82,2%	0,281	0,756	0,196	Slight
RoLLaMA 2 7B	EN v1	76,1%	0,175	0,700	0,033	Slight
XLM-RoBERTa	EN v4	8,9%	0,040	0,126	−0,007	Poor

Tabelul 5.9 prezintă cele mai bune configurații per model pe clasificarea tipului de neconcordanță, evaluată pe toate cele 180 de conversații cu 6 clase (inclusiv *none*). Modelele API obțin performanțe bune pe ambele dimensiuni ale sarcinii: atât pe identificarea conversațiilor fără probleme, cât și pe clasificarea tipurilor de neconcordanțe. Gemini 2.5 Flash atinge M-F1 = 0,734 și $\kappa = 0,713$ - acord substanțial conform scalei Landis și Koch

[32] - cu o acuratețe de 89,4%. OpenAI o3 (M-F1 = 0,705, $\kappa = 0,682$) urmează cu rezultate similare. Deși o3 obține acuratețe ușor mai mare (90,0% față de 89,4%), Gemini are M-F1 superior deoarece Macro-F1 tratează toate cele 6 clase egal. O3 câștigă pe acuratețe datorită recall-ului mai mare pe *none* (0,979 față de 0,938) - clasifică corect mai multe conversații fără probleme, care reprezintă 80% din dataset. Gemini însă este mai echilibrat pe clasele minoritare (*nealiniat_context*: 0,545 față de 0,400; *irelevant*: 0,769 față de 0,545), ceea ce contează mai mult pentru un sistem de monitorizare care trebuie să identifice natura problemelor, nu doar absența lor.

Diferența dintre Macro-F1 și Weighted-F1 este relevantă pe acest task. Gemini obține W-F1 = 0,898, semnificativ mai mare decât M-F1 = 0,734, deoarece performanța pe clasa dominantă *none* (F1 = 0,951, support = 144) contribuie proporțional mai mult la scorul ponderat. Macro-F1, care tratează toate cele 6 clase egal, este tras în jos de clasele minoritare mai dificile - și este metrica relevantă pentru evaluarea capacității de clasificare.

Un rezultat important: Qwen2.5 3B (M-F1 = 0,284) și Mistral 7B (M-F1 = 0,281) obțin scoruri foarte apropiate de Aya Expanse 8B (M-F1 = 0,291), deși au mai puțini parametri. Toate trei modelele locale se situează în zona *Fair* sau la limita ei, cu o diferență de doar 1 punct M-F1 între Aya (8B) și cele două modele mai mici. Gapul real se află între modelele API și cele locale, nu în interiorul categoriei locale.

RoLLaMA 2 7B (EN v1, M-F1 = 0,175, $\kappa = 0,033$) se situează sub celelalte modele locale. Spre deosebire de Mistral și Qwen, configurația optimă rămâne v1 - modelul nu beneficiază nici de definițiile extinse, nici de exemplele few-shot din versiunile ulterioare.

XLM-RoBERTa prezintă un comportament opus: acuratețe de doar 8,9% și κ negativ ($-0,007$). Mecanismul NLI identifică corect doar 12 din cele 144 de conversații fără probleme (recall = 0,083 pe *none*), asociind aproape orice text cu o etichetă de neconcordanță.

5.5.2. Detecție binară

Detecția binară - dacă o conversație conține sau nu o neconcordanță - este o sarcină mai accesibilă decât clasificarea pe tipuri, deoarece elimină necesitatea discriminării fine între categorii de erori.

Tabelul 5.10: Cele mai bune rezultate pe detecția binară a neconcordanțelor

Model	Config	Acc	F1	κ	Interp
Gemini 2.5 Flash	RO v3	92,8%	0,817	0,772	Substantial
OpenAI o3	EN v4	91,7%	0,762	0,713	Substantial
Aya Expanse 8B	EN v1	78,3%	0,541	0,404	Moderate
Qwen2.5 3B	EN v4	84,4%	0,481	0,402	Moderate
Mistral 7B	EN v4	82,8%	0,244	0,205	Fair
XLM-RoBERTa	EN v4	26,1%	0,345	0,023	Slight
RoLLaMA 2 7B	EN v1	77,2%	0,128	0,038	Slight

Tabelul 5.10 prezintă cele mai bune configurații per model pe detecția binară. Gemini 2.5 Flash obține cel mai bun scor (F1 = 0,817, $\kappa = 0,772$), urmat de OpenAI o3 (F1 = 0,762, $\kappa = 0,713$). Ambele modele se situează în zona *Substantial*, confirmând că pot detecta prezența neconcordanțelor cu o fiabilitate practică. Detecția neconcordanțelor este singura sarcină din cele trei pe care Gemini depășește o3 ca cel mai bun model.

Dintre modelele locale, Aya Expanse 8B (F1 = 0,541) și Qwen2.5 3B (F1 = 0,481) obțin

scoruri în zona *Moderate* ($\kappa \approx 0,40$). Mistral 7B (F1 = 0,244, $\kappa = 0,205$) se situează în zona *Fair*.

RoLLaMA 2 7B ilustrează paradoxul acurateții pe date dezechilibrate: acuratețe de 77,2% - aparent rezonabilă - dar F1 = 0,128 și $\kappa = 0,038$. Modelul prezice aproape exclusiv clasa majoritară (nu există neconcordanță), obținând acuratețe ridicată fără nicio capacitate reală de detecție. Acesta este exact scenariul pentru care metrica κ este esențială: prin corecția pentru acordul întâmplător, expune diferența dintre un model care clasifică și unul care doar ghicește clasa majoritară.

5.5.3. Analiza per clasă

Tabelul 5.11: Metrici per clasă - modele API pe clasificarea tipului de neconcordanță (P = Precizie, R = Recall)

Tip	Nr.	Gemini 2.5 Flash (EN v4)			OpenAI o3 (EN v4)		
		P	R	F1	P	R	F1
none	144	0,964	0,938	0,951	0,922	0,979	0,949
incomplet	8	0,778	0,875	0,824	0,714	0,625	0,667
irelevant	7	0,833	0,714	0,769	0,750	0,429	0,545
contradictoriu	6	0,714	0,833	0,769	1,000	1,000	1,000
nealiniat_context	5	0,500	0,600	0,545	0,400	0,400	0,400
halucinatie	10	0,500	0,600	0,545	1,000	0,500	0,667

Tabelul 5.12: Metrici per clasă - modele locale pe clasificarea tipului de neconcordanță (P = Precizie, R = Recall)

Tip	Nr.	Aya 8B (EN v3)			Mistral 7B (EN v4)			Qwen 3B (EN v4)		
		P	R	F1	P	R	F1	P	R	F1
none	144	0,894	0,819	0,855	0,823	1,000	0,903	0,858	0,965	0,908
incomplet	8	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
irelevant	7	0,227	0,714	0,345	1,000	0,429	0,600	1,000	0,143	0,250
contradictoriu	6	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
nealiniat_context	5	0,130	0,600	0,214	0,000	0,000	0,000	0,143	0,400	0,211
halucinatie	10	1,000	0,200	0,333	1,000	0,100	0,182	1,000	0,200	0,333

Tabelele 5.11 și 5.12 evidențiază structura detaliată a erorilor.

Ambele modele API clasifică excelent clasa *none*: Gemini cu F1 = 0,951 (ratează doar 9 din 144 conversații corecte) și o3 cu F1 = 0,949 (ratează doar 3). Profilurile celor două modele sunt complementare pe clasele minoritare. O3 excelează pe *contradictoriu* (F1 = 1,000 - identifică toate cele 6 conversații în care voicebotul contrazice direct informația din context) și pe *halucinatie* (F1 = 0,667, precizie = 1,000). Gemini este mai echilibrat pe clasele subtile: *nealiniat_context* (F1 = 0,545 față de 0,400 la o3) și *irelevant* (F1 = 0,769 față de 0,545).

Clasa *nealiniat_context* (5 exemple) rămâne cea mai dificilă din taxonomie. Conversațiile din această clasă implică un voicebot care răspunde tehnic corect, dar într-un context care nu se potrivește cu situația clientului - o distincție care necesită raționament despre coerența conversațională, nu doar verificarea conținutului. Chiar Gemini obține doar F1 = 0,545, iar o3 F1 = 0,400.

Modelele locale prezintă un contrast puternic între *none* și restul claselor. Mistral clasifică corect toate cele 144 de conversații fără neconcordanțe (recall = 1,000 pe *none*), dar obține F1 = 0,000 pe 3 din 5 tipuri. Recunoaște parțial *irelevant* (F1 = 0,600, precizie = 1,000 - când prezice *irelevant* nu greșește niciodată, dar identifică doar 3 din 7 cazuri) și detectează 1 din 10 halucinații (F1 = 0,182). Qwen prezintă un tipar similar: *none* excelent (F1 = 0,908), dar recunoaște parțial doar *halucinatie* (F1 = 0,333), *irelevant* (F1 = 0,250) și *nealiniat_context* (F1 = 0,211). Aya identifică parțial 3 din 5 tipuri, cu cel mai bun scor pe *irelevant* (F1 = 0,345).

Un tipar comun la toate modelele locale pe v4: precizia este 1,000 pe clasele unde obțin F1 nenul (*irelevant* și *halucinatie* la Mistral și Qwen). Aceasta înseamnă că modelele sunt extrem de conservatoare - când prezic un tip specific de neconcordanță, au întotdeauna dreptate, dar ratează majoritatea cazurilor reale. Comportamentul sugerează că exemplele few-shot din v4 ajută modelele locale să învețe tipare de detecție, dar doar pentru cazurile cele mai evidente din fiecare clasă.

5.5.4. Evoluția prompturilor

Evoluția prompturilor pe detecția neconcordanțelor este consistentă cu celelalte sarcini: toate modelele care beneficiază de iterarea prompturilor obțin configurația optimă pe v4. Gemini progresează pe EN de la M-F1 = 0,542 (v1) la 0,595 (v2), 0,679 (v3) și 0,734 (v4), un câștig total de +19,2 puncte. OpenAI o3 urmează un tipar similar: 0,548 (v1), 0,668 (v2), 0,681 (v3), 0,705 (v4), un câștig de +15,7 puncte.

Spre deosebire de celelalte două sarcini, unde modelele locale nu beneficiau de prompturile avansate, pe detecția neconcordanțelor exemplele few-shot din v4 produc câștiguri semnificative: Qwen crește de la M-F1 = 0,183 (v2) la 0,284 (v4), un salt de +10,1 puncte, iar Mistral de la 0,213 la 0,281, un câștig de +6,8 puncte. Analiza per clasă explică mecanismul: pe v2, ambele modele recunoșteau doar 1-2 tipuri de neconcordanțe; pe v4, exemplele few-shot le permit să identifice tipuri suplimentare (*irelevant* și *halucinatie*), chiar dacă doar pe cele mai evidente cazuri.

Aya Expanse 8B fluctuează între M-F1 = 0,062 și 0,291 fără o tendință clară, cu vârful la EN v3 (0,291). Versiunea v4 nu aduce nicio îmbunătățire (EN v4 = 0,162), iar latența crește dramatic la 172 secunde pe EN și 382 secunde pe RO - sugerând că promptul v4 cu exemple few-shot depășește fereastra de context efectivă a modelului cuantizat de 8B. XLM-RoBERTa și RoLLaMA rămân stabile la valori aproape de zero pe toate versiunile.

5.6. Analize comparative transversale

5.6.1. Comparație între taskuri

Tabelul 5.13: Comparație pe cele 3 sarcini - cel mai bun Macro-F1 per model

Model	Tip	Intenție	Status final	Neconcordanțe	Medie
OpenAI o3	API	0,983	0,860	0,705	0,849
Gemini 2.5 Flash	API	0,977	0,834	0,734	0,848
Aya Expanse 8B	Local	0,909	0,620	0,291	0,607
Mistral 7B	Local	0,887	0,548	0,281	0,572
Qwen2.5 3B	Local	0,697	0,493	0,284	0,491
XLM-RoBERTa	Local	0,552	0,224	0,040	0,272
RoLLaMA 2 7B	Local	0,119	0,155	0,175	0,150
RoBERT-base	Local	0,012	-	-	0,012

Tabelul 5.13 sintetizează performanța celor 8 modele pe toate cele 3 sarcini. OpenAI o3 și Gemini 2.5 Flash obțin medii aproape identice (0,849 și 0,848), dar cu profiluri diferite: o3 domină pe extragerea intenției (0,983) și statusul final (0,860), în timp ce Gemini câștigă pe detecția neconcordanțelor (0,734 față de 0,705). Ierarhia modelelor locale este consistentă: Aya Expanse 8B (medie 0,607), Mistral 7B (0,572), Qwen2.5 3B (0,491).

Un tipar clar este că dificultatea crește progresiv de la intenție la status final la neconcordanțe. Toate modelele urmează această ierarhie, dar decalajul se amplifică pentru modelele locale: Aya pierde 6,2 puncte de la intenție la status, dar 32,9 puncte de la status la neconcordanțe. La modelele API, pierderea echivalentă este de doar 12,9 puncte (Gemini, de la 0,834 la 0,705). Taskurile complexe accentuează diferența de capabilitate.

5.6.2. Modele API vs. modele locale

Tabelul 5.14: Modele API vs. locale - Extragerea intenției

Model	Tip	Param. / Cuant.	Acc	M-F1	κ	Latență
OpenAI o3	API	nedeclarat	98,3%	0,983	0,982	4035 ms
Gemini 2.5 Flash	API	nedeclarat	97,8%	0,977	0,976	3499 ms
Aya Expanse 8B	Local	8B / Q4_K_M	91,1%	0,909	0,903	94257 ms
Mistral 7B	Local	7B / NF4	89,4%	0,887	0,885	9306 ms
Qwen2.5 3B	Local	3B / NF4	75,0%	0,697	0,727	6044 ms
XLM-RoBERTa	Local	∼560M	53,9%	0,552	0,497	11503 ms
RoLLaMA 2 7B	Local	7B / NF4	15,6%	0,119	0,084	7438 ms
RoBERT-base	Local	∼125M	1,7%	0,012	−0,073	3075 ms

Tabelul 5.15: Modele API vs. locale - Clasificarea statusului final

Model	Tip	Param. / Cuant.	Acc	M-F1	κ	Latență
OpenAI o3	API	nedeclarat	90,0%	0,860	0,856	8916 ms
Gemini 2.5 Flash	API	nedeclarat	86,1%	0,834	0,804	4769 ms
Aya Expanse 8B	Local	8B / Q4_K_M	72,2%	0,620	0,580	77595 ms
Mistral 7B	Local	7B / NF4	68,9%	0,548	0,537	22721 ms
Qwen2.5 3B	Local	3B / NF4	51,1%	0,493	0,365	13930 ms
XLM-RoBERTa	Local	∼560M	22,8%	0,224	0,069	4819 ms
RoLLaMA 2 7B	Local	7B / NF4	41,1%	0,155	0,003	12077 ms

Tabelul 5.16: Modele API vs. locale - Detecția neconcordanțelor

Model	Tip	Param. / Cuant.	Acc	M-F1	κ	Latență
Gemini 2.5 Flash	API	nedeclarat	89,4%	0,734	0,713	6264 ms
OpenAI o3	API	nedeclarat	90,0%	0,705	0,682	7036 ms
Aya Expanse 8B	Local	8B / Q4_K_M	71,1%	0,291	0,285	113521 ms
Qwen2.5 3B	Local	3B / NF4	80,0%	0,284	0,278	10107 ms
Mistral 7B	Local	7B / NF4	82,2%	0,281	0,196	19720 ms
RoLLaMA 2 7B	Local	7B / NF4	76,1%	0,175	0,033	6074 ms
XLM-RoBERTa	Local	∼560M	8,9%	0,040	−0,007	5922 ms

Tabelele 5.14-5.16 cuantifică diferența de performanță API vs. local pe fiecare task. Este important de notat că toate modelele generative locale au fost cuantizate la 4 biți - Aya Expanse prin Ollama (Q4_K_M), iar Mistral, Qwen și RoLLaMA prin bitsandbytes (NF4) pe Google Colab cu GPU T4. Cuantizarea reduce amprenta de memorie și permite rularea pe hardware accesibil, dar implică o pierdere de precizie care poate afecta performanța, în special pe sarcini complexe. Modelele API rulează la precizie completă pe infrastructura furnizorilor.

