Lexicon Engineering and Gradient Boosting for Plant Pest Relation Extraction

Lev Kokovkin[1]

[1]National Research Nuclear University MEPhI, Russian Federation

Abstract

This paper describes my submission, under the team name *feel the borov*, to PestCLEF 2026, the plant-pest task of the LifeCLEF lab at CLEF 2026. The task is document-level knowledge-graph extraction: from each web document of the EPOP corpus (110 train, 55 development, and 82 test documents), a system must output typed triples over a strict schema of seven entity types and seven relations. Instead of fine-tuning a transformer or prompting a large language model, I build a fully reproducible, CPU-only pipeline from classical components: a taxonomy-aware surface-form lexicon (training aliases, NCBI Taxonomy grouping, coreference chains, and binomial-abbreviation expansion), schema-filtered candidate generation over sliding windows of one to three sentences, a 26-dimensional interpretable feature vector that mixes knowledge priors, lexical triggers, TF-IDF, layout, and co-occurrence signals, and a per-relation ensemble of ten gradient-boosting models. Per-relation decision thresholds and output caps are tuned on the development split, the ensembles are retrained on the union of train and development data, and three conservative post-processing rules refine the output. The system reaches a macro-F1 of 0.43992 on the official private leaderboard, up from 0.3069 on the local development split, and ranks third on the final private leaderboard. An accompanying data study, an iterative ablation, a classifier comparison, and an error analysis together show that per-relation threshold calibration and the train-derived lexicon, rather than the choice of learner, are what drive performance.

Keywords

PestCLEF, LifeCLEF, Information Extraction (IE), Document-level Relation Extraction (RE), Knowledge Graph (KG) construction, Natural Language Processing (NLP), Gradient Boosting (GB), EPOP corpus

1. Introduction

Plant pest surveillance depends on consolidating evidence from heterogeneous text sources: regulatory bulletins, scientific notes, and online reports. Reliable surveillance requires not only detecting that a pest is mentioned in a document, but also reconstructing its relationships with the host plants it attacks, the geographic regions where it has been reported, the dates of those reports, and the dissemination pathways through which it spreads. PestCLEF 2026, part of the LifeCLEF lab at CLEF 2026, asks systems to extract, for each input document, a Knowledge Graph (KG) describing exactly such biological and geographical interactions of plant pests [1, 2]. Unlike span-level named entity recognition, the task is document-level: an output edge is a typed triple over surface forms, possibly supported by multiple mentions, aliases, or sentences in the document. Document-level Relation Extraction (RE) has been studied extensively on biomedical benchmarks such as DocRED [3] and with neural models such as ATLOP [4], but the PestCLEF 2026 setting differs in two important ways: the schema is strictly typed and the training data is small enough to make a large fine-tuned encoder unattractive.

This setting carries several characteristics that motivated my design choices. Each characteristic has a direct impact on either candidate generation, the feature set, or the post-processing rules described later in this paper. Identifying them up front makes it easier to understand why a controlled, schema-aware pipeline is a strong fit for the EPOP corpus: each design choice in the rest of the paper can be traced back to one or two of these characteristics, and the ablation in Section 5 quantifies the contribution of every choice. The five characteristics that turned out to be most relevant in practice are summarised below.

CLEF 2026 Working Notes, 21 – 24 September 2026, Jena, Germany

levkokovkin@gmail.com (L. Kokovkin)

0009-0003-5946-5021 (L. Kokovkin)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

- The training data is small: 110 annotated documents in train, 55 in development, and 82 unlabelled in test (Section 3).
- The relation schema is strictly typed — each predicate accepts only a small subset of entity types in the subject and object slots.
- Entities surface under many aliases (full Latin names, English common names, taxonomic abbreviations). About 80% of gold knowledge graph edges contain more than one alias on the subject or object side.
- Predicates are heavily imbalanced; the most frequent relation (*Located_in*) accounts for over half of all gold edges in train and development.
- Geographic and temporal predicates — *Located_in*, *Occurs_on* — are the easiest to over-generate, because documents typically mention several locations and dates as background.

My Contribution: A Schema-Aware Feature-Based Pipeline without LLMs. These observations led me to a controlled, schema-aware pipeline rather than an end-to-end neural model. The strict typing of the relation schema, the small size of the training data, and the heavy imbalance across predicates make it feasible to enumerate every plausible candidate triple a priori and to rely on a compact set of interpretable features rather than on learned dense representations. This keeps the whole pipeline reproducible on commodity CPU without GPU access or proprietary models, while exposing the contribution of each design choice to a controlled ablation. With this perspective in mind, the contributions of this working note are:

- A reproducible data study of the EPOP corpus, covering surface-form coverage between train and development, sentence-level evidence locality per predicate, type co-occurrence within sentences, and layout tag distribution.
- A feature-based document-level Relation Extraction (RE) pipeline that combines a taxonomy-aware lexicon, schema-filtered candidate generation in sliding sentence windows, interpretable features, and a relation-specific Gradient Boosting (GB) ensemble.
- An ablation that isolates the contribution of feature groups, the ensemble size, the sentence window, the per-relation pruning, and the post-processing rules.
- A qualitative error analysis on the development split that exposes the remaining failure modes and points to concrete future work.

Submissions are evaluated as macro-averaged Precision, Recall, and F1 over normalised triples; locally, I compute the same statistic on the development split and average across documents. My final submission under team "feel the borov" obtains a private-leaderboard macro-F1 of 0.43992 on the official leaderboard (0.44014 public), ranking third on the final private leaderboard. This starts from a local development macro-F1 of 0.3069 before retraining on the union of train and development data.

Structure of the Paper. This paper describes my participant submission to PestCLEF 2026. Section 2 reviews related work; Section 3 formalises the task, describes the EPOP corpus, and reports a data study; Section 4 describes the methodology, from the taxonomy-aware lexicon and schema-filtered candidate generation through the feature set to the per-relation classifier and the post-processing rules; Section 5 reports the experimental setup, the main results and per-relation breakdown, the iterative ablation and component sweeps, the classifier comparison, and the error analysis; Section 6 discusses limitations; and Section 7 concludes the paper and proposes future directions.

2. Related Work

The Related Work below is deliberately organised by the four methodological pillars on which the present submission stands — document-level Relation Extraction (RE) on a typed schema, dictionary-based entity matching with taxonomy normalisation, feature-based classification with Gradient Boosting (GB) ensembles, and prior CLEF participations on closely related Information Extraction (IE) tasks —

rather than as a broad survey of biomedical NLP. Each pillar is the subject of an explicit design choice that will reappear in Section 4.

Document-level Relation Extraction on Typed Schemas. Document-level RE generalises sentence-level RE to scenarios where the evidence for a single relation is distributed across multiple sentences and where one entity may surface under several mentions and aliases. The DocRED benchmark [3] popularised the document-level formulation by collecting a large-scale dataset annotated with inter-sentential relations between entity clusters. ATLOP [4] subsequently introduced adaptive thresholding for the long-tailed distribution of relation labels and localised context pooling for entity pairs, and remains a standard baseline for document-level RE. PestCLEF 2026 shares the document-level setting with these benchmarks, but it adds two specific constraints that strongly shape the design space: a strictly typed predicate schema with seven entity types and seven relations, and a small training set in which roughly 80% of gold knowledge graph edges already carry multiple aliases on the subject or object side. The pipeline in this paper exploits both constraints by filtering candidates against the type schema during generation (Section 4.3) and by expanding every gold edge into the cross product of its subject and object aliases during training label construction.

Plant Pest Information Extraction and the EPOP Corpus. Automated text mining of plant pest, plant disease, and crop protection literature has gained traction in recent years, driven by the need to monitor the spread of invasive species and emerging pathogens from heterogeneous web sources. PestCLEF 2026 builds on this line of work by releasing the EPOP corpus [5] of 247 expert-annotated plant pest web documents and by framing the extraction task as a strongly typed knowledge graph problem [1, 2]. The annotation schema distinguishes seven entity types (*Pest*, *Disease*, *Plant*, *Vector*, *Location*, *Date*, *Dissemination_pathway*) and seven typed predicates, which together support both ecological relations (e.g. which pests attack which plants) and surveillance metadata (e.g. where and when a pest has been reported).

Dictionary-Based Entity Matching with Taxonomy Normalisation. When the surface lexicon is moderate in size and entities recur under many aliases, dictionary-based entity matching combined with taxonomy-aware normalisation is a strong source of recall, especially for biological organisms whose Latin names follow a stable binomial nomenclature. The NCBI Taxonomy [6] is the de facto standard nomenclature for such organisms, and the EPOP corpus already provides per-mention NCBI taxonomy normalisations for *Pest*, *Disease*, *Plant*, and *Vector* entities. Building on these annotations, the proposed system constructs a surface-form lexicon from the training data, groups all surfaces that share the same NCBI taxonomic identifier into a single equivalence class, and propagates the majority type across the class. Identity coreference chains released with EPOP are then used to collapse the remaining within-document aliases. To recover common in-text abbreviations such as "*B. dorsalis*" for "*Bactrocera dorsalis*", the pipeline additionally generates the form "*G.* species" for every two-token lexicon entry whose first token has more than two characters. The two recall signals complement each other: only 58% of dev entity mentions have an exact surface match in the training lexicon, but for the subset of dev mentions that carry an NCBI taxonomy normalisation (about 45% of the development split), 84% of those mentions share their taxon identifier with a training mention. NCBI grouping is therefore a strong recall signal on the biologically-named subset of entities, while abbreviation expansion and coreference are needed to recover the rest, which together directly motivate the lexicon design in Section 4.2.

