RÉPUBLIQUE FRANÇAISE
Liberté Égalité Fraternité

Université de Toulouse

Doctorat de l'Université de Toulouse

préparé à l'Université Toulouse - Jean Jaurès

Alignement de sources de données disparates à l'aide de grands modèles de langage

Thèse présentée et soutenue, le 2 décembre 2025 par

Nour Elhouda KIRED

École doctorale
EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse

Spécialité
Informatique et Télécommunications

Unité de recherche
IRIT : Institut de Recherche en Informatique de Toulouse

Thèse dirigée par
Olivier TESTE

Composition du jury
Mme Cassia TROJAHN DOS SANTOS, Présidente, Université Grenoble Alpes
M. Laurent D'ORAZIO, Rapporteur, Université de Rennes
M. Nicolas TRAVERS, Rapporteur, Ecole Supérieure d'Ingénieurs Léonard de Vinci
M. Franck RAVAT, Examinateur, Université Toulouse Capitole
Mme Faten ATIGUI, Examinatrice, CNAM Paris
M. Olivier TESTE, Directeur de thèse, Université Toulouse - Jean Jaurès

Membres invités
M. Jiefu SONG, Université Toulouse Capitole

Contents

Remerciements

Au terme de ce voyage scientifique et humain qu'a représenté cette thèse, je souhaite d'abord rendre hommage à celles et ceux qui ont accompagné chacune de ses étapes, parfois visibles, parfois silencieuses, mais toujours décisives.

Je tiens à exprimer ma profonde reconnaissance à Olivier Teste, directeur de cette thèse. Au-delà de son rôle d'encadrant, il a été un soutien constant, quelqu'un qui a su conjuguer exigences scientifiques et compréhension humaine, rigueur méthodologique et patience, fermeté académique et bienveillance quotidienne. Sa capacité à éclairer les moments d'incertitude, à guider sans imposer, à laisser mûrir les idées tout en les structurant, a largement contribué à modeler la chercheuse que je suis devenue. Ses conseils, toujours pertinents, son regard clairvoyant sur mes travaux et sa disponibilité, même dans les périodes les plus complexes, ont profondément marqué cette aventure doctorale.

Ma gratitude s'étend également à Franck Ravat, co-directeur de la thèse, dont le professionnalisme et l'acuité intellectuelle ont constamment nourri ma réflexion. Sa manière de saisir les points essentiels, d'identifier immédiatement ce qui pouvait être amélioré, et de proposer des orientations justes et constructives a permis à ce travail d'évoluer avec cohérence et ambition. Son exigence méthodologique et son sens du détail ont été des repères solides tout au long du chemin.

Je souhaite aussi souligner l'apport précieux de Jiefu Song, dont les remarques, toujours pertinentes et justes, ont apporté un éclairage précis à des moments clés du projet. Chacune de ses interventions a contribué à clarifier, renforcer ou réorienter certains aspects du travail avec une précision remarquable. Tous trois — Olivier, Franck et Jiefu — ont accompagné avec patience une thèse qui avait commencé sous la forme d'un projet CIFRE et qui, au fil des années, a traversé plusieurs phases, parfois incertaines, parfois exigeantes. Leur constance, leur soutien et leur confiance ont rendu possible l'achèvement de ce manuscrit.

J'adresse également toute mon appréciation aux rapporteurs, Nicolas Travers et Laurent d'Orazio, pour avoir pris le temps de lire ce manuscrit et pour la finesse de leurs commentaires, qui ont renforcé la qualité rédactionnelle et la clarté de plusieurs sections. Je remercie également les membres du jury, Faten Atigui et Cassia Trojahn Dos Santos, pour avoir accepté d'évaluer cette thèse et pour l'attention qu'elles ont portée à sa présentation finale.

Mon parcours a été rendu possible par l'environnement scientifique dans lequel il s'est inscrit. Je tiens à souligner l'importance du laboratoire IRIT et de l'équipe SIG, dans lesquels j'ai trouvé un espace stimulant, exigeant et amical. Les discussions, les échanges, les moments de réflexion ou de partage, parfois brefs, parfois longs, ont été autant de pierres qui ont façonné cette thèse. Parmi celles et ceux que j'ai côtoyés au quotidien, Oumaima, collègue devenue complice, occupe une place particulière. Notre rencontre, dès le premier jour de la thèse, a été l'un de ces hasards heureux que l'on n'oublie pas. Sa présence, son écoute, son soutien et cette fraternité intellectuelle et humaine qui s'est construite au fil du temps ont enrichi cette aventure bien au-delà de la recherche.

Je tiens également à reconnaître le rôle de l'entreprise OKP4, dont la convention CIFRE a financé la première année de ce travail. Cette opportunité a représenté un tremplin important, permettant de poser les bases du projet. Les deux années d'ATER qui ont suivi ont offert un cadre institutionnel essentiel et les conditions nécessaires pour poursuivre et achever ce manuscrit.

Avant de mentionner les amis qui ont accompagné ce parcours, il me tient à cœur de tourner mes pensées vers ma famille, socle, refuge et moteur de tout ce que j'entreprends. À mes parents, Rachid et Aïcha, dont la force, la patience, l'amour et les sacrifices ont façonné mon chemin, je dois bien plus que des mots ne peuvent le dire. Leur confiance, même dans les moments où je doutais de moi-même, a été un soutien silencieux mais inébranlable, une lumière constante au milieu des incertitudes.

Je souhaite également exprimer toute ma gratitude à Amine et Halima Djad pour leur soutien indéfectible, leur affection sincère et pour m'avoir toujours traitée comme leur propre fille. Leur bienveillance, leur présence attentive et la chaleur de leur foyer ont été pour moi une source précieuse de réconfort et de force tout au long de ce parcours.

À mes frères et sœurs : Omar, Nadjlaa, Nahawand, Assinat et Rayane, je veux exprimer toute ma gratitude pour leur affection, leur présence, leurs encouragements simples mais essentiels, et cette chaleur familiale qui m'a accompagné même à distance. À mes grands-parents, dont les prières, la sagesse et la tendresse demeurent des repères, et à l'ensemble de ma famille — tantes, cousins, cousines — qui, chacun à leur manière, ont contribué à ce que je puisse mener ce projet à terme, je souhaite adresser une pensée profonde et reconnaissante.

Viennent ensuite les amis, compagnons de route précieux qui, à travers leurs mots, leurs conseils ou simplement leur présence, ont offert des respirations nécessaires dans ce long processus. Oumaima El Haddadi, dont l'amitié est devenue au fil du temps une évidence ; Lyes Attouche, dont l'aide, la disponibilité et le soutien intellectuel comme humain ont été déterminants ; Fouzia Sadli, pour cette amitié sincère et constante. À tous ceux avec qui j'ai partagé des discussions, des pauses, des instants de doute ou de joie, et qui ont su alléger les jours les plus lourds : votre présence a compté, parfois plus que vous ne l'imaginez.

Cette thèse est le fruit d'années de travail et de persévérance, mais elle est aussi le résultat de l'entrelacement de toutes ces présences — professionnelles, familiales, amicales — qui, chacune à leur manière, ont rendu le chemin possible. Ce manuscrit porte la trace de toutes ces voix, de toutes ces mains tendues, de ces regards encourageants, de ces instants partagés.

À tous ceux qui ont, de près ou de loin, accompagné ce parcours, je souhaite exprimer non seulement ma gratitude, mais aussi ma reconnaissance profonde pour avoir fait de cette aventure scientifique une aventure humaine.

Chapter 1

Introduction

1.1 Motivation and Context

In today's data-driven landscape, modern machine learning and analytics increasingly depend on access to large-scale, diverse, and high-quality datasets. However, in practice, such data is often fragmented, inconsistently structured, and dispersed across numerous systems. The emergence of data lakes—repositories built to store raw, diverse, and loosely organized data—has further accentuated these challenges.

Similar complexities arise in federated databases, multi-model systems, and enterprise-level integration environments, where data originates from fundamentally different paradigms, such as relational, not only SQL representations.

Structured Tabular Data. Our work focuses on *structured tabular data*, represented as two-dimensional tables characterized by a defined schema and an associated collection of instances.

Formally, a dataset is defined as $\mathcal{DS} = (\mathcal{S}, \mathcal{I})$, where:

- $\mathcal{S} = [a_1, \ldots, a_n]$ is the schema, the set of attributes,
- $\mathcal{I} = [t_1, \ldots, t_p]$ is the instances, the set of records where each $t_j = \{a_1 : v_1^j, \ldots, a_n : v_n^j\}$ aligns with attributes in $\mathcal{S}$.

Example. A patient dataset might have:

- Schema: `[PatientID, Age, Diagnosis]`
- Instances: [`{PatientID:001, Age:45, Diagnosis:"Diabetes"}`, `{PatientID:002, Age:53,Diagnosis:"Asthma"}`, ...]

Tabular data remains at the heart of decision-making and many ML workflows across all domains [111, 98]. Ensuring that this typically heterogeneous data can be discovered, interpreted, and combined is therefore an essential prerequisite for scalable and reproducible analysis [41, 49, 87]. This thesis addresses the need for improved approaches to schema alignment in settings where tabular data may be incomplete or unevenly represented. This is especially relevant in data lakes, public administration platforms (open data), healthcare repositories, and internal enterprise stores. Traditional solutions often assume both datasets are well-defined and structurally aligned. In real-world settings, however, this assumption breaks down. Fields may differ in naming, representation, and level of detail—or be entirely absent from one of the sources.

Mismatched Datasets. It is common to encounter mismatched datasets in one of two ways:

- **Schema-rich but lacking instance data**, where column descriptions are present but few records exist. In healthcare, for example, data dictionaries are often shared across institutions to support interoperability, while access to patient-level records is restricted, creating schema-rich but instance-poor resources that must be matched with instance-rich but weakly annotated datasets.
- **Instance-rich but sparse in schema**, where values exist but structure or attribute labels are absent or unusable. A similar situation occurs in enterprises, where legacy systems or external applications export large, unlabeled tables of values that need to be aligned with well-defined internal schemas.

Schemas, when present, may be incomplete, inconsistent, or intentionally removed for reasons such as privacy or sensitivity. This irregularity introduces significant challenges to the alignment process. These difficulties go beyond structure and include semantic variation, where similar concepts are labeled differently or described using varying formats or assumptions. To better articulate the alignment problem, we introduce three key concepts: heterogeneity, compliance, and disparity.

- **Heterogeneity:** Heterogeneity refers to fundamental differences between datasets that complicate integration, interpretation, or direct interoperability. These differences are not limited to naming conventions or formats, but also reflect deeper variations in how data is structured and what it means. The three primary and most widely recognized types of data heterogeneity are [100, 40, 110]:
  - **Syntactic heterogeneity:** Refers to differences in data formats, naming styles, or representations. For example, the same attribute might appear as `DOB`, `DateOfBirth`, or `birth_date` [100].
  - **Structural heterogeneity:** Involves differences in how data is organized or modeled. One dataset might store a person's name in a single field like `FullName`, while another splits it into `FirstName` and `LastName` [100, 40].
  - **Semantic heterogeneity:** Arises when the same data has different meanings in different contexts, or when different data represents the same concept. For instance, the attribute `HR` may refer to "heart rate" in medical data or "human resources" in organizational data [110, 40].
- **Compliance.** Compliance refers to the extent to which two datasets provide comparable information—whether they both include similar structures $\mathcal{S}$, similar content $\mathcal{I}$, or both $(\mathcal{S}, \mathcal{I})$. When datasets are compliant, they share a sufficient level of alignment to support effective matching.
- **Disparity.** Disparity describes situations where one dataset is complete while the other is partial, lacking either structure or content. These imbalances often disrupt traditional matching methods.

Six Alignment Scenarios: Compliance vs. Disparity. To link these real-world dataset mismatches to our framework, we cast them as input scenarios that distinguish compliant cases from disparate ones.

- **Compliant scenarios (C1–C3):** Both sources contain similar forms of information

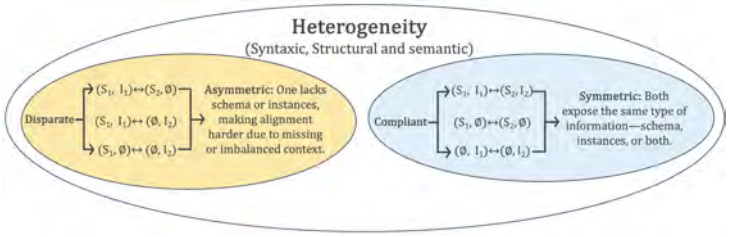


Figure 1.1: Taxonomy of input scenarios under heterogeneity.

- **Disparate scenarios (D4–D6):** One dataset lacks schema or instances

Heterogeneity (syntactic, structural, semantic) can occur in every scenario in Figure 1.1; compliance or disparity only specify whether $\mathcal{S}$ and/or $\mathcal{I}$ are available symmetrically (compliant) or asymmetrically (disparate).

Compliant.

C1: Complete $(\mathcal{S}_1, \mathcal{I}_1) \leftrightarrow$ Complete $(\mathcal{S}_2, \mathcal{I}_2)$

C2: Schema-only $(\mathcal{S}_1, \varnothing) \leftrightarrow$ Schema-only $(\mathcal{S}_2, \varnothing)$

C3: Instance-only $(\varnothing, \mathcal{I}_1) \leftrightarrow$ Instance-only $(\varnothing, \mathcal{I}_2)$

Disparate.

D4: Complete $(\mathcal{S}_1, \mathcal{I}_1) \leftrightarrow$ Schema-only $(\mathcal{S}_2, \varnothing)$

D5: Complete $(\mathcal{S}_1, \mathcal{I}_1) \leftrightarrow$ Instance-only $(\varnothing, \mathcal{I}_2)$

D6: Schema-only $(\mathcal{S}_1, \varnothing) \leftrightarrow$ Instance-only $(\varnothing, \mathcal{I}_2)$

Real-world datasets often reflect the six scenarios described earlier (see Figure 1.1). For example, in an open government data lake, one agency might publish a dataset listing patient monitoring attributes such as `Patient_ID`, `Date_Of_Birth`, or `Heart_Rate`, but with no associated instances. In contrast, another institution might provide thousands of instances with vital signs, but minimal or ambiguous labels—such as `ID`, `DOB`, or `HR`. Despite covering similar subjects, the two datasets remain difficult to compare without additional context (Scenario 6: Schema-only $(\mathcal{S}_1, \varnothing) \leftrightarrow$ Instance-only $(\varnothing, \mathcal{I}_2)$).

Such disparities are not limited to naming. Units may differ (e.g., metric vs. imperial), abbreviations might be unclear, and some attributes may appear only on one side. Furthermore, critical metadata might be incomplete, ambiguous, or entirely missing. These inconsistencies introduce fundamental obstacles to matching, particularly when traditional schema alignment methods rely heavily on the presence of either descriptive names or representative data values.

The challenge intensifies when both sides of the match are only partially specified. A human analyst may be able to deduce that a column labeled `HR` corresponds to `Heart_Rate`, or that `BP` refers to blood pressure—possibly split into `SystolicBP` and `DiastolicBP` in another dataset. However, most automated matchers cannot perform such reasoning without access to richer contextual clues.

These cases correspond most directly to the disparate scenarios (D4–D6) in our taxonomy, where one dataset lacks schema or instances. They also highlight that syntactic and structural mismatches are particularly problematic when information is missing, whereas semantic mismatches remain even more challenging and cannot always be resolved without additional reasoning (D4–D6).

1.2 Problem Statement

Existing matchers are not equipped to handle such input heterogeneity, especially in scenarios where sources may be:

- **Incomplete:** Missing metadata, ambiguous labels, or lacking instance values.

  This can occur in scenarios (C2–D6), whenever one or both datasets lack either schema or instances.

- **Asymmetrical:** One schema may contain structure but no values, while the other has values but no labels.

  This corresponds most directly to the disparate scenarios (4–6), where the available information is unbalanced across the two sources.

- **Ambiguous or noisy:** Featuring cryptic column names, domain-specific jargon or abbreviations, or inconsistent formatting.

  These cases reflect forms of heterogeneity (syntactic or semantic) that may arise across both compliant and disparate scenarios.

Classical matchers—whether based on string similarity, ontology lookups, or instance-level statistics—are brittle in such environments.

Early systems such as CUPID [78], Similarity Flooding [81], or COMA/COMA++ [36, 4, 79] rely heavily on schema labels and structural cues, which makes them sensitive to missing or noisy metadata. Ontology-based systems like AgreementMaker [27] or AgreementMakerLight [44] improve coverage but remain constrained by domain ontologies and often struggle with heterogeneity across contexts. Probabilistic approaches such as PARIS [113] or corpus-driven methods [77] leverage instance data, but their effectiveness drops sharply when instances are sparse or unavailable.

Hybrid approaches that combine schema and instance cues also degrade when one modality is missing or insufficient.

As summarized in surveys [100, 110], these methods assume access to both schema and instance-level information. In the disparate scenarios (D4–D6), where one source lacks schema or instances, their performance deteriorates significantly.

Problem Statement. The goal of this thesis is to develop a unified pairwise schema matching framework capable of aligning structured tabular data sources under varying types of input compliant and disparate scenarios—including schema-only, instance-only, and mixed incomplete scenarios—.

This framework assumes a 1:1 attribute matching, where each attribute in the source schema is mapped to at most one corresponding attribute in the target schema. When explicit attribute labels are present, correspondence is defined over names; when labels are absent (instance-only scenarios), each column is instead identified by its positional index. This restriction is appropriate for many practical integration tasks involving tabular data. To achieve this goal, several subproblems must be addressed. First, we need a method to *generate missing schema or instance information* for a given dataset using minimal information. Second, we need to determine how to best use this augmented information to compare attributes between datasets. Finally, we must design matching algorithms or learning models that can operate effectively on the enriched representations, handling noise or inconsistencies in the generated context. The following sections outline our approach to tackling these challenges.

1.3 Proposed Approach Overview

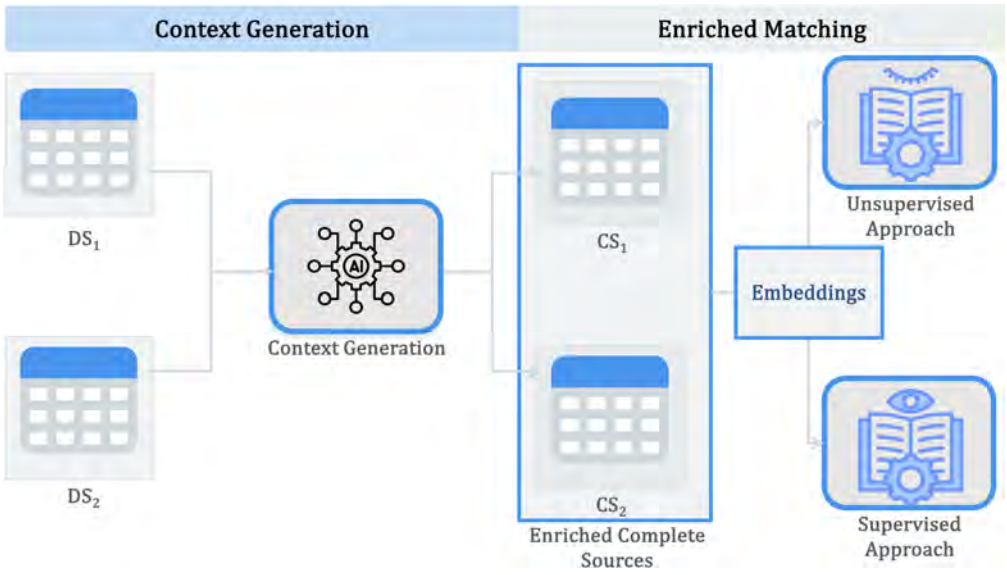


Figure 1.2: Overview of the proposed schema matching framework.

To address the challenge of aligning disparate data sources, this thesis proposes an approach built around the idea of **context generation** and **enriched matching**. Figure 1.2 depicts the high-level framework. The key insight is to proactively compensate for missing information in one source by using external knowledge or generative models, so that both sources can be represented in a common, comparable form. Once this is achieved, we can apply or adapt existing matching techniques on the enriched data. We briefly outline the major components of our approach below.

Generating Context for Incomplete Data. The first component of our solution is a mechanism to automatically generate the *missing modality* for any given dataset. For a schema-only table $(\mathcal{S}, \varnothing)$, we generate synthetic instance values for each attribute, aiming to produce a small but representative set of values that one would expect in that column. For an instance-only table $(\varnothing, \mathcal{I}$, we infer plausible attribute names or descriptions for each column based on the values it contains. We explore multiple strategies for this context generation:

- **Lexical and Knowledge-based Inference.**

  This strategy leverages lexical databases such as WordNet [82] and domain ontologies such as UMLS [8] to map schema labels or value tokens to established semantic categories. It enables semantic disambiguation by grounding ambiguous column names or recurring patterns in structured vocabularies.

  For example, detecting that a column's values follow a date format may support assigning a header like "Date" or "Timestamp"; if a schema-only attribute is labeled "Company", external knowledge bases can provide representative examples such as company names.

- **Rule-based Data Profiling.**

This strategy applies profiling methods based on statistical regularities and syntactic patterns [1, 23]. By analyzing distributions, ranges, or character occurrences, it infers candidate semantic types for columns.

For instance, a numeric column with values restricted to 0–120 may be interpreted as "Age", while a text column containing frequent '@' characters could plausibly represent "Email".

- **Large Language Models (LLMs).**

  This strategy exploits generative LLMs to synthesize the missing modality of a column. Given either a label or a set of sample values, the model is prompted to generate complementary metadata or representative values. Prior studies demonstrate the effectiveness of this approach for schema matching tasks [96, 21, 107].

  For example, given a schema-only column "Disease_Name", an LLM may generate values such as ["$Influenza$", "$Diabetes$", "$Hypertension$"]; given an unlabeled numeric column with values $[120, 135, 110, 140]$, it might suggest "Systolic Blood Pressure" as a candidate header.

By applying such techniques, each originally incomplete dataset is transformed into a *more complete* form: schema-only tables are enriched with synthetic but representative data instances, and instance-only tables are complemented with inferred schema annotations (e.g., descriptive labels or semantic categories). Importantly, the generated context is not guaranteed to be perfect or exhaustive, but it provides additional information for matching that was previously absent. Our framework subsequently evaluates and refines this context to minimize noise.

Embedding-Based Matching. After augmenting the data sources with inferred context, the next step is to perform the schema alignment. We investigate two complementary approaches in this Phd thesis:

1. **Unsupervised Alignment using Matcher.** In one approach, we treat each enriched dataset pair as an input to existing matchers (both schema-based and instance-based), and then combine their outputs to decide the final matches.

2. **Supervised Alignment using Meta-Features.** In a second approach, we cast the matching problem as a supervised learning task. Here, we make use of the enriched data to compute vector embeddings for each attribute (for example, by feeding the attribute name and a sample of values into a pre-trained language model). This yields a unified representation for schema and instance information in the same semantic space. We then define a rich set of *meta-features* that quantify the similarity between two attribute embeddings from different tables. These meta-features go beyond the standard cosine similarity, encompassing various geometric, statistical, and topological comparisons between the embedding vectors or their local neighborhoods. With up to hundreds of such meta-features computed for each candidate pair of attributes, we then train a binary classifier to predict whether a pair is a match or not.

By pursuing both an unsupervised and a supervised strategy, we cover a broad spectrum of use cases. The unsupervised method is applicable in scenarios where no labeled correspondences are available. The supervised method can leverage examples to tune a more precise matcher for higher accuracy when such training data is available. Both strategies share the initial step of context generation via LLMs, underlining the central role of context enrichment in our framework. Furthermore, both make heavy use of modern representation learning (embeddings) to encode the semantics of attributes in a comparable form, which is crucial for dealing with the subtle semantic differences in disparate data.

1.4 Scientific Contributions and Supporting Resources

The research presented in this thesis yields several contributions to the field of data integration and schema matching:

- We introduce an unsupervised schema-instance matching framework that leverages large language models to generate synthetic schemas and instances for incomplete data sources, and combines multiple matching algorithms to align disparate tables without requiring any labeled training examples. To our knowledge, this is one of the first frameworks to successfully match a schema-only source with an instance-only source by augmenting and reconciling both sides.
- We propose a supervised schema matching approach that uses LLMs to enrich incomplete datasets and embeds the augmented data into a unified latent space. We define a comprehensive meta-feature space. This rich meta-feature representation allows a classifier to capture subtle semantic correspondences that would be missed by any single metric.
- Through extensive experiments on the Valentine benchmark [66] and additional real-world datasets, we demonstrate that our approaches achieve state-of-the-art performance in challenging disparate scenarios. The unsupervised method also shows substantial improvements over traditional individual matchers, confirming the benefit of both context generation and matcher ensembling. We further show that our methods remain robust under varying levels of noise and mismatch in the input, outperforming comparables even as schema information degrades.
- We release all artifacts required to fully reproduce our contributions: implementations, experimental datasets, execution scripts, and detailed results. All repositories share a uniform, production-ready layout—packaged with `poetry` for dependency management; source code under `src/`; tests under `tests/`; and detailed documentation (`README`) with a one-line command to run the experiments. $\mathcal{I}$ For DAWAK [61], we provide a ready-to-run package[1] and a separate repository that hosts all DAWAK datasets, configurations, and final results[2]. (ii) For RCIS [62] and BDA [63] (unsupervised, matcher-based framework), the code, tests, documentation, and results are available[3]. (iii) For the supervised framework, we release the meta-feature computation code, experimental pipelines, and a complete usage example[4]. These resources ensure transparency, enable exact reproducibility, and facilitate reuse.

  Publications:

  1. Kired, N. E., Ravat, F., Song, J., Teste, O., "Embedding-based data matching for disparate data sources," *Proc. 26th International Conference on Big Data Analytics and Knowledge Discovery (DAWAK 2024).*
     DOI: 10.1007/978-3-031-68323-7_5 HAL: hal-04612345.
  2. Kired, N. E., Ravat, F., Song, J., Teste, O., "Alignment of Schema-Only and Instance-Only Data Sources Using Large Language Models," *19th International Conference on Research Challenges in Information Science(RCIS 2025).*
     DOI: 10.1007/978-3-031-92471-2_5 HAL: hal-05091439.

[1] https://gitlab.irit.fr/sig/theses/nour-kired/data-matching
[2] https://gitlab.irit.fr/sig/theses/nour-kired/Embedding-based-data-matching-for-disparate-data-sources
[3] https://gitlab.irit.fr/sig/theses/nour-kired/Alignment-of-schema-only-and-instance-only-data
[4] https://gitlab.irit.fr/sig/theses/nour-kired/MetaMatch

3. Kired, N. E., Ravat, F., Song, J., Teste, O., "Alignment of Schema-Only and Instance-Only Data Sources Using Large Language Models",*41èmes journées de la conférence Gestion de Données – Principes, Technologies et Applications (BDA 2025).*

In summary, this thesis advances the state of the art in schema matching by introducing mechanisms to handle previously intractable scenarios of disparate data. By generating context with LLMs and thoughtfully integrating multiple similarity measures, we pave the way for more autonomous and resilient data integration techniques. In the following chapters, we delve into each aspect of this work in detail, starting with a survey of related literature in the next chapter.

1.5 Structure of the Thesis

This dissertation is organized as follows.

- **Chapter 1: Introduction** – Introduces the research context and motivation, defines the problem of schema matching across disparate data sources, presents the proposed approach, and outlines the objectives and research questions.
- **Chapter 2: Related Work** – Reviews the literature on schema matching, including classical matching techniques (schema-based, instance-based, hybrid approaches), advances in embedding-based and machine learning methods for schema or ontology matching, and recent efforts to apply large language models or other AI techniques for data matching. This chapter positions our work in relation to existing approaches and highlights the gap addressed by our framework.
- **Chapter 3: Framework for Disparate Data Matching** – Details the methodologies developed in this work. We first describe the context generation procedures for schema-only and instance-only data (using LLM prompting, knowledge bases, etc.). Then we present the unsupervised matching pipeline, including the set of base matchers and the strategy for combining their outputs. Next, we describe the supervised matching pipeline, defining the embedding-based meta-features and the training of the matching classifier.
- **Chapter 4: Experimental Evaluation** – Presents an extensive evaluation of our approaches. We describe the experimental setup, including datasets, evaluation metrics, and baseline methods used for comparison. We report results for both the unsupervised and supervised methods under various scenarios: schema-only vs instance-only matching, complete vs incomplete matching, the effects of using or not using LLM-generated context, etc. We also include ablation studies (e.g., assessing the contribution of different meta-feature types) and analyze the robustness of the methods to noise and varying parameter settings.
- **Chapter 5: Conclusion and Future Work** – Summarizes the contributions and findings of the thesis, discusses the implications for the field of data integration, and outlines potential directions for future research. We consider limitations of the current approach and how they might be addressed, as well as other applications of the context generation and matching techniques developed.

Each chapter builds on the previous one: from defining the problem, to reviewing related work, to presenting the framework, validating it experimentally, and concluding with broader perspectives.

Chapter 2

A Literature Review on Schema Matching in Aligning Tabular Data Sources

2.1 Introduction

Schema matching is a fundamental task in data integration, aiming to identify relationships between attributes in different datasets that refer to the same concept. It is essential in applications such as data warehousing, federated query processing, data cleaning, and knowledge graph construction [100, 36]. Despite decades of research, schema matching remains challenging in practice due to inconsistencies in how data is labeled, organized, and structured across sources.

In the classical setting, each dataset provides a schema consisting of attribute names, types, and constraints, along with instance data. Matching relies on identifying correspondences between attributes based on name similarity, type compatibility, or statistical patterns in data values. However, real-world scenarios often deviate from this setting: datasets may lack meaningful attribute names, provide few or no instance values, or originate from independent sources without shared conventions [41].

Modern data environments—such as open data platforms, web tables, and data lakes—frequently involve incomplete, noisy, or asymmetric inputs [13, 87]. For example, one dataset may contain attribute names without values (schema-only), while another provides values without names (instance-only) [59, 77]. In such conditions, traditional methods fail to produce reliable matches [100, 36]. These difficulties are compounded by the presence of different naming styles, attribute groupings, and context-dependent meanings [100, 110].

To address these challenges, schema matching has evolved from rule-based approaches to approaches that incorporate machine learning, probabilistic models, and external knowledge. Early approaches used string similarity and handcrafted rules [100, 36]. Later methods introduced instance-based evidence and hybrid scoring functions [89, 7]. More recently, *representation learning methods* have emerged, that is, approaches in which models automatically learn vector-based representations (embeddings) of attribute names, values, or entire column. In such embeddings, semantically similar attributes are mapped close to each other in the representation space, allowing similarity to be computed via distance metrics [86, 73].

The latest advances use large pretrained language models (LLMs), such as BERT [32], T5 [99], and GPT [10], which can capture semantic information even in low-resource or ambiguous cases.

These models support techniques such as synthetic context generation and zero-shot matching, allowing approaches to align attributes despite missing schema or instance information [52, 96].

To guide our literature review, we rely on two criteria: (i) **Input coverage**, i.e., which dataset configurations (C1–C3 compliant and D4–D6 disparate) each method can handle, and (ii) **Similarity space**, i.e., whether similarity is restricted to a single metric (e.g., cosine) or explored across richer families (statistical, spectral, topological, etc.). These criteria are used throughout the section and summarized in the comparison tables 2.2 and 2.3.

Given this evolving landscape, this chapter provides a structured review of schema matching techniques, with a focus on handling incomplete, heterogeneous, and asymmetric data. It is organized as follows:

- Section 2 formalizes key definitions, covering tabular data, the concept of schema matching, types of matching, and corresponding similarity metrics for aligning data sources.
- Section 3 reviews traditional approaches, including constraint-based, instance-based, schema-based, and hybrid methods.
- Section 4 examines embedding-based and LLM-driven techniques for schema matching, with a focus on their use in contextual enrichment and fine-tuning for domain-specific scenarios.
- Section 5 provides a comparative summary of existing approaches and highlights their limitations.
- Section 6 concludes with a synthesis of key findings and outlines the motivation behind the proposed framework.