Pe extragerea intenției, cel mai bun model local (Aya Expanse 8B, M-F1 = 0,909) se situează la 7,4 puncte sub cel mai bun model API (o3, 0,983) - un decalaj relativ mic care face modelele locale viabile pentru acest task chiar și în condiții de cuantizare. Mistral 7B (0,887) se apropie de Aya la o fracțiune din latența acestuia (9,3 secunde față de 94,3 secunde), oferind un raport performanță/cost operațional semnificativ mai bun.

Pe clasificarea statusului final, decalajul crește la 24 de puncte (o3 0,860 vs. Aya 0,620). Mistral (0,548) și Qwen (0,493) rămân funcționale dar cu limitări pe clasele rare.

Pe detecția neconcordanțelor, decalajul devine dramatic: 44,3 puncte între Gemini (0,734) și cel mai bun model local Aya (0,291). Un rezultat notabil este că Aya (8B cuantizat Q4_K_M), Qwen (3B cuantizat NF4) și Mistral (7B cuantizat NF4) obțin scoruri aproape identice pe acest task (0,291 vs. 0,284 vs. 0,281) - diferență de sub 1 punct M-F1, deși dimensiunile originale variază de la 3 la 8 miliarde de parametri. Aceasta sugerează un plafon de capabilitate al modelelor locale cuantizate pe sarcini complexe, unde nici triplarea numărului de parametri nu aduce câștiguri semnificative.

Din perspectiva unei implementări reale, diferența API vs. local implică un compromis fundamental: modelele API oferă performanță superioară, dar datele conversaționale - potențial sensibile în contextul bancar - sunt transmise pe rețea unui furnizor terț. Modelele locale elimină această problemă de confidențialitate, cu prețul unei performanțe mai slabe și al necesității unui hardware cu GPU. Pe baza rezultatelor, Aya Expanse 8B sau Mistral 7B reprezintă alternative

locale viabile pentru extragerea intenției (M-F1 $> 0,88$) și parțial pentru clasificarea statusului final (M-F1 $> 0,54$), chiar în condiții de cuantizare la 4 biți. Pentru detecția neconcordanțelor, modelele locale cuantizate rămân insuficiente pentru un sistem de producție care necesită clasificarea tipurilor de neconcordanțe.

5.6.3. Analiza latenței și infrastructura hardware

Latența de inferență depinde nu doar de dimensiunea modelului, ci și de infrastructura pe care rulează. Experimentele au folosit trei medii de execuție diferite, ceea ce face comparația directă între modele doar parțial validă:

- **API cloud** (OpenAI o3, Gemini 2.5 Flash): infrastructura furnizorilor, latența include transferul de rețea.
- **Google Colab** (Mistral 7B, Qwen2.5 3B, RoLLaMA 2 7B): GPU NVIDIA T4 (16 GB VRAM), cuantizare NF4 la 4 biți prin bitsandbytes.
- **Laptop local** (Aya Expanse 8B, XLM-RoBERTa, RoBERT-base): Huawei Matebook 14s, Intel Core i5-11300H (4 nuclee / 8 fire, 3,1 GHz), 16 GB RAM, fără GPU dedicat - inferență exclusiv pe CPU prin Ollama (Aya) sau PyTorch (XLM-RoBERTa, RoBERT-base).

Tabelul 5.17: Latența medie per conversație (ms) pe cele 3 sarcini

Model	Infra.	Intenție	Status	Neconcord.	Medie
Gemini 2.5 Flash	API	3 499	4 769	6 264	4 844
OpenAI o3	API	4 035	8 916	7 036	6 662
RoBERT-base	CPU	3 075	-	-	3 075
XLM-RoBERTa	CPU	11 503	4 819	5 922	7 415
Qwen2.5 3B	Colab T4	6 044	13 930	10 107	10 027
Mistral 7B	Colab T4	9 306	22 721	19 720	17 249
RoLLaMA 2 7B	Colab T4	7 438	12 077	6 074	8 530
Aya Expanse 8B	CPU	94 257	77 595	113 521	95 124

Tabelul 5.17 prezintă latența medie per conversație pentru cea mai bună configurație a fiecărui model pe fiecare sarcină.

Modelele API sunt cele mai rapide: Gemini răspunde în medie în 4,8 secunde per conversație pe cele 3 taskuri, iar o3 în 6,7 secunde. Aceste valori includ transferul de rețea și reflectă performanța reală a unui sistem de producție bazat pe API.

Pe Google Colab cu T4, Qwen2.5 3B procesează o conversație în 6-14 secunde (medie 10 secunde), Mistral 7B în 9-23 secunde (medie 17 secunde), iar RoLLaMA 2 7B în 6-12 secunde (medie 8,5 secunde). Latența crește odată cu complexitatea promptului: pe statusul final (promptul cel mai lung, cu definiții detaliate în română), Mistral ajunge la 22,7 secunde. Cu un GPU dedicat mai performant decât T4, aceste valori ar putea scădea semnificativ.

Aya Expanse 8B, rulată pe CPU fără GPU dedicat, este cu un ordin de mărime mai lentă: 77-114 secunde per conversație (medie 95 secunde, adică peste 1,5 minute). Aceasta este o consecință directă a infrastructurii - un model de 8 miliarde de parametri cuantizat Q4_K_M necesită calcul intensiv pe care un procesor i5 cu 4 nuclee nu îl poate efectua eficient. Pe un GPU dedicat, latența Aya ar fi probabil comparabilă cu cea a Mistral pe Colab T4.

Comparația directă a latenței între modele care rulează pe infrastructuri diferite nu este echitabilă. Cu toate acestea, datele permit o concluzie practică: pentru un sistem de monitorizare

care trebuie să proceseze conversații în timp real sau aproape de timp real, modelele API sunt soluția cea mai directă (sub 10 secunde per conversație). Modelele locale pe un GPU de tip T4 sau superior pot procesa o conversație în 6-23 secunde - acceptabil pentru procesare în batch, dar la limită pentru timp real. Inferența pe CPU fără GPU nu este viabilă decât pentru procesare offline.

5.6.4. Impactul limbii promptului

Tabelul 5.18: Impactul limbii promptului - Extragerea intenției (M-F1 pe cea mai bună versiune per limbă)

Model	M-F1 RO	M-F1 EN	Δ	Câștigător
OpenAI o3	0,972	0,983	−1,1	EN
Gemini 2.5 Flash	0,977	0,973	+0,4	≈
Aya Expanse 8B	0,898	0,909	−1,1	EN
XLM-RoBERTa	0,552	0,552	0,0	≈
RoLLaMA 2 7B	0,094	0,119	−2,5	EN

Tabelul 5.19: Impactul limbii promptului - Clasificarea statusului final

Model	M-F1 RO	M-F1 EN	Δ	Câștigător
OpenAI o3	0,860	0,839	+2,1	RO
Gemini 2.5 Flash	0,806	0,834	−2,8	EN
Aya Expanse 8B	0,620	0,620	0,0	≈
XLM-RoBERTa	0,224	0,224	0,0	≈
RoLLaMA 2 7B	0,155	0,131	+2,4	RO

Tabelul 5.20: Impactul limbii promptului - Detecția neconcordanțelor

Model	M-F1 RO	M-F1 EN	Δ	Câștigător
OpenAI o3	0,681	0,705	−2,4	EN
Gemini 2.5 Flash	0,677	0,734	−5,7	EN
Aya Expanse 8B	0,238	0,291	−5,3	EN
XLM-RoBERTa	0,040	0,040	0,0	≈
RoLLaMA 2 7B	0,148	0,175	−2,7	EN

Ipoteza că prompturile în limba română ar produce rezultate mai bune pe conversații românești se confirmă doar selectiv.

Pe extragerea intenției (Tabelul 5.18), engleza câștigă la o3 (−1,1 puncte) și Aya (−1,1), în timp ce Gemini prezintă o diferență neglijabilă. Avantajul pentru limba engleză pe intenție este modest, dar consistent. Mistral și Qwen au fost evaluate exclusiv pe limba engleză pe acest task.

Pe clasificarea statusului final (Tabelul 5.19), apare singurul caz clar în favoarea limbii române: OpenAI o3 obține +2,1 puncte pe RO față de EN. O interpretare posibilă este că terminologia statusului conversațional (*rezolvata*, *partial_rezolvata*, *redirectionata*) este mai

natural mapată în limba română, reducând ambiguitatea în promptul care definește aceste clase. Gemini preferă în schimb engleza (−2,8), iar Aya prezintă performanță identică.

Pe detecția neconcordanțelor (Tabelul 5.20), engleza câștigă consistent la toate modelele testate bilateral. Diferența este cea mai mare la Gemini (−5,7 puncte) și Aya (−5,3). Vocabularul tehnic din taxonomia neconcordanțelor (*halucinatie*, *nealiniat_context*) se aliniază mai bine cu reprezentările interne ale modelelor antrenate preponderent în limba engleză.

Modelele generative multilingve (Gemini, o3, Aya, Mistral, Qwen) gestionează fără dificultate combinații de limbi: prompt în engleză cu etichete în română pe conversații românești, prompt în română cu etichete în engleză, sau exemple few-shot în română inserate într-un prompt englezesc. Diferența de performanță între EN și RO rămâne sub 5 puncte M-F1, confirmând eficiența transferului cross-lingvistic.

Prin contrast, RoLLaMA 2 7B - bazat pe LLaMA 2 și adaptat pentru română - obține performanțe foarte slabe pe toate configurațiile, inclusiv cu etichete în română. Aceasta sugerează că pentru sarcini de clasificare bazate pe prompturi, capacitatea de urmărire a instrucțiunilor structurate și dimensiunea modelului contează mai mult decât adaptarea lingvistică.

5.6.5. Fiabilitatea outputului

Rata de eșec la parsarea JSON este un indicator al capacității modelelor de a respecta instrucțiunile de format structurat.

Modelele API (o3, Gemini) produc sub 1% eșecuri pe toate sarcinile și versiunile - practic fiecare răspuns este un JSON valid. Mistral 7B și Qwen2.5 3B obțin 0% pe intenție și neconcordanțe; pe statusul final, Mistral are 1,1% iar Qwen 3,3% - rate acceptabile pentru producție. Aya Expanse 8B produce 0% pe intenție și statusul final pe toate configurațiile, dar 25% eșecuri pe neconcordanțe EN v2 (45 din 180 conversații), scăzând la 0% pe v4 - exemplele few-shot ajută modelul să respecte formatul.

RoLLaMA 2 7B prezintă rate de eșec foarte variabile: de la 0% pe configurațiile scurte la 75,6% pe statusul final EN v1, unde trei sferturi din răspunsuri sunt text liber nestructurat în loc de JSON. Experimentul cu etichete în română (v3_ro_labels pe intenție) produce 30% eșecuri, confirmând că modelul nu stăpânește generarea de output structurat. XLM-RoBERTa și RoBERT-base nu generează text liber (folosesc mecanisme NLI și clasificare), deci conceptul de eșec la parsare nu se aplică.

Concluzia practică: modelele API și cele locale de dimensiune medie (Mistral, Qwen, Aya) sunt suficient de fiabile pentru un pipeline automatizat de procesare. RoLLaMA necesită mecanisme suplimentare de validare și retry, ceea ce îl face nepotrivit pentru o sarcină în viața reală.

5.7. Discuție și sinteza rezultatelor

5.7.1. Principalele constatări

1. Decalajul API-local crește odată cu gradul de complexitate al sarcinii. Diferența de performanță între cel mai bun model API și cel mai bun model local este de 7,4 puncte M-F1 pe extragerea intenției (o3 0,983 vs. Aya 0,909), 24 de puncte pe clasificarea statusului final (o3 0,860 vs. Aya 0,620) și 44,3 puncte pe detecția neconcordanțelor (Gemini 0,734 vs. Aya 0,291). Creșterea progresivă a decalajului sugerează că sarcinile care necesită raționament conversațional complex - cum ar fi evaluarea coerenței răspunsurilor voicebotului - sunt semnificativ mai sensibile la capacitățile modelului. Pe sarcini simple (extragerea unei

intenții unice), chiar și modele locale cuantizate la 4 biți oferă rezultate aproape de cele API; pe sarcini complexe, decalajul devine prea mare pentru a fi compensat.

2. Rafinarea prompturilor are valoare dovedită, dar condiționată de capacitatea modelului. Modelele API câștigă consistent prin iterarea prompturilor: +19,2 puncte M-F1 pe neconcordanțe (Gemini, de la v1 la v4), +15,9 pe statusul final (o3), +3,1 pe intenție (o3). Modelele locale de dimensiune medie beneficiază și ele, dar selectiv: Aya câștigă +6,7 puncte pe intenție (de la v1 la v4), iar Mistral și Qwen câștigă +6,8 și respectiv +10,1 puncte pe neconcordanțe (de la v2 la v4). Prin contrast, XLM-RoBERTa rămâne stabil indiferent de versiunea promptului (M-F1 = 0,552 pe intenție pe toate cele 8 configurații), iar RoLLaMA oscilează fără o tendință clară. Concluzia este că rafinarea prompturilor nu poate compensa limitele arhitecturale, dar aduce câștiguri semnificative pentru modelele cu o capacitate suficientă de înțelegere a instrucțiunilor.

3. Exemplele (few-shot) ajută chiar și modelele locale pe sarcini complexe. Pe detecția neconcordanțelor, introducerea exemplelor few-shot la v4 produce câștiguri nu doar la modelele API, ci și la Mistral (M-F1 de la 0,213 la 0,281) și Qwen (de la 0,183 la 0,284). Analiza per clasă arată mecanismul: pe v2, ambele modele recunoșteau doar 1-2 tipuri de neconcordanțe; pe v4, identifică tipuri suplimentare (*irelevant*, *halucinatie*), cu precizie perfectă dar recall scăzut - adică învață din exemple să recunoască cazurile cele mai evidente. Excepția este Aya Expanse 8B, care degradează la v4 pe neconcordanțe (M-F1 de la 0,291 la 0,162), probabil din cauza ferestrei de context insuficiente pentru un prompt cu 7 exemple.

4. Dimensiunea modelului nu garantează performanța. Pe detecția neconcordanțelor, Aya Expanse 8B (0,291), Qwen2.5 3B (0,284) și Mistral 7B (0,281) obțin scoruri aproape identice - o diferență de sub 1 punct M-F1, deși Aya are de aproape 3 ori mai mulți parametri decât Qwen. Acest rezultat sugerează existența unui plafon de capabilitate al modelelor locale cuantizate pe sarcini complexe, independent de numărul de parametri. Contrastul cu RoLLaMA 2 7B (0,175 pe același task) arată că arhitectura și calitatea antrenamentului contează mai mult decât dimensiunea.

