POLITECNICO DI TORINO

ICT For Smart Societies

Master's Thesis

Handoff Recommendation in Multi-Agent Conversational Systems

by

Luca Bernardi

Supervisor (Academic)

Prof. Guido Albertengo

Industrial Supervisor

Stefano Rizzo, Twiper S.r.l.

Academic Year 2025–2026

Abstract

This thesis investigates the design, implementation, and production deployment of a machine learning–based handoff recommendation system for Kora.ai, a multi-agent conversational platform developed by Twiper S.r.l. Enterprise chatbots increasingly rely on specialised agents—retrieval modules, expert responders, and ticket processors—to handle diverse user requests. In such workflows, deciding *when* and *to which* agent a conversation should be transferred is critical for accuracy, operational efficiency, and user satisfaction.

The work comprises two machine learning modules trained on operational conversation logs (407 conversations for auto-tagging; 397 handoff events across both chatbots, of which 313 belong to chatbot 1 after eligibility filtering). Module A is a conversation-level *auto-tagging* system that assigns semantic labels using text embeddings and One-vs-Rest logistic regression. On a cleaned evaluation set with nine macro-topics, it achieves a Macro-F1 score of 0.67. These semantic tags provide interpretable topic distributions for analytics and can optionally be used as features for downstream models. Module B is an event-based *handoff recommender* that predicts the target agent from pre-handoff signals only, including pre-trained text embeddings, auto-tag probability scores, and structured metadata. On the primary chatbot configuration, involving two eligible target agents, a text-only variant achieves a Top-1 accuracy of 95.5%, outperforming a rule-based router baseline (92.4%). An ablation study indicates that embeddings alone are sufficient for the current binary routing regime (C=2), while auto-tag features provide no measurable gain.

For production deployment, the system is integrated into Kora.ai's Python workflow engine in **suggestion-only** mode: recommendations are presented to operators for validation, enabling feedback collection and gradual rollout. A hybrid strategy further combines deterministic guards (triggered when historical routing purity ≥ 0.99 and support ≥ 10) with an ML fallback to guarantee coverage at runtime. Strict anti-leakage protocols (timestamp filtering and feature whitelisting) ensure valid offline evaluation. Overall, this work demonstrates that lightweight statistical models, supported by careful feature engineering and production-grade integration, can effectively support multi-agent orchestration under realistic constraints such as small data, noisy labels, and class imbalance.

Keywords: conversational AI, handoff routing, multi-agent systems, semantic embeddings, machine learning

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and Motivation

While single-agent chatbots can handle many simple tasks, real-world applications often require multiple specialised agents. For example, an organisation may combine a retrieval agent, a domain expert agent, and a ticket-creation agent within the same conversational workflow. In this setting, a central challenge emerges: *when* a conversation should be handed off from one agent to another, and *which* agent should take over next. Misaligned handoffs can lead to inconsistent answers, unnecessary ticket creation, and a degraded user experience.

Twiper S.r.l. is developing **Kora.ai**, a multi-agent conversational platform designed to simplify the creation and management of such workflows. The platform enables organisations to orchestrate retrieval, domain expertise, and operational tools (e.g., ticket creation) through a unified conversational interface. However, as the number of agents and supported use cases grows, manual routing rules become difficult to maintain and may not generalise well to diverse user intents. This motivates the need for **data-driven handoff recommendation**: learning from historical conversation traces to predict which agent should handle the next step of a conversation.

This thesis investigates the design, implementation, and production deployment of a machine learning system for agent handoff recommendation within Kora.ai. The work is conducted under realistic constraints, including limited annotated data (407 conversations;

313 handoff events after filtering for chatbot 1), operational label noise, and severe class imbalance. The resulting system achieves strong performance on a binary routing task (C=2 eligible agents) and has been integrated in **suggestion-only** mode, where operators validate recommendations before finalising transfers. This approach follows the principles of Human-Centered Artificial Intelligence [1], where AI is designed to amplify human capabilities rather than replace them, and is consistent with the Human-in-the-loop paradigm. [2]

1.2 Objectives of the Thesis

The objective of this thesis is to design, implement, and evaluate a machine learning pipeline that improves handoff decisions between conversational agents within the Kora.ai platform. The work addresses three main goals:

- **Semantic Enrichment (Module A):** develop a multi-label auto-tagging module that assigns semantic labels (e.g., `FISCO_IVA`, `CONTABILITA`, `TICKET`) to conversations based on their textual content. These tags provide interpretable summaries for analytics and serve as optional features for downstream models.

- **Handoff Recommendation (Module B):** train a recommender that predicts the most appropriate next agent using pre-trained text embeddings, auto-tag probability scores, and structured metadata. The model is trained under strict anti-leakage protocols, specifically through timestamp-based filtering and feature whitelisting, to avoid the common pitfall of temporal leakage in production data mining [3].

- **Production Integration:** integrate the models into the live Kora.ai workflow engine, enabling suggestion-only deployment where operators review and validate recommendations. This includes implementing feedback mechanisms, monitoring signals, and graceful degradation to support a gradual and safe rollout.

1.3 Research Questions

Given the operational constraints of the setting (small data, label noise, and class imbalance), this thesis is guided by the following research questions:

- **RQ1:** Can a lightweight ML model based on pre-trained text embeddings provide reliable handoff recommendations under limited supervision?
- **RQ2:** Do semantic auto-tags (Module A) improve routing performance when used as additional features for the handoff recommender (Module B), beyond a text-only baseline?
- **RQ3:** How can an ML-based router be deployed safely in production, ensuring anti-leakage evaluation offline and robust behaviour at runtime via operator validation and fallback strategies?

1.4 Scope and Contributions

The contributions of this thesis include:

- A **reproducible data extraction pipeline** that constructs training datasets from production logs while enforcing strict anti-leakage constraints (Chapter 3).
- A **multi-label semantic tagging model** (Module A) A system capable of assigning multiple topics to a single conversation. The model leverages text embeddings and One-vs-Rest logistic regression. On a evaluation set of 9 topics, it achieves a Macro-F1 score of ≈ 0.67 on a deduplicated test set (0.695 on the full test split). Although learning curves indicate diminishing returns, the model still shows measurable gains with more data, suggesting that further labeled examples will continue to drive better performance.
- A **handoff recommender** (Module B) based on logistic regression over pre-trained embeddings (`text-embedding-3-small`, 1536 dimensions), achieving Top-1 accuracy $= 95.5\%$ on the primary chatbot (C=2 eligible agents, $N_{\text{test}} = 66$). An ablation

study shows that embeddings alone are sufficient in the current binary regime, with no measurable gain from auto-tag features (Chapter 5).

- A **hybrid routing strategy** combining deterministic guards (purity ≥ 0.99) with ML fallback to guarantee runtime coverage (Chapter 5).
- A **production integration** in suggestion-only mode, including versioned model artifacts, feedback collection endpoints, graceful degradation, and monitoring infrastructure (Chapter 6).

Limitations acknowledged: the work is constrained by the small sample size (313 handoff events after filtering), severe class imbalance, and a binary routing regime ($C = 2$) due to eligibility filtering. While these results demonstrate feasibility under production constraints, extension to a true multi-class setting (e.g., $C > 5$) will likely require additional data and further validation. These constraints highlight the inherent challenges of deploying machine learning in real-world environments, where data scarcity and label noise can significantly affect model stability and performance evaluation [4].

1.5 Structure of the Thesis

The thesis is organised as follows.

Chapter 2 presents an overview of the Kora.ai platform architecture, including the Backoffice, API Core, RAG/ML service, and database design. This chapter establishes the operational context and identifies where handoff decisions occur within the system.

Chapter 3 describes the dataset construction process and formalises the two machine learning tasks: conversation-level auto-tagging (Module A) and event-based handoff recommendation (Module B). Special attention is given to anti-leakage protocols, including strict timestamp filtering and feature whitelisting.

Chapter 4 presents Module A, the auto-tagging pipeline. This includes macro-topic design, embedding-based representation, One-vs-Rest logistic regression training, and validation experiments (near-duplicate impact, learning curves, and error analysis).

Chapter 5 introduces Module B, the handoff recommender. This chapter covers the model formulation, training protocol (group-wise splitting by conversation ID), ablation studies, cross-seed stability, calibration analysis, and hybrid strategy design.

Chapter 6 illustrates the production integration, including runtime feature building, inference workflow, suggestion-only handoff logic, feedback collection, versioned artifacts, and graceful degradation strategies.

Chapter 7 concludes with a summary of results, discussion of limitations, and directions for future work.

Chapter 2

System Overview: The Kora.ai Platform

2.1 Platform Architecture Overview

This chapter provides a high-level overview of the Kora.ai platform used in this thesis. Kora.ai adopts a modular, service-oriented architecture that separates configuration, orchestration, retrieval, and storage concerns, adhering to the principles of microservices and bounded contexts defined by [5] and [6]. This approach ensures that the configuration plane remains decoupled from the intelligence plane. At the top layer, two user-facing applications interact with the system: a *Backoffice* web application for administrators and a *Chat UI* for end users. Both clients communicate via HTTP with the *API Core*, which acts as the main gateway and orchestration layer.

The API Core coordinates authentication, conversation lifecycle management, and access to persistent storage. It also dispatches inference requests to a dedicated *RAG/ML service* that performs retrieval-augmented generation and executes the runtime decision logic (e.g., tool invocation and handoff suggestions). Document ingestion and indexing are decoupled into a separate pipeline that processes uploads, segments content, computes embeddings, and stores vector representations in a vector database. Persistent storage is split between a relational database (PostgreSQL), used for transactional application data and logs, and a vector store optimized for similarity search.

This separation supports scalability and maintainability: components can be upgraded independently while preserving clear interfaces between the configuration plane (Backof-

fice), the conversation plane (Chat UI + API Core), and the intelligence plane (RAG/ML service). Figure 2.1 summarizes the major components and their data flows.

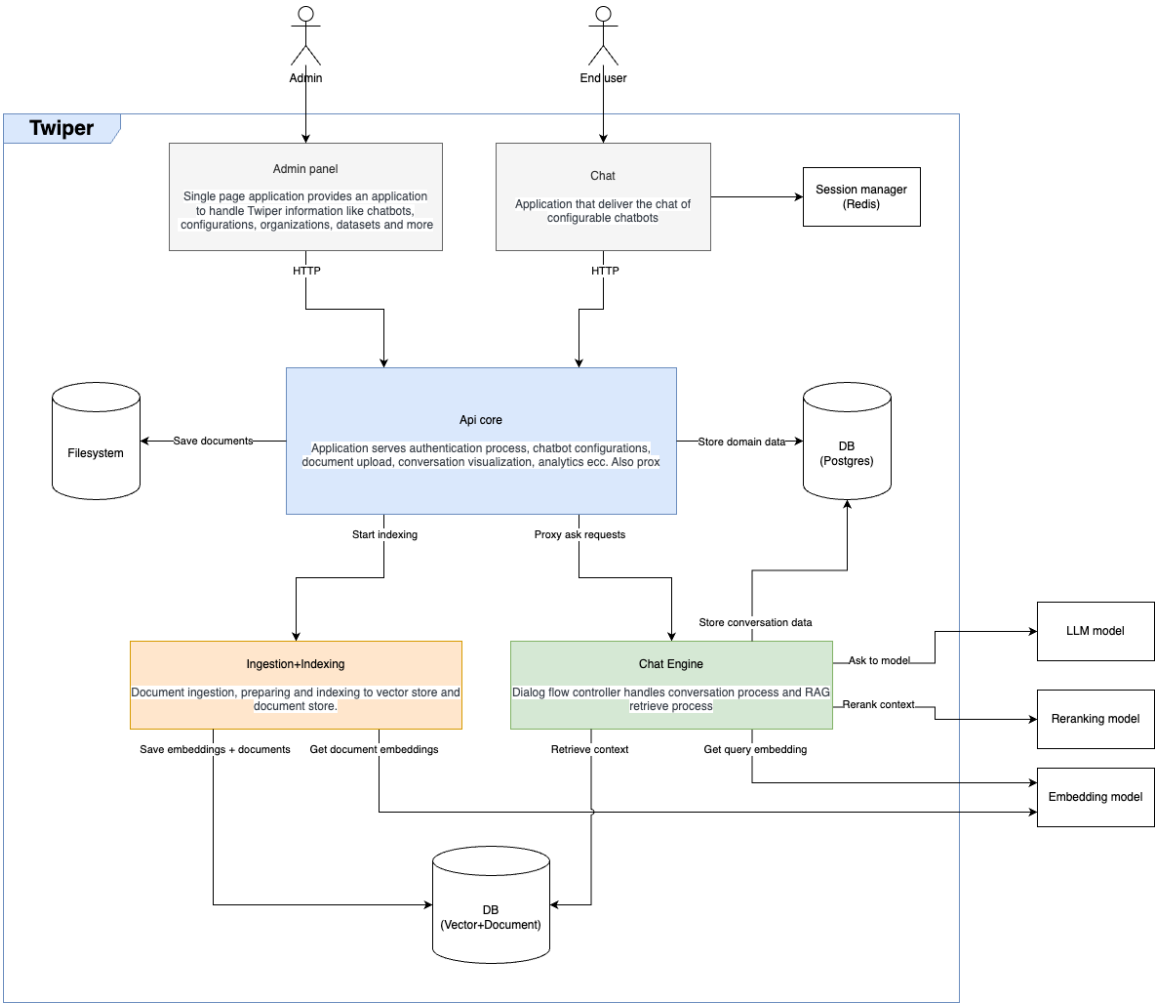


Figure 2.1 – High-level architecture of the Kora.ai platform and main data flows between clients, API Core, RAG/ML service, and storage layers.

2.2 Observability Boundaries and Anti-Leakage Motivation

A key methodological aspect of this thesis is the distinction between signals available *before* a handoff is executed and signals that only become available *after* the platform has persisted workflow outcomes. At runtime, the ML module can observe the current user context, retrieved passages, and agent configuration metadata available at the decision time,

whereas post-hoc artefacts (e.g., operator confirmations/actions or downstream ticket outcomes) may appear later in logs.

This motivates the strict pre-event feature construction and timestamp-based filtering adopted in Chapter 3, aiming to avoid temporal leakage—a common pitfall in data mining where information from the future inadvertently reaches the model [3].

2.3 Backoffice Platform

The Backoffice is a React/TypeScript dashboard designed to support non-technical administrators in configuring and monitoring the platform. It provides a single-page interface to create and manage chatbots, define agent dependencies, upload knowledge documents, and inspect operational behavior. By centralizing configuration tasks in a graphical interface, the Backoffice reduces the overhead of deploying and iterating on conversational agents.

Architecturally, the Backoffice is stateless and relies entirely on the API Core for data persistence and business logic. Administrators can modify chatbot parameters, upload datasets, and define routing rules without direct access to the underlying infrastructure. The Backoffice also supports organization-level settings, enabling multiple chatbots and use cases within the same platform instance.

In this thesis, the Backoffice matters because it defines (i) the knowledge corpus available for retrieval and (ii) the configuration constraints that define the space of possible routing and handoff options. Documents uploaded in the Backoffice become the retrieval knowledge base. In addition, agent metadata and configuration defined in the Backoffice determine which agents are eligible and which routing choices the machine learning modules introduced later can recommend. In particular, Backoffice configuration determines the set of eligible agents per chatbot, their activation rules, and the operational identifiers that later appear as structured context during dataset construction.

2.4 API Core

The API Core is the central gateway and orchestration service. It exposes HTTP endpoints to client applications, handles authentication and authorization, persists conversation state, and provides access to operational logs and metadata. In addition, it coordinates the conversation lifecycle: it stores incoming user messages, associates them with the correct organization/chatbot context, and forwards inference requests to the RAG/ML service.

From an engineering perspective, the API Core is responsible for ensuring traceability and auditable storage. It records system outputs and relevant metadata (e.g., trace events, agent transitions, and retrieval diagnostics), which later enable offline reconstruction of decision points. Concretely, it acts as the system of record for time-stamped conversational events, making it possible to align message histories with structured traces on a common time axis. These persisted records form the empirical basis for the datasets used in this thesis (Chapter 3).

2.5 RAG and ML

The RAG module is a Python-based component that implements a Retrieval-Augmented Generation pipeline [7].

It receives conversation requests from the API Core, retrieves relevant knowledge from the vector database, assembles context, and produces responses using a configured language model.

The ML service manages agent-level workflow logic: it tracks the active agent state, applies reasoning steps, and determines whether a handoff or tool invocation is required, depending on configuration. Importantly, the RAG/ML service does not persist long-term state: conversation history and operational records are stored by the API Core, while the databases provide retrieval and logging. This separation is important for leakage prevention: models deployed in the ML module must rely only on pre-handoff information available at the decision timestamp (e.g., user messages, retrieved context, and agent metadata), excluding post-handoff signals such as operator confirmations/actions or downstream

workflow outcomes.

From the research perspective of this thesis, the ML service is the main integration point. During deployment, the ML module operates in suggestion-only mode: it recommends a target agent, while the final transfer remains validated by an operator, following a Human-in-the-loop (HITL) approach to ensure reliability and trust [2].

2.6 Database Architecture

Kora.ai relies on PostgreSQL as the primary relational database for application data, and on a specialized vector store for similarity search. The relational database stores structured entities such as users, organizations, chatbots, agents, conversations, messages, and logs. This data provides the operational backbone of the system and supports accountability and traceability of interactions.

A separate vector database stores embeddings and retrieval metadata. Documents uploaded through the Backoffice are processed by the ingestion pipeline, segmented, and transformed into vector representations using semantic embedding models. These vectors are stored alongside document identifiers and metadata to enable efficient similarity search during inference.

Conceptually, the platform maintains two logical databases: (i) a transactional store for application state and conversation history, and (ii) a retrieval store optimized for nearest neighbor search. The separation supports different scaling requirements: high-consistency operations in PostgreSQL and high-throughput retrieval in the vector store.

For this thesis, database design is particularly relevant because message logs and trace events provide the primary empirical evidence for training and evaluation. The structure and timing of logged artefacts determine which signals can be extracted safely and how decision points can be reconstructed without temporal leakage (Chapter 3).