2.2 Definitions of Key Concepts

This PhD Thesis focuses on the field of schema matching for tabular data. Therefore, in this section, we define the concepts of tabular datasets, schema matching, types of tabular matching, measures associated with schema matching, and the application of tabular schema matching.

2.2.1 Tabular Dataset

This section builds on Chapter 1 and puts it into practice. We formally define what a column is, clearly distinguish between attributes, schemas, values, and instances, and then clarify the three input scenarios for data sources (schema-only, instance-only, and complete). We then present the fundamental concepts used throughout the chapter: what constitutes a column correspondence; the main forms of heterogeneity (syntactic, structural, semantic); standard matching settings (pairwise, holistic, mediated); families of similarity measures; and how these support applications such as entity resolution, data cleaning, and data augmentation.

A tabular dataset $\mathcal{DS}$ consists of two components: a schema $\mathcal{S}$ and an instance set $\mathcal{I}$:

$$\mathcal{DS} = (\mathcal{S}, \mathcal{I})$$

where:

- $\mathcal{S} = [a_1, \ldots, a_n]$ is the schema, the set of attributes,
- $\mathcal{I} = [t_1, \ldots, t_p]$ is the instances, the set of records where each $t_j = \{a_1 : v_1^j, \ldots, a_n : v_n^j\}$ aligns with attributes in $\mathcal{S}$.

A *column* is defined as the pair $(a_i, \{v_1^i, \ldots, v_p^i\})$, where $a_i \in \mathcal{S}$ is the attribute label and $\{v_1^i, \ldots, v_p^i\}$ are the values of a_i across all records in $\mathcal{I}$. In this way, a column captures both the schema-level attribute and its associated instance-level values.

Depending on the presence or absence of schema or instance data, we define:

- **Schema-only dataset**: $\mathcal{I} = \emptyset$
- **Instance-only dataset**: $\mathcal{S} = \emptyset$
- **Complete dataset**: both $\mathcal{S} \neq \emptyset$ and $\mathcal{I} \neq \emptyset$

2.2.2 Schema Matching

Given two datasets $\mathcal{DS}_1 = (\mathcal{S}_1, \mathcal{I}_1)$ and $\mathcal{DS}_2 = (\mathcal{S}_2, \mathcal{I}_2)$, the schema matching task aims to find semantic correspondences between attributes in $\mathcal{S}_1$ and attributes in $\mathcal{S}_2$.

Formally, let:

$$\text{match} : \mathcal{S}_1 \times \mathcal{S}_2 \to [0, 1]$$

be a similarity function that assigns a score to each attribute pair (a, b) with $a \in \mathcal{S}_1$ and $b \in \mathcal{S}_2$. A higher score indicates stronger evidence that a and b refer to the same concept. A threshold θ is typically used to define the final matching set:

$$\mathcal{M}_\theta = \{(a, b) \in \mathcal{S}_1 \times \mathcal{S}_2 \mid \text{match}(a, b) \geq \theta\}$$

Understanding Column Matches Schema matching for tabular data is not based solely on the correspondence of attribute names, but also on the values of its attributes (columns). Determining whether two columns from different tabular data refer to the same concept is essential. A column match indicates that both attributes relate to the same semantic domain and can be linked meaningfully. However, this notion can be nuanced and context-dependent. We distinguish the following types of column correspondences, as illustrated in Figure 2.1:

Type 1: Columns whose attributes and values are both syntactically and semantically identical or similar. For instance, the `Full Name` column in both tables contains comparable string values representing employee names.

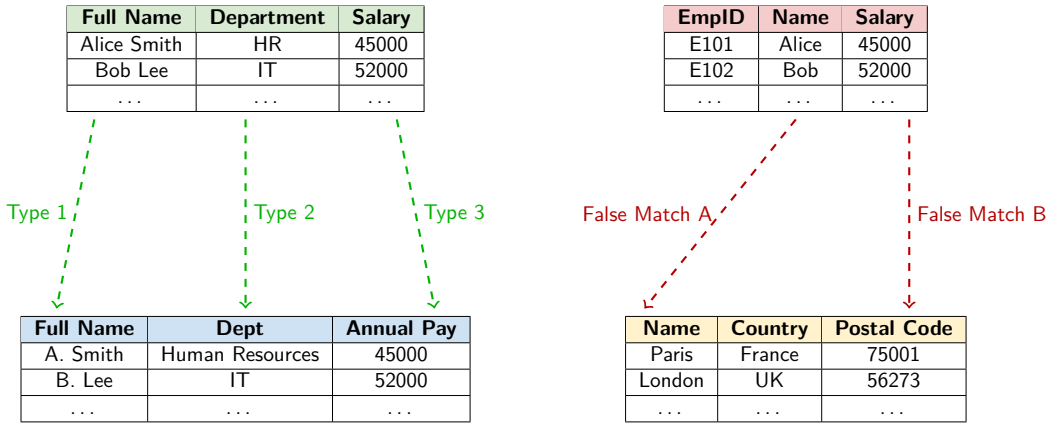
Type 2: Column attributes are semantically equivalent but differ in spelling, structure, or abbreviation. For example, `Department` and `Dept` both refer to organizational units, but one uses the full term while the other uses an abbreviation.

Type 3: Columns represent the same concept using different terminologies or levels of abstraction. For instance, `Salary` and `Annual Pay` both refer to an employee's compensation but use different labels or phrasing.

Conversely, some column pairs should not be matched:

False Match A: Two columns labeled `Name` may refer to different entities—such as an employee's name versus a city name.

False Match B: Columns like `Salary` and `Year` may contain similar numeric formats, but represent unrelated concepts.



(a) Valid Column Matches **(b) False Column Matches**

Figure 2.1: Illustration of valid (left) and false (right) column matches with example tables.

These variations in column semantics originate from different types of heterogeneity across datasets. They must be understood and addressed in schema matching strategies. Table 2.1 shows how the different match cases relate to common heterogeneity types.

Heterogeneity in Schema Matching

As introduced in Chapter 1, we use *heterogeneity* to denote the different ways equivalent or related information can be represented across data sources. For completeness, we recall the standard taxonomy [100, 48, 110]:

1. Syntactic heterogeneity. Syntactic heterogeneity refers to differences in the form or representation of schema elements. These include variations in naming conventions, formatting, abbreviations, delimiters, or data encoding.

For example, `DateOfBirth`, `DOB`, and `birth_date` are syntactically different but semantically equivalent.

2. Structural heterogeneity. Structural heterogeneity arises when the same or similar concepts are organized differently within the schema. This includes differences in schema granularity, nesting, grouping, and data modeling choices.

For example, one schema may use a composite field `Address`, while another decomposes it into `Street`, `City`, and `ZipCode`.

3. Semantic heterogeneity. Semantic heterogeneity involves differences in the meaning or interpretation of schema elements. These mismatches can result from synonyms (different names for the same concept), homonyms (same name for different concepts), or variations in conceptual granularity, context, or domain assumptions. For example, `HR` may refer to "heart rate" in a medical domain and "human resources" in a corporate setting.

Table 2.1: Mapping between column match cases and types of heterogeneity

Column Match Case	**Valid**	**Heterogeneity Type**	**Comment**
Case #1 — `Full Name` vs. `Full Name`	Yes	None	Identical values and meaning
Case #2 — `Department` vs. `Dept`	Yes	Syntactic	Same domain, different naming
Case #3 — `Age` vs. `Years`	Yes	Semantic	Different terms, same meaning (age in years)
Case #4 — `DOB` vs. (`Birth_Year`, `Birth_Month`, `Birth_Day`)	Yes	Structural + Semantic	Abbreviation/synonym (`DOB` vs. `Date of Birth`) and 1:n decomposition; valid after recomposing the date
False Match #1 — `Name` (person) vs. `Name` (location)	No	Semantic	Label matches, meanings differ
False Match #2 — `Years` vs. `Salary`	No	Semantic	Values overlap (numbers), but concepts unrelated
— Concept split or merged across columns (`FullName` vs. `FirstName`, `LastName`)	Yes	Structural	Not illustrated in current figure but common

The aforementioned valid and invalid match cases illustrate the diverse ways in which columns can appear related or misleadingly similar. These cases are not isolated but stem from more fundamental sources of variation across datasets. In the schema matching literature, such differences are generally referred to as *heterogeneity*, and can be systematically categorized into distinct types. Understanding these types is crucial to designing effective matching strategies. We now review the main forms of heterogeneity that affect schema matching tasks.

These three categories have guided much of the research on schema matching. Syntactic heterogeneity is typically addressed by name-based matchers that rely on string similarity or tokenization (e.g., edit distance, Jaccard similarity, COMA++ [4], Cupid [78]). Structural heterogeneity has been approached through graph-based or constraint-based methods, such as Similarity Flooding [81] and Clio [83]. Semantic heterogeneity motivated the use of external knowledge sources and ontologies, including WordNet [82], S-Match [48], and more recent embedding-based techniques [86, 73]. Recent surveys confirm that these categories of heterogeneity remain central even in modern data ecosystems such as data lakes and web tables [24, 87]. Emerging approaches based on pretrained LLMs (e.g., [52, 96, 11]) explicitly target cases of semantic and contextual heterogeneity that cannot be resolved with traditional similarity measures.

In summary, the three forms of heterogeneity remain the backbone of schema matching research. Classical methods tend to assume that both attribute names and instance values are available and reliable, which restricts their **input coverage** mostly to complete datasets (C1–C3). Moreover, the **similarity space** they explore is often narrow, relying primarily on lexical, structural, or distributional measures that capture surface-level resemblance but not deeper semantics. While embedding-based methods have widened this similarity space and LLM-driven techniques push it even further by introducing contextual reasoning, both families still face limitations when applied to incomplete or asymmetric datasets (schema-only or instance-only). Thus, existing solutions cannot fully resolve heterogeneity in realistic, noisy environments. This gap directly motivates our thesis, which aims to design schema matching methods that remain robust under asymmetry, instead of assuming its availability.

2.2.3 Types of Matching

Given $a \in \mathcal{S}_1$ and $b \in \mathcal{S}_2$, a matching can take one of several forms:

- **One-to-one (1:1)**: Each attribute in $\mathcal{S}_1$ matches to at most one attribute in $\mathcal{S}_2$, and vice versa.
- **One-to-many (1:n)**: One attribute in $\mathcal{S}_1$ maps to multiple attributes in $\mathcal{S}_2$.
- **Many-to-one (n:1)**: Multiple attributes in $\mathcal{S}_1$ map to one attribute in $\mathcal{S}_2$.
- **Many-to-many (n:m)**: Multiple attributes in $\mathcal{S}_1$ map to multiple attributes in $\mathcal{S}_2$ simultaneously.
- **Null match ($\perp$)**: An attribute in one schema has no suitable match in the other schema (e.g., all similarity scores are below threshold θ).

Although various methods have been proposed in the literature for schema matching in tabular data, their objectives often differ depending on the specific setting they address. Notably, the context in which a schema matching method is applied can influence how it operates. Figure 2.2 illustrates the three main settings commonly found in tabular schema matching studies, which we briefly describe below.

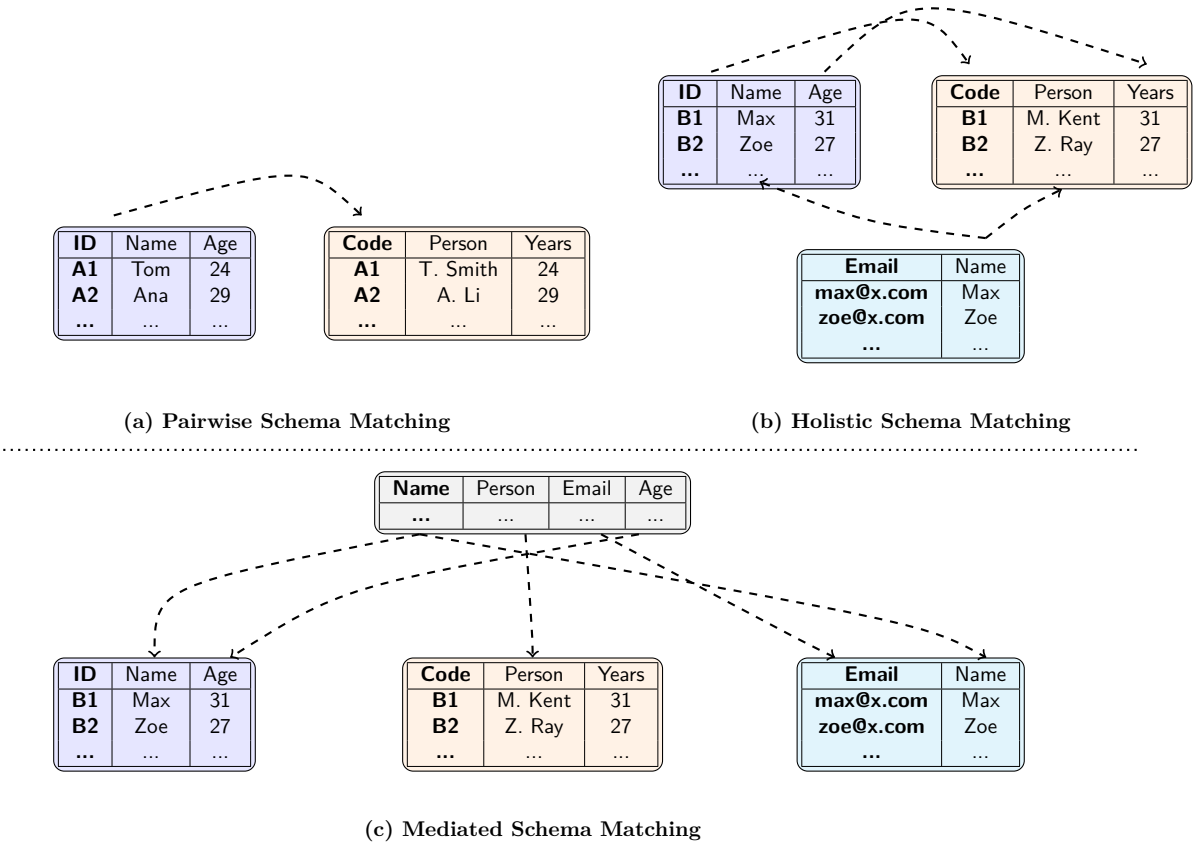
1. Pairwise Schema Matching. Pairwise schema matching is the task of identifying correspondences between the columns of two tabular datasets. In this setting, one table is typically designated as the *source* and the other as the *target*, and the goal is to align attributes that refer to the same underlying concept [100, 36, 110]. This scenario is common when integrating data from two related datasets—such as employee tables from different repositories—that describe similar entities but use different schema representations.

An example is shown in Figure 2.2a, where semantically related columns are identified between two person-related tables. To perform such matching, most methods compute similarity scores between every possible pair of columns from the two datasets. Then, based on these scores, they select the most likely matches—often by simply choosing the highest-scoring pairs one by one, while making sure that each column is matched only once.

While conceptually simple, pairwise schema matching introduces foundational techniques and insights that serve as building blocks for more complex scenarios like holistic or mediated schema matching, which we describe next.

2. Holistic Schema Matching. When dealing with large collections of datasets, schema matching must go beyond individual table pairs. In these cases, we aim to discover column relationships across an entire repository or database collection — a task referred to as *holistic schema matching*. Unlike the pairwise setting, holistic schema matching considers all available datasets at once [80][102], allowing it to uncover global patterns and ensure more coherent alignments across sources. Figure 2.2b illustrates an example involving column relationships across three different tables.

To address this broader challenge, various strategies have been proposed. Some approaches extend pairwise methods across all combinations of datasets [29, 12], others treat columns as standalone data elements and apply clustering techniques to group related attributes [125, 18], while still others leverage external knowledge sources — such as ontologies or structured corpora — to guide the matching process [45, 74].



(a) Pairwise Schema Matching

(b) Holistic Schema Matching

(c) Mediated Schema Matching

Figure 2.2: The three settings studied in the tabular schema matching literature: (a) pairwise schema matching, (b) holistic schema matching, and (c) mediated schema matching.

Each approach has its own strengths and trade-offs, depending on the structure of the data, the scalability requirements, and the availability of auxiliary information. Holistic schema matching remains an active area of research, particularly relevant to large-scale data integration and schema discovery tasks.

3. Mediated Schema Matching. As discussed earlier in the chapter, user queries in data repositories often require combining information scattered across multiple datasets. To support such scenarios, it is useful to define a unified global view — called a *mediated schema* — that allows users to interact with the data without needing to know the structure or location of the original sources.

In this setting, known as *mediated schema matching*, the goal is to connect the columns of each dataset to the most relevant attributes in the mediated schema, based on their semantic meaning. Unlike holistic schema matching, which seeks to discover all possible relationships among columns across datasets, mediated schema matching focuses only on mapping dataset-specific columns to a shared global schema. Figure 2.2c illustrates this scenario: a central schema describes person-related attributes, and each dataset matches only the columns that correspond to those attributes.

While holistic and mediated schema matching are important paradigms in large-scale integration and query mediation, this thesis focuses specifically on the pairwise setting. Pairwise matching is the most fundamental case, serving as the basis for holistic and mediated strategies, which often rely on pairwise correspondences as building blocks. More importantly, the central research question of this work concerns how to perform matching when the available inputs are incomplete or asymmetric (schema-only, instance-only, or partially missing). Such conditions are naturally defined and studied in the pairwise setting, where two datasets must be aligned despite missing or unbalanced information. Extending these ideas to holistic or mediated scenarios is possible, but lies beyond the scope of this dissertation.

2.2.4 Metrics Associated with Schema Matching

A core step in schema matching is estimating the similarity between two attributes $a \in \mathcal{S}_1$ and $b \in \mathcal{S}_2$, where $\mathcal{S}_1$ and $\mathcal{S}_2$ denote the schemas of two tabular datasets. The similarity score $\text{sim}(a, b) \in [0, 1]$ reflects the likelihood that the two attributes refer to the same underlying concept. This section presents the main families of similarity metrics used in schema matching, categorized according to the type of information they exploit: syntactic (name-based), value-based (instance-level), type-based, embedding-based, and statistical.

I. Attribute-Based Metrics

These metrics rely on the syntactic similarity between attribute names, often treating names as strings or token sequences.

- **Edit Distance (Levenshtein)** [71]. Measures the minimum number of character-level insertions, deletions, and substitutions needed to transform one string into another. For example, the edit distance between `"cat"` and `"cut"` is 1 (one substitution: `'a'` → `'u'`).

- **Jaccard Similarity** [55]. Measures the similarity between two sets:

$$\text{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

  In *name-based matching*, A and B are the sets of tokens extracted from the attribute names. For example, comparing `"Full Name"` and `"Employee Name"`:

$$A = \{\text{"full"}, \text{"name"}\}, \quad B = \{\text{"employee"}, \text{"name"}\}$$

  We have $A \cap B = \{\text{"name"}\}$ and $A \cup B = \{\text{"full"}, \text{"employee"}, \text{"name"}\}$, so:

$$\text{Jaccard}(A, B) = \frac{1}{3}$$

- **N-gram Similarity** [116]. Measures the similarity between two strings based on the overlap of their character n-grams — that is, all contiguous substrings of length n.

  For example, consider the two attribute names `"Name"` and `"Names"`, using trigrams ($n = 3$):

$$\text{Trigrams}(\text{"Name"}) = \{\text{"Nam"}, \text{"ame"}\}, \quad \text{Trigrams}(\text{"Names"}) = \{\text{"Nam"}, \text{"ame"}, \text{"mes"}\}$$

  The sets share two trigrams: $\{\text{"Nam"}, \text{"ame"}\}$. Using Jaccard similarity on these sets:

$$A = \{\text{"Nam"}, \text{"ame"}\}, \quad B = \{\text{"Nam"}, \text{"ame"}, \text{"mes"}\}$$
$$\text{N-gram Similarity} = \frac{|A \cap B|}{|A \cup B|} = \frac{2}{3}$$

- **TF-IDF with Cosine Similarity** [104]. Each attribute name is treated as a short document, and converted into a TF-IDF vector based on token frequency and inverse document frequency over the dataset. The resulting sparse vectors are then compared using cosine similarity:
$$\cos(v_1, v_2) = \frac{v_1 \cdot v_2}{\|v_1\| \cdot \|v_2\|}$$
where v_1 and v_2 denote the TF-IDF vectors of the two attribute names.

  For example, Comparing `"Full Name"` and `"Employee Name"`, the shared token `"Name"` contributes strongly to the similarity, while `"Full"` and `"Employee"` may contribute little or nothing, depending on corpus frequency.

- **SoftTF-IDF** [26]. A variant of TF-IDF that accounts for approximate matches between tokens. Instead of requiring exact token overlap, SoftTF-IDF gives partial credit for similar tokens (e.g., "Names" and "Name"). It combines TF-IDF weighting with a secondary string similarity (like Jaro-Winkler or Edit Distance) for token matching.

  For example, Comparing `"Last Names"` and `"Surname"` might yield a low score with standard TF-IDF, but SoftTF-IDF may increase the score by recognizing that `"Names"` and `"Surname"` are semantically or string-wise similar.

- **Monge-Elkan Similarity** [84]: This method compares two multi-word attribute names by looking for the best matches between individual words (tokens) in the names, and then averaging those best scores.

  For example, compare the attribute names `"Full Name"` and `"Employee Name"`.

  - Tokens in the first string: {Full, Name} - Tokens in the second string: {Employee, Name}

  The best matches between the tokens could be: - `"Full"` vs. `"Employee"` → low similarity (e.g., 0.1) - `"Name"` vs. `"Name"` → perfect match (1.0)

  Monge-Elkan computes the average of the best similarities:
$$\text{MongeElkan} = \frac{0.1 + 1.0}{2} = 0.55$$

- **Damerau–OSA (Optimal String Alignment)** [28, 117]. Extends Levenshtein distance by also allowing adjacent transpositions of characters.

  For example, comparing `"ca"` vs. `"ac"`: Levenshtein = 2 (substitution + insertion), Damerau–OSA = 1 (one transposition).

- **Longest Common Subsequence (LCS)** [91]. Measures the longest ordered sequence of characters common to both strings. The normalized version divides by the maximum string length:
$$\text{LCS-sim}(s_1, s_2) = \frac{\text{LCS}(s_1, s_2)}{\max(|s_1|, |s_2|)}$$
For example, for `"customer"` (8 chars) and `"consumer"` (8 chars), the LCS is `"c, s, t, m, e, r"` of length 6. Normalized score = $6/8 = 0.75$.

- **Common Prefix and Suffix Ratios**. Measure the proportion of matching characters from the beginning (prefix) or end (suffix) of two strings.

  For example, for `"international"` vs. `"internet"`, prefix length = 6, ratio = $6/12 = 0.5$. For `"manager"` vs. `"danger"`, suffix length = 5, ratio = $5/7 \approx 0.71$.

- **Exact Equality (case-sensitive / case-insensitive)**. A binary indicator checking whether two names are strictly identical, or identical up to case-folding.

  For example, `"Name"` vs. `"Name"` $\rightarrow 1$ (exact); `"Name"` vs. `"name"` $\rightarrow 1$ (case-insensitive); `"Name"` vs. `"Names"` $\rightarrow 0$.

- **Substring Checks**. A binary heuristic that checks whether one name occurs as a substring of the other.

  For example, `"birth"` in `"birthdate"` $\rightarrow 1$; otherwise 0.

- **Dice Coefficient** [34]. Measures token or n-gram overlap:

$$\text{Dice}(A, B) = \frac{2|A \cap B|}{|A| + |B|}$$

  For example, for bigrams of `"data"` ({"da", "at", "ta"}) vs. `"date"` ({"da", "at", "te"}), $|A \cap B| = 2$, $|A| + |B| = 6$, so Dice $= 2 \cdot 2/6 = 0.67$.

- **Cosine Similarity on n-gram counts** [103]. Represents each string as an n-gram frequency vector. Similarity is then measured with cosine similarity.

  For example, for bigrams of `"night"` and `"nacht"`, the overlap of {`"n"`, `"t"`} yields a partial score reflecting shared structure.

- **Jaro Similarity** [56]. Measures similarity by considering matching characters within a search window and penalizing transpositions. Values range in $[0, 1]$.

  For example, `"MARTHA"` vs. `"MARHTA"` $\rightarrow$ Jaro $= 0.944$, since characters mostly align with one transposition.

- **Jaro-Winkler Similarity** [119]. An extension of Jaro that gives more weight to common prefixes. For example, `"MARTHA"` vs. `"MARHTA"` $\rightarrow$ Jaro-Winkler $= 0.961$, slightly higher than Jaro due to the shared prefix `"MAR"`.

II. Value-Based Metrics

These Metrics compare the actual values observed in the columns, treating them as sets, distributions, or samples.

- **Kolmogorov–Smirnov Distance (KS Distance)**. A statistical measure used to compare the distributions of two numerical column values. It calculates the maximum difference between their cumulative distribution functions (CDFs).

  For example, suppose column a contains salaries from dataset A, and column b contains salaries from dataset B. The KS distance measures how much the two distributions differ overall — for instance, whether one tends to have higher values than the other. A value close to 0 means the distributions are very similar; a value close to 1 means they differ significantly.

- **Earth Mover's Distance (EMD)**. Measures how different two distribution values are by calculating the minimum cost of transforming one into the other. The idea comes from transporting "mass" (like piles of dirt) to match one shape to another, where the cost depends on how far the mass must be moved.

  For example, suppose column a contains product prices from one dataset, and column b contains prices from another. If one column mostly contains prices around \$10 and the other around \$50, the EMD will be high, as a greater cost is required to align the two distributions. Conversely, if both distributions are similar (e.g., centered around \$30), the EMD will be low.

- **Distributional Statistics**. This approach compares basic statistical properties of the values in each column, such as the mean, variance, histogram shape, or entropy. It allows similarity to be achieved when columns follow similar data distributions, even if their raw values differ.

  For example, suppose column a contains ages of customers from dataset A and column b contains ages from dataset B. If both columns have a similar average age (e.g., 35), standard deviation, and overall distribution shape (e.g., bell curve), then they are likely to refer to the same concept.

III. Type-Based Metrics

These methods compare the data types or semantic categories of two attributes to estimate how compatible they are.

- **Exact Type Match**. Returns 1 if the data types of both columns are exactly the same (e.g., both are of type `int`), and 0 otherwise.

- **Type Compatibility Score**: Assigns a partial similarity score to types that are not exactly the same but are still compatible. For example, an `int` and a `float` may be considered more similar than an `int` and a `string`, because both are numeric.

- **Semantic Type Annotation**. Instead of relying on low-level data types, this approach assigns higher-level semantic types (or categories) to each column using external tools, ontologies, or pre-trained models. These types represent real-world concepts like `"City"`, `"Country"`, `"Person"`, or `"Date"`.

  Example: A column containing values like `"London"`, `"Paris"`, and `"Berlin"` might be annotated as `"City"`, while another with `"France"`, `"Germany"` as `"Country"`. Semantic similarity can then be computed between these high-level types using a taxonomy or ontology (e.g., WordNet or DBpedia). Columns sharing the same semantic type receive high similarity scores.

IV. Embedding-Based Metrics

These techniques represent attributes (either column names or values) as dense vectors in a high-dimensional space $\mathbb{R}^d$. Let $v_1 = (v_1^1, v_1^2, \ldots, v_1^d)$ and $v_2 = (v_2^1, v_2^2, \ldots, v_2^d)$ be two such vectors, where the index i refers to the i-th dimension of the representation.

The following metrics can be applied:

- **Cosine Similarity**.

$$\cos(v_1, v_2) = \frac{v_1 \cdot v_2}{\|v_1\| \cdot \|v_2\|}$$

Measures the angle between the two vectors. Values range from -1 (opposite) to 1 (identical direction).

For example, $v_1 = [1, 2],\ v_2 = [2, 3] \Rightarrow \cos(v_1, v_2) \approx 0.99$

- **Euclidean Distance**.
$$d(v_1, v_2) = \sqrt{\sum_i (v_1^i - v_2^i)^2}$$
Measures the L2 distance (straight-line) between vectors.
For example, $\sqrt{(1-2)^2 + (2-3)^2} = \sqrt{2} \approx 1.41$

- **Minkowski Distance (with $p = 3$)**. The parameter p specifies the order of the norm, thereby determining the importance given to larger coordinate differences in the overall distance.
$$d(v_1, v_2) = \left(\sum_i |v_1^i - v_2^i|^3 \right)^{1/3}$$
A generalization of Euclidean and Manhattan distances.
For example, $\left(|1-2|^3 + |2-3|^3\right)^{1/3} = (1+1)^{1/3} \approx 1.26$

- **Canberra Distance**.
$$d(v_1, v_2) = \sum_i \frac{|v_1^i - v_2^i|}{|v_1^i| + |v_2^i|}$$
where by convention $\frac{0}{0} = 0$.
For example, $\frac{|1-2|}{1+2} + \frac{|2-3|}{2+3} = \frac{1}{3} + \frac{1}{5} \approx 0.53$

- **Chebyshev Distance**.
$$d(v_1, v_2) = \max_i |v_1^i - v_2^i|$$
Measures the maximum difference across any dimension.
For example, $\max(|1-2|, |2-3|) = 1$

V. Statistical Correlation-Based Metrics

These metrics quantify the degree of association between two numerical vectors, typically representing either column values or column-level embeddings. They are particularly useful when both attributes contain numeric or ordinal data.

- **Pearson Correlation Coefficient**. Measures the strength and direction of the linear relationship between two numerical vectors v_1 and v_2.
$$\rho(v_1, v_2) = \frac{\mathrm{Cov}(v_1, v_2)}{\sigma_{v_1} \sigma_{v_2}}$$
where $\mathrm{Cov}(v_1, v_2)$ is the covariance between v_1 and v_2, and $\sigma_{v_1}, \sigma_{v_2}$ are their standard deviations.

- **Spearman Rank Correlation**. Measures the monotonic (increasing or decreasing) relationship between two variables based on their ranks, rather than actual values. This makes it robust to outliers and non-linear but monotonic trends.

$$\rho_s = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

where d_i is the difference between the ranks of the i-th values in the two columns, and n is the number of observations.

Note: Pearson correlation captures strictly linear relationships, whereas Spearman correlation captures monotonic trends (linear or non-linear).

Individually, each family of metrics captures only a fragment of the similarity —lexical measures overlook semantics, distributional ones ignore labels, and embedding-based often collapse to simple cosine comparisons. Yet, their union could prove far more powerful, especially under disparate conditions (e.g., schema-only or instance-only datasets), where relying on a single metric is rarely sufficient. It therefore remains an open and promising direction to study how these metrics can be combined and adapted to asymmetric scenarios, rather than treating them in isolation.

2.2.5 Applications of Tabular Schema Matching

In this section we elaborate on how the output of schema matching can facilitate solutions to other important problems, in settings where datasets come in the form of tables. Specifically, we investigate the utilization of schema matching in the following three applications: *i*) Entity Resolution [24, 47], *ii*) Data Cleaning [101, 25], and *iii*) Data Augmentation [41, 93].

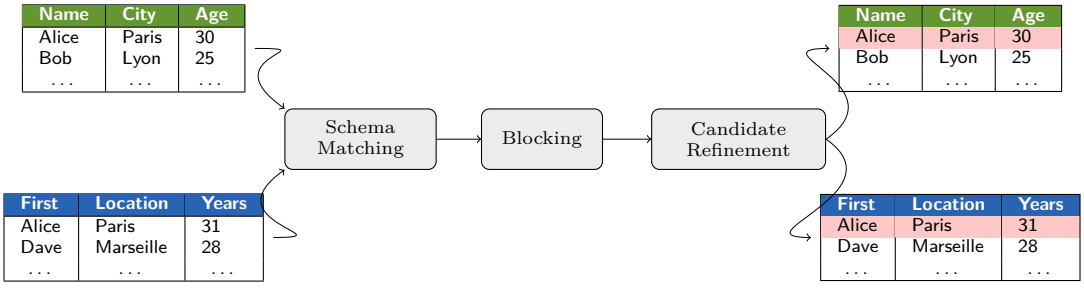


Figure 2.3: An example of an entity resolution pipeline applied to a pair of tables. The process starts with schema matching to align corresponding columns, followed by blocking and candidate refinement, ultimately producing the final set of matched tuple pairs.