5. Arhitecturile fără capacitate de generare nu sunt viabile pentru clasificare prin prompting. RoBERT-base ($\kappa = -0,073$ pe intenție, exclus după primul task) și XLM-RoBERTa (M-F1 stagnant la 0,552 pe intenție, 0,040 pe neconcordanțe) demonstrează că mecanismul de inferență prin limbaj natural (NLI) nu reușește să clasifice sarcini care necesită înțelegerea structurii conversaționale. Aceste modele funcționează pe principiul potrivirii semantice între text și etichetă, fără capacitatea de a raționa asupra întregii conversații.

6. RoLLaMA 2 7B - o problemă de urmărire a instrucțiunilor, nu de limbă. Deși este singurul model cu componentă de antrenament specifică limbii române, RoLLaMA 2 7B obține cele mai slabe rezultate dintre modelele generative pe toate cele trei sarcini. Experimentul cu etichete românești pe intenție (*blocare_card*, *deblocare_card* etc.) produce rezultate și mai slabe (M-F1 = 0,015, cu 30% eșecuri de parsare), confirmând că problema nu este limba, ci capacitatea de a respecta instrucțiunile structurate - modelul generează text liber în loc de JSON pe configurațiile mai lungi.

7. Mistral 7B - cel mai bun compromis pentru rulare locală. Mistral 7B obține performanță apropiată de Aya Expanse 8B pe intenție (0,887 vs. 0,909) și statusul final (0,548 vs. 0,620), la o latență de aproape 10 ori mai mică (9 secunde vs. 94 secunde pe intenție, pe infrastructuri diferite). Pe neconcordanțe, Mistral (0,281) este practic la egalitate cu Aya (0,291). Rata de eșec la parsare este sub 1,1% pe toate sarcinile. Aceste caracteristici fac din Mistral opțiunea cea mai practică pentru procesarea în volum mare a conversațiilor în context local.

5.7.2. Limitări ale evaluării

Mai multe limitări trebuie luate în considerare la interpretarea rezultatelor.

Dimensiunea datasetului. Cele 180 de conversații reprezintă un dataset de dimensiune mică, suficient pentru comparații relative între modele, dar insuficient pentru estimări statistice robuste ale intervalelor de încredere. Clasele cu 5-10 exemple (tipurile de neconcordanțe) sunt deosebit de sensibile la variația de eșantionare - o singură conversație clasificată diferit poate modifica F1 cu 10-20 de puncte.

Condiții de inferență neuniforme. Modelele API rulează pe infrastructura cloud a furnizorilor; Aya Expanse rulează local pe CPU (Intel i5-11300H, fără GPU dedicat); Mistral, Qwen și RoLLaMA rulează pe Google Colab cu GPU T4. Comparațiile de latență reflectă condițiile reale de utilizare, dar nu permit izolarea performanței modelului de performanța hardwareului.

Cuantizarea. Toate modelele generative locale sunt evaluate în formă cuantizată la 4 biți (Q4_K_M pentru Aya, NF4 pentru restul). Impactul cuantizării asupra calității clasificării nu a fost izolat experimental - rezultatele la precizie completă ar putea fi superioare, în special pe sarcinile complexe unde diferențe mici de precizie numerică se pot propaga.

Domeniu specific. Rezultatele sunt obținute pe conversații bancare simulate în limba română, generate cu o structură predefinită. Generalizarea la alte domenii, la conversații reale cu zgomot de transcriere sau la alte limbi nu este garantată.

Parametri de generare fixați. Temperatura și alți hiperparametri de generare au fost fixați la valorile implicite (decodare greedy unde era posibil). Optimizarea lor per model și per sarcină ar putea îmbunătăți rezultatele, în special la modelele locale.

5.7.3. Implicații practice

Pe baza rezultatelor experimentale se pot formula recomandări concrete pentru diferite scenarii de implementare a unui sistem de monitorizare a voicebotilor.

Scenariul cu acces la API. OpenAI o3 și Gemini 2.5 Flash oferă performanță substanțială pe toate cele trei sarcini (medie M-F1 = 0,849 și respectiv 0,848). Configurațiile optime sunt EN v4 pe intenție și neconcordanțe, RO v4 pe statusul final pentru o3, și EN v4 pe toate sarcinile pentru Gemini. Un pipeline complet care rulează cele 3 sarcini secvențial procesează o conversație în aproximativ 15-20 de secunde prin API. Costul per apel și politicile de confidențialitate ale furnizorilor trebuie evaluate în raport cu valoarea practică a performanței superioare.

Scenariul care are constrângeri de confidențialitate. Aya Expanse 8B sau Mistral 7B oferă performanță locală viabilă pe extragerea intenției (M-F1 $>$ 0,88) și parțial pe statusul final (M-F1 $>$ 0,54). Mistral este preferabil când viteza de procesare contează (9 secunde vs. 94 secunde per conversație pe intenție), iar Aya când performanța maximă este prioritară și latența este acceptabilă. Pe detecția neconcordanțelor, toate modelele locale obțin M-F1 $\approx$ 0,28 - suficient pentru a semnala prezența unei probleme (detecție binară), dar insuficient pentru clasificarea fiabilă a tipului de neconcordanță într-un context de producție.

Scenariul cu resurse limitate. Qwen2.5 3B cu o cuantizare NF4 oferă M-F1 = 0,697 pe intenție și 0,493 pe statusul final, cu o latență de 6-14 secunde pe un GPU T4 și consum redus de memorie ($\approx$2 GB VRAM). Performanța este inferioară modelelor mai mari, dar reprezintă o opțiune accesibilă când hardware-ul dedicat nu este disponibil. Pe neconcordanțe, Qwen (0,284) obține un scor comparabil cu Mistral (0,281) și Aya (0,291), confirmând plafonul de capabilitate observat la modelele locale.

6. PIPELINE DEMONSTRATIV PENTRU EVALUAREA INTERACȚIUNILOR CU VOICEBOȚI

6.1. Motivație și obiective

Experimentele descrise în capitolele anterioare au fost realizate pe un set static de 180 de conversații bancare, iar rezultatele au fost salvate în fișiere JSON și rapoarte text. Deși aceste rezultate sunt complete din punct de vedere statistic, ele rămân dificil de consultat fără un context operațional: un evaluator care vrea să testeze o conversație nouă trebuie să editeze manual scripturile, să cunoască structura directoarelor și să ruleze separat fiecare pas al pipeline-ului.

Pentru a face trecerea de la experimentul offline la o utilizare interactivă, a fost implementat un pipeline demonstrativ care reunește aceleași date, prompturi și configurații de evaluare, dar le expune în trei forme complementare: o interfață de linie de comandă pentru rulări programatice, o pagină web pentru demonstrații live și un dashboard Streamlit pentru explorarea rezultatelor deja existente.

Obiectivul principal este ca o conversație nouă - introdusă manual prin interfața de chat, încărcată dintr-un fișier JSON sau generată în sesiune live cu un voicebot demonstrativ - să poată fi evaluată pe cele trei sarcini definite în lucrare: extragerea intenției, clasificarea statusului final și detecția neconcordanțelor. Pentru fiecare sarcină, pipeline-ul produce o predicție structurată, iar în paralel face vizibile deciziile din spatele evaluării: ce model este folosit, ce șablon de prompt a fost randat, câte exemple au fost încărcate și ce furnizor a efectuat inferența.

Un obiectiv secundar este reproductibilitatea. Toate configurațiile de modele, recomandările implicite, fișierele de prompturi și exemplele few-shot sunt centralizate în pachetul `src/evaluation_pipeline`, astfel încât o evaluare rulată din interfața web, din terminal sau dintr-un notebook folosește aceleași surse de date.

6.2. Arhitectura generală

Pipeline-ul este organizat modular, astfel încât logica de evaluare să fie separată de fiecare interfață grafică sau de linie de comandă. La nivel logic, fluxul de evaluare a unei conversații parcurge cinci pași:

a) preluarea conversației, fie din sesiunea live, fie dintr-un text introdus manual, fie dintr-un identificator de conversație din dataset;
b) normalizarea transcriptului într-o listă de replici cu rolurile `user` și `assistant`;
c) selectarea configurației de evaluare - model, limbă, versiune de prompt - pe baza recomandărilor experimentale sau a alegerii utilizatorului;
d) randarea promptului Jinja2 și apelarea providerului corespunzător;
e) parsarea răspunsului JSON și afișarea rezultatului.

Această separare face posibilă rularea aceluiași mecanism de evaluare din terminal (`cli.py`), din serverul web (`web_demo_server.py`) sau dintr-un notebook Jupyter, fără a duplica logica experimentală.

Din punct de vedere al structurii de directoare, codul este distribuit în trei zone:

- `src/evaluation_pipeline/` — nucleul de evaluare, compus din modulele `config.py`, `prompting.py`, `providers.py`, `local_eval.py` și `cli.py`;
- `demo_voicebot_web/` — serverul HTTP, logica voicebotului demonstrativ, baza de cunoștințe conversațională și integrarea cu serviciile de vorbire;
- `streamlit_results_dashboard.py` — dashboard-ul pentru explorarea rezultatelor experimentale.

Cele trei zone folosesc aceleași fișiere din `configs/` (definiții de clase, exemple few-shot) și din `prompts/` (template-uri Jinja2 versionizate), ceea ce asigură consistența între evaluarea experimentală și demonstrația interactivă.

6.3. Nucleul de evaluare

Pachetul `src/evaluation_pipeline` conține logica partajată de toate interfețele și este organizat în cinci module cu responsabilități distincte.

Modulul `config.py` centralizează toate configurațiile. Acesta definește un registru de modele, în care fiecare model are asociate o etichetă afișabilă, un furnizor, un nume de rulare și o notă explicativă. Furnizorii suportați sunt `openai`, `gemini`, `ollama` și `encoder`. Tot aici sunt definite recomandările implicite pentru fiecare sarcină, derivate din rezultatele experimentale: OpenAI o3 cu prompt în limba engleză v4 pentru extragerea intenției, OpenAI o3 cu prompt în limba română v4 pentru statusul final și Gemini 2.5 Flash cu prompt în limba română v4 pentru neconcordanțe. Modulul conține și harta completă a fișierelor de prompturi și a exemplelor few-shot, astfel încât orice combinație validă de sarcină, limbă și versiune poate fi rezolvată automat.

Modulul `prompting.py` se ocupă de randarea prompturilor. Funcția principală, `render_prompt`, primește sarcina, conversația, limba și versiunea de prompt, completează șablonul Jinja2 corespunzător și returnează textul final al promptului. Pentru versiunile v4, funcția încarcă automat exemplele few-shot din fișierele JSON aferente. De exemplu, fișierul de exemple pentru neconcordanțe conține câte un exemplu pentru fiecare tip de eroare și două exemple negative, menite să prevină supraclasificarea.

Modulul `providers.py` direcționează promptul randat către furnizorul corect, folosind SDK-ul oficial al fiecăruia: `openai.OpenAI` pentru OpenAI, `google.genai.Client` pentru Gemini și `ollama.chat` pentru modelele locale. Răspunsul brut este parsat automat pentru a extrage primul obiect JSON valid. Pentru modelele Ollama, modulul încearcă mai multe aliasuri ale numelui de rulare (de exemplu, `mistral:7b-instruct`, `mistral:7b`, `mistral`) și, dacă niciun model nu este găsit, semnalează eroarea alături de comanda de instalare.

Modulul `local_eval.py` implementează un evaluator bazat pe reguli, util pentru demonstrații rapide fără chei API sau modele descărcate. Acesta detectează intenția prin potrivirea cuvintelor cheie, clasifică statusul final prin formulări asociate rezolvării sau eșecului și identifică neconcordanțe simple. Evaluatorul local nu înlocuiește modelele lingvistice și nu este raportat ca rezultat experimental. Este implementat ca o metoda de urgență în cazul nefuncționării evaluării obișnuite.

Modulul `cli.py` expune o interfață de linie de comandă care permite evaluarea unei conversații direct din terminal, cu opțiuni pentru selectarea sarcinii, a modelului, a limbii și a

furnizorului. Conversația poate fi specificată printr-un identificator din dataset, printr-un fișier JSON sau prin text scris direct în terminal. Opțiunea `--compare-models` permite compararea mai multor modele pe aceeași conversație într-o singură comandă.

6.4. Sistemul de prompturi

Prompturile sunt organizate în directorul `prompts/`, cu subdirectoare separate pentru fiecare sarcină: `intent_extraction`, `final_status` și `incongruities`. Fiecare sarcină are template-uri Jinja2 versionizate (de la v1 la v4) în ambele limbi (română și engleză). Numele fișierelor urmează convenția `{limbă}_{sarcină}_{versiune}.jinja`, iar pentru versiunile v4 se folosește prefixul `few_shot` în locul lui `zero_shot`.

Evoluția de la v1 la v4 este descrisă detaliat în Capitolul **??**. Pe scurt, v1 conține definiții narative ale claselor într-un format zero-shot; v2 adaugă reguli mai stricte de format JSON; v3 introduce criterii operaționale și reguli anti-eroare; v4 adaugă exemple few-shot calibrate peste structura v3. Toate versiunile cer modelului să returneze un obiect JSON cu câmpuri specifice: `intent`, `confidence` și `reasoning` pentru extragerea intenției; `final_status`, `confidence` și `reasoning` pentru statusul final; `has_incongruity`, `incongruity_type`, `confidence` și `reasoning` pentru neconcordanțe. Această structură comună permite parsarea uniformă și afișarea comparabilă în toate interfețele.

Exemplele few-shot sunt stocate separat, în directorul `configs/`, sub formă de fișiere JSON. De exemplu, `few_shot_examples_incongruities.json` conține un câmp `examples` cu o listă de obiecte, fiecare având conversația, clasificarea corectă și raționamentul. Încărcarea lor este controlată de funcția `render_prompt`: doar versiunile v4 injectează exemplele în template, prin variabila `few_shot_examples` sau `examples` din contextul Jinja.

Separarea template-urilor de definiții și de exemple face posibilă modificarea independentă a oricărui element — de exemplu, adăugarea unui nou exemplu few-shot — fără a rescrie textul promptului. Totodată, fișierele de definiții ale claselor (`intent_definitions.json`, `final_status_definitions.json`, `incongruities_definitions.json`) sunt sursa unică de adevăr pentru numele și descrierile claselor, atât în prompturi, cât și în scripturile de evaluare.

6.5. Interfața web și voicebotul demonstrativ

Interfața web poate fi pornită prin comanda `python demo_voicebot_web/web_demo_server.py` și este accesibilă în browser la adresa `http://127.0.0.1:8787`. Serverul servește o pagină formată dintr-un fișier HTML, un fișier JavaScript și un fișier CSS, și expune un set de endpoint-uri API pentru conversație, evaluare și sinteză vocală.

Pagina are trei moduri de funcționare, selectabile din bara de navigare din panoul din stânga: *Live*, *Evaluator* și *Rezultate*.