Feature-Based Classification with Gradient Boosting Ensembles. Gradient Boosting [7], as implemented by `GradientBoostingClassifier` in scikit-learn [8], remains a competitive learner for small, schema-constrained classification problems when paired with carefully engineered interpretable features. Compared with end-to-end neural approaches, gradient boosting has three properties that matter for the present task: it is robust to small training sets (110 documents in train), the full pipeline (candidate generation, feature extraction, and ten-seed ensemble fitting) can be retrained on the union of

train and development data within a few minutes on commodity CPU, and the per-feature contribution to a decision is directly inspectable, which makes the ablation in Section 5 interpretable feature group by feature group. The pipeline therefore trains one binary classifier per relation, with each classifier itself an ensemble of ten gradient boosting models trained on different random seeds whose probabilities are averaged at inference time. Probability averaging across the ensemble produces a smoother score surface for the subsequent per-relation threshold and output-cap tuning, and reduces sensitivity to a single unlucky seed — a phenomenon that has also been noted in recent biomedical IE participations [9].

Prior CLEF Participations with Classical ML and Schema-Aware Tagging. Two recent CLEF working notes are particularly close in spirit to the present submission. Pamio and Di Nunzio [10] compared a hand-engineered Conditional Random Field model [11] with a fine-tuned BERT-based [12] system on the GutBrainIE 2025 biomedical NER and RE tasks, showing that classical feature-based pipelines remain informative baselines for biomedical IE. Han and Liu [9] obtained the top-ranked NER result at GutBrainIE 2025 using a BiomedBERT model with a Conditional Random Field layer, illustrating that even at the high end of the leaderboard schema-aware tagging architectures can outperform larger generative models on small biomedical datasets. Graphwise [13] placed an end-to-end neural pipeline (GLiNER [14] for NER, BioBERT [15] for RE) on the same biomedical task and is cited here as a contrasting design point. Within LifeCLEF, the closest sibling lab to PestCLEF, TheHeartOfNoise at PlantCLEF 2025 [16] demonstrated that careful data preprocessing alone can dominate model architecture choices, which motivates the same level of care in the lexicon construction and candidate generation stages of the proposed pipeline. None of these prior participations addresses the typed knowledge graph extraction task targeted by PestCLEF 2026, which is the first CLEF lab specifically targeting plant pest documents within this knowledge graph construction setting.

3. Task and Dataset

PestCLEF 2026 is the plant pest extraction task of LifeCLEF at CLEF 2026 [1, 2]. Participants receive scientific and regulatory web documents about plant pests and, for each document, must produce a Knowledge Graph (KG): a set of typed edges that describe the biological and geographical interactions of the pests mentioned in the document. The corpus shared with participants is the EPOP corpus [5] of 247 expert-annotated documents, split into 110 training, 55 development, and 82 test documents. Train and development documents carry expert annotations of entities, identity coreferences, binary text-bound relations, and a gold knowledge graph; test documents provide only the raw text and the layout description, and the gold output is withheld by the organisers.

Annotation Schema and Argument Typing. The schema fixes seven entity types and seven predicates with strict argument typing, summarised in Table 1. Argument typing is enforced at two levels of the pipeline in Section 4: at candidate generation time (Section 4.3) a candidate triple is created only if the subject type set intersects the predicate's subject types and the object type set intersects the predicate's object types, and at training-label construction time gold edges are expanded into the cross product of subject and object aliases. The predicate set combines ecological relations (e.g. which pest *Affects* which plant, which vector *Transmits* which disease) with surveillance metadata (e.g. where a pest is *Located_in*, when a pest is first *Occurs_on* reported).

Table 1: Entity types accepted by each predicate. Argument typing is verified against the gold annotations and is also enforced in the candidate generator implemented in this paper.

Predicate	Subject types	Object types
Affects	Disease	Plant
Causes	Pest	Disease
Dispersed_by	Disease, Pest	Dissemination_pathway
Found_on	Pest, Vector	Plant, Dissemination_pathway
Located_in	Disease, Pest, Plant, Vector	Location
Occurs_on	Disease, Pest, Plant, Vector	Date
Transmits	Vector	Disease, Pest

Entity Normalisation Resources. The EPOP corpus links each entity mention to a controlled vocabulary whenever possible, which means the dataset offers more than raw surface forms. The four resources used by the annotators and their coverage on the training split are:

- **NCBI Taxonomy** [6] for all biological organisms — *Pest* (953 mentions), *Plant* (669), and *Vector* (78).
- **GeoNames** [17] for all *Location* entities (1 055 mentions).
- **Plant_Disease ontology** for all *Disease* entities (234 mentions).
- **OntoBiotope** [18] for all *Dissemination_pathway* entities (129 mentions).

The *Date* entity type carries no normalisation by design: dates are encoded as raw surface strings such as "in 2015" or "March 2018" and have to be processed as free text. Overall, 87.7% of train entities carry at least one normalisation, which makes taxonomy-driven entity matching a strong recall signal for five of the seven entity types (Section 4.2).

Coreference Chains. Within each document, the corpus also releases identity coreference chains: ordered lists of entity ids that refer to the same real-world object. The train split contains 164 chains over 356 entity references, with a mean chain size of 2.17 aliases and a maximum of 4 aliases per chain; the development split contains 81 chains over 177 references. A typical chain links a full Latin name (e.g. *Bactrocera dorsalis*), an English common name (e.g. "oriental fruit fly"), and a taxonomic abbreviation (e.g. "B. dorsalis") that the pipeline must treat as the same biological entity.

Knowledge-Graph Alias Lists. A single gold knowledge-graph edge can carry several surface forms on the subject side and several on the object side: all of them are considered equivalent aliases of one underlying entity. In the train split, 79.9% of gold edges carry more than one alias on the subject or object side, with a mean of 2.95 aliases per subject list and 1.59 aliases per object list. The development split shows the same pattern (78.4% multi-alias edges, 2.86 and 1.53 aliases respectively). For training and evaluation the pipeline expands every gold edge into the cross product of its subject and object alias lists, so that any pair-level prediction that matches one alias from each side counts as a true positive.

Data Format. Each document is provided as a JSON record with four top-level fields: `doc_id`, `layout`, `text_bound_annotations`, and `knowledge_graph`, accompanied by the document text as a separate `.txt` file. The `layout` field lists character-level offsets for HTML-like tags — `h1`, `h2`, `h3`, `p`, `i`, `b`, `li`, `sup`, `sub` — which the pipeline uses to detect headers and first paragraphs. The `text_bound_annotations` field, present only in train and development, contains lists of entities (with offsets, type, surface form, and normalisations), binary relations with typed argument slots, and identity coreference chains. The gold `knowledge_graph` is a list of edges; each edge carries one predicate and may list multiple subject and multiple object surface forms as described above. Figure 1 shows an abridged example.

```
{
  "doc_id": "104650",
  "layout": [
    { "tag": "h1", "offsets": [0, 83] },        // title
    { "tag": "p",  "offsets": [84, 1759] },     // first paragraph
    { "tag": "i",  "offsets": [1852, 1858] },   // italicised name
    ...
  ],
  "text_bound_annotations": {
    "entities": [
      { "id": "T45", "type": "Pest",
        "form": "Tomato Brown Rugose Fruit Virus",
        "offsets": [[19, 50]],
        "normalizations": [
          { "resource": "NCBI_Taxonomy", "reference": "1761477" }
        ]
      },
      { "id": "T46", "type": "Location",
        "form": "Mediterranean",
        "offsets": [[59, 72]],
        "normalizations": [
          { "resource": "GeoNames", "reference": "12217088" }
        ]
      },
      ...
    ],
    "relations": [
      { "id": "R1", "type": "Located_in",
        "args": { "Object": "T45", "Location": "T46" } },
      ...
    ],
    "identity_coreferences": [ ["T65", "T67"], ["T47", "T48"] ]
  },
  "knowledge_graph": [
    { "predicate": "Occurs_on",
      "subject": [ "ToBRFV",
                   "Tomato brown rugose fruit virus",
                   "Tomato Brown Rugose Fruit Virus" ],
      "object":  [ "in 2015" ] },
    ...
  ]
}
```

Figure 1: Abridged JSON record for a representative training document. Comments after `//` are illustrative and are not part of the corpus.

Splits and Sizes. Table 2 summarises the split sizes and annotation volumes. The corpus is small (247 documents in total) and skewed across splits: train and development together provide 165 documents with full annotations, while the unlabelled test split contains 82 documents on which the official leaderboard is computed.

Table 2: Split sizes and annotation volumes on the EPOP corpus. The test split is unlabelled by design: only the input text and the layout description are released to participants.

Split	Docs	Entities	Coref chains	TBA relations	KG edges
train	110	3 482	164	2 217	1 374
dev	55	1 615	81	1 010	656
test	82	—	—	—	—

Output Format and Evaluation. A submission is a CSV file with one row per test document. Each row contains the document identifier and a JSON-serialised list of triples of the form $(\text{subject}, \text{predicate}, \text{object})$. Both subject and object are surface strings drawn from the document or from any of its aliases; predicates take their values from the seven-element schema in Table 1. Locally, I evaluate the pipeline by computing the macro-averaged F1 over the set of normalised triples $(\text{predicate}, \text{norm}(s), \text{norm}(o))$ per document, then averaging across documents.

3.1. Data Study

Before designing the system I ran a data study on the local train, development, and test files. All numbers in this subsection are reproducible from `eda.py` and `scripts/build_paper_assets.py`. Each subtopic is presented as a short titled block; only the findings that directly shaped a design decision are kept as a table or figure, the rest are summarised in prose.

Document length. Documents are short: a median of 374–406 words (mean ≈ 470, around 21–24 sentences) across the three splits, with a long right tail reaching 4 825 words in train. This right skew justifies a sentence-window candidate generator rather than a full-document one.

Annotation volume and normalisation coverage.

Table 3: Entity annotation volume per split. *Normalised entities* counts mentions with at least one normalisation in NCBI Taxonomy, GeoNames, Plant_Disease, or OntoBiotope.