1. Schema Matching for Entity Resolution. Entity Resolution[1] [47] addresses the challenge of identifying records across datasets that refer to the same real-world entities. In the context of tabular data, this means finding tuples in different tables that describe the same object. As illustrated in Figure 2.3, schema matching is typically a foundational step in any

[1]We regard Entity Resolution, Entity Matching, Record Matching, and Deduplication as research problems of the same nature, where Schema Matching plays a similarly crucial role.

entity resolution pipeline. It enables the blocking of tuple pairs that are unlikely to match [94], thus reducing the search space, and ultimately supports the identification of candidate matches through subsequent refinement stages.

When tuples are viewed as entities, columns can be seen as attributes that characterize them. Therefore, schema matching allows the meaningful comparison of tuples across tables by aligning columns that describe the same attribute. The accuracy of an entity resolution method depends heavily on the quality of these column matches: incorrect or missed matches can lead to poor resolution outcomes, including both false positives and false negatives.

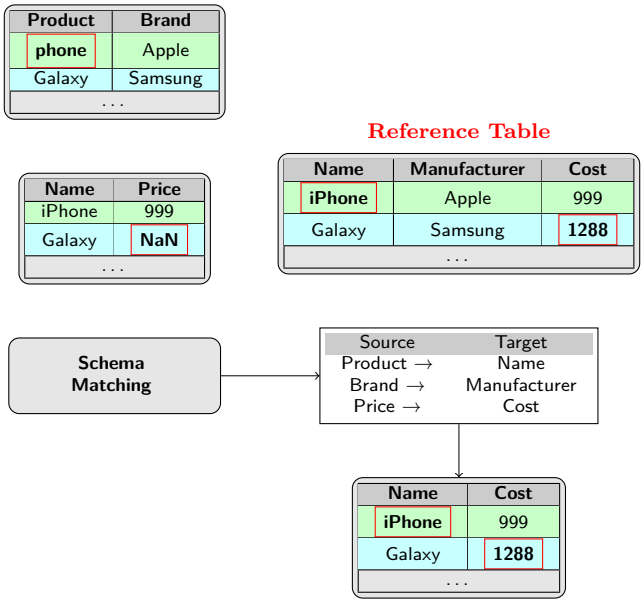


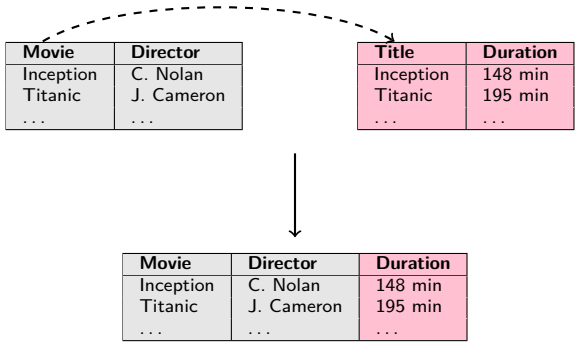
Figure 2.4: Schema matching can facilitate error detection and repair by aligning columns with a reference table containing ground truth values. This enables the identification and correction—or even imputation—of missing or incorrect data in other tables.

2. Schema Matching for Data Cleaning. Data cleaning is a well-established area that focuses on identifying and repairing errors in datasets [25]. For tabular data, common issues include misspellings, incorrect values, or missing entries. Schema matching can support data cleaning when a trusted reference table is available — that is, a table known to contain accurate and complete information.

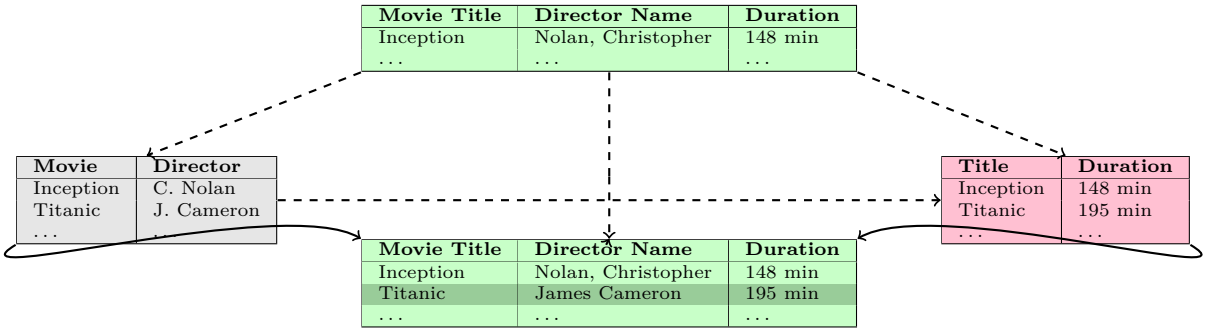
As shown in Figure 2.4, schema matching aligns columns from noisy or incomplete tables with the corresponding columns in a clean reference table. This enables both the detection of incorrect values and the completion of missing data. For example, the value `phone` in the `Product` column can be corrected to `iPhone` by aligning it with the `Name` column of the reference table. Similarly, a missing `Price` value for `Galaxy` can be completed using the aligned `Cost` value (1288) from the reference row.

This process reflects a mediated schema matching scenario, where the reference table defines a consistent global view. To apply such corrections accurately, schema matching must be coupled with entity resolution — i.e., detecting that two rows referring to "Galaxy" correspond to the same product. Once aligned, values can be copied or reconciled across tables.

Such mappings also support building dictionaries for value normalization or standardization (e.g., unifying naming conventions or formats). In this way, schema matching contributes not only to structural integration but also to improving overall data quality.



(a) Augmenting Columns



(b) Augmenting Tuples

Figure 2.5: Two illustrations of table augmentation via column alignment: (a) matching columns between two tables allows joining them to enrich the original with additional attributes; (b) leveraging column matches across a repository of tables can be used to add new tuples to the target table.

3. Schema Matching for Data Augmentation. In practice, data scientists often rely on tabular datasets to train predictive models and extract insights. However, these datasets frequently suffer from limitations: they may lack important attributes (i.e., features) or contain too few tuples to support robust model performance. To overcome this, data augmentation techniques [121, 124, 126, 127] aim to enrich a given table by adding new columns, additional tuples, or both.

Schema matching plays a central role in this process. Column correspondences between tables can indicate viable join operations, allowing one table to be augmented with additional attributes drawn from another. This is illustrated in Figure 2.5a, where a table of movies and directors is

extended with a `Duration` column sourced from a related table via matching `Movie` and `Title`.

Beyond attribute expansion, schema matching can also facilitate tuple augmentation. When a table shares column correspondences with multiple others in a repository, matched rows from these sources can be merged into the original dataset. As shown in Figure 2.5b, information from several related tables is integrated to enrich both the schema and the content of the base table [70].

Such schema-guided augmentations have been demonstrated to improve the predictive performance of machine learning models, without modifying model architectures [22, 129]. Moreover, dataset discovery systems that retrieve related tables for a given one [88, 29, 12, 9] often rely on schema matching as a core component to enable these augmentation opportunities.

Across entity resolution [24, 47], data cleaning [101, 25], and data augmentation [41, 121, 127], a common limitation is the implicit assumption that reliable schema correspondences are available. In practice—particularly with web tables, data lakes, or open collections [13, 87]—schemas are often incomplete, noisy, or asymmetric (e.g., schema-only or. instance-only). Under such conditions, classical pipelines become vulnerable: ER fails to compare tuples meaningfully, cleaning cannot leverage reference tables, and augmentation risks propagating noise instead of adding value. This recurring weakness motivates our focus on schema matching methods that remain robust under disparate inputs, ensuring downstream tasks can operate reliably even in incomplete scenarios.

2.3 Traditional Schema Matching Techniques

In this section, we answer to the question of how relevance among columns is captured, by summarizing related work on tabular schema matching. To guide the discussion, we introduce the main categories under which schema matching methods fall, based on the information they exploit [100]. Particularly, we distinguish three categories, as shown in Figure 2.6, which we define below together with a brief discussion of related methods in the literature [65].

2.3.1 Schema-Based Approaches

Schema-based schema matching techniques operate solely on metadata-level information—such as attribute names, data types, column positions, or structural relationships—without relying on instance-level data. These methods are particularly useful when dealing with sensitive, large-scale, or incomplete datasets where values may be inaccessible or unreliable. Although limited in expressiveness, schema-based approaches offer efficiency, interpretability, and applicability in early-stage data integration [100, 41, 109]. Figure 2.6a illustrates a limitation of this approach: relying solely on column names makes it hard to determine whether Title and Movie refer to the same concept. However, the surrounding table names can provide valuable contextual clues—suggesting that both columns likely contain movie titles. Throughout this subsection, categories are organized from older to more recent approaches, progressing from syntactic and structural features to logical constraints, external semantic knowledge, and LLMs.

Rule- and Structure-Based Methods. Early schema matchers rely on string-based and syntactic similarity functions applied to column names and schema labels. Foundational systems such as *Cupid* [78], *SEMINT* [72], and *COMA/COMA++* [36, 4] model schemas as trees or graphs and combine linguistic, structural, and constraint-based similarity scores in configurable workflows. *SEMINT*, for instance, uses neural networks to identify attribute correspondences based on metadata properties. Other studies have evaluated the effectiveness of such approaches in controlled settings [35], highlighting the variability of matching quality across domains. Extensions such as *S-Match* [48] enrich this process with logic-based reasoning and ensemble matching

(a) Schema-based Matching

MovieID	Title	Duration
FH12	Dune	896
SK43	It	1138
...	...	...

Ref	Director	Movie
FHD	F. Herbert	Dune
GO1	G. Orwell	1984
...	...	...

(b) Instance-based Matching

MovieID	Title	Duration
FH12	Dune	896
SK43	It	1138
...	...	...

Ref	Director	Movie
FHD	F. Herbert	Dune
GO1	G. Orwell	1984
...	...	...

(c) Hybrid Matching

MovieID	Title	Duration
FH12	Dune	896
SK43	It	1138

Ref	Director	Movie
FHD	F. Herbert	Dune
GO1	G. Orwell	1984

Figure 2.6: The main categories of schema matching methods on tabular data: (a) Schema-based methods, (b) Instance-based matching approaches, (c) Hybrid matching methods.

techniques. Other approaches such as POMap++ explore local match learning [67]. These approaches remain effective when schema metadata is well-structured and adheres to standard naming conventions.

Constraint-Aware Methods. Some schema-based techniques incorporate relational constraints such as keys, foreign keys, and inclusion dependencies—to improve alignment decisions. A well-known example is *Similarity Flooding* [81], which propagates similarity over schema graphs using fixpoint computation. More recent work, such as *Adnev* [108], applies learned adjustments to schema similarity matrices based on relational context. Constraint-aware variants of *COMA++* [79] also integrate cardinality, key, and structural features to guide matching in complex relational or XML-based schemas. Additionally, efforts like(Castano et al., 2001) [16] explored global views of heterogeneous data sources, providing schema-level abstraction to facilitate matching across distributed systems. These methods remain relevant, but their reliance on well-defined relational metadata often restricts applicability in unstructured or incomplete data sources.

Ontology-Enriched Approaches. To overcome the limitations of purely syntactic similarity, several schema matchers augment their reasoning with external knowledge sources. Systems like *YAM++* [92], *AgreementMaker* [27], and *LogMap* [57] use WordNet, Wikipedia, or curated ontologies to match semantically related terms even when lexical similarity is low. In ontology-rich domains, methods such as *PARIS* [113], *DeepAlignment* [58] exploit concept hierarchies, embeddings, or logical constraints to align entities and properties based on schema-level semantics alone. More recent works include holistic alignment strategies by Megdiche et al. [80] and domain-adaptive ontology matching by Laadhar et al. [68], which extend scalability and adaptability.

LLM-Based Schema Matching. More recently, large language models (LLMs) have been leveraged to perform schema-only alignment by interpreting attribute names, table context, or documentation. These systems prompt pretrained models to reason over column semantics using only metadata. Notable examples include the evaluation by *Parciak et al.* [96], as well as purpose-built systems like *Matchmaker* [21], *ReMatch* [107], *LLMATCH* [118], and *Buss et al.* [11]. Related LLM-based ontology matching approaches include *OLaLa* [53] and *LLMs4OM* [5].

Overall, schema-based techniques remain valuable for scalable and privacy-sensitive settings, but their accuracy depends heavily on metadata quality. When combined with external semantics or pretrained models, these methods can bridge part of the gap toward robust, domain-agnostic matching. Nonetheless, they tend to underperform in open-domain or noisy environments lacking structured and informative schema labels.

Schema-based methods mostly address scenarios (C1, C2, D4), with similarity spaces limited to lexical and structural cues. They provide important baselines but cannot operate effectively under instance-less datasets, such as in scenarios (C3, D5, D6), which motivates the exploration of hybrid, embedding-based, and LLM-enhanced approaches.

2.3.2 Instance-Based Approaches

Instance-based schema matching techniques identify correspondences between attributes by analyzing the values stored within columns. These methods are particularly valuable when schema metadata—such as column names or data types—is missing, ambiguous, abbreviated, or inconsistent [59, 33, 38, 90, 7]. By leveraging instance-level evidence such as value co-occurrence, statistical distributions, or token overlap, they uncover semantic relationships that are not detectable from schema-level attributes alone. For example, as illustrated in Figure 2.6b, the similarity between columns *Title* and *Movie* may be inferred from shared cell values even when their schemas are ambiguous.

Rather than comparing raw values directly, some approaches compute similarity based on the statistical distribution of attribute contents. *Bilke and Naumann* [7] propose clustering columns by their frequency distributions to detect attribute correspondences. *Sema-Join* [51] exploits large corpora of relational tables to identify commonly co-occurring attribute types. *EmbDI* [14] creates embeddings for attributes based on their co-occurrence with other columns across datasets, capturing distributional semantics without relying on labels. *SEMINT* [72] takes a supervised approach, using neural networks to infer attribute semantics from value features.

Despite their strengths, instance-based methods can be resource-intensive, particularly when applied to high-dimensional or large-scale datasets. While instance-based methods are more expressive than schema-only techniques, they critically depend on the availability of sufficient and reliable values. In schema-less datasets or disparate scenarios (C2, D4, D6), their performance collapses. Preprocessing, normalization, and sampling strategies are often essential to reduce noise and enable tractable comparisons. Nevertheless, instance-based matching remains indispensable when schema metadata is unreliable or unavailable, and it forms a crucial component

in hybrid and learning-based systems.

2.3.3 Hybrid-Based Approaches

Hybrid schema matching methods combine both schema-level and instance-level information to identify correspondences between attributes. Instead of relying on a single type of evidence—such as column names or cell values—these methods integrate multiple forms of similarity (e.g., names, data types, value overlap, structural roles) to improve robustness and accuracy. This makes them particularly effective in real-world datasets, where metadata may be incomplete, inconsistent, or noisy. For example, in Figure 2.6c, a hybrid matcher that begins with schema-based filtering might incorrectly discard a valid correspondence between the columns Title and Movie due to low name similarity. In contrast, a hybrid method that emphasizes instance-level evidence could correctly identify the match based on value similarity.

In principle, hybrid methods can outperform schema-only or instance-only approaches, since they use a richer set of clues. In practice, however, their effectiveness depends on how different types of information are weighted and combined. For example, a system that filters candidates using only column names may miss valid matches where instance values provide strong evidence. On the other hand, relying too much on noisy or sparse data values can also lead to errors. Hybrid systems must therefore combine multiple features while managing efficiency and scalability.

Modular and Configurable Architectures. Several foundational systems are designed as modular frameworks that allow combining diverse matchers with tunable parameters. A prime example is *COMA++* [4, 79], which provides an extensible matcher library, customizable workflows, and support for both schema- and instance-based features. Similarly, *AML (AgreementMakerLight)* [44] integrates lexical, structural, and logical matchers with a repair-based alignment pipeline. *YAM++* [92] offers automatic matcher selection based on schema profiles and is tailored for ontology matching. *AgreementMaker* [27] also exemplifies hybrid design with scalable performance on large ontologies. Recently, *Matchmaker* [105] applies self-improving LLM-based agents to coordinate hybrid matching strategies.

Learning-to-Match Frameworks. Machine learning has been widely applied to combine heterogeneous types of similarity into a unified matching decision. Early systems like *LSD* [39] train classifiers over hand-engineered features derived from both schema and instance-level content. *iMAP* [33] explores the full space of possible matchings using a set of similarity functions and heuristics to rank candidates. *Data Tamer* [112] adopts a similar strategy, treating matching as a classification task supported by multiple expert matchers. More recent systems such as *SAHM* [3], *SMUTF* [128], and *LLMATCH* [118] combine deep embeddings with logical rules, prompting or fine-tuning LLMs for schema matching tasks.

Search-Driven and Graph-Based Integration. Some hybrid systems go beyond pairwise matching and explore broader alignment possibilities through heuristic search or graph construction. *iMAP* [33] exemplifies exhaustive candidate space exploration. *Aurum* [17] builds knowledge graphs from relational corpora, linking attributes and tables via shared schema and instance-based patterns. Other systems construct matching graphs from co-occurring attributes or relational dependencies [124, 70] to improve coverage and enable match propagation across datasets. *Santos* [60] also uses graph techniques to support table union discovery at scale.

Semantic Embedding and Enrichment. Hybrid approaches increasingly integrate word or contextual embeddings with traditional matching features to capture deeper semantics. *DeepMatcher* [86] and *Ditto* [73] encode schema and instance representations with deep neural networks, using both structure and content. *BERTMap* [52] combines ontology-aware alignment with BERT-based contextual embeddings. *AI-Match* [50] uses a two-step approach where em-

beddings are combined with symbolic constraints. Ontology-enriched methods [126] further enhance matching accuracy by grounding representations in domain knowledge.

Table Search and Dataset Linking. Hybrid matching also plays a central role in systems for table discovery, augmentation, and integration. *WebTables* [12], *InfoGather* [121], and *InfoGather+* [124] use combinations of schema similarity, content overlap, and entity-based context to find related tables on the web. *EntiTables* [126] enhances entity-focused table search by modeling column semantics jointly. *Santos* [60], *Stitching* [70], and *FindingRelatedTables* [29] combine several approaches to support dataset discovery and table union. Recent work such as *DatasetDiscovery* [9] and *DataLakeMatching* [127] adapts hybrid matching to large and diverse data lake environments.

Challenges and Trade-offs. While hybrid matchers offer higher accuracy and flexibility, they raise practical concerns related to integration complexity, parameter tuning, and computational efficiency. Choosing how to combine or prioritize different types of information is non-trivial, and poor design may lead to noisy or incomplete results. A common risk involves rejecting valid matches if schema-based filters override useful instance-level cues—or the opposite, when unreliable values dominate. Effective hybrid matching therefore requires thoughtful design and sometimes domain-specific guidance. Nonetheless, hybrid approaches remain the most adaptable and powerful solutions for schema matching in practice.

Yet, despite this potential, most existing hybrid methods mostly address compliant scenarios (C1–C3) and do not explicitly address disparate scenarios (D4–D6). Moreover, while their similarity space is broader than that of purely schema- or instance-based approaches, it is still underexploited, with work exploring metrics beyond lexical and distributional ones.

2.4 Embedding-Based and LLM-Driven Approaches

Recent schema matching research has leveraged advances in representation learning and deep neural architectures to capture the latent semantics of schema elements. These approaches aim to overcome the limitations of traditional techniques by embedding tables into continuous vector spaces, enabling similarity computation based on learned contextual and distributional information.

Representation Learning for Schema Elements without LLMs

Prior to the emergence of large language models (LLMs), several embedding-based methods were proposed to learn semantic representations of schema elements in an unsupervised or self-supervised manner. These methods treat columns and their values as textual sequences or sets, and apply neural embedding techniques to derive fixed-length vector encodings that can be compared using distance metrics such as cosine similarity. EmbDI [14] introduces a self-supervised learning framework that creates column embeddings based on their co-occurrence with other columns across large corpora of tables. The method generates relational embeddings that reflect both syntactic structure and inter-column relationships. EmbDI is particularly effective in cold-start scenarios where schema labels are incomplete or noisy, as it relies on learned statistical regularities rather than fixed rules. Table2Vec is inspired by word embedding models, Table2Vec [30] constructs vector representations of entire tables and their columns using a combination of local context and global co-occurrence patterns. It captures both syntactic and relational semantics and is suitable for matching tasks in low-resource environments. ColNet [20] is designed for semantic column annotation. It leverages convolutional neural networks (CNNs) to learn column

representations from instances and combines them with knowledge base type information. While not a pure matcher, ColNet's embeddings can be repurposed for schema alignment tasks.

These representation learning methods significantly reduce the dependence on handcrafted similarity functions and can generalize across diverse domains. However, their effectiveness is constrained by the expressiveness of their architecture and the size of their training corpus. Compared to transformer-based models, they are less capable of capturing nuanced semantics or handling zero-shot matching scenarios. As a result, more recent work has shifted toward pretrained LLMs, which we discuss in the next subsection.

Classical Deep Representation Models (Pre-LLM Transformers)

Before the dominance of large language models, several deep learning-based models attempted to improve schema matching by applying neural networks to encode schema elements more expressively than traditional embeddings. DeepMatcher [86] applies BiLSTM encoders with pretrained word embeddings to represent and align attributes. It was among the earliest neural architectures to outperform traditional matchers on structured entity matching benchmarks. Ditto [73] adapts the RoBERTa transformer model for table-to-table entity matching. While based on pretrained transformers, Ditto is not considered an LLM-scale model and requires fine-tuning on target domains. Its performance bridges classical embedding models and the early wave of transformer-based alignment approaches. TaBERT [122] and TURL [31] represent structured table elements using transformer-based architectures trained on large corpora. They capture fine-grained table semantics by jointly modeling schemas, cell values, and inter-column relations. These models are pivotal in transitioning from word-level embedding approaches to structured-aware transformers.

Together, these deep representation learning models mark the convergence of traditional neural architectures and transformer innovations. They offer improved semantic sensitivity and representation fidelity but remain limited in zero-shot generalization and out-of-domain robustness. These limitations are addressed by the rise of LLM-powered matchers, discussed below.

Applications of Large Language Models (LLMs)

Contextual Enrichment: From Symbolic Knowledge to Generative Understanding. Before the emergence of large language models (LLMs), schema matching systems enriched attribute comparisons by relying on external symbolic knowledge—such as ontologies, taxonomies, and lexical resources. These tools helped detect semantic correspondences between schema elements that were lexically dissimilar, for instance by leveraging synonymy or hierarchical relationships. Examples include WordNet [82] for general-purpose enrichment and UMLS [8] in the biomedical domain.

However, such symbolic resources had several limitations: they required manual curation, lacked coverage for many domains, and were often inflexible when faced with noisy, emerging, or multilingual vocabularies.

LLMs have dramatically broadened the scope of contextual enrichment. Rather than relying on predefined taxonomies, LLMs can generate rich semantic information on demand—including type predictions, value summaries, label interpretations, and even natural language descriptions of a column's role in a table. For example, systems like BERTMap [52] and LLMATCH [118] use pretrained transformer models not only to encode schema elements, but also to infer their meaning, relate them to one another, or produce explanations for a match. Some matchers employ LLMs to generate prompts, candidate match descriptions, or clarifying context that was never explicitly provided in the data.

A recent evaluation by Parciak et al. [96] shows that LLMs outperform earlier enrichment approaches, especially in low-supervision and zero-shot scenarios. Unlike previous methods, which relied on static knowledge bases, LLMs offer generative capabilities—a key shift that enables dynamic interpretation and alignment of schema elements even in open or ambiguous settings.

This shift toward generative semantic enrichment introduces new opportunities—such as schema explanation, paraphrase generation, and schema translation—but also brings challenges: LLMs can be sensitive to prompt design, computationally expensive, and difficult to audit in terms of decision transparency.

Nevertheless, LLMs currently represent the most versatile and scalable tools for semantic schema matching—extending far beyond traditional embeddings to encompass contextual inference, linguistic generation, and adaptive reasoning. In particular, their zero-shot ability to generate semantic context without requiring rich schemas or extensive instance data makes them highly attractive for tackling asymmetric configurations, where traditional methods fail due to missing or incomplete modalities.

Fine-Tuning and Domain Adaptation. While general-purpose LLMs offer strong zero-shot performance for schema matching, specialized domains often require more precise control over terminology, semantics, and contextual cues. To address this, recent work has explored fine-tuning pretrained language models using domain-specific examples—real or synthetic—to adapt their behavior to particular sectors or vocabularies.

Liu et al. [76], for instance, propose generating synthetic training pairs for schema alignment in niche domains, enabling effective fine-tuning even in the absence of labeled data. Their method significantly improves matching accuracy in biomedical and business settings, where schema elements often involve specialized abbreviations or non-standard phrasing.

Domain-adapted LLMs like BioBERT[69] (for medical texts) or FinBERT[2] (for financial documents) provide pretrained representations grounded in domain-specific corpora. Combined with few-shot or task-specific fine-tuning, these models can align attributes with subtle semantic differences or sparse contextual cues—tasks where general models often struggle.

Importantly, fine-tuned models not only generate better embeddings, but also enhance higher-level capabilities: improved label interpretation, more accurate paraphrase generation, and more domain-aware prompt responses.

These advantages come with trade-offs:

- **Computational overhead:** Fine-tuning requires significant compute and careful hyperparameter tuning.
- **Data curation effort:** Domain-specific training data—real or synthesized—must be representative and carefully constructed to avoid bias or overfitting.

Despite these challenges, domain adaptation is particularly valuable for high-stakes applications such as clinical trials, regulatory submissions, or enterprise systems, where alignment errors can have real-world consequences.

In parallel, researchers are exploring hybrid symbolic-neural approaches that combine rule-based transparency with neural flexibility, as well as cost-aware prompting techniques that reduce resource usage while preserving LLM performance. Together, these directions aim to make LLM-based schema matching both more accurate and more accessible across domains. Our contribution focuses instead on schema matching under asymmetric and incomplete settings, leveraging pretrained models in zero-shot or lightweight adaptation modes. Fine-tuning remains an interesting and promising research direction, but is considered out of scope here and is left as a potential avenue for future work, depending on the results achieved in this thesis.

2.5 Limitations of Existing Approaches

Despite decades of progress in schema matching, most existing approaches remain ill-suited to the challenges posed by disparate tabular data—particularly when one or both datasets lack complete schema or instance information. Schema-based methods rely heavily on attribute names and structural cues, making them vulnerable in the presence of ambiguous, abbreviated, or domain-specific labels. Instance-based techniques, while powerful in principle, struggle when values are sparse, unstructured, or noisy, and typically require substantial overlap between datasets to infer reliable correspondences.

Hybrid methods attempt to overcome the limitations of schema- or instance-only techniques by combining both types of information. While effective in well-formed datasets, their performance degrades sharply in asymmetric scenarios—such as schema-only vs. instance-only tables—where one modality is missing. In other words, most existing methods do not explicitly account for *data disparity*, where structural and content information are unequally distributed across sources. This limitation is particularly problematic in modern data ecosystems such as data lakes, decentralized integration settings, and privacy-sensitive environments, where incomplete inputs are common.

In addition to modality asymmetry, another critical limitation lies in how similarity is computed. Many approaches rely on a single metric—typically cosine similarity—to compare string representations or embeddings. Although computationally efficient, these metrics capture only a narrow view of similarity and often overlook deeper structural, spectral, or topological properties of the data. As a result, matchers may fail to detect meaningful correspondences in noisy, heterogeneous, or low-context settings where more expressive similarity models are needed.
As summarized in Table 2.2, most systems support only the compliant configurations (C1–C3), while **disparate settings (D4–D6)** remain largely underexplored. Our framework is designed to operate effectively across both families: it handles compliant scenarios with robust enrichment and comparison strategies, and explicitly addresses disparate cases by simulating the missing modality (schema or instance) using LLMs. This enables more complete and meaningful alignment even when inputs are partial or asymmetric.

2.5.1 Coverage Across Input Configurations

As detailed in Chapter 1, schema matching tasks may arise under six canonical configurations, depending on the presence or absence of schema and instance data on each side. These fall into two categories:

- **Compliant settings**, where both sources expose the same type of information:

  (C1) **Complete** $(\mathcal{S}_1, \mathcal{I}_1) \leftrightarrow$ **Complete** $(\mathcal{S}_2, \mathcal{I}_2)$
  (C2) **Schema-only** $(\mathcal{S}_1, \varnothing) \leftrightarrow$ **Schema-only** $(\mathcal{S}_2, \varnothing)$
  (C3) **Instance-only** $(\varnothing, \mathcal{I}_1) \leftrightarrow$ **Instance-only** $(\varnothing, \mathcal{I}_2)$

- **Disparate settings**, where one side lacks a modality:

  (D4) **Complete** $(\mathcal{S}_1, \mathcal{I}_1) \leftrightarrow$ **Schema-only** $(\mathcal{S}_2, \varnothing)$
  (D5) **Complete** $(\mathcal{S}_1, \mathcal{I}_1) \leftrightarrow$ **Instance-only** $(\varnothing, \mathcal{I}_2)$
  (D6) **Schema-only** $(\mathcal{S}_1, \varnothing) \leftrightarrow$ **Instance-only** $(\varnothing, \mathcal{I}_2)$

The table below summarizes the support of recent and classical methods across these configurations. **We include only methods designed or evaluated for *tabular data***, since this thesis focuses exclusively on the tabular schema matching setting.

Table 2.2: Comparison of schema matching methods across input configurations (C1–D6).

Method	C1	C2	C3	D4	D5	D6
Cupid (2001)[78]	✓	✓		✓		
Similarity Flooding(2002)[81]	✓	✓		✓		
COMA++ (2002)[4]	✓	✓		✓		
iMap(2004)[33]	✓					
Ditto(2020)[73]	✓		✓		✓	
Adnev(2020)[108]	✓	✓		✓		
EmbDI(2021)[14]	✓		✓		✓	
SMAT(2021)[123]	✓	✓		✓		
AI-Match(2022)[50]	✓	✓	✓	✓	✓	
SAHM(2023)[3]	✓					
Parciak et al.(2024)[96]	✓	✓		✓		
ReMatch(2024)[107]	✓	✓		✓		
Matchmaker(2024)[105]	✓	✓		✓		
LLMATCH(2025)[118]	✓	✓		✓		
SMUTF(2025)[128]	✓					
Buss, C et al.(2025)[11]	✓	✓		✓		
Our - unsupervised (2025)	✓	✓	✓	✓	✓	✓
Our - supervised (2025)	✓	✓	✓	✓	✓	✓

2.5.2 Functional Comparison of Schema Matching Systems

Beyond input modality coverage, schema matchers vary significantly in their support for features such as enrichment strategies, embedding types, cardinality support, and supervision modes. The following table 2.3 summarizes these aspects:
Compared to prior work, our method uniquely supports asymmetric schema matching by using LLMs to generate missing information—such as plausible column names or example values. In the unsupervised setting, we combine the results of five complementary matchers: rule-based, type-based, instance-based, hybrid, and schema-based. In the supervised setting, we compute a wide range of geometric meta-features from LLM embeddings and train a classifier to predict matches.

To overcome the above limitations, we propose a unified schema matching framework that:

- Uses large language models to **simulate the missing modality**—generating values from schemas or schemas from values.
- Builds a wide set of **meta-features** capturing similarity across classical, spectral, and topological perspectives.
- Supports both **unsupervised and supervised modes**, allowing flexible deployment across real-world datasets with varying constraints.

Unlike most systems that require clean and similar inputs on both sides, our method is designed to handle highly unbalanced scenarios—such as schema-only with. instance-only (D6). It remains effective even when column names are unclear, values are sparse, or formats differ significantly—where many existing methods break down.

Table 2.3: Comparison of schema matching methods across embeddings, enrichment, cardinality support, and learning mode.