Modul *Live*, ilustrat în Figura 6.1, permite purtarea unei conversații cu voicebotul demonstrativ Bănuțel. Panoul din stânga conține configurația sesiunii: selectarea vocii TTS (prin serviciul Zevo), starea cache-ului audio, disponibilitatea furnizorilor (OpenAI, Gemini, Ollama) și selectoarele de model pentru fiecare sarcină, cu indicarea configurației recomandate. Modul de rulare poate fi comutat între *Local, fără API* (evaluare euristică) și modul real, cu apel către furnizorul corespunzător. Panoul central afișează conversația, care poate fi textuală sau vocală: în modul vocal, audio-ul este transcris prin serviciul Zevo STT, iar răspunsul botului poate fi redat prin Zevo TTS. Panoul din dreapta afișează recomandările de model pentru fiecare sarcină

și baza de cunoștințe activă. Textul generat este păstrat în transcript, astfel încât analiza finală se aplică aceleiași structuri de conversație, indiferent dacă input-ul a fost tastat sau rostit.

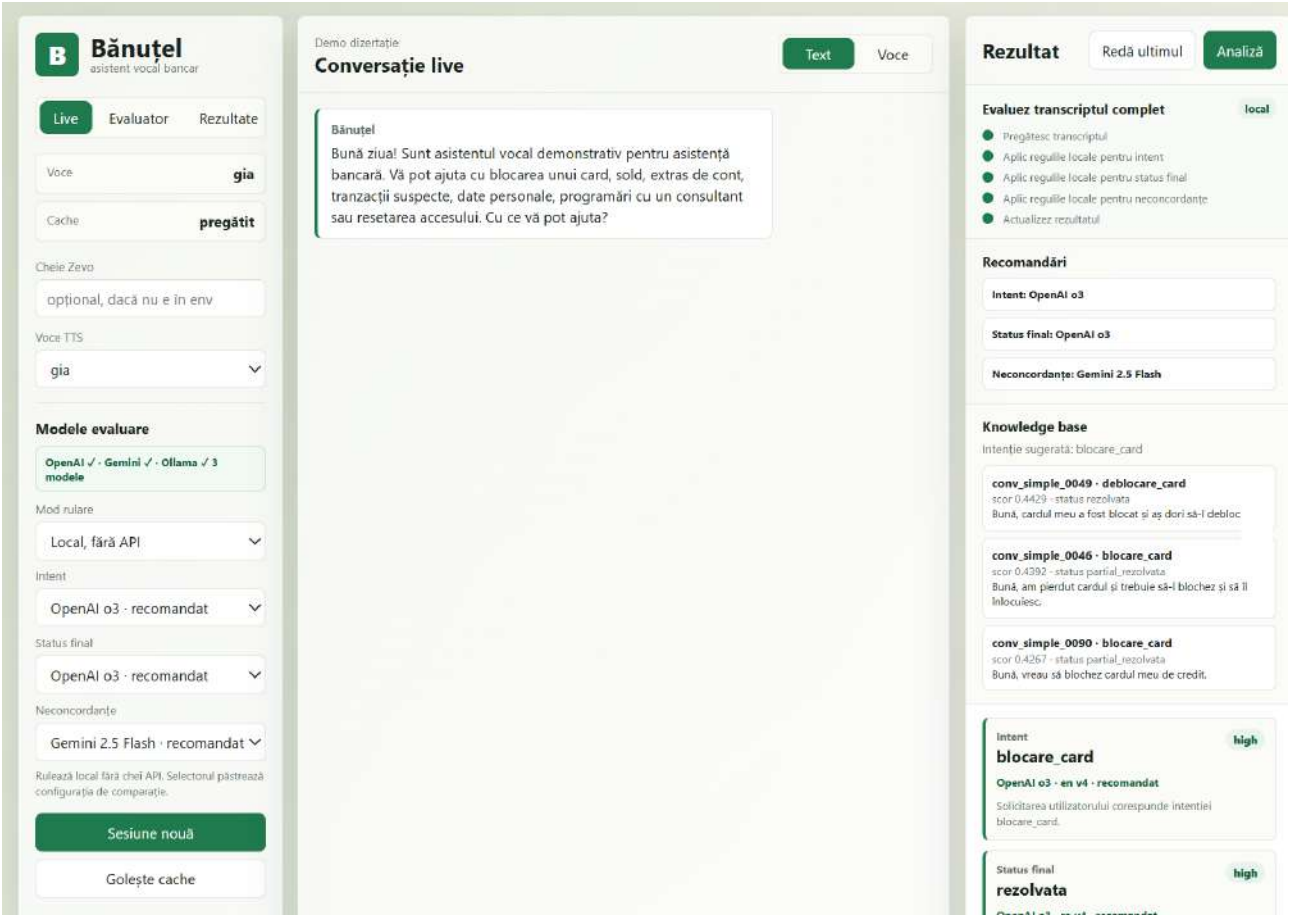


Figura 6.1: Interfața web în modul *Live*.

Modul *Evaluator*, ilustrat în Figura 6.2, permite introducerea unei conversații complete în format text. Un exemplu de conversație introdusă în câmpul de text:

```
USER: Vreau să-mi blochez cardul de credit.
ASSISTANT: Pentru siguranță, spuneți ultimele 4 cifre.
USER: Ultimele cifre sunt 4321.
ASSISTANT: Cardul de debit terminat în 4321 a fost blocat.
```

După introducerea conversației, pipeline-ul o parsează și rulează cele trei sarcini de evaluare. În panoul din dreapta, progresul este afișat pas cu pas. Baza de cunoștințe identifică automat intenția probabilă (`blocare_card`) și afișează cele mai similare conversații din dataset. Rezultatele apar sub formă de carduri care arată eticheta prezisă, nivelul de încredere, modelul folosit și raționamentul. Modul *Rezultate* deschide direct dashboard-ul Streamlit descris în secțiunea următoare.

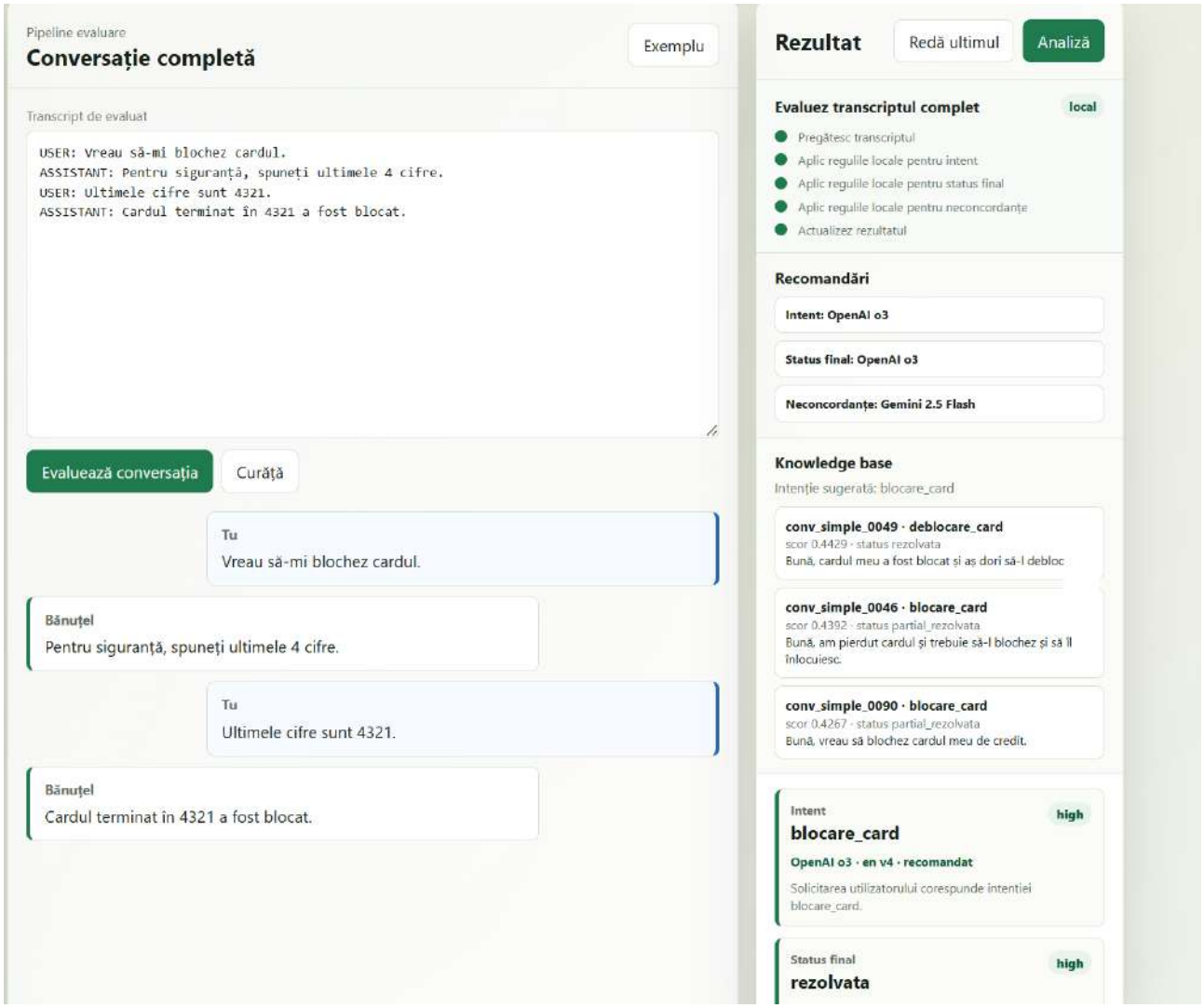


Figura 6.2: Interfața web în modul *Evaluator*

6.5.1. Mecanismul RAG dataset-first

În modul *Live*, botul Bănuțel trebuie să genereze răspunsuri plauzibile în timp real. Pentru aceasta, sistemul folosește un mecanism de tip RAG (Retrieval-Augmented Generation), implementat în clasa `ConversationKnowledgeBase` din `conversation_kb.py`.

Rolul bazei de cunoștințe este de a ancora răspunsurile modelului în conversațiile reale din setul de date. Fără această componentă, modelul conversațional (Gemini 2.5 Flash) ar răspunde exclusiv din cunoștințele sale generale, fără nicio legătură cu scenariile bancare studiate în lucrare. Cu baza de cunoștințe, modelul primește ca și context exemple concrete de conversații similare și își formulează răspunsul pe baza lor.

Fluxul funcționează astfel: la fiecare mesaj nou al utilizatorului, sistemul caută în setul de date de 180 de conversații pe cele mai similare. Fiecare conversație din setul de date este reprezentată ca un vector de termeni normalizați, iar asemănarea cu mesajul curent este calculată printr-o măsură de tip cosinus. Rezultatele sunt apoi reordonate în funcție de intenția estimată - de exemplu, dacă primele replici sugerează o blocare de card, conversațiile care aparțin aceluiași tip de solicitare primesc un scor mai mare. Cele mai relevante exemple, împreună cu transcriptul curent al sesiunii, sunt transmise modelului pentru a genera răspunsul botului. Conversațiile

găsite și scorurile lor de similaritate sunt vizibile în panoul din dreapta al interfeței, în secțiunea *Knowledge base*.

Această abordare a fost aleasă pentru a echilibra două extreme. Pe de o parte, un voicebot cu răspunsuri fixe ar funcționa doar pentru câteva scenarii și nu ar reflecta flexibilitatea modelelor lingvistice. Pe de altă parte, un model folosit fără nicio legătură cu setul de date poate genera răspunsuri care nu corespund conversațiilor evaluate în disertație. Strategia implementată este *dataset-first*: modelul poate formula liber, dar pornind de la exemplele cele mai similare din setul de date.

Evaluarea propriu-zisă a conversației are loc separat, la apăsarea butonului *Analiză*. În acel moment, transcriptul complet al sesiunii este transmis prin pipeline-ul de evaluare descris în secțiunile anterioare, unde sunt folosite prompturile versionizate (inclusiv v4 cu exemple) pentru fiecare dintre cele trei sarcini.

6.5.2. Integrarea vocală prin Zevo STT și TTS

Pe lângă modul text, interfața web oferă și posibilitatea de a interacționa vocal cu botul Bănuțel, valorificând infrastructura de recunoaștere automată a vorbirii în cadrul laboratorului de specialitate [34]. Comutarea între cele două moduri se face din butoanele *Text* și *Voce* din colțul din dreapta sus al panoului central, vizibile în Figura 6.3.

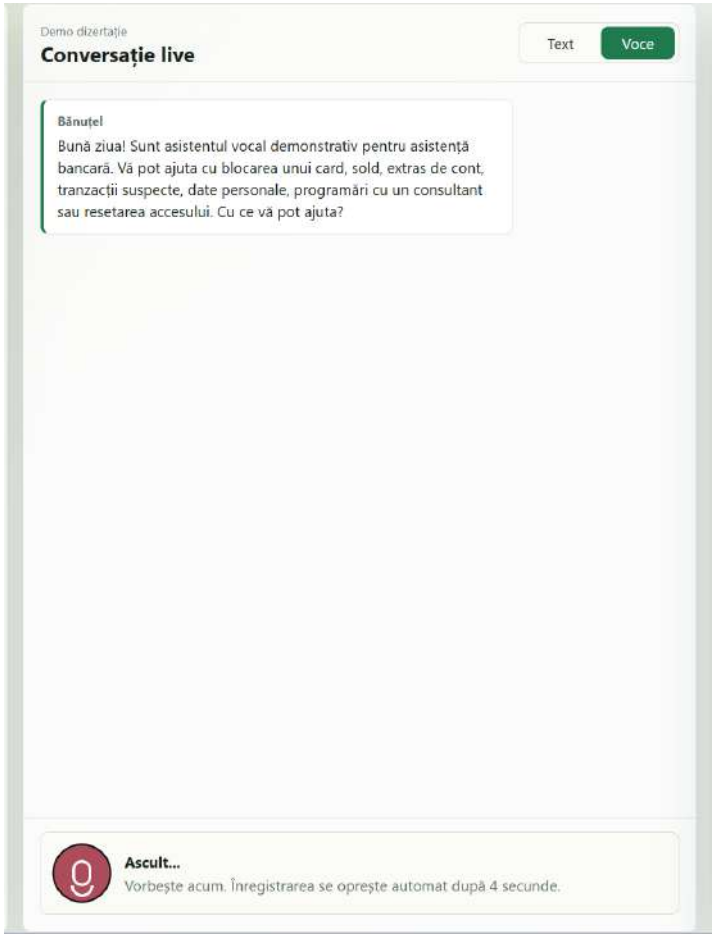


Figura 6.3: Modul vocal al interfeței web.

Fluxul vocal parcurge trei etape. Mai întâi, vocea utilizatorului este captată prin microfonul browserului și trimisă ca fișier audio codificat în format base64 către server. Serverul transmite

audio-ul către serviciul Zevo STT (Speech-to-Text), un serviciu de recunoaștere vocală cu suport pentru limba română, care returnează transcriptul textual al mesajului. Componenta STT este implementată în `zevo_stt.py` și comunică cu serviciul Zevo printr-o conexiune WebSocket.

După obținerea transcriptului, mesajul utilizatorului este procesat la fel ca în modul text: baza de cunoștințe caută conversații similare, iar modelul conversațional generează răspunsul botului. Răspunsul generat este apoi transmis către serviciul Zevo TTS (Text-to-Speech), implementat în `zevo_tts.py`, care sintetizează textul într-un fișier audio redat automat în browser. Vocea TTS poate fi configurată din panoul din stânga al interfeței (opțiunea *Voce TTS*), iar fișierele audio generate sunt păstrate într-un cache local.