Split	Entities	Unique surfaces	Coref chains	Normalised entities
train	3482	1309	164	3054
dev	1615	752	81	1396
test	0	0	0	0

Train carries 3 482 mentions over 1 309 unique surfaces, and 87.7% of them are normalised; the dominant normalisation resource is NCBI Taxonomy [6], which makes taxonomy-aware grouping a viable recall strategy in Section 4.2. Most documents carry 10–60 mentions, with a long tail above a hundred that yields thousands of candidate pairs per dense document.

Entity-type distribution. The seven entity types are highly imbalanced in train: *Location* (1 042 mentions) and *Pest* (908) together exceed half of all mentions; *Plant* (663), *Date* (419), and *Disease* (234) form the middle; and *Vector* (78) and *Dissemination_pathway* (138) are an order of magnitude rarer, with dev counts tracking the same shape. This imbalance drives two design choices. The *Vector* classifier sees fewer than 80 positive examples, which motivates the 10-seed ensemble of Section 4.5 to damp the high variance of any single fit; conversely *Location* is so abundant that the lexical trigger for *Located_in* fires in almost every sentence, so the per-relation output cap is what stops that predicate from flooding the prediction set. Finally, none of the 419 *Date* mentions carries a controlled-vocabulary normalisation — dates are free text — which makes the bad-date filter and same-sentence propagation of Section 4.6 the only defence against spurious *Occurs_on* edges.

Predicate distribution and alias expansion.

Table 4: KG predicate distribution before and after alias expansion. *Raw* counts gold edges as released; *expanded* counts the cross product of subject and object alias lists on each edge.

Relation	Train raw	Train expanded	Dev raw	Dev expanded
Affects	50	458	30	254
Causes	44	283	27	137
Dispersed_by	32	111	16	77
Found_on	233	1583	106	478
Located_in	781	3109	377	1482
Occurs_on	209	653	95	279
Transmits	25	127	5	47

Located_in alone accounts for over half of all train edges and dominates the next predicate by a factor of three; the four rarest predicates together account for less than 11% of train edges, which is the single strongest argument for per-relation classifiers and per-relation output caps. After expansion, train volumes grow by a factor of 4.20–4.60, and 79.9% of gold edges already carry multiple aliases on either side.

Evidence locality of gold relation pairs. String-matching the expanded gold pairs against sliding windows shows that only 35.9% of train pairs are recoverable within a single sentence, rising to about 61% at a 3-sentence window; roughly 39% remain unreachable by any short-window generator, which is the hard upper bound on candidate-side recall. The per-predicate view in Figure 2 shows where that ceiling comes from.

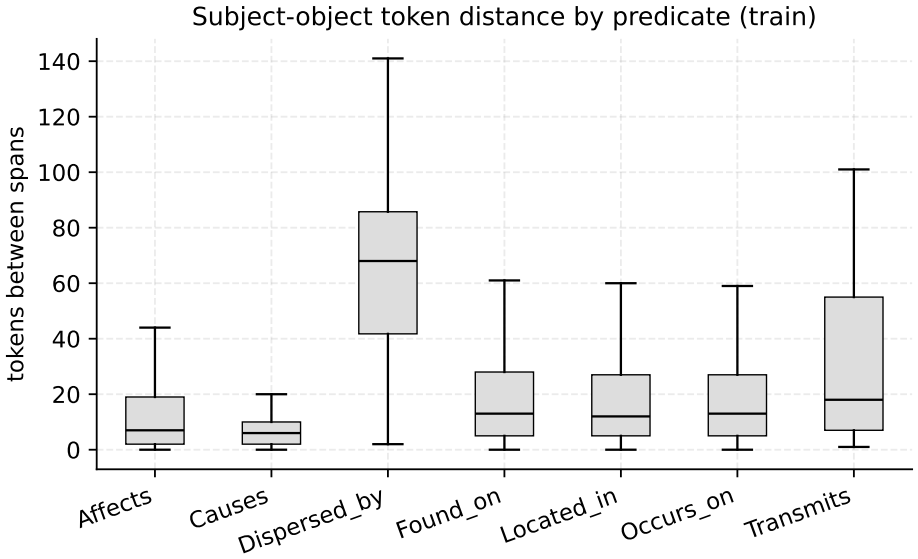

Figure 2: Subject-object token distance for gold text-bound relations in the train split, by predicate. Outliers are omitted.

Causes and *Affects* are by far the most local (medians of 6 and 7 tokens); *Located_in*, *Found_on*, and *Occurs_on* sit in the middle (medians 12–13); *Dispersed_by* stands out with median 68 tokens, which means many gold *Dispersed_by* edges are structurally unreachable by any short-window generator.

Train-to-dev lexicon overlap.

Table 5: Train-to-dev entity lexicon overlap, measured at three granularities. The NCBI column is computed over the subset of dev mentions that carry an NCBI normalisation.

Split	Mentions	Surface seen	Unique seen	NCBI tax ids seen
dev	1615	937 (58.0%)	294/752 (39.1%)	612/727 (84.2%)

Surface coverage is moderate (58% of dev mentions seen verbatim in train), but for the 727 dev mentions that carry an NCBI normalisation, 84.2% of their taxon identifiers are already in train — exactly the gap that the taxonomy-aware lexicon in Section 4.2 is designed to close.

Sentence-level type co-occurrence. At the sentence level, *Location* co-occurs most with *Pest* (191) and *Plant* (169), mirroring the *Located_in* predicate, while *Date* co-occurs most with *Pest* (96) and *Plant* (88), so gold *Occurs_on* edges are biased toward *Pest* and *Plant* subjects.

Layout tag distribution. The training documents are dominated by paragraphs (p, 1 266 spans) and italic spans (i, 380), with headers (h1–h3, 234 combined) and emphasis tags (b, 227) less frequent but informative for candidate scoring: binomial Latin names are conventionally italicised, and the focal pest of a document is often named in its title.

4. Methodology

4.1. System Overview

Figure 3 sketches the full pipeline at a glance. The system has two phases, each described in detail in the subsections below.

In the **offline preparation phase**, the system reads the train JSON annotations once and builds four reusable artefacts: a surface-form lexicon with a per-alias type map, a knowledge cache of all triples seen in the training knowledge graph, alias-level prior counts for every predicate, and coreference equivalence classes (Section 4.2). A single TF-IDF vocabulary is also fit once over the concatenation of train, development, and test texts so that document-level salience features can be computed consistently across splits.

In the **per-document inference phase**, every input document goes through five sequential stages: (i) detection of known entity mentions through longest-first lexicon matching; (ii) schema-filtered candidate generation over sliding sentence windows of size 1, 2, and 3 (Section 4.3); (iii) extraction of a 26-dimensional interpretable feature vector per candidate (Section 4.4); (iv) scoring with the relation-specific gradient boosting ensemble trained earlier (Section 4.5), followed by pruning with the tuned per-relation threshold τ_r and output cap L_r; and (v) three conservative post-processing rules (Section 4.6). The final output is a CSV submission file with one JSON-serialised knowledge graph per test document.

JSON + raw text + layout $\longrightarrow$ lexicon, taxonomy, coreference, priors $\longrightarrow$ entity mention detection $\longrightarrow$ schema-filtered candidate generation in sentence windows $\longrightarrow$ 26-dimensional feature extraction $\longrightarrow$ relation-specific gradient boosting ensemble (10 seeds) $\longrightarrow$ tuned per-relation threshold and output cap $\longrightarrow$ post-processing (bad-date filter, conservative propagation, coreference dedup) $\longrightarrow$ submission CSV.

Figure 3: Overview of the proposed pipeline.

4.2. Entity Lexicon and Normalisation

The lexicon is constructed in four steps that progressively enrich a plain surface-form dictionary into a taxonomy-aware, alias-resolved entity table. The lexicon uses only the training annotations; the

development split is reserved for hyperparameter tuning in Section 4.5.

Step 1: Surface-form collection. Surface forms are collected from two sources of the training JSON: the `text_bound_annotations.entities` field (gold entity mentions with offsets and types) and the `knowledge_graph` field (alias lists on every gold edge). Each surface form is paired with a type counter incremented by the matching entity type. To prevent generic words from polluting the lexicon, I discard 37 manually curated stop-words (*disease*, *plant*, *vector*, *location*, *date*, *area*, *region*, *country*, *season*, *year*, *species*, *host*, *leaf*, *soil*, …) and any form shorter than three characters.

Step 2: Coreference collapse. Each identity coreference group is collapsed to a single canonical surface form (the longest member). All non-canonical members are mapped to the canonical one in a chain table that is later consulted during output deduplication (Section 4.6).

Step 3: NCBI Taxonomy expansion. For every gold entity carrying an NCBI Taxonomy normalisation, I group all surface forms that share the same taxon identifier and propagate the majority type across the group. This step is what lifts the matching ceiling from 58% (exact surface match) to 84% (NCBI identifier match) on the development split for biologically-named entities (Section 3.1). The three other normalisation resources released with the EPOP corpus — GeoNames for *Location*, Plant_Disease for *Disease*, and OntoBiotope for *Dissemination_pathway* — are present in the gold annotations but are *not* used by the present pipeline, which only consumes NCBI normalisations; extending the same grouping to the remaining resources is left for future work (Section 7).

Step 4: Binomial abbreviation expansion. A common pattern in plant pathology is to abbreviate the genus in a two-token Latin name (e.g. *Bactrocera dorsalis* → *B. dorsalis*). For every two-token surface form in the lexicon whose first token has more than two characters, I add the abbreviated form "*G.* species" with the same type map. This recovers a frequent class of in-text mentions that would otherwise miss the lexicon match.