Method	Schema-based	Instance-based	Hybrid-based	Local Emb	LLM Emb	Know. Enr.	Constr Based	LLM Enr.	**1:1**	**1:n**	**n:1**	**Sup.**	**Unsup.**
Cupid (2001)[78]	✓	–	–	–	–	✓	✓	–	✓	✓	–	–	✓
Similarity Flooding(2002)[81]	✓	–	–	–	–	–	–	–	✓	–	–	–	✓
COMA++ (2002)[4]	✓	*Opt*	–	–	–	✓	✓	–	✓	–	–	✓	✓
iMap(2004)[33]	–	–	✓	–	–	✓	✓	–	✓	✓	✓	–	✓
Ditto(2020)[73]	–	–	✓	–	✓	✓	–	–	✓	–	–	✓	–
Adnev(2020)[108]	✓	–	–	–	–	✓	✓	–	✓	–	–	✓	–
EmbDI(2021)[14]	–	✓	✓	✓	*Opt*	–	–	–	*Opt*	✓	–	*Opt*	✓
SMAT(2021)[123]	✓	–	–	✓	–	–	–	–	✓	–	–	✓	–
AI-Match(2022)[50]	✓	✓	✓	✓	–	–	✓	✓	✓	–	–	✓	–
SAHM(2023)[3]	–	–	✓	✓	–	✓	✓	–	✓	–	–	✓	✓
Parciak et al.(2024)[96]	✓	–	–	–	–	✓	–	✓	✓	✓	✓	–	✓
ReMatch(2024)[107]	✓	–	–	–	✓	–	–	–	✓	✓	✓	–	✓
Matchmaker(2024)[105]	✓	–	–	–	✓	–	✓	✓	✓	–	–	✓	✓
Magneto(2024)[75]	–	–	✓	–	✓	–	–	✓	✓	–	–	✓	✓
LLMATCH(2025)[118]	✓	–	–	–	✓	–	–	✓	✓	–	–	–	✓
SMUTF(2025)[128]	–	–	✓	–	✓	–	✓	✓	✓	–	–	✓	–
Buss, C et al.(2025)[11]	✓	–	–	–	✓	–	✓	✓	✓	–	–	✓	✓
Our - unsupervised (2025)	✓	✓	✓	–	✓	–	✓	✓	✓	–	–	–	✓
Our - Supervised(2025)	✓	✓	✓	–	✓	–	–	✓	✓	–	–	✓	–

2.6 Conclusion and Discussion

This chapter has outlined the evolution of schema matching approaches and the conceptual and empirical limitations of current systems. We began with traditional rule-based and constraint-driven systems, which offer simplicity and interpretability but struggle in the face of label ambiguity, structural heterogeneity, and incomplete metadata.

Instance-based and hybrid methods partially mitigate these issues by leveraging data values in addition to schema elements. However, they typically require sufficient value overlap or co-occurrence patterns to infer reliable matches—making them ineffective in sparse or noisy tables, where missing values or inconsistencies distort statistical or distributional evidence.

The introduction of embeddings and neural methods marked a major shift. Vector-based techniques, such as word and table embeddings, enable semantic matching beyond lexical similarity, supporting generalization across domains and formats. Yet, these models often depend on symmetric inputs and remain limited by their reliance on a narrow set of similarity metrics, most commonly cosine similarity. This narrow **similarity space** fails to capture more expressive properties such as structural alignment, geometric form, or topological invariants.

Large Language Models (LLMs) have emerged as a powerful alternative for semantic schema matching. These models not only provide contextualized representations of schema elements but also allow for flexible enrichment via prompt-based generation and reasoning. LLM-based techniques support zero-shot and few-shot matching, and in many cases outperform prior methods—especially in open-domain or low-resource settings. However, their effectiveness still varies with prompt design and domain specificity.

Our analysis highlights two fundamental limitations in existing work, consistently confirmed throughout this chapter: (i) a lack of support for *disparate input scenarios*—where schema and instance data are asymmetrically distributed between tables. In such settings, classical matchers collapse because they are restricted to compliant (C1–C3) cases and cannot operate when one modality is missing (D4–D6). (ii) an over-reliance on narrow similarity metrics. Most systems stay confined to lexical, distributional, or cosine-based comparisons, which provide only partial evidence of equivalence.

To address these gaps, our framework is explicitly designed around the two criteria introduced in

Section 1: - *Input coverage*: we cover both compliant (C1–C3) and disparate (D4–D6) settings by using LLMs to simulate the missing modality. - *Similarity space*: we extend beyond standard lexical and cosine metrics by constructing a **meta-space** that integrates classical, statistical, spectral, and topological meta-features.

These innovations enable schema matching to remain robust in *context-poor* environments (e.g., schema-only or instance-only), as well as in *noisy* data lakes and heterogeneous repositories, where incomplete or ambiguous information would otherwise undermine traditional methods.

Chapter 3

Schema Matching for Disparate Data Sources

3.1 Overview

In the face of increasingly disparate and incomplete tabular data, conventional schema matching techniques struggle to maintain performance due to the missing schema or instance information. This thesis addresses these limitations by proposing a unified framework that combines semantic enrichment to improve alignment across disparate data sources.

This thesis advances two complementary approaches to schema matching. Both begin with a common preprocessing stage that enriches disparate data sources with contextual information. When annotations are available, we apply a supervised matcher; in their absence, we adopt an unsupervised alternative that does not rely on ground-truth labels. To pursue these objectives, we introduce a unifying framework, outlined in Figure 3.1. Data sources—whether complete, schema-only, or instance-only—first pass through an *enrichment phase* that augments each source with missing attribute names and/or plausible instance values. The enriched sources are then aligned using one of the two approaches described below. The framework developed in this thesis proceeds as follows:

1 **Enrichment through Context Generation.** Leveraging the contextual reasoning capabilities of Large Language Models (LLMs) to infer plausible instances for schema-only tabular data and to predict likely attribute names for instance-only tabular data. This process generates enriched representations that transform disparate sources ($\mathcal{DS}_1$ and $\mathcal{DS}_2$) into a complete structure with more complete data ($\mathcal{CS}_1$ and $\mathcal{CS}_2$), enabling matching.

2 **Embedding-Based Matching.** The enriched tabular data is then encoded into dense vector representations using a pre-trained embedding model. These embeddings capture semantic similarities in a shared latent space, facilitating comparison between attributes across different sources. The resulting vector representations serve as the input for the subsequent schema matching phase. We explore two alternative matching frameworks:

   - **Unsupervised Schema Matching.** Pre-matchers are applied to the enriched data sources in order to identify and filter out incompatible attribute pairs (e.g., incompatible types like string and number). In this thesis, we use the Type Matcher as a pre-matcher for this initial comparison, which checks data types and formats from

metadata, the Rule Matcher validates alignment using constraints (e.g., ranges, regular expressions), the Schema Matcher aligns schema elements, the Instance Matcher compares instance values, and the Schema-Instance matcher combines schema and instance information for hybrid alignment. Each matcher produces a similarity matrix representing the confidence of alignment. Finally, the decision process aggregates the similarity matrices into a unified matrix (M_{sim}). Thresholding using the median is then applied to produce the final matched attribute pairs (M_f) and their corresponding similarity scores. The final output consists of a set of matched index pairs (i, j) representing correspondences between attributes from the two enriched input sources.

- **Supervised schema matching.** A set of meta-features is computed between the enriched data vectors, including: (i) *classical meta-features* such as cosine and Euclidean distances; (ii) *spectral meta-features* such as singular values and stable rank; and (iii) *topological meta-features* such as persistent homology and entropy, which capture geometric and semantic properties in high-dimensional space. These meta-features define a unified meta-space M_{ms}, *yielding* $h = 112$ *meta-features per attribute pair.* We train a classifier on M_{Train}, where each pair of attributes is represented by a meta-feature vector and a label. The model is then used to predict matches on unseen pairs in M_{Test}. The predicted alignment set consists of index pairs (i, j) between columns from the data source pair.

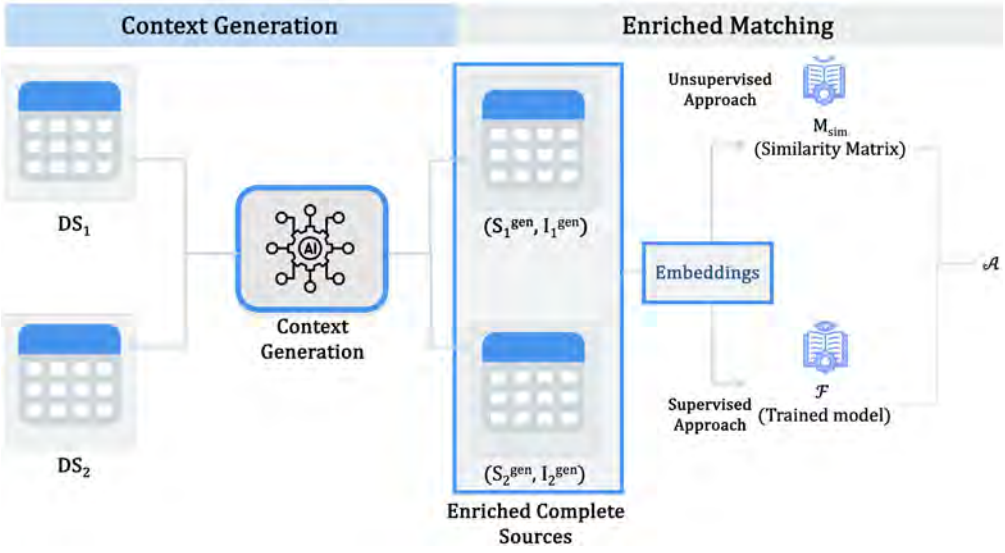


Figure 3.1: Overview of the proposed schema matching framework.

In the following sections, we detail each component of the framework. Section 3.2 introduces the enrichment process, including LLM prompting for schema/instance generation. Section 3.3 describes the embedding generation process, in which enriched tabular sources are projected into a shared latent space through pre-trained models. This phase produces dense vector representations that serve as the foundation for both alignment pipelines. Sections 3.4 and 3.5 describe the unsupervised and supervised matching pipelines, respectively, with a focus on matcher composition, similarity matrix aggregation, and meta-feature extraction.

3.2 Enrichment to Bridge Data Disparities

A core challenge in aligning disparate data sources lies in the structural and semantic incompleteness of the inputs. In real-world data integration settings, it is common to encounter schema-only sources (i.e., attribute names without values) and instance-only sources (i.e., data values without column headers). Traditional schema matching algorithms cannot operate effectively under such asymmetrical conditions due to the lack of comparable information.

To overcome this limitation, we introduce context generation, which reconstructs the missing modality and transforms incomplete tables into semantically richer representations. Large language models (LLMs) are well suited to this role: from minimal information, they can generate plausible schema or representative values, leverage extensive linguistic and general knowledge, and generalize across different domains without supervision. Specifically, we implement a semantic enrichment module that uses the generative prompt of LLMs (e.g., GPT or LLaMA) to infer missing information on each side of a source pair, resulting in comparable representations that can be aligned. The process is illustrated in Figure 3.2.

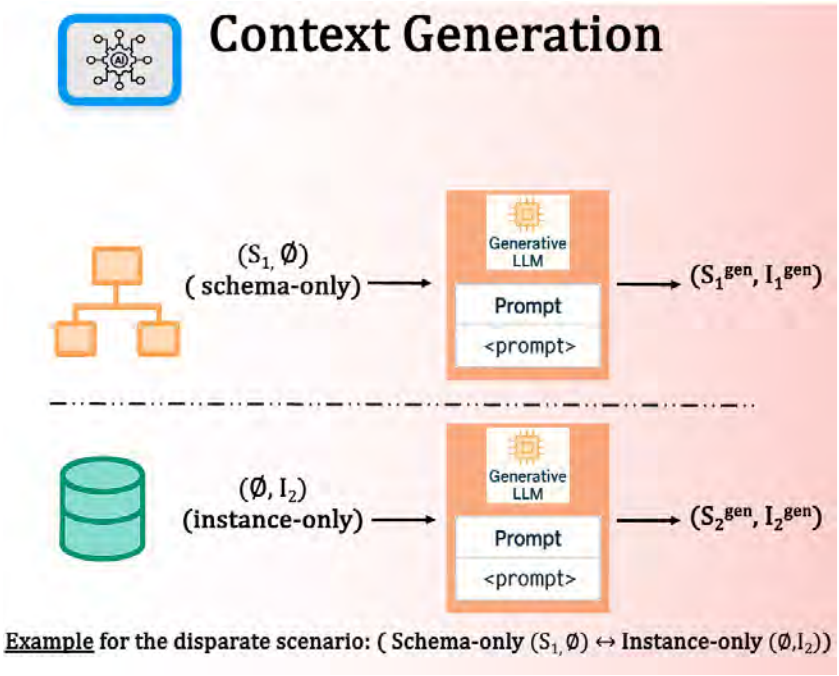


Figure 3.2: LLM-based enrichment for incomplete data sources: generating instances for schema-only sources (S_1) and plausible attribute names for instance-only sources (I_2).

Table 3.1 provides a comprehensive summary of the notations introduced in this section to facilitate understanding.

This phase forms the foundation of our framework by using LLMs to simulate plausible content for missing modalities. Each incomplete source—either schema-only or instance-only—is enriched through a prompt-driven generation step, yielding an augmented representation that better captures its semantics. We detail both subcomponents of the enrichment process below.

We distinguish two prompt families: (i) an *unsupervised* prompt family that produces lists of plausible values per attribute for schema-only sources, and lists of candidate attribute names per column for instance-only sources, defined in Section 4.2.1; and (ii) a *supervised* prompt family

Table 3.1: Summary of notations

$\mathcal{S}_1$	$\mathcal{S}_1 = \{a_1, \ldots, a_n\}$, $\forall i \in [1, \ldots, n]$, a_i is an attribute.
$\mathcal{S}_2$	$\mathcal{S}_2 = \{b_1, \ldots, b_m\}$, $\forall i \in [1, \ldots, m]$, b_i is an attribute.
$\mathcal{I}_1$	$\mathcal{I}_1 = \{t_1, \ldots, t_p\}$ $\forall j \in [1, \ldots, p]$, $t_j =< v_j^1, \ldots, v_j^n >$ is an instance. $\forall i \in [1, \ldots, n]$, $V^i = \{v_1^i, \ldots, v_p^i\}$ where $\mathcal{I}_1$ lists p instances; t_j is the j-th instance (tuple), and V^i stores the values of column i.
$\mathcal{I}_2$	$\mathcal{I}_2 = \{s_1, \ldots, s_q\}$ $\forall j \in [1, \ldots, q]$, $s_j =< w_j^1, \ldots, w_j^m >$ is an instance; $\forall i \in [1, \ldots, m]$, $W^i = \{w_1^i, \ldots, w_q^i\}$ where $\mathcal{I}_2$ lists q instances; s_j is the j-th instance (tuple), and W^i stores the values of column i.
$\mathcal{DS}_1$	$\mathcal{DS}_1 = (\mathcal{S}_1, \mathcal{I}_1)$, with possibly $(\mathcal{S}_1 = \varnothing)$ or $(\mathcal{I}_1 = \varnothing)$.
$\mathcal{DS}_2$	$\mathcal{DS}_2 = (\mathcal{S}_2, \mathcal{I}_2)$, with possibly $(\mathcal{S}_2 = \varnothing)$ or $(\mathcal{I}_2 = \varnothing)$.
$\mathcal{S}_1^{\text{gen}}$	$\mathcal{S}_1^{\text{gen}} = \{\hat{S}_1^1, \ldots, \hat{S}_1^N\}$, $\forall k \in [1, \ldots, N]$, $\hat{S}_1^k = \{\hat{a}_1^k, \ldots, \hat{a}_n^k\}$, $\forall i \in [1, \ldots, n]$, $A_1^i = \{\hat{a}_i^1, \ldots, \hat{a}_i^N\}$ where $\mathcal{S}_1^{\text{gen}}$ contains N schemas; A_1^i is the attributes of column i.
$\mathcal{I}_1^{\text{gen}}$	$\mathcal{I}_1^{\text{gen}} = \{\hat{t}_1, \ldots, \hat{t}_P\}$, $\forall l \in [1, \ldots, P]$, $\hat{t}_l =< \hat{v}_l^1, \ldots, \hat{v}_l^n >$, $\forall i \in [1, \ldots, n]$, $\hat{V}^i = \{\hat{v}_1^i, \ldots, \hat{v}_P^i\}$ where $\mathcal{I}_1^{\text{gen}}$ lists P instances; $\hat{t}_l$ is the l-th instance (tuple), and $\hat{V}^i$ stores the values of column i.
$\mathcal{S}_2^{\text{gen}}$	$\mathcal{S}_2^{\text{gen}} = \{\hat{S}_2^1, \ldots, \hat{S}_2^M\}$, $\forall k \in [1, \ldots, M]$, $\hat{S}_2^k = \{\hat{b}_1^k, \ldots, \hat{b}_m^k\}$, $\forall i \in [1, \ldots, m]$, $A_2^i = \{\hat{b}_i^1, \ldots, \hat{b}_i^M\}$ where $\mathcal{S}_2^{\text{gen}}$ contains M schemas; A_2^i is the attributes of column i.
$\mathcal{I}_2^{\text{gen}}$	$\mathcal{I}_2^{\text{gen}} = \{\hat{s}_1, \ldots, \hat{s}_Q\}$, $\forall l \in [1, \ldots, Q]$, $\hat{s}_l =< \hat{w}_l^1, \ldots, \hat{w}_l^m >$, $\forall i \in [1, \ldots, m]$, $\hat{W}^i = \{\hat{w}_1^i, \ldots, \hat{w}_Q^i\}$ where $\mathcal{I}_2^{\text{gen}}$ lists Q instances; $\hat{s}_l$ is the l-th instance (tuple), and $\hat{W}^i$ stores the values of column i.
$\mathcal{DC}_1$	$(\mathcal{S}_1^{\text{gen}}, \mathcal{I}_1^{\text{gen}})$.
$\mathcal{DC}_2$	$(\mathcal{S}_2^{\text{gen}}, \mathcal{I}_2^{\text{gen}})$.
$\mathcal{MD}_1$	Metadata for $\mathcal{DS}_1$ (e.g., data types, context, table name, examples), used for prompting.
$\mathcal{MD}_2$	Metadata for $\mathcal{DS}_2$ (e.g., data types, context, table name, examples), used for prompting.
$name_{\mathcal{DS}_1}$	Contextual name or description of $\mathcal{DS}_1$, used in prompts.
$name_{\mathcal{DS}_2}$	Contextual name or description of $\mathcal{DS}_2$, used in prompts.
$types(a_i)$	Data type of attribute a_i (e.g., string, int, date), used to guide value generation.
$types(\hat{V}^i)$	Data type of instance column $\hat{V}^i$, used to guide attribute name generation.
$\mathcal{R}_1$	$\mathcal{R}_1 = \{r^1, r^2, r^3, \ldots, r^n\}$ is a set of LLM-generated validation rules for source $\mathcal{S}_t$, where each r^i is constraints for attribute a_i (e.g., regular expressions, range, uniqueness).

that generates instances or schemas, introduced and evaluated in Section 4.3.1. We formalize these prompts as follows:

1.1 Generating Instances for Schema-only Data Sources For a schema-only source $(\mathcal{S}_1, \varnothing)$, the supervised prompts generate full instances $\mathcal{I}_1^{\text{gen}} = \{\hat{t}_1, \ldots, \hat{t}_P\}$, where each $\hat{t}_l =< \hat{v}_l^1, \ldots, \hat{v}_l^n >$. $\forall l \in [1, \ldots, P]$

Formally, $\forall i \in [1, \ldots, n]$ $\forall l \in [1, \ldots, P]$, the column values $\hat{V}^i = \{\hat{v}_1^i, \ldots, \hat{v}_P^i\}$ are induced from $\hat{t}_l$ (whereas in the unsupervised setting, $\hat{V}^i$ are generated directly as column values lists).

1.2 Generating Schema for Instance-Only Data Sources Conversely, for an instance-only source $(\varnothing, \mathcal{I}_1)$, the supervised prompts generate a full schema $\mathcal{S}_1^{\text{gen}} = \{\hat{S}_1^1, \ldots, \hat{S}_1^N\}$, where each $\hat{S}_1^k = \{\hat{a}_1^k, \ldots, \hat{a}_n^k\}$ $\forall k \in [1, \ldots, N]$.

Formally, $\forall i \in [1, \ldots, n]$ $\forall k \in [1, \ldots, N]$, the attribute values $\hat{A}_1^i = \{\hat{a}_i^1, \ldots, \hat{a}_i^N\}$ are induced from $\hat{S}_1^k$ (whereas in the unsupervised setting, A_1^i are generated directly as column attributes lists).

1.3 Constraint Generation for Rule-Based Matching In the context of the unsupervised matching pipeline only, we generate validation rules that serve as the foundation for the rule-based matcher. These rules are not optional metadata but are explicitly designed to guide matching decisions in the absence of labeled data. They capture key attribute-level constraints such as valid ranges, formats, and logical restrictions.

The output includes a set of rules $\mathcal{R}_1 = \{r^i\}$ that define acceptable values and structural constraints for each attribute a_i in source $\mathcal{DS}_1$. Examples of such constraints include:

- Regex patterns for format validation (e.g., dates, phone numbers)
- Range and length constraints (e.g., `age` $\in [0, 120]$)
- Cardinality or uniqueness checks

The resulting rule set $\mathcal{R}_1$ is used directly within the rule-based matcher of the unsupervised pipeline. It enables the system to assess compatibility between attributes based on inferred constraints—for instance, determining whether two `age` columns from different sources share similar value domains or formats, and are therefore likely to correspond.

Summary and Discussion. The enrichment phase adapts based on the available data in each source. Six configurations are supported:

Each $\mathcal{DS}_1$ may be schema-only $(\mathcal{S}_1, \varnothing)$, instance-only $(\varnothing, \mathcal{I}_1)$, or complete $(\mathcal{S}_1, \mathcal{I}_1)$, and the same applies to $\mathcal{DS}_2$. Any combination of these states can occur across the two sources, leading to the six scenarios described below:

- **Compliant settings**, where both sources expose the same type of information:

  (C1) $(\mathcal{S}_1, \mathcal{I}_1) \leftrightarrow (\mathcal{S}_2, \mathcal{I}_2)$**:** Both sources are complete. No enrichment needed.

  (C2) $(\mathcal{S}_1, \varnothing) \leftrightarrow (\mathcal{S}_2, \varnothing)$**:** Both schema-only. Generate $\mathcal{I}_1^{\text{gen}}$ and $\mathcal{I}_2^{\text{gen}}$.

  (C3) $(\varnothing, \mathcal{I}_1) \leftrightarrow (\varnothing, \mathcal{I}_2)$**:** Both instance-only. Generate $\mathcal{S}_1^{\text{gen}}$ and $\mathcal{S}_2^{\text{gen}}$.

- **Disparate settings**, where one side lacks a modality:

  (D4) $(\mathcal{S}_1, \mathcal{I}_1) \leftrightarrow (\mathcal{S}_2, \varnothing)$**:** Generate $\mathcal{I}_2^{\text{gen}}$ from $\mathcal{S}_2$.

  (D5) $(\mathcal{S}_1, \mathcal{I}_1) \leftrightarrow (\varnothing, \mathcal{I}_2)$**:** Generate $\mathcal{S}_2^{\text{gen}}$ from $\mathcal{I}_2$.

  (D6) $(\mathcal{S}_1, \varnothing) \leftrightarrow (\varnothing, \mathcal{I}_2)$**:** Generate $\mathcal{I}_1^{\text{gen}}$ and $\mathcal{S}_2^{\text{gen}}$.

Unified Output. Regardless of the initial configuration of $\mathcal{DS}_1$ and $\mathcal{DS}_2$, the enrichment module outputs the enriched forms:

$$\mathcal{CS}_1 = (\mathcal{S}_1^{\text{gen}}, \mathcal{I}_1^{\text{gen}}) \longleftrightarrow \mathcal{CS}_2 = (\mathcal{S}_2^{\text{gen}}, \mathcal{I}_2^{\text{gen}})$$

where each component is taken from the original data if available, or generated otherwise. At the end of this step, we obtain datasets with a complete schema and instances.

3.3 Embedding Generation

Once both sources have been enriched, we integrate them into a common vector space to enable and facilitate direct comparison. The objective at this stage is to encode each enriched column as a vector in a shared latent space. Below, we explain how embeddings are constructed in the unsupervised and supervised frameworks.

Unsupervised embedding construction. For each enriched source $\mathcal{CS}_1 = (\mathcal{S}_1^{\text{gen}}, \mathcal{I}_1^{\text{gen}})$ and $\mathcal{CS}_2 = (\mathcal{S}_2^{\text{gen}}, \mathcal{I}_2^{\text{gen}})$, we build one vector *per column* by combining the attribute name and its values.

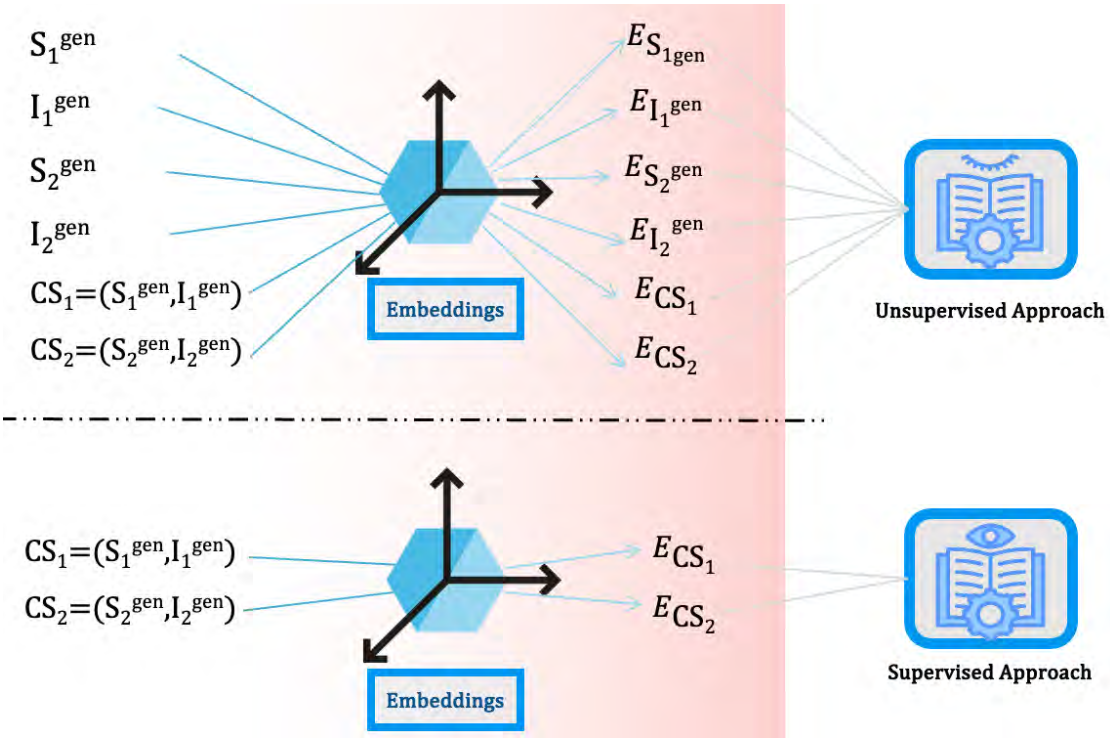


Figure 3.3: Generating embeddings for enriched data sources ($\mathcal{CS}_1$ and $\mathcal{CS}_2$)

- For each $A_1^i \in \mathcal{S}_1^{\text{gen}}$, concatenate A_1^i with its values column $\hat{V}^i$, then encode into a d-dimensional vector e_1^i. Stack the row vectors to obtain

$$E_{\mathcal{CS}_1} = \begin{bmatrix} \mathbf{e}_1^1 \\ \vdots \\ \mathbf{e}_1^n \end{bmatrix}^\top \in \mathbb{R}^{n \times d}.$$

- For each $A_2^j \in \mathcal{S}_2^{\text{gen}}$, concatenate A_2^j with its values column $\hat{W}^j$, encode into e_2^j, and stack into

$$E_{\mathcal{CS}_2} = \begin{bmatrix} \mathbf{e}_2^1 \\ \vdots \\ \mathbf{e}_2^m \end{bmatrix}^\top \in \mathbb{R}^{m \times d}.$$

We also generate embedding matrices used by the unsupervised matchers: schema-only matrices $E_{\mathcal{S}_1^{\text{gen}}} \in \mathbb{R}^{n \times d}$ and $E_{\mathcal{S}_2^{\text{gen}}} \in \mathbb{R}^{m \times d}$ (encoding only A_1^i or A_2^j), and instance-only matrices $E_{\mathcal{I}_1^{\text{gen}}} \in \mathbb{R}^{n \times d}$ and $E_{\mathcal{I}_2^{\text{gen}}} \in \mathbb{R}^{m \times d}$ (encoding only $\hat{V}^i$ or $\hat{W}^j$). A transformer encoder (e.g., BERT or Sentence-Transformers) is used to produce the d-dimensional vectors.

Supervised embedding construction. Exactly the same as in the unsupervised case: for each enriched source, we build a matrix by encoding each column using its attribute name combined with its column values, and we stack the resulting row vectors. In both frameworks, we therefore produce only the embedding matrices $E_{\mathcal{CS}_1} \in \mathbb{R}^{n \times d}$ and $E_{\mathcal{CS}_2} \in \mathbb{R}^{m \times d}$.

Summary and Discussion. Up to this point, we have described the common modules of the two pipelines: enrichment and embedding generation. These embeddings would enable the alignment of the two schema matching methods. Next, Section 3.4 details the unsupervised framework, and Section 3.5 presents the supervised framework. This separation clarifies how both approaches build on the same enriched and embedded representations, but diverge in their matching strategies.

3.4 Unsupervised Schema Matching Using Matchers

We leverage multiple complementary matchers applied to LLM-augmented data embeddings to perform unsupervised matching. By integrating these matchers, we capture structural, semantic, lexical, statistical, and rule-based signals, mitigate individual matcher failures, and achieve greater robustness to noise and domain shift. This multi-matcher strategy exploits diverse similarity perspectives, improving alignment without requiring labeled data.

3.4.1 Overview of the Framework

Our proposed framework, illustrated in Figure 3.4, addresses schema–instance alignment by comparing enriched sources $\mathcal{CS}_1$ and $\mathcal{CS}_2$.

1. **Multiple Schema-Instance Matchers**: Pre-matchers compare the two enriched data sources and remove incompatible candidates (e.g., mismatched types). The Type Matcher checks data types and formats from metadata, the Rule Matcher validates alignment using constraints (e.g., ranges, regex), the Schema Matcher aligns structural elements, the

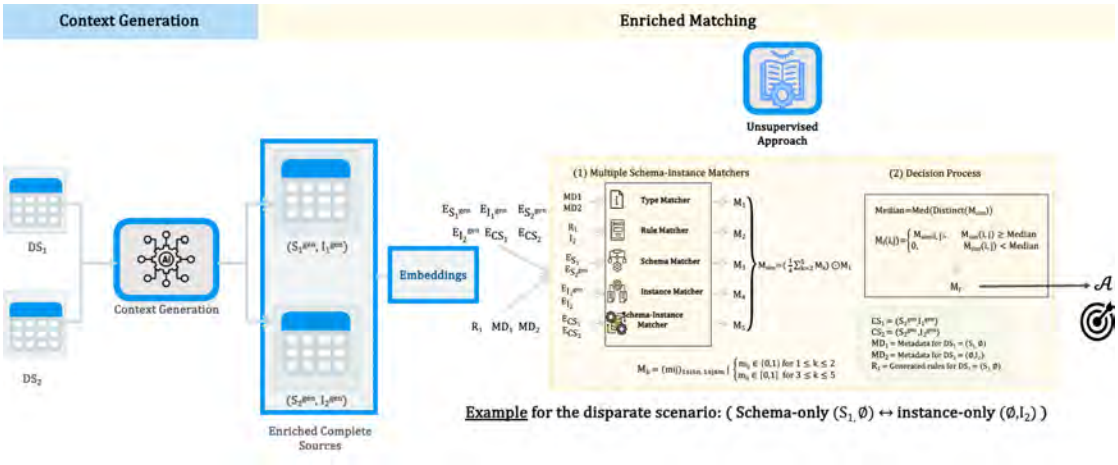


Figure 3.4: Overview of the Unsupervised Tabular Schema Matching Framework

Instance Matcher compares instance values, and the Schema-Instance Matcher combines schema and instance features for hybrid alignment. Each matcher outputs a similarity matrix.