Din punctul de vedere al evaluării, modul vocal nu introduce diferențe față de modul text. Transcrierea obținută prin STT este stocată în aceeași structură de replici ca și textul tastat manual, astfel încât analiza finală (la apăsarea butonului *Analiză*) se aplică identic, indiferent de modul de introducere a mesajelor.

6.6. Integrarea furnizorilor de modele și tratarea erorilor

Pipeline-ul suportă două moduri de rulare. Modul `local` folosește evaluatorul euristic implementat în `local_eval.py` și este util pentru demonstrații rapide, fără chei API și fără modele descărcate. Modul `auto` trimite prompturile către furnizorul corespunzător modelului selectat - OpenAI, Gemini sau Ollama.

Pentru modelele API (OpenAI și Gemini), singura condiție este prezența cheilor de autentificare în fișierul `.env`: `OPENAI_API_KEY` pentru OpenAI și `GOOGLE_API_KEY` pentru Gemini. Serverul web citește aceste variabile la pornire și afișează în interfață dacă au fost detectate. Pentru modelele locale, pipeline-ul folosește Ollama. Deoarece modelele Ollama trebuie descărcate manual, modulul `providers.py` verifică lista de modele instalate și încearcă mai multe variante ale numelui (de exemplu, `mistral:7b-instruct`, `mistral:7b`, `mistral`) înainte de a semnala eroarea. Interfața web afișează pentru fiecare model Ollama lipsă comanda de instalare corespunzătoare.

Tratarea erorilor este importantă pentru continuitatea demonstrației. Într-un context de prezentare, este preferabil ca analiza să producă un rezultat local explicabil, nu să se oprească din cauza unui model lipsă. Dacă un model Ollama nu este disponibil, serverul web afișează un avertisment și folosește evaluatorul local ca soluție de rezervă.

6.7. Dashboard-ul Streamlit

A doua componentă vizuală a pipeline-ului este dashboard-ul Streamlit, implementat în fișierul `streamlit_results_dashboard.py`. Spre deosebire de pagina HTML, care evaluează conversații noi, dashboard-ul explorează rezultatele experimentale deja salvate în repository. Acesta citește fișierele JSON din directoarele `outputs_final_status` și `outputs_incongruities`, agregă metricile și le prezintă interactiv. Dashboard-ul este organizat în cinci secțiuni, accesibile din meniul lateral.

Secțiunea *Metodologie* prezintă spațiul experimental: numărul de conversații, cele trei sarcini, distribuția celor 128 de experimente pe tipuri de modele (API, local, encoder) și observații despre configurațiile testate pentru fiecare sarcină.

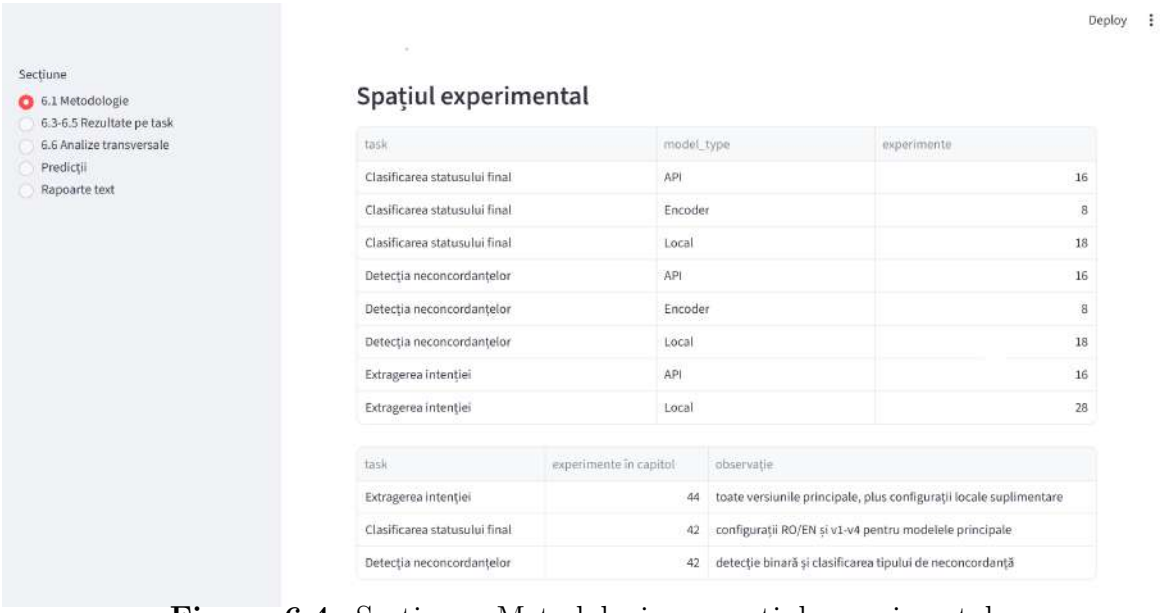


Figura 6.4: Secțiunea Metodologie — spațiul experimental.

Secțiunea *Rezultate pe task* (6.3–6.5) conține, pentru fiecare sarcină, configurațiile cu cele mai bune scoruri, evoluția M-F1 pe versiunile de prompt (v1–v4), comparațiile între modele API și locale și impactul limbii. Filtrele interactive permit restrângerea rezultatelor după model, limbă sau versiune de prompt.

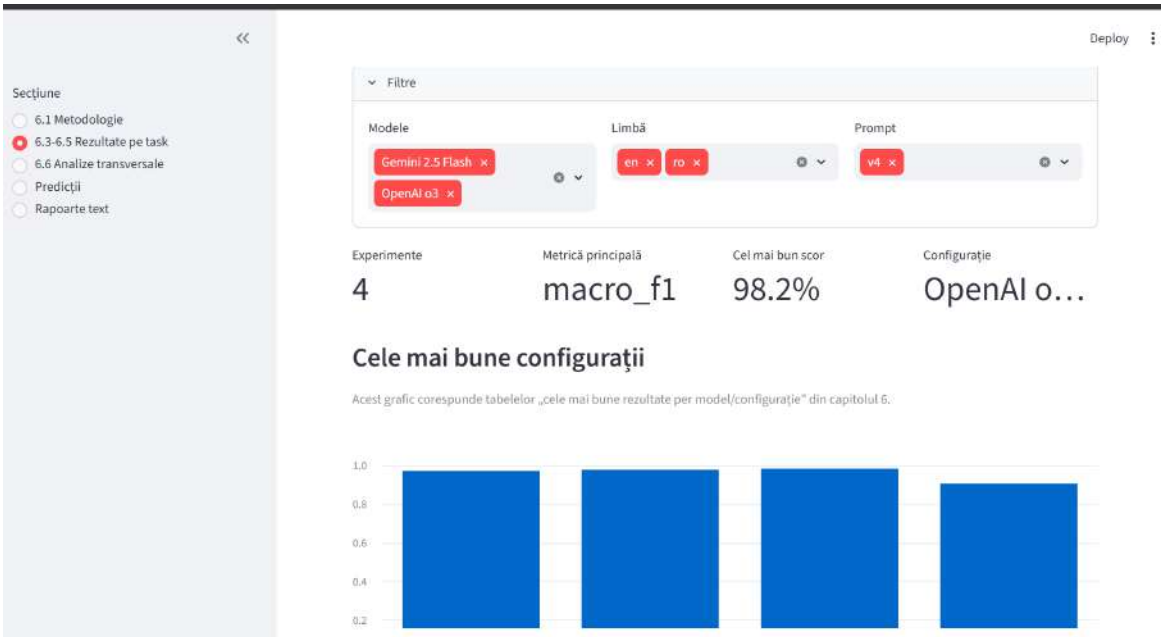


Figura 6.5: Secțiunea Rezultate pe task — filtrare și cele mai bune configurații.

Secțiunea *Analize transversale* (6.6) reunește comparațiile între sarcini: decalajul API–local, impactul rafinării prompturilor, latența pe model și configurație, și rata de eșec la parsare.

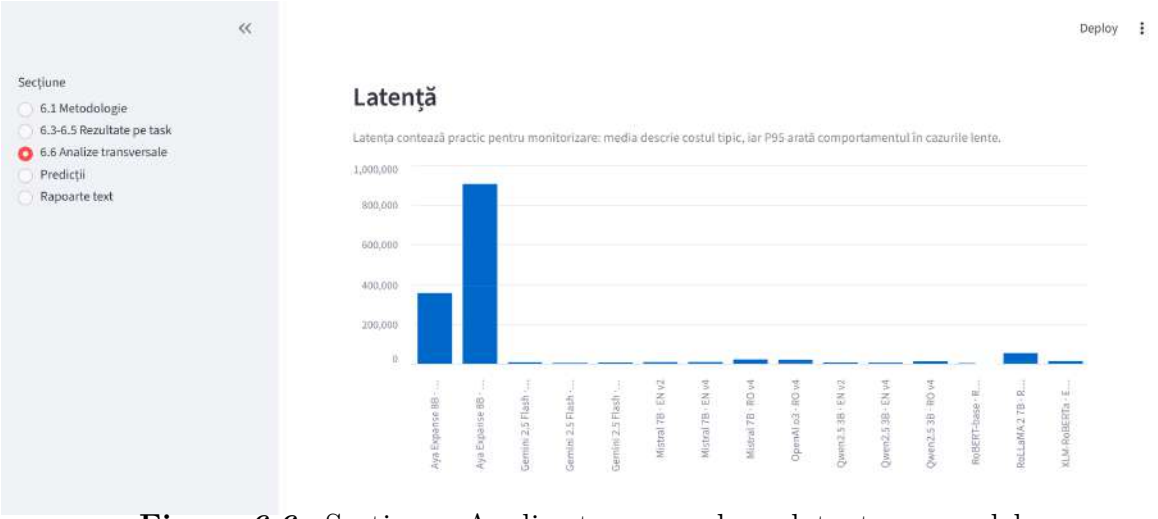


Figura 6.6: Secțiunea Analize transversale — latența pe model.

Secțiunea *Predicții* permite explorarea predicțiilor individuale din fișierele JSON experimentale. Utilizatorul selectează sarcina și fișierul experimental, iar un tabel afișează pentru fiecare conversație statusul așteptat, statusul prezis, modelul, limba, versiunea de prompt și nivelul de încredere. Opțional, pot fi afișate doar erorile sau eșecurile de parsare.

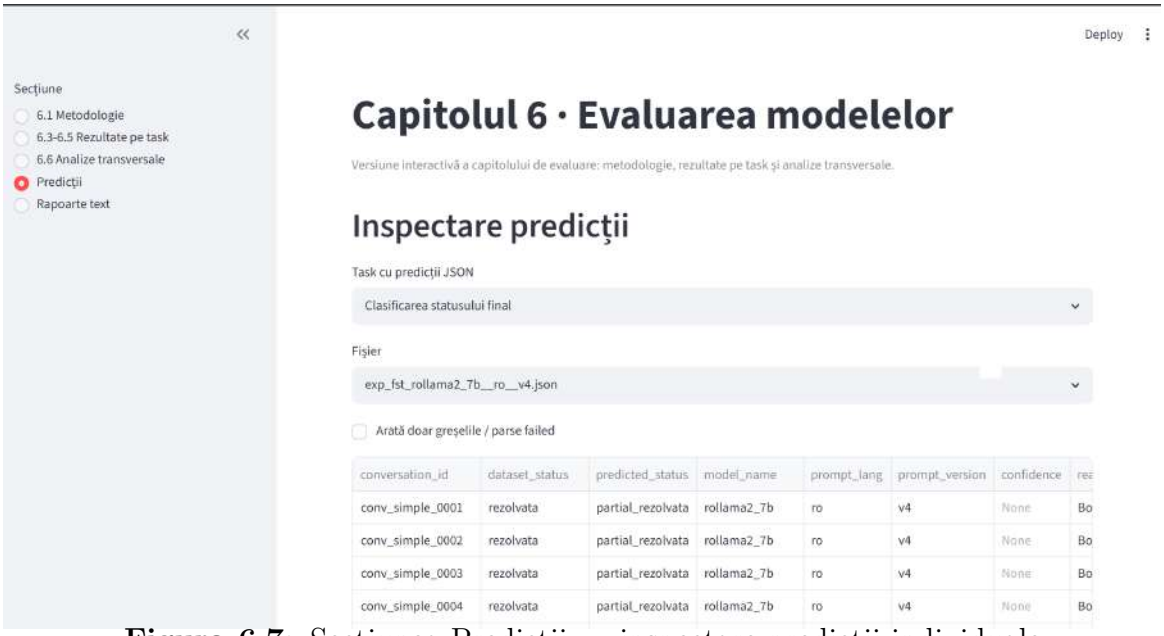


Figura 6.7: Secțiunea Predicții — inspectare predicții individuale.

Secțiunea *Rapoarte text* afișează rapoartele generate automat de scriptul de evaluare, în format text, pentru fiecare sarcină și pentru analiza transversală.

Streamlit a fost ales pentru abilitatea sa de a construi instrumente analitice interactive fără a implementa manual componentele de filtrare și vizualizare. Dashboard-ul este separat de serverul HTML: pagina web demonstrează utilizarea pipeline-ului pe conversații noi, iar Streamlit contextualizează rezultatele deja obținute.

6.8. Rulare din terminal și reproductibilitate

Pe lângă interfețele grafice, pipeline-ul poate fi rulat integral din terminal. Această variantă este importantă pentru reproductibilitate, deoarece permite evaluarea unei conversații fără

browser și fără componente audio. Câteva exemple de utilizare:

```
# Evaluare pe conversația conv_simple_0001, toate sarcinile, mod local
python -m src.evaluation_pipeline.cli \
  --conversation-id conv_simple_0001 --task all

# Evaluare cu modelul recomandat, provider automat
python -m src.evaluation_pipeline.cli \
  --conversation-id conv_simple_0001 --task all --provider auto

# Comparație între mai multe modele pe sarcina de intenție
python -m src.evaluation_pipeline.cli \
  --conversation-id conv_simple_0001 --task intent \
  --compare-models openai_o3,gemini_2.5_flash,mistral_7b

# Conversație introdusă inline
python -m src.evaluation_pipeline.cli --task all \
  --text "USER: Vreau sa-mi blochez cardul.
ASSISTANT: Pentru siguranta, spuneti ultimele 4 cifre.
USER: Revin mai tarziu."
```

Rularea din terminal are și un rol de verificare tehnică. Ea confirmă că prompturile, configurațiile și providerii funcționează independent de interfața web. Dacă apare o problemă în pagina HTML, se poate izola rapid sursa erorii: interfața grafică, serverul local, randarea promptului sau providerul modelului.

Structura modulară contribuie la reproductibilitate în trei moduri. În primul rând, sursele de date, definițiile claselor și exemplele few-shot sunt păstrate în fișiere versionizate în repository. În al doilea rând, registrul de modele și recomandările implicite sunt centralizate într-un singur modul (`config.py`), ceea ce reduce riscul ca interfața web să folosească o configurație diferită de cea raportată. În al treilea rând, evaluarea din terminal produce un JSON complet, care păstrează metadate despre model, limbă, versiune de prompt, fișierul de template folosit și numărul de exemple few-shot încărcate.

6.9. Concluzii

Pipeline-ul demonstrativ extinde partea experimentală a lucrării printr-un instrument care poate fi folosit atât pentru prezentare, cât și pentru verificare. El leagă setul de date, prompturile, modelele și rezultatele într-un flux unic: conversația este introdusă sau generată live, este evaluată pe cele trei sarcini, iar rezultatul este afișat într-o formă interpretabilă.