2.7 Conversational Flow

A conversation in Kora.ai follows a structured processing flow. First, an end-user message is submitted via the Chat UI and received by the API Core. The API Core authenticates the request, stores the message, and identifies the active conversational agent. It then forwards the request to the RAG/ML service for processing.

Within the RAG/ML service, the system performs retrieval over the vector database, assembles the relevant context, and generates a response using the configured language model. If reranking is enabled, retrieved documents are scored before being included in the final context. Based on the conversation state, the service may also decide to invoke tools or trigger an agent handoff.

Once a response is produced, the API Core records the output and associated metadata, such as retrieval results, trace events, and agent transitions, and returns the response to the Chat UI. All intermediate events are logged, enabling post-hoc analysis and offline evaluation of system behavior. In this work, handoff events are treated as explicit decision points: each event defines a time index at which a model can be evaluated using only the prefix of the conversation available before the transition.

2.8 Handoff Mechanisms

In Kora.ai, a *handoff* denotes a transition between agents (or conversational roles) during a dialogue. Handoffs can be triggered by static rules (e.g., keyword- or intent-based routing), by tool-based transfers (e.g., delegating a subtask to a specialized workflow), or by learned recommendations based on historical data. The objective is to route the conversation to the agent most appropriate for the user's current intent.

Static handoffs are deterministic and easy to audit, but may lack flexibility in ambiguous or multi-domain conversations. Tool-based transfers support structured workflows that temporarily delegate an action and then return control. Machine-learning-based handoffs—the focus of this thesis—aim to predict the best target agent dynamically from the pre-handoff context and observed outcomes in historical traces.

In the supervised formulation adopted here, each logged handoff event provides a target agent identifier that serves as the label for Module B. The model is evaluated using only the pre-handoff conversation prefix, mirroring the information available at decision time. During deployment, the ML module operates in suggestion-only mode: it recommends a target agent, while the final transfer remains validated by an operator.

The platform records handoff events, including source agent, target agent, and (when available) a rationale. These trace logs provide supervision for the handoff recommendation task and enable offline reconstruction of decision points under strict pre-event constraints (Chapter 3).

Chapter 3

Dataset and Problem Definition

This chapter describes the production data used in this work and formalizes the two machine learning problems addressed in the thesis: semantic auto-tagging (Module A) and handoff recommendation (Module B). The two modules share the same underlying production sources, but differ in their unit of analysis and in the methodological constraints required to avoid temporal leakage.

3.1 Data Sources

Both datasets are extracted from the production PostgreSQL database underlying the Kora.ai platform. The core tables involved are:

- **`conversations`:** one row per conversation, including identifiers (`conversation_id`, `organization_id`, `chatbot_id`, `area_id`) and operational metadata (e.g., the currently active agent).
- **`cnv_messages`:** message-level logs containing dialogue content, author role `user` /`assistant`/`system` , and timestamps.
- **`cnv_conversation_metrics`:** conversation-level annotations and platform signals (e.g., `topics`, summaries). Since metrics can be updated over time, exports deterministically select one record per conversation to ensure reproducible supervision.

- `cnv_conversation_traces`: trace events describing structured system actions during the dialogue. In this thesis, traces of type `agent_transfer` provide supervision for Module B.
- `cbt_agents`: agent registry used to validate that transfer events reference agents compatible with the originating chatbot.

All sources are joined into flat, reproducible CSV datasets to enable offline experimentation, model training, and auditability of preprocessing steps.

3.2 Dataset Construction

Two datasets are constructed, each aligned with one module of the thesis.

3.2.1 Module A dataset: conversation-level export

For auto-tagging, the unit of analysis is a **conversation**. The export produces one row per `conversation_id` and aggregates message logs in chronological order. Three text views are materialized:

- `conversation_text`: full transcript with role prefixes, retained for debugging and qualitative inspection.
- `recent_conv_text`: last *N* messages across all roles (chronological), used for inspection and error analysis.
- `recent_user_text`: last *N* *user-only* messages (chronological, no role prefixes). This is the primary model input for Module A.

Choice of user-only input and deterministic cleaning. Using only user-authored text is a deliberate design choice to reduce shortcut learning. Assistant messages may contain workflow templates or confirmations (e.g., ticket creation acknowledgements) that can inflate offline performance without improving real generalization.

After export, a deterministic preprocessing step cleans `recent_user_text` by removing recurrent noise patterns observed in production logs, including:

- JSON-like payloads or fenced blocks (e.g., "‘`json ...  "‘` or short `{...}` objects with fields such as `title/description`), typically produced by internal workflows rather than genuine user intent;
- confirmation-only replies (e.g., “sì”, “ok”, “confermo”) with little semantic content;
- system-like phrases associated with workflow steps (e.g., “procedo con la creazione”, “ecco un’anteprima”).

In addition, near-identical consecutive lines are collapsed to reduce repetition. This rigorous approach to data quality reflects the growing emphasis in data-centric AI on the importance of dataset understanding and curation for production performance. [8]

Cleaning impact. We quantify cleaning effects by tracking: (i) the fraction of conversations whose `recent_user_text` is modified, (ii) changes in the character-length distribution, and (iii) the number of rows that become empty after cleaning. These checks ensure that noise removal does not systematically erase informative user intent.

Dataset size and export summary. Running the Module A export on the production snapshot yields 407 conversation rows. After deterministic cleaning, one conversation has an empty `recent_user_text` input and is dropped; therefore, downstream analyses and experiments operate on 406 usable conversations. A non-trivial fraction of conversations has incomplete or missing topic annotations (all-zero rows), reflecting the noisiness of operational labels.

Conversation-level supervision. The raw supervision signal for topics is extracted from `cnv_conversation_metrics.topics` and stored as `topics_raw`. Because `cnv_conversation_metric` may contain multiple records per conversation, the export selects the *latest* entry per `conversation_id` using a deterministic ordering on `updated_at`, `created_at`, and a tie-breaker, ensuring reproducible labels.

Label timing (post-hoc supervision). Operational annotations are not necessarily available at message time and may be updated after the conversation concludes. Accordingly, Module A targets *offline enrichment* of completed conversations rather than real-time prediction at each turn. This is consistent with selecting a single, deterministic metrics record per conversation.

3.2.2 Context window size selection (N)

The auto-tagging input is defined as the last N user messages. The choice of N trades off semantic coverage and noise: a small window may miss relevant context, while a large window increases redundancy and the probability of including irrelevant or workflow-related content.

To guide this choice, three alternatives ($N = 3, 6, 10$) are generated and compared using simple statistics on `recent_user_text`. As shown in Table 3.1, moving from $N = 3$ to $N = 6$ increases the 90th percentile of input length substantially, while $N = 10$ provides only marginal additional gain. Based on this coverage/complexity trade-off, $N = 6$ is used as the default configuration in the experiments.

Window (N)	Empty rows	Median chars	p90 chars
3	3	66	153
6	1	67	208
10	1	70	211

Table 3.1 – Input length statistics for the `recent_user_text` field at different window sizes.

3.2.3 Module B dataset: event-based pre-handoff export (anti-leakage)

For handoff recommendation, the unit of analysis is a **handoff event**. The export produces one row per `agent_transfer` trace, i.e., one row per decision point at which the system transfers a conversation from one agent to another. Each event is associated with a timestamp t defined by the trace creation time `trace_created_at`.

Strict pre-event text definition. To prevent temporal leakage, the pre-handoff input text is defined as:

$$\texttt{conversation\_text\_pre}(t) = \{m \in \texttt{cnv\_messages} \mid m.\texttt{role} = \texttt{user} \wedge m.\texttt{created\_at} < t\}.$$

That is, only user messages *strictly before* the transfer event are included. The strict inequality avoids ambiguous boundary cases and prevents post-event information (e.g., transfer confirmations or workflow outcomes) from contaminating the feature set.

Export summary and feature safety. Running the Module B export yields 404 handoff-event rows. After applying eligibility and quality filters (Section 5.4), the usable dataset contains 397 events: 313 for chatbot 1 and 84 for chatbot 2.

Each row contains structured fields observable before the event. Some are safe by construction (e.g., `message_count_pre`, `transfer_index`, `n_prev_transfers`), while others (notably `active_agent_id_pre`) can encode near-deterministic routing policies and are therefore treated carefully in Module B (used for baselines/guards but excluded from the ML feature matrix). The label is the target agent identifier `target_new_agent_id`.

The pipeline writes both (i) a full CSV including audit columns for traceability and (ii) a strict training CSV retaining only whitelisted features and the label. Audit fields (e.g., `__audit_trace_id`, `__audit_trace_created_at`) are retained for debugging and dataset reconstruction but are **explicitly excluded** from the model feature matrix.

Validation constraints. To ensure consistency, transfer events are filtered so that both the old and new agents belong to the same chatbot as the conversation (validated via joins on `cbt_agents`). Events are discarded if the target agent is missing, if agent–chatbot compatibility checks fail, or if the pre-handoff user context is empty (`message_count_pre`=0).

Train/test splitting (conversation-level grouping). To avoid information leakage across multiple events from the same dialogue, data splits are performed by grouping on `conversation_id`, ensuring that all handoff events from a given conversation appear in only one split.

3.3 Agent Distribution

Module A is trained at conversation level and does not directly depend on the distribution of handoff events. Module B, instead, is inherently constrained by the empirical distribution of `agent_transfer` traces. In practice, production systems are typically imbalanced: a small number of dominant agents handle most traffic, while specialized agents receive fewer transfers. This power-law distribution is a known challenge in real-world recommender systems [9], where learning from imbalanced data requires specific strategies to prevent the model from ignoring minority classes [10].

This imbalance has two implications. First, rare agents provide limited supervision and may be unsuitable as targets for supervised learning without additional data, grouping, or hierarchical labeling. Second, evaluation should consider ranking-based metrics (e.g., Top-K accuracy and Average Precision (AP)) in addition to plain accuracy, because returning a shortlist of plausible agents can be operationally useful even when the top-1 decision is uncertain.

Accordingly, the recommender models in this thesis focus on agents with sufficient support and on transfer events that satisfy strict consistency constraints, so that both training and evaluation reflect realistic production decision points. When eligibility filtering reduces the effective number of classes (e.g., $C = 2$), Top-K metrics for large K become less informative; in these cases we emphasize Top-1 performance and minority-class ranking diagnostics.

3.4 Semantic Embeddings of Conversations

Both modules rely on a shared semantic representation of text. Each input is converted into a dense embedding vector using the OpenAI `text-embedding-3-small` model, producing a 1536-dimensional representation. Inputs are truncated when necessary to bound compute cost and ensure consistent behavior across runs.

These embeddings capture semantic similarity beyond surface keywords and provide a compact feature space suitable for downstream modeling. The choice of this specific

architecture is motivated by the effectiveness of contrastive pre-training for learning dense representations, as demonstrated by Neelakantan et al. [11]. This approach enables the model to map diverse user-authored texts into a latent space where semantic proximity reflects functional and operational relatedness, even in the absence of exact keyword overlap.

For Module A, embeddings are computed from `recent_user_text` (last N user messages). For Module B, embeddings are computed from `conversation_text_pre`, i.e., the user context available at the handoff decision point.

To ensure reproducibility and reduce compute cost, embeddings are cached to disk together with validation metadata. The cache is reused only when model identifier (and fingerprint/revision, when available), embedding dimensionality, number of rows, and dataset identity match the current export. Dataset identity is checked by computing a stable hash over the ordered row IDs and the preprocessed input text. If this hash changes, the cache is discarded and embeddings are recomputed.

3.5 Final Topic Definition for Auto-Tagging

The raw supervision signal `topics_raw` is derived from operational annotations and is not designed as an ML label space: it is fragmented, partially inconsistent, and includes many rare labels. Training directly on `topics_raw` would therefore yield an unstable multi-label problem under the available dataset size.

For this reason, raw topics are mapped into a smaller set of deterministic *macro-topics* to reduce sparsity and improve interpretability. The macro-topic set, mapping rules, and validation diagnostics are presented in Chapter 4.

3.6 Problem A: Auto-Tagging

Module A is formulated as a **multi-label classification** problem at conversation level. Given an input conversation x (user-only, last N messages) represented as an embedding

vector, the model predicts a set of macro-topics with independent probabilities:

$$f_A(x) \to \{(y_i, p_i)\}_{i=1}^{L},$$

where L is the number of macro-topics, $y_i \in \{0,1\}$ indicates label presence, and $p_i \in [0,1]$ is the predicted probability.

3.7 Problem B: Handoff Recommendation

Module B is formulated as a **multi-class ranking** problem at handoff-event level. Given the pre-handoff context available at time t (user messages strictly before the transfer) and a set of explicitly whitelisted pre-event structured variables, the model estimates a score for each candidate agent and produces a Top-K recommendation list:

$$f_B(x_t) \to \text{ranked list of agents.}$$

Framing the handoff as a ranking problem rather than a rigid classification allows the system to support human operators with a shortlist of candidates, as per standard Recommender Systems frameworks [9]. Supervision is provided by the historical outcome `target_new_agent_id`. Unlike Module A, the dataset is event-based: a single conversation can contribute multiple training instances if multiple transfers occur.

A central methodological requirement is **anti-leakage evaluation**: features must be constructed only from information observable before the transfer decision point. Anti-leakage enforcement is implemented at extraction time (Section 3.2.3) through strict timestamp filtering and a hard feature whitelist; audit identifiers and any post-event fields are excluded by construction. By enforcing strict timestamp filtering and feature whitelisting, we adhere to the formal definition of leakage prevention in data mining, ensuring that no information from the future influences the model's decision at time t [3].

3.8 Dataset Limitations

The datasets reflect realistic constraints of production logs and limited annotation.

- **Small sample size.** Both modules are trained on a few hundred examples, increasing variance across random splits and making rare classes fragile.
- **Imbalance.** Macro-topics and especially target agents can be highly imbalanced, limiting performance on rare labels.
- **Noisy supervision.** Operational topics are not designed as ML labels and may include inconsistencies, duplicates, and overly broad categories.
- **Redundancy and near-duplicates.** Recurring requests and templates can lead to highly similar examples across splits, potentially producing optimistic estimates if not controlled.
- **Temporal leakage risk.** Conversation logs include workflow artefacts and confirmations that can leak future outcomes. This is mitigated by user-only text for Module A and strict pre-event filtering plus feature whitelisting for Module B, but remains a key evaluation concern.

Chapter 4

Auto-Tagging

This chapter presents the semantic auto-tagging module developed to enrich Kora.ai conversations with structured labels. These labels provide compact, reusable semantic signals that are later consumed as additional features by the handoff recommendation module (Part B). The module is designed for production constraints: limited annotated data, noisy operational labels, and a strong requirement to avoid leakage from assistant templates.

4.1 Objective and Tasks

The auto-tagging module operates at *the conversation level* to interpret structured semantic signals from user input. The objective is formulated as a **multi-label prediction task** where, given the conversation text, the model simultaneously identifies a set of relevant *macro-topics*, providing a probability score for each.

This multi-label formulation is necessary because real-world user requests often span multiple functional areas. For example, a single interaction might involve VAT processing (`FISCO_IVA`), inquiries about electronic invoicing (`DOCUMENTI_FATTURAZIONE`), and an explicit intent to open a support ticket (`TICKET`).

By predicting these components together rather than forcing a single-label assignment, the module preserves the full semantic complexity of the request. This ensures that the derived tags serve as rich, high-fidelity features for the downstream handoff recommendation module, avoiding the loss of relevant information that would occur with a mutually

exclusive classification.

4.2 Input Data and Export Format

Training and evaluation use an offline dataset exported from production logs (Chapter 3). Each row corresponds to a unique conversation and includes: (i) the main user-authored text input, (ii) supervision signals derived from operational annotations, and (iii) lightweight metadata used for reproducibility and analysis.

Main text input. The production auto-tagger operates on `recent_user_text`, defined as the concatenation of the last N user messages (chronological order, no role prefixes). Additional text views (`recent_conv_text`, `conversation_text`) are retained only for qualitative inspection and error analysis, as described in Chapter 3.

Dataset size. The exported dataset contains 407 conversations; after cleaning and dropping the single empty-input row, all results and tables in this chapter refer to 406 usable conversations.

4.3 Semantic Representation and Context Window

The auto-tagger operates on dense semantic embeddings of `recent_user_text`. The context window size is set to $N = 6$ user messages, selected using the coverage analysis reported in Chapter 3 (Table 3.1). This choice balances semantic coverage with input conciseness and reduces the likelihood of including workflow-related noise.

4.4 From Raw Topics to Macro-Topics

The dataset includes a `topics_raw` signal derived from operational annotations entered by humans. While valuable as supervision, raw topics are typically *too fragmented* and *too sparse* to serve as stable targets for supervised learning in a small dataset. As highlighted

by Frénay and Verleysen [4], label noise and high cardinality in real-world datasets can severely degrade model performance; in our case, many labels occur only a handful of times, and semantically similar labels are often split across multiple variants. A direct multi-label model over such a poorly conditioned raw topic space would be difficult to train and evaluate reliably.

To obtain a compact label space with clearer semantics, raw topics are mapped into a smaller set of macro-topics using deterministic rules. This aggregation strategy serves to mitigate the impact of annotation noise by (i) reducing label cardinality to a tractable number of targets, and (ii) improving label consistency by grouping synonyms and closely related operational categories into stable semantic units.

Why cosine distance and Jaccard similarity are used. We use cosine *distance* in embedding space as an indirect measure for semantic cohesion (within a macro-topic) and separability (between macro-topics). We use Jaccard similarity and conditional overlap $P(B \mid A)$ to quantify label co-occurrence and raw-topic ambiguity independently of text semantics.

