# Thread Classificator Mail
# Brainstorming v3.0 (Feb 2026)

## Indice

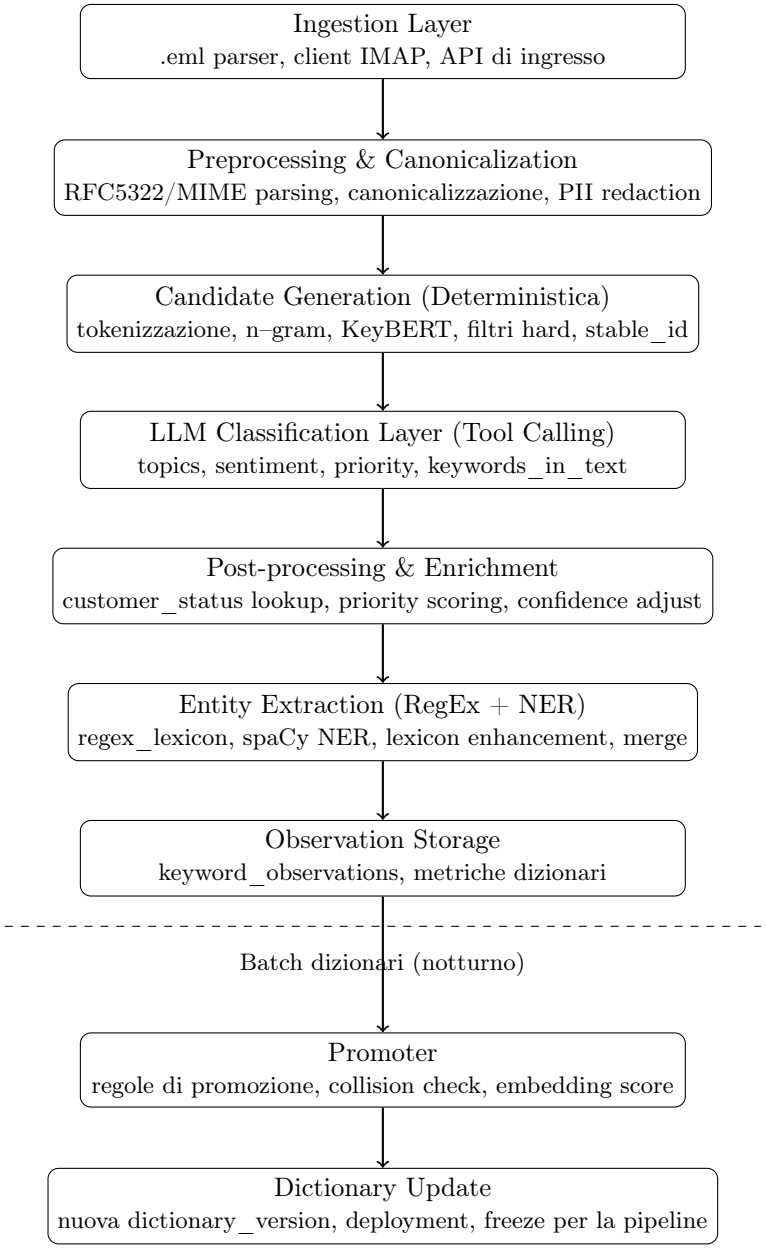
# 1 Descrizione del progetto

Il progetto *Thread Classificator Mail* ha come obiettivo la progettazione di una pipeline deterministica e auditabile per il triage automatico di email di customer service. A partire da messaggi in formato Internet Message Format (`.eml`) o flussi IMAP, il sistema esegue parsing, normalizzazione e classificazione multi–asse per assegnare a ciascuna email: `topics[]` multi–label, `customer_status`, `priority`, `sentiment` e un insieme di `keywords_in_text[]` utilizzate per aggiornare dinamicamente dizionari RegEx e NER di dominio.[file:1]

L'invariante chiave è il determinismo statistico: a parità di versione di dizionari, modelli, parser, stoplist, NER, schema e tool calling, lo stesso input produce lo stesso output, garantendo ripetibilità degli esperimenti, assenza di drift silenzioso e tracciabilità completa per audit e backtesting.[file:1]

