RADBOUD UNIVERSITY

Faculty of Science

and

ING Bank B.V

Dense Retrievers for Entity Ranking

Master's Thesis

Deval Patel (s1148414)

Supervised by:
Prof. Dr. Faegheh Hasibi, Radboud University
Simon Brugman, ING Bank

Abstract

Entity linking matches a mention to its corresponding record in a knowledge base. When the knowledge base holds millions of records, systems narrow the space with a fast candidate-generation stage before a slower disambiguation stage decides the match. This thesis studies candidate generation, cast as entity ranking: retrieving an ordered shortlist of entities for a mention. This thesis studies entity ranking in a low-context setting: matching short company-name mentions, with no surrounding text, against a large client registry. We ask whether dense retrievers can match or complement the strong lexical TF-IDF baseline. We compare TF-IDF against multilingual E5 dense retrievers, both zero-shot and after task-specific fine-tuning with LoRA, on two evaluation distributions, a categorised benchmark we construct from the public GLEIF registry and a set of real transaction counterparty names, analysing performance by category of name variation.

We find that dense retrieval does not replace the lexical baseline: no dense model, zero-shot or fine-tuned, outperforms TF-IDF overall, and dense retrieval's advantage is confined to the hard categories, partial matches and pairs with little lexical overlap, while TF-IDF dominates the exact and legal-form cases that make up most of the data. Fusing a dense retriever specialised on the hard categories with TF-IDF through reciprocal rank fusion, however, significantly improves recall on both distributions, with the gains concentrated on exactly those categories. The productive role of dense retrieval in this setting is therefore not as a replacement for lexical retrieval, but as a targeted complement to it.

Contents

Chapter 1

Introduction

Entity linking addresses the problem of recognizing mentions of entities in text and associating them with their corresponding entries in a knowledge base [1]. It is a core component in applications such as search engines, question answering systems, and enterprise data integration pipelines, all of which must operate over databases of millions of records [2, 3, 4, 5]. This task is computationally challenging due to its quadratic complexity: naively matching n records against m records requires evaluating $n \times m$ candidate pairs, making exhaustive comparison on a single machine infeasible for large-scale datasets [6]. Candidate generation is one of the stages in the entity linking pipeline. In our work it takes the form of entity ranking: given an entity *mention* (a company name to be linked), which we refer to as the *query* when describing the retrieval task, and a large candidate registry, the goal is to return an ordered list of the most likely matches, which is subsequently passed as input to a model for entity disambiguation [1]. Typically the ranking stage employs a fast, approximate model to prune the search space to a manageable shortlist, while the disambiguation stage can afford a slower, more precise model that performs disambiguation over only a handful of candidates [7].

At the Wholesale Banking department of ING Bank, entity linking underpins several operational tasks. These include matching external company name lists, obtained from news organisations, against the client registry when Legal Entity Identifier codes (LEI code) are absent or only partially available, as well as linking names from transaction data to known wholesale clients. ING employs the "Entity Matching Model", developed internally, for this purpose [1]. It relies on a TF-IDF model for entity ranking. The sparsity of TF-IDF representations makes this approach scalable. However, despite its efficiency, the TF-IDF model may fail to retrieve matches in certain cases, for instance, when company names appear in abbreviated form: *Foobar Industries B.V.* $\rightarrow$ *F.I. B.V.*

One way to tackle this challenge is to replace the TF-IDF model with a dense retrieval model [8, 9, 10]. Unlike sparse representations, dense models encode semantic similarity and can in principle match abbreviations, acronyms, and morphological variants to their full-form counterparts [8, 9, 10, 11]. However, their effectiveness relies on the availability of rich contextual signals, an assumption that rarely holds in banking and data integration

[1] EntityMatchingModel

settings, where mentions often consist of nothing more than a bare company name, and candidate entities are represented only by their names and a small set of structured attributes. The absence of context limits the ability of these models to match names with lexically different or superficially similar names without targeted adaptation to the task.

In this work we investigate whether dense retrieval models can match or complement the TF-IDF baseline for low-context entity ranking. We evaluate pretrained dense retrievers against the baseline, examine where each approach succeeds and fails across categories of name variation, and test whether a task-fine-tuned dense retriever, fused with the baseline, improves retrieval.

This thesis is organised as follows. Chapter 2 reviews the relevant work on entity ranking, lexical and dense retrieval, and the adaptation of pretrained encoders. Chapter 3 formalises the task and describes the datasets, models, and evaluation protocol, and Chapter 4 details the data construction and training procedure. Chapter 5 reports and analyses the results, and Chapter 6 concludes with the limitations of the study and directions for future work.

1.1 Research Questions

In this thesis we investigate the following research questions:

RQ1 How do zero-shot dense retrievers compare to a `TF-IDF` baseline for low-context entity ranking?

RQ2 To what extent does task-specific fine-tuning improve a dense retriever's entity-ranking performance?

RQ3 Which categories of name variation does dense retrieval help on, relative to lexical retrieval?

RQ4 How does fusing a category-specialised dense retriever with the TF-IDF baseline affect retrieval, and where do the gains come from?

1.2 Contributions

To address the research questions stated above, we treat entity ranking as a bi-encoder retrieval task: mentions and candidate clients are independently encoded into a shared d-dimensional embedding space, and similarity is measured using the inner product. We compare a lexical baseline based on TF-IDF with character n-grams against a multilingual dense bi-encoder based on the E5 family of pretrained text embedding models, and additionally investigate whether parameter-efficient fine-tuning of the dense encoder can

yield gains that complement rather than replace the `tfidf` model. The fine-tuning procedure uses a contrastive InfoNCE loss with hard negatives mined from the dense model itself and uses Low-Rank Adaptation (LoRA), thereby keeping the number of trainable parameters small.

Our main findings are as follows. Zero-shot dense retrievers do not outperform the `tfidf` baseline; they reach parity at best on recall (RQ1). Task-specific fine-tuning improves a dense retriever over its zero-shot starting point but not past the baseline (RQ2). The categories on which dense retrieval helps are the low-overlap ones, partial matches and no-overlap pairs, while the baseline keeps its advantage on the high-overlap categories (RQ3). Fusing a category-specialised dense retriever with the baseline through reciprocal rank fusion significantly improves recall, with the gains concentrated on exactly those hard categories (RQ4).

The specific contributions of this thesis are as follows:

- **A categorised entity-ranking dataset for the low-context setting.** We construct a labelled (mention, client) dataset by joining the public GLEIF legal-entity registry with ING's internal registry of corporate clients and other relations, annotating each pair with a five-way scheme that captures the type of surface variation between mention and matched client, and partitioning it with a tree-disjoint protocol that prevents both mention- and client-level leakage across splits. The construction is fully specified and reproducible from the public registry, though the dataset itself cannot be released because the client-registry side is proprietary. We additionally evaluate on a separate distribution of counterparty names drawn from real ING transaction data.

- **A systematic comparison of dense retrievers against the lexical baseline.** We evaluate multilingual E5 (base and large), zero-shot and after task-specific fine-tuning, against the TF-IDF baseline, both in aggregate and per category. This establishes how dense retrieval compares to the lexical baseline zero-shot (RQ1), how task-specific fine-tuning changes this (RQ2), and the categories of name variation on which dense retrieval helps relative to lexical retrieval (RQ3).

- **A dense-lexical ensemble that complements TF-IDF.** We show that fusing the TF-IDF baseline with a dense retriever fine-tuned specifically on the hard categories, via reciprocal rank fusion, significantly improves recall over the baseline on both distributions, with the gains concentrated on the partial-match and no-overlap categories where lexical similarity is weakest (RQ4).

Chapter 2

Background and Related Work

In this Chapter we review background and related work on which this thesis is built. Section 2.1 sets the entity-ranking problem within the broader literature on record linkage, entity matching, and entity linking, and identifies the low-resource, short-string setting this thesis addresses. Section 2.2 covers lexical retrieval, which provides the baseline against which our methods are measured. Section 2.3 covers dense retrieval and the pretrained text-embedding models we build on, and Section 2.4 reviews how such models are adapted to a task through fine-tuning.

2.1 Entity Ranking

Identifying which real-world entity is referred to by a textual mention is a long-standing problem that has been studied across statistics, databases, information retrieval, and natural language processing under names that overlap but do not fully coincide: record linkage in statistics and databases, entity matching and entity resolution in data integration, and entity linking in natural language processing. Each of these terms refers to the full pipeline of resolving a mention to an entity, which is roughly decomposed into three stages: detecting the mention to be resolved (mention detection); narrowing the registry of possible entities down to a small ordered candidate set (variously named as blocking, candidate generation, or entity ranking); and selecting the correct entity from that candidate set (disambiguation) [1, 7]. This thesis focuses on the middle stage and refers to it as entity ranking: given a mention and a registry of candidate entities, return an ordered list of the most likely matches. When the task is expressed as a retrieval problem, we refer to the input mention as a query. We assume the mention is given upstream (no mention-detection step) and treat any downstream disambiguation or pairwise re-ranking as out of scope of the present work; the contribution is on the ranking step in isolation.

Classical work in record linkage traces back to the probabilistic framework of Fellegi and Sunter [12], which formalised the record-matching problem as a binary classification on candidate pairs using attribute-level agreement vectors. For decades the dominant research thread focused on how to compute similarity between corresponding string-valued attributes [13] and on how to combine such per-attribute scores into a final match decision

through hand-engineered rules or supervised classifiers [14, 15]. The classical approach has two well-known limitations that become acute when the data consists of short, free-text strings rather than structured records: string distances cannot capture semantic similarity that is invisible at the surface (*e.g.* "IBM" versus "International Business Machines"), and the pairwise scoring model does not scale to large candidate registries where the target entity must be retrieved from hundreds of thousands to millions of possibilities rather than confirmed against a handful.

The introduction of distributed representations to entity linking addressed the first of these limitations. Mudgal et al.'s DeepMatcher [16] showed that learned word embeddings combined with simple neural similarity architectures could outperform string similarity on benchmark entity-linking tasks. The pretrained language model wave reset the state of the art again: Ditto [17] cast entity linking as a sequence-pair classification task on top of BERT and reported substantial improvements over the embedding-based predecessors, while Brunner and Stockinger [18] examined a range of transformer architectures for the same task. Subsequent work has probed how transformers achieve these gains, Paganelli et al. [19] show that pretrained models retain substantial entity-linking capability even before fine-tuning, particularly when the surface forms are similar, suggesting that much of the gain over classical methods comes from the pretrained semantic representation rather than from any matching-specific architectural innovation. These results are encouraging but they are mostly reported in a cross-encoder formulation, in which the query and the candidate are concatenated and encoded jointly to produce a single match score per pair. The cross-encoder architecture is expressive, the representation of the candidate is computed in the context of the query, allowing token-level interactions, but this same property makes it computationally incompatible with retrieving from a large candidate registry: at deployment time the system would need one forward pass through the transformer for every (query, candidate) pair, which is infeasible at registry sizes beyond a few hundred. Cross-encoder entity linking is therefore typically evaluated on pre-blocked benchmarks, where candidate generation has happened elsewhere and the model is only asked to score a small set of plausible pairs.

The shift toward bi-encoder dense retrieval addressed the second limitation. Wu et al.'s BLINK system [20] was a demonstration that entity linking could be decomposed into two stages, a bi-encoder retrieving a small set of candidates from a large registry, followed by a cross-encoder re-ranking the candidates, and that the bi-encoder stage alone, despite its independence assumption between mention and candidate, was strong enough to retrieve the correct entity from nearly six million candidates in milliseconds. This is the same bi-encoder architecture later adopted by general-purpose dense retrievers such as DPR [8], and the idea is clear: an entity-linking bi-encoder can simply be a dense retriever whose corpus happens to be a registry of entities. The two-stage decomposition of BLINK maps cleanly onto the mention-resolution pipeline described above: the bi-encoder retrieves an

ordered candidate set and the cross-encoder performs pairwise disambiguation on that set. The contributions of our work concern only the first of these two stages.

The progress described above depends on a common assumption that is sometimes implicit: that large quantities of labelled supervision are available to train the dense retriever. DPR is trained on Natural Questions and TriviaQA; BLINK on roughly nine million Wikipedia-mention pairs; the E5 family on a billion weakly-supervised pairs harvested from the web [9, 10]. Where the target task departs substantially from these training distributions, performance degrades sharply, the BEIR benchmark of Thakur et al. [21] shows that dense retrievers, despite their strength on in-domain MS-MARCO evaluation, are often outperformed by BM25 on heterogeneous, out-of-domain tasks. This observation has motivated a substantial line of work on *low-resource* and *few-shot* dense retrieval. One thread focuses on unsupervised or self-supervised pretraining objectives that do not require labelled query-document pairs: Contriever [22] and coCondenser [23] both produce strong retrievers from purely unlabelled corpora. A second thread focuses on synthetic data generation, using either neural query generators or large language models to manufacture labelled pairs from unlabelled documents: GPL [24], InPars [25], and Promptagator [26] are examples of this direction. These methods share an assumption that the candidate items are documents with rich textual content from which queries can be plausibly generated. They do not directly transfer to a setting in which the corpus consists of short name strings without descriptive paragraphs, because the generative or pseudo-label mechanisms have little material to work with.

Entity linking research has also considered settings where the available context and supervision are substantially more limited than in conventional document-based entity linking. ELQ, for example, addresses entity linking in questions using a bi-encoder architecture and is explicitly designed for short, noisy inputs [27]. Other work has investigated entity linking with incomplete knowledge bases and limited domain resources, showing that the assumptions of standard entity-linking systems do not always hold in practical settings [28, 29]. These studies are closer to the setting considered here than general-purpose dense retrieval because they explicitly address entity linking under restricted context, incomplete resources, or limited supervision. However, their focus remains on linking mentions occurring in questions, conversations, or natural-language text, rather than ranking corporate-name mentions against a large registry of short entity-name strings.

This thesis addresses a less-explored intersection of these two areas. Entity linking in queries has been studied explicitly, with work examining both the task and its evaluation, efficient candidate linking, and the trade-off between effectiveness and efficiency [30, 31, 32]. These studies show that entity linking over short inputs is an established research problem, but their focus is primarily on web-search queries and natural-language queries containing one or more entity mentions. More broadly, canonical entity-linking benchmarks [20,

33] typically treat queries as mention-in-context strings drawn from natural-language text, while candidate entities are paired with rich textual descriptions, such as Wikipedia abstracts, that can be encoded by the model. Similarly, standard low-resource dense-retrieval approaches [22, 25, 26] generally assume documents with sufficient textual content to support query generation or contrastive pretraining objectives. In contrast, the setting considered in this thesis involves short company-name mentions matched against short entity-name strings, with no surrounding context or rich candidate descriptions available. The setting in this thesis differs in three respects. First, both mentions and candidates are *short strings without surrounding context*: a mention is a company name as written in some external source; a candidate is a company name as recorded in the bank's internal registry, and there is no descriptive paragraph to fall back on. Second, the surface variation between mention and matched entity is structured in a way that general entity ranking does not specifically address, legal-form suffixes (`B.V.`, `GmbH`, `Ltd.`), abbreviations, acronyms, partial references, and noise from upstream pipelines all contribute. Third, and most importantly, the labelled supervision is comparatively *small*: while the general dense-retrieval literature operates at the scale of millions to billions of training pairs and explicitly treats anything smaller as the "low-resource" corner case, the labelled data available to us is smaller, and is dominated by categories of name variation (exact matches, simple legal-form differences) on which lexical methods already perform well. The interesting categories from a fine-tuning perspective – partial matches, abbreviations/acronyms, and pairs with no word overlap – are comparatively sparser.

2.2 Lexical Retrieval Methods

While the entity-linking literature has gravitated toward classification-style scoring, the information retrieval community has developed a parallel line of work that addresses the ranking problem directly: given a query, retrieve and rank documents from a potentially very large corpus by a similarity function operating on term statistics. The methods of this line of work, typically called *lexical* or *sparse* retrieval are old, well understood, and still surprisingly competitive. They form the lexical baseline against which the dense methods of this thesis are measured.

The foundational ideas date to Spärck Jones's analysis of inverse document frequency (IDF) [34] as the appropriate re-weighting for term importance, and to the probabilistic framework of Robertson and Walker [35] that produced the BM25 ranking function. Both methods rest on the same basic premise: a query and a document are represented as sparse vectors over a vocabulary of terms, and similarity is computed as a weighted sum over matching terms. TF-IDF weights each term by its frequency in the document multiplied by the logarithm of its inverse document frequency [34]. BM25 refines this with two additional ingredients: document-length normalisation and a saturating term-frequency

function, which together make the score more robust to length variation and to repeated terms.