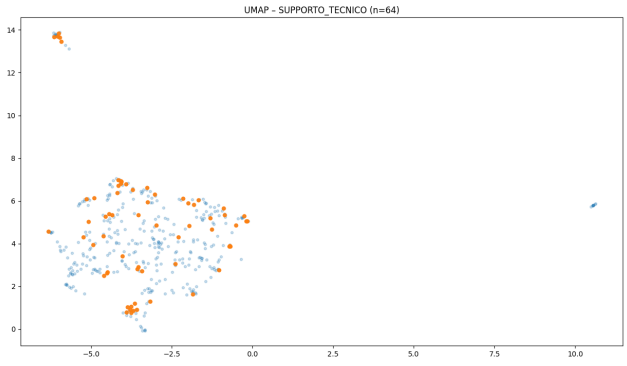
4.4.1 Qualitative validation with UMAP

UMAP projections are used as a qualitative sanity check. As shown by McInnes et al. [12], UMAP is effective at preserving local neighborhood structure and can provide useful low-dimensional visualizations of high-dimensional embeddings.

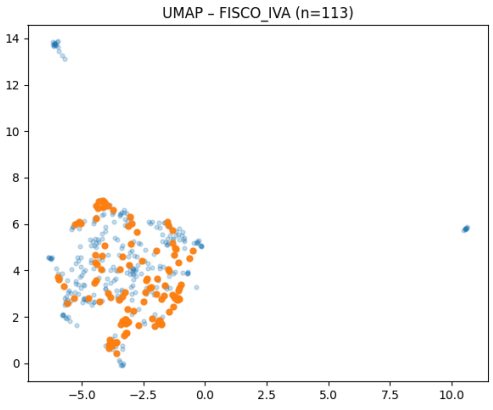
Although this visualization is not used for any quantitative claim, it provides an intuitive check that several macro-topics form coherent neighborhoods in embedding space, while boundary regions tend to concentrate mixed or ambiguous cases. This qualitative evidence supports the macro-topic mapping choices and motivates the quantitative cohesion/separability diagnostics reported in Section 4.4.2.

Macro-topic set. The final macro-topics used as supervised targets are:

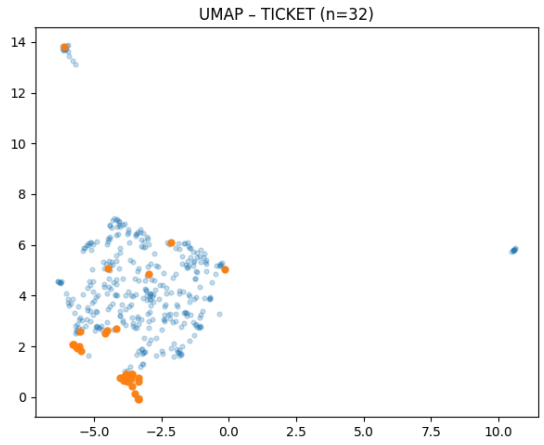
- `FISCO_IVA` (tax/VAT)



(a) SUPPORTO_TECNICO



(b) FISCO_IVA



(c) TICKET

Figure 4.1 – UMAP projections of conversation embeddings (visualization only).

- ANAGRAFICHE (master data: clients, suppliers, entities)
- DOCUMENTI_FATTURAZIONE (documents and invoicing)
- CONTABILITA (accounting)
- MAGAZZINO_ORDINI (warehouse and orders)
- TICKET (explicit ticket creation intent)
- IMPORTAZIONI_INTEGRAZIONI (imports/exports/integrations)
- SUPPORTO_TECNICO (technical support)
- RISORSE (manuals, webinars, documentation resources)

A fallback label, ALTRO, is retained in the dataset but treated as an *absorption class* rather than a semantic target. It collects rare, ambiguous, multi-domain, or noisy requests and serves as a low-confidence indicator; accordingly, ALTRO is not included among supervised targets for the production auto-tagger.

Separating configuration from support. A key labeling choice is to keep configuration requests distinct from generic technical support at the *raw-topic mapping* level. Configuration signals refer to setup and parameterization (permissions, SMTP, feature flags, connectivity settings) and are prevented from being absorbed into SUPPORTO_TECNICO. In the final macro-topic set we focus on the targets listed above; configuration-related requests are either mapped into the most relevant operational macro-topic or retained for analysis outside the supervised target set, depending on support.

Topic prevalence. On the usable dataset ($n = 406$ conversations; with context window $N = 6$), macro-topic prevalence is:

Co-occurrence analysis. Because conversations can span multiple domains, label co-occurrence is expected and supports the multi-label formulation. Empirically, several macro-topics overlap (Table 4.3). For example, DOCUMENTI_FATTURAZIONE co-occurs

Table 4.1 – Macro-topic distribution among conversations.

Macro-topic	# Conversations
FISCO_IVA	113
CONTABILITA	90
DOCUMENTI_FATTURAZIONE	74
ANAGRAFICHE	68
SUPPORTO_TECNICO	64
MAGAZZINO_ORDINI	57
RISORSE	37
ALTRO	33
TICKET	32
IMPORTAZIONI_INTEGRAZIONI	22

with `SUPPORTO_TECNICO` (Jaccard ≈ 0.15), and `FISCO_IVA` co-occurs with `CONTABILITA` (Jaccard ≈ 0.15). These overlaps suggest that a single-label formulation would systematically discard useful semantic information.

4.4.2 Quantitative diagnostics for macro-topic design

Raw-topic overlap and aggregation evidence. Residual overlap between macro-topics is analyzed using conditional overlap $P(B \mid A)$ and Jaccard similarity. Table 4.2 reports the strongest directional overlaps, while Table 4.3 reports the strongest normalized co-occurrences. Jaccard values remain low (max ≈ 0.15), indicating limited redundancy between macro-topics. Observed conditional overlaps reflect common multi-domain workflows rather than semantic equivalence.

Purity (exclusive rate). To quantify how often a macro-topic behaves as a "standalone" label versus a cross-domain tag, we compute an exclusive rate: the fraction of conversations where the macro-topic appears without any other macro-topic label. Higher values indicate "purer" domain labels, while lower values suggest cross-domain intents/modifiers.

Semantic cohesion (intra-macro distance). We measure semantic cohesion using the mean cosine distance among conversation embeddings within each macro-topic (lower is more cohesive). We report both the overall intra-macro distance and an exclusive-only

Pair	$P(B \mid A)$	Intersection / $\lvert A \rvert$
$P(\texttt{ANAGRAFICHE} \mid \texttt{TICKET})$	0.39	13 / 32
$P(\texttt{DOCUMENTI\_FATTURAZIONE} \mid \texttt{IMPORTAZIONI\_INTEGRAZIONI})$	0.32	7 / 22
$P(\texttt{FISCO\_IVA} \mid \texttt{CONTABILITA})$	0.29	26 / 90
$P(\texttt{FISCO\_IVA} \mid \texttt{SUPPORTO\_TECNICO})$	0.28	18 / 64
$P(\texttt{DOCUMENTI\_FATTURAZIONE} \mid \texttt{SUPPORTO\_TECNICO})$	0.28	18 / 64
$P(\texttt{SUPPORTO\_TECNICO} \mid \texttt{IMPORTAZIONI\_INTEGRAZIONI})$	0.27	6 / 22
$P(\texttt{FISCO\_IVA} \mid \texttt{DOCUMENTI\_FATTURAZIONE})$	0.26	19 / 74
$P(\texttt{SUPPORTO\_TECNICO} \mid \texttt{DOCUMENTI\_FATTURAZIONE})$	0.24	18 / 74
$P(\texttt{SUPPORTO\_TECNICO} \mid \texttt{TICKET})$	0.24	8 / 32
$P(\texttt{FISCO\_IVA} \mid \texttt{TICKET})$	0.24	8 / 32

Table 4.2 – Top conditional overlaps $P(B \mid A)$ computed over macro-topic labels. Values highlight directional co-occurrence patterns rather than semantic equivalence.

Pair	Jaccard	Intersection	Union
(DOCUMENTI_FATTURAZIONE, SUPPORTO_TECNICO)	0.15	18	120
(ANAGRAFICHE, TICKET)	0.15	13	88
(FISCO_IVA, CONTABILITA)	0.15	26	178
(FISCO_IVA, SUPPORTO_TECNICO)	0.11	18	160
(FISCO_IVA, DOCUMENTI_FATTURAZIONE)	0.11	19	169
(TICKET, SUPPORTO_TECNICO)	0.09	8	89
(DOCUMENTI_FATTURAZIONE, CONTABILITA)	0.09	13	151
(DOCUMENTI_FATTURAZIONE, IMPORTAZIONI_INTEGRAZIONI)	0.08	7	89
(CONTABILITA, SUPPORTO_TECNICO)	0.08	11	143
(SUPPORTO_TECNICO, IMPORTAZIONI_INTEGRAZIONI)	0.07	6	80

Table 4.3 – Top Jaccard similarities between macro-topics. Jaccard captures normalized co-occurrence and should not be interpreted as semantic equivalence.

variant computed on conversations where the macro-topic appears alone.

Interpretation and caveats. Exclusive-rate and cohesion diagnostics suggest that some macro-topics behave as relatively "pure" domain labels (e.g., MAGAZZINO_ORDINI, RISORSE), while others behave as cross-domain intents or modifiers (notably SUPPORTO_TECNICO and TICKET), which naturally co-occur with multiple domains. Cohesion scores are interpreted comparatively (relative cohesion across macro-topics) rather than as absolute thresholds. Exclusive-only cohesion for very small $n_{\text{exclusive}}$ (e.g., TICKET, IMPORTAZIONI_INTEGRAZIONI) should be interpreted cautiously due to higher variance.

Macro-topic	n_{all}	$n_{\text{exclusive}}$	Exclusive rate
MAGAZZINO_ORDINI	57	37	0.649
RISORSE	37	21	0.568
CONTABILITA	90	38	0.422
FISCO_IVA	113	43	0.381
ANAGRAFICHE	68	24	0.353
DOCUMENTI_FATTURAZIONE	74	22	0.297
SUPPORTO_TECNICO	64	11	0.172
TICKET	32	5	0.156
IMPORTAZIONI_INTEGRAZIONI	22	3	0.136

Table 4.4 – Exclusive-rate per macro-topic (purity proxy).

Macro-topic	n	Intra mean cosine distance
TICKET	32	0.478
ANAGRAFICHE	68	0.491
MAGAZZINO_ORDINI	57	0.542
RISORSE	37	0.542
FISCO_IVA	113	0.564
CONTABILITA	90	0.571
IMPORTAZIONI_INTEGRAZIONI	22	0.576
DOCUMENTI_FATTURAZIONE	74	0.579
SUPPORTO_TECNICO	64	0.587

Table 4.5 – Intra-macro mean cosine distance (all labeled conversations). Lower values indicate higher semantic cohesion.

Inter-macro separability (centroid distances). To characterize how separated macro-topics are in embedding space, we compute cosine distances between macro-topic centroids (higher means more separated).

4.5 Model Architecture

The module is designed as a multi-label classification system. Specifically, we employ a Binary Relevance transformation, which decomposes the multi-label task into independent binary problems—one for each macro-topic. This remains a foundational and robust strategy in multi-label learning literature [13]. The auto-tagger is implemented as a lightweight linear model on top of semantic embeddings:

Macro-topic	$n_{\text{exclusive}}$	Intra mean cosine distance (exclusive)
ANAGRAFICHE	24	0.400
TICKET	5	0.449
RISORSE	21	0.518
MAGAZZINO_ORDINI	37	0.522
FISCO_IVA	43	0.586
CONTABILITA	38	0.600
DOCUMENTI_FATTURAZIONE	22	0.617
IMPORTAZIONI_INTEGRAZIONI	3	0.624
SUPPORTO_TECNICO	11	0.678

Table 4.6 – Intra-macro mean cosine distance on exclusive-only conversations (semantic "core" proxy).

	F-IVA	**ANAG**	**DOC-F**	**CONT**	**MAG**	**TICK**	**IMP**	**SUPP**	**RIS**
FISCO_IVA	0.000	0.089	0.079	0.032	0.077	0.132	0.107	0.039	0.115
ANAGRAFICHE	0.089	0.000	0.123	0.090	0.095	0.114	0.114	0.093	0.154
DOCUMENTI_FATT.	0.079	0.123	0.000	0.063	0.093	0.172	0.071	0.042	0.113
CONTABILITA	0.032	0.090	0.063	0.000	0.054	0.155	0.087	0.034	0.108
MAGAZZINO_ORDINI	0.077	0.095	0.093	0.054	0.000	0.189	0.108	0.074	0.149
TICKET	0.132	0.114	0.172	0.155	0.189	0.000	0.195	0.117	0.177
IMPORTAZIONI_INT.	0.107	0.114	0.071	0.087	0.108	0.195	0.000	0.075	0.137
SUPPORTO_TECNICO	0.039	0.093	0.042	0.034	0.074	0.117	0.075	0.000	0.069
RISORSE	0.115	0.154	0.113	0.108	0.149	0.177	0.137	0.069	0.000

Table 4.7 – Inter-macro cosine distances between macro-topic centroids. Column abbreviations: F-IVA (Fisco IVA), ANAG (Anagrafiche), DOC-F (Documenti Fatturazione), CONT (Contabilità), MAG (Magazzino), TICK (Ticket), IMP (Importazioni), SUPP (Supporto Tecnico), RIS (Risorse).

- **Multi-label head:** One-vs-Rest Logistic Regression, trained independently for each macro-topic (`class_weight=balanced`).

This architecture is intentionally simple: embeddings provide a strong semantic representation, while a linear classifier offers robustness on small datasets, fast iteration, and interpretability. Each macro-topic corresponds to a separate linear decision function in embedding space.

4.6 Training Procedure

Train/validation/test split. The training run uses a 70/15/15 split (train=284, validation=61, test=61) on a cleaned dataset of 406 usable rows. Split indices and hashes are recorded to ensure reproducibility across runs and to enable exact comparisons between experimental variants.

Class balance. The number of positives per class is monitored in each split to avoid degenerate conditions, especially for rarer topics (e.g., `IMPORTAZIONI_INTEGRAZIONI`, `TICKET`). For this split, positives per subset are:

Topic	Train pos	Val pos	Test pos
FISCO_IVA	79	19	15
ANAGRAFICHE	47	12	9
DOCUMENTI_FATTURAZIONE	50	13	11
CONTABILITA	67	10	13
MAGAZZINO_ORDINI	42	6	9
TICKET	22	5	5
IMPORTAZIONI_INTEGRAZIONI	15	4	3
SUPPORTO_TECNICO	45	10	9
RISORSE	25	7	5

Threshold tuning. For each macro-topic, a decision threshold is selected on the validation set to maximize F1. The tuned thresholds are then applied unchanged to the test set. In this run, the optimized thresholds are mostly close to 0.5, with modest deviations (e.g., `TICKET`: $t = 0.40$, `RISORSE`: $t = 0.60$), reflecting class-specific calibration differences.

Engineering for reproducibility. Embeddings are computed once for the full dataset and reused across splits and repeated runs. Model artifacts (weights, thresholds) and metrics reports are saved to disk to support auditability and replication.

4.7 Evaluation Metrics

The multi-label nature of the task requires metrics that capture both overall behavior and per-class performance:

- **Micro-F1:** aggregates decisions across all labels, emphasizing frequent classes and reflecting overall system behavior.
- **Macro-F1:** averages F1 across topics, weighting each topic equally and highlighting performance on rarer or more difficult classes.
- **Average Precision (AP):** computed per topic to assess ranking quality of predicted probabilities under class imbalance.

We report **Macro-AP** as the unweighted mean of per-topic Average Precision (AP) scores.

4.8 Results

Topic classification. On a representative single split (N6), test performance is:

$$\text{Macro-F1} = 0.695, \qquad \text{Micro-F1} = 0.649, \qquad \text{Macro-AP} = 0.792.$$

Note on small-sample interpretation. Some per-topic results may appear unusually strong (e.g., AP close to 1.0) despite limited test support. This is expected in a small-sample regime: when a topic has only a handful of positive instances in the test set (e.g., 3–5 positives), both F1 and AP have high variance and can change substantially by flipping only one or two predictions. In particular, AP can reach 1.0 if the few positive examples are ranked above all negatives, even when the model would not generalize equally well under a different split [14]. For this reason, per-topic results with low support should be interpreted as indicative rather than definitive, and are complemented by split-variability and redundancy diagnostics (Section 4.9). Moreover, a measurable fraction of the test set

is near-duplicate of training conversations in embedding space (Section 4.9), which can further inflate the perceived performance of a favorable split.

Per-topic test performance (with thresholds tuned on validation) is summarized in Table 4.8.

Topic	F1 (test)	AP (test)	Test positives
FISCO_IVA	0.882	0.987	15
ANAGRAFICHE	0.824	0.853	9
DOCUMENTI_FATTURAZIONE	0.700	0.785	11
CONTABILITA	0.714	0.691	13
MAGAZZINO_ORDINI	0.800	0.904	9
TICKET	0.526	1.000	5
IMPORTAZIONI_INTEGRAZIONI	0.800	0.767	3
SUPPORTO_TECNICO	0.211	0.327	9
RISORSE	0.800	0.834	5

Table 4.8 – Per-topic performance on the test set for the representative auto-tagging run (N6). Topics with fewer than 10 test positives should be interpreted with caution due to higher variance.

Performance is strong for well-defined semantic domains such as VAT, invoicing documents, and warehouse/orders. In contrast, `SUPPORTO_TECNICO` is the most challenging label, with low F1 and AP. This is consistent with its heterogeneous nature and its cross-domain usage (Tables 4.4–4.6).

Relation to split variability. The above results refer to a single realization of the train/-validation/test split. As discussed in Section 4.9, repeating the experiment across multiple random seeds yields lower mean scores with non-negligible variance, which is consistent with the small dataset size and the presence of rare topics. Therefore, single-split results should be interpreted together with the across-seed summary as an estimate of best-case vs. expected performance under different splits.

4.9 Quality and Validity Checks

True label leakage scan. A scan for direct label tokens in the user input reports zero cases (0/406, 0.00%), indicating that ground-truth labels are not trivially present in the training

text. The scan checks `recent_user_text` for exact matches of macro-topic labels and common raw-topic variants.