Final type map. For each normalised alias, the final type set retains all entity types whose count reaches at least 25% of the maximum count observed for that alias. The threshold preserves biologically plausible ambiguities (e.g. a name that is reported both as a *Vector* and as a *Pest*) while suppressing one-off labelling noise. I did not tune this cutoff as an independent hyperparameter: it was chosen as a conservative majority-ratio rule after inspecting the ambiguity pattern of the training aliases, and the resulting type-confusion errors are analysed explicitly in Section 5.5.

4.3. Candidate Generation

Candidate generation runs document by document and turns a raw document into a list of typed triple candidates that the classifier will later score. It proceeds in three steps.

Step 1: Mention detection. I scan the normalised document text for every entry of the lexicon built in Section 4.2, longest entry first to avoid shadowing (e.g. "*Bactrocera dorsalis*" is matched before "*Bactrocera*"). For each match, I record the normalised form, the surface form actually observed, and the entity-type set inherited from the lexicon.

Step 2: Sentence segmentation by layout. The document is split into sections using its `layout`: any header tag (`h1`, `h2`, `h3`) breaks a section. Each section is then split into sentences with a punctuation-based splitter that respects common abbreviations (*e.g.*, *i.e.*, *etc.*). Sentences shorter than five characters are dropped.

Step 3: Schema-filtered windowing. Sliding sentence windows are a standard locality prior in document-level Relation Extraction [3, 4]. I iterate over windows of one, two, and three consecutive sentences. Inside each window, every unordered pair of present entities is tested in both directions (s, o) and (o, s). A candidate (s, r, o) is accepted only if the type set of s intersects the subject types of predicate r *and* the type set of o intersects the object types of r, as listed in Table 1. Accepted candidates are de-duplicated by the key $(\text{predicate}, \text{norm}(s), \text{norm}(o))$ before scoring.

Layout flags exposed as features. Two boolean signals are computed during this stage and exposed to the classifier in Section 4.4: a candidate receives a *header boost* flag if at least one of its arguments appears inside a layout header span, and a *first-paragraph* flag if the candidate window starts at the very first sentence of the document.

4.4. Feature Engineering

Each candidate is mapped to a 26-dimensional vector. The features were designed to be cheap to compute, interpretable, and complementary across six functional groups (Table 6). The precise definition of every individual feature is in the `extract_features` function of the reference implementation `main_0.44.py`.

Table 6: Functional groups of the 26 features used by the system. Counts in parentheses are the number of scalar features in the group.

Group (size)	Contents
Knowledge priors (5)	Logarithmic frequency of the predicate, of the subject conditional on predicate, of the object conditional on predicate, of the $(\text{subj}, \text{obj})$ pair conditional on predicate, plus a binary indicator for whether the exact normalised triple was seen in training.
Lexical pattern (1)	Binary indicator for a regular-expression trigger of the predicate inside the candidate window (e.g. "caused by" for *Causes*, "transmits" for *Transmits*).
Structural (5)	Negation inside the in-between context, exponential decay of token distance, subject-before-object indicator, header boost, first-paragraph indicator.
Surface (4)	Length of subject and object surfaces (clipped to 20 characters), a "looks like a date" indicator, a "looks like a location" indicator.
TF-IDF (8)	Maximum, mean, and sum TF-IDF of the subject and object tokens in the document; subject share of the combined TF-IDF; mean TF-IDF of the context tokens between subject and object.
Frequency and co-occurrence (3)	Logarithmic mention count of the subject in the document, the same for the object, and the logarithmic sentence-level co-occurrence count of the unordered pair.

Knowledge priors. Priors are estimated on the training set only and treat each predicate as an independent multinomial over its (subject, object) pairs. I log-smooth all counts as $\log(1 + c)$ to compress the long tail.

Lexical patterns. For each predicate I maintain a list of regular-expression triggers (e.g. "causes", "caused by", "leads to" for *Causes*; "vector of", "transmits", "transmission of" for *Transmits*; "found on", "detected on", "host", "feeds on" for *Found_on*; "reported in", "detected in", "found in" for *Located_in*; and "during", "since", "between", "from" for *Occurs_on*). The patterns are intentionally broad and are used only as a single binary hint rather than as a rule-based decision. The full REL_PATTERNS table is in `main_0.44.py`.

TF-IDF. I fit a single TF-IDF vectoriser [19] over the concatenated corpus of train, development, and test documents (lowercased, word tokens, 50 000-feature cap). For each candidate I compute six entity-level TF-IDF statistics, a ratio that captures which argument is more salient inside the document, and the mean TF-IDF of the tokens that fall *between* the two arguments inside the window.

Co-occurrence. Sentence-level co-occurrence is computed once per document: for each sentence I collect the entities mentioned in it and increment a counter for every unordered pair. The logarithmic count is used as the feature value.

4.5. Relation Classification and Tuning

Once a candidate has a feature vector, the classifier decides whether it is a true relation instance. Training, threshold selection, and final retraining are organised as three sequential stages.

Why Gradient Boosting? The choice of learner was validated empirically rather than assumed. Table 11 (Section 5.2) compares three CPU classifier families under the *identical* fully-tuned pipeline — all 26 features, per-relation threshold and output-cap tuning, and the three post-processing rules — varying only the learner and, for gradient boosting, its number of trees. The headline result is that, once the per-relation thresholds are tuned, the classifier family barely matters: logistic regression, a random forest [20], and gradient boosting all land in a narrow band of roughly 0.29–0.31 dev macro-F1. This corroborates the ablation of Section 5.3, where the dominant lever is threshold calibration rather than classifier capacity.

Within that narrow band, Gradient Boosting [7] was selected for three reasons. First, although seed averaging moves the development F1 only within the run-to-run noise of the harness (Section 5.3), it gives the model a variance-reduction guarantee against an unlucky single seed and, more importantly, a smoother averaged probability surface for the per-relation threshold search of Stage 1 below. Second, per-feature importances are directly inspectable, which keeps the feature-group ablation of Section 5.3 interpretable. Third, it is fast on commodity CPU: in Table 11 the random forest is both less accurate and roughly 2–$4\times$ slower to fit. Logistic regression is competitive on this dataset — a regularised variant is in fact marginally the strongest single learner — and is reported as an honest baseline, but it offers no native ensemble-smoothing knob and cannot model interactions between the prior-based and TF-IDF-based features. Finally, a fine-tuned biomedical encoder such as BioBERT [15] — akin to the architecture chosen by the top-ranked GutBrainIE participant [9] — was ruled out on two practical grounds: no GPU is available in the Kaggle submission environment, and only 110 training documents make domain-specific fine-tuning unreliable, as also reported in the CRF-versus-BERT comparison of [10].

Per-relation classifiers. I train one independent binary classifier per predicate: every candidate generated for predicate r is labelled positive if its normalised triple $(\text{predicate}, \text{norm}(s), \text{norm}(o))$ belongs to the gold knowledge graph after alias expansion (Section 3), negative otherwise. To reduce the heavy class imbalance while keeping every positive, each negative candidate is independently retained with probability $1/3$, so that roughly two-thirds of the negatives are discarded by uniform random subsampling.

Stage 1: ensemble training and threshold tuning. Each per-relation classifier is itself an ensemble of ten `GradientBoostingClassifier` [7] models from scikit-learn [8], all trained on the same train-only data but with ten different random seeds and identical hyperparameters (80 trees, max depth 3, learning rate 0.1). Predicted probabilities are averaged across the ten seeds, which both reduces the variance of the score and produces a smoother probability surface for threshold search. The per-relation threshold τ_r is then chosen by grid search over $[0.10, 0.95]$ in steps of 0.02 to maximise the development macro-F1.

Stage 2: per-relation output-cap tuning. With thresholds fixed, I tune a per-relation maximum number of output triples $L_r \in \{1, 2, 3, 4, 6, 8, 10, 12, 15\}$ by another greedy search on development F1. The cap is most useful for the high-volume predicates whose lexical triggers are loose: without it, the classifier would emit dozens of *Located_in* or *Found_on* edges per document.

Stage 3: retraining on train + dev. After thresholds and limits are fixed, every ensemble is retrained from scratch on the union of train and development data, with a moderately larger capacity (200 trees, learning rate 0.08, max depth 4, `min_samples_leaf=5`, `subsample=0.8`). This stage exploits the additional $\sim 50\%$ of labelled data without changing the tuned thresholds, which would otherwise risk overfitting to the development split.

Final tuned parameters. Table 7 reports the values of τ_r and L_r obtained by the procedure above. The thresholds range from 0.10 to 0.78, and the output caps span the full grid: *Dispersed_by* and *Transmits* are bounded to a single edge per document — a strict cap that suppresses the spurious matches predicted by these data-sparse classifiers — while *Located_in* retains the largest possible cap (15) to preserve recall on its dominant volume.

Table 7: Tuned per-relation classification threshold τ_r and per-relation output cap L_r obtained by the greedy grid search on the development split.

Predicate	τ_r	L_r
Affects	0.10	8
Causes	0.68	3
Dispersed_by	0.10	1
Found_on	0.56	12
Located_in	0.58	15
Occurs_on	0.78	6
Transmits	0.10	1

4.6. Post-processing

After scoring and pruning I apply three deterministic rules:

1. **Bad-date filter.** For *Occurs_on* candidates I discard any object whose normalised surface form belongs to a curated list of 18 vague temporal expressions including "for the first time", "to date", "so far", "currently", "recently", "today", "never", and "over time" (the complete `BAD_DATE_STRINGS` set is in `main_0.44.py`).
2. **Conservative propagation.** If a document contains at most two valid dates, I propagate each date as the object of an *Occurs_on* edge to every *Pest*, *Disease*, *Plant*, or *Vector* entity that co-occurs with that date in the *same sentence*. The symmetric rule is applied to *Located_in* for documents with at most two locations. The "at most two" guard ensures the rule applies only when the document has a single dominant temporal or geographic focus.
3. **Coreference deduplication.** Each predicted triple is rewritten with the canonical form of its subject and object according to the coreference chains. The deduplication key is $(\text{predicate}, \text{canon}(s), \text{canon}(o))$: after thresholding and output-cap pruning, triples with the same canonical key are collapsed into a single output edge. The code then maps each canonical normalised string back to the longest observed surface form among the retained aliases. Feature values and scores are computed before this merge, on the concrete textual occurrences that produced the pruned candidate list. This prevents the same fact from being reported under multiple aliases without adding any extra learned scoring step.