Crucially for the ranking problem identified in 2.1, lexical methods scale gracefully to large candidate registries. Queries and documents are encoded independently of one another, and the per-term contributions are accumulated through an inverted index that maps each term to the set of documents containing it. The cost of ranking a query against a corpus of N documents is sublinear in N, driven by the lengths of the postings lists for the query terms rather than by the corpus size [36]; in practice it remains on the order of milliseconds per query even over corpora of tens of millions of documents. This is the same algorithmic structure, independent encoding plus an indexed nearest-neighbour search, that the bi-encoder methods (Section 2.3) reproduce in dense embedding space.

For entity ranking on short strings such as company names, the term unit of choice is typically *character n-grams* rather than word tokens. A character n-gram representation splits each string into overlapping sliding windows of length n, often with one or both endpoints anchored to word boundaries, and treats each resulting substring as a vocabulary item. This carries two practical advantages over word-level tokenisation in the short-string regime. First, it is robust to spelling variations, missing or inconsistent punctuation, and concatenation artefacts: "Foobar Industries B.V." and "Fobar Ind. B.V." share some character trigrams even though their word-level tokenisations differ. Second, it captures sub-word similarity that word-level methods miss entirely: "Microsoft" and "Microsystems" share several character n-grams that signal a shared lexical root, whereas their word-token representations are disjoint.

A decade of progress in neural retrieval has not made lexical methods obsolete. In the BEIR benchmark, Thakur et al. [21] showed that BM25 remains competitive under zero-shot evaluation, with dense retrievers trained on MS MARCO frequently failing to generalize across domains and often underperforming BM25 on several benchmark datasets. Subsequent work has examined the conditions under which lexical methods retain an advantage, queries and documents that share substantial surface overlap, training distributions that differ from the target domain, and corpora containing specialised vocabulary not well represented in the dense retriever's pretraining [37]. The contemporary view is that lexical and dense retrieval methods offer complementary strengths: lexical methods provide robust exact matching and strong zero-shot performance, whereas dense methods improve semantic matching. This has motivated hybrid and sparse-neural approaches such as SPLADE [38], which combine neural representation learning with lexical retrieval mechanisms.

The strength of lexical methods, however, is also their fundamental limitation. They reward surface overlap and they cannot reward anything else. A mention and its true positive client that share few or no character n-gram tokens ("IBM" and "International Business Machines") receive a low score, regardless of how semantically related they may

be. The categories of name variation discussed in 3.2.2 are explicitly designed to expose this boundary: TF-IDF should be expected to perform near-perfectly on exact and near-exact matches, and its performance should degrade as the lexical overlap between mention and positive fades. This sets up the central question this thesis addresses with respect to lexical methods: whether a dense retriever, once fine-tuned on the task, can pick up the cases on which lexical similarity is failing without sacrificing the cases on which it succeeds.

2.3 Dense Retrieval Methods

Dense retrieval represents text not as a sparse vector over a term vocabulary but as a low-dimensional dense vector produced by a neural encoder. Modern bi-encoders are transformer-based language models that map a text to a fixed-length embedding, and the similarity between two texts is measured by a simple function of their embeddings, most commonly the inner product or cosine similarity [39, 37]. For retrieval, the standard architecture is the bi-encoder, in which the query and each candidate are encoded independently by the same model [37]. This independence is what makes the approach practical at scale: candidate embeddings can be computed once and stored in a vector index, so that retrieval reduces to a nearest-neighbour search over that index [39]. Unlike a lexical model, whose score is a function of shared terms, a dense encoder assigns a high similarity to two strings that overlap little on the surface but are semantically related, such as an acronym and its expanded form, provided its parameters have been trained to capture that relationship.

Those parameters are typically learned with a contrastive objective: given pairs of matching text, the encoder is trained so that matched pairs lie closer together in the embedding space than unmatched, or negative, pairs [22]. The choice of negatives has a strong effect on the result and is discussed in Section 2.4. A second factor is the pretraining of the underlying encoder, which supplies the general language understanding that the contrastive step refines.

Because a dense retriever learns semantic similarity from its training data rather than from explicit lexical matching, its behaviour is tied to that data. As discussed in Section 2.2, this is why dense retrievers can be outperformed by BM25 out of domain, and why lexical and dense methods are regarded as complementary rather than one strictly dominating the other.

The E5 models reduces this domain sensitivity by pretraining the models on large and diverse corpora. They are trained with weakly-supervised contrastive pretraining on a large collection of naturally occurring text pairs gathered from the web, followed by supervised fine-tuning, and report strong results across standard embedding and retrieval benchmarks [9, 40, 10]. A practical detail of E5 is that inputs are marked with short prefixes, “query:” and “passage:”, that indicate the role of each text; these prefixes are part

of the training procedure and are reused when the model is applied [9]. The multilingual variant of E5 follows the same recipe but is initialized from a multilingual transformer, XLM-RoBERTa [41], and trained on text pairs spanning one hundred languages, so that a single model embeds text from many languages and scripts into a shared space [10]. General-purpose encoders of this kind provide a strong off-the-shelf starting point for retrieval, although, as the BEIR results indicate, benchmark performance does not by itself guarantee effectiveness on narrow matching tasks.

2.4 Adapting Pretrained Encoders

A pretrained encoder does not always transfer well to a task or domain it was not originally trained for. To obtain good performance on a target task it is therefore commonly adapted by fine-tuning on task/domain specific data.

2.4.1 Parameter-Efficient Fine-Tuning

The most direct way to adapt a pretrained encoder is full fine-tuning, where in every parameter of the model is updated on the target data [42]. While effective full fine-tuning is parameter inefficient because it requires storing a separate copy of the model weights for each downstream task. Parameter-efficient fine-tuning (PEFT) methods were introduced to reduce this cost. They keep the pretrained weights frozen and train only a small number of task-specific parameters. Adapter tuning, which inserts small trainable modules into the layers of a frozen Transformer, was one of the earliest examples of this approach [42].

Low-Rank Adaptation (LoRA) is a widely used PEFT method [43]. Rather than adding new layers, LoRA keeps the pretrained weight matrix 'W' frozen and learns a small update to it. It trains two low rank matrices 'A' and 'B' and adds their product 'BA' to the pretrained weight matrix 'W'. Only A and B change during training, W remains fixed. Because the rank of the update is much smaller than the dimension of W, the number of trainable parameters is a small fraction of the full model, and the learned update can be merged back into the original weights at inference time so that no additional latency is introduced [43]. This directly addresses the expense of full fine-tuning: because only the small low-rank matrices are trained and stored for each task, the memory required during training and the storage required per adapted model are reduced by a large factor, while the frozen base weights are shared across tasks. The original work reports that LoRA matches the performance of full fine-tuning across a range of tasks while training far fewer parameters [43].

2.4.2 Negative Sampling for Contrastive Training

The contrastive loss used to train a dense retriever requires, for each positive pair, a set of negatives to contrast against, and the way these negatives are chosen affects the resulting model [37, 22]. The simplest strategy is to use in-batch negatives, treating the other positives in a training batch as negatives for a given query; this is computationally cheap, but the resulting negatives are often easy, sharing little with the query [22].

To provide a stronger signal, one approach is to mine hard negatives, candidates that are similar to the query but are not correct matches. One cheap source of hard negatives is from a lexical retriever such as BM25, whose top-ranked non-matching candidates are lexically close to the query. Other methods obtain negatives from the dense model itself: ANCE periodically re-encodes the corpus during training and draws negatives from the model's own current nearest neighbours [44], while RocketQA combines cross-batch negatives with a denoising step intended to remove false negatives from the mined set [45]. The literature reports that hard negatives tend to improve retrieval quality relative to in-batch negatives [44, 45]. The evidence on lexically mined BM25 negatives specifically is, however, mixed. They are simple to obtain and in many settings effective [37], but because a lexical retriever's top-ranked candidates can include unlabeled true matches or candidates which are too similar to the true match, BM25 mining can introduce false negatives that degrade training; this has led some work to add an explicit denoising step [45] and other work to prefer negatives mined from the dense model itself over static lexical ones [44].

Chapter 3

Methodology

This chapter describes the methodology used in this work and is structured as follows: a formal task definition (Section 3.1); the two datasets used in this work (Section 3.2); the retrieval systems compared (Section 3.3); the experiment pipeline shared across systems (Section 3.4); the evaluation metrics (Section 3.5); and the significance testing (Section 3.6).

3.1 Task Formulation

The task addressed in this thesis is entity ranking, defined as follows. Given a mention string q referring to a company and a fixed candidate registry $\mathcal{C}$ of client names (c_i) and their corresponding unique identifiers $(oid(c_i))$,

$$\mathcal{C} = \{(c_1, oid(c_1)), (c_2, oid(c_2)), \ldots, (c_N, oid(c_N))\}$$

return a ranked list of the top-K candidate identifiers that are most likely to refer to the same legal entity as q. For simplicity, we refer to the candidate registry as $\mathcal{C} = \{c_1, c_2, \ldots, c_N\}$ in the following sections, with the understanding that each candidate c_i is associated with a unique identifier $oid(c_i)$. Formally, a model is a function $f_\theta : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ mapping a (mention, candidate) pair to a similarity score, where $\mathcal{S}$ denotes the space of strings. The score is bounded to [0,1] for the non-negative TF-IDF vectors, but the dense encoders produce sign-varying embeddings whose inner-product similarity can be negative, so we keep the codomain general. At inference time, the top-K retrieval for a mention q is

$$\mathrm{TopK}(q;\, f_\theta, \mathcal{C}) \;=\; \underset{K}{\mathrm{argtop}} \left\{ f_\theta(q, c) \;:\; c \in \mathcal{C} \right\},$$

where argtop_K returns the K identifiers of the candidates with the highest scores. Each mention q has a set of ground truth positive clients $G(q) \subseteq \mathcal{C}$, drawn from labels linking the mention to one or more clients in the candidate corpus. Most mentions have a single positive, but mentions with multiple positives are permitted. For example, a mention may match several alternate names for the same corporate entity, each recorded as a separate client in the registry.

This formulation admits both classical lexical methods (TF-IDF with character n-grams) and learned dense embeddings (transformer bi-encoders) as instances of f_θ, which permits direct comparison under a unified evaluation protocol.

3.2 Datasets

We used two distinct datasets: the GLEIF dataset, created in this work by joining publicly available legal-entity reference data against the client registry: and a secondary transactions dataset, in which transaction descriptions act as mentions against the same client corpus. Both datasets are partitioned into train, validation, and test splits using the tree-disjoint protocol detailed in Section 4.1. The two datasets are described below.

3.2.1 The candidate corpus: internal client registry

The candidate corpus, referred to as *client registry*, is a table of 446,501 rows sampled from ING's internal registry. We refer to the entities in this registry as *clients* for conciseness, although it also contains non-client relations such as prospects and market-analysis targets. Each row associates a canonical client name with an `org_id`, a unique identifier, and with associated metadata. The metadata includes a `parent_id` which points towards the parent company. We derive the `tree_id` by following each client up the ownership chain to the top level parent. All clients sharing the common ultimate parent are assigned the same `tree_id`. The intent is to build a tree for all clients belonging to the same corporate group, whether ultimate parent, an intermediate subsidiary, or a direct alias. For the transactions dataset we use an extended client registry, since it contains clients not present in the original candidate corpus: appending these adds 5,500 rows, giving an extended registry of 452,001 clients. GLEIF experiments retrieve against the 446,501-row registry; transactions experiments retrieve against the 452,001-row extended registry.

3.2.2 GLEIF Dataset

The Global Legal Entity Identifier Foundation (GLEIF) [1] maintains a public registry of legal entities indexed by Legal Entity Identifier (LEI), a 20-character ISO 17442 code uniquely identifying each legal entity. Each GLEIF record contains a primary legal name and may contain up to five alternative legal names along with their corresponding LEI codes. The client registry maps internal organisation identifiers (`org_id`) to client names along with associated metadata; entries in client registry carry the LEI of the underlying legal entity where known. The complete table contains several million legal-entity records, each a unique legal entity.

[1] GLEIF

The GLEIF dataset is constructed by left-joining the GLEIF records against the client registry on LEI codes, retaining only those rows for which an `org_id` could be resolved. Both the primary legal name and any alternative names are then used to form (mention, client) pairs, where the mention is a GLEIF-side legal name (primary or alternative) and the client is the corresponding client name from the registry. The clients from the resultant dataset create the client registry for the experiments.

It is worth noting that we can construct many more (mention, client) pairs by using the names of the parent or sibling entities of a given client from its corporate tree, but we avoid this as some trees are very large (more than 1000 clients) and many of the names might not be valid matches for the mention depending on what we consider a match to be. For example, consider a client "XYZ Bank B.V", which is a parent of "ABC investment Holdings B.V". A mention which matches to "ABC investment Holdings B.V" should not be considered a match for "XYZ Bank B.V" because even though they are part of the same corporate tree, they are separate entities. On the other hand, we might want to match a mention matching to "XYZ Bank B.V" to also match to "XYZ Bank gmbh" because they are part of the same corporate tree and are likely to be just a branch in different countries. In the interest of simplicity, we only consider the direct matches between the GLEIF legal/alternative names and the client names in the registry.

After deduplication, the raw dataset contains 510,402 pair-level rows. We categorize these pairs by comparing the two strings after light normalisation (lowercase, strip punctuation, collapse whitespace). The category is decided by the first rule that matches, in order:

- **Exact match**: Two normalised strings are byte-identical.
- **Only Legal Form Difference**: They become byte-identical after also stripping legal-form suffixes (Ltd, Inc, B.V., etc.)[2]
- **Abbreviations/Acronyms**: One string equals the initials of the other (e.g. "IBM" $\leftrightarrow$ "International Business Machines").
- **Partial match**: They share at least one word (after legal-form stripping) but none of the above apply. A word in this context is defined as a string between two whitespace characters.
- **No Overlap**: None of the above conditions holds.

The distribution of the five categories in the raw dataset is shown in Table 3.1. Exact matches and only-legal-form differences jointly account for over two thirds of the pairs, while abbreviations are very rare (446 pairs, under 0.1%).

[2]Complete list can be found in Appendix A.

Category	Pair Count	Percentage Share
Exact match	266,116	52.1%
Only legal-form difference	102,056	20.0%
Partial match	95,625	18.7%
No overlap	46,159	9.0%
Abbreviations/acronyms	446	0.1%
Total	510,402	100%

Table 3.1: Category distribution of the raw GLEIF dataset (510,402 pair-level rows), assigned by the first-matching rule above.

The dataset is naturally dominated by easy cases: exact matches and legal-form differences are straightforward for lexical methods to match, while partial matches are a bit more challenging. No-overlap cases and abbreviations, while rare, represent a particularly difficult case for lexical methods, as they often share little surface overlap with their full-form counterparts.

3.2.3 Transactions Dataset

The transactions dataset comprises 54,812 (mention, client) pairs where the mention is a free-text transaction counterparty names entered in payment narrative and the client is the matched canonical entity. Whereas GLEIF mentions are themselves curated legal names, transaction mentions are noisier: they may contain location strings, padding, spelling errors, concatenated fields, etc. The transactions dataset contains clients which were not in the client registry which was created with the GLEIF dataset (section 3.2.2), hence, we expand the client registry by appending these clients. For the experiments we use the the client registry with GLEIF dataset and the expanded version with the transactions dataset. For simplicity we refer to both as the client registry throughout this thesis.

The category distribution of the transactions dataset (Table 3.2) differs markedly from GLEIF: it is dominated by *partial matches* (52.0%) rather than exact matches, reflecting the noisier, free-text nature of transaction data. Abbreviations are effectively absent (a single pair in the entire dataset), so this category cannot be meaningfully evaluated on the transactions distribution.

The transactions dataset serves a different purpose from the GLEIF data: it stress-tests the generalisation of methods that have only seen GLEIF-shaped training data to a substantively different mention distribution.

Category	Pair Count	Percentage Share
Partial match	28,486	52.0%
Exact match	18,396	33.6%
Only legal-form difference	6,400	11.7%
No overlap	1,529	2.8%
Abbreviations/acronyms	1	0.0%
Total	54,812	100%

Table 3.2: Category distribution of the raw transactions dataset (54,812 pair-level rows).

3.3 Models

We compare nine retrieval systems, summarised in Table 3.3: a lexical baseline (`tfidf`); two zero-shot pretrained dense retrievers (`e5-base`, `e5-large`); four LoRA fine-tunes of e5-base; and two reciprocal-rank-fusion ensembles. This section describes each family in turn; the fine-tuning and ensemble configurations are described in chapter 4.