Keyword overlap. A keyword overlap check finds 191/406 conversations (47.04%) that include terms correlated with one or more labels. This is expected in an operational domain where users naturally mention domain keywords (e.g., "IVA", "XML", "F24", "ticket"). While keyword overlap can make some classes easier, it does not constitute leakage because these terms are genuine parts of user intent.

Assistant-template leakage mitigation. To minimize shortcuts driven by workflow confirmations, the model is trained exclusively on user-authored text (`recent_user_text`), as defined in Chapter 3. This removes the most direct source of template-driven leakage observed in production logs.

Near-duplicates across splits. A near-duplicate analysis identifies test conversations whose embedding cosine similarity (computed in the same `text-embedding-3-small` space) exceeds 0.95 with at least one training conversation. In this dataset, 19.67% of the test set meets this criterion (about 12 out of 61 test conversations). We use a high similarity threshold to capture only near-paraphrases and template-like repetitions, rather than broad semantic relatedness. This suggests a degree of redundancy (templates, repeated requests, or semantically equivalent questions) and may lead to mildly optimistic performance estimates compared to a fully independent test set. Future work can reduce this effect by ensuring that highly similar conversations are not split across training and test sets.

Sensitivity to random splits (across seeds). To estimate how sensitive the reported metrics are to the random split, the full training and evaluation procedure is repeated across three different seeds. Given the small dataset and rare topics, the variance is non-negligible and the average performance is lower than the best single split. Across seeds we observe:

$$\text{Macro-AP} = 0.630 \pm 0.040, \quad \text{Macro-F1} = 0.497 \pm 0.048, \quad \text{Micro-F1} = 0.513 \pm 0.046.$$

This confirms that single-split results can be optimistic in small-sample conditions and motivates reporting uncertainty alongside point estimates.

Check	Result
True label leakage	0/406 (0.00%)
Keyword overlap	191/406 (47.04%)
Near-duplicates (test, sim $>$ 0.95)	12/61 (19.67%)

Table 4.9 – Summary of validity checks for the auto-tagging dataset and split.

4.10 Qualitative Error Analysis: SUPPORTO_TECNICO

The poor performance on `SUPPORTO_TECNICO` (F1=0.211, AP=0.327, Table 4.8) motivates a qualitative inspection of failure modes. We analyzed false negatives (FN) and false positives (FP) to identify systematic error patterns.

4.10.1 Error Categories

1. Semantic overlap with functional topics (majority of FN). Requests involving technical issues with functional modules (e.g., invoice generation errors, PDF printing failures) are predicted as `DOCUMENTI_FATTURAZIONE` or `CONTABILITA` rather than `SUPPORTO_TECNICO`. The model prioritizes the functional domain over the "support" intent modifier.

Example FN:

> *"Errore nell'emissione della fattura elettronica"*
>
> Gold labels: {`DOCUMENTI_FATTURAZIONE`, `SUPPORTO_TECNICO`}
>
> Predicted: {`DOCUMENTI_FATTURAZIONE`} only
>
> Interpretation: Technical assistance on invoicing is semantically closer to the invoicing domain than to generic support.

This pattern is consistent with the low exclusivity rate (17.2%, Table 4.4) and high intra-macro distance (0.587, Table 4.5) of `SUPPORTO_TECNICO`: the topic frequently co-occurs with functional domains and lacks a cohesive semantic core.

2. Threshold sensitivity . Three false negatives have predicted probabilities just below the tuned threshold (margin $\in [-0.02, -0.04]$), suggesting that a more permissive threshold would increase recall at the cost of precision. This indicates that the decision boundary is reasonable but that some borderline cases are misclassified due to conservative thresholding.

3. Conflict with `TICKET` label (majority of FP). False positives frequently contain explicit escalation keywords ("aprire un ticket", "contattare assistenza", "escalation"), with `TICKET` often ranking among the top predicted classes. This indicates conceptual overlap: requests for support and requests to open tickets are semantically similar and may not be consistently distinguished in the operational annotation guidelines.

4.10.2 Implications

`SUPPORTO_TECNICO` errors stem primarily from **label ambiguity** and **cross-domain semantics** rather than model incapacity. The classifier reliably detects technical content, but it struggles to distinguish *support as a modifier of a functional domain* from *support as a standalone intent*.

Mitigation strategies.

- **Refine annotation guidelines**: Clarify the distinction between `SUPPORTO_TECNICO` (technical assistance on functional modules) and functional topics without support intent. Alternatively, introduce hierarchical labels: domain topic (primary) + support intent (modifier).
- **Collect boundary-case supervision**: Use active learning to identify ambiguous cases near the `SUPPORTO_TECNICO` decision boundary and request re-annotation or consensus labels.
- **Adjust threshold per topic**: Allow per-topic thresholds to reflect different class characteristics. For `SUPPORTO_TECNICO`, a lower threshold may improve recall without excessive precision loss.

Conclusion. While SUPPORTO_TECNICO achieves the lowest F1 among macro-topics, the error analysis suggests that failures are *interpretable and addressable* through improved annotation and threshold calibration, rather than fundamental model limitations.

4.11 Qualitative Error Analysis Beyond SUPPORTO_TECNICO

The qualitative inspection in Section 4.10 focuses on SUPPORTO_TECNICO, the most challenging macro-topic in this run. To reduce the risk of topic-specific conclusions, we extend the analysis to two mid-range classes with non-trivial support in the test set: DOCUMENTI_FATTURAZIONE (F1=0.700, 11 positives) and CONTABILITA (F1=0.714, 13 positives).

4.11.1 Sampling protocol and "fixable" vs. ambiguous coding

For each macro-topic c, we collect false negatives (FN) and false positives (FP) on the test set. To avoid cherry-picking, we rank errors by their probability margin with respect to the validation-tuned threshold t_c (Section 4.6):

$$\Delta_c(x) = p_c(x) - t_c.$$

FN cases with $\Delta_c(x) \approx 0^-$ are treated as borderline threshold errors, while larger negative margins suggest semantic mismatch or label ambiguity. FP cases with $\Delta_c(x) \approx 0^+$ indicate over-triggering near the decision boundary.

Each error is coded as either (i) *fixable* (threshold/calibration or guideline/mapping refinements) or (ii) *intrinsically ambiguous* (semantic overlap or under-specified requests from text alone).

4.11.2 Error Analysis: DOCUMENTI_FATTURAZIONE

Pattern 1: borderline misses suggest threshold sensitivity. Several FN cases for DOCUMENTI_FATTURAZIONE lie just below the tuned threshold ($t = 0.50$), indicating that a small calibration/threshold adjustment could increase recall with limited precision loss.

Example FN (borderline, support intent present).

"lo sdi non riconosce il cap"

Gold labels: {DOCUMENTI_FATTURAZIONE, SUPPORTO_TECNICO}

Predicted: {SUPPORTO_TECNICO}

$p_{\texttt{DOC\_F}}(x) = 0.491$, $t_{\texttt{DOC\_F}} = 0.50$, $\Delta = -0.009$.

Example FN (borderline, single-label target).

"le note dell'xml quanti caratteri possono avere?"

Gold labels: {DOCUMENTI_FATTURAZIONE}

Predicted: {} (no macro-topic above threshold)

$p_{\texttt{DOC\_F}}(x) = 0.488$, $t_{\texttt{DOC\_F}} = 0.50$, $\Delta = -0.012$.

Pattern 2: confusion with adjacent operational domains. In a subset of FN cases, document-related questions are expressed using operational language that is also common in accounting workflows, which can pull the representation towards CONTABILITA or other domains.

Example FN (document label under-ranked vs. adjacent topics).

"Come registro un pagamento di scadenze in euro e in valuta estera a parità di fornitore e data pagamento?"

Gold labels: {DOCUMENTI_FATTURAZIONE}

Predicted: {}

$p_{\texttt{DOC\_F}}(x) = 0.488$, $t_{\texttt{DOC\_F}} = 0.50$, $\Delta = -0.012$ (top scores include CONTABILITA).

Fixability assessment. The dominant error mode for DOCUMENTI_FATTURAZIONE in this run is threshold-adjacent false negatives (small $|\Delta|$). This suggests that errors are largely *fixable* via better calibration (e.g., more conservative per-topic threshold tuning, or adding boundary-case supervision), rather than stemming from a fundamentally incoherent label.

4.11.3 Error Analysis: CONTABILITA

Pattern 1: overlap with FISCO_IVA. Accounting requests frequently mention VAT-related operations or terminology. This creates systematic overlap between CONTABILITA

and FISCO_IVA, consistent with the co-occurrence diagnostics (Section 4.4.2) and can lead to FN/FP errors near the boundary.

Example FN (gold includes accounting, model predicts tax domain).

"dobbiamo inserire una nuova aliquota iva a %"

Gold labels: {FISCO_IVA, CONTABILITA}

Predicted: {FISCO_IVA}

$p_{\texttt{CONT}}(x) = 0.488$, $t_{\texttt{CONT}} = 0.55$, $\Delta = -0.062$.

Pattern 2: accounting intent expressed as a short, underspecified command. Short or command-like requests can be semantically thin; in such cases the embedding-based classifier may fail to retrieve enough context to activate CONTABILITA above the tuned threshold.

Example FN (underspecified, borderline).

"IDENTIFICAZIONE CENTRI DI COSTO REG.IN PRIMA NOTA"

Gold labels: {CONTABILITA}

Predicted: {}

$p_{\texttt{CONT}}(x) = 0.508$, $t_{\texttt{CONT}} = 0.55$, $\Delta = -0.042$.

Pattern 3: mild over-triggering on VAT-heavy questions (FP). Some FP cases occur when VAT-related questions are close to accounting workflows. For example, the model predicts CONTABILITA jointly with FISCO_IVA even when the gold label includes only the tax macro-topic.

Example FP (VAT comparison question).

"differenza tra ventilazione iva e scissione dei pagamenti"

Gold labels: {FISCO_IVA}

Predicted: {FISCO_IVA, CONTABILITA}

$p_{\texttt{CONT}}(x) = 0.576$, $t_{\texttt{CONT}} = 0.55$, $\Delta = +0.026$.

Fixability assessment. Errors for CONTABILITA appear to be a mixture of (i) threshold-adjacent cases (small $|\Delta|$), which are plausibly *fixable* through calibration and additional

boundary-case examples, and (ii) intrinsic overlap with FISCO_IVA, which reflects real operational workflows. In other words, part of the residual error may be irreducible under text-only supervision unless the annotation guidelines explicitly encode a hierarchy (domain topic vs. tax modifier) or allow systematic co-labeling in VAT-heavy accounting intents.

Takeaway across topics. Extending the analysis confirms that failure modes differ across labels: SUPPORTO_TECNICO is primarily limited by cross-domain ambiguity and intent-as-modifier semantics, whereas DOCUMENTI_FATTURAZIONE is dominated by borderline threshold errors, and CONTABILITA exhibits both threshold effects and systematic overlap with FISCO_IVA. This supports the interpretation that most residual errors are explainable and that improvements should focus on calibration, guideline clarity for co-labeling, and targeted supervision on boundary cases.

4.12 Learning Curve and Data Requirements

Motivation. The auto-tagging dataset is small ($n = 406$ usable conversations) and contains rare labels (e.g., IMPORTAZIONI_INTEGRAZIONI, TICKET). As a consequence, headline metrics (e.g., Macro-F1) exhibit non-negligible variance across random splits (Section 4.9). A learning-curve analysis is used to quantify how performance scales with additional supervision. Studying the shape of these curves is a standard practice to characterize model behavior in low-data regimes and to estimate the point of diminishing returns [15].

4.12.1 Experimental protocol

We fix a single validation/test split and vary only the *effective training size*. Specifically, we keep the 70/15/15 split described in Section 4.6 (train=284, val=61, test=61), and for each training size n we sample a subset of n conversations from the training pool. For each n, we repeat the sampling and training procedure $R = 10$ times with different random seeds and report mean and standard deviation of Macro-F1 on the fixed test set.

Training sizes. We evaluate $n \in \{50, 75, 100, 125, 150, 175, 200, 230, 260, 284\}$. Note that $n = 284$ corresponds to using the full training pool. At $n = 284$ there is no subset-sampling variance; any remaining variability would come only from optimizer/numerical effects.

Model and evaluation. Each run trains the same One-vs-Rest logistic regression model over text embeddings (Section 4.5). Per-label decision thresholds are tuned on the validation set to maximize F1 and then applied unchanged on the test set. The primary learning-curve metric is Macro-F1.

4.12.2 Curve fitting and data requirement estimate

To convert discrete learning-curve points into a data requirement estimate, we fit a simple bounded and monotonic function to $(n, \text{MacroF1})$:

$$\hat{y}(n) = \sigma(a + b \log n), \qquad \sigma(z) = \frac{1}{1 + e^{-z}}.$$

This parametrization is stable in a bounded metric regime and captures diminishing returns as n grows. We use the fit only for modest extrapolation beyond the largest observed n.

4.12.3 Results and Estimated Data Requirements

Figure 4.2 shows Macro-F1 (mean $\pm$ std) versus training size n. Performance improves rapidly from $n = 50$ to $n \approx 150$, and then exhibits diminishing returns, consistent with a small-data regime approaching saturation.

Notably, the full-training point ($n = 284$) lies slightly above the fitted smooth trend, suggesting a favorable realization of the fixed validation/test split. This observation is consistent with the presence of near-duplicates and template-like redundancy discussed in Section 4.9, where a larger training set increases the likelihood of finding a high-similarity match for test instances. For this reason, we interpret the fitted curve as a conservative, "de-biased" trend estimate rather than forcing it to pass through the last observed point.

For a target MacroF1 $= 0.72$, the fitted model estimates $n \approx 361$ training conversations. Under the current 70/15/15 protocol, this corresponds to roughly $361/0.70 \approx 516$ total labeled conversations, assuming the same split proportions are maintained.

To quantify uncertainty, we bootstrap the $R = 10$ runs at each training size, refit the curve for each bootstrap sample, and recompute n_{required}. This yields a 95% confidence interval of $[323, 419]$ training conversations (with a median ≈ 357), providing a robust range for future data collection efforts.

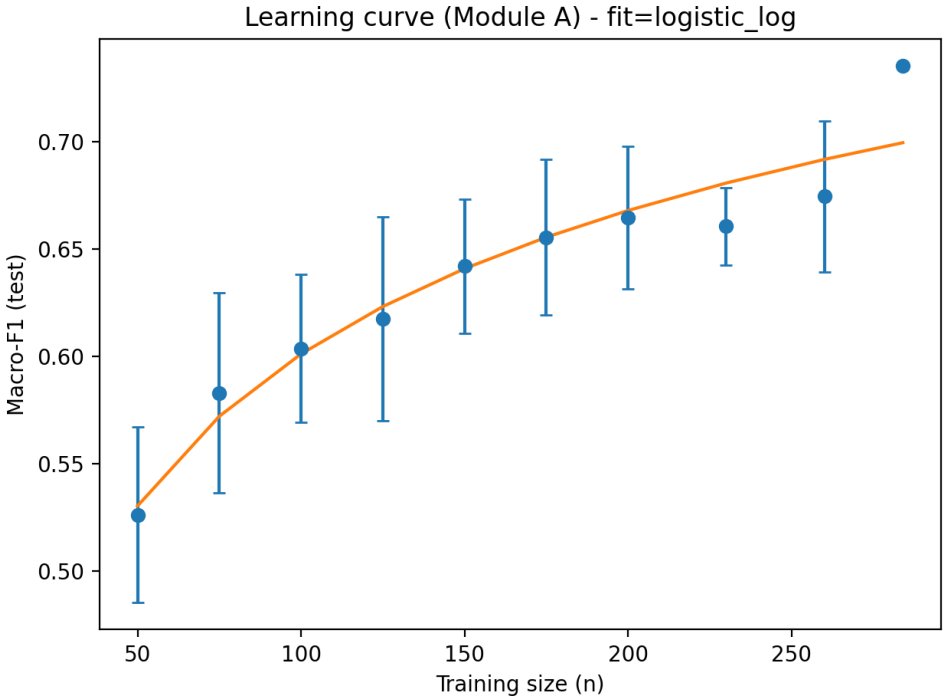


Figure 4.2 – Learning curve for multi-label macro-topic prediction (Module A): Macro-F1 on a fixed test set as a function of training size n. Points report mean$\pm$std across $R = 10$ repeats per n. The fitted curve uses a bounded $\sigma(a + b \log n)$ model.

Caveats. The extrapolation assumes that future data are drawn from the same distribution as the current logs. In addition, the dataset contains near-duplicate conversations across splits (Section 4.9), which may mildly inflate absolute scores; however the learning-curve trend (relative gains vs. n) remains informative for planning additional annotation.

4.12.4 Impact of Near-Duplicates on Performance

To assess the impact of near-duplicates on performance estimates, we identified test conversations with cosine similarity > 0.95 to at least one training conversation. This threshold captures near-paraphrases and template-like repetitions that may inflate offline metrics.

In the test set (N=61), 12 conversations (19.67%) meet this criterion. We re-evaluated the model on a *cleaned test set* excluding these near-duplicates (N=49).

Metric	**test_full** (N=61)	**test_clean** (N=49)	**Δ (%)**
Macro-F1	0.695	0.669	−3.7%
Micro-F1	0.649	0.645	−0.6%
Macro-AP	0.792	0.737	−6.9%

Table 4.10 – Performance comparison: full test set vs cleaned test set (near-duplicates removed).

The performance drop is moderate: -3.7% in Macro-F1 and -6.9% in Macro-AP. This indicates that near-duplicates contribute to mildly optimistic estimates, but the cleaned-test results remain informative and suggest that the model captures semantic regularities beyond direct memorization of training templates.

Implication for future work. Future dataset construction should enforce similarity-based deduplication across splits. A conservative approach would exclude from the test set any conversation with cosine similarity > 0.90 to training, ensuring more realistic generalization estimates. The current `test_full` results should be interpreted as an upper-bound estimate, while `test_clean` provides a more conservative and reproducible benchmark.

4.13 Output Integration

Auto-tagging predictions are persisted as versioned artifacts and exported features. For each conversation, the pipeline stores:

- a binary indicator for each macro-topic;
- predicted probabilities for each macro-topic;

- the tuned decision thresholds (saved as artifacts to guarantee consistent deployment).