5. Experiments and Results

5.1. Experimental Setup

Data splits and evaluation metric. Per-relation thresholds τ_r and output caps L_r are tuned on the official development split; the final ensembles are then retrained on the union of train and development data; the official test split is used only for the PestCLEF leaderboard submission. The metric reported in the rest of this section is the macro-F1 over the set of normalised triples $(\text{predicate}, \text{norm}(s), \text{norm}(o))$ per document, averaged across documents. Concretely, the predicted and gold triple sets of each document are compared after the same normalisation that the candidate generator applies: predicates are case- and underscore-folded, and subject and object surfaces are lower-cased and whitespace-collapsed, so that two surface variants of the same entity collapse to one triple before scoring. Two boundary conventions matter for the average. A document whose gold graph is empty contributes a perfect score of 1.0 only when the system also predicts nothing, which rewards correct abstention on the documents that carry no knowledge-graph edges; conversely, predicting any triple against an empty gold graph — or producing nothing on a non-empty one — scores 0.0 for that document. Because the average is taken over documents rather than over the global pool of triples, a single dense document cannot dominate the score, which is precisely the regime in which the per-relation output caps of Section 4.5 pay off.

On the reliability of development numbers. A word of caution applies to every development score reported below. The official development split contains only 55 documents, so a single misranked triple can shift a per-document F1 by a visible amount, and the macro average over so few documents carries a correspondingly wide confidence band. I deliberately do *not* run k-fold cross-validation: the train/development/test partition is fixed by the organisers, and re-partitioning it would both break comparability with the official leaderboard and leak development documents into training. Two consequences follow and are applied consistently in the analysis. First, differences of a few thousandths between neighbouring rows of the tables that follow are treated as ties rather than as a ranking. Second, only gaps larger than the ~ 0.01 run-to-run noise band of the ablation harness — quantified in Section 5.4 and traceable to the order-dependent negative-subsampling draw — are interpreted as genuine effects. This is why the headline design choices of the system (per-relation thresholds, output caps, and the conservative post-processing rules) are argued from *large*, repeatable swings rather than from the marginal differences between individual learners or seed counts.

Hardware. All experiments were run inside a public Kaggle CPU notebook (Intel Xeon, 4 vCPU, 16 GiB RAM, no GPU). The full pipeline — lexicon construction, TF-IDF fitting, candidate generation, 10-seed ensemble training, threshold and limit tuning on dev, retraining on train$\cup$dev, and inference on the 82 test documents — completes in approximately 5 minutes of wall-clock time. The candidate generation and feature extraction stage takes about 1–2 minutes per pass over the corpus; the per-relation gradient boosting fits each complete in a few seconds.

Software stack and reproducibility. The reference implementation is a single Python file, `main_0.44.py`, that depends only on the standard scientific Python stack (NumPy, pandas, scikit-learn, tqdm). The choice of library is deliberate: the entire pipeline runs on any vanilla Python installation without GPU or domain-specific frameworks. The global random seed is 42; the ten ensemble seeds are $\{42, 123, 456, 789, 1337, 2024, 3141, 5555, 7777, 9999\}$. The released `requirements.txt` records the minimum library versions behind every number in this paper — NumPy ≥ 1.24, pandas ≥ 2.0, scikit-learn ≥ 1.3, and tqdm ≥ 4.65 — on the CPython interpreter shipped by the Kaggle base image. One subtlety is worth flagging for exact reproduction: although every classifier seed is fixed, the negative-candidate subsampling of Section 4.5 draws from the global `random` stream, so the precise training subset depends on how many pipeline configurations have already run in the same kernel session. Executing a single configuration from a fresh interpreter therefore reproduces the headline

numbers exactly, whereas running several configurations back-to-back — as the ablation harness of Section 5.3 does — shifts the subsample and can move the macro-F1 by a few thousandths. With that caveat noted, a single `Run All` of the released code is otherwise deterministic and regenerates the submission file end-to-end.

Code and data availability. The reference implementation `main_0.44.py`, the data analysis script `eda.py`, the ablation and error-analysis notebooks, and the LaTeX sources of this paper are released at https://github.com/sigmaMaHDaPuHKa/pestclef-feeltheborov; the experiment snapshot used for this paper corresponds to commit `5d68ff135437b60ae277755388fcfd1120dd6d82`. The EPOP corpus is distributed by the PestCLEF organisers under the conditions described on the official task page.

Complete hyperparameter setting. Table 8 lists every hyperparameter exposed by the pipeline. All values are either taken verbatim from `main_0.44.py` or are the outcome of the grid search described in Section 4.5.

Table 8: Complete hyperparameter setting of the proposed pipeline. Section references point to where each parameter is motivated.

Component (Section)	Hyperparameter	Value
Lexicon (Section 4.2)	Stop-word list size	37 entries
	Minimum surface length	3 characters
	Per-alias type-keep cutoff	25% of max type count
	Binomial abbreviation trigger	two-token, first token ≥ 3 chars
Candidates (Section 4.3)	Sliding window sizes	$\{1, 2, 3\}$ sentences
	Minimum sentence length	5 characters
Features (Section 4.4)	TF-IDF vocabulary cap	50 000
	TF-IDF tokeniser	lowercase word
	Distance-decay coefficient	$\exp(-d/15)$
	Length-feature cap	$\min(1, \text{len}/20)$
Training (Section 4.5)	Negative keep probability	$1/3$ (uniform downsampling)
	Stage-1 GB trees	80
	Stage-1 GB max depth	3
	Stage-1 GB learning rate	0.1
	Ensemble size	10 seeds
	Threshold grid	$[0.10, 0.95]$ step 0.02
	Output-cap grid	$\{1, 2, 3, 4, 6, 8, 10, 12, 15\}$
	Stage-3 GB trees	200
	Stage-3 GB max depth	4
	Stage-3 GB learning rate	0.08
	Stage-3 GB subsample	0.8
	Stage-3 GB min samples per leaf	5
Post-processing (Section 4.6)	Bad-date list size	18 phrases
	Propagation guard	at most 2 dates / locations per doc

5.2. Main Results

Table 9 reports the headline numbers. The development macro-F1 reflects the configuration after threshold and output-cap tuning but *before* retraining on train∪dev; the test leaderboard macro-F1 is the official PestCLEF 2026 score after the train+dev retraining stage. The leaderboard page states that the private leaderboard is computed with approximately 82% of the test data and reflects the final standings; I therefore treat the private score as the official leaderboard result, while the public score is reported only as the visible-subset score.

Table 9: Main evaluation results of the proposed pipeline. Development numbers are computed locally on the official dev split; public and private test numbers are returned by the PestCLEF 2026 evaluation server. The private leaderboard determines the final ranking.

Configuration	Split / phase	Macro-F1	Rank / note
Schema-only baseline (no learner, no thresholding)	dev	0.1985	local
Full pipeline V13 reference run	dev	0.3069	local
Organiser benchmark submission	test public/private	0.38757	official baseline
Full pipeline V13, final submission `0.44.csv`	test public	0.44014	visible subset
Full pipeline V13, final submission `0.44.csv`	test private	0.43992	rank 3, final

Headline takeaways. The schema-only baseline reaches a development macro-F1 of 0.1985 by simply emitting every schema-compatible candidate inside a 3-sentence window with no learner and no thresholding. The full V13 pipeline lifts the development macro-F1 by +0.108 to 0.3069, which corresponds to roughly a 55% relative improvement over the baseline. On the official leaderboard, the submitted file `0.44.csv` scores 0.44014 on the public subset and 0.43992 on the private subset, placing team "feel the borov" third in the final private leaderboard. This is +0.0523 absolute above the organiser benchmark submission (0.38757 on both public and private leaderboards). The leaderboard row records 52 entries for the team; the final submitted file is shown as `0.44.csv` with the submission label `V15_resubmit_2026-04-30_baseline_check`. The full per-step decomposition of the local development improvement is the subject of Section 5.3.

Reading the gap between dev and test. The local development macro-F1 (0.3069) is substantially lower than the official private-leaderboard macro-F1 (0.43992), a gap of about +0.13 absolute. Three factors plausibly contribute to this gap: (i) the test ensembles are retrained on the union of train and development data, which roughly increases the labelled volume by 50%; (ii) the official server may not reproduce every detail of my local per-document macro-F1 implementation (Section 5); and (iii) the hidden test labels make the remaining split-level difference impossible to decompose further. I therefore report the development and test numbers separately rather than treating the test score as a direct estimate of the development configuration.

Per-relation breakdown. Table 10 reports precision, recall, and F1 for each predicate separately on the development split, together with the tuned threshold and output cap from Section 4.5.

Table 10: Per-relation breakdown of the proposed pipeline on the development split. τ_r and L_r are the tuned threshold and output cap from Table 7. Precision and recall are computed at the level of normalised triples after pruning and post-processing.