2. **Decision Process**: Aggregates the similarity matrices into M_{sim}, applies median-threshold filtering, and outputs the final matched attribute pairs (M_f) with their similarity scores.

3.4.2 Multiple Schema-Instance Matchers

This component compares and aligns attributes and instances between $\mathcal{S}_1^{\text{gen}}$ and $\mathcal{S}_2^{\text{gen}}$, as well as between $\mathcal{I}_1^{\text{gen}}$ and $\mathcal{I}_2^{\text{gen}}$.

- Type Matcher (M_1), which captures similarity at the type level.

$$M_1 = (m_{1,ij})_{1\le i\le n,\ 1\le j\le m}, \quad m_{1,ij} = \mathbb{1}_{\{\texttt{types}(\mathcal{A}_1^i)=\texttt{types}(\mathcal{A}_2^j)\}}, \quad m_{1,ij} \in \{0,1\}.$$

  where $\mathcal{A}_1^i \in \mathcal{S}_1^{\text{gen}}$, $\mathcal{A}_2^j \in \mathcal{S}_2^{\text{gen}}$.

- Rule Matcher (M_2), which enables comparison taking constraints into account.

$$M_2 = (m_{2,ij})_{1\le i\le n,\ 1\le j\le m}, \quad m_{2,ij} = \mathbb{1}_{\{\hat{\mathcal{W}}^j \text{ satisfies all } r^i\}}, \quad m_{2,ij} \in \{0,1\}.$$

  where $\hat{\mathcal{W}}^j \in \mathcal{I}_2^{\text{gen}}$.

- Schema Matcher (M_3), which computes similarity between attribute names.

$$M_3 = (m_{3,ij})_{1\le i\le n,\ 1\le j\le m}, \quad m_{3,ij} = \frac{\vec{\mathcal{A}}_1^i \cdot \vec{\mathcal{A}}_2^j}{\|\vec{\mathcal{A}}_1^i\|\ \|\vec{\mathcal{A}}_2^j\|}, \quad m_{3,ij} \in [0,1].$$

  where $\vec{\mathcal{A}}_1^i \in E_{\mathcal{S}_1^{\text{gen}}}$, $\vec{\mathcal{A}}_2^j \in E_{\mathcal{S}_2^{\text{gen}}}$.

- Instance Matcher (M_4), which computes similarity between column values.

$$M_4 = (m_{4,ij})_{1\le i\le n,\ 1\le j\le m}, \quad m_{4,ij} = \frac{\vec{\hat{\mathcal{V}}}^i \cdot \vec{\hat{\mathcal{V}}}^j}{\|\vec{\hat{\mathcal{V}}}^i\|\ \|\vec{\hat{\mathcal{V}}}^j\|}, \quad m_{4,ij} \in [0,1].$$

  where $\vec{\hat{\mathcal{V}}}^i \in E_{\mathcal{I}_1^{\text{gen}}}$, $\vec{\hat{\mathcal{V}}}^j \in E_{\mathcal{I}_2^{\text{gen}}}$.

- Schema-Instance Matcher (M_5), which compares attribute names and column values as one.

$$M_5 = (m_{5,ij})_{1\le i\le n,\ 1\le j\le m}, \quad m_{5,ij} = \frac{e_1^i \cdot e_2^j}{\|e_1^i\|\ \|e_2^j\|}, \quad m_{5,ij} \in [0,1].$$

  where $e_1^i \in E_{\mathcal{CS}_1}$ is obtained by concatenating $(\mathcal{A}_1^i, \hat{\mathcal{V}}^i)$, and $e_2^j \in E_{\mathcal{CS}_2}$ is obtained by concatenating $(\mathcal{A}_2^j, \hat{\mathcal{W}}^j)$.

Integration and Aggregation. The five matchers described above contribute different but complementary views of similarity. We treat the *Type Matcher (M_1)* as a strict *pre-filter*: if two attributes are type-incompatible, no other matcher is considered meaningful, and their overall similarity is set to zero. In contrast, the remaining four matchers — *Rule Matcher (M_2)*, *Schema Matcher (M_3)*, *Instance Matcher (M_4)*, and *Schema-Instance Matcher (M_5)* — are integrated uniformly, since each provides weak but useful evidence. This uniform treatment reflects our philosophy of considering schema-level, instance-level, and hybrid-level information as complementary rather than hierarchical.

Formally, the aggregated similarity matrix is defined as:

$$M_{\text{sim}} = (m_{\text{sim},ij})_{1\le i\le n,\ 1\le j\le m}, \quad M_{\text{sim}} = \left(\frac{1}{4}\sum_{k=2}^{5} M_k\right) \odot M_1, \quad m_{\text{sim},ij} \in [0,1].$$

Where $\odot$ denotes the Hadamard (element-wise) product.

3.4.3 Decision Process

To determine the final alignment, we apply a median-threshold filter over the distinct values of M_{sim}. This step eliminates weak or spurious correspondences, while retaining those with significant similarity. Formally, the filtered similarity matrix M_f is given by:

$$m_{f,ij} = \begin{cases} m_{\text{sim},ij}, & \text{if } m_{\text{sim},ij} \ge \text{median}(\mathit{Distinct}(M_{\text{sim}})), \\ 0, & \text{otherwise.} \end{cases}$$

The use of the median as a threshold provides robustness by adapting the filtering to the overall distribution of scores, avoiding arbitrary cut-offs. Thus, the decision process ensures that only the most consistent matches, supported jointly by schema, instance, and hybrid information, are retained in the final mapping.

Final Output. The final matrix M_f contains the accepted alignments between $E_{\mathcal{CS}_1}$ and $E_{\mathcal{CS}_2}$ together with their similarity scores. The resulting alignment set is

$$\mathcal{A} = \{(i,j) \mid \hat{y}_{ij} = 1\},$$

where i indexes the i-th column of $\mathcal{DS}_1$ and j indexes the j-th column of $\mathcal{DS}_2$.

Discussion. An explicit integration of schema-only, instance-only, and hybrid matchers within a single uniform framework has not been widely studied in the literature. Most existing works either focus on schema-level matchers or instance-level matchers, but rarely address their systematic joint integration.

The general idea of combining multiple sources of evidence, however, has been explored extensively. For example, Rahm and Bernstein [100] already pointed out in their study that effective schema matching generally requires combining structural, linguistic, and example-based evidence. More recently, Mudgal et al. [86] demonstrated how deep learning methods can integrate schema and instance information into a unified representation for entity matching.

Our contribution builds on this work by explicitly integrating schema-only (M_3), instance-only (M_4), and hybrid (M_5) matchers, together with type-based filtering (M_1) and rule-based evidence (M_2), under a uniform aggregation framework.

Limitations. While the integration strategy provides a balanced and uniform view across multiple matchers, it also comes with some limitations. First, the strict role of the Type Matcher (M_1) may cause false negatives: if types are incorrectly declared or overly generic, potentially valid correspondences are discarded prematurely. Second, the uniform weighting of M_2–M_5 assumes equal reliability among schema-, instance-, rule-, and hybrid-based evidence, which may not hold in practice. Third, all similarity computations for vector embeddings (M_3, M_4, M_5) rely exclusively on cosine similarity; this choice provides robustness and normalization, but it may overlook other distance measures that could capture complementary aspects of similarity. Finally, the use of the median as a global threshold, though robust, may still filter out meaningful matches when similarity distributions are highly skewed.

To address some of these limitations, we explored a supervised pipeline. Instead of relying on uniform weighting and cosine similarity alone, we trained a classifier in a *meta-space*, where the outputs of the individual matchers serve as input features. This supervised approach allows the model to learn more adaptive weighting schemes, to combine heterogeneous evidence in a principled manner, and to move beyond cosine similarity toward more expressive similarity functions. Such a meta-level integration provides a more flexible framework, at the cost of requiring labeled training data. The following section details this supervised pipeline.

3.5 Supervised Matching Using Meta-Space

The objective of this section is to define a supervised schema matching framework designed to align data sources. Our setting considers compliant and disparate scenarios.

Once the embeddings are generated, we use them in a supervised pipeline that consists of two main steps, as illustrated in Figure 3.5, Table 3.2 provides a comprehensive summary of the notations used throughout the framework to facilitate a clear understanding. The details of our framework are as follows:

1. **Meta Space.** The enriched tabular data sources are encoded into dense vectors ($E_{\mathcal{CS}_1}, E_{\mathcal{CS}_2}$) using a pre-trained embedding model. We then compute a set of meta-features between these vectors, including: (i) *classical meta-features* such as cosine and Euclidean distances; (ii) *spectral meta-features* such as singular values and stable rank; and (iii) *topological meta-features* such as persistent homology and entropy, which capture geometric and semantic properties in high-dimensional space. All computed meta-features are concatenated to form the unified meta-space M_{ms}.

2. **Meta Learner.** A supervised classifier is trained on M_{Train}, where each attribute pair is represented by a meta-feature vector and its corresponding label. The trained model is then applied to M_{Test} to predict matches. The final predicted alignment set consists of index pairs (i, j).

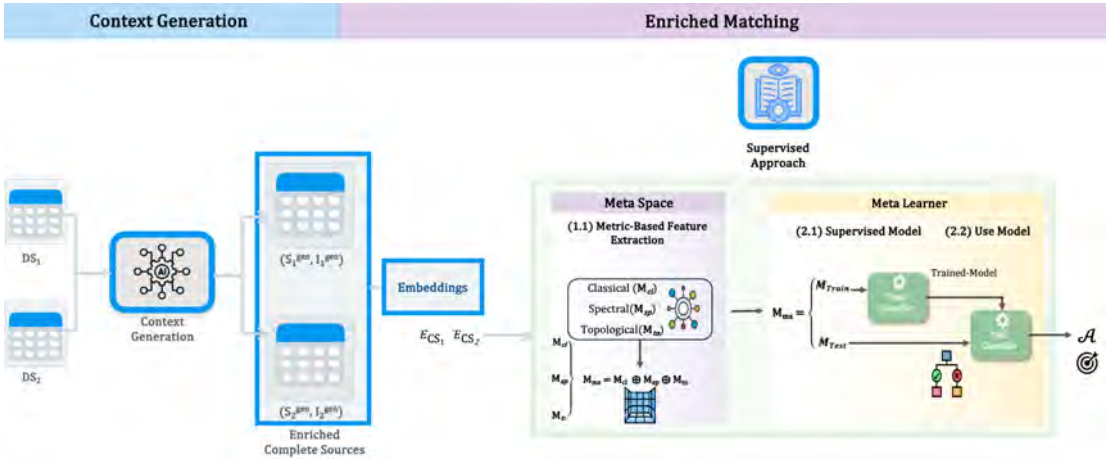


Figure 3.5: Overview of the Supervised Tabular Schema Matching Framework

The introduction of a meta-space and a meta-learner is central to our supervised framework. This idea is not entirely new: Doan et al. [37] already proposed to combine multiple schema matchers using supervised learning, while more recent systems such as MOMA [114] and Magellan [64] rely on a feature space and a meta-learner to integrate heterogeneous evidence in entity matching. However, these works mainly exploit classical similarity features (e.g., edit distances, Jaccard, TF-IDF). A less explored direction in schema matching is to exploit not only the proximity of two vectors, but also the *geometry* of the representation space. Classical distances (cosine, Euclidean, Manhattan) are useful and complementary, but remain local. By contrast, *spectral meta-features* summarize global properties of embedding matrices—singular value distributions, stable rank—and are widely used to assess representation quality [85, 42, 115]. Topological Data Analysis (TDA) provides global invariants of point clouds via persistent homology (bottleneck/Wasserstein distances, persistence entropy) [54, 19]. In practice, TDA is mostly used to characterize time series and dynamics (recurrent patterns, cycles, regime shifts) [97, 106], and rarely in tabular schema matching—overlooking rich geometric and topological cues (density, spread, multi-scale structure) that meaningfully complement classical metrics. We address this gap by integrating classical, spectral, and topological *meta-features* into a unified meta-space that captures intrinsic non-linear relations among attributes and improves alignment decisions. This meta-space retains information that standard distances ignore—neighborhood density and spread, local spectral decay, and patterns that persist across scales [6, 46, 130]. Crucially, these meta-features are *language-agnostic* and remain useful with ambiguous, noisy, and heterogeneous schemas or non-textual values [100, 43, 96]. In this way, we leverage the geometry of pretrained embeddings to better align heterogeneous tabular data [42, 107].

Geometric intuition. Two embedding vectors that carry the same underlying signal (e.g., a similar oscillatory trend under a sliding window) produce point clouds with nearly the same *shape*. Our pipeline applies a sliding-window transform that maps each vector to a local point cloud;

Table 3.2: Summary notations used in the proposed framework

Meta Space Construction	
h	Number of meta-features extracted for each attribute pair ($h = 112$ in our framework).
d	Dimension of the embedding vector associated with each attribute ($d = 768$ for BERT).
$E_{\mathcal{CS}_1}$, $E_{\mathcal{CS}_2}$	Embedding matrices: $E_{\mathcal{CS}_1} = [\mathbf{e}_1^1, \ldots, \mathbf{e}_1^n]^\top$ for the incomplete source, and $E_{\mathcal{CS}_2} = [\mathbf{e}_2^1, \ldots, \mathbf{e}_2^m]^\top$ for the complete one. Each vector $\mathbf{e}_1^i, \mathbf{e}_2^j \in \mathbb{R}^d$.
$M_{\mathbf{ms}}$	Meta-space matrix $$M_{\text{ms}} = (m_{ij})_{1 \le i \le n,\ 1 \le j \le m}, \quad m_{ij} \in \mathbb{R}^h$$ Each m_{ij} leverages classical, spectral, and topological similarity features.
Meta Learner (Supervised Matching)	
$M_{\mathbf{Train}}$, $M_{\mathbf{Test}}$	Subsets of the meta-space matrix M_{ms} used for supervised training and evaluation, respectively. Each m_{ij} meta-feature vector is associated with a binary label $y_{ij} \in \{0, 1\}$ indicating whether the attribute pair at position (i, j) is a match.
Predicted Alignments (final output)	
$\mathcal{A}$	The final alignment set is defined as $\mathcal{A} = \{(i, j) \mid \hat{y}_{ij} = 1\}$. Each predicted alignment is a pair (i, j) between an incomplete source i-th column and a complete source j-th column, with predicted label $\hat{y}_{ij} \in \{0, 1\}$ where 1 indicates a match and 0 a non-match.

vectors encoding the same information yield congruent clouds (Figure 3.6(b)). Importantly, this shape is preserved (up to affine/orthogonal re-projection) when the clouds are embedded or projected into a new space, which is why our spectral and topological *meta-features*—computed on these clouds—remain stable across models and projections. This invariance under re-projection enables our framework to compare attributes by their geometry rather than by a single pairwise angle or token overlap.

We argue that this pipeline is pertinent because it unifies complementary views of similarity and embedding geometry, remains robust even without explicit type information, and is easily extensible to additional meta-features. The geometric perspective is especially important: spectral meta-features capture global structural stability, while topological ones describe persistent shape patterns. Together with classical similarity measures, they enrich the meta-space with semantic and syntactic context that a supervised learner can exploit for alignment. Our meta-space and meta-learner pipeline offers a middle ground: flexible, supervised, and interpretable, while remaining practical with limited annotations. The two components are described in detail in the following sections.

3.5.1 Meta-Space

At this stage, we build a representation meta-space in which each attribute pair is encoded as a meta-feature vector. The embedding matrices $E_{\mathcal{CS}_1}$ and $E_{\mathcal{CS}_2}$ used here are introduced in Section 3.3.

For each pair of attributes, we extract their corresponding embedding vectors:

$$\mathbf{e}_1^i \in E_{\mathcal{CS}_1}, \quad \mathbf{e}_2^j \in E_{\mathcal{CS}_2}$$

These vectors serve as the basis for computing the meta-features described below.

Meta-Feature Extraction. For each pair of embedding vectors $(\mathbf{e}_1^i, \mathbf{e}_2^j)$, we derive a unified meta-feature vector in two main steps.

1. Sliding window transformation. Spectral and topological meta-features require matrix inputs. For simplicity, we denote each d-dimensional vector as $\mathbf{e}$, although it actually refers to either $\mathbf{e}_1^i$ or $\mathbf{e}_2^j$, and similarly use $\mathbf{P}$ to denote either $\mathbf{P}_1^i$ or $\mathbf{P}_2^j$. Each vector $\mathbf{e}$ is thus reshaped into a $(d-w+1) \times w$ matrix $\mathbf{P} = \mathrm{SW}(\mathbf{e}) \in \mathbb{R}^{(d-w+1)\times w}$ (cf. Figure 3.6.a). This transformation enables the use of methods such as singular value decomposition and persistent homology. As shown in Figure 3.6.b, applying the sliding window transformation to functions exhibiting similar global behavior (e.g., sinusoidal functions) yields point clouds that retain coherent geometric structures. This empirical evidence supports the stability and structure-preserving properties of the transformation, which are critical for downstream comparative analyses. Throughout all experiments, we fix the window size to $w = 5$.

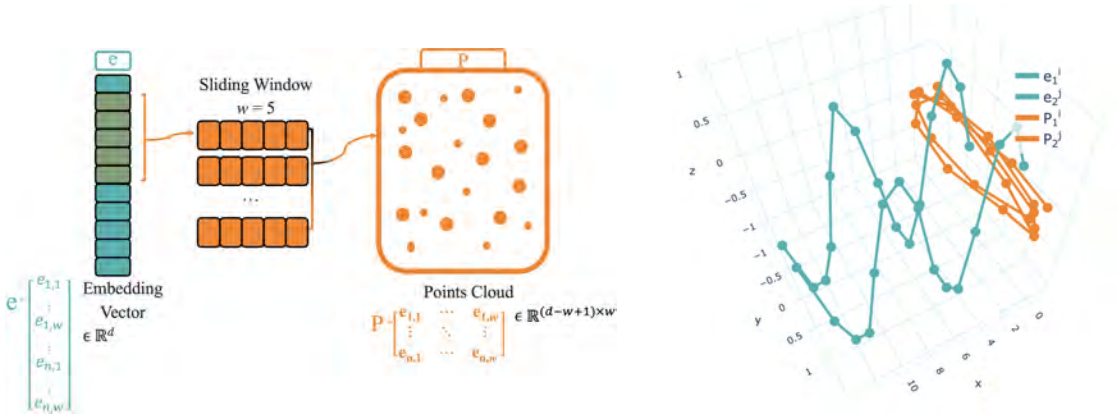
2. Meta-feature computation. From the embedding vectors $\mathbf{e}_1^i$ and $\mathbf{e}_2^j$, as well as their sliding window matrices $\mathbf{P}_1^i$ and $\mathbf{P}_2^j$, we compute meta-features from three families:

- *Classical* (Cosine, Euclidean, Minkowski-p, Chebyshev, Canberra, Pearson, Spearman, string-based similarities such as Levenshtein, Damerau–OSA, LCS, common prefix/suffix ratios, Jaccard/Dice on tokens or n-grams, Cosine on character n-grams, and Jaro/Jaro–Winkler) on $\mathbf{e}_1^i$, $\mathbf{e}_2^j$.
- *Spectral* [115, 120] (Stable Rank, RankMe, Alpha-ReQ, NESum, SelfCluster) on $\mathbf{P}_1^i, \mathbf{P}_2^j$, the concatenated vector $[\mathbf{e}_1^i \mid \mathbf{e}_2^j]$, and the full embedding matrices $E_{\mathcal{CS}_1}, E_{\mathcal{CS}_2}$.
- *Topological* [130, 15] (persistence entropy, bottleneck distance) via persistence diagrams built from $\mathbf{P}_1^i, \mathbf{P}_2^j$, the concatenated vector $[\mathbf{e}_1^i \mid \mathbf{e}_2^j]$, and the full embedding matrices $E_{\mathcal{CS}_1}, E_{\mathcal{CS}_2}$.

Table 3.4, and 3.5 provide the formal definitions of all meta-features, and Algorithm 1 details the full construction of the meta-space M_{ms}.

Explanation of Algorithm 1. The algorithm details how the meta-space M_{ms} is constructed step by step. Given two embedding matrices $E_{\mathcal{CS}_1}$ and $E_{\mathcal{CS}_2}$, each attribute pair (i, j) is first represented by its embedding vectors $\mathbf{e}_1^i$ and $\mathbf{e}_2^j$ (Step 1). Since spectral and topological meta-features require matrix structures rather than raw vectors, each embedding is transformed into a matrix using the sliding-window operator (Step 2). On this basis, three families of meta-features are computed: (i) classical meta-features based on vector or string distances (Step 3), (ii) spectral meta-features capturing structural properties of the embedding matrices (Step 4), and (iii) topological meta-features derived from persistence diagrams (Step 5). Finally, all features are concatenated into a single vector $m_{ij} \in \mathbb{R}^h$ that serves as one row of the meta-space matrix (Step 6).

This construction ensures that each attribute pair is described not by a single similarity score, but by a multi-dimensional representation combining complementary views (semantic, syntactic, structural). The meta-learner can then exploit this enriched representation to adaptively weight features and predict alignments.



(a) Left-aligned sliding window (window size = 5, step = 1)

(b) Example: applying the sliding window to sinusoidal functions

Figure 3.6: Illustration of the sliding window transformation over embedding vectors.

3.5.2 Meta-Learner

This stage maps extracted meta-features to alignment decisions using supervised learning. To enhance robustness to outliers and ensure comparability across meta-features, we apply a *RobustScaler* to all vectors in M_{ms}, normalizing each dimension by its median and interquartile range. The scaler is fit on the training split and then applied to the test split, ensuring consistent normalization of the meta-features. The meta-learner then proceeds in two steps:

Supervised Model. Let $M_{\text{Train}} = \{(m_{ij}, y_{ij})\}$ denotes the training dataset, where each $m_{ij} \in \mathbb{R}^h$ is a feature vector computed between a pair of enriched columns $(\mathbf{e}_1^i, \mathbf{e}_2^j)$, and $y_{ij} \in \{0, 1\}$ is the corresponding binary label indicating semantic match (1) or non-match (0). We frame schema matching as a binary classification task. To this end, we train supervised classifiers. Each model is trained over M_{Train}. The trained model $\mathcal{F}$ maps each meta-features vector m_{ij} to a match probability $\mathbb{P}(y_{ij} = 1 \mid m_{ij})$.

Use Model. The trained model $\mathcal{F}$ is applied to test pairs $(m_{ij}, y_{ij}) \in M_{\text{Test}}$, where m_{ij} is a meta-features vector and $y_{ij} \in \{0, 1\}$ the true label. Predictions are computed via a threshold τ (default 0.5):

$$\hat{y}_{ij} = \mathbb{1}\left\{\mathbb{P}(y_{ij} = 1 \mid m_{ij}) > \tau\right\}$$

Alignment Output The final alignment set is $\mathcal{A} = \{(i, j) \mid \hat{y}_{ij} = 1\}$, where i refers to the i-th column of $\mathcal{DS}_1$ and j to the j-th column of $\mathcal{DS}_2$.

Choice of $h = 112$. In our current framework, we designed and implemented $h = 112$ meta-features by combining classical, spectral, and topological meta-features. This number is not fixed: it reflects a balance between diversity of information and computational feasibility. In principle, the meta-space can be extended with additional features derived from other sources of evidence (e.g., instance-level statistics, schema-level constraints) or even reduced to a smaller set. Our current set of 112 meta-features should thus be understood as a *baseline instantiation* of the meta-space, guided by our intuitions about what complementary information is most relevant. Exploring alternative sets—either larger, more specialized, or more compact—constitutes

a promising direction for future research, as it would allow us to study the trade-offs between expressiveness, computational efficiency, and generalization.

Summary and Discussion. The design of our meta-space is novel and was developed entirely from scratch to address limitations observed in prior schema matching approaches. Our intuition is that both **spectral** and **topological** meta-features capture aspects of embedding spaces that are rarely exploited in schema matching. Taken together, they can be interpreted as new dimensions that enrich the representation of attribute embeddings, offering additional perspectives to understand their relative positions in the space.

A key technical contribution enabling this is the **sliding window transformation**, which reshapes embedding vectors into matrices. This step is essential because it allows us to apply spectral and topological meta-features—originally defined for time series or continuous data—to high-dimensional textual embeddings. It also makes it possible to capture local structural information from different overlapping subspaces of the embeddings, thereby reflecting diverse views on similarity.

Finally, we acknowledge that individual meta-features can be difficult to interpret in isolation. This is precisely where the **meta-learner** becomes critical: by training on labeled pairs, it learns to capture the relations and relative importance of each meta-feature within the enriched meta-space. The result is a robust supervised pipeline that unifies different meta-features into a coherent and discriminative representation for schema matching.

3.6 Conclusion

This chapter has presented a unified framework for schema matching that addresses disparity and incompleteness through enrichment, embedding, and alignment. The enrichment step relies on large language models to infer plausible schema and instance information, restoring the contextual basis for comparison. On this foundation, the framework develops two complementary pipelines. The unsupervised pipeline combines multiple matchers—type, rule, schema, instance, and schema–instance—within a decision process that aggregates their outputs into coherent alignments, ensuring robustness when annotated data is unavailable. The supervised pipeline, in contrast, builds a meta-space where enriched embeddings are compared through *classical*, *spectral*, and *topological* meta-features. These meta-features capture semantic, syntactic, and contextual information from the geometry of the embeddings, allowing the meta-learner to predict correspondences with high precision when labels are present. Together, the two pipelines embody complementary logics: one prioritizing generality and adaptability in low-resource settings, the other exploiting supervision to refine alignment through a richer geometric characterization. This dual design establishes a flexible and extensible methodology that advances schema matching and lays the groundwork for the empirical evaluation that follows.

Algorithm 1: Construction of the meta-space M_{ms}

Input: $E_{\mathcal{CS}_1} \in \mathbb{R}^{n\times d}$: embeddings of enriched columns from the source (CS_1); $E_{\mathcal{CS}_2} \in \mathbb{R}^{m\times d}$: embeddings from the source (CS_2); $w = 5$: window size for sliding window transformation; $S_1^{gen} = \{A_1^i, \ldots, A_1^n\}$, $S_2^{gen} = \{A_2^i, \ldots, A_2^m\}$: attribute labels (strings) for CS_1 and CS_2

Output: $M_{\text{ms}} \in \mathbb{R}^{n\times m\times h}$: Meta-space matrix

for $i \leftarrow 1$ **to** n **do**

  for $j \leftarrow 1$ **to** m **do**

    `/* Step 1: Embedding extraction */`

    Let $\mathbf{e}_1^i \in \mathbb{R}^d$ be the i-th column of $E_{\mathcal{CS}_1}$; let $\mathbf{e}_2^j \in \mathbb{R}^d$ be the j-th column of $E_{\mathcal{CS}_2}$;

    `/* Step 2: Sliding window embedding */`

    Compute $\mathbf{P}_1^i = \text{SW}(\mathbf{e}_1^i) \in \mathbb{R}^{(d-w+1)\times w}$ and $\mathbf{P}_2^j = \text{SW}(\mathbf{e}_2^j) \in \mathbb{R}^{(d-w+1)\times w}$;

    `/* Step 3: Classical meta-features */`

    Compute pairwise distances on $(\mathbf{e}_1^i, \mathbf{e}_2^j)$: Cosine, Euclidean, Minkowski-p, Chebyshev, Canberra, Pearson, Spearman;

    Set $\xi \leftarrow$ normalized(A_1^i), $\zeta \leftarrow$ normalized(A_2^j) Compute: Levenshtein, Damerau–OSA, LCS, cp, cs, Jaccard (tokens/char-n-grams), Dice (tokens/char-n-grams), Cosine (char-n-grams), Jaro, Jaro–Winkler, plus $|\xi|, |\zeta|$, equality, substring containment;

    `/* Step 4: Spectral meta-features */`

    On each of $\mathbf{P}_1^i$, $\mathbf{P}_2^j$, $[\mathbf{e}_1^i \mid \mathbf{e}_2^j]$, $E_{\mathcal{CS}_1}$, and $E_{\mathcal{CS}_2}$, compute: Stable Rank, NESum, Alpha-ReQ, RankMe, Spectral Entropy, SelfCluster;

    `/* Step 5: Topological meta-features */`

    Build persistence diagrams (H0/H1) for $\mathbf{P}_1^i$, $\mathbf{P}_2^j$, $[\mathbf{e}_1^i \mid \mathbf{e}_2^j]$, $E_{\mathcal{CS}_1}$, $E_{\mathcal{CS}_2}$; Compute corresponding features for each diagram:

    - Persistence entropy and lifetime statistics
    - Bottleneck and Wasserstein distances (on H_0 and H_1 homology groups)

`/* Step 6: Meta-Space construction */`

Concatenate all features into $m_{ij} \in \mathbb{R}^h$ (with $h = 112$)

return $M_{\text{ms}} = (m_{ij})_{1\le i\le n,\ 1\le j\le m}$

Table 3.4: Summary of mathematical notations used in Table 3.5, grouped by meta-feature family.

Annotation	Definition
Shared	
h	#meta-features per attribute pair (h=112 here).
$\mathbf{e}_1^i, \mathbf{e}_2^j$	Embeddings of attributes i and j; $e_1^{(i,c)}, e_2^{(j,c)}$ are c-th comps.
d	Embedding dimension.
$\|\mathbf{x}\|_p$	p-norm: $\left(\sum_{c=1}^{d} \|x_c\|^p\right)^{1/p}$; $\max_c \|x_c\|$ if $p = \infty$.
$\mathbf{x}\cdot\mathbf{y}$	Inner product: $\sum_{c=1}^{d} x_c y_c$.
Classical	
$\bar{e}_1^i, \bar{e}_2^j$	Component means (Pearson).
$r_1^{(c)}, r_2^{(c)}$	Ranks of $e_1^{(i,c)}, e_2^{(j,c)}$ among d (ties averaged, Spearman).
$LCS(\xi, \zeta)$	Longest common subsequence length.
$cp(\xi, \zeta), cs(\xi, \zeta)$	Longest common prefix/suffix lengths.
M, H	Jaro: M = #matches in window $\lfloor \max(\|\xi\|, \|\zeta\|)/2 \rfloor - 1$; H = #transpositions/2.
L, π	Jaro–Winkler: L = common-prefix length (≤ 4), π scale (typ. 0.1).
String & Syntactic	
ξ, ζ	Normalized strings being compared.
Normalization	NFKD accent strip → remove combining marks → lowercase → non-alnum→space → compress spaces.
tokens(ξ)	Space-split words from normalized ξ.
char_ngrams(ξ, n)	Character n-grams from normalized ξ (no spaces).
$\mathcal{G}_\xi, \mathcal{G}_\zeta$	*Sets* of tokens/char-n-grams (no multiplicity).
$\mathbf{c}_\xi, \mathbf{c}_\zeta$	Count vectors of char-n-grams.
g	Index over char-n-grams.
Spectral	
$X \in \mathbb{R}^{n_r \times n_c}$	Data with n_r rows (samples), n_c cols (features).
n_r, n_c	Row/column counts.
$\|X\|_F$	Frobenius norm $\left(\sum_{i,j} X_{ij}^2\right)^{1/2}$.
$\|X\|_2$	Spectral norm $\sigma_{\max}(X)$.
SVD	$X = U\,\mathrm{diag}(\sigma_1, \ldots, \sigma_r)\,V^\top$, $r = \mathrm{rank}(X)$.
k, v	Indices over singular/eigen orders.
p_k	Normalized singular value: $\sigma_k / \sum_{j=1}^{r} \sigma_j$.
$K = XX^\top,\ C = X^\top X$	Gram matrices (symmetric, PSD).
$\lambda_k = \sigma_k^2$	Nonzero eigenvalues of K and C.
Topological	
$D = \{(b_\tau, d_\tau)\}$	Persistence diagram (birth–death pairs).
m_d	#pairs in D.
$\ell_\tau = d_\tau - b_\tau$	Persistence (lifetime).
$p_\tau = \ell_\tau / \sum_\nu \ell_\nu$	Normalized persistence weight ($0/0 := 0$).
γ	Bijection between points of D_1 and D_2 (matches to diagonal allowed).
q	Wasserstein order ($q \in [1, \infty)$).