These enriched features are subsequently used as inputs by the handoff recommendation module, enabling routing decisions that account for both semantic topic context and operational intent.

4.14 Discussion

The auto-tagger shows that modern semantic embeddings combined with a lightweight linear classifier can produce useful and interpretable labels on a small, operationally grounded dataset. The approach is computationally efficient, reproducible, and easy to deploy: embeddings capture semantic structure, while logistic regression provides stable training and interpretable decision boundaries.

The diagnostics also highlight important limitations. First, `SUPPORTO_TECNICO` remains difficult due to semantic heterogeneity and its cross-domain usage (low exclusivity and relatively low cohesion). This mismatch between human-centric operational categories and machine-readable semantic signals highlights the challenge of *label ambiguity* in natural language tasks.

As argued by Plank et al. (2014) [16], disagreement in human annotations often reflects inherent linguistic complexity rather than simple noise. In the specific case of `SUPPORTO_TECNICO`, the operational logic used by human agents (intent-driven) does not align perfectly with the objective semantic clusters of the text. This creates a fuzzy boundary that is difficult for any model to learn without a deeper refinement of the underlying *data contract* and annotation guidelines.

Improving labeling guidelines, introducing hierarchical labels (e.g., “support” as a modifier on top of a domain topic), or collecting additional supervision could reduce ambiguity. Second, near-duplicate detection suggests that offline evaluation may be mildly optimistic because a portion of the test set is highly similar to training conversations. Future iterations should therefore consider splitting strategies that reduce semantic redundancy and provide more conservative estimates of generalization performance.

Finally, even when downstream ablations show limited incremental gain from auto-tag features in the current binary routing regime, this null result is regime-specific rather than a fundamental limitation of the approach. In the present C=2 setting, decision boundaries are already well-separated in embedding space, leaving little headroom for additional features. However, Module A represents a necessary infrastructure investment for the expected evolution of the system: as the agent catalog grows and routing transitions to a multi-class setting (C>5), topic-based features are expected to provide discriminative signal for disentangling overlapping domains—for instance, distinguishing technical support on invoicing from pure invoicing queries, or separating accounting workflows from VAT-specific requests. Beyond its role as a downstream feature, Module A provides interpretable, auditable semantic signals for analytics, monitoring, and human debugging, and it enables hybrid routing policies where deterministic rules or guards can be conditioned on structured intent rather than raw text alone.

Chapter 5

Handoff Recommendation

This chapter describes the machine learning module that recommends the most appropriate agent to handle the next step of a conversation. At handoff time, the system observes only the information available *before* a transfer and predicts the *target agent* that should receive the handoff.

5.1 Problem Definition and Objective

At each handoff event, the system must select one agent from a finite set of candidates. Let x denote the feature representation of the pre-handoff conversation state, and let $y \in \{1, \ldots, C\}$ be the target agent. The objective is to learn a scoring function $s(\cdot)$ that assigns a real-valued suitability score to each agent:

$$s(x) = \big(s_1(x), \ldots, s_C(x)\big),$$

where larger values indicate more suitable agents. At inference time, the model produces a ranked shortlist:

$$\mathrm{TopK}(x) = \mathrm{argsort}_{c \in \{1,\ldots,C\}}\, s_c(x) \text{ (descending)}.$$

The runtime can either select the top-1 recommendation directly or apply deterministic guard rules (Section 5.9) before finalizing the transfer.

This ranking framing matches the operational requirement: in practice, an ordered

shortlist is often more useful than a single hard decision, particularly under class imbalance and when several agents are plausible [9]. In the current experimental snapshot, eligibility filtering yields $C = 2$ for both evaluated chatbots; nevertheless, we retain a ranking interface to keep the design consistent with the expected multi-agent evolution.

5.2 Available Information and Feature Design

A central constraint is **causality**: only signals observable *before* the handoff may be used. Any post-transfer information would introduce target leakage and inflate offline evaluation [3]. Features are therefore constructed from pre-handoff text and a small set of explicitly whitelisted structured fields. Anti-leakage is enforced at two levels: (i) extraction-time timestamp filtering for text, and (ii) a hard feature whitelist for structured variables.

5.2.1 Text Semantics via Embeddings

The primary signal is the pre-handoff user context available at the decision point t, constructed from user messages strictly before the transfer event (cf. Chapter 3). Formally, only messages with `created_at` $< t$ are included; messages that collide at the same timestamp as the trace are conservatively excluded to avoid ambiguous boundary cases. This text is embedded using the OpenAI model `text-embedding-3-small`, yielding a dense representation $x_{\text{text}} \in \mathbb{R}^{1536}$.

Embeddings provide a compact representation of intent and domain-specific vocabulary, and often suffice to disambiguate which specialist agent is best suited for the request [11].

5.2.2 Safe Metadata

In addition to text, the model can incorporate structured variables that are safe by construction, i.e., recorded before the handoff and not derived from the target. We include:

- `message_count_pre`: number of messages before the transfer;
- `n_prev_transfers`: number of previous transfers in the same conversation;

- `transfer_index`: ordinal index of the current handoff within the conversation;
- `is_text_suspect`: heuristic flag for truncated/noisy inputs.

These variables are explicitly *whitelisted*. Potentially leaky fields (e.g., `target_new_agent_id`, any post-handoff variables, audit identifiers) are excluded from the ML feature matrix.

Separating learned intent from historical routing signals. We explicitly exclude `active_agent_id_pre` from the ML feature matrix to avoid encoding historical routing policies as learned behavior. The same variable is used only (i) to define a *router baseline* for comparison and (ii) in a separate *guard layer* applied at runtime under high-support, high-purity conditions (Section 5.9). This separation yields a clean interpretation: the learned model captures semantics, while policy-like behavior is handled explicitly and conservatively when it is statistically reliable.

5.2.3 Auto-tag Probabilities

The system also includes an auxiliary topic classifier (Module A) that outputs a probability distribution over 9 macro-topics (e.g., *FISCO_IVA*, *CONTABILITA*, *DOCUMENTI_FATTURAZIONE*, ...). These probabilities can be appended as a compact feature block $x_{\text{tag}} \in \mathbb{R}^9$.

No-leak generation protocol (fold-aligned OOF). To avoid leakage, topic probabilities are generated with an out-of-fold protocol aligned with the recommender split at `conversation_id` level: out-of-fold (OOF) probabilities are computed only from training conversations, and test probabilities are generated by fitting topic models on training conversations only and applying them to recommender-test. This ensures that Module B never consumes topic features produced by a model that has seen the same conversations during fitting.

5.3 Model Architecture

The recommender is implemented as a One-vs-Rest logistic regression (OvR) [17]. For each agent c, a binary classifier estimates:

$$p(y = c \mid x) = \sigma(w_c^\top x + b_c),$$

where $\sigma(\cdot)$ is the logistic sigmoid. The resulting probabilities are used as ranking scores.

Although eligibility filtering yields a binary problem ($C = 2$) for the current experiments, we adopt OvR logistic regression because it generalizes to $C > 2$ without changing the training and inference interface. This design choice is aligned with production evolution: new agents may be introduced over time and, as the number of conversations grows, additional classes can become eligible. Using a single scalable formulation avoids re-architecting the system when the routing problem transitions from binary to multi-class.

Feature blocks are concatenated and fed through a pipeline that includes `StandardScaler(with_mean=False)` to preserve compatibility when sparse or mixed representations are present.

5.4 Training and Data Splitting

Each dataset row corresponds to a handoff event. Because multiple handoffs can originate from the same conversation, a naive random split would leak conversation-level information [3] between train and test. To prevent this, we use a **group-wise** split by `conversation_id` implemented with `StratifiedGroupKFold`. Since grouping is done at `conversation_id` level while labels are event-based, stratification is necessarily approximate: because entire conversations are assigned to a single split, fold-level class counts can deviate from global proportions when the minority class is concentrated in a small number of conversations (especially for Chatbot 2). Nevertheless, this procedure aims to approximately preserve class proportions across folds while preventing conversation-level leakage.

Note on stratification with grouped data. Stratification operates on the event-level label distribution within each conversation to guide the fold assignment, but the final allocation assigns *entire conversations* to folds. As a consequence, minority-class support in a fold may deviate from the target proportion when the minority class is concentrated in a small number of conversations (especially for Chatbot 2, where agent 6 appears in only a handful of conversations). We accept this as a necessary trade-off to prevent conversation-level leakage: splitting events from the same conversation across train/test would introduce information contamination and invalidate offline evaluation.

For the main results we report a single group-wise holdout split; we then quantify split sensitivity via cross-seed evaluation (Section 5.7.2).

5.4.1 Eligibility Filtering: `MIN_AGENT_SAMPLES`

The training set is filtered to include only agents with sufficient support. Let $\text{count}_{\text{train}}(c)$ be the number of training samples for agent c. The set of eligible agents is:

$$\mathscr{C}_{\text{eligible}} = \{c \mid \text{count}_{\text{train}}(c) \geq \texttt{MIN\_AGENT\_SAMPLES}\}.$$

This avoids pathological cases where a classifier would be trained on only a handful of examples for an agent.

Event-level filtering (not conversation-level). Eligibility is applied at the *handoff-event* level (dataset rows), not at the conversation level. We first split conversations into train/test using `conversation_id` as a grouping key. Then we compute $\mathscr{C}_{\text{eligible}}$ on the training split only and *drop individual events* whose target agent y is not eligible. As a consequence, a single conversation may contribute fewer events after filtering; a conversation disappears only if all its events are removed.

Example. Suppose conversation X contains three handoff events with targets $[2,3,1]$ and $\mathscr{C}_{\text{eligible}} = \{2,3\}$ (computed on train only). Then conversation X contributes two rows to the filtered dataset (targets 2 and 3), while the event targeting agent 1 is dropped due to

insufficient training support.

Why lowering the threshold can be legitimate (Chatbot 2). Chatbot 2 is a more recent deployment iteration of Chatbot 1 and therefore has fewer historical conversations and fewer handoff traces. As a consequence, with a high threshold only one class can survive in the train split, making supervised learning difficult to formulate reliably. Lowering `MIN_AGENT_SAMPLES` to a small value is thus a pragmatic choice to keep at least two eligible classes and to treat Chatbot 2 explicitly as a low-data case study. This is stated as a limitation: metrics on Chatbot 2 have higher variance and are more sensitive to the group split.

5.4.2 Dataset Statistics and Agent Mapping

Agent mapping (IDs → names). For interpretability, we refer to agents by both identifier and human-readable name (from `cbt_agents`). For Chatbot 1: agent 1 = *Conversator*, agent 2 = *Retriever*, agent 3 = *Ticket creator*. For Chatbot 2: agent 4 = *Receptionist* (commonly appearing as the active pre-handoff agent), agent 5 = *Arca expert*, and agent 6 = *Ticket reporter*.

Chatbot 1. Raw label counts over all handoff events for `chatbot_id=1` are: agent 2: 284, agent 3: 29, agent 1: 5. A group-wise split by `conversation_id` yields 232 unique conversations in train and 57 in test. With `MIN_AGENT_SAMPLES=10` computed on train only, the eligible target agents are $\{2,3\}$. After filtering to eligible target classes at the event level, the split contains 247 training handoff events (rows) and 66 test handoff events (rows), with:

$$\text{train: } \{2:228, 3:19\}, \qquad \text{test: } \{2:56, 3:10\}.$$

The distribution of `active_agent_id_pre` is heavily unbalanced (the pre-handoff active agent is most often agent 1), which motivates both the router baseline and the guard analysis. Most conversations contain a single handoff event; a minority contributes multiple events (mean: 1.06 events/conv in train, 1.16 in test).

Chatbot 2. Chatbot 2 is newer and has substantially fewer logged handoff events. With the same extraction protocol, a group-wise split yields 39 training rows and 45 test rows after eligibility filtering. With `MIN_AGENT_SAMPLES=3` (train-only), the eligible agents are $\{5, 6\}$, i.e., *Arca expert* vs *Ticket reporter*, with:

$$\text{train: } \{5 : 36, 6 : 3\}, \qquad \text{test: } \{5 : 36, 6 : 9\}.$$

This is a small-sample regime for the minority agent. In particular, with only three minority-class training examples the corresponding classifier is effectively in a few-shot setting; results on Chatbot 2 should therefore be interpreted as illustrative and primarily motivate the hybrid router+ML strategy described later.

5.4.3 Class Imbalance and Minority-Class Challenges

Both chatbots exhibit severe class imbalance, reflecting operational reality where specialized agents handle fewer handoff events than general-purpose agents:

- **Chatbot 1**: The minority class (agent 3, *Ticket creator*) has only **19 training examples** (7.7% of train set) and 10 test examples.
- **Chatbot 2**: The imbalance is extreme—agent 6 (*Ticket reporter*) has only **3 training examples** (7.7% of train set), constituting a **few-shot regime**.

This distribution is not an artifact of data collection but a consequence of operational design: high-frequency agents (e.g., agent 2, *Retriever* in Chatbot 1) handle the majority of conversational flow, while specialized agents (e.g., ticket creation) intervene rarely.

5.4.3.1 Mitigation Strategies

To address class imbalance, we employ several techniques:

1. **Balanced class weights** (`class_weight='balanced'` in `LogisticRegression`): Minority examples are up-weighted during training to mitigate imbalance. We use `class_weight='balanced'`, which assigns weights inversely proportional to class

frequency (equivalently, proportional to $1/n_c$). Specifically, each class c receives weight $w_c = \frac{n_{\text{total}}}{k \cdot n_c}$, where k is the number of classes and n_c is the support for class c. In our setting, this effectively amplifies the importance of minority handoff events (e.g., for Chatbot 1, the minority class weight is approximately 12 times larger than the majority one), forcing the model to learn a decision boundary that accounts for rarer routing patterns despite their low frequency.

2. **One-vs-Rest formulation**: Each agent has an independent binary classifier, allowing minority classes to be learned separately rather than competing directly with the majority in a single multi-class decision function.

3. **Minority-focused evaluation**: We report minority-class precision, recall, F1, and Average Precision (AP) alongside overall accuracy to ensure minority performance is not masked by majority dominance. For Chatbot 1, the minority class (agent 3) achieves F1=0.84 and AP=0.96, showing reasonable performance despite limited support.

4. **Hybrid router strategy** (Section 5.9): For cases where routing is deterministic (high purity, high support), we bypass ML entirely and use policy-based rules, reducing reliance on minority-class predictions where supervision is insufficient.

5.4.3.2 Limitations

Despite mitigation efforts, minority-class predictions remain fragile:

- **Chatbot 1**: With 19 training examples, the minority classifier has high variance. Performance is reasonable (minority F1=0.84, AP=0.96) but sensitive to split selection. Cross-validation would further fragment this already-small sample, so we use a single holdout (Section 5.7.2).

- **Chatbot 2**: With 3 training examples, supervised learning is **not reliable**. The router baseline (Top-1=0.978) vastly outperforms ML (Top-1=0.889), suggesting that policy-based routing is safer under extreme low-data conditions.

Implication for future work. Future deployments should prioritize collecting minority-class examples through:

- **Active learning**: Identify ambiguous cases near decision boundaries and request operator labels [18]
- **Extended deployment**: Collect more handoff events over time as the system matures
- **Synthetic generation** (with caution): Generate plausible minority examples via paraphrasing or back-translation, validated by domain experts

A target of **50+ examples per class** would enable more robust classifiers and reduce reliance on guard heuristics. Until this threshold is reached, the hybrid router+ML strategy (Section 5.9) provides a safe and auditable deployment path.

5.5 Evaluation Metrics

Although the model outputs ranking scores, after eligibility filtering both chatbots have $C = 2$ eligible agents. In this binary setting, Top-K metrics with $K = 2$ are trivial and are therefore omitted in the main discussion. We still keep the ranked shortlist output at runtime (Top-K) because it supports guard logic and human-in-the-loop review, even when offline Top-K evaluation is uninformative.

We primarily report:

- **Top-1 accuracy:** fraction of examples where the true agent is ranked first;
- **Minority-class precision/recall/F1:** to assess behavior on the rarer agent under imbalance;
- **Average Precision (AP) for the minority class:** to summarize the precision–recall trade-off using ranking scores (threshold-free) [19].

For completeness, we also report MAP@2 in tables as an interface-consistent ranking metric; in the binary case it closely tracks accuracy and should not be over-interpreted.

5.6 Baselines

Two baselines contextualize the recommender:

- **Majority baseline:** always predict the most frequent agent in training.
- **Router baseline:** predict the most frequent target agent conditioned on `active_agent_id_pre`. This captures near-deterministic historical routing policies and provides a strong reference for policy-driven routing.

Router baseline (train-only). The router predicts the most frequent target agent conditioned on the pre-handoff active agent:

$$\hat{y}_{\text{router}}(a) = \arg\max_{y} \hat{p}_{\text{train}}(y \mid \texttt{active\_agent\_id\_pre} = a),$$

where $\hat{p}_{\text{train}}(y \mid a)$ is estimated from training events only (after eligibility filtering). If an a value is unseen in training, we fall back to the training majority class.

5.7 Results and Ablation Study

An ablation study compares four feature configurations:

1. **text-only** ($x = x_{\text{text}}$),
2. **text + safe metadata**,
3. **text + auto-tag probabilities**,
4. **text + auto-tag probabilities + safe metadata**.

The goal is to quantify the marginal benefit of each additional feature block under a controlled experimental protocol. In the current $C = 2$ setting with limited test size, multiple variants can tie; ablation outcomes are therefore interpreted conservatively.

5.7.1 Chatbot 1 Results

Baselines and ML. On Chatbot 1 (eligible agents $\{2,3\}$), the majority baseline achieves Top-1 accuracy 0.848 and the router baseline achieves 0.924. The text-only ML model achieves Top-1 accuracy 0.955 and MAP@2 = 0.977.

The minority agent (agent 3, *Ticket creator*) has test support = 10 with precision 0.89, recall 0.80, and F1 0.84. The minority-class AP is 0.957.