Short name	System	Training data
`tfidf`	TF–IDF (char 1–3-gram)	Vectoriser fitted on client registry
`e5-base`	e5-base, zero-shot	Pretrained only
`e5-large`	e5-large, zero-shot	Pretrained only
`gleif-ft`	e5-base + LoRA	GLEIF (all cats., rebalanced)
`txn-ft`	e5-base + LoRA	Transactions (all cats.)
`gleif-ft-hard`	e5-base + LoRA	GLEIF (hard cats. only)
`txn-ft-hard`	e5-base + LoRA	Transactions (hard cats. only)
`ens-gleif`	RRF(`tfidf`, `gleif-ft-hard`)	(rank fusion)
`ens-txn`	RRF(`tfidf`, `txn-ft-hard`)	(rank fusion)

Table 3.3: Retrieval systems compared in this thesis. "FT" denotes LoRA fine-tuning of e5-base; ensembles are reciprocal-rank fusions, not trained models.

We organise the experiments into two comparisons. The first is a standalone-retriever comparison between `tfidf`, two zero-shot `E5` models, and the two all-category fine-tuned models (`gleif-ft`,`txn-ft`). This comparison addresses how zero-shot dense retrievers perform relative to the TF-IDF baseline (RQ1), how task-specific fine-tuning changes this (RQ2), and in which categories dense retrieval helps (RQ3). The second is an ensemble comparison between `tfidf`, each ensemble (`ens-gleif`,`ens-txn`), and the hard-category fine-tuned models it was built from (`gleif-ft-hard`,`txn-ft-hard`). This comparison addresses whether fusing a category-specialised dense retriever with TF-IDF improves retrieval and where the gains come from (RQ4).

3.3.1 Lexical Baseline

The lexical baseline of a `tfidf` model with character (1-3)-gram tokenisation serves as a simple yet strong baseline.

Tokenisation. Unlike the word-level tokenisation, character n-gram tokenisation is robust to spelling variations. We use character n-grams of length 1 through 3 with word-boundary awareness, corresponding to the `char_wb` analyser in `scikit-learn`'s `TfidfVectorizer`. Using this setting each token is a sequence of 1–3 consecutive characters that does not cross a whitespace boundary, the empty positions immediately inside a word boundary are padded with a space character so that initial and final n-grams remain distinguishable from their internal counterparts. For example, the string "ING Bank" produces (unigram, bigram, and trigram) token including "`_I`", "`_IN`", "`IN`", "`ING`", "`NG`", "`NG_`", "`_B`", "`_Ba`", and so on, where "`_`" denotes a boundary-space.

Weighting. Each string s is represented as a sparse TF–IDF vector $\mathbf{v}_s \in \mathbb{R}^{|V|}$ over the token vocabulary V, with

$$v_{s,t} \;=\; \mathrm{tf}(t,s) \cdot \log\!\left(\frac{1+N}{1+\mathrm{df}(t)}\right) + 1 \tag{3.1}$$

where N is the fitted-corpus size and $\mathrm{df}(t)$ is the document frequency of token t (the smoothed IDF is scikit-learn's default). Vectors are L^2-normalised so cosine similarity reduces to a sparse inner product. The vectoriser is fitted on the preprocessed client registry(lowercase, whitespace strip, unidecode) with the `TfidfVectorizer` and is reused across both evaluation distributions.

Retrieval. At inference each test mention is transformed to $\mathbf{v}_q$ by the fitted vectoriser and scored against every candidate as

$$f_{\text{TF–IDF}}(q,c) \;=\; \cos(\mathbf{v}_q, \mathbf{v}_c) \;=\; \mathbf{v}_q^{\top}\mathbf{v}_c \tag{3.2}$$

computed in bulk as a single sparse matrix product between the mention batch and the candidate matrix using the `sparse_dot_top_n` library[3]. The top-K candidates per mention are extracted from the resulting score matrix.

3.3.2 Pretrained Dense Retrievers

The pretrained dense retriever is based on the E5 family of multilingual text embedding models [9, 10], which have been trained on a large corpus of text and can capture semantic

[3] sparse_dot_top_n library

similarity between mentions and candidates. With names in the dataset appearing in multiple languages and scripts, these models are well suited for the task. We evaluate both the E5-base and E5-large variants, which differ in their model size and capacity.

3.3.3 Fine-Tuned Dense Retrievers

The fine-tuned dense retrievers are obtained by performing parameter-efficient fine-tuning of the E5-base model on the GLEIF and transactions datasets. We perform PEFT using Low-Rank Adaptation (LoRA) [43], which allows us to adapt the model to the specific task of entity ranking while keeping the number of trainable parameters small.

We produce four fine-tunes in total. Two are *standalone* retrievers, `gleif-ft` and `txn-ft`, trained on the all-category training data of the GLEIF(rebalanced) and transactions datasets respectively. The other two, `gleif-ft-hard` and `txn-ft-hard`, are trained only on the hard categories for use as ensemble components (Section 4.4 and 4.2.1). All four share an identical fine-tuning configuration (Table 4.2) and the same procedure (Chapter 4); they differ *only* in their training data. Both datasets are split 80/20 into train and test using the tree-disjoint protocol (Section 4.1), which prevents any client, mention, or corporate tree from appearing on both sides, and we additionally enforce no cross-distribution leakage between the two datasets' splits. The training sets are further split 80/20 into train and validation using the same tree-disjoint protocol for training.

The training of the fine-tuned dense retrievers is performed using a contrastive InfoNCE loss, which takes in a mention, a positive candidate, and hard negatives mined from the `e5_base` model predictions. We also considered training with hard negatives mined from the `tfidf` model predictions, but we found that this leads to the fine-tuned models losing their ability to match exact match and legal-form difference cases, which should be easy to match for any models. The details of the training procedure, including the hyperparameters and data preparation are described in detail in chapter 4

3.3.4 Ensemble Model

Additionally, we also compare the baseline and dense retrievers to an ensemble model that combines `tfidf` and the fine-tuned models. `tfidf` model returns a cosine similarity score over sparse vectors, while the dense retriever returns cosine similarity scores over dense embeddings. These scores are on different scales and so we cannot simply combine the rankings and re-rank based on the similarity score. We create the ensemble model using reciprocal-rank fusion (RRF) [46], which combines the rankings from multiple models to produce a single ranking based purely on rank positions, without requiring the models scores to be on the same scale. For each candidate c appearing in the top-K list of either input ranking, RRF computes a combined score as follows:

$$\mathrm{RRF}(c) = \sum_{r \in R} \frac{1}{k_{\mathrm{rrf}} + r(c)}$$

Where R is the set of input rankings, r(c) is the rank of c in ranking r, and k_{rrf} is a smoothing constant fixed to 60, following [46].

The fine-tuned models we use for the ensemble models differ from the earlier fine-tuned models. For the ensemble model we train the `e5-base` model using the same procedure as before but only on the harder mentions i.e, partial match, no-overlap, and abbreviations. We create two ensemble models by combining the `tfidf` model with an e5-base model fine-tuned on GLEIF and an `e5-base` model trained on transactions dataset.

Each ensemble combines the `tfidf` baseline with a hard-category fine-tuned model (`gleif-ft-hard` or `txn-ft-hard`). `tfidf` already covers the exact-match and legal-form-difference categories almost perfectly, so using a dense component specialised on precisely the categories that lexical matching handles poorly should give better coverage. We therefore train the ensemble's dense component only on the partial-match, no-overlap, and abbreviation categories. This gives us two ensemble models, `ens-gleif` = RRF(`tfidf`, `gleif-ft-hard`) and `ens-txn` = RRF(`tfidf`, `txn-ft-hard`), evaluated on the GLEIF and transactions distributions respectively.

3.4 Experiment Pipeline

All the models run through roughly the same pipeline over the two datasets: preparing data from the client registry and either the GLEIF or transactions test set(Section 3.2), creating vectorizer/index, retrieving ranked lists for each mention.

Data Preparation The client registry and mentions from the test sets have to be preprocessed before running `tfidf` model on them as the lexical model distinguishes between a uppercase 'A' and a lowercase 'a'. We also use the unidecode library to convert all non-ASCII characters to ASCII characters, this preprocessing normalises the strings to a level where the task becomes very easy for `tfidf` model. For the pre-trained dense retriever models and the fine-tuned models we use the raw texts, as they are trained to understand the underlying semantics and variations.

Encoding Client Registry The client registry is encoded into a searchable index. For `tfidf` model, this is a sparse matrix over fitted character n-gram vocabulary, L^2-normalised so that cosine similarity reduces to a sparse inner product. For the e5 Models, this is a matrix of L^2-normalised dense embeddings produced by the encoder, indexed via FAISS with an inner-product metric.

Encoding Mentions and Retrieval At inference, each test mention is encoded by the same vectoriser or encoder used for the candidates (with the "`query: `" prefix for E5) and a cosine similarity score is computed against every candidate. The top-K candidates by score are retrieved per mention. All mentions are encoded simultaneously rather than one-mention at a time, this reflects real-world usage of such systems in banking scenarios. The search operations for `tfidf` model are carried out using the `sparse_dot_top_n library` for quick sparse matrix multiplications and for the dense model we use FAISS search.

Creating Submission The per-mention top-K candidate identifiers and their similarity scores are written to a submission in a uniform format, with columns `test_index`, `company_id` (comma-separated list of the predicted `org_id`s in rank order), and `distances`. This same format is produced by every model, which allows downstream evaluation and significance testing to be model-agnostic.

Ensemble The ensemble stage takes two model submissions, the TF–IDF baseline and one fine-tuned dense retriever, and combines them via the reciprocal rank fusion mentioned in Section 3.3. The two component top-15 lists per mention are fused into a single re-ranked list of length between 15 and 30, which is written as a new submission in the same format described above. Two ensembles are produced, one per dataset: `gleif-ft-hard` + `tfidf` for the GLEIF-side evaluation, and `txn-ft-hard` + `tfidf` for the transactions-side evaluation.

3.5 Evaluation

To evaluate the performance of the models on the test sets of both datasets, we use the standard information retrieval metrics Mean Average Precision (MAP) and Recall@k. Both metrics are computed at the mention level and averaged across all mentions for each category and overall dataset. Category is a property of a (mention, positive) pair, so a mention with positives in more than one category contributes to every category cell that applies to it. Per-category mention counts therefore need not sum to the overall count, and the overall cell always includes every mention exactly once.

Correct Match: A retrieval system produces a ranked list of candidates for each mention, $R_q = (c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(K)})$ where K is the number of candidates retrieved and π is the ranking permutation. A mention q and a ranked candidate $c_{\pi(k)}$ at position k are considered a correct match if the candidate $c_{\pi(k)}$ is in the set of ground truth positives $G(q)$ for the mention q.

Recall@k: Recall at k (Recall@k) measures the fraction of a mention's ground truth positives that the system retrieves in its ranked list of candidates up to position k. It is defined as:

$$\text{Recall@}k\,(q) = \frac{\left|\{\, j \le k \;:\; c_{\pi(j)} \in G(q)\,\}\right|}{|G(q)|}$$

where $c_{\pi(j)}$ is the candidate at position j in the ranked list R_q, and $G(q)$ is the set of ground truth positives for mention q. For a mention with a single positive, this reduces to binary values {0,1},checking if the correct candidate is retrieved in top-k. For mentions with multiple positives, it returns a value in the range [0,1].

Recall@k is the appropriate primary metric for a candidate-generation task, as the downstream disambiguation stage can only rank what the retrieval stage returns. If the correct candidate is not retrieved, it cannot be ranked correctly. The metrics we report differ between the two comparisons. For the standalone-retriever comparison we report recall@10 and recall@20 together with MAP, for each category and over the whole test set. For the ensemble comparison we report recall@20 and recall@30; MAP is not reported for this comparison.

Mean Average Precision: Mean Average Precision (MAP) measures how well the system ranks all of a mention's ground-truth positives, not just the first. Average precision AP is defined as:

$$\text{AP}(q) = \frac{1}{|G_q|} \sum_{k=1}^{|R_q|} \mathbf{1}[c_{\pi(k)} \in G(q)] \cdot \text{prec@}k(q)$$

Where prec@$k(q)$ is the precision at k for mention q measured as the fraction of the first k predictions that are correct and $\mathbf{1}[c_{\pi(k)} \in G(q)]$ is an indicator function that returns 1 if the candidate at position k is a ground truth positive and 0 otherwise. AP rewards rankings where correct matches appear at the top of the list, and penalizes rankings where correct matches appear lower down. MAP is the mean of AP over all mentions. For mentions with a single positive, MAP reduces to the reciprocal rank of that positive.

3.6 Significance Tests

Aggregate mean scores are informative but insufficient on their own as differences of a fraction of a percentage point, or comparisons within small categories such as abbreviations. These require statistical grounding to be interpreted, we compare every evaluated system against reference systems using: a paired significance test, a bootstrap confidence interval on the mean per-mention difference, and a multiple-comparison correction. The reference systems differ between the two comparisons of Section 3.3. In the standalone-retriever comparison, each system is tested against the `tfidf` baseline and the `e5-base` zero-shot

model. In the ensemble comparison, each ensemble is tested against `tfidf` and against the fine-tuned dense retriever it is built from. The remainder of this section describes the tests, corrections and the families over which the correction is applied.

Throughout this section, we let $q_1, \ldots, q_n$ denote the set of mentions in the test set, and $x_i, y_i \in [0, 1]$ denote the per-mention metric values (recall@K or MAP) for models A and B on mention q_i. The paired differences are $d_i = x_i - y_i$.

Initially we considered the paired t-test, which assumes that the differences d_i are normally distributed. However, this assumption fails in our setting. Recall@K takes values in $\{0, 1/m, 2/m, \ldots, 1\}$ where m is the number of ground-truth positives for the mention. In both our datasets, majority of the mentions have only one ground-truth positive, so the differences are mostly in $\{-1, 0, 1\}$, which is clearly not normal. MAP is also bounded in $[0, 1]$ and has a skewed distribution with many mentions having MAP=1.0. Therefore we choose a non-parametric test.

Wilcoxon Signed-Rank Test: We use the Wilcoxon signed-rank test in place of the t-test. The test statistic W is constructed from the ranks of the absolute differences rather than the differences themselves.

Formally, let R_i denote the rank of $|d_i|$ within $\{|d_1|, \ldots, |d_n|\}$, and let $\text{sign}(d_i) \in \{-1, 0, +1\}$. The test statistic W is

$$W = \sum_{i:d_i \neq 0} \text{sign}(d_i) \cdot R_i \tag{3.3}$$

Under the null hypothesis, W has a distribution symmetric around zero determined by the ranks alone, independent of the actual values of the differences. By the central limit theorem, for large n, W is approximately normal with mean 0 and known variance, giving a two-sided p-value via the normal approximation. For small n, the exact distribution is used.

Wilcoxon test drops observations with $d_i = 0$ before ranking, which looses information about how many pairs are tied. Keeping in mind that the majority of our mentions produce zero differences, we use the zsplit variant of the Wilcoxon test, which splits the zero-difference pairs evenly between the two groups. The implementation is done using `scipy.stats.wilcoxon`.

Paired Bootstrap Confidence Intervals: A p-value indicates only wether an effect is non zero, it says nothing about the effect's magnitude. For this we report a 95% confidence interval on the mean paired difference $\bar{d}$ computed via a paired bootstrap.

Given n paired observations $(d_1, \ldots, d_n)$, we draw $B = 5000$ bootstrap samples, where each sample resamples the index set $\{1, \ldots, n\}$ with replacement to produce indices $(j_1^{(b)}, \ldots, j_n^{(b)})$ and computes the bootstrap replicate mean $\bar{d}^{(b)} = \frac{1}{n} \sum_i d_{j_i^{(b)}}$. The 95% confi-

dence interval is the interval between the 2.5th and 97.5th percentiles of $\{\bar{d}^{(1)}, \ldots, \bar{d}^{(B)}\}$.

Holm-Bonferroni correction: Reporting each test at $\alpha = 0.05$ without correction would inflate the family-wise error rate, since we run many tests per metric. We therefore apply the Holm-Bonferroni procedure to control the family-wise error rate at $\alpha = 0.05$ within each family. The two comparisons define their families differently. In the standalone-retriever comparison, all tests for a given metric form a single family: every candidate system, against both references (`tfidf` and `e5-base`), across the five surface-variation categories and the overall cell, is corrected together. In the ensemble comparison, each (comparison, metric) pair forms its own family of six tests (the five categories plus the overall cell); the two ensemble-reference pairs (ensemble vs. `tfidf` and ensemble vs. its dense component) and the two recall cutoffs are corrected independently.