Table 3.5: Formal definitions of the meta-features used in our meta-space.

Feature	Mathematical definition
Classical	
Cosine distance	$1 - \dfrac{\mathbf{e}_1^i \cdot \mathbf{e}_2^j}{\|\mathbf{e}_1^i\|_2 \, \|\mathbf{e}_2^j\|_2}$
Euclidean distance	$\|\mathbf{e}_1^i - \mathbf{e}_2^j\|_2$
Minkowski (p=3)	$\left(\sum_{c=1}^{d} \|e_1^{(i,c)} - e_2^{(j,c)}\|^3\right)^{1/3}$
Chebyshev distance	$\max_{1 \le c \le d} \|e_1^{(i,c)} - e_2^{(j,c)}\|$
Canberra distance	$\sum_{c=1}^{d} \dfrac{\|e_1^{(i,c)} - e_2^{(j,c)}\|}{\|e_1^{(i,c)}\| + \|e_2^{(j,c)}\|}$
Pearson correlation	$\dfrac{\sum_{c=1}^{d} (e_1^{(i,c)} - \bar{e}_1^i)(e_2^{(j,c)} - \bar{e}_2^j)}{\sqrt{\sum_{c=1}^{d} (e_1^{(i,c)} - \bar{e}_1^i)^2} \sqrt{\sum_{c=1}^{d} (e_2^{(j,c)} - \bar{e}_2^j)^2}}$
Spearman distance	$1 - \dfrac{6 \sum_{c=1}^{d} (r_1^{(c)} - r_2^{(c)})^2}{d(d^2-1)}$
Levenshtein distance	Minimum edit operations to transform ξ into ζ.
Damerau–OSA distance	Levenshtein with restricted adjacent transpositions.
Longest Common Subsequence (LCS)	$LCS(\xi, \zeta)$
Common prefix/suffix ratios	$cp(\xi,\zeta) = \dfrac{\max\{u :\ \xi[1..u] = \zeta[1..u]\}}{\max(\|\xi\|, \|\zeta\|)}$, $cs(\xi,\zeta) = \dfrac{\max\{v :\ \xi[\|\xi\|-v+1..] = \zeta[\|\zeta\|-v+1..]\}}{\max(\|\xi\|, \|\zeta\|)}$
Jaccard similarity (tokens/n-grams)	$J(\mathcal{G}_\xi, \mathcal{G}_\zeta) = \dfrac{\|\mathcal{G}_\xi \cap \mathcal{G}_\zeta\|}{\|\mathcal{G}_\xi \cup \mathcal{G}_\zeta\|}$
Dice similarity (tokens/n-grams)	$D(\mathcal{G}_\xi, \mathcal{G}_\zeta) = \dfrac{2\,\|\mathcal{G}_\xi \cap \mathcal{G}_\zeta\|}{\|\mathcal{G}_\xi\| + \|\mathcal{G}_\zeta\|}$
Cosine similarity (char n-grams)	$\dfrac{\sum_g \mathbf{c}_\xi(g)\, \mathbf{c}_\zeta(g)}{\sqrt{\sum_g \mathbf{c}_\xi(g)^2} \sqrt{\sum_g \mathbf{c}_\zeta(g)^2}}$
Jaro / Jaro–Winkler similarity	$J = \frac{1}{3}\left(\frac{M}{\|\xi\|} + \frac{M}{\|\zeta\|} + \frac{M-H}{M}\right)$, $J_W = J + L\,\pi\,(1-J)$
Classical meta-features	
Spectral	
Stable Rank	$\mathrm{StableRank}(X) = \dfrac{\|X\|_F^2}{\|X\|_2^2}$
RankMe	$\mathrm{RankMe}(X) = -\sum_{k=1}^{r} p_k \log p_k$
Alpha–ReQ	$\alpha = -\dfrac{d \log \sigma_k}{d \log k}$
NESum	$\mathrm{NESum}(X) = \sum_{v=1}^{r} \dfrac{\lambda_v}{\lambda_1}$
SelfCluster	$\mathrm{SelfCluster}(X) = \dfrac{n_c \, \|K\|_F - n_r(n_c + n_r - 1)}{(n_c - 1)(n_r - 1)\, n_r}$
Spectral meta-features	
Topological	
Persistence entropy	$\mathrm{PersEnt}(D) = -\sum_{\tau=1}^{m_d} p_\tau \log p_\tau$
Bottleneck distance	$d_{\mathrm{BN}}(D_1, D_2) = \inf_\gamma \sup_{x \in D_1} \|x - \gamma(x)\|_\infty$
Wasserstein distance (q)	$d_{W,q}(D_1, D_2) = \left(\inf_\gamma \sum_{x \in D_1} \|x - \gamma(x)\|^q\right)^{1/q}$
Topological meta-features	

Chapter 4

Evaluation

This chapter reports the experimental evaluation of our schema matching approach. We assess: (i) effectiveness, (ii) robustness to structural and semantic variation, and (iii) computational cost. Section 4.1 details datasets, relatedness scenarios, input-modality cases, metrics, and protocol. Sections 4.2 and 4.3 report the experimental results for the unsupervised and the supervised schema matching, respectively.

4.1 Experimental Setup

We implement our approach and its variations using PyTorch version 1.6.0. Most experiments, including meta-feature construction and supervised learning, were conducted on a CPU-based system equipped with an Intel Core i7-10750H processor (6 cores, 2.6 GHz base frequency, up to 5.0 GHz with Turbo Boost) and 16 GB of RAM. For computationally heavier phases, such as embedding generation with pre-trained LLMs, and for certain baseline models relying on deep neural architectures, we leveraged GPU (NVIDIA RTX-series). This setup highlights that our framework can be executed efficiently on CPU-only machines for the meta-space pipeline, while remaining compatible with GPUs.

4.1.1 Datasets

We chose the Valentine benchmark [66] because it is one of the most widely used benchmarks for schema matching, offering broad domain coverage, gold-standard ground truth, controllable noise, and standardized evaluation protocols for fair, reproducible comparisons. The operational details of pair fabrication and noise injection are documented in Koutras's dissertation[1]; here we retain the essential definitions needed to interpret our experiments.

Relatedness scenarios. Given relations $R_1(\mathcal{A})$ and $R_2(\mathcal{B})$:

- **Unionable:** $|\mathcal{A}| = |\mathcal{B}|$ and there exists a bijection $h : \mathcal{A} \to \mathcal{B}$ with semantically equivalent attributes (types may be compatible, not identical).
- **View-unionable:** $\exists\, S_1 \subseteq \mathcal{A}$, $S_2 \subseteq \mathcal{B}$ such that $\pi_{S_1}(R_1)$ and $\pi_{S_2}(R_2)$ are unionable (partial column overlap; typically zero row overlap).

[1] https://pure.tudelft.nl/ws/portalfiles/portal/221944179/christos_koutras_phd_dissertation.pdf

- **Joinable:** $\exists\, A \in \mathcal{A}, B \in \mathcal{B}$ where an equality join on (A, B) is non-trivial (sufficient value overlap under consistent encodings).
- **Semantically-joinable:** As above, but equality is broken by representation differences; normalization or external semantics (e.g., code $\leftrightarrow$ name) restores joinability.

Fabrication protocol (high level). Starting from real base tables, Valentine constructs paired relations and ground truth by:

- *Horizontal splits* to control row overlap for unionability (0%/50%/100%).
- *Vertical splits* to control column overlap for view-unionability and joinability (one shared key or $\{30\%, 50\%, 70\%\}$ shared columns).
- *Combined splits* (vertical + horizontal) to realize zero-overlap view-unionable pairs and realistic joinable cases.

Noise model. Each pair appears in verbatim vs. noisy variants at schema and/or instance level:

- *Schema noise (NS):* table-prefixing, abbreviation, vowel deletion (e.g., `PostalCode`$\rightarrow$`P_Code`).
- *Instance noise (NI):* keyboard-proximity typos for strings; distribution-preserving perturbations for numerics. In Semantically-joinable cases, literals are altered to preserve meaning but defeat equality (e.g., `USA`$\leftrightarrow$`United States`).

We denote verbatim schema/instances by *VS/VI* and noisy schema/instances by *NS/NI*. Where:

- VS/NS: verbatim/noisy schema;
- VI/NI: verbatim/noisy instances.

We evaluate on five Valentine families: *TPC-DI*, *Open Data*, *ChEMBL*, *Wikidata*, and *Magellan* Data, selected to span attribute overlap, syntactic drift, semantic variation, and size diversity. Here, #Pairs refers to the number of data source pairs $(\mathcal{DS}_1, \mathcal{DS}_2)$ provided by each dataset category. Each pair constitutes an independent schema matching task: given two data sources, the goal is to align their attributes under the corresponding relatedness scenario (unionable, joinable, etc.).

TPC-DI (180 pairs). OLTP-style customer/prospect data with moderate regularity. Pairs cover all scenarios via controlled splits; both NS and NI variants are provided.

Open Data (180 pairs). Government tables (Canada/USA/UK) with heterogeneous administrative vocabularies and units. Emphasizes view-unionability (partial schemas, zero row overlap) and semantic joins under inconsistent encodings; NS/NI variants included.

ChEMBL (180 pairs). Bio-activity *assays* aligned to domain ontologies. Emphasizes semantic equivalence when literals diverge; all scenarios and noise settings are instantiated.

Wikidata (4 pairs). Curated US-singer tables with targeted schema changes and value variants (expanded names, aliases). Designed to challenge purely syntactic matchers; focuses on semantic-joinability.

Table 4.1: Dataset sources, pair counts, schema sizes, and tuple ranges. Fabricated pairs: ground truth by construction; curated pairs: manual annotation.

Source	#Pairs	Attributes (min–max)	Tuples (min–max)	Type
TPC-DI	180	11–22	7,492–14,983	Fabricated
Open Data	180	26–51	11,628–23,255	Fabricated
ChEMBL	180	12–23	7,500–15,000	Fabricated
Wikidata	4	13–20	5,423–10,846	Curated
Magellan	7	3–7	864–131,099	Curated

Table 4.2: Six scenarios (matcher inputs and gold).

ID	Scenario	$\mathcal{DS}_1$	$\mathcal{DS}_2$	Gold
C1	Complete ↔ Complete	$(\mathcal{S}_1, \mathcal{I}_1)$	$(\mathcal{S}_2, \mathcal{I}_2)$	$\mathcal{G}^\star$
C2	Schema-only ↔ Schema-only	$(\mathcal{S}_1, \varnothing)$	$(\mathcal{S}_2, \varnothing)$	$\mathcal{G}^\star$
C3	Instance-only ↔ Instance-only	$(\varnothing, \mathcal{I}_1)$	$(\varnothing, \mathcal{I}_2)$	$\mathcal{G}^\star$
D4	Complete ↔ Schema-only	$(\mathcal{S}_1, \mathcal{I}_1)$	$(\mathcal{S}_2, \varnothing)$	$\mathcal{G}^\star$
D5	Complete ↔ Instance-only	$(\mathcal{S}_1, \mathcal{I}_1)$	$(\varnothing, \mathcal{I}_2)$	$\mathcal{G}^\star$
D6	Schema-only ↔ Instance-only	$(\mathcal{S}_1, \varnothing)$	$(\varnothing, \mathcal{I}_2)$	$\mathcal{G}^\star$

Magellan Data (7 pairs). Widely used real pairs (e.g., product/entity collections) that mainly instantiate unionable cases. Narrow schemas but broad size range, probing robustness to sample size and population skew.
Table 4.1. Summary of sources, pair counts, schema sizes, and tuple ranges; fabricated pairs provide ground truth by construction, whereas curated pairs rely on manual annotation.
Table 4.2. Definition of the six input scenarios derived from each pair, together with the corresponding gold alignment $\mathcal{G}^\star$ used for evaluation.

4.1.2 Construction of Dataset Scenarios

All experiments use Valentine pairs and their VS/NS, VI/NI variants. Each pair is

$$\big(\mathcal{DS}_1(\mathcal{S}_1, \mathcal{I}_1),\ \mathcal{DS}_2(\mathcal{S}_2, \mathcal{I}_2),\ \mathcal{G} : \mathcal{S}_1 \to \mathcal{S}_2\big),$$

where $\mathcal{G}$ is the gold 1:1 alignment on attributes.

$\mathcal{S}$: schema; $\mathcal{I}$: instances; $\varnothing$: removed modality; $\mathcal{DS}_t = (\mathcal{S}_t, \mathcal{I}_t)$: matcher inputs;
For each pair, we build six scenarios. The gold set is

$$\mathcal{G}^\star \subseteq \{1, \ldots, n\} \times \{1, \ldots, m\},$$

obtained by restricting $\mathcal{G}^\star$ as pairs (i, j), where i (resp. j) is the i-th column in $\mathcal{DS}_1$ (resp. $\mathcal{DS}_2$). From $\mathcal{DS}_1 = (\mathcal{S}_1, \mathcal{I}_1)$ and $\mathcal{DS}_2 = (\mathcal{S}_2, \mathcal{I}_2)$ under fixed relatedness and VS/NS, VI/NI:

- Schema-only: drop instances $\Rightarrow (\mathcal{S}, \varnothing)$.
- Instance-only: drop schemas $\Rightarrow (\varnothing, \mathcal{I})$.

4.1.3 Evaluation: Metrics and Protocol

We evaluate predicted matches as *index pairs* (i, j) with $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$. The candidate set is $\mathcal{P} = \{1, \ldots, n\} \times \{1, \ldots, m\}$, and the gold set is $\mathcal{G}^\star \subseteq \mathcal{P}$. Each method outputs a score matrix $S = (s_{ij})$; for the unsupervised setting, we also use a ranking ρ that orders pairs by decreasing s_{ij}.

Unsupervised decision. Use a per-pair threshold given by the median of distinct values:

$$\widetilde{\mathcal{V}} = \text{unique}\{s_{ij} : (i, j) \in \mathcal{P}\}, \quad \tau = \text{median}(\widetilde{\mathcal{V}}), \quad \mathcal{A} = \{(i, j) : s_{ij} \geq \tau\}.$$

Supervised decision. Apply a validated threshold $\hat{\tau}$ learned by a classifier trained on disjoint pairs:

$$\mathcal{A} = \{(i, j) : s_{ij} \geq \hat{\tau}\}.$$

Precision, Recall, F1.

$$\text{TP} = |\mathcal{A} \cap \mathcal{G}^\star|, \quad \text{FP} = |\mathcal{A} \setminus \mathcal{G}^\star|, \quad \text{FN} = |\mathcal{G}^\star \setminus \mathcal{A}|.$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \quad \text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

NDCG@k (unsupervised). Let ρ be the ranking of pairs $(i, j) \in \mathcal{P}$ by decreasing score s_{ij}. Given a threshold τ, define

$$k \;=\; \big|\{(i, j) \in \mathcal{P} :\; s_{ij} \geq \tau\}\big| \quad \text{and} \quad (r_1, \ldots, r_k) \text{ the top-}k \text{ items of } \rho,$$

so that $\mathcal{P}_\tau = \{(i, j) \in \mathcal{P} :\; s_{ij} \geq \tau\}$. Let $rel_t = \mathbb{1}\{r_t \in \mathcal{G}^\star\}$ denote an indicator that r_t belongs to the ground-truth set $\mathcal{G}^\star$.

The discounted cumulative gain and its ideal counterpart are

$$\text{DCG@}k = \sum_{t=1}^{k} \frac{rel_t}{\log_2(t + 1)}, \qquad R_\tau = |\mathcal{G}^\star \cap \mathcal{P}_\tau|, \qquad \text{IDCG@}k = \sum_{t=1}^{\min(k, R_\tau)} \frac{1}{\log_2(t + 1)}.$$

The unsupervised NDCG@k is then

$$\text{NDCG@}k = \begin{cases} \text{DCG@}k/\text{IDCG@}k, & \text{if } R_\tau > 0, \\ 0, & \text{otherwise.} \end{cases}$$

This formulation evaluates the threshold-induced top-k and normalizes by the best achievable ordering of the R_τ relevant items.

4.2 Unsupervised Matching

This section details the design and evaluation of our unsupervised framework. It is organized as follows:

- Section 4.2.1 introduces the prompt-based enrichment strategy. We explain how large language models (LLMs) are used to generate the missing components of schema-only and instance-only data sources. Since these prompts differ from those used in the supervised framework, we define them explicitly here.

- Section 4.2.2 presents the experimental evaluation of our unsupervised approach. We describe the baseline methods, metrics, and the research questions that guide our empirical study. This includes assessing effectiveness, efficiency, robustness to noise, and the contribution of each matcher through ablation analysis.
- Finally, we provide a conclusion 4.2.3 summarizing the main takeaways from the unsupervised framework. This discussion highlights the strengths and limitations of the approach.

4.2.1 LLM Enrichment Prompts

We consider the most disparate scenario $\mathcal{DS}_1 = (\mathcal{S}_1, \varnothing) \leftrightarrow \mathcal{DS}_2 = (\varnothing, \mathcal{I}_2)$, in which the two sources share no common modality: $\mathcal{DS}_1$ provides only a schema (no instances), whereas $\mathcal{DS}_2$ provides only instances (no schema). As a result, structural/lexical overlap is minimal, and alignment requires generating both missing modalities with LLM-based enrichment.

Let:

- a_i: the i-th attribute in $\mathcal{S}_1$.
- types(a_i): the declared type of a_i.
- $name_{\mathcal{DS}_1}$: the name of source $\mathcal{DS}_1$.
- W^j: the j-th value column in $\mathcal{I}_2$.
- types(W^j): the inferred/declared type of W^j.
- $name_{\mathcal{DS}_2}$: the name of source $\mathcal{DS}_2$.

In accordance with the current research work of state of the art [96], we have chosen to use prompts consisting of three parts: the first contains attribute information (name, values, and type), the second explains the data generation task, and the last gives the format of the results useful for the other steps in our pipeline.

P1 — Synthetic instances for schema-only attributes ($\mathcal{S}_1$) Per-attribute prompt (for a_i):

> *Provide a list of possible values that are meaningful and representative of the given attribute a_i and its type* types(a_i)*, within the context of $name_{\mathcal{DS}_1}$. The output must be a valid Python list literal, e.g.,* `[value1, value2, ...]`*. Return only the list, with no comments, no headings, and no surrounding prose.*

P2 — Candidate attribute names for instance-only columns ($\mathcal{I}_2$) Per-column prompt (for W^j):

> *Provide a list of possible attribute names that are meaningful and representative of the given values W^j and type* types(W^j) *in the context of $name_{\mathcal{DS}_2}$. The output must be a valid Python list literal, e.g.,* `[attribute1, attribute2, ...]`*. Return only the list, with no comments, no headings, and no surrounding prose.*

P3 — Validation rules for schema-only attributes ($\mathcal{S}_1$) Per-attribute prompt (for a_i):

> *Generate strict validation rules for the attribute a_i in the dataset $\text{name}_{\mathcal{DS}_1}$ with the data type* types(a_i). *Return the rules as a Python list literal, where each element is a concise string describing a constraint (e.g., ranges, regex formats, length checks, uniqueness). Return only the list, with no comments, no headings, and no surrounding prose.*

Enrichment is executed using the exact prompts above (Section 4.2.1); embeddings are then obtained as specified in Section 3.3. The unsupervised pipeline (Section 3.4) consumes these outputs and produces the final alignment $\mathcal{A}$.

4.2.2 Experiments

In this section, we present our experiments aimed at evaluating our data matching approach. We formulate our investigations around the following research questions:

- **RQ1 (Effectiveness)** Does the framework perform effectively across various datasets in terms of F1 score and NDCG? Which combination of large language models (LLM) —one for generating auxiliary information and another for generating embeddings— yields the best performance in these metrics?
- **RQ2 (Baseline)** How does the proposed framework compare to traditional data matching methods—such as COMA++, Similarity Flooding, or Jaccard/Levenshtein—in terms of effectiveness (F1 Score) across all datasets?
- **RQ3 (Efficiency)** Which large language model (LLM) provides the best performance in terms of execution time across various datasets?
- **RQ4 (Noise Sensitivity)** How effectively does the proposed data matching solution perform across varying levels of dataset noise? How consistent is it in maintaining F1 Score and NDCG?
- **RQ5 (Ablation Study)** How does the ablation of our framework matchers impact the performance in terms of F1 Score and NDCG across diverse datasets?

Experimental Setup. The framework was evaluated by testing various models for generating auxiliary information and embeddings. For generating auxiliary information, two models, GPT-3.5 and Llama-3.2, were used. Additionally, seven different embedding models were tested across schema, instance, and hybrid matchers, including BERT (bert-base-uncased), RoBERTa, DistilBERT (distilbert-base-uncased), ALBERT (albert-base-v2), Bart (facebook/bart-base), the Sentence-transformer model (sentence-transformers/all-MiniLM-L6-v2), and GPT-3.5. Table 4.3 summarizes each model's type, layer count, hidden size, parameter count and output dimension. The performance of these combinations was assessed using the F1 score and NDCG metrics. The Type and Rule matchers stayed fixed because they check types and rules, which give binary results. For further details, please refer to this website [2]. We restricted the evaluation to four Valentine families to keep the experiment tractable (371 pairs × 1,000 runs per pair) and balanced: they already cover the relation types and schema-size spectrum we need. Adding Open

[2] https://gitlab.irit.fr/sig/theses/nour-kired/Alignment-of-schema-only-and-instance-only-data.git

Data would mostly increase volume without introducing new conditions for this question, so we defer it outside the baseline section.

Table 4.3: Specifications of embedding models used in our framework.

Model	**Type**	**Layers**	**Hidden Size**	**Params**	**Output Dim**
ALBERT-base-v2	Encoder	12	768	12M	768
BERT-base	Encoder	12	768	110M	768
BART-base	Enc–Dec	6+6	768	139M	768
DistilBERT	Encoder	6	768	66M	768
GPT-3.5 (ada-002)	—	—	1536	—	1536
MiniLM-L6-v2	Encoder	6	384	∼22M	384
RoBERTa-base	Encoder	12	768	125M	768

RQ1. Effectiveness. Table 4.5 highlights that the framework achieves modest results across all combinations, with consistent performance observed for both the F1 score and NDCG. On average, GPT-3.5 achieves an F1 score of 0.53 and an NDCG of 0.71, while Llama-3.2 yields an average F1 score of 0.48 and an NDCG of 0.62. These results suggest the framework works effectively, with GPT-3.5 showing a slight edge over Llama-3.2.

Table 4.5: Performance of LLMs for auxiliary information generation and embedding tasks based on Precision, Recall, F1 Score and NDCG (mean ± std)

Model	**Precision**		**Recall**		**F1 Score**		**NDCG**	
	GPT-3.5	Llama-3.2	GPT-3.5	Llama-3.2	GPT-3.5	Llama-3.2	GPT-3.5	Llama-3.2
ALBERT	.46 ± .29	.42 ± .25	.64 ± .32	.67 ± .26	.52 ± .29	.48 ± .25	.66 ± .30	.51 ± .29
BART	.46 ± .29	.37 ± .23	.65 ± .32	.52 ± .27	.52 ± .29	.42 ± .23	.70 ± .29	.54 ± .30
BERT	.48 ± .29	.42 ± .25	.72 ± .29	.63 ± .28	.54 ± .29	.47 ± .25	.70 ± .29	.64 ± .29
DistilBERT	.48 ± .29	.41 ± .26	.74 ± .27	.60 ± .29	.54 ± .28	.46 ± .26	.71 ± .27	.64 ± .27
GPT	**.51 ± .28**	.44 ± .26	**.84 ± .18**	.70 ± .25	**.58 ± .26**	.50 ± .25	**.84 ± .18**	.79 ± .24
MiniLM	.48 ± .28	.40 ± .25	.75 ± .26	.58 ± .28	.54 ± .27	.45 ± .25	.71 ± .30	.68 ± .26
RoBERTa	.46 ± .27	.39 ± .25	.76 ± .23	.55 ± .29	.53 ± .26	.43 ± .25	.67 ± .29	.51 ± .24

Table 4.6: Wilcoxon signed-rank test results (GPT-3.5 vs Other models)

Model	**Statistic**	**p-value**
ALBERT	8276.0	9.60×10^{-18}
BART	8595.0	3.10×10^{-16}
BERT	9386.5	6.30×10^{-9}
DistilBERT	10582.0	1.12×10^{-6}
MiniLM-L6-v2	9091.5	5.25×10^{-9}
RoBERTa	6374.0	1.57×10^{-13}

Among embedding models, GPT embeddings achieve the best performance (with GPT-3.5 for generation), with an average F1 score of 0.58 and NDCG of 0.84, showing lower variability.

Other models like DistilBERT (F1 = 0.54, NDCG = 0.71) and MiniLM (F1 = 0.54, NDCG = 0.71) perform slightly lower, while RoBERTa (F1 = 0.53, NDCG = 0.67) and BART (F1 = 0.52, NDCG = 0.70) show less effective results. To ensure the validity of these results, we applied the Wilcoxon signed-rank test—appropriate due to the non-normal distribution of our data [66]. The statistical analysis was performed in the context of data generated by GPT-3.5, where GPT embeddings were compared against other embedding models. The Wilcoxon test confirms that the superior performance of GPT embeddings over other models is statistically significant across all evaluations, with p-values consistently below 0.05 (see Table 4.6).

Beyond the performance of LLMs, the relationship type and the number of matches to identify within a pair of data sources could also be key factors. Datasets can be categorized into four types: *Unionable*; *View-Unionable*; *Joinable*; and *Semantically-joinable*, linked via semantic similarities. The *attribute overlap* percentage quantifies schema similarity, directly influencing data matching performance: high overlap enables easier merging, while low overlap necessitates complex techniques.

Table 4.7 shows that **Unionable relations** (100% overlap) lead to the best performance, with GPT embeddings achieving an F1 score of 0.83 and NDCG of 0.87. In contrast, more complex relations like **Semantically-joinable** (8–82% overlap) result in a significant performance drop (mean F1 = 0.50). Figure 4.1 further confirms these observations, illustrating **that higher attribute overlap consistently leads to better results.**

Table 4.7: LLMs for auxiliary information generation and embedding tasks based on Precision, Recall, F1 Score and NDCG (mean ± std) across relations.

Model	Relation	Precision		Recall		F1 Score		NDCG	
		GPT-3.5	Llama-3.2	GPT-3.5	Llama-3.2	GPT-3.5	Llama-3.2	GPT-3.5	Llama-3.2
ALBERT	Joinable	.37 ± .26	.33 ± .22	.60 ± .34	.62 ± .30	.44 ± .28	.41 ± .23	.62 ± .32	.45 ± .29
	Semantically-joinable	.39 ± .26	.33 ± .22	.66 ± .34	.59 ± .31	.46 ± .28	.40 ± .24	.65 ± .32	.47 ± .31
	Unionable	.81 ± .11	.73 ± .12	.81 ± .10	.73 ± .13	.81 ± .10	.73 ± .12	.83 ± .10	.71 ± .12
	View-Unionable	.34 ± .24	.36 ± .21	.52 ± .32	.75 ± .22	.40 ± .26	.44 ± .21	.56 ± .33	.45 ± .29
BART	Joinable	.38 ± .27	.29 ± .21	.64 ± .35	.48 ± .29	.46 ± .30	.35 ± .23	.65 ± .34	.51 ± .33
	Semantically-joinable	.38 ± .25	.31 ± .21	.66 ± .34	.50 ± .31	.46 ± .28	.37 ± .24	.70 ± .30	.51 ± .33
	Unionable	.79 ± .10	.62 ± .09	.79 ± .10	.62 ± .09	.79 ± .10	.62 ± .09	.82 ± .09	.73 ± .13
	View-Unionable	**.35 ± .24**	.31 ± .20	.54 ± .34	.49 ± .29	.41 ± .28	.37 ± .23	.65 ± .30	.46 ± .30
BERT	Joinable	.38 ± .26	.34 ± .23	.68 ± .33	.59 ± .31	.46 ± .28	.40 ± .25	.66 ± .32	.60 ± .32
	Semantically-joinable	.39 ± .26	.33 ± .24	.70 ± .32	.56 ± .33	.47 ± .28	.40 ± .26	.68 ± .31	.55 ± .32
	Unionable	.82 ± .11	.70 ± .11	.82 ± .11	.70 ± .11	.82 ± .11	.70 ± .11	.84 ± .08	.79 ± .11
	View-Unionable	.38 ± .24	.35 ± .21	.69 ± .31	.69 ± .26	.46 ± .26	.42 ± .22	.66 ± .30	.63 ± .31
DistilBERT	Joinable	.40 ± .26	.32 ± .22	.74 ± .30	.56 ± .31	.48 ± .27	.39 ± .24	.67 ± .31	.60 ± .29
	Semantically-joinable	.39 ± .25	.33 ± .23	.73 ± .30	.54 ± .32	.47 ± .26	.40 ± .26	.69 ± .28	.61 ± .28
	Unionable	.82 ± .12	.71 ± .12	.82 ± .12	.71 ± .12	.82 ± .12	.71 ± .12	**.87 ± .08**	.78 ± .10
	View-Unionable	.37 ± .24	.34 ± .21	.69 ± .30	.61 ± .30	.45 ± .25	.41 ± .23	.65 ± .28	.59 ± .29
GPT	Joinable	**.42 ± .25**	.34 ± .22	**.84 ± .20**	.63 ± .30	**.51 ± .25**	.42 ± .24	**.82 ± .23**	.72 ± .31
	Semantically-joinable	**.41 ± .24**	.34 ± .22	**.84 ± .20**	.64 ± .29	**.50 ± .25**	.42 ± .24	**.82 ± .20**	.72 ± .30
	Unionable	**.83 ± .11**	.75 ± .12	**.83 ± .11**	.74 ± .13	**.83 ± .11**	.74 ± .13	**.87 ± .08**	.86 ± .07
	View-Unionable	**.42 ± .24**	.38 ± .21	**.85 ± .19**	.79 ± .18	**.51 ± .25**	.46 ± .22	**.87 ± .15**	.87 ± .15
MiniLM	Joinable	.39 ± .26	.32 ± .21	.73 ± .30	.55 ± .31	.48 ± .27	.39 ± .24	.69 ± .32	.65 ± .30
	Semantically-joinable	.39 ± .24	.32 ± .22	.73 ± .29	.57 ± .31	.47 ± .26	.38 ± .24	.68 ± .34	.65 ± .30
	Unionable	.80 ± .10	.68 ± .13	.80 ± .10	.68 ± .13	.80 ± .10	.68 ± .13	.86 ± .08	.76 ± .10
	View-Unionable	.38 ± .24	.32 ± .21	.75 ± .27	.56 ± .31	.47 ± .25	.38 ± .23	.63 ± .33	.68 ± .25
RoBERTa	Joinable	.38 ± .24	.30 ± .21	.74 ± .27	.49 ± .30	.46 ± .25	.36 ± .24	.65 ± .30	.50 ± .25
	Semantically-joinable	.38 ± .24	.31 ± .22	.77 ± .24	.50 ± .31	.47 ± .25	.37 ± .25	.66 ± .31	.51 ± .26
	Unionable	.78 ± .13	.69 ± .14	.78 ± .13	.69 ± .14	.78 ± .13	.69 ± .14	.84 ± .09	.60 ± .16
	View-Unionable	.36 ± .21	.31 ± .20	.75 ± .23	.54 ± .30	.45 ± .22	.37 ± .22	.57 ± .31	.45 ± .22

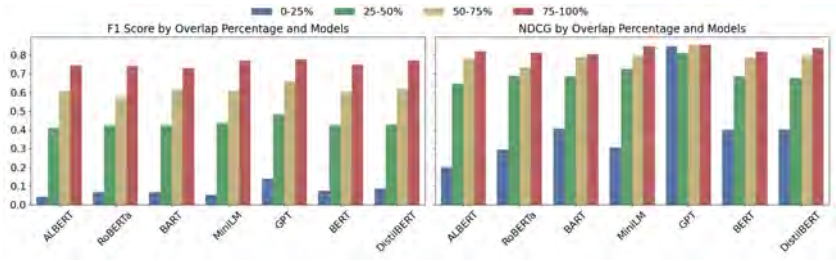


Figure 4.1: Average F1 Score and NDCG for GPT-3.5 across embeddings models and overlap attributes.