Ablation (Chatbot 1). All four variants achieve identical Top-1 accuracy and MAP@2 on this test split. Given the small test size (N=66), binary regime (C=2), and severe class imbalance (56 vs. 10 samples), the ablation lacks statistical power to detect marginal improvements from additional features. We therefore interpret this outcome as **no measurable gain under current evaluation conditions**, rather than evidence that metadata or auto-tag features are inherently uninformative. In larger multi-class settings (C>5) or with more balanced data, structured features may provide complementary signals for disentangling cross-domain intents. In the current regime ($C = 2$), routing decisions are already well separated in embedding space, leaving limited headroom for additional features. Therefore, the lack of measurable gains from auto-tag probabilities should be interpreted as a regime effect rather than evidence of uselessness: topic signals are expected to matter more as the routing problem becomes truly multi-class ($C > 2$) and intents overlap.

Variant	Top-1	MAP@2	#features
text-only	0.955	0.977	1536
text + meta_safe	0.955	0.977	1540
text + autotag_probs	0.955	0.977	1545
text + autotag_probs + meta	0.955	0.977	1549

5.7.2 Cross-Seed Stability (Chatbot 1)

To quantify sensitivity to the group split, we ran a cross-seed evaluation on Chatbot 1 using three random seeds (42, 43, 44) with the same fold index and the same feature configuration

Table 5.1 – Cross-seed stability for Chatbot 1 (text-only). Mean±Std computed over seeds 42, 43, 44.

Metric	Seed 42	Seed 43	Seed 44	Mean±Std
Top-1 accuracy	0.955	0.968	1.000	0.974±0.019
Minority F1	0.842	0.800	1.000	0.881±0.086
Minority AP	0.959	0.976	1.000	0.978±0.017

(*text-only*). We fix the minority class globally as the rarest eligible agent (agent 3; 29 events in the full dataset) to keep minority metrics comparable across seeds.

Top-1 accuracy is relatively stable across seeds, while minority F1 shows higher variance due to the small number of minority positives in the test fold (10, 6, and 3 across seeds). In particular, the seed-44 split achieves perfect scores (Top-1=1.000, minority F1=1.000) with only 3 minority positives in the test fold; such estimates are high-variance and can occur when those few cases are comparatively easy or lack hard negatives nearby. This supports interpreting the single-split minority F1 as a high-variance estimate, while Top-1 and minority AP appear more stable in this low-data regime.

5.7.3 Chatbot 2 Results

Baselines and ML. On Chatbot 2 (eligible agents $\{5,6\}$), the majority baseline achieves 0.800 and the router baseline is extremely strong at 0.978. The text-only ML model achieves Top-1 accuracy 0.889 and MAP@2 $= 0.944$.

Interpretation (low-data + policy-driven routing). Chatbot 2 is a newer version of Chatbot 1 and therefore has fewer historical handoff events available for training. In addition, routing behavior is strongly driven by the active pre-handoff agent, making the conditional router a very competitive baseline. The ML model remains useful as a semantic fallback in ambiguous regions, but it is not expected to consistently outperform a near-deterministic policy under such limited minority support (agent 6 has only 3 training examples).

The minority agent (agent 6, *Ticket reporter*) has test support= 9 and minority-class AP reaches 1.0. This value should be interpreted cautiously: in small-sample regimes, AP

can be inflated when a handful of positives are ranked above negatives, even if the same ranking quality would not persist under a different split.

Ablation (Chatbot 2). As in Chatbot 1, all feature variants yield identical Top-1 accuracy and MAP@2 in this split. This is consistent with a binary, low-data regime where additional features have limited measurable headroom on a 45-example test set.

Variant	Top-1	MAP@2	#features
text-only	0.889	0.944	1536
text + meta_safe	0.889	0.944	1540
text + autotag_probs	0.889	0.944	1545
text + autotag_probs + meta	0.889	0.944	1549

5.7.4 Low-Data Case Study: Chatbot 2 (Few-shot Minority Regime)

Chatbot 2 provides a useful *low-data* (and particularly *few-shot*) case study that highlights a key limitation of semantic recommenders trained on heavily imbalanced handoff data. After eligibility filtering, the recommender is trained on only 39 handoff events, with an extreme imbalance between the two eligible agents:

$$\text{train: } \{5:36, 6:3\}, \qquad \text{test: } \{5:36, 6:9\}.$$

In this regime, the classifier for the minority agent (agent 6, *Ticket reporter*) is effectively trained from only three positive examples. While balanced class weights mitigate majority collapse, they do not create new supervision: generalization remains intrinsically high-variance, especially in mixed-intent contexts where multiple plausible agents can be semantically related to the same input.

A representative failure mode: topic continuity dominates the action request. We observed multiple real test cases where the semantic model (text-only OvR logistic regression on embeddings) predicts agent 5 (*Arca expert*) with very high confidence even though the true handoff target is agent 6 (*Ticket reporter*). A representative example is shown below.

User: che cos'è arca evolution erp?

Assistant: ARCA Evolution è un software gestionale sviluppato da Wolters Kluwer, pensato specificamente per le piccole e medie imprese (PMI)...

User: mi aiuti a creare un ticket?

At this handoff (conversation_id=A, handoff_id=A), the pre-handoff active agent is the *Receptionist* (active_agent_id_pre=4) and the ground-truth target is agent 6 (*Ticket reporter*). The conditional router baseline correctly recommends transferring to the ticket reporter agent. The ML recommender instead assigns almost all probability mass to agent 5:

Signal	Value
chatbot_id	2
active_agent_id_pre	4 (*Receptionist*)
target_agent_id	6 (*Ticket reporter*)
router output	transfer_to_ticket_reporter_agent (correct)
ML top-1	agent 5 score 0.9575 (wrong)
ML score (target agent 6)	0.0425

This failure illustrates an important interaction between (i) the semantic representation and (ii) the supervision regime. The input contains two distinct intents: an informational request about *ARCA Evolution* followed by an operational request (*create a ticket*). Because agent 6 is extremely underrepresented in training, the model has insufficient signal to reliably learn that ticket-creation phrasing should override topical continuity in mixed contexts. As a result, the embedding-driven scorer tends to follow the dominant topic association (ARCA), which is strongly aligned with agent 5, even when the correct operational action is to open a ticket via agent 6.

Why the router can legitimately dominate under low-data conditions. In Chatbot 2, routing behavior is largely policy-driven by the pre-handoff active agent. In such settings, the conditional router $\hat{y} = \arg\max_y \Pr(y \mid \texttt{active\_agent\_id\_pre} = a)$ can be both strong and auditable. When the empirical distribution is sufficiently concentrated (high purity) and

backed by enough observations (high support), it is statistically safer to use a deterministic router than to rely on a semantic model trained with only a handful of minority examples.

This motivates an explicit *router-vs-ML* decision rule based on support and purity (further operationalized in Section 5.9): prefer deterministic routing when

$$\text{purity}(a) \geq 0.99 \quad \text{and} \quad \text{support}(a) \geq 10,$$

and fall back to ML otherwise. Under this rule, Chatbot 2 naturally falls into a regime where routing from the *Receptionist* is near-deterministic and therefore should override ML in routine cases, while the semantic recommender remains useful as a fallback for low-support or lower-purity active-agent states.

Additional evidence. A second observed example (conversation_id=660, handoff_id=5093) is even shorter:

> **User:** come apro un ticket per arca evolution?

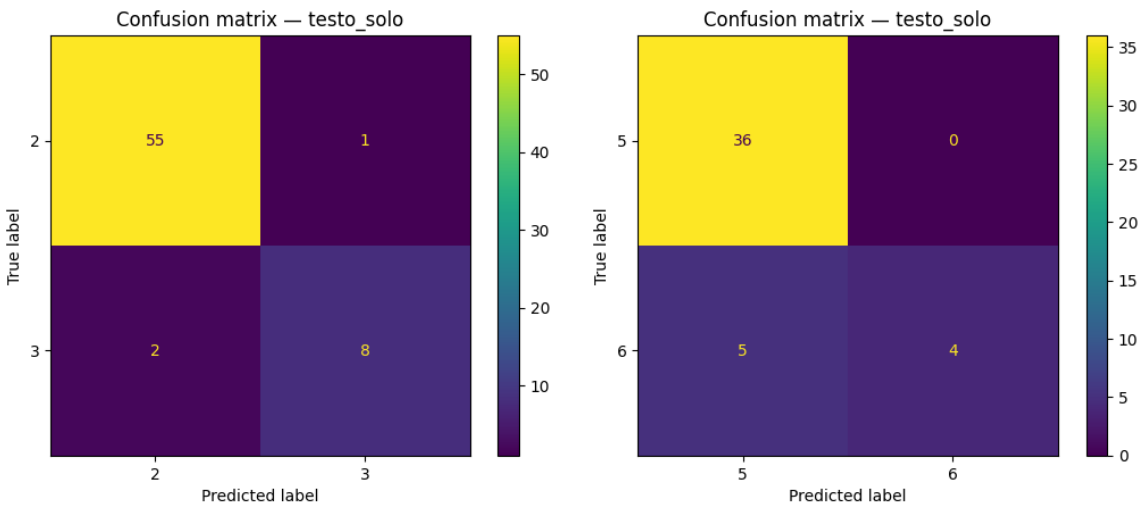
Despite the explicit ticket intent, the ML model again predicts agent 5 with very high confidence (score 0.9985) while assigning negligible probability to the true target agent 6 (0.00146), whereas the router recommends transferring to the ticket reporter agent. These cases reinforce that, in an extreme few-shot regime, high-confidence ML scores should not be interpreted as reliable evidence of correctness, and that hybrid routing with conservative deterministic guards is a defensible deployment strategy.

5.8 Error Analysis

Confusion matrices and Precision–Recall (PR) curves provide additional insight under class imbalance.

5.8.1 Confusion Matrices

Figures 5.1a and 5.1b show confusion matrices for the text-only models.



(a) Chatbot 1 (text-only). **(b)** Chatbot 2 (text-only).

Figure 5.1 – Confusion matrices for the text-only models.

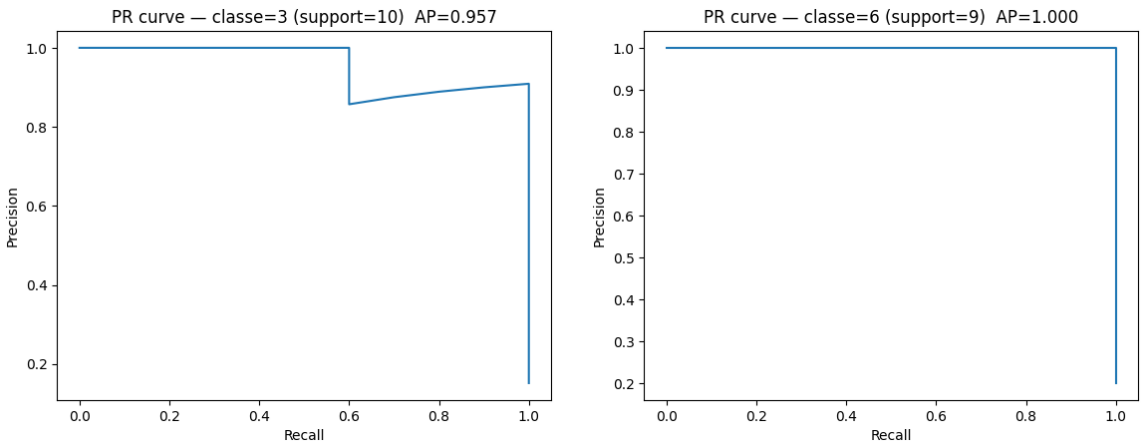
Chatbot 1. Most errors occur on the minority agent (agent 3, *Ticket creator*): precision is 0.89 and recall is 0.80 (support = 10), consistent with severe imbalance.

Chatbot 2. Errors concentrate on the minority agent (agent 6, *Ticket reporter*): with only 3 training examples after the group split, the model behaves conservatively on the minority class and can miss true agent-6 cases. This is a direct consequence of low support rather than a modeling failure.

5.8.2 Minority-class PR Curves

Figures 5.2a and 5.2b show PR curves for the minority classes: agent 3 for Chatbot 1 and agent 6 for Chatbot 2.

Note on interpretation (small-sample regime). For Chatbot 2, PR curves and AP can appear optimistic because the minority class has very small support. They are therefore best interpreted qualitatively (shape and trade-off), rather than as standalone quantitative claims.



(a) Chatbot 1 minority class (agent 3), AP= 0.957.

(b) Chatbot 2 minority class (agent 6), AP= 1.000 (small-sample regime).

Figure 5.2 – Precision–Recall curves for the minority classes (text-only models).

5.9 Post-processing and Guards

Given the strength of the router baseline, a deterministic guard layer can override ML predictions in near-deterministic regions. The key idea is:

$$\text{If } \Pr(\text{target} = y \mid \texttt{active\_agent\_id\_pre} = a) is extremely concentrated (high purity) and the supporti$$

Operationally, the guard enforces the router output when:

- **purity** ≥ 0.99 and
- **support** ≥ 10 (configurable).

In the experiments reported in this chapter we use $\tau = 0.99$ and $m = 10$ unless otherwise stated.

How purity and support are computed. Purity and support are estimated on the *training split only* (after eligibility filtering) and the resulting guard policy is *frozen* for evaluation and deployment. For each pre-handoff active agent a, we compute the empirical distribution $\hat{p}_{\text{train}}(y \mid a)$ from training events and define:

$$\text{support}(a) = \sum_y \#_{\text{train}}(a, y), \qquad \text{purity}(a) = \max_y \hat{p}_{\text{train}}(y \mid a).$$

If purity$(a) \geq \tau$ and support$(a) \geq m$, the guard routes deterministically to $\arg\max_y \hat{p}_{\text{train}}(y \mid a)$; otherwise it falls back to the ML ranker. Test-set purity does not affect the guard decision.

Guard statistics (from training data). For Chatbot 1, the guard triggers on `active_agent_id_pre` $\in \{1,2\}$ (both satisfy the purity/support thresholds in train). For Chatbot 2, the guard triggers on `active_agent_id_pre` $= 4$ (*Receptionist*) under the same thresholds.

This motivates a hybrid strategy: apply deterministic routing when it is statistically reliable, and fall back to ML for ambiguous cases where purity is lower or support is insufficient.

5.10 Model Output and Runtime Integration

At runtime, the module outputs:

- an ordered shortlist of candidate agents (Top-K),
- a score (probability) for each candidate,
- optional guard diagnostics (whether a router rule was applied, and why).

Note on scores. We use predicted probabilities as ranking scores. Calibration is not required for Top-1 accuracy or AP, but it becomes relevant when interpreting scores as confidence at runtime [20].

This output supports both fully automated routing (choose Top-1) and human-in-the-loop workflows (where a shortlist is presented for validation).

5.11 Summary and Limitations

The proposed recommender achieves strong results with a simple and interpretable architecture. For Chatbot 1, In this evaluation setup, semantic embeddings alone are sufficient and outperform baselines. For Chatbot 2, the task is constrained by limited training data

and near-deterministic routing patterns; in this setting, a hybrid router+ML strategy is the most defensible approach.

Key limitations are class imbalance and low support for minority agents (especially in Chatbot 2), which increase variance across splits. Future work includes collecting more minority examples, calibrating thresholds per agent, and evaluating stability across multiple group splits.

Chapter 6

Production Integration

This chapter describes how the machine learning module presented in this thesis is integrated into the production architecture of KORA.AI. The focus is on system boundaries, runtime data flow, anti-leakage constraints enforced at inference time, API contracts, and operational concerns such as versioning, testing, and observability.

The integration is intentionally designed as *suggestion-only*: the model produces ranked recommendations, but the final handoff decision is executed by a human operator through the user interface. This choice improves operational safety and auditability and enables gradual rollout before any form of automation is considered.

6.1 Architecture Overview

At a high level, the system consists of four main components:

- **Chat frontend (TypeScript + React, Vite).** The UI used by operators to conduct the conversation, visualize model suggestions, and provide feedback (accept/reject).
- **API gateway (PHP, Laravel).** A compatibility and security layer that receives chat requests from the frontend, forwards them to the backend services, normalizes the response payload, and exposes dedicated feedback endpoints.
- **RAG service (Python, FastAPI).** The orchestration service that handles the user turn: it performs retrieval (RAG), calls the LLM to generate the response, runs the

Handoff Recommendation inference, and emits traces for audit and monitoring. In the current deployment, the ML inference is *integrated within this service*.

- **Backoffice (administrative UI).** An operational interface used for configuration and auditing. In the current description, it does not execute ML inference and is treated as a control plane rather than a runtime dependency (details may vary across deployments).

All services are deployed as **Docker containers** orchestrated via **Docker Compose**. This setup enables reproducible environments and a clear separation between runtime components.

6.1.1 Design Rationale: Modularity and Boundaries

The integration follows three architectural principles:

- **ML isolation by interface.** Even when integrated inside the RAG service process, the ML module is treated as an internal component with its own artifacts, configuration, and trace types. This supports independent development and rollback of ML changes.
- **Stable API contract.** The API gateway acts as an adapter: it normalizes payload structure and naming conventions, so the frontend can depend on a stable contract while backend services evolve.
- **UI as a pure consumer.** The frontend does not contain ML logic. It only consumes the recommendation payload and displays suggestions and confidence labels, keeping ML behavior testable and centrally controlled.

6.2 End-to-End Data Flow for a User Turn

For each user message, the system executes a synchronous workflow that produces (i) a natural-language answer and (ii) a ranked shortlist of recommended agents.

```
User/Operator -> Frontend -> API Gateway (/chat/ask)
API Gateway   -> RAG Service (/api/v1/answer)
RAG Service:
  - retrieval context (RAG docs)
  - build features (anti-leakage filtering, truncation audit)
  - LLM answer
  - handoff recommendation (Top-K + confidence)
RAG Service   -> API Gateway (answer + recommended_agents + trace_id)
API Gateway   -> Frontend (normalized payload)
Frontend: display answer + suggestions; operator may accept/reject
```

Figure 6.1 – Simplified runtime sequence for a user turn in KORA.AI.