We apply Holm-Bonferroni correction to control the error-rate at $\alpha = 0.05$ for each family of p-values. The procedure operates as follows. Let $p_{(1)} \leq p_{(2)} \leq \ldots \leq p_{(m)}$ denote the ordered p-values from m tests in the family, and let $H_{(1)}, \ldots, H_{(m)}$ denote the corresponding null hypotheses. The adjusted p-value for the k-th ranked test is

$$\tilde{p}_{(k)} = \max_{j \leq k} \min\Big((m - j + 1) \cdot p_{(j)},\ 1\Big) \tag{3.4}$$

The outer maximum enforces monotonicity of adjusted p-values with respect to raw ranks. A hypothesis $H_{(k)}$ is rejected when $\tilde{p}_{(k)} < \alpha$.

Chapter 4

Experimental Setup

This chapter describes the technical details and decisions that chapter 3 skipped over: the tree-disjoint splitting used to partition both the datasets; the construction of training pairs for fine-tuning; the configurations used for fine-tuning and ensemble; and the implementation environment.

4.1 Tree-Disjoint Splitting

To split the datasets into training and test sets, we can implement a naive mention-level train/test split but this can leak data in two ways: two mentions that share a positive client can end up on opposite sides of the split, and two positive clients sharing a mention can split apart, leaving the same mention on both sides. To solve this initially we considered splitting on components, where a component is a set of mentions and clients connected by shared linkages. Formally, we construct a bipartite graph over the pair-level dataset with mention nodes on one side and client nodes on the other, adding an edge (q_i, c_j) for every (mention, positive) pair. The connected components are assigned to the same side of the split. This ensures that no mention or client is leaked across splits.

Although this strategy solves the leakage issues of the naive split, it still can leak clients from the same tree. Consider two mentions q_1 and q_2 with positives c_1 and c_2 belonging to the same corporate tree T. If the two mentions end up on opposite sides of the split, a model that sees q_1 at training time and learns to represent tree T's naming pattern in embedding space will retrieve c_2 for q_2 at test time more easily than it would for a genuinely unseen entity, not because the model has learned to match q_2 specifically but because the tree structure has been indirectly memorised. The bipartite component construction has no visibility into this as c_1 and c_2 are simply two unconnected nodes.

To prevent this leakage, we create tree-component disjoint splits. We construct components via a union-find data structure over three kinds of node: mention nodes (q_i), client nodes (c_j), and tree nodes (t_k). For each row of the pair-level dataset with mention q_i whose positive client is c_j belonging to tree t_k, we insert the edges (q_i, c_j) and (c_j, t_k). The union-find data structure merges the components of connected nodes as edges are inserted. After all rows are processed, each connected component of the resulting graph

corresponds to a set of mentions that must be kept together during splitting. We refer to this as the tree-disjoint splitting protocol and implement it on both the GLEIF and transactions datasets. Mentions whose positives have no tree information are treated as singleton trees, so they participate in components on the basis of client identity alone. It must be noted that this strategy can still leak data due to missing tree links resulting from data quality issues. We work under the assumption that these cases are rare and that the resulting leakage is negligible in aggregate.

The two datasets, GLEIF and transactions, may share mentions, clients, trees. A model trained on the transactions distribution should not have seen any tree that appears at evaluation on the GLEIF distribution, since we want the GLEIF eval to measure generalisation. The symmetric constraint applies to the GLEIF-trained model on transactions eval. Overlaps within the same "side" i.e, between the two training partitions or between the two test partitions, are not evaluation leakage and are permitted. We therefore enforce cross-distribution disjointedness on the combinations that affect the training and evaluation of the fine-tuned models. `Txn_train` is required to be disjoint from `gleif_test` in mentions, clients, and trees, and `txn_test` is disjoint from `gleif_train` on the same. Overlaps between `txn_train` and `gleif_train` and between `txn_test` and `gleif_test` are permitted, as neither creates evaluation leakage.

Once the components are identified, we assign each component to train and test using multi-label stratified sampling [1]. Because each component contains multiple mentions spanning multiple categories, its "label" is a set rather than a single class, hence the multi-label formulation. We compute the set of categories represented by each component's positives and use `iterstrat` to partition components such that each category's presence is preserved to the extent possible while respecting the component-grouping constraint.

The final target proportions are 80% train and 20% test at the component level; the training-side split further partitions into 80% train and 20% validation. The resulting proportions at the row (mention–positive pair) level deviate slightly from these targets because component sizes vary, but the deviation is small (within about 2 percentage points).

Table 4.1 gives the resulting pair-level category composition of the splits. For GLEIF, the raw 510,402 pairs partition into a 103,367-pair test set and a 407,035-pair training partition; the latter is then rebalanced (Section 4.2.1) to the 143,243 pairs shown. For transactions, the split is discards 4,791 of the 54,812 raw pairs whose mentions, clients, or trees would otherwise overlap the GLEIF evaluation, leaving 50,021 that partition into the 39,042 train–validation and 10,979 test pairs shown.

Table 4.1 gives the resulting pair-level category composition of the splits. For GLEIF, the raw 510,402 pairs partition into a 103,367-pair test set and a 407,035-pair training partition; the latter is then rebalanced (Section 4.2.1) to the 143,243 pairs shown. For

[1] Iterative-Stratification library

transactions, the split is applied without rebalancing.

	GLEIF		Transactions	
Category	Train–val	Test	Train–val	Test
Exact match	30,000	53,803	13,281	3,659
Only legal-form difference	30,000	20,469	4,652	1,301
Partial match	30,000	19,621	19,967	5,686
Abbreviations/acronyms	23,243	88	1	0
No overlap	30,000	9,386	1,141	333
Total	143,243	103,367	39,042	10,979

Table 4.1: Pair-level category composition of the train–validation and test splits. GLEIF train–validation figures are *after* rebalancing (Section 4.2.1); all other columns reflect the natural distribution.

4.2 Training Data Construction

We use the training sets and split it again using tree-disjoint splitting protocol into train-validation sets for fine-tuning the e5-base model. For the resulting train set we construct the input to the contrastive fine-tuning loss as a set of records containing one mention, one positive client, and up to five hard negatives.

4.2.1 Rebalancing GLEIF

The raw training partition inherits the category imbalance of the underlying GLEIF dataset described in 3.2.2. This means that the training set has roughly 70% of rows accounted by the exact match and only legal-form difference. Training a model on this distribution produces can produce undesirable effects like the loss being dominated by easy-category gradients, which the pretrained representation already handles well and which therefore contributes little useful signal. Another effect can be that the model fails to learn from the hard categories. We redistribute the training partition across categories before constructing training pairs. For each category we set a hard limit of $30,000$ on number of rows and then sample randomly from the categories. All categories except abbreviations have more samples than this limit. For abbreviations we create synthetic examples, as the training partition contains only 358 genuine abbreviation pairs (of the 446 in the raw dataset, 88 fall in the test split).

We generate these synthetic examples by taking the exact match and legal-form difference mentions which were rejected during redistribution and abbreviate them by taking the first letter of each word in the mention after stripping legal-forms. After adding these synthetic mentions the abbreviation category reaches 23,243 pairs, and the rebalanced

GLEIF training partition totals 143,243 pairs (30,000 in each of the four non-abbreviation categories plus 23,243 abbreviations). We don't hit the $30,000$ limit due to two filters rejecting the candidates for creating synthetic mentions:

- If the original string had less than two words, for example, Apple Inc, Apple Computers. Abbreviations created from these are rare in real settings and so we drop them.
- If the new abbreviation equals any eval-set mention or any abbreviation already accepted in the run.

We redistribute only for GLEIF train dataset and keep the original distribution for Transactions dataset, because this is not a dataset curated by us rather something used in actual practice.

4.2.2 Hard Negative Mining

We run the pretrained E5-multilingual-base model on ever mention from the redistributed train split against the full client registry and record the top-K_{hn} predicted `org_id`s per mention, with $K_{\text{hn}} = 20$. This produces a per-mention ranked list from which we select hard negatives. For each (mention,client) pair we locate the rank of its corresponding ground truth positive in the ranked list. Three cases arise:

- **Positive at rank 1:** The base model already ranks the positive at the top. so no candidate is ranked above it. This mention contributes its positive to the training set but no explicit hard negatives. The contrastive loss for this mention depends entirely on in-batch negatives.
- **Positive at rank $r > 1$ within the top-k:** All candidates from ranks $1, \ldots, r-1$ are taken as potential hard negatives, excluding any alternate positives.
- **Positive not in top-k:** We consider all the candidates in the list as hard negatives.

The potential hard negative candidates are filtered before being retained as hard negatives. We apply two filters, in order:

Tree filter: Any candidate whose tree_id matches that of the mention's positive is dropped. This prevents the model from pushing embeddings of companies from the same corporate tree apart.

Surface form filter: Any candidate whose surface string, after normalisation, matches the normalised surface string of the mention or of any positive client for the mention is dropped. To normalise we apply Unicode NFKD decomposition with removal of combining characters (accent-folding), lowercasing, punctuation stripping, and whitespace stripping. This prevents the model from pushing away the embeddings of two string which look the same to a normal person.

From the surviving candidates we pick up to top-5 ranked candidates as hard negatives. If a mention had fewer than five candidates surviving the filters, then we use as many explicit hard negatives as it has; the remainder of the loss for that mention depends on in-batch negatives.

Our first consideration for mining hard negatives was to take top-5 non-positive predictions from the `tfidf` baseline predictions. This was rooted in literature but we found empirically that this led the model to push similar looking strings away and degrade the models performance on easier cases of exact match and only legal-form difference. TF-IDF hard negatives frequently share high character overlap with the positive, so the contrastive loss pushes the dense model's representation to distinguish lexically-similar strings even in cases where the surface similarity is genuinely correct evidence of a match. The model unlearns some of its pretraining behaviour on the easy categories. Using hard negatives sourced form the same model and filtering them in this manner made the model correct its own errors while leaving alone the cases it already handles well.

4.3 Fine-Tuning Procedure

We fine-tune the e5-base model with a symmetric InfoNCE contrastive loss computed in each mini-batch. For a batch of B mention–positive pairs (q_i, p_i) with associated hard negatives $\{n_{i,1}, \ldots\}$, the loss for mention q_i is

$$\mathcal{L}_i \;=\; -\log \frac{\exp\left(f(q_i, p_i)/\tau\right)}{\sum_{c \in \mathcal{N}(i)} \exp\left(f(q_i, c)/\tau\right)} \tag{4.1}$$

where $f(q, c) = \cos\left(\mathrm{enc}(q), \mathrm{enc}(c)\right)$ is the cosine similarity between the encoded mention and encoded candidate, τ is the temperature, and $\mathcal{N}(i)$ is the set of candidates for mention i in the batch: the positive p_i, this mention's hard negatives, and all other mentions positives and hard negatives as in-batch negatives. The total batch loss is the mean of $\mathcal{L}_i$ over the batch. Symmetric InfoNCE additionally computes a mirror loss with the roles of mention and candidate swapped i.e., a candidate-to-mention softmax over the mentions in the batch, and averages the two terms. This provides gradient signal from both directions of the pairwise comparison and encourages symmetry in the learned embedding geometry, matching the objective used during E5 pretraining.

Tree-Safe Batching Standard in-batch negative sampling would use every other mention's positive as a negative for the current mention. When two mentions in the same batch have positives from the same corporate tree, this creates the same issue that the tree filter on hard negatives is designed to prevent, the in-batch negative is not genuinely negative.

To handle this, we use a sampler that batches in a tree-safe manner. Before adding a record to the current batch, the sampler checks whether the record's positive-tree set overlaps with the union of positive-tree sets already seen in the batch. If it does, the record is rejected and returned to a deferred queue; otherwise it is added. The deferred queue is drained in subsequent batches when the tree conflict is no longer present. In practice this reduces effective batch size by only a small margin (typically $< 5\%$) while eliminating in-batch tree collisions.

LoRA Configuration Table 4.2 lists the optimisation hyperparameters. The values are conservative by design: a low learning rate, few epochs, high loss temperature τ, and a moderate LoRA rank together limit how far the fine-tune can move from the pretrained representation, on the grounds that excessive movement was observed in early experiments to cause the model to lose ground on the easy categories. The complete list of hyper parameters used can be found in appendix B.

Table 4.2: Fine-tuning hyperparameters. The same configuration is used for the GLEIF-based and the transactions-based fine-tunes.

Hyperparameter	Value
Base model	E5-multilingual-base (110M params)
Epochs	3
Batch size	128
Optimiser	AdamW
Learning rate	2×10^{-5}
Loss temperature τ	0.1
LoRA rank r	16
LoRA scaling α	32

During training, validation loss is computed at the end of each epoch on the validation split. the checkpoint with lowest validation loss is retained as the final model. In our runs the validation loss keeps decreasing at the end of the final epoch, indicating that the model is not yet converged and longer training runs would likely reduce the loss, but at the cost of increased risk of forgetting on the easy categories.

4.4 Ensemble Configuration

The ensemble combines the character 3-gram TF–IDF baseline with one fine-tuned dense retriever via reciprocal rank fusion. We take the top 15 predictions from each of the two component models before fusing. At $K = 15$ the component lists individually have high recall on both distributions, so their union is a reasonable candidate pool for the fused ranking. The resulting fused list has length between 15 (in the case of full overlap between the two component lists) and 30 (in the case of no overlap). At evaluation time we report recall@20 and recall@30; when the fused list is shorter than the evaluation K, the metric saturates at the actual list length. The RRF smoothing constant is fixed at $k_{\text{rrf}} = 60$, the value from the original RRF paper [46]. We did not tune this constant.

4.5 Implementation

The pipeline is implemented in Python. Tabular data manipulation uses `pandas`; the TF–IDF vectoriser is the `TfidfVectorizer` from scikit-learn with the character-word-boundary analyser (`analyzer='char_wb'`, `ngram_range=(1,3)`). The pretrained E5 checkpoints are loaded via `sentence-transformers`; the LoRA adapters are implemented using the `peft` library. Statistical testing uses `scipy.stats` for the Wilcoxon test and `statsmodels` for the Holm–Bonferroni correction.

The full pipeline (data preparation, splitting, training-pair construction, inference, evaluation, and significance testing) is using `ordeq` framework. All splits use a fixed random seed and are deterministic given the input tables. The fine-tuning random seed is also fixed, but exact reproducibility of the trained model depends on GPU-level nondeterminism in the underlying CUDA kernels.

Chapter 5

Results and Analysis

5.1 Results

We report the two comparisons defined in Section 3.3, each on both evaluation distributions. The *standalone-retriever* comparison (Section 5.1.1) evaluates the `tfidf`, the two zero-shot E5 models, and the two all-category fine-tunes, reporting MAP and recall@20. The *ensemble* comparison (Section 5.1.2) evaluates each reciprocal-rank-fusion ensemble against the `tfidf` and against its fine-tuned dense component, reporting recall@20 and recall@30. In every table, rows are ordered by increasing overall recall@20.

Significance is shown relative to the `tfidf`: a value marked * is significantly higher than `tfidf` and a value marked † significantly lower, under the paired Wilcoxon signed-rank test with Holm correction described in Section 3.6 ($p_{\text{adj}} < 0.05$, two-sided). The full per-reference, per-category significance tables are given in Appendix C.

5.1.1 Standalone-Retriever Comparison

GLEIF test set

Table 5.1 reports the standalone results on the GLEIF test set. The `tfidf` gets the highest MAP (0.841) significantly higher than every dense model, whereas at recall@20 the top of the table is a statistical tie: `e5-large` (0.882), `txn-ft` (0.881) and `tfidf` (0.878) do not differ significantly. All three tested dense models, however, do out-perform `tfidf` significantly on no-overlap pairs, and `txn-ft` does so on partial matches (0.809 vs. 0.779).

Transactions test set

Table 5.2 reports the standalone results on the transactions test set. Here `tfidf` is significantly higher than all the dense models on MAP and on recall@20 against `e5-large` and `gleif-ft`. The exception is `txn-ft`, whose recall@20 does not differ significantly from `tfidf` (-0.006, $p_{\text{adj}} = 1.0$). By category, no dense model significantly beats `tfidf`, the closest cases are `txn-ft` on partial match (0.934 vs 0.949) and on no-overlap (0.927 vs 0.906), neither significant.