Table 4.8: Baseline F1 and Top F1 Scores (mean ± std)

Algorithm	**Distance**	**F1** (mean ± std)	**F1 (Top)** (mean ± std)
COMA_INS	–	.33 ± .34	.33 ± .34
COMA_SCH	–	.33 ± .34	.33 ± .34
JaccardMatcher	DamerauLevenshtein	.30 ± .25	.31 ± .25
JaccardMatcher	Hamming	.31 ± .26	.31 ± .26
JaccardMatcher	JaroWinkler	**.39 ± .25**	**.41 ± .25**
JaccardMatcher	Levenshtein	.32 ± .26	.33 ± .26
SimilarityFlooding	–	.10 ± .11	.11 ± .12

Discussion. GPT-3.5 is the best choice for auxiliary information generation, producing high-quality inputs that enhance embedding performance. GPT embeddings consistently outperform other models across all tasks, delivering reliable and high-quality results with lower variability. While simpler Unionable relations achieve the best outcomes, the robustness of GPT embeddings and GPT-3.5's generation capabilities ensure competitive performance even in more complex relations requiring semantic reasoning.

RQ2. Baseline. To address RQ2, we compared the effectiveness of our LLM-based matching framework with traditional approaches, including COMA++, Similarity Flooding, and string similarity methods (Jaccard combined with Damerau-Levenshtein, Hamming, Jaro-Winkler, and Levenshtein). The experiments were conducted on 371 datasets. For each dataset, we generated auxiliary schemas and instances using GPT-3.5, sampling 100 schema variations for instances-only data sources and repeating the process 10 times, resulting in a total of 1000 executions per dataset.

In this setting, noise refers to the variability in the quality of the generated schemas. Some of the proposed schemas can significantly deviate from the real underlying schema, introducing inconsistencies and ambiguity in the matching process. In addition to the average F1 scores, we also report the Top F1 score. The Top F1 represents the best F1 score obtained for each dataset across all schema variations. It reflects the optimal potential performance of each method under ideal conditions, where the generated schema is most aligned with the target.

Discussion. Table 4.8 shows that traditional methods, such as COMA_INS and COMA_SCH, achieve low F1 scores (.33 ± .34), with minimal improvement in their Top F1 scores. Similarity Flooding performs worse, with an F1 of .10 ± .11. Among string similarity methods, Jaro-Winkler yields the highest F1 (.39 ± .25) and a Top F1 of .41 ± .25.

However, the limited gains between the average F1 and Top F1 scores suggest that traditional

approaches struggle to adapt to heterogeneous and noisy data, even when the generated schemas are closer to the real ones. In contrast, our LLM-based framework significantly outperforms these baselines, offering higher accuracy and robustness across all datasets (see Table 4.5).

In particular, compared to the best-performing traditional method (Jaro-Winkler, F1 = 0.39), our LLM-based framework with GPT embeddings (F1 = 0.58) achieves an absolute gain of +0.19, which corresponds to a relative improvement of +48.7%. Similarly, when compared to COMA-based methods (F1 = 0.33), the gain reaches +0.25 in absolute terms, i.e., a relative improvement of +75.7%.

Table 4.9: Execution times for auxiliary-information models (GPT-3.5, Llama-3.2) across dataset families and embedding models.

(a) Execution times by Dataset Source.

Dataset Source	**GPT (s)**	**Llama (s)**
ChEMBL	212.89	128.34
TPC-DI	139.84	144.89
Wikidata	704.90	930.10
Magellan	116.60	76.75

(b) Execution times by embeddings model.

Model	**GPT (s)**	**Llama (s)**
ALBERT	150.72	82.11
BART	432.64	166.94
BERT	151.94	166.94
DistilBERT	104.65	107.82
GPT	181.51	101.80
MiniLM	68.77	65.67
RoBERTa	174.82	177.49

(c) Magellan datasets characteristics and execution times for GPT-3.5 as an auxiliary information model and GPT as an embeddings model.

Dataset	**#Rows**	**#Atts**	**GPT (s)**	**Llama (s)**
amazon_google_exp	3,226	4	13.26	15.94
beeradvo_ratebeer	3,000	5	21.08	23.60
dblp_acm	2,349	5	25.95	29.28
walmart_amazon	22,076	6	41.60	35.87
fodors_zagats	331	7	56.50	52.37
dblp_scholar	66,774	5	37.54	29.73
itunes_amazon	57,742	9	149.95	124.15

RQ3. Efficiency. Table 4.9a shows that execution times depend on the dataset source and its characteristics. Wikidata, with 13–20 attributes, exhibits the longest execution times (704.90 s for GPT-3.5 and 930.10 s for Llama-3.2). In contrast, smaller datasets with fewer attributes, such as Magellan (4–9 attributes), are processed significantly faster (116.60 s for GPT-3.5 and 76.75 s for Llama-3.2). Table 4.9b highlights variations in execution times across embedding models. Llama-3.2 demonstrates a range of execution times, from 65.67 s (MiniLM) to 177.49 s (RoBERTa). Similarly, GPT-3.5 processes lightweight models like MiniLM efficiently (68.77 s) but requires substantially more time for models with larger architectures and deeper layers, such as BART (432.64 s). Table 4.9c shows that execution times for Magellan datasets scale with the number of attributes and rows. Smaller datasets, such as *amazon_google_exp* (13.26 s for GPT-3.5), are processed faster, whereas larger datasets, like *itunes_amazon* (149.95 s for GPT-3.5), take significantly longer due to their higher row counts (57,742 rows) and attribute

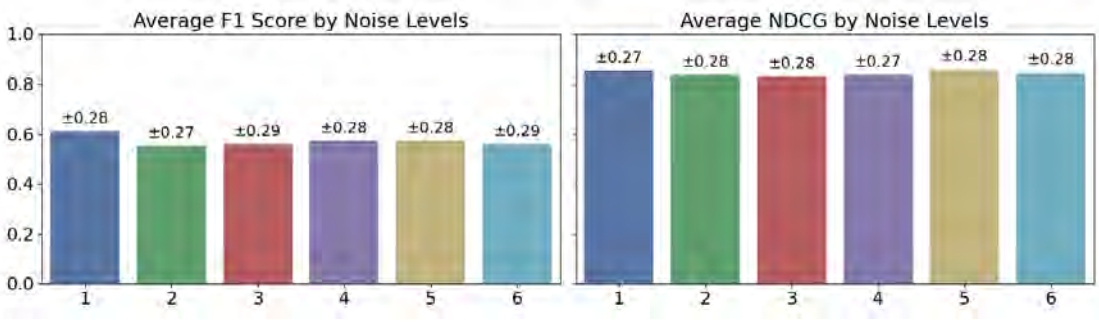


Figure 4.2: Impact of noise levels on F1 Score and NDCG across datasets for GPT embeddings model using GPT-3.5 for generation.

size (9 attributes).

Discussion. Execution times are influenced by dataset characteristics (#Atts, #Rows), generation models, and embedding models. While Llama-3.2 is generally faster, GPT-3.5 performs better on datasets with complex relationships between attributes, such as Wikidata. The number of attributes is the primary factor affecting execution time, as auxiliary information and embeddings are generated for each attribute-instance pair. Lightweight embedding models like MiniLM are efficient but may struggle with datasets requiring more detailed or nuanced analysis. GPT-3.5, although slower ($\approx 2.67\times$ slower than MiniLM; 181.51 s vs. 68.77 s), delivers superior performance on tasks involving higher attribute counts or complex data, making it a dependable choice for tasks requiring both accuracy and stability.

RQ4. Noise Sensitivity. To assess robustness, varying levels of noise were introduced across the datasets at the attribute level. These transformations simulate real-world inconsistencies in attribute naming, including typographical changes, structural modifications, and abbreviations. The noise levels range from minimal changes, such as slight typographical errors, to extreme distortions, where attribute names become highly unclear or ambiguous. For example, the attribute *assay_subcellular_fraction* progressively becomes *ASS_SUBCFR* (abbreviation), then *ss_sbclllr_frctn* (aggressive vowel removal), and finally *b_frmt* (complete semantic drift). These perturbations challenge the matching process by reducing or removing semantic cues, reflecting realistic scenarios in schema integration tasks. This approach ensures a rigorous evaluation of the algorithm's ability to adapt to inconsistencies in heterogeneous datasets.

Figure 4.2 shows the F1 Score and NDCG performance for various noise levels. The evaluation uses GPT-3.5 for embeddings and auxiliary information generation, identified as the most effective combination in RQ1. The F1 Score remains stable, ranging from 0.57 ± 0.29 at level 1 (minimal changes, e.g., typographical errors) to 0.60 ± 0.28 at level 6 (significant distortions). This indicates minimal impact of noise on performance. Similarly, NDCG values show strong consistency, ranging from 0.79 ± 0.28 to 0.80 ± 0.28, with marginal standard deviation differences, demonstrating the solution's ability to maintain ranking quality despite noise.

Discussion. The solution's robustness across different noise levels highlights its effectiveness in handling schema-only and instance-only data sources. The minimal impact of schema-level noise, such as typographical or structural changes, is attributed to GPT's data enrichment process, which contextualizes noise and enhances comprehension. In fact, the apparent stability under noisy conditions can be explained by the intrinsic setting of our problem: since one data source already lacks either schema or instances, the information gap dominates the matching difficulty.

Injecting additional noise in the available part (schema or instances) does not drastically worsen the situation, because the framework is already designed to compensate for missing information. In other words, the "handicap" of schema-only versus instance-only matching inherently absorbs part of the injected noise. This robustness ensures high performance in both classification and ranking tasks, making the solution well-suited for real-world noisy datasets. Nevertheless, we acknowledge that in scenarios where both schema and instances are fully available, noise may have a stronger impact than what is observed here.

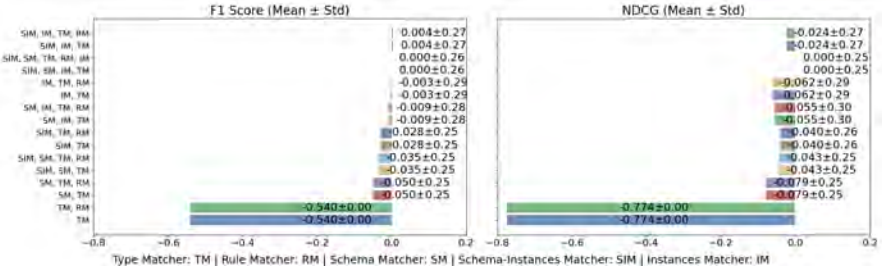


Figure 4.3: Impact of matcher ablation on F1 Score and NDCG for the GPT embeddings model across diverse datasets using GPT-3.5 for generation auxiliary information.

Table 4.10: Logistic regression results and model statistics for the optimal matcher combination tested on the GPT embeddings model using GPT-3.5 for generation.

Regression Results		
Metric	**Value**	***p*-value**
Intercept	-1.96	< 0.001 *
Instance Matcher	1.82	0.028 *
Schema-Instance matcher	2.63	0.005

Model Stats	
Metric	**Value**
Null Deviance	116.18
Residual Deviance	24.27
AIC	279.78
McFadden's R^2	0.791

RQ5. Ablation. The RQ4 experiments analyzed the impact of ablating individual matchers on the framework's performance (F1 Score and NDCG), using GPT-3.5 for embeddings and auxiliary information generation. The findings highlight the critical contribution of specific matchers to the framework's overall performance. Type Matcher was always included as a default component since it serves as a prerequisite for activating other matchers when the type condition is satisfied. Regression analysis identified Instance Matcher and Schema-Instance matcher (Considering Type Matcher) as the most significant contributors, with statistically significant p-values of 0.0282 and 0.0054, respectively (Table 4.10). These matchers substantially reduced the model's AIC from 497.18 to 279.78, demonstrating improved model quality. The model explained 79% of the variance (McFadden's $R^2 = 0.791$), further confirming the importance of these matchers. Performance analysis of matcher combinations in Figure 4.3 revealed that the optimal configuration includes Instance Matcher, Schema-Instance matcher, and Type Matcher, which consistently achieved the highest results. This combination produced an F1 Score increase of (0.004 ± 0.27) but an NDCG decrease of (-0.024 ± 0.27), in comparison with "All matchers" configuration, which scored (0.54 ± 0.26) for F1 and (0.79 ± 0.25) for NDCG.

Discussion. The findings underscore the critical role of Instance Matcher and Schema-Instance matcher in achieving optimal performance. These matchers demonstrated significant contributions not only statistically but also practically, enhancing precision but decrease ranking quality. The observed reduction in AIC suggests improved model parsimony and efficiency, while the slight improvements in F1 Score (+0.004) emphasize the impact of small optimizations on overall performance.

The results also highlight the potential to further improve performance by excluding less impactful matchers, such as Rule Matcher and Schema Matcher, which may detract from the overall quality. This suggests that specific matcher combinations may perform better for certain dataset characteristics, presenting an opportunity to develop recommendations tailored to dataset attributes. These findings align with the hypothesis that matchers interact synergistically, emphasizing the need to adapt matcher combinations dynamically for different data scenarios.

4.2.3 Conclusion

This section introduced an unsupervised framework for aligning schema-only and instance-only data sources by leveraging large language models (LLMs) to generate auxiliary information and combining multiple schema-instance matchers. Our evaluation on public benchmarks demonstrated the effectiveness and robustness of the approach, particularly in heterogeneous and noisy settings.

Compared to traditional baselines such as COMA or Jaccard/Levenshtein, which achieve average F1 scores around 0.33–0.39, our best-performing configuration (GPT embeddings with GPT-3.5 generation) reaches an average F1 score of 0.58. This corresponds to an improvement of approximately +75.7% over COMA (0.33 $\rightarrow$ 0.58) and +48.7% over Jaro-Winkler (0.39 $\rightarrow$ 0.58). These gains confirm the substantial advantage of LLM-based enrichment in heterogeneous and noisy contexts, where traditional methods fail to adapt.

Several important limitations, however, remain open. First, as our experiments rely on public benchmark data, one cannot exclude the possibility that LLMs leverage prior knowledge to infer correct schema information, which could artificially inflate robustness. Future experimentation will therefore involve domains unseen during LLM training or fully synthetic datasets specifically designed to test this risk. Second, while our results on noisy data showed limited degradation, this robustness can be partly explained by the intrinsic handicap of schema-only versus instance-only alignment, where missing information dominates over injected noise. Further controlled experiments will investigate whether similar robustness holds in more balanced scenarios where both schema and instances are available. Third, data confidentiality remains a crucial concern: although only public data were used here, applying LLM-based enrichment in enterprise contexts will require secure processing and privacy-preserving strategies.

From a methodological standpoint, our ablation study revealed that not all matchers contribute equally to the final similarity matrix. While our current integration uses uniform averaging, future work will explore differential weighting, potentially with learned weights, to optimize contributions from each matcher. Moreover, although we mainly reported F1 and NDCG, precision and recall values—as well as additional experimental results—are available on the companion website, offering deeper insights into trade-offs between correctness and completeness. Still, designing richer evaluation protocols remains an avenue for improvement. Finally, ambiguity in attribute names continues to pose challenges: despite contextual anchoring and rule-based filtering, LLMs may misinterpret polysemous attributes. This suggests that specialized LLM architectures trained for schema and instance inference could provide more reliable results. A promising direction is therefore the design of LLMs specifically tailored to schema and instance

generation, which could reduce errors due to polysemy and strengthen privacy guarantees.

In short, the experiments demonstrate that LLM-based enrichment yields substantial improvements over traditional baselines, with relative gains of up to +75% in F1 score. At the same time, they open avenues for addressing robustness, privacy, and specialization of models for more reliable real-world deployments.

4.3 Supervised Matching

This section presents the evaluation of our supervised framework, introducing two central components: a *meta-space* that encodes each attribute pair into a multi-dimensional vector of meta-features, and a *meta-learner* that exploits these features to predict semantic alignments. The section is structured as follows:

- Section 4.3.1 describes the prompt-based enrichment strategy used to handle incomplete sources. Unlike the unsupervised case where enrichment is performed *column by column*, here large language models generate entire *schemas* or *instances*. We also evaluate the quality and limitations of this enrichment process.
- Section 4.3.2 details the experimental study of the supervised framework. We construct the meta-space M_{ms}, combining *classical*, *spectral*, and *topological* meta-features into vectors, and train supervised classifiers to learn adaptive combinations of these meta-features. Our study addresses multiple research questions, including the effectiveness and efficiency of the framework, the relative importance of meta-feature families, and the impact of meta-feature selection.
- Section 4.3.3 concludes with a summary of the main findings. We highlight how the supervised framework achieves flexibility, interpretability, and robustness with limited annotations, while also discussing its current limitations.

4.3.1 LLM Enrichment Prompts

We consider a tabular source $\mathcal{DS} = (\mathcal{S}, \mathcal{I})$ that can be schema-only $(\mathcal{S}, \varnothing)$, instance-only $(\varnothing, \mathcal{I})$, or complete $(\mathcal{S}, \mathcal{I})$, with metadata $\mathcal{MD}$ (types, dataset name).

In accordance with the current research work of state of the art [96], we have chosen to use prompts consisting of three parts: the first contains data information (schema, instances, and attribute types), the second explains the data generation task, and the last gives the format of the results useful for the other steps in our pipeline.

For reproducibility, we list the exact prompts.

Instance generation (schema-only). *You are given:*
- *Dataset name:* `<dataset_name>`
- *Schema:* `<schema>`
- *Attribute types:* `<types>`
- *Target number of instances:* `n`

Task: Generate exactly `n` *realistic instances consistent with the above.*

Output strictly as a Python object (list of lists of strings), single quotes, no extra text.
Shape: `[['v11','v12',...], ['v21','v22',...], ...]`.

Schema generation (instance-only). *You are given:*
- *Dataset name:* `<dataset_name>`

– *Instances:* `<instances>`
– *Inferred column types:* `<types>`
– *Max candidate schemas:* `max_schemas`

Task: Propose up to `max_schemas` *candidate schemas.*

Output strictly as a Python object (list of lists of attribute names), single quotes, no extra text.
Shape: `[['a1','a2',...], ['b1','b2',...]]`.

All enrichment runs use `gpt-4o-mini`.

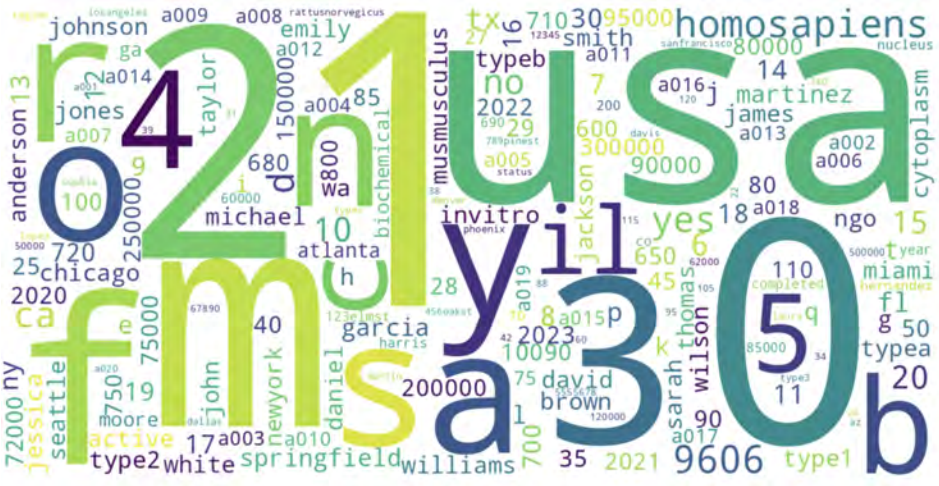


Figure 4.4: Word cloud of the most frequent generated values (model: `gpt-4o-mini`).

Table 4.11: Evaluation of synthetic generation (`gpt-4o-mini`). We report process metrics (*Attempts*, *Avg. Time (s)*) and output metrics (*Avg. Row/Col Diff*, *Avg. Null Ratio*, *Avg. Invalid Ratio*) for $\mathcal{I}^{\text{gen}}$ and $\mathcal{S}^{\text{gen}}$.

Category (#Pairs)	Relation	Attempts		Avg. Time (s)		Avg. Row Diff		Avg. Col Diff		Avg. Null Ratio		Avg. Invalid Ratio	
		$\mathcal{I}^{\text{gen}}$	$\mathcal{S}^{\text{gen}}$	$\mathcal{I}^{\text{gen}}$	$\mathcal{S}^{\text{gen}}$	$\mathcal{I}^{\text{gen}}$	$\mathcal{S}^{\text{gen}}$	$\mathcal{I}^{\text{gen}}$	$\mathcal{S}^{\text{gen}}$	$\mathcal{I}^{\text{gen}}$	$\mathcal{S}^{\text{gen}}$	$\mathcal{I}^{\text{gen}}$	$\mathcal{S}^{\text{gen}}$
ChEMBL	Joinable	102.12	1.00	985.28	6.66	-11634.53	-12185.42	-0.05	-0.69	0.00	0.01	0.00	0.01
	Semantically-joinable	100.31	1.00	465.75	5.85	-12707.63	-12185.36	0.00	-0.78	0.01	0.02	0.01	0.02
	Unionable	100.43	1.00	573.88	8.08	-8838.86	-9373.42	-0.27	-2.36	0.00	0.02	0.00	0.02
	View-Unionable	103.25	1.02	1216.99	6.08	-6054.49	-7497.79	-0.05	-0.67	0.00	0.01	0.00	0.01
Magellan	Unionable	72.43	67.05	875.58	77.81	-10732.64	-11975.79	0.21	0.00	0.01	0.00	0.01	0.00
OpenData	Joinable	1.44	16.84	56.36	123.49	-18877.39	-18883.08	-0.89	-0.92	0.00	0.02	0.00	0.02
	Semantically-joinable	1.27	1.00	54.61	12.65	-18877.20	-18891.85	-1.25	-0.92	0.00	0.01	0.00	0.01
	Unionable	1.56	1.00	129.17	19.53	-14518.83	-14532.58	-1.38	-3.69	0.01	0.00	0.01	0.00
	View-Unionable	41.44	67.26	1486.58	370.13	-11314.11	-11499.19	-1.00	-0.77	0.01	0.09	0.01	0.09
TPC-DI	Joinable	89.20	1.00	854.95	5.34	-11593.81	-12170.04	-0.04	-0.14	0.00	0.00	0.00	0.00
	Semantically-joinable	101.14	1.02	1060.54	5.77	-11188.60	-12170.09	-0.04	-0.07	0.00	0.00	0.00	0.00
	Unionable	100.97	1.00	1328.13	8.99	-8177.26	-9361.46	-0.06	-0.07	0.00	0.00	0.00	0.00
	View-Unionable	48.61	1.02	359.88	4.96	-7220.32	-7488.60	-0.03	-0.04	0.00	0.00	0.00	0.00
Wikidata	Joinable	54.75	50.25	1122.35	80.02	-9664.00	-10784.00	0.00	-1.50	0.01	0.01	0.01	0.01
	Semantically-joinable	116.00	101.50	2561.12	221.61	-9968.00	-10718.50	0.00	0.00	0.00	0.02	0.00	0.02
	Unionable	116.00	83.50	3076.67	580.48	-9293.00	-10776.50	0.00	-5.00	0.00	0.06	0.00	0.06
	View-Unionable	107.50	81.00	1742.02	759.18	-3832.50	-5335.00	0.00	-1.00	0.01	0.03	0.01	0.03

LLM Enrichment Evaluation

We evaluate two enrichment tasks with `gpt-4o-mini`: (i) instance generation ($\mathcal{I}^{\text{gen}}$) from a given schema, and (ii) schema generation ($\mathcal{S}^{\text{gen}}$) from given instances. Table 4.11 reports simple,

observable process and output metrics to characterize the enrichment; for clarity and reproducibility, we define each metric below.

- **Parsable**: an output that matches the required format (Python list with single quotes) and passes basic validity checks.
- **Attempts**: number of model tries until the first parsable output (averaged over pairs).
- **Avg. Time (s)**: Execution time from the first try to the first parsable output; includes retries, validation, and any API backoff.
- **Avg. Row Diff**: $\#\text{rows}_{\text{enriched}} - \#\text{rows}_{\text{original}}$ (negative = under-generation).
- **Avg. Col Diff**: $\#\text{cols}_{\text{enriched}} - \#\text{cols}_{\text{original}}$ (negative = missing attributes; positive = extra attributes).
- **Avg. Null Ratio**: fraction of cells that are empty or explicit `null`-like blanks in the enriched table.
- **Avg. Invalid Ratio**: fraction of cells that contain value tokens (e.g., `"null"`, `"n/a"`, `"unknown"`, `"field"`).

Schema generation is fast and stable: ≈1 attempt and sub-20 s latencies across categories. Instance generation is slower and less stable: many retries, long latencies (often in the hundreds or thousands of seconds), large row deficits due to the 20-row cap, and column counts that are mostly preserved. Null ratios are low ($< 1\%$ in most cases), but invalid ratios can be non-negligible for some categories (e.g., OpenData View-Unionable; Wikidata Unionable).

We observe that schema generation is faster and more stable than instance generation. This is potentially due to: (i) *output size*—each instance call emits up to $20 \times |\mathcal{S}|$ values (often long strings), whereas a schema call emits at most $5 \times |\mathcal{S}|$ short labels; (ii) *token length and variability*—instance values (names, addresses, IDs, phrases) are longer and more diverse than attribute names, which raises decoding time; (iii) *format sensitivity*—instance outputs fail strict parsing more often, which triggers retries and repeated validation; and (iv) *API backoff*—occasional throttling introduces idle time. Other, unknown factors may also contribute.

Figure 4.4 shows the most frequent generated tokens. They align with our dataset families: *ChEMBL* → biomedical terms and IDs (`9606`, *Homo sapiens*, `nucleus`, `cytoplasm`); *Open Data* → U.S. geo/administrative tokens (`usa`, state codes `ca`/`tx`/`ny`, cities like `chicago`, `miami`); *TPC-DI* → customer/contact placeholders (names, addresses, phone-like `555xxxx`, ZIP-like `90210`); *Wikidata* → person names and aliases (`smith`, `johnson`, `emily`, `david`); *Magellan* → generic product/entity identifiers and simple category codes. Across all sets, short tokens dominate (digits; single letters), which speeds generation but reduces semantic richness. Taken together, these patterns match our data mix (geo-administrative tables, person/entity records, biomedical resources). The model can produce domain-specific terms when needed (e.g., taxon IDs, cell compartments, U.S. places), but it still prefers short, frequent tokens and templates. This makes generation fast and easy to parse, but it can reduce detail and shift the value distribution away from the originals.

Limitations. In our experiments, we did not fully control the generated data. In particular: (i) no systematic semantic checks (meaning, domain logic); (ii) no systematic syntax checks (types, formats); (iii) no rule/pattern checks (regex, ranges, keys). We also did not tune prompts for a best configuration, and we did not audit missing data in a rigorous way. The results may therefore contain errors, gaps, or biases.

Discussion. Now that the data has been generated, we observe in practice that the *schema* is generated faster than the *instances*, and that we can reliably recover the *schema size*. By contrast, for instances it is harder to recover both the values and—especially—the original instance count, with a likely loss of variability and detail. Our analyses also indicate that the generated data remains diverse and consistent with the target families (biomedical for ChEMBL, geo-administrative for Open Data, customers/contacts for TPC-DI, names/people for Wikidata, product identifiers for Magellan), even though *short tokens* (numbers, single letters) dominate—speeding generation but potentially reducing semantic richness.

4.3.2 Experiments

In this section, we report experiments that evaluate the proposed supervised data-matching model. Our framework considers six schema-matching scenarios determined by the completeness of the two input sources ($\mathcal{DS}_1$ and $\mathcal{DS}_2$):
Compliant:

(C1) Complete $(\mathcal{S}_1, \mathcal{I}_1) \leftrightarrow$ Complete $(\mathcal{S}_2, \mathcal{I}_2)$

(C2) Schema-only $(\mathcal{S}_1, \varnothing) \leftrightarrow$ Schema-only $(\mathcal{S}_2, \varnothing)$

(C3) Instance-only $(\varnothing, \mathcal{I}_1) \leftrightarrow$ Instance-only $(\varnothing, \mathcal{I}_2)$

Disparate:

(D4) Complete $(\mathcal{S}_1, \mathcal{I}_1) \leftrightarrow$ Schema-only $(\mathcal{S}_2, \varnothing)$

(D5) Complete $(\mathcal{S}_1, \mathcal{I}_1) \leftrightarrow$ Instance-only $(\varnothing, \mathcal{I}_2)$

(D6) Schema-only $(\mathcal{S}_1, \varnothing) \leftrightarrow$ Instance-only $(\varnothing, \mathcal{I}_2)$

We investigate the following research questions across all six scenarios.

- **RQ1 (Effectiveness).** How effective is the proposed framework at aligning heterogeneous tabular sources across scenarios in terms of F1-Score, Precision, and Recall? How stable is this effectiveness across multiple runs, diverse datasets, embedding models, and classifiers?
- **RQ2 (Efficiency).** What are the computational costs, in terms of runtime, of each framework step (embedding, meta-feature computation, and classification) across scenarios? How does runtime vary across embedding models and classifiers? Can we balance performance and efficiency in this setting?
- **RQ3 (Meta-Feature Importance).** Which families of meta-features (classical, spectral, topological) contribute the most to the overall alignment performance? What is the impact of removing one or more families on the stability and effectiveness of the framework across scenarios and embedding models?
- **RQ4 (Meta-Feature Selection).** Can automatic selection of meta-features improve performance or stability?
- **RQ5 (Baselines).** How does the proposed framework compare to supervised schema matching baselines in terms of F1-Score, Precision, and Recall, when evaluated across all scenarios?

Experimental Setup

We use pre-trained transformer embeddings—BERT, RoBERTa, DistilBERT, MiniLM, ALBERT, and BART—in zero-shot mode. Table 4.12 reports each model's type, layer count, hidden size, parameter count, and output dimension. These embeddings form the basis for our meta-space matching.

For evaluation, each meta-space matrix is split 70/30 (train/test) with stratification to preserve the positive/negative ratio. We run 10 different random seeds 10 times and report the mean and std results. In every run, classifiers are retrained from scratch on the 70% split and evaluated on the held-out 30%. No hyperparameter tuning is performed; we use library defaults (scikit-learn v1.2.2, XGBoost v1.7.5, CatBoost v1.1.1). This protocol ensures reproducibility and a fair comparison across embeddings, classifiers, and seeds.

Table 4.12: Specifications of embedding models used in our framework.

Model	Type	Layers	Hidden Size	Params	Output Dim
BERT-base	Encoder	12	768	110M	768
RoBERTa-base	Encoder	12	768	125M	768
DistilBERT	Encoder	6	768	66M	768
ALBERT-base-v2	Encoder	12	768	12M	768
BART-base	Enc-Dec	6+6	768	139M	768
MiniLM-L6-v2	Encoder	6	384	∼22M	384

RQ1. Effectiveness

Table 4.13: Precision, Recall, and F1-score (mean± std) per classifier and embedding model for scenario (S1, I1) ↔ (S2, I2).