La pipeline è strutturata in layer ben separati: ingestion e canonicalizzazione, generazione deterministica di candidate keywords, classificazione LLM tramite tool calling con output strutturati, arricchimento applicativo (customer status, priority, confidence), estrazione entità (RegEx + NER dinamico) e gestione di dizionari versionati con promoter deterministico per l'auto–espansione controllata del lessico.[file:1]

# Schema ZTK della pipeline di triage email



## 2 Obiettivo e invarianti

Obiettivo: prendere una mail (`.eml` o via IMAP), normalizzarla e produrre:

- `topics[]` (multi–label piatto, taxonomy chiusa);
- `customer_status` (`new`/`existing`/`unknown`, deterministico via CRM);
- `priority` (`low`/`medium`/`high`/`urgent`, ordinale);
- `sentiment` (`positive`/`neutral`/`negative`);
- `keywords_in_text[]` usate per aggiornare i dizionari RegEx + NER (dynamic lexicon).

Determinismo statistico: a parità di `dictionary_version`, `model_version`, `parser_version`, `stoplist_version`, `ner_model_version`, `schema_version`, `tool_calling_version`, la stessa mail produce lo stesso output (topics, sentiment, priority, customer_status, keyword, match RegEx/NER).[1]

### PipelineVersion

```
from dataclasses import dataclass

@dataclass(frozen=True)
class PipelineVersion:
    """Contratto di versione per garantire ripetibilita"""
    dictionary_version: int
    model_version: str # es. "deepseek-r1-2026-01" o "qwq-32b-2025-12"
    parser_version: str # es. "email-parser-1.3.0"
    stoplist_version: str # es. "stopwords-it-2025.1"
    ner_model_version: str # es. "it_core_news_lg-3.8.2"
    schema_version: str # es. "json-schema-v2.2"
    tool_calling_version: str # es. "openai-tool-calling-2026"

    def __repr__(self):
        return (
            f"Pipeline-{self.dictionary_version}-"
            f"{self.model_version}-{self.tool_calling_version}"
        )
```

Ogni run salva la `PipelineVersion` completa nei metadati per audit e backtesting.[1]

## 3 Piano di valutazione e metriche

### 3.1 Topics (multi–label)

Metriche standard: micro–F1, macro–F1, Hamming Loss, breakdown per etichetta (Precision/Recall/F1).[1]

```
from sklearn.metrics import hamming_loss, f1_score, classification_report
import numpy as np

def evaluate_multilabel_topics(y_true, y_pred, label_names):
    """
    y_true, y_pred: array [n_samples, n_labels] binari
    """
    micro_f1 = f1_score(y_true, y_pred, average='micro', zero_division=0)
    macro_f1 = f1_score(y_true, y_pred, average='macro', zero_division=0)
    hamming = hamming_loss(y_true, y_pred)
```

```
    report = classification_report(
        y_true, y_pred,
        target_names=label_names,
        zero_division=0,
        output_dict=True
    )
    return {
        "micro_f1": micro_f1,
        "macro_f1": macro_f1,
        "hamming_loss": hamming,
        "per_label": report
    }
```

### 3.2 Priority (ordinale)

```
from sklearn.metrics import cohen_kappa_score, confusion_matrix

PRIORITY_ORDER = {"low": 0, "medium": 1, "high": 2, "urgent": 3}

def evaluate_priority(y_true, y_pred):
    """Metriche per classificazione ordinale priority"""
    kappa = cohen_kappa_score(y_true, y_pred, weights='linear')
    cm = confusion_matrix(
        y_true, y_pred,
        labels=["low", "medium", "high", "urgent"]
    )
    y_true_ord = np.array([PRIORITY_ORDER[p] for p in y_true])
    y_pred_ord = np.array([PRIORITY_ORDER[p] for p in y_pred])
    distances = np.abs(y_true_ord - y_pred_ord)
    exact_match = np.mean(distances == 0)
    off_by_one = np.mean(distances <= 1)
    under_triage = np.mean((y_true_ord - y_pred_ord) >= 2)
    over_triage = np.mean((y_pred_ord - y_true_ord) >= 2)
    return {
        "kappa": kappa,
        "exact_match": exact_match,
        "off_by_one": off_by_one,
        "under_triage_rate": under_triage,
        "over_triage_rate": over_triage,
        "confusion_matrix": cm
    }
```