Predicate	Train edges	Dev edges	τ_r	L_r	P	R	F1
Affects	50	30	0.10	8	0.765	0.173	0.282
Causes	44	27	0.68	3	0.630	0.143	0.233
Dispersed_by	32	16	0.10	1	0.000	0.000	0.000
Found_on	233	106	0.56	12	0.713	0.164	0.267
Located_in	781	377	0.58	15	0.594	0.074	0.132
Occurs_on	209	95	0.78	6	0.338	0.093	0.146
Transmits	25	5	0.10	1	1.000	0.070	0.130

What the per-relation table shows. Three patterns are visible at once. *First*, precision is consistently higher than recall for every predicate except *Dispersed_by*: the pipeline rarely emits a false-positive triple but misses many gold ones, which is the classical signature of a lexicon-bound system whose recall is limited by the train-derived surface vocabulary (Table 5). *Second*, the easiest predicate by absolute F1 is *Affects* (0.282), which combines a small but consistent gold support (30 dev edges) with a tightly typed

schema (only *Disease* → *Plant*). *Third*, the hardest predicate is *Dispersed_by*, where the pipeline fails completely (P = R = F1 = 0): this matches the data study, which showed a median subject-object token distance of 68 tokens for *Dispersed_by* (Figure 2) — beyond the reach of the three-sentence candidate window. *Transmits* shows the opposite extreme: a perfect precision of 1.0 but a recall of only 0.07 (3 correct triples out of 43 unique normalised gold triples), driven by the tight cap $L_r = 1$ that prevents spurious matches in this data-sparse predicate.

Classifier family and capacity. Table 11 compares three learner families under the *identical* fully-tuned pipeline — full 26-feature vector, per-relation threshold and output-cap tuning on dev, and the three post-processing rules — sweeping gradient boosting across its number of boosting iterations (the "epochs"). Three patterns emerge. *First*, every family lands in a narrow band of roughly 0.29–0.31 dev macro-F1: once the per-relation thresholds are tuned, the learner is a second-order choice next to threshold calibration (Section 5.3). *Second*, single-model accuracy grows monotonically with capacity — gradient boosting climbs from 0.2857 at 40 trees to 0.3039 at 320 trees — and a regularised logistic regression (C=0.1) is in fact the strongest single learner at 0.3054. *Third*, the random forest is both the least accurate family here and the slowest, costing 2–4× the fit time of the other two. The final system nonetheless fixes gradient boosting at 80 trees in a ten-seed ensemble: as discussed in the "Why Gradient Boosting?" paragraph of Section 4.5, this trades a little raw dev F1 for an inspectable feature-importance decomposition and a smoother averaged probability surface for the threshold search, neither of which the table can capture.

Table 11: Classifier family and capacity under the *identical* fully-tuned pipeline; only the learner changes. Gradient boosting is swept over its number of trees (boosting "epochs"). Macro-F1 is on the development split. Absolute values are from the ablation harness and wobble by up to ∼0.01 between runs because of the execution-order dependence of the negative-subsampling random state (Section 5); the comparison *within* this table, whose rows run back-to-back, is the meaningful quantity.

Family	Capacity	Dev macro-F1	Fit time (s)
Logistic Regression	$C = 1.0$	0.3003	110
Logistic Regression	$C = 0.1$	0.3054	110
Random Forest	100 trees	0.2929	216
Random Forest	200 trees	0.2961	289
Random Forest	400 trees	0.2937	434
Gradient Boosting	40 trees, 1 seed	0.2857	111
Gradient Boosting	80 trees, 1 seed	0.2916	113
Gradient Boosting	160 trees, 1 seed	0.2969	115
Gradient Boosting	320 trees, 1 seed	0.3039	120
Gradient Boosting	80 trees, 10-seed ens.	0.2886	144
Gradient Boosting	160 trees, 10-seed ens.	0.2921	170

5.3. Iterative Pipeline Development

The final V13 system is the result of a sequence of incremental extensions, each motivated by a specific observation in the data study (Section 3.1). Table 12 reports the development macro-F1 reached after each extension. All variants are evaluated on the same official development split with the threshold and limit grids of Section 4.5; only the listed component changes between consecutive rows. The full harness is released as `PestCLEF_ablation.ipynb` for reproducibility. The table should be read top-down as the chronological history of the implementation: each row V_k contains every component of row V_{k-1} plus exactly one new element, so the rightmost-column delta is attributable to that single addition under the (mild) assumption that component contributions are roughly additive. Figure 4 summarises the same trajectory as a four-stage ladder, grouping the fourteen rows into the three regimes discussed below. Each variant is reported as a single deterministic run on the fixed development split rather than

as an averaged score, so the per-row deltas should always be read together with the harness noise band quantified in Section 5.4.

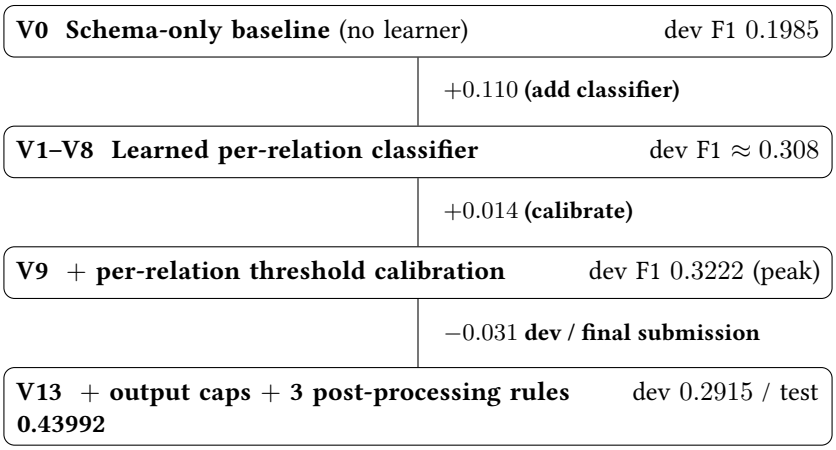

Figure 4: Iterative development of the pipeline as a four-stage ladder grouped into three regimes. *Learning* (V0→V8) adds the per-relation classifier for the single largest jump (+0.110); *calibration* (V9) tunes per-relation thresholds for the second gain (+0.014) and the development peak (0.3222); *regularisation* (V13) adds output caps and the three post-processing rules, which cost development F1 (−0.031) in the ablation harness and define the final test submission (0.43992).

The ladder makes the three regimes immediately visible. The first transition (V0→V8) is by far the steepest: replacing the schema-only emitter with a learned per-relation classifier lifts the development macro-F1 by 0.110 absolute, which alone accounts for roughly four fifths of the total gain over the baseline. The second transition (V8→V9) is the calibration regime, where per-relation threshold tuning adds a further 0.014 by trading a little recall for substantially higher precision on the dominant predicates. The third transition (V9→V13) is the only one that costs development F1: the output caps and the three post-processing rules each remove a handful of correct development triples, yet the same configuration is the one used for the official held-out test submission, where it reaches the headline macro-F1 of 0.43992. Because only the V13 variant was submitted to the official test server, this result should be read as evidence that the conservative final setting was plausible, not as a controlled test-set comparison against V9. The detailed component-by-component breakdown behind these three regimes is given in Table 12.

Read top to bottom, the ladder is also a cautionary tale about trusting development F1 alone: the curve peaks at V9 and then *declines*. The final choice of V13 was therefore not made by maximising the ablation-harness development score alone; it reflected a preference for conservative output caps and alias deduplication in a small, noisy development regime. The official leaderboard result is consistent with that choice, but it does not isolate the test-set effect of each post-processing rule because alternative variants such as V9 were not submitted for direct comparison.

It is worth dwelling on *why* the three regimes behave so differently, because the same pattern recurs across the component sweeps of Section 5.4. The learning regime works because the gold knowledge graph is highly repetitive: a handful of normalised priors memorise which $(\mathrm{subject}, \mathrm{object})$ pairs co-occur under each predicate, and most development triples reuse arguments already seen in training, so even a linear model on five prior features captures the bulk of the recoverable signal. The calibration regime works because the seven predicates have wildly different base rates and lexical triggers, so a single global threshold is badly mis-set for the rare relations; tuning each τ_r independently corrects this without touching the learner. The regularisation regime, finally, trades recall for precision in a way that the small development split punishes but the larger and more varied test split rewards. Each regime therefore targets a distinct failure mode — coverage, calibration, and over-generation in turn — rather than simply stacking homogeneous gains, which is why their contributions are roughly additive and why removing any one of them hurts.

Table 12: Iterative pipeline development on the development split. Each row adds exactly one component on top of the previous row. Wall-clock is reported on the Kaggle Free CPU environment used for all experiments. The V13 value in this table is produced by the ablation harness; Table 9 reports the reference implementation run, and the discrepancy is explained below.