6.2.1 Request/Response Sequence

The following sequence summarizes the runtime flow for a single user turn:

1. The user/operator submits a message in the **chat frontend**.
2. The frontend calls the API gateway endpoint `/chat/ask`.
3. The API gateway forwards the request to the RAG service endpoint `/api/v1/answer`.
4. The RAG service builds the retrieval context and the ML feature representation.
5. The RAG service executes:
   - retrieval over an internal document repository [7],
   - an LLM call to generate the assistant answer,
   - *Handoff Recommendation* inference to produce a Top-K ranked shortlist.
6. The RAG service returns a payload including: `answer`, `active_responder`, `sources`, `recommended_agents`, `suggestion_trace_id`, `confidence_label`, and `model_version`.
7. The API gateway normalizes and enriches the payload (e.g., naming conventions, agent name resolution), then returns it to the frontend.
8. The frontend displays the answer and the suggestion panel (*suggestion-only*).

6.2.2 Synchronous Recommendation Timing

The recommendation is computed **synchronously during the user turn** and is available before the final response is returned to the frontend. This ensures that suggestions are always aligned with the most recent user input and the same context used for response generation.

6.3 Runtime Feature Builder

The *Handoff Recommendation* model consumes a feature vector built at inference time. The builder combines a text representation derived from user messages with optional structured signals (when available), and enforces strict anti-leakage constraints.

6.3.1 Text Field Construction

At runtime, the system constructs a single text field:

- `conversation_text_pre`: concatenation of **all pre-handoff user messages** and the **current user query**.

The concatenation is designed to preserve the most relevant conversational context for handoff decisions, while remaining simple and deterministic.

6.3.2 Anti-Leakage Filtering

The feature builder applies a strict **anti-leakage** rule:

- Only messages with `role == "user"` are eligible.
- Only messages with `created_at < suggestion_time` are eligible, where the inequality is **strict** to avoid boundary ambiguity.

The `suggestion_time` is computed inside the RAG service as `datetime.now(timezone.utc)` at the moment the recommendation is produced. This choice makes the cutoff explicit, reproducible, and aligned with the trace metadata.

This inference-time filtering mirrors the dataset construction logic used for training: the recommender must not observe assistant content, transfer confirmations, or any post-hoc artifacts that would not be available at decision time [3].

6.3.3 Truncation and Audit Fields

To control payload size and bound the embedding cost, the text is truncated to a maximum character budget:

- `max_chars = settings.recommender_max_chars = 8000`.
- Truncation strategy preserves the **most recent** content by keeping the last **8000 characters** of `conversation_text_pre`.

The builder logs audit fields that document how the representation was produced:

- `audit.message_count_pre`: number of eligible user messages included.
- `audit.text_char_len`: character length of the final `conversation_text_pre`.
- `audit.truncation_applied`: boolean flag indicating whether truncation occurred.
- `audit.n_prev_transfers`: number of previous transfer events in the conversation (if available).
- `audit.transfer_index`: ordinal index of the current handoff decision point (if available).

These fields are primarily used for **auditability** and debugging; they are not intended to introduce post-hoc signals into the model inputs.

6.3.4 Embeddings at Inference Time

The runtime text embedding is computed using `text-embedding-3-small`. The embedding dimension is fixed by the provider and is treated as an implementation detail (it is not

required by the API contract, but it must match the model artifacts trained with the same embedding configuration).

Caching is applied selectively:

- The **conversation embedding** is computed per turn and is not cached.
- **Agent description embeddings** used by the similarity fallback (Section 6.7.2) are cached to avoid repeated computation.

If embedding computation fails (e.g., provider error or timeout), the service logs a warning and proceeds without producing recommendations. The chat response remains available, ensuring graceful degradation.

6.4 Structured Features and Candidate Agents

At runtime, the handoff recommender currently uses **text embeddings only**; structured signals (tags/metadata) are not included in the inference input.

In the current deployment, only embeddings are passed to the model; structured features are not wired into the runtime inference yet. The architecture nonetheless supports the integration of structured features, and the system design separates the *candidate generation* step from the *scoring* step.

6.4.1 Candidate Set Construction

Candidate agents are retrieved from the database (e.g., `chatbot.agents`). The precise eligibility rules (disabled/hidden/non-handoffable) are deployment-dependent and can be expressed as a filter over this list. The output ranking always operates over the resulting candidate set.

To ensure a stable mapping between model outputs and agent identifiers, each chatbot has an associated `agent_order.npy` artifact (Section 6.10), which defines the exact order used during training and inference.

6.4.2 Optional Integration with Auto-Tagging

The *Auto-Tagging* module produces conversation-level labels that are used for **analytics and monitoring**. However, these labels are **not currently used** as structured features for handoff inference at runtime.

However, the modular design supports such an extension: predicted topics could be attached to the conversation state and passed as additional structured signals to the handoff model, as long as they obey the same **anti-leakage** constraints and are derived solely from pre-`suggestion_time` user content.

6.5 Runtime Inference

6.5.1 Model Loading and Execution

At each user turn, the RAG service executes inference using a scikit-learn pipeline loaded from versioned artifacts:

- Model family: `LogisticRegression`.
- Serialization: `joblib`.
- Output method: `predict_proba`, producing one probability per candidate class.

The model is loaded *on-demand* with an in-memory cache (Section 6.10).

6.5.2 Top-K Recommendation

The recommender returns Top-K suggestions with $K = 3$:

- Output list format (core): {`agent_id`, `score`}.
- Output enrichment (gateway/UI): `agent_name` is added by resolving IDs.

The list is sorted by score in descending order. The score is interpreted as a predicted probability and is used for ranking and confidence labeling (Section 6.5.3).

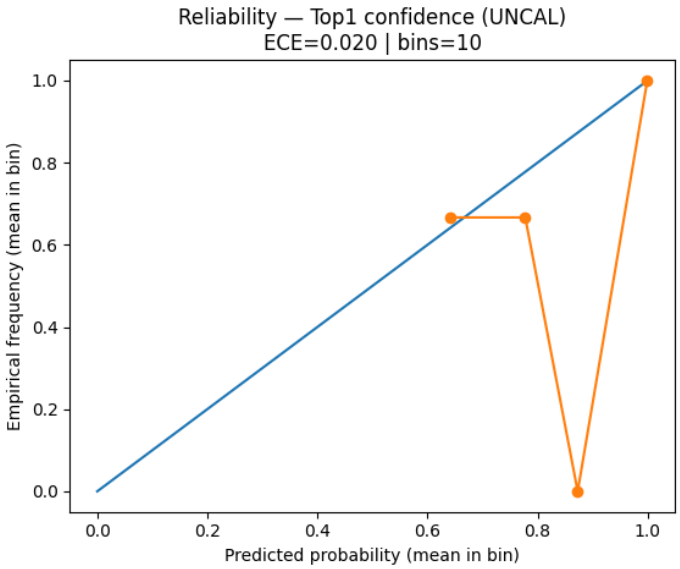


Figure 6.2 – Reliability diagram for Chatbot 1 Top-1 confidence on the test set ($N = 66$). ECE$= 0.020$ with $B = 10$ bins. With small N, some bins may contain few samples and the curve can appear jagged.

6.5.3 Calibration Analysis and Score Interpretation

To assess whether predicted probabilities can be interpreted as confidence scores in the UI, we evaluate calibration using a reliability diagram and Expected Calibration Error (ECE) with $B = 10$ bins. We focus on Top-1 confidence $p_{\text{top1}} = \max_c p(y = c \mid x)$ and correctness $\mathbb{I}[\hat{y} = y]$.

For Chatbot 1 ($N = 66$), the raw logistic regression probabilities are well-calibrated (ECE$= 0.020$, Figure 6.2). Given the small test size, this evidence should be interpreted as indicative rather than definitive; nevertheless, it supports using raw predicted probabilities as confidence scores for ranking and for the confidence label thresholds in production.

6.5.4 Confidence Label

In addition to the Top-K list, the system computes a categorical `confidence_label` (e.g., `high`, `medium`, `low`) based on:

- the top-1 probability p_{top1}, and
- the margin between the top-1 and top-2 probabilities $\Delta = p_{\text{top1}} - p_{\text{top2}}$.

Thresholds are configured in the RAG service and, in the current deployment, are set to:

- `high` if $p_{\text{top1}} \geq 0.85$,
- `medium` if $0.70 \leq p_{\text{top1}} < 0.85$,
- `low` otherwise.

Furthermore, a safety margin is enforced to handle ambiguous rankings: if the distance between the two best candidates is too small ($\Delta < 0.10$), the `confidence_label` is automatically downgraded to `low`, regardless of the absolute value of p_{top1}. This guardrail prevents misleading recommendations in near-tie scenarios where the model cannot clearly discriminate between the top candidates.

Operationally, the label is consumed by the UI (for visualization) and by the workflow logic (for fallbacks): when confidence is low or the candidate list is empty, the similarity-based fallback may be applied (Section 6.7).

6.6 Suggestion-Only Handoff Logic

The platform explicitly separates *suggesting* a handoff from *executing* a transfer:

- The recommender produces suggestions; it does not trigger an automatic handoff.
- The operator decides whether to accept a suggestion via the UI.
- Actual transfer events (e.g., `agent_transfer`) are logged as separate traces when performed through the appropriate operational workflow.

This separation has three practical benefits:

- **Safety.** It reduces the risk of undesired transfers caused by model error.
- **Auditability.** It preserves a clear causal chain: suggestion $\rightarrow$ operator action $\rightarrow$ eventual transfer.

- **Gradual rollout.** It enables production validation and feedback collection before enabling any automated behavior.

Figure 6.3 – Production UI for the suggestion-only handoff workflow. The panel displays the Top-2 recommended agents (*Retriever*: 98.8%, *Ticket creator*: 1.2%) with a HIGH confidence badge. The operator can accept individual suggestions or reject all. Feedback is collected via dedicated endpoints (Section 6.9) without triggering automatic transfers.

6.7 Guards, Overrides, and Similarity Fallback

The workflow includes guard rules that stabilize behavior under uncertainty.

6.7.1 Guards

The following guards are applied:

- **Anti-leakage enforcement** (user-only messages and strict temporal cutoff `created_at < suggestion_time`).

- **Deterministic ML ranking:** classifier candidates are sorted by descending probability.
- **ML-first fallback integration:** similarity candidates (when enabled) are appended after the ML list and do not override the ML ordering.
- **Empty or low-confidence handling:** if the Top-K list is empty or `confidence_label` is low, the system enables fallback candidate generation.

6.7.2 Similarity-Based Fallback

When the model output is uncertain (low confidence) or lacks candidates, the system can generate additional candidates using semantic similarity:

- Method: cosine similarity between the **conversation embedding** and cached embeddings of **agent descriptions**.
- Fallback size: `top_n = 2`.
- Score handling: similarity scores are rescaled into $[0, 1]$ for consistent presentation.

The final shortlist follows an **ML-first + append** strategy: the Top-K ranking produced by the classifier is preserved, and up to `top_n` similarity-based candidates are appended at the end of the list. This design treats similarity suggestions as a *coverage guardrail* (additional options for the operator) rather than a re-ranking signal. Because ML probabilities and cosine similarity scores are heterogeneous, the system does not re-sort across the two score types in production. As a consequence, the fallback cannot change the Top-1 recommendation in production; its intended effect is to increase shortlist coverage by providing additional options to the operator.

6.7.3 Fallback Usage and Offline Evaluation

To quantify the practical impact of the similarity fallback, we evaluated the production strategy on an offline evaluation set of 404 handoff decision points extracted from the full

Strategy	Coverage@1	Top-1 Acc	Top-3 Acc
ML + conditional fallback (production)	100.00%	97.77%	98.27%
Fallback-only	100.00%	1.73%	1.73%
No suggestions	0.00%	0.00%	0.00%

Table 6.1 – Offline evaluation of similarity fallback strategies on 404 test cases. No-suggestions is reported with accuracy set to 0 by convention, since the system outputs an empty list.

Module B log export. The fallback was triggered in 1/404 cases (0.25%), indicating that under the current confidence thresholds the classifier almost always provides usable candidates. While the trigger rate is low under the current thresholds, the mechanism remains valuable as a guardrail for rare or ambiguous edge cases and for graceful degradation when ML candidates are unavailable.

Table 6.1 reports Top-1 and Top-3 accuracy and Coverage@1 under three strategies: (i) the production behavior (ML + conditional fallback with ML-first + append), (ii) fallback-only, and (iii) no suggestions. The production strategy achieves strong overall performance (Top-1 97.77%, Top-3 98.27%) with full coverage. By contrast, fallback-only performs poorly (Top-1 1.73%), indicating that similarity over short agent descriptions is not sufficiently discriminative as a standalone ranking signal in this setting.

Finally, because the production integration appends similarity candidates after the ML ranking, the fallback is designed to improve *shortlist coverage* rather than to change the Top-1 prediction. This is reflected in the single triggered case (case_id=A), where the classifier was nearly tied between agents 2 and 3 (0.5066 vs. 0.4934, $\Delta \approx 0.013$) and the fallback appended an additional candidate (agent 1) as a third option.

On the fallback-triggered subset (1 case), Top-1 accuracy is 0% and Top-3 accuracy is 100% (n=1).

6.8 API Gateway and Frontend Integration

6.8.1 RAG Inference Endpoint

The RAG service exposes the main inference endpoint: `POST /api/v1/answer`. The response includes both the chat answer and the recommender payload, including:

- `answer`: the assistant text response,
- `active_responder`: active agent/assistant metadata,
- `sources`: retrieval sources (when applicable),
- `recommended_agents`: list of recommended agents,
- `suggestion_trace_id`: identifier used to link suggestions to feedback traces,
- `confidence_label`: categorical confidence,
- `model_version`: version identifier for the loaded recommender artifacts.

6.8.2 Gateway Normalization

The PHP/Laravel API gateway provides a stable contract to the frontend. Its responsibilities include:

- mapping response fields from `snake_case` to `camelCase`,
- validating payload types and required fields,
- enriching recommendations by resolving `agent_name` given `agent_id`.

This adapter pattern decouples frontend releases from backend internals and allows backend services to evolve without breaking UI consumers.

6.8.3 Frontend Consumption

The frontend consumes `recommendedAgents`, `confidenceLabel`, `modelVersion`, and `suggestionTraceId`. The UI displays the suggestion panel when available; otherwise, it silently hides it (Section 6.13), ensuring that chat functionality is never blocked by ML failures.

6.9 Feedback Loop: Accept/Reject Endpoints

The system collects operator feedback via dedicated endpoints:

- `POST /chat/handoff-suggestions/{suggestion_trace_id}/accept`
- `POST /chat/handoff-suggestions/{suggestion_trace_id}/reject`

Both endpoints write a feedback trace (`handoff_suggestion_feedback`) into `cnv_conversation_t` Critically, accept/reject actions **do not create** an `agent_transfer` event; they only record feedback for offline analysis and future retraining.

Feedback traces include at least:

- `suggestion_trace_id`,
- `action` (accept/reject),
- `selected_agent_id` (when applicable),
- `actor` (operator identity or system actor),
- `feedback_time`.

This design preserves a clean separation between *recommendation evaluation* and *operational execution*.

6.10 Model Storage and Versioning

6.10.1 Artifact Layout

Model artifacts are stored in the container filesystem under a mounted directory:

```
/app/models/recommender/production/v1/chatbot{chatbot_id}/
  - clf_recommender.joblib
  - agent_order.npy
```

This layout supports:

- **per-chatbot specialization** (different models per chatbot),
- **versioning** by directory (`v1`, `v2`, ...),
- environment-based pinning to a specific production version.

The `agent_order.npy` artifact is stored separately to avoid class-index mismatches. It defines the canonical ordering of agents used by the classifier and must match the order used during training.

6.10.2 Runtime Loading and Fallbacks

Artifacts are loaded on-demand and cached in memory. If artifacts are missing or corrupted, the service logs a warning and disables suggestions for that request (graceful degradation). This ensures that the core chat functionality remains operational even when ML artifacts are unavailable.

Rollback is performed by switching the pinned version path (e.g., `v1` $\rightarrow$ `v0`) through deployment configuration, without requiring changes to the frontend contract.

6.11 Testing and Validation

Integration quality is enforced with unit tests in the RAG service using `pytest`. Key tests include:

- **Temporal filtering:** verify strict cutoff `created_at < suggestion_time`.
- **Role filtering:** verify `role == "user"` is enforced.
- **Trace metadata presence:** verify recommendation traces contain required fields.
- **Feedback endpoints safety:** verify accept/reject do not produce `agent_transfer` events and only write feedback traces.

To avoid foreign key violations in test databases, fixtures create *FK-safe* parent records (e.g., conversation, message, and agent revision entities) prior to inserting traces and feedback.

End-to-end tests are not assumed in this description; however, the modular design makes them straightforward: the gateway can be tested against the RAG service with mocked LLM calls while asserting that the UI contract remains stable.

6.12 Observability and Feedback-Driven Iteration

The production integration emits structured traces for monitoring, auditing, and future evaluation. The main trace types include:

- `handoff_suggestion`
- `handoff_suggestion_feedback`
- `ml_recommender`

A `handoff_suggestion` trace contains, at minimum: `suggestion_time`, `conversation_id`, `chatbot_id`, `active_agent_id_pre`, `recommended_agents`, `confidence`, `audit`, and `model` metadata.

A feedback trace contains the linking identifier and action metadata: `suggestion_trace_id`, `action`, `selected_agent_id`, and `feedback_time`.

These traces enable the computation of offline metrics such as:

- **accept_rate:** fraction of suggestions accepted by operators,
- **time_to_accept:** delay between suggestion emission and operator acceptance,
- **fallback_rate:** frequency of similarity fallback usage,
- **error_rate:** fraction of turns where suggestions could not be produced.

The RAG service uses structured logging. The same trace and metric design supports drift monitoring and retraining triggers once sufficient feedback volume is collected.

6.13 Failure Modes and Graceful Degradation

The integration is designed so that ML failures never block the chat.