	Overall		recall@20 by category				
Model	MAP	R@20	exact	legal	partial	no-ovl	abbr
`tfidf`	**0.841**	0.878	**0.999**	**0.997**	0.779	0.126	0.070
`e5-base`	0.773	0.860	0.992	0.949	0.741	0.140	0.144
`gleif-ft`	0.831†	0.874	0.997	0.990	0.754†	0.154*	**0.246**
`txn-ft`	0.810†	0.881	0.996	0.977†	**0.809***	0.159*	0.167
`e5-large`	0.822†	**0.882**	0.998	0.988	0.790	**0.177***	0.161

Table 5.1: Standalone comparison on the GLEIF test set: overall MAP and recall@20, and per-category recall@20. Rows after the `tfidf` baseline ordered by increasing overall recall@20; best per column in bold. $^{*}/^{\dagger}$ = significantly higher/lower than `tfidf` (paired Wilcoxon, Holm-corrected, $p_{\text{adj}} < 0.05$). `e5-base` is a second reference and was not tested against `tfidf`.

	Overall		recall@20 by category			
Model	MAP	R@20	exact	legal	partial	no-ovl
`tfidf`	**0.897**	**0.966**	**0.986**	**0.989**	**0.949**	0.906
`e5-base`	0.795	0.907	0.970	0.957	0.860	0.819
`gleif-ft`	0.854†	0.937†	0.985	0.966	0.903†	0.862
`e5-large`	0.845†	0.949†	0.984	0.975	0.923†	0.897
`txn-ft`	0.856†	0.959	0.980	0.976	0.943	**0.927**

Table 5.2: Standalone comparison on the transactions test set: overall MAP and recall@20, and per-category recall@20. Rows after the `tfidf` baseline ordered by increasing overall recall@20; best per column in bold. $^{*}/^{\dagger}$ = significantly higher/lower than `tfidf`. `e5-base` was not tested against `tfidf`. (No abbreviation mentions.)

5.1.2 Ensemble Comparison

Each ensemble fuses the `tfidf` with a hard-category fine-tuned model via reciprocal rank fusion; these hard-category components (`gleif-ft-hard`, `txn-ft-hard`) differ from the all-category `gleif-ft`/`txn-ft` of the previous subsection. On each test distribution we report both the *matched* ensemble, whose dense component was trained on that distribution, and the *cross-distribution* ensemble, whose dense component was trained on the other distribution, together with the two dense components in isolation. Only the ensemble rows carry significance markers, as the hard-category components were not separately tested against `tfidf`.

GLEIF test set

Table 5.3 reports the ensemble results on the GLEIF test set. The matched ensemble, RRF(`tfidf`, `gleif-ft-hard`), gets higher recall than either of its components overall (recall@20 0.898 vs. `tfidf` 0.878 and its dense component 0.881). Its overall gain over `tfidf` is significant (+0.019, 95% CI $[0.018, 0.020]$) and is carried entirely by partial matches (+0.075) and no-overlap pairs (+0.054), both significant; exact and legal-form

recall are unchanged (both ≈ 0.999), and the large relative gain on abbreviations (+0.131, $n = 88$) is not significant after correction. On GLEIF the cross-distribution ensemble performs almost the same as the matched one (recall@20 0.897 vs. 0.898) and is significantly better than `tfidf` (+0.019, 95% CI [0.017,0.020]).

	Overall		Exact		Legal-form		Partial		No overlap		Abbrev.	
System	@20	@30	@20	@30	@20	@30	@20	@30	@20	@30	@20	@30
`tfidf`	0.878	0.882	**0.999**	**0.999**	0.997	**0.998**	0.779	0.797	0.126	0.133	0.070	0.087
`gleif-ft-hard` *(matched)*	0.881	0.881	0.997	0.997	0.968	0.968	0.806	0.806	0.170	0.170	0.187	0.187
`txn-ft-hard` *(cross)*	0.886	0.891	0.997	0.997	0.984	0.988	0.820	0.838	0.165	0.175	0.167	0.184
RRF(`tfidf`, `txn-ft-hard`) *(cross)*	0.897*	0.900*	**0.999**	**0.999**	**0.998**	**0.998**	**0.855***	0.866*	0.176*	0.184*	0.178	0.178
RRF(`tfidf`, `gleif-ft-hard`) *(matched)*	**0.898***	**0.901***	**0.999**	**0.999**	**0.998**	**0.998**	0.854*	**0.867***	**0.180***	**0.188***	**0.201**	**0.201**

Table 5.3: Ensemble comparison on the GLEIF test set (recall@20 / recall@30). Rows after the `tfidf` baseline ordered by increasing overall recall@20; best per column in bold. "matched"/"cross" denote the ensemble whose dense component was trained on GLEIF / on transactions. * = ensemble significantly higher than `tfidf` (paired Wilcoxon, Holm-corrected). Dense components were not tested against `tfidf`.

Transactions test set

Table 5.4 reports the ensemble results on the transactions test set. The matched ensemble, RRF(`tfidf`, `txn-ft-hard`), gets the highest recall overall (recall@20 0.982 vs. `tfidf` 0.966 and its dense component 0.968); its overall gain over `tfidf` is significant (+0.017, 95% CI [0.014, 0.019]), carried by partial matches (+0.028, significant), while the large no-overlap gain (+0.067, $n = 329$) is not significant after correction and exact and legal-form recall are unchanged. The cross-distribution ensemble, RRF(`tfidf`, `gleif-ft-hard`), improves over `tfidf` by less (recall@20 0.971) and its overall gain is *not* significant after correction (+0.006, $p_{\text{adj}} = 0.78$); its dense component is much weaker on this distribution (overall recall@20 0.916 vs. `txn-ft-hard`'s 0.968).

	Overall		Exact		Legal-form		Partial		No overlap	
System	@20	@30	@20	@30	@20	@30	@20	@30	@20	@30
`tfidf`	0.966	0.972	0.986	0.986	0.989	0.989	0.949	0.961	0.906	0.915
`gleif-ft-hard` *(cross)*	0.916	0.929	0.981	0.982	0.949	0.955	0.873	0.894	0.801	0.833
`txn-ft-hard` *(matched)*	0.968	0.973	0.982	0.983	0.978	0.980	0.958	0.965	0.937	0.946
RRF(`tfidf`, `gleif-ft-hard`) *(cross)*	0.971	0.976	**0.987**	**0.987**	**0.991**	0.991	0.958	0.966	0.953	0.959
RRF(`tfidf`, `txn-ft-hard`) *(matched)*	**0.982***	**0.985***	**0.987**	**0.987**	**0.991**	**0.992**	**0.977***	**0.983***	**0.973**	**0.973**

Table 5.4: Ensemble comparison on the transactions test set (recall@20 / recall@30). Rows after the `tfidf` baseline ordered by increasing overall recall@20. "matched"/"cross" denote the ensemble whose dense component was trained on transactions / on GLEIF. * = ensemble significantly higher than `tfidf` (paired Wilcoxon, Holm-corrected). Dense components were not tested against `tfidf`. (No abbreviation mentions.)

5.2 Analysis

The results in Section 5.1 answer the four research questions, and they build on one another: dense retrieval does not beat the lexical baseline, zero-shot (RQ1) or after fine-tuning (RQ2); the per-category view shows where it does help (RQ3); and that diagnosis motivates fusing a category-specialised dense retriever with the baseline (RQ4). Two things about how we evaluate these results shape the analysis. The first is that we care about recall, not MAP. What matters for the ranking stage is whether the correct entity in the ranked list, not its exact position within it. We point out the cases where recall and MAP disagree, but everything we talk about is a statement about recall. The second is that the categories are not as clean as they look. They are assigned by a rule about surface overlap, not by how hard a mention actually is, a "partial match" can be trivial or genuinely hard, and some "no-overlap" pairs are really acronyms in disguise (e.g., *S.A.S.E.* → *Saudi Arabia Solar Energy*). On top of that, both test sets are mostly easy cases: exact matches and legal-form differences, where everything scores around 0.99. That inflates the overall numbers for every system and hides the differences that matter, which is why we focus on the per-category results rather than the aggregate.

5.2.1 Standalone dense retrieval (RQ1, RQ2)

No dense retriever, zero-shot or fine-tuned, outperforms the `tfidf` baseline on overall recall (RQ1). On both distributions the top of the standalone table is a statistical tie: on GLEIF, `e5-large` (0.882), `txn-ft` (0.881) and `tfidf` (0.878) do not differ significantly at recall@20, and on transactions `tfidf` is highest (0.966) with only `txn-ft` (0.959) reaching it without a significant difference (Tables 5.1, 5.2). The best a dense model achieves is parity, not improvement.

On ranking quality `tfidf` is significantly better than every other model on both distributions. This is worth stating but with the caveat that the fine-tuning objective, and the task framing more broadly, target recall@k and not the ordering within it. A lower MAP is therefore the expected consequence of optimising for recall@k rather than rank, not evidence that dense retrieval ranks poorly where it was trained to.

Fine-tuning helps, but models still don't beat the baseline (RQ2). The one model to match the baseline on recall@20 is `txn-ft`, the transactions-trained fine-tuned model evaluated on the transactions distribution. No fine-tuned model overtakes `tfidf` on either distribution. The `e5-base` is the weakest system almost everywhere, and while the `e5-large` is competitive on GLEIF recall, it falls behind on MAP and on the transactions distribution.

Overall, dense retrievers do not outperform lexical retrieval on this task. Fine-tuning brings it level with the baseline on recall but no further. One thing tempers how far this

can be pushed: the dense models ran under a single, conservative training configuration that was never tuned (Section 6.1). So our take away is that fine-tuned dense retrievers do not beat `tfidf` here, under these conditions, but we cannot rule out that a model fine-tuned with better configuration would close the remaining gap.

5.2.2 Where dense retrieval helps (RQ3)

The category breakdown explains the aggregate tie of RQ1. `tfidf` dominates the easy categories, on exact matches and legal-form differences every system scores near 0.99, and no dense model significantly exceeds `tfidf` on either category on either distribution. This is where the majority of mentions of both test sets sits, and it is why the baseline is so hard to beat in aggregate.

The dense models' advantage is real but slim, and confined to GLEIF. On no-overlap pairs, all three tested dense models significantly exceed `tfidf` (0.177, 0.159, 0.154 vs. 0.126 at recall@20), and on partial matches `txn-ft` significantly exceeds it (0.809 vs. 0.779). On the transactions distribution, no dense model significantly beats `tfidf` in any category, the closest cases are `txn-ft` on partial match (0.943 vs. 0.949) and on no-overlap (0.927 vs. 0.906), but both are not significant.

Two qualifications keep this from being oversold. First, the categories on which dense holds an advantage are categories on which every system is weak. No overlap recall is roughly 0.13–0.18 across all systems on GLEIF. This is also a characteristic of the dataset, the GLEIF mentions are curated company names, so a pair labelled no-overlap could genuinely be a case where the positive match does not share much surface overlap with the mention. Neither a lexical model nor a dense model without any previous context can be expected to perform well on this category. This isn't the case for the transactions dataset as the mentions are from human descriptions of transactions, so many of these mentions have no-overlap cases which still have some surface form overlap with the positive match. Second, abbreviations appear to favour fine-tuned dense models by a wide margin (`gleif-ft` 0.246 vs. `tfidf` 0.070), but with only 88 abbreviation mentions no dense-baseline difference is significant after correction, this result should therefore be interpreted cautiously.

So the two approaches are complementary. `tfidf` wins on the high-overlap categories that make up most of the data, while dense retrievers have a narrow edge, on the low-overlap categories where surface matching breaks down and even there its recall stays low. There are two limitations to this interpretation. First, the categories where dense looks most interesting, abbreviations and transactions no-overlap, are also the smallest, too small for their differences to reach significance, so whatever they suggest stays a suggestion. The second is that a category label does not mean the same thing across the two distributions. As noted above, a no-overlap pair on GLEIF genuinely shares little surface form, whereas

on transactions it often does not so comparing a system's "no-overlap" performance across the two datasets is not really comparing like with like, and the per-category picture is only as clean as the labelling beneath it.

5.2.3 Fusing dense with lexical (RQ4)

The matched ensemble improves overall recall over `tfidf` significantly on both distributions (GLEIF +0.019, 95% CI [0.018,0.020]; transactions +0.017, 95% CI [0.014,0.019]; Tables 5.3, 5.4). The gain comes entirely from the hard categories. Partial matches (+0.075 on GLEIF, +0.028 on transactions) and no-overlap pairs account for the whole overall improvement, while exact and legal-form recall are unchanged. The fusion improves performance on the categories where lexical matching fails without giving up the baseline's near-perfect performance on the easy ones. This is by design as the dense component was trained only on partial-match, no-overlap and abbreviation mentions precisely to cover the baseline's weak spots. So the result supports the claim that a purpose-built dense specialist fused with `tfidf` using RRF improves recall.

There are two caveats to this. It is a recall result only, MAP was not computed for the ensembles (Section 6.1), so we can show that fusion improves shortlist coverage but not that it preserves ranking quality. And the transactions no-overlap gain, large as its point estimate is (+0.067), is not significant (n=329), so on that distribution the complement rests on partial matches alone.

Chapter 6

Conclusions

In this thesis we looked to answer whether dense retrievers are useful for entity ranking in a low-resource setting and, if so, how. We find that dense retrievers do not replace a lexical `tfidf` baseline in this setting. Pretrained dense retrievers do not beat it, and fine-tuning them for the task only closes the gap but does not push past it. However, dense retrievers are useful in a narrower role, a dense model built to specialise on the categories the baseline misses, fused with it, improves recall.

No zero-shot dense retriever we evaluated outperforms the `tfidf` baseline: the larger model reaches parity on GLEIF recall while the base model is weaker, and both fall short on transactions and on ranking quality (RQ1). Task-specific fine-tuning improves a dense retriever significantly over its zero-shot starting point, but the improvement is comparable to using a larger pretrained model and in no case reaches the baseline (RQ2).

The categories on which dense retrieval helps are the low-overlap ones (partial matches and no-overlap pairs) where lexical similarity is weakest and significant only on GLEIF; the baseline keeps its firm, broad advantage on the high-overlap exact and legal-form categories that dominate both datasets (RQ3). This is caveated by the category definitions themselves, which are surface-rule proxies for difficulty and are not comparable across the two distributions.

Fusing a category-specialised dense retriever with `tfidf` via reciprocal rank fusion significantly improves recall on both distributions (+0.019 on GLEIF, +0.017 on transactions), and the gain comes entirely from the hard categories, without cost to the easy ones (RQ4). This is a recall result; it holds for shortlist coverage, and the dense component must be strong on the target distribution for the fusion to help.

The contributions of this work are threefold. The first is a labelled entity-ranking dataset constructed from the public GLEIF registry and ING's client data: a set of (mention, entity) pairs annotated with five categories of surface variation and partitioned by a tree-disjoint scheme that prevents leakage between splits. The construction is fully specified and reproducible from the public registry, though the dataset itself cannot be released because the client-registry side is proprietary.

The other two are diagnostic. Evaluating on this dataset and on a separate transactions distribution, we give a significance-tested characterisation showing that zero-shot dense

retrieval does not replace lexical retrieval for low-context company-name ranking, and we locate where each approach succeeds and fails. Following from that diagnosis, we show that dense retriever's can be used as a targeted complement, a hard-category specialist fused with to `tfidf` to recover recall on precisely the cases the baseline misses.

6.1 Limitations

Several limitations caveat the findings above, and motivate the future directions (Section 6.2)

Abbreviation and no-overlap mention count Two of the categories most relevant to the dense models are also the smallest. Abbreviations number only 88 mentions on GLEIF and are effectively absent from transactions (a single pair), and transactions no-overlap has 329 mentions. In both, the point estimates favouring dense retrieval are large but never reach significance after correction. Claims that rest on these categories are therefore suggestive rather than established.

Category definitions The five categories are assigned by a rule over surface overlap, not by intrinsic difficulty, so they are proxies: a partial match may be trivial or hard, and some no-overlap pairs are effectively acronym expansions. The same label also does not mean the same thing across the two distributions, as noted above. Every per-category conclusion is therefore a conclusion as categorised, and cross-distribution comparisons of a single category are not like-for-like.