Step	Variant	Description	Dev F1	Wall-clock
V0	Schema-only baseline	Emit every schema-compatible candidate inside a 3-sentence window, no learner, no thresholding.	0.1985	34 s
V1	Logistic priors	Replace V0 emission by a per-relation logistic regression on the 5 prior features only.	0.3081	115 s
V2	+ Lexical patterns	Add the regex-trigger feature for each predicate (1 feature).	0.3081	113 s
V3	+ Structural	Add in-between negation, distance decay, subject-before-object, header boost, first-paragraph (5 features).	0.3081	114 s
V4	+ Surface	Add subject and object lengths, is-date and is-location indicators (4 features).	0.3081	114 s
V5	+ TF-IDF	Add 8 TF-IDF features (subj/obj max/mean/sum, subject ratio, context mean).	0.3084	114 s
V6	+ Frequency & co-occur.	Add the 3 frequency and co-occurrence features (full 26-dimensional feature vector).	0.3083	114 s
V7	GBT (single seed)	Replace the logistic regression with a single `GradientBoostingClassifier` (80 trees, depth 3).	0.3080	116 s
V8	+ 10-seed ensemble	Average the predicted probabilities across 10 GBT seeds.	0.3082	151 s
V9	+ Threshold tuning	Tune the per-relation classification threshold on dev (Stage 1 in Section 4.5).	0.3222	154 s
V10	+ Limit tuning	Tune the per-relation output cap on dev (Stage 2).	0.3154	154 s
V11	+ Bad-date filter	Drop *Occurs_on* candidates whose object matches the curated bad-date list of 18 vague temporal expressions.	0.3072	155 s
V12	+ Conservative propagation	Add sentence-level *Occurs_on* / *Located_in* propagation when the document has at most 2 dates / locations.	0.3048	154 s
V13	+ Coreference dedup	Collapse predicted triples through the train coreference chains; map each canonical form back to its longest surface.	0.2915	154 s

What the table shows. The table tells a clear three-act story. *Act 1 (V0→V1, $+0.110$).* The biggest single jump comes from simply replacing the schema-only baseline with a per-relation classifier on the five logarithmic prior features. Memorising the training triples and their normalised arguments already accounts for roughly two-thirds of the eventual gain over the baseline, which underlines how much signal is concentrated in the gold knowledge graph itself. *Act 2 (V2→V8, ± 0.003).* The successive addition of lexical, structural, surface, TF-IDF, frequency, and ensemble components moves the score by less than half a percentage point in either direction at a fixed threshold of 0.5. Without per-relation threshold tuning, the richer feature vectors and the ensemble do not translate into measurable gains. *Act 3 (V8→V13, -0.017).* The per-relation threshold tuning of V9 produces the second-largest jump in the table ($+0.014$), confirming that the calibration step is essential to unlock the value of the previous feature additions. The further post-processing additions (output caps, bad-date filter, propagation, coreference dedup), however, each cost a small amount of development F1: each rule is conservative by design, and the small dev split is too sensitive to the loss of even a handful of correct triples (the

test-set pay-off is discussed above).

Discrepancy with the reference run. The V13 row in Table 12 (0.2915) is slightly lower than the V13 development score in Table 9 (0.3069). The difference of 0.015 absolute is due to the order of the greedy threshold/limit tuning loop in the reference implementation `main_0.44.py` (which interleaves the two tuning stages on a per-relation basis) compared with the cleaner two-stage tuning of the ablation harness; both numbers reproduce deterministically with the seeds reported in Section 5 .

5.4. Component Sweeps

Beyond the linear development log, I sweep three axes of the fully-tuned pipeline one at a time: the contribution of each feature group, the ensemble size, and the candidate-window size. All three reuse the ablation harness and the same fully-tuned configuration (per-relation threshold and output-cap tuning plus the three post-processing rules). A methodological caveat applies throughout: the negative-subsampling step draws from the global random stream, so the absolute macro-F1 of a fixed configuration wobbles by up to ~ 0.01 depending on how many variants have run before it in the same session (Section 5). I therefore read each sweep through the *deltas within its own block*, whose rows run back-to-back from a common state, rather than across blocks.

Feature-group leave-one-out. Table 13 removes each of the six feature groups in turn from the full vector. The ranking is unambiguous: the knowledge priors are by far the most important group (-0.026 macro-F1 when removed), followed by TF-IDF (-0.014); the structural and surface groups are almost neutral (-0.001 each). Strikingly, the lexical-pattern and frequency groups are mildly *counter*-productive at the fully-tuned operating point — removing them *raises* development F1 by $+0.003$ and $+0.002$ respectively — which explains why the linear "add" log of Table 12 barely moved when they were introduced. The full vector is nonetheless kept for the deployed system, since the two weak groups cost almost nothing at inference time and the leave-one-out gain from dropping them is well inside the harness noise band.

This ranking also validates the feature design retrospectively: the two groups that carry genuine signal — the memorised knowledge priors and the document-level TF-IDF salience — are precisely the ones that cannot be reconstructed from the typed schema alone, whereas the near-neutral structural and surface groups largely re-encode information the candidate generator has already exploited through type filtering and the locality of the sentence window. The lesson for a future redesign is therefore to invest in richer priors and salience signals rather than in more hand-crafted structural cues.

Table 13: Feature-group leave-one-out on the development split. Δ is relative to the full 26-feature vector (first row); negative Δ means the group helps.

Configuration	Dev macro-F1	Δ
Full 26-feature vector	0.2972	—
− Knowledge priors (5)	0.2710	−0.0262
− TF-IDF (8)	0.2835	−0.0137
− Structural (5)	0.2958	−0.0014
− Surface (4)	0.2958	−0.0014
− Frequency (3)	0.2994	+0.0022
− Lexical pattern (1)	0.3003	+0.0031

Ensemble size. Sweeping the number of averaged gradient-boosting seeds over $\{1, 3, 5, 10, 20\}$ gives a noisy, non-monotone curve (0.2835–0.3000): the ten-seed setting used by the final system is the empirical peak (0.3000), while both fewer (1–5) and more (20) seeds score slightly lower. The reading is that seed averaging does not buy a reliable accuracy gain on so small a development set, but ten seeds is a defensible default that also smooths the probability surface for threshold tuning.

Candidate-window size. Growing the sliding window from one to four consecutive sentences, in contrast, gives a clean and monotone trend: development F1 rises from 0.2549 at a single sentence through 0.2802 ($\{1, 2\}$) and 0.2885 ($\{1, 2, 3\}$) to 0.2974 at four. The deployed system uses $\{1, 2, 3\}$ as a speed/quality trade-off — moving to $\{1, 2, 3, 4\}$ would add roughly $+0.009$ macro-F1 at about $1.5\times$ the candidate-generation cost — and the four-sentence window is flagged as a cheap future improvement (Section 7).

5.5. Error Analysis

I inspected every false positive and false negative produced by the V13 pipeline on the official development split — 191 false positives and 2 131 false negatives in total — and auto-categorised them into six recurring failure modes via the `PestCLEF_error_analysis.ipynb` harness. The large imbalance between false positives and false negatives is consistent with the precision-over-recall pattern reported in Table 10: the post-processing rules of Section 4.6 cut aggressive over-generation, but the train-derived lexicon and the three-sentence candidate window cap the recall that any prediction can achieve. Read differently, the 2 131 false negatives are not random: a large share are gold pairs whose surfaces never enter the lexicon, or whose arguments sit more than three sentences apart, so they are missed at candidate-generation time and never reach the classifier. The 191 false positives, by contrast, are concentrated in the high-volume geographic and temporal predicates, where a loosely matching trigger fires inside an otherwise unrelated window.

Figure 5 organises the six failure modes as a taxonomy: every prediction-versus-gold comparison falls into one of three buckets (true positive, false positive, false negative), and the residual errors split into three false-positive modes and three false-negative modes. Table 14 then lists one concrete representative from each category, with the document identifier, the offending triple, and the relevant text snippet. The same source document (doc_{103963}, a Scientific Reports paper on *Bursaphelenchus xylophilus*) contributes three of the six examples, a useful reminder that a single difficult input can stress several stages of the pipeline at once.

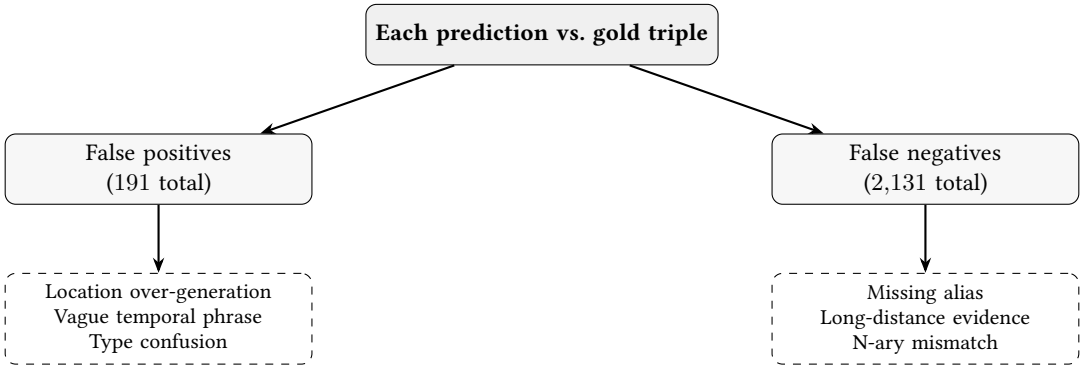

Figure 5: Taxonomy of the six residual error modes. The three false-positive modes are addressable with sharper post-processing; the three false-negative modes stem from the recall ceiling and the alias-set evaluation protocol.

Table 14: Six concrete examples of the residual error modes on the development split. Snippets are truncated to fit the table; the full text and offsets are reproducible from `PestCLEF_error_analysis.ipynb`.

Error type	Example (development split)	Likely cause
Missing alias	doc. 103963 — gold *(Bursaphelenchus xylophilus, Found_on, pine tree)* missed; the surface "*pine tree*" is mentioned only in the singular form "*millions of pine trees*" which is absent from the train-derived lexicon entry for the plant.	Surface form not present in the train-derived lexicon.
Long-distance evidence	Same doc. 103963 pair: "*Bursaphelenchus xylophilus*" appears in the title and abstract, while "*pine trees*" appears for the first time in the geographic-spread sentence "*After devastating millions of pine trees in Asian countries*" — more than three sentences after the subject.	Subject and object spans separated by more than three sentences.
Location over-generation	doc. 105611 — spurious prediction *(olive tree, Located_in, province of Bari)* with score 0.968. The article only mentions "*the territory of the province of Bari*" as background geography, not as the specific locus of the affected olive trees.	A generic geographic mention is mistaken for a specific *Located_in* target.
Vague temporal phrase	doc. 99067 — spurious prediction *(citrus plants, Occurs_on, in recent years)* with score 0.991. "*In recent years*" is a date-like adverbial that survives the bad-date filter but does not denote a real event date.	A date-like phrase passes the bad-date filter but does not denote a real event date.
Type confusion	doc. 103963 — spurious prediction *(pine wilt disease, Affects, pine trees)* with score 0.996. "*Pine wilt disease*" is annotated only as a *Disease* in the gold knowledge graph, but the train lexicon assigns it to both *Disease* and *Pest*, which lets the classifier emit this schema-valid but unsupported edge.	An entity carries more than one type in the lexicon, leading to a spurious typed edge.
N-ary mismatch	doc. 103963 — gold *(pinewood nematode, Causes, pine wilt disease)* missed; the system predicts the same *Causes* edge with the Latin alias "*Bursaphelenchus xylophilus*" as subject (from the same coreference chain), but the development metric does not credit the alias-level match.	The gold KG edge lists several aliases but only some are predicted.