6.13.1 Common Failure Modes

The primary failure modes include:

- **Missing model artifacts** (e.g., `clf_recommender.joblib` absent),
- **Embedding error** (provider failure, timeout),
- **No candidates** (empty candidate set),
- **Timeout** in downstream dependencies.

6.13.2 Degradation Strategy

For each failure mode, the system:

- continues to generate and return the chat answer,
- omits `recommended_agents` (or returns an empty list),
- optionally uses similarity fallback when enabled and feasible,
- avoids surfacing blocking errors in the UI; instead, the suggestion box is hidden.

This approach preserves user experience and keeps the ML layer as an enhancement rather than a hard dependency.

6.14 Discussion

The production integration described in this chapter operationalizes the ML modules under real-world constraints: strict anti-leakage enforcement, deterministic feature building, versioned artifacts, and a trace-first approach to auditability. The *suggestion-only* handoff design provides a safe path to deployment by keeping the human operator in the loop [1], while still collecting feedback signals that can guide iterative improvement, recalibration, and (potentially) future automation.

In summary, the engineering choices—interface-driven modularity, explicit `suggestion_time` cutoffs, and robust fallback behavior—align the system with the goals of reliability, observability, and controlled evolution in production.

Chapter 7

Conclusions and Future Work

This thesis investigated the design, implementation, and production deployment of a machine learning system for agent handoff recommendation in Kora.ai, a multi-agent conversational platform developed by Twiper S.r.l. The work addressed three core objectives: semantic enrichment via auto-tagging (Module A), handoff recommendation (Module B), and safe production integration with operator validation.

7.1 Summary of Contributions

The main contributions of this work include:

- **A reproducible dataset extraction pipeline** that enforces strict anti-leakage constraints (timestamp filtering, feature whitelisting, group-wise splitting by `conversation_id`) to ensure valid offline evaluation. The pipeline produced 406 usable conversations for auto-tagging and 313 handoff events for recommendation training.

- **A multi-label semantic auto-tagger (Module A)** using text embeddings and One-vs-Rest logistic regression, achieving Macro-F1=0.695 on the test split (N=61), and Macro-F1=0.669 on a near-duplicate-filtered (deduplicated) test set (N=49). Learning curve analysis suggests that 2–3$\times$ more supervision (360 training conversations) would reach Macro-F1=0.72, providing a concrete data collection target.

- **A handoff recommender (Module B)** based on pre-trained embeddings (`text-embedding-3-smal` 1536 dimensions), achieving Top-1 accuracy=95.5% on the primary chatbot (C=2 eligible agents, $N_{\text{test}} = 66$). An ablation study confirmed that embeddings alone are sufficient in the current binary regime, with no measurable gain from auto-tag features or safe metadata.

- **A hybrid routing strategy** combining deterministic guards (purity ≥ 0.99, support ≥ 10) with ML fallback to ensure runtime coverage, alongside a similarity-based fallback for low-confidence cases.

- **A production integration in suggestion-only mode**, including versioned model artifacts, feedback collection endpoints, graceful degradation under ML failures, and monitoring infrastructure (traces: `handoff_suggestion`, `handoff_suggestion_feedback`, `ml_recommender`).

These contributions demonstrate that lightweight statistical models, supported by careful feature engineering and production-grade integration, can effectively support multi-agent orchestration under realistic constraints such as small data, noisy labels, and class imbalance.

7.2 Research Questions Revisited

This section revisits the three research questions posed in Chapter 1.

7.2.1 RQ1: Can lightweight ML models provide reliable handoff recommendations under limited supervision?

Answer: Yes, with important caveats.

For Chatbot 1 (C=2, 247 training rows, 66 test rows), the text-only recommender achieved Top-1 accuracy=95.5%, outperforming both the majority baseline (84.8%) and the router baseline (92.4%). Minority-class performance was reasonable despite severe imbalance (agent 3: F1=0.84, AP=0.96, 19 training examples).

For Chatbot 2 (C=2, 39 training rows, 45 test rows), the router baseline (Top-1=97.8%) outperformed ML (Top-1=88.9%), confirming that **policy-based routing is safer under extreme low-data conditions** (minority class: 3 training examples, few-shot regime). This motivates the hybrid router+ML strategy deployed in production.

Key insight: Pre-trained embeddings provide strong semantic representations that enable reasonable generalization even with 200–250 training examples, but minority-class robustness requires at least 20–30 examples per class. Below this threshold, deterministic guards should be prioritized.

7.2.2 RQ2: Do semantic auto-tags improve routing performance beyond a text-only baseline?

Answer: Not measurably in the current C=2 regime, but the infrastructure is valuable.

The ablation study (Section 5.7) showed that all four feature configurations (text-only, text+metadata, text+autotags, text+autotags+metadata) achieved identical Top-1 accuracy and MAP@2 on both chatbots. This suggests that **binary routing decisions are already well-separated in embedding space**, leaving no measurable headroom for additional features.

However, this null result is **regime-specific**:

- In C=2 settings with strong majority classes, embeddings capture sufficient signal
- In C>5 scenarios with overlapping domains (e.g., technical support on invoicing vs. pure invoicing queries), topic probabilities may provide complementary signals by disentangling cross-domain intents

Moreover, Module A provides **standalone value** for analytics, monitoring, and human-interpretable summaries, independently of its contribution to Module B performance.

7.2.3 RQ3: How can an ML-based router be deployed safely in production?

Answer: Through suggestion-only deployment, anti-leakage enforcement, hybrid routing, and graceful degradation.

The production integration (Chapter 6) operationalizes the following safety mechanisms:

- **Suggestion-only mode:** The model recommends agents but does not execute transfers autonomously. Final handoff decisions remain under operator validation, enabling feedback collection and gradual rollout.
- **Anti-leakage enforcement:** Runtime feature construction mirrors training constraints: user-only messages, strict temporal cutoff (`created_at < suggestion_time`), and explicit feature whitelisting prevent post-hoc signals from contaminating predictions.
- **Hybrid routing:** Deterministic guards (purity ≥ 0.99, support ≥ 10) bypass ML in near-deterministic regions, while ML handles ambiguous cases. A similarity-based fallback ensures coverage under low-confidence scenarios.
- **Graceful degradation:** ML failures (missing artifacts, embedding errors, timeouts) never block chat functionality. The UI silently hides suggestions when unavailable, preserving user experience.
- **Feedback loop:** Accept/reject endpoints collect operator feedback via `handoff_suggestion_feed` traces, supporting offline evaluation of production behavior without coupling feedback to operational transfers.

Key insight: Safe deployment of ML in production systems requires **interface discipline** (stable contracts, versioned artifacts), **observability** (structured traces, audit fields), and **fallback strategies** that preserve core functionality under ML degradation.

7.3 Limitations and Threats to Validity

This work is subject to several limitations that should be considered when interpreting results:

7.3.1 Small Sample Size and Class Imbalance

Both modules are trained on limited data (Module A: 406 conversations; Module B: 313 handoff events after filtering). This leads to:

- **High variance across splits:** Across-seed evaluation of Module A shows Macro-F1=0.497$\pm$0.048 (vs. single-split best-case=0.695), confirming that single-split results can be optimistic. This behavior is consistent with the instability and slow saturation typically observed in low-data learning curves [15].
- **Fragile minority-class predictions:** For Module B, minority classes have as few as 3–19 training examples, placing them in a few-shot regime where supervised learning is unreliable without aggressive class weighting or guard heuristics.
- **Limited generalization claims:** Results are dataset-specific and may not transfer to chatbots with different agent structures, conversation patterns, or domain vocabularies.

7.3.2 Near-Duplicates and Redundancy

Near-duplicate detection (Section 4.12.4) revealed that 19.67% of the auto-tagging test set has cosine similarity > 0.95 with training conversations. While this reflects real operational patterns (template requests, recurring workflows), it leads to:

- Mildly optimistic performance estimates (Macro-F1 drops by 3.7% when near-duplicates are excluded)
- Potential overestimation of generalization to novel user intents

Future dataset construction should enforce similarity-based deduplication across splits (e.g., exclude test conversations with similarity > 0.90 to any training conversation).

7.3.3 Binary Routing Regime (C=2)

Both evaluated chatbots have C=2 eligible agents after filtering, limiting the scope of conclusions:

- **Ablation study insensitivity:** In binary settings, Top-K metrics for K>1 are trivial, and additional features show no measurable gain because decision boundaries are already well-separated.
- **Uncertain multi-class scaling:** Extension to C>5 will likely require additional data, hierarchical labeling, or re-calibration of thresholds. The current results demonstrate feasibility but do not guarantee performance in true multi-class scenarios.

7.3.4 Operational Label Noise

Operational annotations (`topics_raw`, transfer targets) are derived from production logs and were not designed as ML labels. As widely discussed in the literature on label noise in supervised learning [4], such operational annotations may introduce systematic inconsistencies and weak supervision signals. This introduces:

- Inconsistent granularity (e.g., `SUPPORTO_TECNICO` as both domain topic and intent modifier)
- Potential mislabeling or incomplete coverage (all-zero topic rows, ambiguous transfer rationales)
- Difficulty in defining ground truth for error analysis

These noise sources are mitigated through deterministic macro-topic aggregation and qualitative error inspection, but fundamental label quality remains a limiting factor.

7.3.5 Production Metrics Unavailable

At the time of writing, the system is deployed in suggestion-only mode but **production acceptance metrics are not yet available**. Key unknowns include:

- Operator acceptance rate (fraction of suggestions validated)
- Time-to-accept distribution
- Real-world error patterns (do offline test errors reflect production failures?)
- Drift in conversation patterns over time

These metrics are critical for validating deployment success and should be prioritized in future monitoring efforts.

7.3.6 Proposed Monitoring Dashboard

While production acceptance metrics are not yet available, the suggestion-only deployment enables a concrete monitoring and evaluation plan.

7.3.6.1 Metrics to track (first 30/60/90 days)

- **30 days (instrumentation & baselines):** acceptance rate, median/P90 time-to-accept, suggestion coverage, fallback rate (guards vs. ML vs. failures), latency P95, error/degradation rate.
- **60 days (diagnostics):** breakdown by `chatbot_id` and suggested agent and confidence distribution drift.
- **90 days (stability & scaling):** trend monitoring for drift in request mix (topic prevalence) and fallback usage; decision on rollout expansion or tighter guard logic.

7.3.6.2 Alert thresholds (initial defaults)

- If acceptance rate $< 70\%$ for 7 consecutive days $\rightarrow$ investigate (trace samples, drift, top error clusters).
- If coverage $< 80\%$ for 3 days or degradation/error rate $> 1\%$ daily $\rightarrow$ inspect artifacts/embedding failures.
- If fallback rate $> 25\%$ for 3 days $\rightarrow$ re-check confidence thresholds and guard rules.

7.3.6.3 Online A/B test: router-only vs. hybrid

- **Arms:** (A) router-only suggestions, (B) hybrid (guards + ML fallback).
- **Randomization:** by `conversation_id` (stratified by `chatbot_id`), 50/50 split, 2 weeks.
- **Success:** higher acceptance rate in (B) (e.g., $+5$ pp) with non-worse median time-to-accept.
- **Stop conditions:** acceptance rate -10 pp vs. baseline for 3 days, or SLO violations (latency/error spikes).

7.4 Future Work

7.4.1 Data Collection and Minority-Class Expansion

Priority: High. The most impactful improvement is expanding the dataset, particularly for minority classes:

- **Target:** Collect 50+ examples per agent to enable robust supervised learning without guard heuristics
- **Mechanisms:**
  - ▶ Active learning: Identify ambiguous cases near decision boundaries and request operator labels,following standard active learning strategies for efficient data acquisition [18].
  - ▶ Extended deployment: Accumulate handoff events naturally as the system matures
  - ▶ Synthetic generation (with caution): Paraphrase or back-translate minority examples, validated by domain experts
- **Auto-tagging roadmap:** Learning curve analysis (Section 4.12) suggests 360 training conversations would reach Macro-F1=0.72

7.4.2 Multi-Class Expansion (C>5)

Priority: Medium. As the agent catalog grows, the routing problem will transition from binary to multi-class:

- **Expected challenges:**
  - Increased class imbalance (more rare agents)
  - Overlapping semantic domains (e.g., billing vs. invoicing vs. payments)
  - Higher ambiguity in Top-1 decisions
- **Mitigation strategies:**
  - Hierarchical labels: Primary domain + intent modifier (e.g., "invoicing + support" vs. "invoicing + creation")
  - Top-K evaluation (K=3–5) with ranking-based metrics (Mean Average Precision, Normalized Discounted Cumulative Gain)
  - Calibrated confidence thresholds per agent

7.4.3 Online A/B Testing and Production Metrics

Priority: High (immediate next step). Offline evaluation cannot fully predict production behavior. Key metrics to track:

- **Acceptance rate:** Fraction of suggestions validated by operators
- **Time-to-accept:** Delay between suggestion and operator action (proxy for cognitive load)
- **Error analysis:** Do offline test errors correlate with production rejections?
- **Drift monitoring:** Track embedding distribution shift, topic prevalence changes, and fallback usage over time

An A/B test comparing router-only vs. hybrid (router+ML) workflows would quantify the incremental value of ML under real operator workloads.

7.4.4 Advanced Modeling and Feature Engineering

Priority: Low (until data collection improves). With sufficient data, the following extensions could be explored:

- **Contextual embeddings:** Fine-tune a domain-specific encoder on conversation data (e.g., sentence-transformers on Italian business dialogues)
- **Conversational features:** Intent clustering, turn-level sentiment, or dialogue act classification as auxiliary signals
- **Multi-task learning:** Joint training of auto-tagging and handoff recommendation with shared representations
- **Neural architectures:** Gradient-boosted trees (LightGBM, XGBoost) or shallow neural networks as alternatives to logistic regression

However, given the current dataset size, these approaches are likely to introduce overfitting without measurable gains. **Data collection should precede architectural complexity.**

7.4.5 Annotation Quality and Guidelines

Priority: Medium. Improve label consistency through:

- **Refined guidelines:** Clarify ambiguous cases (e.g., "technical support on invoicing" vs. "invoicing only")
- **Active learning loops:** Flag low-confidence predictions for human review and relabeling

7.4.6 Expanded Guard Logic and Confidence Calibration

Priority: Low (current guards are sufficient). Potential refinements:

- Per-agent confidence thresholds (instead of global thresholds)
- Margin-based guards (trigger router if $p_{\text{top1}} - p_{\text{top2}} < \varepsilon$)

7.5 Closing Remarks

This thesis demonstrates that machine learning can augment multi-agent conversational systems even under severe data constraints, provided that:

1. **Anti-leakage discipline** is enforced rigorously at both training and inference time
2. **Simple, interpretable models** are prioritized over architectural complexity when data is scarce
3. **Hybrid strategies** combine ML with deterministic guards to leverage high-confidence policy rules
4. **Production integration** treats ML as an *enhancement* rather than a *dependency*, ensuring graceful degradation under failures

The suggestion-only deployment model, feedback collection infrastructure, and structured observability pipelines provide a solid foundation for iterative improvement. As the dataset grows and production metrics become available, the system can evolve toward more autonomous routing while maintaining operator trust and auditability.

Ultimately, the success of ML-based handoff recommendation will be measured not by offline test accuracy, but by **operator acceptance, reduced cognitive load, and improved user satisfaction**—metrics that can only be validated through sustained production deployment and feedback-driven iteration.

Bibliography

[1] B. Shneiderman, "Human-centered artificial intelligence: Reliable, safe & trustworthy," 2020.

[2] F. M. Zanzotto, "Human-in-the-loop artificial intelligence," *Journal of Artificial Intelligence Research*, vol. 64, pp. 243–252, 2019.

[3] S. Kaufman, S. Rosset, C. Perlich, and O. Stitelman, "Leakage in data mining: Formulation, detection, and avoidance," *ACM Transactions on Knowledge Discovery from Data*, vol. 6, no. 4, pp. 15:1–15:21, 2012.

[4] B. Frenay and M. Verleysen, "Classification in the presence of label noise: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 5, pp. 845–869, 2014.

[5] S. Newman, *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 2021.

[6] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, 2004.

[7] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," 2021.

[8] N. Sambasivan, S. Kapania, H. Highfill, D. Akrong, P. Paritosh, and L. M. Aroyo, ""everyone wants to do the model work, not the data work": Data cascades in high-stakes ai," in *Proceedings of the 2021 CHI Conference on Human Factors in Com-*

puting Systems. New York, NY, USA: Association for Computing Machinery, 2021, pp. 1–15.

[9] F. Ricci, L. Rokach, and B. Shapira, "Recommender systems handbook," in *Recommender Systems Handbook*, 2010, pp. 1–35.

[10] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[11] A. Neelakantan, T. Xu, R. Puri, A. Radford, J. M. Han, J. Tworek, Q. Yuan, N. Tezak, J. W. Kim, C. Hallacy, J. Heidecke, P. Shyam, B. Power, T. E. Nekoul, G. Sastry, G. Krueger, D. Schnurr, F. P. Such, K. Hsu, M. Thompson, T. Khan, T. Sherbakov, J. Jang, P. Welinder, and L. Weng, "Text and code embeddings by contrastive pre-training," 2022.

[12] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," 2020.

[13] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *International Journal of Data Warehousing and Mining*, vol. 3, pp. 1–13, 2009.

[14] J. Davis and M. H. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd International Conference on Machine Learning (ICML '06)*. Association for Computing Machinery, 2006, pp. 233–240.

[15] T. Viering and M. Loog, "The shape of learning curves: A review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 6, pp. 7799–7819, 2023.

[16] B. Plank, D. Hovy, and A. Søgaard, "Linguistically debatable or just plain wrong?" in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Baltimore, Maryland: Association for Computational Linguistics, 2014, pp. 507–511.

[17] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY: Springer, 2009.

[18] B. Settles, “Active learning literature survey,” 2009.

[19] T. Saito and M. Rehmsmeier, “The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets,” *PLOS ONE*, vol. 10, no. 3, p. e0118432, 2015.

[20] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, “On calibration of modern neural networks,” 2017.