Contribuția practică a pipeline-ului constă în faptul că face vizibile deciziile tehnice din spatele evaluării. Utilizatorul poate vedea ce model este recomandat, ce prompt este folosit, ce exemple sunt încărcate pentru variantele few-shot și ce se întâmplă atunci când un provider nu este disponibil. Prin mecanismul RAG *dataset-first*, demo-ul live rămâne ancorat în conversațiile studiate, evitând răspunsurile arbitrare ale unui model neconstrâns. Prin dashboard-ul Streamlit, rezultatele agregate devin explorabile și pot fi corelate cu metodologia experimentală.

Astfel, sistemul rezultat nu este doar o interfață de prezentare, ci o punte între experimentul offline și utilizarea interactivă a modelelor lingvistice mari în monitorizarea conversațiilor dintre utilizatori și voiceboți bancari.

7. CONCLUZII GENERALE

7.1. Privire de ansamblu

Lucrarea de față a pornit de la o întrebare practică: în ce măsură pot modelele lingvistice mari să automatizeze monitorizarea conversațiilor purtate de voiceboții bancari, o activitate care în prezent rămâne predominant manuală și care nu poate scala la volumele zilnice ale unei bănci comerciale. Pentru a răspunde acestei întrebări, am formulat trei obiective complementare, ordonate de la componenta de cercetare la cea de implementare: evaluarea comparativă a opt modele lingvistice pe trei sarcini de clasificare, analiza impactului designului prompturilor și construirea unui pipeline funcțional care să integreze rezultatele evaluării într-un sistem demonstrabil.

Cele trei obiective au fost atinse integral. Componenta experimentală a produs 128 de configurații evaluate sistematic pe un dataset de 180 de conversații bancare în limba română, componenta de inginerie a prompturilor a documentat patru iterații succesive în două limbi, iar componenta de implementare a livrat un pipeline complet, cu trei interfețe complementare. Figura 7.1 sintetizează relația dintre componente.

Rezultatele confirmă că automatizarea monitorizării este fezabilă pentru cel puțin două dintre cele trei sarcini analizate. Modelele API de frontieră - OpenAI o3 și Gemini 2.5 Flash - ating performanțe ridicate pe extragerea intenției (M-F1 $\geq$ 0,977) și pe clasificarea statusului final (M-F1 $\geq$ 0,834), suficiente pentru utilizare în producție cu supervizare umană redusă. Detecția neconcordanțelor rămâne sarcina cea mai dificilă, cu cel mai bun rezultat de M-F1 = 0,734, reflectând complexitatea raționamentului semantic necesar pentru identificarea erorilor subtile ale voicebotului. Modelele locale de talie medie nu ating praguri operaționale pe niciuna dintre sarcini în configurație zero-shot, însă contribuie la conturarea unui mod de eșec bifurcat - supra-detecție la modelele API și sub-detecție la modelele locale - cu implicații directe pentru proiectarea unui sistem de producție în cascadă.

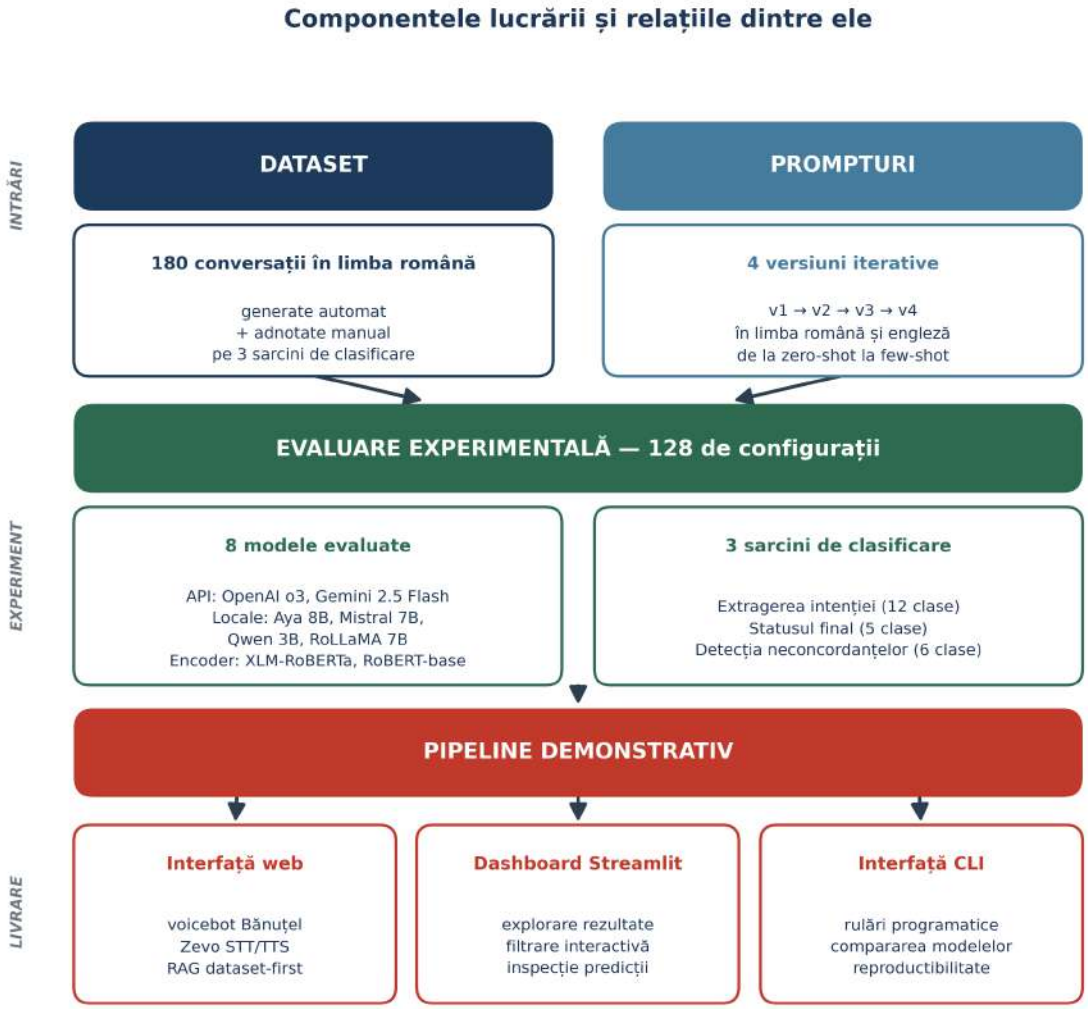


Figura 7.1: Componentele lucrării și relațiile dintre ele, de la datele de intrare la pipeline-ul demonstrativ

7.2. Contribuții personale

Pe parcursul lucrării au fost realizate următoarele contribuții:

Un dataset bancar adnotat pe trei dimensiuni. Au fost generate și adnotate manual 180 de conversații bancare simulate în limba română, inspirate din situații reale de interacțiune cu un voicebot bancar și organizate pe trei sarcini de clasificare: extragerea intenției (12 clase), clasificarea statusului final (5 clase) și detecția neconcordanțelor (6 clase). O parte importantă a contribuției personale a constat în cercetarea preliminară necesară pentru proiectarea corpusului: identificarea celor mai frecvente scenarii întâlnite în practică (blocarea cardului, verificarea soldului, raportarea tranzacțiilor suspecte, resetarea accesului, programări cu consultantul), definirea claselor de status final pornind de la modul în care se încheie efectiv apelurile reale și catalogarea tipurilor de neconcordanțe observate în literatură și în interacțiunile cu sisteme conversaționale.

Procesul de construire a presupus mai multe etape: stabilirea obiectivului cantitativ, generarea automată asistată de un model lingvistic, revizuire manuală, rafinarea definițiilor categoriilor și regenerarea conversațiilor cu prompturi îmbunătățite pentru a elimina ambiguitățile observate.

O metodologie iterativă de proiectare a prompturilor. Construirea prompturilor a fost un proces deliberat, întins pe mai multe etape, în care contribuția personală a depășit cu mult simpla scriere a unor instrucțiuni. Procesul a pornit de la o etapă de cercetare a literaturii și a bunelor practici în prompt engineering, pentru a stabili un cadru de design coerent (structura zero-shot, separarea definițiilor de instrucțiuni, formatul JSON pentru output, principiile de scriere a exemplelor few-shot). Pe baza acestei cercetări am construit patru versiuni succesive de prompturi pentru fiecare sarcină, ordonate ca o progresie controlată: v1 conține doar definiții narative ale claselor în format zero-shot, v2 adaugă reguli stricte de format JSON, v3 introduce criterii operaționale și reguli anti-eroare, iar v4 adaugă exemple few-shot calibrate manual.

Fiecare versiune există în limba română și engleză, ceea ce a permis analiza separată a contribuției limbii promptului față de cea a structurii instrucțiunii. Trecerea de la o versiune la următoarea nu a fost arbitrară: după fiecare iterație a fost realizată analiză manuală a predicțiilor greșite, pentru a identifica tiparele de eroare specifice fiecărui model. Aceste analize manuale au fost combinate cu rapoarte automate generate din matricele de confuzie, iar concluziile obținute au fost folosite pentru a redacta metaprompturi care să ghideze rafinarea versiunii următoare.

O preocupare constantă pe parcursul iterațiilor a fost evitarea overfitting-ului pe exemplele cunoscute. Exemplele few-shot din v4 au fost construite manual, separat de conversațiile din dataset, pentru a evita scurgerile între setul de testare și setul de exemple folosite în prompt. În plus, prompturile au fost testate de mai multe ori pe submulțimi diferite și pe modele cu capacități diferite, pentru a verifica dacă îmbunătățirile sunt consistente sau dependente de un singur model.

O evaluare comparativă pe 128 de configurații. Cele opt modele au fost evaluate sistematic pe cele trei sarcini, cu metrici multiple (accuracy, Macro-F1, Weighted-F1, Cohen's kappa, rata de eșec la parsare, latența medie).

Contribuția personală a depășit simpla rulare a experimentelor. Rezultatele au fost monitorizate în timp real, nu doar la final, ceea ce a permis identificarea devreme a problemelor. Astfel s-a observat că RoBERT-base producea predicții aproape aleatorii încă din primele rulări, ceea ce a condus la excluderea sa rapidă din experimentele ulterioare. Pe măsură ce capacitățile limitate ale modelelor locale inițiale deveneau evidente, am decis adăugarea modelelor Mistral 7B și Qwen2.5 3B pentru a oferi o comparație mai relevantă. Deoarece resursele locale erau depășite pentru aceste modele, am optat pentru rularea lor pe Google Colab cu cuantizare NF4, ceea ce a presupus atenție suplimentară la gestionarea sesiunilor.

O componentă semnificativă a fost și urmărirea erorilor de parsare și rerularea selectivă a configurațiilor problematice. Am identificat și corectat cauze precum nerandarea corectă a exemplelor few-shot pentru v4 (o nepotrivire de nume de variabilă între șablon și logica de încărcare) sau depășirea ferestrei de context pe conversațiile lungi. Fără această monitorizare continuă, multe rezultate ar fi reflectat probleme tehnice, nu performanța reală a modelelor.

Identificarea unui plafon de performanță al modelelor locale cuantizate. Pe sarcina cea mai dificilă, detecția neconcordanțelor, trei modele locale de dimensiuni diferite (Aya 8B, Mistral 7B, Qwen 3B) au atins scoruri Macro-F1 aproape identice, în jurul valorii de 0,28. Această constatare nu a fost presupusă la începutul experimentelor, ci a rezultat din analiza personală a rezultatelor pe măsură ce noi modele erau adăugate în evaluare. Faptul că trei arhitecturi diferite, cu dimensiuni de la 3B la 8B parametri, converg la același scor sugerează existența unui plafon intrinsec al modelelor locale cuantizate pe sarcini complexe. Această observație are implicații directe asupra alegerii modelelor în scenarii cu resurse limitate: dacă

plafonul este același, devine preferabil modelul cel mai rapid și mai fiabil, nu cel mai mare.

Un pipeline funcțional de monitorizare. Plecând de la cele mai bune configurații identificate experimental, am proiectat și implementat de la zero un sistem complet care primește o conversație și produce predicțiile pe cele trei sarcini. Toate deciziile arhitecturale au fost personale: separarea nucleului de evaluare de interfețe, centralizarea configurațiilor într-un registru unic de modele, tratarea erorilor prin fallback local când un model nu este disponibil, și integrarea recomandărilor experimentale direct în configurația implicită a sistemului.

Pipeline-ul este organizat modular, cu logică partajată între o interfață web pentru demonstrații live, un dashboard Streamlit pentru explorarea rezultatelor experimentale și o interfață de linie de comandă pentru rulări programatice. Am implementat interfața web, inclusiv voicebotul demonstrativ Bănuțel, pentru care am proiectat un mecanism RAG ancorat în setul de date al lucrării, astfel încât răspunsurile generate să rămână conforme cu conversațiile evaluate, nu improvizate liber de model. Am integrat serviciul Zevo STT/TTS pentru suportul vocal, am construit dashboard-ul Streamlit cu secțiuni de filtrare, vizualizare și inspecție a predicțiilor individuale, și am implementat interfața CLI pentru reproductibilitate completă.

7.3. Concluziile cele mai importante

Fezabilitatea sistemului este confirmată, dar cu diferențe semnificative între sarcini. Pe extragerea intenției, atât modelele API cât și cele locale ating performanțe ridicate (M-F1 peste 0,88 pentru Aya și Mistral, peste 0,97 pentru API), ceea ce face sarcina viabilă pentru un mediu de monitorizare real, inclusiv cu modele rulate local. Pe clasificarea statusului final, doar modelele API rămân fiabile (M-F1 în jur de 0,85), iar modelele locale pierd semnificativ. Pe detecția neconcordanțelor, doar modelele API oferă rezultate utilizabile (M-F1 peste 0,70 pentru Gemini și o3), restul rămânând sub un prag practic. Concluzia este că un sistem unificat de monitorizare este realist doar prin combinarea acestor configurații.

Decalajul dintre modelele API și cele locale crește cu complexitatea sarcinii. Diferența de Macro-F1 între cel mai bun model API și cel mai bun model local este de 7,4 puncte procentuale pe intenție, 24 de puncte pe statusul final și 44,3 puncte pe neconcordanțe. Această progresie cantitativă arată că arhitecturile mari și antrenarea extensivă au un avantaj din ce în ce mai mare pe sarcini care necesită raționament conversațional avasnat.

Designul promptului contează, dar nu compensează limitele arhitecturale. Rafinarea iterativă de la v1 la v4 aduce câștiguri consistente pentru modelele cu capacitate suficientă: Gemini câștigă 19,2 puncte procentuale pe neconcordanțe între v1 și v4. Modelele locale beneficiază selectiv: Mistral și Qwen câștigă câteva puncte din exemplele few-shot, dar RoLLMa rămâne practic neschimbat, indiferent de versiunea de prompt. Encoderii sunt insensibili la designul prompturilor, ceea ce era de așteptat, dată fiind diferența arhitecturală.

Modelele encoder-only nu sunt viabile pentru această familie de sarcini. RoBERT-base, antrenat exclusiv prin masked language modeling, a obținut un kappa negativ pe intenție și a fost exclus după primul task. XLM-RoBERTa, antrenat cu obiectiv NLI, este mai funcțional dar plafonează la valori mult sub modelele generative. Aceste rezultate confirmă faptul că adaptarea zero-shot prin prompting nu reproduce ce a fost învățat prin instruction tuning.