Training configuration The reported models reflect a single, conservative training configuration rather than a tuned one. Early in the work we explored mining hard negatives from `tfidf`, which did not train effectively in our low-context setting, consistent with the mixed evidence on lexically-mined negatives in the literature (Section 2.4). So, we moved to mining negatives from the dense model instead and to enforcing tree-disjoint splitting. By the time the final configuration was fixed we were able to run only a single training run under safe hyperparameters, without a hyperparameter sweep or a multi-seed variance estimate. This constraint biases against the dense models relative to the already-settled baseline, so the RQ1 result should be read as conservative; but it also means small standalone differences may reflect run-to-run variation, and we cannot rule out that a tuned configuration would narrow the gap or overcome it.

Significance testing Two features of the testing protocol caveat how the significance results should be compared. The standalone and ensemble comparisons apply the Holm correction over families of different sizes, so a significant result in one table survived a

harsher correction than one in the other and the two cannot be compared on strength directly. The reported bootstrap confidence intervals are uncorrected and can therefore disagree with the Holm-adjusted tests at the margin, in which case we treat the corrected test as authoritative.

6.2 Future Directions

The most immediate next step would be a hyperparameter sweep together with multiple training seeds. This would establish whether a tuned dense model narrows/overcomes the gap to `tfidf` model, and would replace the current point estimates with confidence intervals over training runs, settling which of the small standalone differences are real.

Extending the study to other registries and, most importantly, evaluating the ranking stage end-to-end with a downstream disambiguation model would test whether the recall gains reported here translate into better final entity linking, a question that ultimately matters for the application.

Bibliography

[1] Krisztian Balog. *Entity-Oriented Search*. Vol. 39. The Information Retrieval Series. Springer, 2018. ISBN: 978-3-319-93933-9. DOI: 10.1007/978-3-319-93935-3. URL: https://eos-book.org.

[2] Jeffrey Dalton, Laura Dietz, and James Allan. "Entity query feature expansion using knowledge base links". In: *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*. SIGIR '14. Gold Coast, Queensland, Australia: Association for Computing Machinery, 2014, pp. 365–374. ISBN: 9781450322577. DOI: 10.1145/2600428.2609628. URL: https://doi.org/10.1145/2600428.2609628.

[3] Emma Gerritse, Faegheh Hasibi, and Arjen de Vries. "Graph Embeddings to Empower Entity Retrieval". In: *Information Retrieval Research* 1 (2025), pp. 137–165. DOI: 10.54195/irrj.19877. URL: https://irrj.org/article/view/19877.

[4] Faegheh Hasibi, Krisztian Balog, and Svein Erik Bratsberg. "Exploiting Entity Linking in Queries for Entity Retrieval". In: *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval*. ICTIR '16. Newark, Delaware, USA: Association for Computing Machinery, 2016, pp. 209–218. ISBN: 9781450344975. DOI: 10.1145/2970398.2970406. URL: https://doi.org/10.1145/2970398.2970406.

[5] Darío Garigliotti, Faegheh Hasibi, and Krisztian Balog. "Target Type Identification for Entity-Bearing Queries". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '17. Shinjuku, Tokyo, Japan: Association for Computing Machinery, 2017, pp. 845–848. ISBN: 9781450350228. DOI: 10.1145/3077136.3080659. URL: https://doi.org/10.1145/3077136.3080659.

[6] Max Baak. *Matching company names at scale!* Medium. 2024. URL: https://medium.com/wbaa/matching-company-names-at-scale-af20429a80c7.

[7] Johannes M. van Hulst et al. "REL: An Entity Linker Standing on the Shoulders of Giants". In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '20. ACM, July 2020, pp. 2197–2200. DOI: 10.1145/3397271.3401416. URL: http://dx.doi.org/10.1145/3397271.3401416.

[8] Vladimir Karpukhin et al. “Dense Passage Retrieval for Open-Domain Question Answering”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Bonnie Webber et al. Online: Association for Computational Linguistics, Nov. 2020, pp. 6769–6781. DOI: 10.18653/v1/2020.emnlp-main.550. URL: https://aclanthology.org/2020.emnlp-main.550/.

[9] Liang Wang et al. “Text Embeddings by Weakly-Supervised Contrastive Pre-training”. In: *ArXiv* abs/2212.03533 (2022). URL: https://api.semanticscholar.org/CorpusID:254366618.

[10] Liang Wang et al. “Multilingual E5 Text Embeddings: A Technical Report”. In: *ArXiv* abs/2402.05672 (2024). URL: https://api.semanticscholar.org/CorpusID:267547683.

[11] Jacob Devlin et al. “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, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423/.

[12] Ivan P. Fellegi and Alan B. Sunter. “A Theory for Record Linkage”. In: *Journal of the American Statistical Association* 64.328 (1969), pp. 1183–1210. DOI: 10.1080/01621459.1969.10501049. eprint: https://www.tandfonline.com/doi/pdf/10.1080/01621459.1969.10501049. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1969.10501049.

[13] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. “A comparison of string distance metrics for name-matching tasks”. In: *Proceedings of the 2003 International Conference on Information Integration on the Web*. IIWEB’03. Acapulco, Mexico: AAAI Press, 2003, pp. 73–78.

[14] Peter Christen. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer Publishing Company, Incorporated, 2012. ISBN: 3642311636.

[15] Hanna Köpcke and Erhard Rahm. “Frameworks for entity matching: A comparison”. In: 69.2 (2010). ISSN: 0169-023X. DOI: 10.1016/j.datak.2009.10.003. URL: https://doi.org/10.1016/j.datak.2009.10.003.

[16] Sidharth Mudgal et al. “Deep Learning for Entity Matching: A Design Space Exploration”. In: New York, NY, USA: Association for Computing Machinery, 2018. ISBN: 9781450347037. DOI: 10.1145/3183713.3196926. URL: https://doi.org/10.1145/3183713.3196926.

[17] Yuliang Li et al. “Deep entity matching with pre-trained language models”. In: *Proceedings of the VLDB Endowment* 14.1 (2020). ISSN: 2150-8097. DOI: 10.14778/3421424.3421431. URL: http://dx.doi.org/10.14778/3421424.3421431.

[18] Ursin Brunner and Kurt Stockinger. “Entity Matching with Transformer Architectures - A Step Forward in Data Integration”. In: *International Conference on Extending Database Technology*. 2020. URL: https://api.semanticscholar.org/CorpusID:214613452.

[19] Matteo Paganelli et al. “Analyzing how BERT performs entity matching”. In: *Proceedings of the VLDB Endowment* 15 (Apr. 2022), pp. 1726–1738. DOI: 10.14778/3529337.3529356.

[20] Ledell Wu et al. “Scalable Zero-shot Entity Linking with Dense Entity Retrieval”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Bonnie Webber et al. Online: Association for Computational Linguistics, Nov. 2020, pp. 6397–6407. DOI: 10.18653/v1/2020.emnlp-main.519. URL: https://aclanthology.org/2020.emnlp-main.519/.

[21] Nandan Thakur et al. *BEIR: A Heterogenous Benchmark for Zero-shot Evaluation of Information Retrieval Models*. Apr. 2021. DOI: 10.48550/arXiv.2104.08663.

[22] Gautier Izacard et al. *Unsupervised Dense Information Retrieval with Contrastive Learning*. 2022. arXiv: 2112.09118 [cs.IR]. URL: https://arxiv.org/abs/2112.09118.

[23] Luyu Gao and Jamie Callan. “Unsupervised Corpus Aware Language Model Pre-training for Dense Passage Retrieval”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Smaranda Muresan, Preslav Nakov, and Aline Villavicencio. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 2843–2853. DOI: 10.18653/v1/2022.acl-long.203. URL: https://aclanthology.org/2022.acl-long.203/.

[24] Kexin Wang et al. “GPL: Generative Pseudo Labeling for Unsupervised Domain Adaptation of Dense Retrieval”. In: Jan. 2022, pp. 2345–2360. DOI: 10.18653/v1/2022.naacl-main.168.

[25] Luiz Bonifacio et al. “InPars: Unsupervised Dataset Generation for Information Retrieval”. In: SIGIR ’22. Madrid, Spain: Association for Computing Machinery, 2022. ISBN: 9781450387323. DOI: 10.1145/3477495.3531863. URL: https://doi.org/10.1145/3477495.3531863.

[26] Zhuyun Dai et al. *Promptagator: Few-shot Dense Retrieval From 8 Examples*. 2022. arXiv: 2209.11755 [cs.CL]. URL: https://arxiv.org/abs/2209.11755.

[27] Belinda Z. Li et al. “Efficient One-Pass End-to-End Entity Linking for Questions”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Bonnie Webber et al. Online: Association for Computational Linguistics, Nov. 2020, pp. 6433–6441. DOI: 10.18653/v1/2020.emnlp-main.522. URL: https://aclanthology.org/2020.emnlp-main.522/.

[28] Gizem Aydin et al. “Find the Funding: Entity Linking with Incomplete Funding Knowledge Bases”. In: *Proceedings of the 29th International Conference on Computational Linguistics*. Gyeongju, Republic of Korea: International Committee on Computational Linguistics, Oct. 2022, pp. 1937–1942. URL: https://aclanthology.org/2022.coling-1.168/.

[29] Mohanna Hoveyda et al. “Real World Conversational Entity Linking Requires More Than Zero-Shots”. In: *Findings of the Association for Computational Linguistics: ACL 2024*. Ed. by Lun-Wei Ku, Andre Martins, and Vivek Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 13938–13946. DOI: 10.18653/v1/2024.findings-acl.829. URL: https://aclanthology.org/2024.findings-acl.829/.

[30] Faegheh Hasibi, Krisztian Balog, and Svein Erik Bratsberg. “Entity Linking in Queries: Tasks and Evaluation”. In: *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*. ICTIR ’15. Northampton, Massachusetts, USA: Association for Computing Machinery, 2015, pp. 171–180. ISBN: 9781450338332. DOI: 10.1145/2808194.2809473. URL: https://doi.org/10.1145/2808194.2809473.

[31] Roi Blanco, Giuseppe Ottaviano, and Edgar Meij. “Fast and Space-Efficient Entity Linking for Queries”. In: *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*. WSDM ’15. Shanghai, China: Association for Computing Machinery, 2015, pp. 179–188. ISBN: 9781450333177. DOI: 10.1145/2684822.2685317. URL: https://doi.org/10.1145/2684822.2685317.

[32] Faegheh Hasibi, Krisztian Balog, and Svein Erik Bratsberg. “Entity Linking in Queries: Efficiency vs. Effectiveness”. In: *Advances in Information Retrieval*. Ed. by Joemon M Jose et al. Cham: Springer International Publishing, 2017, pp. 40–53. ISBN: 978-3-319-56608-5.

[33] Özge Sevgili et al. “Neural entity linking: A survey of models based on deep learning”. In: *Semantic Web* 13.3 (Apr. 2022). Ed. by Mehwish Alam et al., pp. 527–570. ISSN: 1570-0844. DOI: 10.3233/sw-222986. URL: http://dx.doi.org/10.3233/SW-222986.

[34] Karen Sparck Jones. “A statistical interpretation of term specificity and its application in retrieval”. In: *Document Retrieval Systems*. GBR: Taylor Graham Publishing, 1988, pp. 132–142. ISBN: 0947568212.

[35] S. E. Robertson and S. Walker. “Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval”. In: *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’94. Dublin, Ireland: Springer-Verlag, 1994, pp. 232–241. ISBN: 038719889X.

[36] Justin Zobel and Alistair Moffat. “Inverted files for text search engines”. In: *ACM Comput. Surv.* 38.2 (July 2006), 6–es. ISSN: 0360-0300. DOI: 10.1145/1132956.1132959. URL: https://doi.org/10.1145/1132956.1132959.

[37] Andrew Yates, Rodrigo Nogueira, and Jimmy Lin. “Pretrained Transformers for Text Ranking: BERT and Beyond”. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Tutorials*. Ed. by Greg Kondrak, Kalina Bontcheva, and Dan Gillick. Online: Association for Computational Linguistics, June 2021, pp. 1–4. DOI: 10.18653/v1/2021.naacl-tutorials.1. URL: https://aclanthology.org/2021.naacl-tutorials.1/.

[38] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. “SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking”. In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’21. Virtual Event, Canada: Association for Computing Machinery, 2021, pp. 2288–2292. ISBN: 9781450380379. DOI: 10.1145/3404835.3463098. URL: https://doi.org/10.1145/3404835.3463098.

[39] Nils Reimers and Iryna Gurevych. “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Ed. by Kentaro Inui et al. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3982–3992. DOI: 10.18653/v1/D19-1410. URL: https://aclanthology.org/D19-1410/.

[40] Niklas Muennighoff et al. “MTEB: Massive Text Embedding Benchmark”. In: *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*. Ed. by Andreas Vlachos and Isabelle Augenstein. Dubrovnik, Croatia: Association for Computational Linguistics, May 2023, pp. 2014–2037. DOI: 10.18653/v1/2023.eacl-main.148. URL: https://aclanthology.org/2023.eacl-main.148/.

[41] Alexis Conneau et al. “Unsupervised Cross-lingual Representation Learning at Scale”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Ed. by Dan Jurafsky et al. Online: Association for Computational Linguistics, July 2020, pp. 8440–8451. DOI: 10.18653/v1/2020.acl-main.747. URL: https://aclanthology.org/2020.acl-main.747/.

[42] Neil Houlsby et al. “Parameter-Efficient Transfer Learning for NLP”. In: *International Conference on Machine Learning*. 2019. URL: https://api.semanticscholar.org/CorpusID:59599816.

[43] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. arXiv: 2106.09685 [cs.CL]. URL: https://arxiv.org/abs/2106.09685.

[44] Lee Xiong et al. *Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval*. July 2020. DOI: 10.48550/arXiv.2007.00808.

[45] Yingqi Qu et al. “RocketQA: An Optimized Training Approach to Dense Passage Retrieval for Open-Domain Question Answering”. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Ed. by Kristina Toutanova et al. Online: Association for Computational Linguistics, June 2021, pp. 5835–5847. DOI: 10.18653/v1/2021.naacl-main.466. URL: https://aclanthology.org/2021.naacl-main.466/.

[46] Gordon V. Cormack, Charles L A Clarke, and Stefan Buettcher. “Reciprocal rank fusion outperforms condorcet and individual rank learning methods”. In: *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’09. Boston, MA, USA: Association for Computing Machinery, 2009, pp. 758–759. ISBN: 9781605584836. DOI: 10.1145/1571941.1572114. URL: https://doi.org/10.1145/1571941.1572114.

[47] Paolo Ferragina and Ugo Scaiella. *Fast and Accurate Annotation of Short Texts with Wikipedia Pages*. Washington, DC, USA, Jan. 2012. DOI: 10.1109/MS.2011.122. URL: https://doi.org/10.1109/MS.2011.122.

[48] Alexandros Zeakis et al. “Pre-Trained Embeddings for Entity Resolution: An Experimental Analysis”. In: *Proc. VLDB Endow.* 16.9 (May 2023), pp. 2225–2238. ISSN: 2150-8097. DOI: 10.14778/3598581.3598594. URL: https://doi.org/10.14778/3598581.3598594.

[49] Marco Cornolti, Paolo Ferragina, and Massimiliano Ciaramita. “A framework for benchmarking entity-annotation systems”. In: *Proceedings of the 22nd International Conference on World Wide Web*. WWW ’13. Rio de Janeiro, Brazil: Association for Computing Machinery, 2013, pp. 249–260. ISBN: 9781450320351. DOI: 10.1145/2488388.2488411. URL: https://doi.org/10.1145/2488388.2488411.

[50] Jan-Philipp Awick and Jorge Marx Gómez. “Entity Matching in the Era of Language Models: A Structured Literature Review”. In: *2025 25th International Conference on Control Systems and Computer Science (CSCS)*. 2025, pp. 368–375. DOI: 10.1109/CSCS66924.2025.00061.

Appendix A

Legal entity form list

Table A.1 lists the legal-entity form suffixes used to identify the *only legal-form difference* category (Section 3.2.2), grouped by the country they are associated with. Matching is case-insensitive and punctuation-insensitive.

Table A.1: Legal-entity form suffixes by country.