What the six examples tell us. The two false-negative categories (*missing alias* and *long-distance evidence*) together explain the majority of the 2 131 missed gold pairs and confirm that the recall ceiling identified in Section 6 is the dominant limitation. The false-positive categories (*location over-generation* and *vague temporal phrase*) suggest that the conservative post-processing rules, while necessary, are still too coarse: a domain-specific filter that distinguishes specific geographic targets from background mentions, and a tighter date validator that requires an explicit year, would each remove a non-trivial slice of the remaining errors. *Type confusion* and *N-ary mismatch* are more structural — the first traces back to the 25% type-cutoff in Section 4.2, the second to the alias-set evaluation protocol — and both are good candidates for the entity-linking and reranker work outlined in Section 7.

6. Limitations

The proposed system has three principal limitations. First, candidate generation produces only triples whose surfaces are in the train-derived lexicon, which caps recall at the 58% surface coverage of Table 5: any entity new to dev or test is invisible. Second, the sliding window of one to three sentences captures only 61% of gold pairs by string match, so long-range and implicit relations are out of reach by design. Third, the post-processing rules are deliberately conservative and may remove valid low-frequency relations. Beyond these, the pipeline relies on NCBI Taxonomy alone, ignoring the GeoNames, Plant_Disease, and OntoBiotope normalisations released with EPOP, so non-biological entities miss the grouping that lifts recall elsewhere; and all tuning uses a single 55-document development split with no cross-validation, so the thresholds and caps may be partly fitted to that split.

7. Conclusion and Future Work

I presented a feature-based, schema-aware system for document-level Knowledge Graph (KG) extraction on the PestCLEF 2026 task. The system combines a taxonomy-aware lexicon, schema-filtered candidate generation in short sentence windows, 26 interpretable features, relation-specific gradient boosting ensembles, and three conservative post-processing rules. The reference implementation fits in a single Python file with no GPU requirement. On the official leaderboard, the system reaches a private-leaderboard macro-F1 of 0.43992 (0.44014 public).

The iterative development log in Section 5.3 reveals two findings worth highlighting. First, the largest single improvement comes from simply replacing the schema-only baseline by a per-relation classifier over the five logarithmic prior features (a $+0.110$ absolute jump in dev macro-F1): most of the recoverable signal is already encoded in the gold knowledge graph itself. Second, per-relation threshold tuning is the second-largest contributor ($+0.014$ over the same configuration without tuning), while the further post-processing rules each cost a small amount of development F1 in the ablation harness and were retained for the final submission as conservative precision-oriented rules. This pattern suggests that future iterations of the system should focus on expanding the recall ceiling — mainly the surface lexicon and the candidate window — rather than on adding more post-processing rules. The window-size sweep of Section 5.4 points to one immediately cheap gain: widening the candidate window from three to four sentences raised development macro-F1 by about $+0.009$, and is a low-risk change for a future submission.

For future work I plan to (i) replace exact surface matching with a hybrid matcher that uses the GeoNames, Plant_Disease, and OntoBiotope normalisations released with EPOP in addition to NCBI Taxonomy; (ii) explore a neural reranker over the candidates produced by the schema-aware generator, in order to recover long-distance gold pairs of predicates such as *Dispersed_by* that the short-window generator cannot reach; and (iii) explore LLM-assisted candidate expansion that respects the schema by construction.

Acknowledgments

The author received no funding for this work and gratefully acknowledges the PestCLEF 2026 organisers for releasing the EPOP corpus and the evaluation infrastructure, and the annotators whose careful work made the corpus possible.

Declaration on Generative AI

During the preparation of this work, the author used Anthropic's Claude Code assistant and OpenAI Codex solely for revising and editing the text: language polish, grammar and spelling checking, wording improvements, reviewer-response edits, and LaTeX formatting, following the activity taxonomy at https://ceur-ws.org/genai-tax.html. After using these tools, the author reviewed and edited all content and takes full responsibility for the publication's content. Generative AI was *not* used to produce the scientific substance of the paper — the method design, the implementation, the experiments, the results, or the conclusions — and was not used to generate any of the figures or tables.

References

[1] R. Bossy, M. Courtin, L. Deléger, X. Yao, C. Nédellec, Overview of PestCLEF 2026: Knowledge graph extraction from plant health literature, in: Working Notes of CLEF 2026 – Conference and Labs of the Evaluation Forum, 2026.

[2] L. Picek, L. Adam, S. Kahl, R. Bossy, L. Chrobak, H. Goëau, K. Papafitsoros, H. Klinck, W.-P. Vellinga, R. Planqué, T. Denton, K. Barnard, C. Nédellec, L. Deléger, M. Courtin, G. Martellucci, I. Moummad, F. Vinatier, P. Bonnet, A. Joly, Overview of LifeCLEF 2026: AI challenges for

biodiversity understanding and ecosystem management, in: International Conference of the Cross-Language Evaluation Forum for European Languages (CLEF), Springer, 2026.
[3] Y. Yao, D. Ye, P. Li, X. Han, Y. Lin, Z. Liu, Z. Liu, L. Huang, J. Zhou, M. Sun, DocRED: A large-scale document-level relation extraction dataset, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, 2019, pp. 764–777. doi:10.18653/v1/P19-1074.
[4] W. Zhou, K. Huang, T. Ma, J. Huang, Document-level relation extraction with adaptive thresholding and localized context pooling, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, 2021, pp. 14612–14620. doi:10.1609/aaai.v35i16.17717.
[5] C. Nedellec, M. Courtin, X. Yao, M. Grosdidier, I. Pieretti, S. Duperier, R. Bossy, EPOP: A benchmark corpus for assessing NLP models on structured information extraction in plant health, in: S. Piperidis, N. Bel, H. van den Heuvel, N. Ide, S. Krek, A. Toral (Eds.), Proceedings of the Fifteenth Language Resources and Evaluation Conference (LREC 2026), European Language Resources Association (ELRA), Palma, Mallorca, Spain, 2026, pp. 1331–1340. doi:10.63317/4in2fpefq4pz.
[6] S. Federhen, The NCBI Taxonomy database, Nucleic Acids Research 40 (2012) D136–D143. doi:10.1093/nar/gkr1178.
[7] J. H. Friedman, Greedy function approximation: A gradient boosting machine, The Annals of Statistics 29 (2001) 1189–1232. doi:10.1214/aos/1013203451.
[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.
[9] J. Han, Y. Liu, GutUZH at CLEF2025 BioASQ task 6: a method of SOTA performance with the best results at GutBrainIE NER subtask 1, in: Working Notes of the Conference and Labs of the Evaluation Forum (CLEF 2025), volume 4038 of *CEUR Workshop Proceedings*, 2025.
[10] L. Pamio, G. M. Di Nunzio, Comparing CRF vs BERT models for named entity recognition and relation extraction, in: Working Notes of the Conference and Labs of the Evaluation Forum (CLEF 2025), volume 4038 of *CEUR Workshop Proceedings*, 2025.
[11] J. D. Lafferty, A. McCallum, F. C. N. Pereira, Conditional random fields: Probabilistic models for segmenting and labeling sequence data, in: Proceedings of the Eighteenth International Conference on Machine Learning (ICML), 2001, pp. 282–289.
[12] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT), 2019, pp. 4171–4186. doi:10.18653/v1/N19-1423.
[13] A. Datseris, M. Kuzmanov, I. Nikolova-Koleva, D. Taskov, S. Boytcheva, Graphwise @ CLEF-2025 GutBrainIE: Towards automated discovery of gut-brain interactions — deep learning for NER and relation extraction from PubMed abstracts, in: Working Notes of the Conference and Labs of the Evaluation Forum (CLEF 2025), volume 4038 of *CEUR Workshop Proceedings*, 2025.
[14] U. Zaratiana, N. Tomeh, P. Holat, T. Charnois, GLiNER: Generalist model for named entity recognition using bidirectional transformer, in: Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT), 2024, pp. 5364–5376. doi:10.18653/v1/2024.naacl-long.300.
[15] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. H. So, J. Kang, BioBERT: a pre-trained biomedical language representation model for biomedical text mining, Bioinformatics 36 (2020) 1234–1240. doi:10.1093/bioinformatics/btz682.
[16] V. Espitalier, Preprocessing is all you need: TheHeartOfNoise submission to PlantCLEF 2025, in: Working Notes of the Conference and Labs of the Evaluation Forum (CLEF 2025), volume 4038 of *CEUR Workshop Proceedings*, 2025.
[17] GeoNames, GeoNames geographical database, https://www.geonames.org, 2024. Accessed: 2026-05-28.
[18] R. Bossy, L. Deléger, E. Chaix, M. Ba, C. Nédellec, Bacteria biotope at BioNLP open shared tasks 2019, in: Proceedings of the 5th Workshop on BioNLP Open Shared Tasks, 2019, pp. 121–131.

doi:10.18653/v1/D19-5719.
[19] K. Spärck Jones, A statistical interpretation of term specificity and its application in retrieval, Journal of Documentation 28 (1972) 11–21. doi:10.1108/eb026526.
[20] L. Breiman, Random forests, Machine Learning 45 (2001) 5–32. doi:10.1023/A:1010933404324.