Pentru implementare practică, Mistral 7B este cel mai bun compromis local. Modelul atinge un Macro-F1 comparabil cu Aya 8B la o latență de aproximativ 10 ori mai mică și o rată de eșec la parsare sub 1,1%. În scenarii cu constrângeri de confidențialitate care impun rularea locală, Mistral oferă cel mai bun raport între performanță, viteză și fiabilitate. Aya a fost în schimb rulat pe o configurație locală, în timp ce Mistral pe o configurație de Google Notebook, cu GPU disponibil. Pentru a fi clar validă, comparația ar trebui efectuată și în

același mediu de rulare (pentru a avea latențe provenite din aceeași sursă).

7.4. Limitări și posibile direcții de continuare

Lucrarea are limitări asumate care deschid drumul către dezvoltări ulterioare. Setul de date este compus din conversații simulate, fără zgomotul tipic al transcrierilor reale produse de un sistem STT dintr-un mediu real; evaluarea pe conversații reale ar arăta în ce măsură rezultatele se transferă în condiții cu erori de transcriere și formulări mai libere. Toate sarcinile sunt formulate ca probleme de clasificare; explorarea unor sarcini deschise, precum sumarizarea sau detectarea cauzei unei nereușite, ar extinde aria de aplicabilitate. Dimensiunea datasetului, deși suficientă pentru un studiu comparativ, limitează granularitatea analizei pe clasele rare; extinderea lui ar permite evaluări mai stabile pentru intențiile și tipurile de neconcordanțe cu frecvență scăzută. De asemenea, extinderea cu un dataset suplimentar care să fie compus dintr-un număr echilibrat de clase ar putea aduce un plus de valoare analizei, mai ales pe sarcina de identificare a neconcordanțelor, unde clasele imită realitatea, acest lucru impunând o mare sensibilitate la o singură predicție greșită care apare în plus.

O direcție naturală de continuare este arhitectura în cascadă: un model local rapid pentru pre-filtrarea conversațiilor simple, urmat de un model API pentru cazurile dificile sau pentru sarcinile pe care modelele locale plafonează. O astfel de configurație ar combina avantajele de cost și confidențialitate ale modelelor locale cu acuratețea modelelor mari, fără a transmite toate conversațiile către un serviciu extern.

7.5. Reflecție finală

Plecând de la o problemă concretă a sectorului bancar, lucrarea a parcurs întregul ciclu de la construirea datasetului și ingineria prompturilor până la evaluarea sistematică și livrarea unui sistem funcțional de demonstrare a conceptului. Rezultatele arată că monitorizarea automată a voiceboților prin modele lingvistice mari nu este nici o promisiune îndepărtată, nici un caz rezolvat, ci o problemă cu soluții parțiale, diferențiate în funcție de sarcină și de constrângerile de implementare. Pentru sarcini relativ simple precum extragerea intenției, soluțiile sunt deja viabile inclusiv în varianta locală. Pentru sarcini complexe precum detecția neconcordanțelor, modelele comerciale rămân singura opțiune practică la momentul actual. Această distincție, susținută cantitativ și ilustrată prin pipeline-ul demonstrativ, este principala contribuție de fond a lucrării.

BIBLIOGRAFIE

[1] 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)*, pp. 5998–6008 2017.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, „BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, pp. 4171–4186 2019.

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, Dario Amodei, „Language Models are Few-Shot Learners", *Advances in Neural Information Processing Systems (NeurIPS)* 2020.

[4] OpenAI, „GPT-4 Technical Report", *arXiv preprint*, vol. nr. 2023.

[5] OpenAI, *o3 and o4-mini System Card*, Accesat la data de 18.01.2026, 2025, Disponibil: `https://cdn.openai.com/pdf/2221c875-02dc-4789-800b-e7758f3722c1/o3-and-o4-mini-system-card.pdf`.

[6] Google DeepMind, *Gemini 2.0 Flash Model Card*, Accesat la data de 12.01.2026, 2024, Disponibil: `https://modelcards.withgoogle.com/assets/documents/gemini-2-flash.pdf`.

[7] John Dang, et al., „Aya Expanse: Combining Research Breakthroughs for a New Multilingual Frontier", *arXiv preprint*, vol. nr. 2024.

[8] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier et al., „Mistral 7B", *arXiv preprint arXiv:2310.06825*, vol. nr. 2023.

[9] Qwen Team, „Qwen2.5 Technical Report", *arXiv preprint arXiv:2412.15115*, vol. nr. 2025.

[10] Mihai Masala, Denis C. Ilie-Ablachim, Dragos Corlatescu, Miruna Zavelca, Marius Leordeanu, Horia Velicu, Marius Popescu, Mihai Dascalu, Traian Rebedea, „OpenLLM-Ro – Technical Report on Open-source Romanian LLMs", *arXiv preprint*, vol. nr. 2024.

[11] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, Veselin Stoyanov, „Unsupervised Cross-lingual Representation Learning at Scale", vol. nr. 2020.

[12] Mihai Masala, Stefan Ruseti, Mihai Dascalu, „RoBERT – A Romanian BERT Model", vol. nr. Pp. 6626–6637 2020.

[13] OpenAI, *Latency Optimization*, Accesat la data de: 22.01.2026, 2024, Disponibil: https://platform.openai.com/docs/guides/latency-optimization.

[14] M. Reid et al., „Gemini 1.5: Unlocking Multimodal Understanding Across Millions of Tokens of Context", *arXiv preprint arXiv:2403.05530*, vol. nr. 2024.

[15] Alex Wang et al., „On the Latency of Transformer-Based Text Classification Models", *arXiv preprint arXiv:2004.08900*, vol. nr. 2020.

[16] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov, „RoBERTa: A Robustly Optimized BERT Pretraining Approach", *arXiv preprint arXiv:1907.11692*, vol. nr. 2019.

[17] Sumanyu Sharma, *Voice AI Latency: What's Fast, What's Slow, and How to Fix It.* [Online] Disponibil: https://hamming.ai/resources/voice-ai-latency-whats-fast-whats-slow-how-to-fix-it Accesat: iunie 2026.

[18] Rost Glukhov, *Reduce LLM Costs: Token Optimization Strategies*, 2025, Disponibil: https://www.glukhov.org/llm-performance/cost-effective-llm-applications/ (accesat în 13.6.2026).

[19] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, Dario Amodei, „Scaling Laws for Neural Language Models", *arXiv preprint arXiv:2001.08361*, vol. nr. 2020.

[20] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, Laurent Sifre, „Training Compute-Optimal Large Language Models", *Advances in Neural Information Processing Systems (NeurIPS)* 2022.

[21] Mihai Masala, Denis C. Ilie-Ablachim, Dragos Corlatescu, Miruna Zavelca, Marius Leordeanu, Horia Velicu, Marius Popescu, Mihai Dascalu, Traian Rebedea, *OpenLLM-Ro – Technical Report on Open-source Romanian LLMs.* [Online] Disponibil: https://arxiv.org/abs/2405.07703.

[22] Mihai Masala, Denis C. Ilie-Ablachim, Alexandru Dima, Dragos Corlatescu, Miruna Zavelca, Ovio Olaru, Simina Terian, Andrei Terian, Marius Leordeanu, Horia Velicu, Marius Popescu, Mihai Dascalu, Traian Rebedea, *"Vorbeşti Românește?" A Recipe to Train Powerful Romanian LLMs with English Instructions.* [Online] Disponibil: https://arxiv.org/abs/2406.18266.

[23] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt et al., „GPT-4 Technical Report", *arXiv preprint arXiv:2303.08774*, vol. nr. 2023.

[24] Guowei Sheng et al., „Efficient Large Language Model Serving: A Survey", *arXiv preprint arXiv:2312.15234*, vol. nr. 2023.

[25] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, Ion Stoica, „Efficient Memory Management for Large Language Model Serving with PagedAttention", *arXiv preprint arXiv:2309.06180*, vol. nr. 2023.

[26] Anthropic, *Claude.* [Online] Disponibil: https://www.anthropic.com/claude Accesat la 27.02.2026.

[27] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, Percy Liang, „Lost in the Middle: How Language Models Use Long Contexts", *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)* 2024.

[28] Shubham Vatsal, Harsh Dubey, „A Survey of Prompt Engineering Methods in Large Language Models for Different NLP Tasks", *arXiv preprint arXiv:2407.12994*, vol. nr. 2024.

[29] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, Graham Neubig, „Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing", *ACM Computing Surveys*, vol. 55, nr. 9, pp. 1–35 2023.

[30] E. M. Freeburg, „The Last Fingerprint: How Markdown Training Shapes LLM Prose", *arXiv preprint arXiv:2603.27006*, vol. nr. 2026.

[31] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, Denny Zhou, „Chain-of-Thought Prompting Elicits Reasoning in Large Language Models", *Advances in Neural Information Processing Systems*, vol. 35, nr. Pp. 24824–24837 2022.

[32] J. Richard Landis, Gary G. Koch, „The Measurement of Observer Agreement for Categorical Data", *Biometrics*, vol. 33, nr. 1, pp. 159–174 1977.

[33] Wenpeng Yin, Jamaal Hay, Dan Roth, „Benchmarking Zero-shot Text Classification: Datasets, Evaluation and Entailment Approach", *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pp. 3914–3923 2019.

[34] Lucian Georgescu, Horia Cucu, *Laborator 6 — Biosinteză vocală și recunoaștere automată a vorbirii*, Material de laborator, ICVSI, Universitatea Națională de Știință și Tehnologie POLITEHNICA București. [Online].

Anexă: Prompturile configurațiilor optime

Această anexă prezintă textul complet al prompturilor Jinja2 corespunzătoare configurațiilor cu cel mai bun Macro-F1 per task: EN v4 pentru extragerea intenției (M-F1 = 0,983, OpenAI o3), RO v4 pentru clasificarea statusului final (M-F1 = 0,860, OpenAI o3) și EN v4 pentru detecția neconcordanțelor (M-F1 = 0,734, Gemini 2.5 Flash). Blocurile {% for ... %} reprezintă directive de templating Jinja2 prin care conversația și exemplele few-shot sunt injectate dinamic la momentul inferenței. Toate celelalte versiuni de prompt (v1–v4, română și engleză, pentru cele trei taskuri) sunt disponibile în repository-ul public al proiectului:

github.com/teodoramirzan/
Sistem-de-monitorizare-a-interac-iunilor-voicebotilor-
folosind-modele-lingvistice-mari-LLM-

Extragerea intenției utilizatorului — v4 EN (few-shot)

```
Developer: # Role and Objective
You are an intent classification assistant for a Romanian banking voicebot system.

Your objective is to identify the **primary intent** of the entire conversation below. The
     intent is determined by what the user wanted to accomplish within the conversation.

# Intent Labels
- **‘block_card‘**: The user wants to block a lost, stolen, retained, or compromised card.
- **‘unblock_card‘**: The user wants to unblock a previously blocked card.
- **‘open_account‘**: The user wants to open a new bank account.
- **‘close_account‘**: The user wants to close an existing bank account.
- **‘check_balance‘**: The user wants to check the account balance or available funds.
- **‘get_account_statement‘**: The user wants to obtain the account statement or
    transaction history.
- **‘report_suspicious_transaction‘**: The user reports a suspicious, fraudulent, or
    unauthorized transaction.
- **‘update_personal_data‘**: The user wants to update personal or contact information
    associated with the account.
- **‘schedule_advisor_meeting‘**: The user wants to schedule a meeting or discussion with
    a banking advisor.
- **‘reset_or_recover_auth‘**: The user wants to reset or recover authentication
    credentials such as password, PIN, or app access.
- **‘general_product_info‘**: The user requests general information about banking products,
     services, fees, or conditions.
- **‘fallback‘**: The conversation contains at least one user request outside the banking
    domain, regardless of whether valid banking requests also appear in the same dialogue.

# Examples

{% for example in examples %}
```

```
---

## Example {{ loop.index }}
{% for turn in example.turns %}
{{ turn.role | upper }}: {{ turn.text }}
{% endfor %}

{''intent'': ''{{ example.intent }}'', ''confidence'': ''high'', ''reasoning'': ''{{
    example.reasoning }}''}

{% endfor %}
---

# Context
## Conversation
{% for turn in conversation %}
{{ turn.role | upper }}: {{ turn.text }}
{% endfor %}

# Instructions
- Classify the intent based on the **goal** with which the user initiated the conversation,
     typically defined in the first exchanges.
- If the user makes **any request** outside the banking domain at any point in the
    conversation, use `fallback`.
- Later turns such as confirmations, providing details, or ''thank you'' do not change the
     intent.

# Planning and Verification
- Identify the primary goal pursued by the user.
- Check whether any request falls outside the banking domain; if so, choose `fallback`.
- Ignore apparent intent changes produced only by confirmations or administrative details.
- Return exactly one intent label, along with a confidence level and a short justification.


# Output Format
Respond with a valid JSON object only. No explanation, no markdown, no preamble.

{
  ''intent'': ''<label>'',
  ''confidence'': ''<high|medium|low>'',
  ''reasoning'': ''<one concise sentence in English explaining your choice>''
}

# Verbosity
- Be concise.
- The reasoning must be a single short sentence in English.

# Stopping Conditions
- Stop after returning the valid JSON object.
- Do not add any additional text outside the JSON.
```

Clasificarea statusului final — v4 RO (few-shot)

```
Developer: Eşti un clasificator al rezultatului conversaţiilor pentru un sistem voicebot
    bancar în limba română.

```

```
Sarcina ta este să clasifici **statusul final** al conversaţiei de mai jos, pe baza modului în care s-a încheiat şi dacă solicitarea utilizatorului a fost îndeplinită.

## Etichete de status

**rezolvata**
Voicebotul a finalizat tot ce putea face în cadrul conversaţiei pentru cererea utilizatorului. Aceasta include: înregistrarea unui raport, blocarea unui card, trimiterea unui extras, furnizarea informaţiilor solicitate, activarea unui serviciu, programarea unei întâlniri sau alte solicitări bancare. Faptul că utilizatorul trebuie să **aştepte** un rezultat ulterior (investigaţie, livrare card, procesare) nu face conversaţia parţial rezolvată --- dacă botul a executat acţiunea solicitată şi nu mai are nimic de făcut, statusul este ‘rezolvata‘.

**partial_rezolvata**
Cererea a fost adresată parţial. Se aplică în două situaţii:
1. Utilizatorul trebuie să **acţioneze activ** după conversaţie pentru a-şi îndeplini scopul: să meargă la sucursală, să completeze un formular, să acceseze o aplicaţie pentru a finaliza o operaţiune pe care botul nu a putut-o face.
2. Utilizatorul a avut mai multe cereri şi doar o parte au fost rezolvate.

Distincţie importantă: ,,a aştepta’’ nu este ,,a acţiona’’. Dacă utilizatorul doar aşteaptă (livrare card, rezultat investigaţie, confirmare SMS), statusul este ‘rezolvata‘. Dacă utilizatorul trebuie să **facă ceva** (să meargă undeva, să completeze documente, să sune din nou), statusul este ‘partial_rezolvata‘.

Foloseşte ‘partial_rezolvata‘ doar când utilizatorul trebuie să programeze, să sune, să confirme, să completeze formulare sau să meargă la sucursală fără ca o programare/iniţiere concretă a unei întâlniri sau a unui apel cu un reprezentant uman să fi fost efectiv realizată în conversaţie.

**nerezolvata**
Voicebotul a încercat să rezolve cererea dar nu a reuşit. Botul a oferit răspunsuri irelevante, a dat erori, nu a procesat cererea, sau acţiunea solicitată depăşeşte capacităţile voicebotului.

Foloseşte ‘nerezolvata‘ doar când botul spune explicit că nu poate ajuta, că solicitarea este în afara capacităţilor/domeniului său, oferă răspunsuri irelevante sau eşuează clar după ce încearcă să rezolve cererea.

**redirectionata**
Conversaţia se încheie cu programarea sau iniţierea unei întâlniri cu un advisor, consultant, specialist sau operator uman. Se aplică în oricare dintre aceste situaţii:
- Utilizatorul a cerut explicit să vorbească cu un specialist şi botul a programat întâlnirea, cu dată şi oră.
- Botul a iniţiat programarea unei întâlniri ca soluţie pentru cererea utilizatorului, definitivând data şi ora.
- Intenţia iniţială a utilizatorului era să programeze o întâlnire, iar aceasta a fost confirmată cu dată şi oră.

Dacă conversaţia include o programare cu un advisor cu dată şi oră, statusul este ‘redirectionata‘, chiar dacă botul a oferit şi informaţii suplimentare în aceeaşi conversaţie. Programarea unei întâlniri are prioritate asupra altor acţiuni parţiale.

Dacă botul a programat deja, a confirmat sau a iniţiat concret o întâlnire sau un apel cu un advisor, consultant, specialist, operator sau alt angajat al băncii, clasifică drept ‘redirectionata‘. Faptul că utilizatorul mai trebuie să participe la întâlnire sau apel nu face cazul ‘partial_rezolvata‘.