Country	Legal-entity forms
Australia	`pty`
Austria	`oeg`
Belgium	`asbl`
Brazil	`ltda`
Bulgaria	`ead`, `ood`
Canada	`ulc`
Czech Republic	`sro`
Denmark	`aps`, `a/s`
EU	`se`, `eeig`
Finland	`oy`
France	`sa`, `sarl`, `sas`, `scp`, `snc`, `eurl`, `cie`
Germany	`gmbh`, `ag`, `kgaa`, `komanditgesellschaft`, `gbr`
Hungary	`kft`, `zrt`
Iceland	`ehf`
Indonesia	`tmi`
Italy	`spa`, `srl`, `srls`
Japan	`kk`
Kazakhstan	`kkt`
Luxembourg	`scs`, `sca`
Malaysia	`sdn bhd`, `berhad`
Netherlands	`bv`, `nv`, `cv`
Norway	`as`
Poland	`sp z oo`
Russia	`ooo`, `jsc`
Serbia	`ado`
Singapore	`pte ltd`
Spain	`sl`
Sweden	`ab`, `komanditbolag`
UK	`ltd`, `limited`, `plc`, `llp`
US	`llc`, `inc`, `incorporated`, `corp`, `corporation`, `co`, `lp`, `lllp`
Unknown	`oab`, `tld`

Appendix B

Full hyperparameter table

Table B.1: Fine-tuning hyperparameters. The same configuration is used for the GLEIF-based and the transactions-based fine-tunes.

Hyperparameter	Value
Base model	E5-multilingual-base (110M params)
Epochs	2
Batch size	128
Optimiser	AdamW
Learning rate	1×10^{-5}
Warmup ratio	0.1
Weight decay	0.01
Gradient clipping (max norm)	1.0
Loss temperature τ	0.1
Max sequence length	32 tokens
LoRA rank r	16
LoRA scaling α	32
LoRA target modules	`q_proj`, `v_proj`
LoRA dropout	0.05
Hard negatives per mention (cap)	5
Base-model retrieval depth K	20

Appendix C

Additional Evaluation Results

This appendix reports every metric computed by the evaluation pipeline for the standalone comparison, extending the summary tables of Section 5.1.

Category	Model	MAP	MRR	R@1	R@5	R@10	R@20
Overall	`tfidf`	0.8411	0.842	0.8237	0.8603	0.8699	0.8782
	`e5-base`	0.7728	0.7738	0.7316	0.8236	0.8435	0.8595
	`e5-large`	0.8216	0.8225	0.791	0.858	0.8716	0.8824
	`gleif-ft`	0.8314	0.8324	0.8108	0.8555	0.8652	0.8737
	`txn-ft`	0.8101	0.8111	0.7739	0.8543	0.8698	0.8814
Exact	`tfidf`	0.9955	0.9965	0.9911	0.999	0.9991	0.9992
	`e5-base`	0.9578	0.9591	0.9369	0.9828	0.9888	0.9922
	`e5-large`	0.9838	0.9849	0.973	0.9954	0.9969	0.9978
	`gleif-ft`	0.9834	0.9845	0.9733	0.9945	0.9963	0.9973
	`txn-ft`	0.9686	0.9698	0.9503	0.9896	0.9935	0.9956
Partial	`tfidf`	0.6326	0.6346	0.5763	0.6972	0.7412	0.7785
	`e5-base`	0.5662	0.568	0.4966	0.6516	0.6984	0.7414
	`e5-large`	0.6282	0.63	0.5617	0.7076	0.7517	0.7899
	`gleif-ft`	0.6327	0.6347	0.5833	0.6913	0.7229	0.7536
	`txn-ft`	0.6396	0.6414	0.5677	0.7286	0.7739	0.8093
Legal-form	`tfidf`	0.9762	0.9782	0.9577	0.9952	0.9965	0.9974
	`e5-base`	0.7929	0.7952	0.7137	0.8924	0.9266	0.9494
	`e5-large`	0.8987	0.9008	0.844	0.9652	0.9801	0.9876
	`gleif-ft`	0.9501	0.952	0.9243	0.9786	0.9861	0.9901
	`txn-ft`	0.8743	0.8764	0.8151	0.947	0.9645	0.9774
Abbrev.	`tfidf`	0.0546	0.108	0.0426	0.0701	0.0701	0.0701
	`e5-base`	0.0726	0.1263	0.0521	0.0843	0.1184	0.1439
	`e5-large`	0.0936	0.146	0.0691	0.1155	0.1496	0.161
	`gleif-ft`	0.1437	0.1985	0.1032	0.1723	0.2235	0.2462
	`txn-ft`	0.0842	0.1384	0.0521	0.0956	0.1553	0.1667
No overlap	`tfidf`	0.0873	0.0925	0.0729	0.1034	0.114	0.1263
	`e5-base`	0.087	0.0912	0.0695	0.1076	0.1218	0.1397
	`e5-large`	0.1158	0.1204	0.0945	0.14	0.1585	0.1771
	`gleif-ft`	0.1058	0.1108	0.09	0.1235	0.1379	0.1536
	`txn-ft`	0.106	0.1104	0.0871	0.1285	0.1443	0.1592

Table C.1: Complete standalone metrics on the GLEIF test set, by category.

Category	Model	MAP	MRR	R@1	R@5	R@10	R@20
Overall	tfidf	0.8971	0.8985	0.8597	0.94	0.9534	0.9657
	e5-base	0.7951	0.7968	0.7462	0.8529	0.8825	0.9073
	e5-large	0.845	0.8466	0.795	0.9064	0.9325	0.9489
	gleif-ft	0.8543	0.8561	0.8161	0.8997	0.9193	0.937
	txn-ft	0.8564	0.8577	0.8065	0.9197	0.9428	0.9593
Exact	tfidf	0.9845	0.9869	0.9811	0.9853	0.9858	0.9858
	e5-base	0.9359	0.9386	0.9167	0.9565	0.9653	0.9703
	e5-large	0.9739	0.9762	0.9644	0.9812	0.9834	0.9844
	gleif-ft	0.9709	0.9731	0.9603	0.9797	0.9816	0.9846
	txn-ft	0.9496	0.9514	0.9312	0.9693	0.9768	0.9802
Partial	tfidf	0.8284	0.831	0.7637	0.9046	0.9261	0.9492
	e5-base	0.6974	0.7004	0.6302	0.7759	0.8219	0.8595
	e5-large	0.7569	0.7596	0.6814	0.8494	0.8944	0.9227
	gleif-ft	0.7799	0.7832	0.7242	0.8458	0.8762	0.9032
	txn-ft	0.7914	0.7941	0.7199	0.8822	0.9174	0.9433
Legal-form	tfidf	0.9766	0.9804	0.9555	0.9876	0.989	0.989
	e5-base	0.8563	0.8606	0.7983	0.9251	0.9414	0.9572
	e5-large	0.9029	0.9063	0.8502	0.963	0.9676	0.9753
	gleif-ft	0.8847	0.8885	0.8389	0.9367	0.9566	0.9657
	txn-ft	0.9064	0.9094	0.8585	0.9629	0.97	0.9764
No overlap	tfidf	0.7561	0.7564	0.7006	0.8212	0.8951	0.9058
	e5-base	0.633	0.6335	0.5684	0.7082	0.7568	0.8191
	e5-large	0.6679	0.6694	0.5699	0.8024	0.8739	0.8967
	gleif-ft	0.6824	0.6825	0.6261	0.7599	0.7994	0.8617
	txn-ft	0.7153	0.7155	0.6322	0.8222	0.8845	0.9271

Table C.2: Complete standalone metrics on the transactions test set, by category.

Split	Unique mentions	(mention, positive) pairs	Unique positive clients	Unique trees
GLEIF train–val	142,647	143,243	129,393	112,685
GLEIF test	102,747	103,367	89,985	77,654
Txn train–val	38,633	39,042	4,701	3,059
Txn test	10,835	10,979	1,264	772

Table C.3: Summary statistics of the dataset splits.

Appendix D

Full significance tables

The complete significance tables referenced in Section 5.1, for both reference systems. All tests are paired Wilcoxon signed-rank with Holm correction (Section 3.6).

D.1 Standalone comparison

Table D.1: Standalone significance on GLEIF, each model versus `tfidf` (paired Wilcoxon signed-rank, Holm-corrected). Diff is mean(model)−mean(reference); ✓ marks $p_{\text{adj}} < 0.05$.

Category	Model	Metric	Diff	95% CI	Sig.
Overall	`e5-large`	R@10	0.0017	[0.0004, 0.003]	–
	`e5-large`	R@20	0.0042	[0.003, 0.0054]	–
	`e5-large`	MAP	-0.0196	[−0.021, −0.0182]	✓
	`gleif-ft`	R@10	-0.0047	[−0.0062, −0.0033]	✓
	`gleif-ft`	R@20	-0.0045	[−0.0059, −0.0031]	–
	`gleif-ft`	MAP	-0.0097	[−0.0111, −0.0083]	✓
	`txn-ft`	R@10	-0.0001	[−0.0015, 0.0012]	–
	`txn-ft`	R@20	0.0031	[0.0018, 0.0044]	–
	`txn-ft`	MAP	-0.031	[−0.0325, −0.0296]	✓
Exact	`e5-large`	R@10	-0.0023	[−0.0027, −0.0019]	–
	`e5-large`	R@20	-0.0014	[−0.0018, −0.0011]	–
	`e5-large`	MAP	-0.0117	[−0.0125, −0.0108]	✓
	`gleif-ft`	R@10	-0.0028	[−0.0033, −0.0024]	–
	`gleif-ft`	R@20	-0.002	[−0.0024, −0.0016]	–
	`gleif-ft`	MAP	-0.0121	[−0.013, −0.0112]	✓
	`txn-ft`	R@10	-0.0056	[−0.0062, −0.005]	–
	`txn-ft`	R@20	-0.0037	[−0.0042, −0.0032]	–
	`txn-ft`	MAP	-0.0269	[−0.0282, −0.0257]	✓
Partial	`e5-large`	R@10	0.0104	[0.005, 0.016]	–
	`e5-large`	R@20	0.0114	[0.0059, 0.0167]	–
	`e5-large`	MAP	-0.0044	[−0.0098, 0.0009]	–
	`gleif-ft`	R@10	-0.0184	[−0.0251, −0.0119]	✓

continued on next page

continued from previous page

Category	Model	Metric	Diff	95% CI	Sig.
	`gleif-ft`	R@20	-0.025	$[-0.0314, -0.0188]$	✓
	`gleif-ft`	MAP	0.0001	$[-0.0058, 0.0061]$	✓
	`txn-ft`	R@10	0.0326	$[0.027, 0.0384]$	✓
	`txn-ft`	R@20	0.0307	$[0.0252, 0.0364]$	✓
	`txn-ft`	MAP	0.007	$[0.0017, 0.0121]$	–
Legal-form	`e5-large`	R@10	-0.0165	$[-0.0184, -0.0146]$	✓
	`e5-large`	R@20	-0.0097	$[-0.0111, -0.0083]$	–
	`e5-large`	MAP	-0.0775	$[-0.0808, -0.0742]$	✓
	`gleif-ft`	R@10	-0.0105	$[-0.012, -0.0089]$	✓
	`gleif-ft`	R@20	-0.0072	$[-0.0085, -0.0059]$	–
	`gleif-ft`	MAP	-0.0261	$[-0.0287, -0.0236]$	✓
	`txn-ft`	R@10	-0.032	$[-0.0345, -0.0296]$	✓
	`txn-ft`	R@20	-0.02	$[-0.022, -0.018]$	✓
	`txn-ft`	MAP	-0.1019	$[-0.1055, -0.0982]$	✓
Abbrev.	`e5-large`	R@10	0.0795	$[0.0227, 0.142]$	–
	`e5-large`	R@20	0.0909	$[0.0284, 0.1591]$	–
	`e5-large`	MAP	0.039	$[0.0071, 0.0775]$	–
	`gleif-ft`	R@10	0.1534	$[0.0795, 0.233]$	–
	`gleif-ft`	R@20	0.1761	$[0.1021, 0.2557]$	–
	`gleif-ft`	MAP	0.0891	$[0.0373, 0.1473]$	–
	`txn-ft`	R@10	0.0852	$[0.0227, 0.1477]$	–
	`txn-ft`	R@20	0.0966	$[0.0341, 0.1648]$	–
	`txn-ft`	MAP	0.0295	$[0.0037, 0.0612]$	–
No overlap	`e5-large`	R@10	0.0445	$[0.0382, 0.0508]$	✓
	`e5-large`	R@20	0.0508	$[0.0444, 0.0573]$	✓
	`e5-large`	MAP	0.0284	$[0.0234, 0.0335]$	✓
	`gleif-ft`	R@10	0.0239	$[0.0181, 0.0296]$	✓
	`gleif-ft`	R@20	0.0273	$[0.0213, 0.0333]$	✓
	`gleif-ft`	MAP	0.0185	$[0.0138, 0.023]$	✓
	`txn-ft`	R@10	0.0302	$[0.0244, 0.0361]$	✓
	`txn-ft`	R@20	0.0329	$[0.0269, 0.039]$	✓
	`txn-ft`	MAP	0.0187	$[0.0139, 0.0234]$	✓

Table D.2: Standalone significance on GLEIF, each model versus `e5-base` (paired Wilcoxon signed-rank, Holm-corrected). Diff is mean(model)−mean(reference); ✓ marks $p_{\text{adj}} < 0.05$.

Category	Model	Metric	Diff	95% CI	Sig.
Overall	`e5-large`	R@10	0.0281	$[0.0268, 0.0293]$	✓
	`e5-large`	R@20	0.0229	$[0.0218, 0.0241]$	✓
	`e5-large`	MAP	0.0488	$[0.0475, 0.0501]$	✓

continued on next page

continued from previous page

Category	Model	Metric	Diff	95% CI	Sig.
	gleif-ft	R@10	0.0217	[0.0202, 0.0231]	✓
	gleif-ft	R@20	0.0142	[0.0128, 0.0156]	✓
	gleif-ft	MAP	0.0587	[0.0571, 0.0602]	✓
	txn-ft	R@10	0.0263	[0.0252, 0.0273]	✓
	txn-ft	R@20	0.0219	[0.0209, 0.0229]	✓
	txn-ft	MAP	0.0373	[0.0363, 0.0384]	✓
Exact	e5-large	R@10	0.008	[0.0072, 0.0089]	✓
	e5-large	R@20	0.0056	[0.005, 0.0063]	–
	e5-large	MAP	0.026	[0.0247, 0.0272]	✓
	gleif-ft	R@10	0.0075	[0.0067, 0.0083]	✓
	gleif-ft	R@20	0.0051	[0.0044, 0.0057]	–
	gleif-ft	MAP	0.0256	[0.0243, 0.0268]	✓
	txn-ft	R@10	0.0047	[0.0041, 0.0053]	–
	txn-ft	R@20	0.0034	[0.0029, 0.0039]	–
	txn-ft	MAP	0.0107	[0.0099, 0.0116]	✓
Partial	e5-large	R@10	0.0533	[0.0488, 0.0578]	✓
	e5-large	R@20	0.0485	[0.0441, 0.0527]	✓
	e5-large	MAP	0.062	[0.0579, 0.0663]	✓
	gleif-ft	R@10	0.0245	[0.0184, 0.0304]	✓
	gleif-ft	R@20	0.0121	[0.0063, 0.0179]	–
	gleif-ft	MAP	0.0666	[0.0609, 0.0721]	✓
	txn-ft	R@10	0.0755	[0.0715, 0.0796]	✓
	txn-ft	R@20	0.0678	[0.0639, 0.0717]	✓
	txn-ft	MAP	0.0734	[0.0701, 0.0767]	✓
Legal-form	e5-large	R@10	0.0534	[0.0502, 0.0567]	✓
	e5-large	R@20	0.0383	[0.0355, 0.041]	✓
	e5-large	MAP	0.1058	[0.102, 0.1097]	✓
	gleif-ft	R@10	0.0594	[0.0561, 0.0628]	✓
	gleif-ft	R@20	0.0408	[0.038, 0.0437]	✓
	gleif-ft	MAP	0.1572	[0.153, 0.1615]	✓
	txn-ft	R@10	0.0379	[0.0352, 0.0406]	✓
	txn-ft	R@20	0.028	[0.0257, 0.0304]	✓
	txn-ft	MAP	0.0814	[0.0784, 0.0845]	✓
Abbrev.	e5-large	R@10	0.0313	[−0.017, 0.0852]	–
	e5-large	R@20	0.017	[−0.0284, 0.0682]	–
	e5-large	MAP	0.021	[−0.0018, 0.051]	–
	gleif-ft	R@10	0.1051	[0.0369, 0.179]	–
	gleif-ft	R@20	0.1023	[0.0455, 0.1705]	–
	gleif-ft	MAP	0.0711	[0.0277, 0.1225]	–
	txn-ft	R@10	0.0369	[0.0028, 0.0795]	–
	txn-ft	R@20	0.0227	[0, 0.0568]	–

continued on next page

continued from previous page

Category	Model	Metric	Diff	95% CI	Sig.
	`txn-ft`	MAP	0.0116	[0.003, 0.0228]	–
No overlap	`e5-large`	R@10	0.0367	[0.0319, 0.0418]	✓
	`e5-large`	R@20	0.0374	[0.0325, 0.0424]	✓
	`e5-large`	MAP	0.0287	[0.0251, 0.0326]	✓
	`gleif-ft`	R@10	0.0161	[0.0105, 0.0217]	✓
	`gleif-ft`	R@20	0.0139	[0.0082, 0.0195]	–
	`gleif-ft`	MAP	0.0188	[0.0145, 0.0232]	✓
	`txn-ft`	R@10	0.0224	[0.0188, 0.0262]	✓
	`txn-ft`	R@20	0.0195	[0.0159, 0.0233]	✓
	`txn-ft`	MAP	0.019	[0.0164, 0.0217]	✓

Table D.3: Standalone significance on transactions, each model versus `tfidf` (paired Wilcoxon signed-rank, Holm-corrected). Diff is mean(model)−mean(reference); ✓ marks $p_{\text{adj}} < 0.05$.