### 3.3 Sentiment

```
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix

def evaluate_sentiment(y_true, y_pred):
    acc = accuracy_score(y_true, y_pred)
    f1_macro = f1_score(y_true, y_pred, average='macro', zero_division=0)
    cm = confusion_matrix(
        y_true, y_pred,
        labels=["positive", "neutral", "negative"]
    )
    return {
        "accuracy": acc,
        "f1_macro": f1_macro,
```

```
        "confusion_matrix": cm
    }
```

### 3.4 Customer status e dizionari

Customer status deterministico: match rate, distribuzione delle confidenze, false–new rate dove è disponibile ground truth da audit.[file:1]

Per i dizionari dinamici: dimensione per label, collision_rate, quantili di frequenza documento, churn tra versioni, promotion_rate del promoter.[file:1]

### 3.5 Drift detection

Test del chi–quadrato sulla distribuzione delle label tra baseline e batch corrente, eseguito ogni 7–14 giorni.[file:1]

## 4 Ingestion e normalizzazione

Qui riporti le funzioni di parsing `.eml`, estrazione testo e canonicalizzazione della v2/v3 (già pronte nel PDF), includendo la classe `EmailDocument` e il flag di versione `CANONICALIZATION_VERSION` aggiornato.[1]

## 5 Candidati keyword deterministici

In questa sezione puoi incollare integralmente:

- `tokenize_it`, `ngrams`, `stable_id`;
- `build_candidates`;
- `enrich_candidates_with_embeddings` con KeyBERT;
- `score_candidate_composite`;
- `filter_candidates` con stoplist italiana e blacklist pattern.

Tutto il codice rimane quello del brainstorming v2, con eventuale aggiornamento a `STOPLIST_VERSION = "stopwords-it-2025.1"`.[1]

## 6 LLM Layer 2026: tool calling + JSON mode

Descrivi:

- la taxonomy `TOPICS_ENUM`;
- lo schema JSON ispirato a PARSE (max 5 topics, max 15 keyword, evidence corta);
- l'uso di tool calling nativo (OpenAI/DeepSeek/Qwen) per ottenere direttamente i parametri tipizzati (topics, sentiment, priority, keywords_in_text);
- i modelli consigliati per topics/sentiment e triage (reasoning models);
- la validazione multi–stadio (schema + business rules + warning di qualità).

[1]

## 7 Customer status, priority, confidence

Qui inserisci:

- la funzione `compute_customer_status` basata su CRM lookup + segnali testuali;
- la classe `PriorityScorer` con pesi parametrizzabili;
- la funzione di correzione delle confidence dei topic (LLM conf + keyword quality + evidence + collision penalty).

[1]

## 8 Dynamic dictionaries: RegEx + NER

In questa sezione riporti:

- lo schema dati (`label_registry`, `lexicon_entries`, `keyword_observations`);
- la classe `KeywordPromoter` con regole deterministiche di promozione;
- la generazione di regex sicure a partire dalle surface forms.

[1]

## 9 Entity Extraction + NER dinamico

Descrivi e (se vuoi) includi il codice per:

- estrazione entità via regex dai dizionari (`regex_lexicon`);
- NER spaCy (`it_core_news_lg-3.8.x`);
- enhancement con `ner_lexicon` come gazetteer;
- merge deterministico delle entità (RegEx > lexicon > NER).

[1]

## 10 Architettura di sistema (2026)

La pipeline complessiva è:

Ingestion (.eml/IMAP) → Canonicalization → Candidate Generation → LLM Tool Calling → Enrichment → Entity Extraction → Observation Storage → Promoter → Dictionary Update.

Stack suggerito:

- Backend: Python 3.11, FastAPI, PostgreSQL, Redis, Celery.
- NLP: spaCy 3.8.x, sentence-transformers, KeyBERT, LLM via OpenRouter.
- Monitoring: Prometheus + Grafana, alert su validation error rate, collision rate, drift, under-triage.

[1]

## 11 Checklist operativa

Punti chiave:

- versioning completo, inclusa `tool_calling_version`;
- logging dettagliato (input canonicalizzato, sezioni rimosse, output LLM, regole priority, delta dizionari);
- privacy by design (hash PII, retention policy GDPR);
- evaluation mensile su test set annotato;
- backtesting e A/B testing prima di ogni major release.

[1]

## Riferimenti bibliografici

[1] Brainstorming-Thread-Classificator-Mail-v2.pdf, "Enhanced with SOTA best practices, evaluation framework, and dynamic NER integration", 2024–2025.[file:1]