```

```
Nu se încadrează aici simpla sugestie de a vizita o sucursală sau de a contacta un
    operator, fără o programare efectivă care să conţină o dată şi o oră.

**intrerupta**
Conversaţia s-a încheiat înainte ca voicebotul să aibă şansa de a rezolva cererea. Semnale
     tipice:
- Utilizatorul nu mai răspunde după ce botul pune o întrebare sau cere informaţii.
- Utilizatorul spune explicit că se va gândi, va reveni mai târziu, sau se răzgândeşte.
- Conversaţia se opreşte brusc fără un schimb final de confirmare.
- Ultimul mesaj este de la bot (întrebare, cerere de date), iar utilizatorul nu continuă.

Dacă conversaţia se opreşte pentru că utilizatorul nu răspunde, nu oferă informaţiile
    cerute, nu confirmă, sau ultimul mesaj relevant este o întrebare/cerere din partea
    botului, clasifică drept ‘intrerupta‘.

## Distincţii cheie

- ‘rezolvata‘ vs ‘partial_rezolvata‘: dacă botul a executat acţiunea şi utilizatorul doar
    **aşteaptă** un rezultat (investigaţie, livrare, procesare), statusul este ‘rezolvata‘.
     Dacă utilizatorul trebuie să **acţioneze** (mers la sucursală, completare formular,
    semnare document), statusul este ‘partial_rezolvata‘.
- ‘nerezolvata‘ vs ‘intrerupta‘: în ‘nerezolvata‘ voicebotul a încercat şi a eşuat sau a
    oferit răspunsuri irelevante. În ‘intrerupta‘ conversaţia s-a oprit înainte ca
    voicebotul să aibă şansa să finalizeze --- de obicei utilizatorul nu mai răspunde sau
    pleacă.
- ‘redirectionata‘ vs ‘partial_rezolvata‘: dacă botul a programat deja, a confirmat sau a
    iniţiat concret o întâlnire sau un apel cu un advisor, consultant, specialist,
    operator sau alt angajat al băncii cu dată şi oră, statusul este întotdeauna ‘
    redirectionata‘, nu ‘partial_rezolvata‘. Faptul că utilizatorul trebuie să participe
    la întâlnire sau apel nu o face parţial rezolvată. Foloseşte ‘partial_rezolvata‘ doar
    când utilizatorul mai trebuie să programeze, să sune, să confirme, să completeze
    formulare, să aştepte confirmarea unei date/ore sau să meargă la sucursală fără o
    programare efectiv realizată în conversaţie.
- ‘redirectionata‘ vs ‘rezolvata‘: chiar dacă programarea întâlnirii a fost cererea iniţ
    ială şi a fost confirmată cu succes, statusul rămâne ‘redirectionata‘, nu ‘rezolvata‘.
     Programarea unei întâlniri este prin natura ei o redirecţionare către un specialist.
- ‘intrerupta‘ vs ‘partial_rezolvata‘: dacă botul a oferit informaţii dar utilizatorul nu
    confirmă, nu continuă, sau spune că revine mai târziu, statusul este ‘intrerupta‘, nu ‘
    partial_rezolvata‘.
- ‘intrerupta‘ vs ‘nerezolvata‘: dacă conversaţia se opreşte pentru că utilizatorul nu ră
    spunde, nu oferă informaţiile cerute, nu confirmă, sau ultimul mesaj relevant este o î
    ntrebare/cerere a botului, statusul este ‘intrerupta‘. Foloseşte ‘nerezolvata‘ doar câ
    nd botul nu poate ajuta, spune că solicitarea este în afara capacităţilor/domeniului s
    ău, oferă răspunsuri irelevante sau eşuează clar după ce încearcă să rezolve cererea.

## Exemple

{% for ex in examples %}
### Exemplul {{ loop.index }}

**Conversaţie:**
{% for turn in ex.conversation %}
{{ turn.role | upper }}: {{ turn.text }}
{% endfor %}

**Clasificare corectă:**
{
  ''final_status'': ''{{ ex.final_status }}'',
  ''reasoning'': ''{{ ex.reasoning }}''
```

```
}

{% endfor %}

## Conversaţie de clasificat

{% for turn in conversation %}
{{ turn.role | upper }}: {{ turn.text }}
{% endfor %}

## Instrucţiuni
- Bazează clasificarea pe modul în care conversaţia **s-a încheiat**, nu pe intenţia iniţ
    ială.
- Citeşte cu atenţie ultimele replici --- ele conţin semnalul de rezoluţie.
- Verifică dacă există o programare cu un advisor --- dacă da, statusul este ‘
    redirectionata‘.
- Verifică dacă utilizatorul a confirmat şi a încheiat conversaţia sau dacă a plecat fără
    confirmare.
- Alege exact o etichetă din cele cinci.

## Output
Răspunde doar cu un obiect JSON valid. Fără explicaţii, fără markdown, fără preambul.

{
  ''final_status'': ''<etichetă>'',
  ''confidence'': ''<high|medium|low>'',
  ''reasoning'': ''<o propoziţie concisă în română care explică alegerea>''
}
```

Detecţia neconcordanţelor — v4 EN (few-shot)

```
You are a quality auditor for a Romanian-language banking voicebot system.

Your task is to analyze the conversation below and determine whether the voicebot's
    responses contain an incongruity according to the definitions in this prompt.

IMPORTANT:
Do not evaluate the conversation based on real banking rules, real security procedures, or
     assumptions about what a voicebot could do in reality.
Evaluate only the text of the conversation and the labeling rules below.

## Objective

For each conversation:
1. determine whether there is an incongruity in the ASSISTANT messages;
2. if there is one, choose a single dominant type from:
‘incomplet‘, ‘irelevant‘, ‘contradictoriu‘, ‘nealiniat_context‘, ‘halucinatie‘.

## Examples

Here are examples of correct classifications to better understand each type of incongruity:


{% for example in few_shot_examples %}
### Example {{ loop.index }}

```

```
**Conversation:**
{% for turn in example.conversation %}
{{ turn.role | upper }}: {{ turn.text }}
{% endfor %}

**Correct classification:**
```json
{
  ''has_incongruity'': {{ example.has_incongruity | lower }},
  ''incongruity_type'': {{ '''' + example.incongruity_type + '''' if example.
    incongruity_type else 'null' }},
  ''reasoning'': ''{{ example.reasoning }}''
}
```

{% endfor %}

General rules

- Evaluate only ASSISTANT messages, not USER messages.
- Mark `has_incongruity: true` only if the error is clear, explicit, and directly
    supported by the text.
- If the issue is only a lack of detail, a redirection, a clarification question, or a
    possible banking procedure, DO NOT mark an incongruity.
- If the bot responds through authentication, verification, redirection, transfer to an
    operator, mobile app, email, SMS, or branch, consider the response valid if it is
    related to the request.
- Do not penalize the bot for not directly solving the request within the conversation, if
     it acknowledges it or provides a relevant next step.
- Do not use assumptions about what is realistic in a real bank.
- Do not mark hallucination only because the bot says it updated, sent, confirmed, blocked,
     scheduled, or processed something.
- Do not mark incomplete only because the bot does not provide the exact value, address,
    amount, balance, or requested detail, if it provides a relevant way to obtain it.
- Do not mark context misalignment only because the bot does not repeat all details
    provided by the user.
- Do not mark contradictory for differences in wording, different steps of the same
    process, or compatible timeframes.

Types of incongruities

incomplet

Choose `incomplet` only when the bot completely ignores an explicit user request.

Mandatory conditions:
- the user makes a clear request;
- the bot does not mention that request at all;
- the bot provides neither an answer, nor a clarification, nor a redirection, nor a
    relevant next step for the request.

It is NOT `incomplet` if:
- the bot acknowledges the request, even without fully solving it;
- the bot asks for authentication or additional information;
- the bot redirects to the app, a branch, email, an operator, or a department;
- the bot provides an alternative resolution method;
- the bot responds partially or procedurally to the request;
- the bot does not provide an exact value, but indicates where it can be obtained.

```

```
irelevant

Choose ‘irelevant‘ only when the bot’s response is completely off-topic from the user’s
    request.

Mandatory conditions:
- the response does not help the current request at all;
- the topic introduced is completely different from the user’s intent;
- there is no functional connection to the request.

It is NOT ‘irelevant‘ if:
- the bot politely refuses a non-banking request and returns to banking services;
- the bot offers a banking product related to the context;
- the bot proposes the app, a branch, an operator, or a department;
- the bot provides a relevant banking next step;
- the response is within the same intent area, even if it is not perfect.

contradictoriu

Choose ‘contradictoriu‘ only when the bot states the same information with explicitly
    incompatible values in two ASSISTANT replies.

Mandatory conditions:
- there are at least two bot statements;
- the statements refer to the same object, the same action, or the same timeframe;
- the values are clearly different and incompatible.

It is NOT ‘contradictoriu‘ if:
- a general timeframe appears next to a maximum timeframe, such as ’’shortly’’ and ’’
    within 24/48 hours’’;
- one step takes a certain amount of time, while the confirmation has a different
    timeframe;
- the bot says it is processing something and then confirms that the request has been
    registered;
- the bot says the action will be completed later, but the request has been registered now
    ;
- two complementary methods are stated, not opposite ones;
- the difference is only procedural or related to wording.

nealiniat_context

Choose ‘nealiniat_context‘ only when the bot explicitly uses information that differs from
     what the user provided.

Mandatory conditions:
- the user provided clear and relevant information;
- the bot uses another value, another product, another channel, another card, another
    account, another date, another time, or another email;
- the error effectively changes the action or the response.

It is NOT ‘nealiniat_context‘ if:
- the bot does not repeat all user details;
- the bot asks clarifying questions;
- the bot asks for authentication;
- the bot provides an alternative path;
- the bot recommends an operator, the app, or a branch;
- the bot responds generally, but compatibly with the existing information;
- the bot continues a relevant procedure without mentioning every previous detail;
- the bot refuses or redirects a non-banking request.
```

```
122
123 ### halucinatie
124
125 Choose ‘halucinatie‘ only when the bot introduces information that is invented, absurd,
        exaggerated, or unprofessional according to the dataset rules.
126
127 It applies especially when the bot:
128 - uses unprofessional, alarmist, or exaggerated vocabulary for a banking voicebot;
129 - mentions extreme or impossible consequences;
130 - mentions specific fees, penalties, commissions, amounts, limits, or thresholds without a
         basis in the conversation;
131 - mentions absurd or impossible timeframes;
132 - presents a specific piece of information as a certain banking rule without support from
        the context.
133
134 It is NOT ‘halucinatie‘ merely because:
135 - the bot says it has sent, processed, blocked, updated, or scheduled something;
136 - the bot mentions email, SMS, app, branch, or operator;
137 - the action seems too fast or too simple for a real bank;
138 - real verification steps are missing;
139 - the model considers the procedure unlikely in the real world;
140 - the bot provides a general but bank-plausible solution.
141
142 ## Special rules against frequent errors
143
144 - Non-banking requests such as weather, jokes, restaurants, guitar, personal advice, or
        fast money may be refused or redirected toward banking services without being an
        incongruity.
145 - If the user asks for an agent, consultant, or specialist, and the bot transfers,
        schedules, redirects, or says that an agent will contact the user, it is NOT ‘
        nealiniat_context‘.
146 - If the user asks for balance, statement, transactions, or documents, and the bot asks
        for authentication or recommends the app / online banking / branch / operator, it is
        NOT ‘incomplet‘.
147 - If the user provides transaction details and the bot confirms reporting without
        repeating all details, it is NOT ‘nealiniat_context‘.
148 - If the bot gives a general timeframe and then a maximum timeframe, it is NOT ‘
        contradictoriu‘.
149 - If the bot says that a request has been registered, but completion takes longer, it is
        NOT ‘contradictoriu‘.
150 - If the bot provides an optional recommendation after answering the main request, it is
        NOT automatically ‘irelevant‘.
151 - If the bot recommends an unsolicited banking product, it is ‘irelevant‘ only if it
        completely replaces the answer to the main request.
152
153 ## Decision order
154
155 Apply this order:
156
157 1. If the bot introduces fees, amounts, penalties, impossible timeframes, extreme
        consequences, or unprofessional language without a basis in the conversation, choose ‘
        halucinatie‘.
158 2. If the bot explicitly uses a different value from the one provided by the user, choose ‘
        nealiniat_context‘.
159 3. If the bot provides two clearly incompatible values for the same information, choose ‘
        contradictoriu‘.
160 4. If the bot completely ignores an explicit request and provides no relevant step, choose
         ‘incomplet‘.
```

```
5. If the bot responds on a completely different topic, with no functional connection to the request, choose ‘irelevant‘.
6. If none of the conditions are clearly met, set ‘has_incongruity‘ to ‘false‘.

## Conversation to evaluate

{% for turn in conversation %}
{{ turn.role | upper }}: {{ turn.text }}
{% endfor %}

## Output instructions

Respond only with a valid JSON object.
Do not add explanations, markdown, or a preamble.
Do not add extra fields.
Follow exactly the field names and label values.

If there is no incongruity:
- ‘has_incongruity‘ must be ‘false‘;
- ‘incongruity_type‘ must be ‘null‘.

If there is an incongruity:
- ‘has_incongruity‘ must be ‘true‘;
- ‘incongruity_type‘ must be one of:
‘incomplet‘, ‘irelevant‘, ‘contradictoriu‘, ‘nealiniat_context‘, ‘halucinatie‘.

‘reasoning‘ must be a single concise sentence in English.
If there is an incongruity, briefly mention the relevant ASSISTANT turn.
If there is no incongruity, briefly explain why the responses are acceptable according to the rules.

{
  ''has_incongruity'': <true|false>,
  ''incongruity_type'': ''<label>|null'',
  ''confidence'': ''<high|medium|low>'',
  ''reasoning'': ''<a concise sentence in English explaining the decision>''
}
```