Category	Model	Metric	Diff	95% CI	Sig.
Overall	`e5-large`	R@10	-0.0208	[−0.0254, −0.0162]	✓
	`e5-large`	R@20	-0.0168	[−0.021, −0.0127]	✓
	`e5-large`	MAP	-0.0521	[−0.0571, −0.0474]	✓
	`gleif-ft`	R@10	-0.034	[−0.039, −0.0292]	✓
	`gleif-ft`	R@20	-0.0287	[−0.0332, −0.0243]	✓
	`gleif-ft`	MAP	-0.0428	[−0.048, −0.0379]	✓
	`txn-ft`	R@10	-0.0106	[−0.0151, −0.0063]	–
	`txn-ft`	R@20	-0.0064	[−0.0104, −0.0026]	–
	`txn-ft`	MAP	-0.0408	[−0.0457, −0.0359]	✓
Exact	`e5-large`	R@10	-0.0024	[−0.0046, −0.0004]	–
	`e5-large`	R@20	-0.0015	[−0.0035, 0.0004]	–
	`e5-large`	MAP	-0.0106	[−0.0136, −0.0078]	–
	`gleif-ft`	R@10	-0.0042	[−0.0065, −0.002]	–
	`gleif-ft`	R@20	-0.0012	[−0.003, 0.0005]	–
	`gleif-ft`	MAP	-0.0136	[−0.0169, −0.0104]	✓
	`txn-ft`	R@10	-0.009	[−0.0124, −0.0057]	–
	`txn-ft`	R@20	-0.0056	[−0.0085, −0.0029]	–
	`txn-ft`	MAP	-0.0349	[−0.0401, −0.0297]	✓
Partial	`e5-large`	R@10	-0.0317	[−0.0402, −0.0234]	✓
	`e5-large`	R@20	-0.0265	[−0.0343, −0.0189]	✓
	`e5-large`	MAP	-0.0715	[−0.0797, −0.0634]	✓
	`gleif-ft`	R@10	-0.0499	[−0.0587, −0.0412]	✓
	`gleif-ft`	R@20	-0.0461	[−0.0541, −0.0382]	✓
	`gleif-ft`	MAP	-0.0485	[−0.0571, −0.0402]	✓

continued on next page

continued from previous page

Category	Model	Metric	Diff	95% CI	Sig.
	txn-ft	R@10	-0.0087	$[-0.0167, -0.0009]$	–
	txn-ft	R@20	-0.006	$[-0.013, 0.0011]$	–
	txn-ft	MAP	-0.0369	$[-0.0451, -0.0291]$	✓
Legal-form	e5-large	R@10	-0.0214	$[-0.0302, -0.0133]$	–
	e5-large	R@20	-0.0138	$[-0.0213, -0.0068]$	–
	e5-large	MAP	-0.0738	$[-0.0862, -0.061]$	✓
	gleif-ft	R@10	-0.0325	$[-0.0425, -0.0232]$	–
	gleif-ft	R@20	-0.0233	$[-0.0321, -0.0151]$	–
	gleif-ft	MAP	-0.092	$[-0.1061, -0.0782]$	✓
	txn-ft	R@10	-0.0191	$[-0.0272, -0.0116]$	–
	txn-ft	R@20	-0.0126	$[-0.0196, -0.0061]$	–
	txn-ft	MAP	-0.0703	$[-0.0824, -0.058]$	✓
No overlap	e5-large	R@10	-0.0213	$[-0.0578, 0.0182]$	–
	e5-large	R@20	-0.0091	$[-0.0456, 0.0274]$	–
	e5-large	MAP	-0.0882	$[-0.1289, -0.0469]$	✓
	gleif-ft	R@10	-0.0957	$[-0.1429, -0.0486]$	–
	gleif-ft	R@20	-0.0441	$[-0.0866, -0.0015]$	–
	gleif-ft	MAP	-0.0737	$[-0.1208, -0.0261]$	✓
	txn-ft	R@10	-0.0106	$[-0.0502, 0.0274]$	–
	txn-ft	R@20	0.0213	$[-0.0122, 0.0578]$	–
	txn-ft	MAP	-0.0408	$[-0.079, -0.0012]$	–

Table D.4: Standalone significance on transactions, each model versus e5-base (paired Wilcoxon signed-rank, Holm-corrected). Diff is mean(model)−mean(reference); ✓ marks $p_{\text{adj}} < 0.05$.

Category	Model	Metric	Diff	95% CI	Sig.
Overall	e5-large	R@10	0.05	$[0.0452, 0.0548]$	✓
	e5-large	R@20	0.0417	$[0.0372, 0.0462]$	✓
	e5-large	MAP	0.05	$[0.0453, 0.0547]$	✓
	gleif-ft	R@10	0.0368	$[0.0314, 0.0421]$	✓
	gleif-ft	R@20	0.0297	$[0.0248, 0.0347]$	✓
	gleif-ft	MAP	0.0593	$[0.0537, 0.0646]$	✓
	txn-ft	R@10	0.0603	$[0.0557, 0.0649]$	✓
	txn-ft	R@20	0.052	$[0.0478, 0.0562]$	✓
	txn-ft	MAP	0.0613	$[0.0573, 0.0652]$	✓
Exact	e5-large	R@10	0.0181	$[0.0138, 0.0225]$	–
	e5-large	R@20	0.014	$[0.0104, 0.018]$	–
	e5-large	MAP	0.038	$[0.0326, 0.0436]$	✓
	gleif-ft	R@10	0.0163	$[0.0119, 0.0209]$	–

continued on next page

continued from previous page

Category	Model	Metric	Diff	95% CI	Sig.
	gleif-ft	R@20	0.0143	[0.0104, 0.0184]	–
	gleif-ft	MAP	0.0349	[0.0293, 0.0407]	✓
	txn-ft	R@10	0.0115	[0.0076, 0.0154]	–
	txn-ft	R@20	0.0099	[0.0065, 0.0135]	–
	txn-ft	MAP	0.0137	[0.0099, 0.0176]	✓
Partial	e5-large	R@10	0.0725	[0.0643, 0.0807]	✓
	e5-large	R@20	0.0633	[0.0557, 0.071]	✓
	e5-large	MAP	0.0595	[0.0519, 0.0672]	✓
	gleif-ft	R@10	0.0543	[0.0453, 0.0633]	✓
	gleif-ft	R@20	0.0437	[0.0355, 0.0521]	✓
	gleif-ft	MAP	0.0825	[0.074, 0.091]	✓
	txn-ft	R@10	0.0955	[0.0878, 0.1034]	✓
	txn-ft	R@20	0.0838	[0.0765, 0.0911]	✓
	txn-ft	MAP	0.094	[0.0876, 0.1007]	✓
Legal-form	e5-large	R@10	0.0263	[0.0159, 0.0367]	–
	e5-large	R@20	0.018	[0.0089, 0.0271]	–
	e5-large	MAP	0.0466	[0.0343, 0.0592]	✓
	gleif-ft	R@10	0.0152	[0.0035, 0.0269]	–
	gleif-ft	R@20	0.0085	[−0.0015, 0.0186]	–
	gleif-ft	MAP	0.0284	[0.0138, 0.0429]	✓
	txn-ft	R@10	0.0286	[0.0201, 0.0379]	–
	txn-ft	R@20	0.0192	[0.0124, 0.0267]	–
	txn-ft	MAP	0.0501	[0.0404, 0.0604]	✓
No overlap	e5-large	R@10	0.117	[0.0821, 0.1535]	✓
	e5-large	R@20	0.0775	[0.0486, 0.1079]	–
	e5-large	MAP	0.0349	[0.0064, 0.0642]	✓
	gleif-ft	R@10	0.0426	[0, 0.0851]	–
	gleif-ft	R@20	0.0426	[0.003, 0.0821]	–
	gleif-ft	MAP	0.0494	[0.0148, 0.0833]	✓
	txn-ft	R@10	0.1277	[0.0912, 0.1657]	✓
	txn-ft	R@20	0.1079	[0.0729, 0.1429]	✓
	txn-ft	MAP	0.0823	[0.0555, 0.1103]	✓

D.2 Ensemble comparison

Table D.5: Ensemble significance on GLEIF (paired Wilcoxon, Holm-corrected). "matched"/"cross" is the ensemble whose dense component was trained on the target / other distribution; the dense reference is that component. ✓ marks $p_{\text{adj}} < 0.05$.

Ensemble	vs	Category / Metric	Diff	95% CI	Sig.
matched	`tfidf`	Overall / R@20	0.0194	[0.0184, 0.0203]	✓
		Overall / R@30	0.0185	[0.0175, 0.0194]	✓
		Exact / R@20	0	[−0.0001, 0.0001]	–
		Exact / R@30	0	[−0.0001, 0]	–
		Partial / R@20	0.0753	[0.0709, 0.0796]	✓
		Partial / R@30	0.0698	[0.0657, 0.0741]	✓
		Legal-form / R@20	0.0007	[0.0003, 0.0011]	–
		Legal-form / R@30	0.0005	[0.0002, 0.0009]	–
		Abbrev. / R@20	0.1307	[0.0682, 0.2045]	–
		Abbrev. / R@30	0.1136	[0.0455, 0.1932]	–
		No overlap / R@20	0.0536	[0.0486, 0.0586]	✓
		No overlap / R@30	0.0554	[0.0503, 0.0605]	✓
matched	`gleif-ft-hard`	Overall / R@20	0.017	[0.0161, 0.0179]	✓
		Overall / R@30	0.0201	[0.0192, 0.021]	✓
		Exact / R@20	0.0023	[0.0019, 0.0027]	–
		Exact / R@30	0.0023	[0.0019, 0.0027]	–
		Partial / R@20	0.0478	[0.0441, 0.0515]	✓
		Partial / R@30	0.0606	[0.0568, 0.0643]	✓
		Legal-form / R@20	0.0303	[0.028, 0.0327]	✓
		Legal-form / R@30	0.0304	[0.028, 0.0327]	✓
		Abbrev. / R@20	0.0142	[0, 0.0312]	–
		Abbrev. / R@30	0.0142	[0, 0.0312]	–
		No overlap / R@20	0.0098	[0.0064, 0.0133]	–
		No overlap / R@30	0.0179	[0.0146, 0.0213]	✓
cross	`tfidf`	Overall / R@20	0.0191	[0.0181, 0.02]	✓
		Overall / R@30	0.018	[0.0171, 0.0189]	✓
		Exact / R@20	0	[−0, 0.0001]	–
		Exact / R@30	0	[−0.0001, 0.0001]	–
		Partial / R@20	0.0759	[0.0716, 0.0803]	✓
		Partial / R@30	0.0695	[0.0654, 0.0738]	✓
		Legal-form / R@20	0.0007	[0.0003, 0.0011]	–
		Legal-form / R@30	0.0006	[0.0002, 0.0009]	–
		Abbrev. / R@20	0.108	[0.0511, 0.1761]	–
		Abbrev. / R@30	0.0909	[0.0227, 0.1648]	–
		No overlap / R@20	0.0493	[0.0446, 0.0542]	✓
		No overlap / R@30	0.0513	[0.0465, 0.0562]	✓
cross	`txn-ft-hard`	Overall / R@20	0.0116	[0.0108, 0.0125]	✓
		Overall / R@30	0.0093	[0.0085, 0.0101]	✓
		Exact / R@20	0.0026	[0.0022, 0.0031]	–

continued on next page

continued from previous page

Ensemble	vs	Category / Metric	Diff	95% CI	Sig.
		Exact / R@30	0.0021	[0.0017, 0.0025]	–
		Partial / R@20	0.0348	[0.0314, 0.0381]	✓
		Partial / R@30	0.0284	[0.0249, 0.0317]	✓
		Legal-form / R@20	0.0141	[0.0125, 0.0158]	✓
		Legal-form / R@30	0.0106	[0.0092, 0.012]	✓
		Abbrev. / R@20	0.0114	[0, 0.0284]	–
		Abbrev. / R@30	-0.0057	[−0.0341, 0.017]	–
		No overlap / R@20	0.0106	[0.0071, 0.0142]	–
		No overlap / R@30	0.0091	[0.0056, 0.0126]	–

Table D.6: Ensemble significance on transactions (paired Wilcoxon, Holm-corrected). "matched"/"cross" is the ensemble whose dense component was trained on the target / other distribution; the dense reference is that component. ✓ marks $p_{\text{adj}} < 0.05$.

Ensemble	vs	Category / Metric	Diff	95% CI	Sig.
matched	`tfidf`	Overall / R@20	0.0166	[0.0139, 0.0192]	✓
		Overall / R@30	0.0132	[0.0106, 0.0158]	✓
		Exact / R@20	0.0011	[0.0004, 0.002]	–
		Exact / R@30	0.0006	[0.0001, 0.0012]	–
		Partial / R@20	0.0281	[0.0231, 0.0328]	✓
		Partial / R@30	0.022	[0.0174, 0.0267]	✓
		Legal-form / R@20	0.0021	[0.0004, 0.0041]	–
		Legal-form / R@30	0.0021	[0.0004, 0.0041]	–
		No overlap / R@20	0.0669	[0.0426, 0.0942]	–
		No overlap / R@30	0.0578	[0.0334, 0.0851]	–
matched	`txn-ft-hard`	Overall / R@20	0.0143	[0.0116, 0.0171]	✓
		Overall / R@30	0.0128	[0.0103, 0.0154]	✓
		Exact / R@20	0.0045	[0.0025, 0.0068]	–
		Exact / R@30	0.004	[0.002, 0.0061]	–
		Partial / R@20	0.0196	[0.0149, 0.0245]	✓
		Partial / R@30	0.0179	[0.0134, 0.0224]	✓
		Legal-form / R@20	0.0131	[0.007, 0.0197]	–
		Legal-form / R@30	0.012	[0.0066, 0.0182]	–
		No overlap / R@20	0.0355	[0.0172, 0.0557]	–
		No overlap / R@30	0.0263	[0.0061, 0.0476]	–
cross	`tfidf`	Overall / R@20	0.0057	[0.0034, 0.008]	–
		Overall / R@30	0.0038	[0.0017, 0.006]	–
		Exact / R@20	0.0007	[0.0002, 0.0014]	–
		Exact / R@30	0.0002	[0, 0.0006]	–
		Partial / R@20	0.0083	[0.0041, 0.0124]	–
		Partial / R@30	0.0048	[0.001, 0.0087]	–

continued on next page

continued from previous page

Ensemble	vs	Category / Metric	Diff	95% CI	Sig.
		Legal-form / R@20	0.0014	[0.0003, 0.0031]	–
		Legal-form / R@30	0.0014	[0.0003, 0.003]	–
		No overlap / R@20	0.0471	[0.0243, 0.0729]	–
		No overlap / R@30	0.0441	[0.0228, 0.0684]	–
cross	`gleif-ft-hard`	Overall / R@20	0.055	[0.0506, 0.0595]	✓
		Overall / R@30	0.0465	[0.0424, 0.0508]	✓
		Exact / R@20	0.0051	[0.0029, 0.0075]	–
		Exact / R@30	0.0042	[0.0022, 0.0064]	–
		Partial / R@20	0.085	[0.0776, 0.0931]	✓
		Partial / R@30	0.072	[0.065, 0.0795]	✓
		Legal-form / R@20	0.0419	[0.0313, 0.0531]	✓
		Legal-form / R@30	0.0361	[0.0263, 0.0465]	✓
		No overlap / R@20	0.152	[0.1094, 0.1945]	✓
		No overlap / R@30	0.1261	[0.0881, 0.1657]	✓