Classifier	Metric	albert	bart	bert	distilbert	minilm	roberta	Classifier Mean
RandomForest	Precision	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00
	Recall	0.82 ± 0.01	0.82 ± 0.01	0.82 ± 0.01	0.82 ± 0.01	0.83 ± 0.01	0.82 ± 0.01	0.82 ± 0.01
	F1	0.90 ± 0.00	0.90 ± 0.00	0.90 ± 0.00	0.90 ± 0.00	0.90 ± 0.00	0.90 ± 0.00	0.90 ± 0.00
LogisticRegression	Precision	0.30 ± 0.00	0.30 ± 0.00	0.30 ± 0.00	0.30 ± 0.00	0.30 ± 0.00	0.30 ± 0.00	0.30 ± 0.00
	Recall	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00
	F1	0.46 ± 0.00	0.46 ± 0.00	0.46 ± 0.00	0.46 ± 0.00	0.46 ± 0.01	0.46 ± 0.00	0.46 ± 0.00
KNN	Precision	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00
	Recall	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01
	F1	0.84 ± 0.01	0.84 ± 0.01	0.84 ± 0.01	0.84 ± 0.01	0.84 ± 0.01	0.84 ± 0.01	0.84 ± 0.01
MLP	Precision	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01
	Recall	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.89 ± 0.02	0.90 ± 0.01	0.90 ± 0.01
	F1	0.90 ± 0.00	0.90 ± 0.00	0.90 ± 0.00	0.90 ± 0.00	0.90 ± 0.01	0.90 ± 0.00	0.90 ± 0.00
CatBoost	Precision	0.88 ± 0.02	0.88 ± 0.02	0.88 ± 0.02	0.88 ± 0.02	0.87 ± 0.02	0.88 ± 0.02	0.88 ± 0.02
	Recall	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00
	F1	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01
XGBoost	Precision	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	$\mathbf{0.96 \pm 0.00}$
	Recall	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	$\mathbf{0.97 \pm 0.00}$
	F1	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	$\mathbf{0.97 \pm 0.00}$
Embedding Mean	Precision	0.84 ± 0.01	0.84 ± 0.01	0.84 ± 0.01	0.84 ± 0.01	0.84 ± 0.01	0.84 ± 0.01	–
	Recall	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	–
	F1	0.83 ± 0.00	0.83 ± 0.00	0.83 ± 0.00	0.83 ± 0.00	0.83 ± 0.01	0.83 ± 0.00	–

We train one joint classifier on 551 pairs of datasets from five Categories (*TPC-DI*, *Open Data*, *ChEMBL*, *Magellan*, *Wikidata*). The data are split globally with 70% for training and 30% for testing. For each schema, we shuffle the data using 10 random seeds, and for every fixed seed we repeat the run 10 times. We report the mean and standard deviation across these repetitions.

Table 4.14: Precision, Recall, and F1-score (mean± std) per classifier and embedding model for scenario (S1,∅) ↔ (S2,∅).

Classifier	**Metric**	albert	bart	bert	distilbert	minilm	roberta	**Classifier Mean**
RandomForest	Precision	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00
	Recall	0.77 ± 0.01	0.77 ± 0.01	0.77 ± 0.01	0.77 ± 0.01	0.77 ± 0.01	0.77 ± 0.01	0.77 ± 0.01
	F1	0.87 ± 0.00	0.87 ± 0.00	0.87 ± 0.00	0.87 ± 0.00	0.87 ± 0.00	0.87 ± 0.00	0.87 ± 0.00
LogisticRegression	Precision	0.27 ± 0.00	0.27 ± 0.00	0.27 ± 0.00	0.27 ± 0.00	0.27 ± 0.00	0.27 ± 0.00	0.27 ± 0.00
	Recall	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00
	F1	0.42 ± 0.00	0.42 ± 0.00	0.42 ± 0.00	0.42 ± 0.00	0.42 ± 0.00	0.42 ± 0.00	0.42 ± 0.00
KNN	Precision	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01
	Recall	0.59 ± 0.01	0.59 ± 0.01	0.59 ± 0.01	0.59 ± 0.01	0.59 ± 0.01	0.59 ± 0.01	0.59 ± 0.01
	F1	0.72 ± 0.01	0.72 ± 0.01	0.72 ± 0.01	0.72 ± 0.01	0.72 ± 0.01	0.72 ± 0.01	0.72 ± 0.01
MLP	Precision	0.88 ± 0.02	0.88 ± 0.02	0.88 ± 0.02	0.88 ± 0.02	0.87 ± 0.01	0.88 ± 0.02	0.88 ± 0.02
	Recall	0.85 ± 0.02	0.85 ± 0.02	0.85 ± 0.02	0.85 ± 0.02	0.85 ± 0.02	0.85 ± 0.02	0.85 ± 0.02
	F1	0.86 ± 0.01	0.86 ± 0.01	0.86 ± 0.01	0.86 ± 0.01	0.86 ± 0.00	0.86 ± 0.01	0.86 ± 0.01
CatBoost	Precision	0.83 ± 0.02	0.83 ± 0.02	0.83 ± 0.02	0.83 ± 0.02	0.83 ± 0.02	0.83 ± 0.02	0.83 ± 0.02
	Recall	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00
	F1	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01
XGBoost	Precision	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	$\mathbf{0.93 \pm 0.01}$
	Recall	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	$\mathbf{0.97 \pm 0.00}$
	F1	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	$\mathbf{0.95 \pm 0.00}$
Embedding Mean	Precision	0.81 ± 0.01	0.81 ± 0.01	0.81 ± 0.01	0.81 ± 0.01	0.81 ± 0.01	0.81 ± 0.01	–
	Recall	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01	–
	F1	0.79 ± 0.01	0.79 ± 0.01	0.79 ± 0.01	0.79 ± 0.01	0.79 ± 0.01	0.79 ± 0.01	–

Table 4.15: Precision, Recall, and F1-score (mean± std) per classifier and embedding model for scenario (∅, I1) ↔ (∅, I2).

Classifier	**Metric**	albert	bart	bert	distilbert	minilm	roberta	**Classifier Mean**
RandomForest	Precision	0.89 ± 0.01	0.89 ± 0.01	0.89 ± 0.01	0.89 ± 0.01	0.89 ± 0.01	0.89 ± 0.01	0.89 ± 0.01
	Recall	0.29 ± 0.01	0.29 ± 0.01	0.29 ± 0.01	0.29 ± 0.01	0.29 ± 0.01	0.29 ± 0.01	0.29 ± 0.01
	F1	0.44 ± 0.00	0.44 ± 0.00	0.44 ± 0.00	0.44 ± 0.00	0.44 ± 0.00	0.44 ± 0.00	0.44 ± 0.00
LogisticRegression	Precision	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00
	Recall	0.83 ± 0.01	0.83 ± 0.01	0.83 ± 0.01	0.83 ± 0.01	0.83 ± 0.01	0.83 ± 0.01	0.83 ± 0.01
	F1	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00
KNN	Precision	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01
	Recall	0.33 ± 0.01	0.33 ± 0.01	0.33 ± 0.01	0.33 ± 0.01	0.33 ± 0.01	0.33 ± 0.01	0.33 ± 0.01
	F1	0.48 ± 0.01	0.48 ± 0.01	0.48 ± 0.01	0.48 ± 0.01	0.48 ± 0.01	0.48 ± 0.01	0.48 ± 0.01
MLP	Precision	0.64 ± 0.04	0.64 ± 0.04	0.64 ± 0.04	0.64 ± 0.04	0.65 ± 0.05	0.64 ± 0.04	0.64 ± 0.04
	Recall	0.47 ± 0.02	0.47 ± 0.02	0.47 ± 0.02	0.47 ± 0.02	0.47 ± 0.02	0.47 ± 0.02	0.47 ± 0.02
	F1	0.54 ± 0.01	0.54 ± 0.01	0.54 ± 0.01	0.54 ± 0.01	0.54 ± 0.01	0.54 ± 0.01	0.54 ± 0.01
CatBoost	Precision	0.32 ± 0.01	0.32 ± 0.01	0.32 ± 0.01	0.32 ± 0.01	0.32 ± 0.01	0.32 ± 0.01	0.32 ± 0.01
	Recall	0.73 ± 0.01	0.73 ± 0.01	0.73 ± 0.01	0.73 ± 0.01	0.73 ± 0.01	0.73 ± 0.01	0.73 ± 0.01
	F1	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01
XGBoost	Precision	0.53 ± 0.01	0.53 ± 0.01	0.53 ± 0.01	0.53 ± 0.01	0.53 ± 0.01	0.53 ± 0.01	$\mathbf{0.53 \pm 0.01}$
	Recall	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	$\mathbf{0.61 \pm 0.01}$
	F1	0.57 ± 0.01	0.57 ± 0.01	0.57 ± 0.01	0.57 ± 0.01	0.57 ± 0.01	0.57 ± 0.01	$\mathbf{0.57 \pm 0.01}$
Embedding Mean	Precision	0.56 ± 0.01	0.56 ± 0.01	0.56 ± 0.01	0.56 ± 0.01	$\mathbf{0.56 \pm 0.01}$	0.56 ± 0.01	–
	Recall	0.54 ± 0.01	0.54 ± 0.01	0.54 ± 0.01	0.54 ± 0.01	$\mathbf{0.54 \pm 0.01}$	0.54 ± 0.01	–
	F1	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01	$\mathbf{0.45 \pm 0.01}$	0.44 ± 0.01	–

Table 4.16: Precision, Recall, and F1-score (mean± std) per classifier and embedding model for scenario (S1, I1) ↔ (S2,∅).

Classifier	**Metric**	albert	bart	bert	distilbert	minilm	roberta	**Classifier Mean**
RandomForest	Precision	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00
	Recall	0.80 ± 0.01	0.80 ± 0.01	0.80 ± 0.01	0.80 ± 0.01	0.80 ± 0.01	0.80 ± 0.01	0.80 ± 0.01
	F1	0.88 ± 0.00	0.88 ± 0.00	0.88 ± 0.00	0.88 ± 0.00	0.88 ± 0.00	0.88 ± 0.00	0.88 ± 0.00
LogisticRegression	Precision	0.28 ± 0.00	0.28 ± 0.00	0.28 ± 0.00	0.28 ± 0.00	0.28 ± 0.00	0.28 ± 0.00	0.28 ± 0.00
	Recall	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01
	F1	0.43 ± 0.00	0.43 ± 0.00	0.43 ± 0.00	0.43 ± 0.00	0.43 ± 0.00	0.43 ± 0.00	0.43 ± 0.00
KNN	Precision	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01
	Recall	0.59 ± 0.01	0.59 ± 0.01	0.59 ± 0.01	0.59 ± 0.01	0.59 ± 0.01	0.59 ± 0.01	0.59 ± 0.01
	F1	0.73 ± 0.01	0.73 ± 0.01	0.73 ± 0.01	0.73 ± 0.01	0.73 ± 0.01	0.73 ± 0.01	0.73 ± 0.01
MLP	Precision	0.87 ± 0.02	0.87 ± 0.02	0.87 ± 0.02	0.87 ± 0.02	0.87 ± 0.02	0.87 ± 0.02	0.87 ± 0.02
	Recall	0.86 ± 0.02	0.86 ± 0.02	0.86 ± 0.02	0.86 ± 0.02	0.86 ± 0.02	0.86 ± 0.02	0.86 ± 0.02
	F1	0.86 ± 0.01	0.86 ± 0.01	0.86 ± 0.01	0.86 ± 0.01	0.86 ± 0.01	0.86 ± 0.01	0.86 ± 0.01
CatBoost	Precision	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01	0.84 ± 0.01	0.85 ± 0.01	0.85 ± 0.01
	Recall	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00
	F1	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01
XGBoost	Precision	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	$\mathbf{0.93 \pm 0.01}$
	Recall	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.00	0.97 ± 0.01	$\mathbf{0.97 \pm 0.01}$
	F1	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	$\mathbf{0.95 \pm 0.00}$
Embedding Mean	Precision	0.81 ± 0.01	0.81 ± 0.01	0.81 ± 0.01	0.81 ± 0.01	0.81 ± 0.01	0.81 ± 0.01	–
	Recall	0.86 ± 0.01	0.86 ± 0.01	0.86 ± 0.01	0.86 ± 0.01	0.86 ± 0.01	0.86 ± 0.01	–
	F1	0.79 ± 0.00	0.79 ± 0.00	0.79 ± 0.00	0.79 ± 0.00	0.79 ± 0.01	0.79 ± 0.00	–

Table 4.17: Precision, Recall, and F1-score (mean ± std) per classifier and embedding model for scenario (S1, I1) ↔ (∅, I2).

Classifier	**Metric**	albert	bart	bert	distilbert	minilm	roberta	**Classifier Mean**
RandomForest	Precision	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.00	0.96 ± 0.01	0.95 ± 0.01	0.95 ± 0.01
	Recall	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01
	F1	0.60 ± 0.01	0.60 ± 0.01	0.60 ± 0.01	0.60 ± 0.01	0.60 ± 0.01	0.60 ± 0.01	0.60 ± 0.01
LogisticRegression	Precision	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00
	Recall	0.83 ± 0.01	0.83 ± 0.01	0.83 ± 0.01	0.83 ± 0.01	0.83 ± 0.01	0.83 ± 0.01	0.83 ± 0.01
	F1	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00
KNN	Precision	0.88 ± 0.00	0.88 ± 0.00	0.88 ± 0.00	0.87 ± 0.01	0.88 ± 0.00	0.88 ± 0.00	0.88 ± 0.01
	Recall	0.47 ± 0.01	0.47 ± 0.01	0.47 ± 0.01	0.48 ± 0.01	0.47 ± 0.01	0.47 ± 0.01	0.47 ± 0.01
	F1	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.62 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01
MLP	Precision	0.69 ± 0.04	0.69 ± 0.04	0.69 ± 0.04	0.71 ± 0.03	0.67 ± 0.04	0.69 ± 0.04	0.69 ± 0.04
	Recall	0.56 ± 0.03	0.56 ± 0.03	0.56 ± 0.03	0.55 ± 0.02	0.57 ± 0.02	0.56 ± 0.03	0.56 ± 0.03
	F1	0.62 ± 0.01	0.62 ± 0.01	0.62 ± 0.01	0.62 ± 0.01	0.61 ± 0.01	0.62 ± 0.01	0.62 ± 0.01
CatBoost	Precision	0.41 ± 0.01	0.41 ± 0.01	0.41 ± 0.01	0.41 ± 0.01	0.41 ± 0.01	0.41 ± 0.01	0.41 ± 0.01
	Recall	0.77 ± 0.01	0.77 ± 0.01	0.77 ± 0.01	0.77 ± 0.01	0.77 ± 0.01	0.77 ± 0.01	0.77 ± 0.01
	F1	0.54 ± 0.01	0.54 ± 0.01	0.54 ± 0.01	0.54 ± 0.00	0.54 ± 0.01	0.54 ± 0.01	0.54 ± 0.01
XGBoost	Precision	0.63 ± 0.02	0.63 ± 0.02	0.63 ± 0.02	0.63 ± 0.01	0.63 ± 0.02	0.63 ± 0.02	$\mathbf{0.63 \pm 0.02}$
	Recall	0.67 ± 0.01	0.67 ± 0.01	0.67 ± 0.01	0.68 ± 0.01	0.67 ± 0.01	0.67 ± 0.01	$\mathbf{0.67 \pm 0.01}$
	F1	0.65 ± 0.01	0.65 ± 0.01	0.65 ± 0.01	0.65 ± 0.01	0.65 ± 0.01	0.65 ± 0.01	$\mathbf{0.65 \pm 0.01}$
Embedding Mean	Precision	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	$\mathbf{0.61 \pm 0.01}$	0.61 ± 0.01	0.61 ± 0.01	–
	Recall	0.62 ± 0.01	0.62 ± 0.01	0.62 ± 0.01	$\mathbf{0.63 \pm 0.01}$	0.62 ± 0.01	0.62 ± 0.01	–
	F1	0.53 ± 0.01	0.53 ± 0.01	0.53 ± 0.01	$\mathbf{0.54 \pm 0.01}$	0.53 ± 0.01	0.53 ± 0.01	–

Table 4.18: Precision, Recall, and F1-score (mean ± std) per classifier and embedding model for scenario (S1, ∅) ↔ (∅, I2).

Classifier	**Metric**	albert	bart	bert	distilbert	minilm	roberta	**Classifier Mean**
RandomForest	Precision	0.98 ± 0.01	0.99 ± 0.01	0.99 ± 0.01	0.98 ± 0.01	0.98 ± 0.01	0.98 ± 0.01	0.98 ± 0.01
	Recall	0.28 ± 0.03	0.30 ± 0.03	0.31 ± 0.04	0.33 ± 0.03	0.20 ± 0.02	0.32 ± 0.03	0.29 ± 0.03
	F1	0.44 ± 0.04	0.46 ± 0.04	0.46 ± 0.04	0.49 ± 0.04	0.33 ± 0.03	0.48 ± 0.03	0.44 ± 0.04
LogisticRegression	Precision	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00
	Recall	0.66 ± 0.03	0.66 ± 0.03	0.65 ± 0.03	0.65 ± 0.03	0.62 ± 0.02	0.66 ± 0.02	0.65 ± 0.03
	F1	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.00
KNN	Precision	0.85 ± 0.02	0.87 ± 0.02	0.87 ± 0.02	0.85 ± 0.02	0.80 ± 0.02	0.86 ± 0.01	0.85 ± 0.02
	Recall	0.32 ± 0.03	0.38 ± 0.03	0.36 ± 0.04	0.37 ± 0.03	0.29 ± 0.02	0.35 ± 0.03	0.35 ± 0.03
	F1	0.47 ± 0.04	0.53 ± 0.03	0.51 ± 0.04	0.51 ± 0.03	0.43 ± 0.03	0.50 ± 0.03	0.49 ± 0.03
MLP	Precision	0.62 ± 0.05	0.66 ± 0.06	0.66 ± 0.06	0.66 ± 0.05	0.59 ± 0.05	0.65 ± 0.06	0.64 ± 0.06
	Recall	0.35 ± 0.03	0.40 ± 0.04	0.37 ± 0.05	0.40 ± 0.04	0.28 ± 0.03	0.39 ± 0.04	0.37 ± 0.04
	F1	0.45 ± 0.03	0.50 ± 0.03	0.48 ± 0.04	0.49 ± 0.03	0.38 ± 0.03	0.49 ± 0.04	0.46 ± 0.03
CatBoost	Precision	0.35 ± 0.02	0.38 ± 0.02	0.38 ± 0.02	0.40 ± 0.02	0.25 ± 0.01	0.40 ± 0.02	0.36 ± 0.02
	Recall	0.75 ± 0.03	0.77 ± 0.03	0.80 ± 0.03	0.79 ± 0.02	0.64 ± 0.03	0.81 ± 0.03	0.76 ± 0.03
	F1	0.48 ± 0.02	0.51 ± 0.02	0.51 ± 0.02	0.53 ± 0.02	0.36 ± 0.01	0.54 ± 0.02	0.49 ± 0.02
XGBoost	Precision	0.67 ± 0.02	0.67 ± 0.03	0.68 ± 0.03	0.71 ± 0.02	0.61 ± 0.02	0.68 ± 0.03	$\mathbf{0.67 \pm 0.03}$
	Recall	0.53 ± 0.04	0.54 ± 0.04	0.56 ± 0.04	0.56 ± 0.03	0.41 ± 0.03	0.59 ± 0.03	$\mathbf{0.53 \pm 0.03}$
	F1	0.59 ± 0.03	0.60 ± 0.03	0.61 ± 0.03	0.62 ± 0.03	0.49 ± 0.03	0.64 ± 0.03	$\mathbf{0.59 \pm 0.03}$
Embedding Mean	Precision	0.59 ± 0.02	0.60 ± 0.02	0.60 ± 0.02	$\mathbf{0.61 \pm 0.02}$	0.55 ± 0.02	$\mathbf{0.61 \pm 0.02}$	–
	Recall	0.48 ± 0.03	0.51 ± 0.03	0.51 ± 0.04	$\mathbf{0.51 \pm 0.03}$	0.41 ± 0.03	$\mathbf{0.52 \pm 0.03}$	–
	F1	0.42 ± 0.03	0.45 ± 0.03	0.44 ± 0.03	$\mathbf{0.46 \pm 0.02}$	0.35 ± 0.02	$\mathbf{0.46 \pm 0.02}$	–

Table 4.19: Precision, Recall, and F1-score (mean ± std) with XGBoost across categories for scenario (S1, I1) ↔ (S2, I2).

Category (#Pairs)	**Metric**	ALBERT	BART	BERT	DistilBERT	MiniLM	RoBERTa
ChEMBL (180)	Precision	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.94 ± 0.01	0.95 ± 0.01
	Recall	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01
	F1	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.00	0.95 ± 0.01
Magellan (7)	Precision	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	Recall	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	F1	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
OpenData (180)	Precision	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01
	Recall	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00
	F1	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00
TPC-DI (180)	Precision	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01
	Recall	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01
	F1	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00
Wikidata (4)	Precision	0.66 ± 0.14	0.66 ± 0.14	0.66 ± 0.14	0.66 ± 0.14	0.67 ± 0.13	0.66 ± 0.14
	Recall	0.74 ± 0.15	0.74 ± 0.15	0.74 ± 0.15	0.74 ± 0.15	0.74 ± 0.15	0.74 ± 0.15
	F1	0.69 ± 0.10	0.69 ± 0.10	0.69 ± 0.10	0.69 ± 0.10	0.69 ± 0.10	0.69 ± 0.10

Table 4.20: Precision, Recall, and F1-score (mean ± std) with XGBoost across categories for scenario (S1, ∅) ↔ (S2, ∅).

Category (#Pairs)	**Metric**	ALBERT	BART	BERT	DistilBERT	MiniLM	RoBERTa
ChEMBL (180)	Precision	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01
	Recall	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01
	F1	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00
Magellan (7)	Precision	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	Recall	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	F1	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
OpenData (180)	Precision	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01
	Recall	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00
	F1	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01
TPC-DI (180)	Precision	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01
	Recall	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01
	F1	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00
Wikidata (4)	Precision	0.80 ± 0.19	0.80 ± 0.19	0.80 ± 0.19	0.80 ± 0.19	0.80 ± 0.19	0.80 ± 0.19
	Recall	0.65 ± 0.08	0.65 ± 0.08	0.65 ± 0.08	0.65 ± 0.08	0.65 ± 0.08	0.65 ± 0.08
	F1	0.71 ± 0.09	0.71 ± 0.09	0.71 ± 0.09	0.71 ± 0.09	0.71 ± 0.09	0.71 ± 0.09

Table 4.21: Precision, Recall, and F1-score (mean ± std) with XGBoost across categories for scenario (∅, I1) ↔ (∅, I2).

Category (#Pairs)	**Metric**	ALBERT	BART	BERT	DistilBERT	MiniLM	RoBERTa
ChEMBL (180)	Precision	0.63 ± 0.02	0.63 ± 0.02	0.63 ± 0.02	0.63 ± 0.02	0.63 ± 0.02	0.63 ± 0.02
	Recall	0.68 ± 0.01	0.68 ± 0.01	0.68 ± 0.01	0.68 ± 0.01	0.68 ± 0.01	0.68 ± 0.01
	F1	0.66 ± 0.01	0.66 ± 0.01	0.66 ± 0.01	0.66 ± 0.01	0.66 ± 0.01	0.66 ± 0.01
Magellan (7)	Precision	0.94 ± 0.07	0.94 ± 0.07	0.94 ± 0.07	0.94 ± 0.07	0.94 ± 0.07	0.94 ± 0.07
	Recall	0.95 ± 0.04	0.95 ± 0.04	0.95 ± 0.04	0.95 ± 0.04	0.95 ± 0.04	0.95 ± 0.04
	F1	0.95 ± 0.05	0.95 ± 0.05	0.95 ± 0.05	0.95 ± 0.05	0.95 ± 0.05	0.95 ± 0.05
OpenData (180)	Precision	0.46 ± 0.02	0.46 ± 0.02	0.46 ± 0.02	0.46 ± 0.02	0.46 ± 0.02	0.46 ± 0.02
	Recall	0.48 ± 0.01	0.48 ± 0.01	0.48 ± 0.01	0.48 ± 0.01	0.48 ± 0.01	0.48 ± 0.01
	F1	0.47 ± 0.01	0.47 ± 0.01	0.47 ± 0.01	0.47 ± 0.01	0.47 ± 0.01	0.47 ± 0.01
TPC-DI (180)	Precision	0.57 ± 0.01	0.57 ± 0.01	0.57 ± 0.01	0.57 ± 0.01	0.57 ± 0.01	0.57 ± 0.01
	Recall	0.82 ± 0.02	0.82 ± 0.02	0.82 ± 0.02	0.82 ± 0.02	0.82 ± 0.02	0.82 ± 0.02
	F1	0.67 ± 0.01	0.67 ± 0.01	0.67 ± 0.01	0.67 ± 0.01	0.67 ± 0.01	0.67 ± 0.01
Wikidata (4)	Precision	0.50 ± 0.14	0.50 ± 0.14	0.50 ± 0.14	0.50 ± 0.14	0.50 ± 0.14	0.50 ± 0.14
	Recall	0.72 ± 0.18	0.72 ± 0.18	0.72 ± 0.18	0.72 ± 0.18	0.72 ± 0.18	0.72 ± 0.18
	F1	0.59 ± 0.14	0.59 ± 0.14	0.59 ± 0.14	0.59 ± 0.14	0.59 ± 0.14	0.59 ± 0.14

Table 4.22: Precision, Recall, and F1-score (mean ± std) with XGBoost across categories for scenario (S1, I1) ↔ (S2, ∅).

Category (#Pairs)	**Metric**	ALBERT	BART	BERT	DistilBERT	MiniLM	RoBERTa
ChEMBL (180)	Precision	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01
	Recall	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01
	F1	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00
Magellan (7)	Precision	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	Recall	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	F1	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
OpenData (180)	Precision	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01
	Recall	0.98 ± 0.01	0.98 ± 0.01	0.98 ± 0.01	0.98 ± 0.01	0.98 ± 0.01	0.98 ± 0.01
	F1	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01
TPC-DI (180)	Precision	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01
	Recall	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01
	F1	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.00	0.96 ± 0.01
Wikidata (4)	Precision	0.80 ± 0.13	0.80 ± 0.13	0.80 ± 0.13	0.80 ± 0.13	0.82 ± 0.13	0.80 ± 0.13
	Recall	0.68 ± 0.13	0.68 ± 0.13	0.68 ± 0.13	0.68 ± 0.13	0.70 ± 0.12	0.68 ± 0.13
	F1	0.72 ± 0.09	0.72 ± 0.09	0.72 ± 0.09	0.72 ± 0.09	0.74 ± 0.08	0.72 ± 0.09

Table 4.23: Precision, Recall, and F1-score (mean ± std) with XGBoost across categories for scenario (S1, I1) ↔ (∅, I2).

Category (#Pairs)	**Metric**	ALBERT	BART	BERT	DistilBERT	MiniLM	RoBERTa
ChEMBL (180)	Precision	0.68 ± 0.02	0.68 ± 0.02	0.68 ± 0.02	0.70 ± 0.03	0.68 ± 0.02	0.68 ± 0.02
	Recall	0.79 ± 0.01	0.79 ± 0.01	0.79 ± 0.01	0.80 ± 0.01	0.79 ± 0.01	0.79 ± 0.01
	F1	0.73 ± 0.01	0.73 ± 0.01	0.73 ± 0.01	0.74 ± 0.01	0.73 ± 0.01	0.73 ± 0.01
Magellan (7)	Precision	0.97 ± 0.05	0.97 ± 0.05	0.97 ± 0.05	0.96 ± 0.05	0.97 ± 0.05	0.97 ± 0.05
	Recall	0.81 ± 0.11	0.81 ± 0.11	0.81 ± 0.11	0.85 ± 0.14	0.79 ± 0.12	0.81 ± 0.11
	F1	0.88 ± 0.07	0.88 ± 0.07	0.88 ± 0.07	0.89 ± 0.09	0.87 ± 0.08	0.88 ± 0.07
OpenData (180)	Precision	0.54 ± 0.03	0.54 ± 0.03	0.54 ± 0.03	0.54 ± 0.02	0.54 ± 0.03	0.54 ± 0.03
	Recall	0.51 ± 0.01	0.51 ± 0.01	0.51 ± 0.01	0.52 ± 0.01	0.51 ± 0.01	0.51 ± 0.01
	F1	0.53 ± 0.02	0.53 ± 0.02	0.53 ± 0.02	0.53 ± 0.01	0.53 ± 0.01	0.53 ± 0.02
TPC-DI (180)	Precision	0.72 ± 0.02	0.72 ± 0.02	0.72 ± 0.02	0.72 ± 0.02	0.72 ± 0.02	0.72 ± 0.02
	Recall	0.90 ± 0.02	0.90 ± 0.02	0.90 ± 0.02	0.90 ± 0.01	0.90 ± 0.02	0.90 ± 0.02
	F1	0.80 ± 0.01	0.80 ± 0.01	0.80 ± 0.01	0.80 ± 0.01	0.80 ± 0.01	0.80 ± 0.01
Wikidata (4)	Precision	0.43 ± 0.12	0.43 ± 0.12	0.43 ± 0.12	0.44 ± 0.09	0.42 ± 0.13	0.43 ± 0.12
	Recall	0.64 ± 0.12	0.64 ± 0.12	0.64 ± 0.12	0.62 ± 0.20	0.65 ± 0.12	0.64 ± 0.12
	F1	0.50 ± 0.10	0.50 ± 0.10	0.50 ± 0.10	0.50 ± 0.11	0.50 ± 0.11	0.50 ± 0.10

Table 4.24: Precision, Recall, and F1-score (mean ± std) with XGBoost across categories for scenario (S1, ∅) ↔ (∅, I2).

Category (#Pairs)	Metric	ALBERT	BART	BERT	DistilBERT	MiniLM	RoBERTa
ChEMBL (180)	Precision	0.72 ± 0.03	0.80 ± 0.02	0.79 ± 0.03	0.84 ± 0.03	0.71 ± 0.04	0.78 ± 0.02
	Recall	0.77 ± 0.04	0.78 ± 0.04	0.78 ± 0.06	0.79 ± 0.04	0.58 ± 0.05	0.83 ± 0.04
	F1	0.75 ± 0.03	0.79 ± 0.03	0.78 ± 0.04	0.81 ± 0.03	0.63 ± 0.05	0.80 ± 0.03
Magellan (7)	Precision	1.00 ± 0.02	1.00 ± 0.01	0.99 ± 0.04	1.00 ± 0.01	1.00 ± 0.02	0.99 ± 0.03
	Recall	0.82 ± 0.10	0.77 ± 0.13	0.80 ± 0.14	0.84 ± 0.13	0.76 ± 0.12	0.85 ± 0.14
	F1	0.90 ± 0.06	0.86 ± 0.09	0.88 ± 0.10	0.91 ± 0.08	0.86 ± 0.07	0.91 ± 0.09
OpenData (180)	Precision	0.49 ± 0.03	0.48 ± 0.03	0.50 ± 0.02	0.53 ± 0.03	0.48 ± 0.03	0.52 ± 0.03
	Recall	0.25 ± 0.02	0.29 ± 0.02	0.32 ± 0.02	0.31 ± 0.02	0.21 ± 0.01	0.36 ± 0.03
	F1	0.33 ± 0.02	0.36 ± 0.02	0.39 ± 0.02	0.39 ± 0.02	0.30 ± 0.02	0.42 ± 0.03
TPC-DI (180)	Precision	0.82 ± 0.02	0.80 ± 0.03	0.82 ± 0.02	0.82 ± 0.02	0.67 ± 0.03	0.84 ± 0.02
	Recall	0.91 ± 0.03	0.91 ± 0.03	0.91 ± 0.02	0.91 ± 0.03	0.69 ± 0.04	0.90 ± 0.03
	F1	0.86 ± 0.02	0.85 ± 0.02	0.86 ± 0.02	0.86 ± 0.02	0.68 ± 0.03	0.87 ± 0.02
Wikidata (4)	Precision	0.55 ± 0.27	0.47 ± 0.19	0.63 ± 0.29	0.47 ± 0.23	0.31 ± 0.13	0.52 ± 0.28
	Recall	0.36 ± 0.17	0.28 ± 0.15	0.26 ± 0.17	0.28 ± 0.16	0.27 ± 0.12	0.39 ± 0.16
	F1	0.41 ± 0.19	0.33 ± 0.14	0.33 ± 0.19	0.32 ± 0.16	0.28 ± 0.12	0.42 ± 0.18

In compliant scenarios (C1, C2, and C3), the framework achieves near-perfect results. In C1 (Table 4.13), where schema and instance information are available, XGBoost achieves an F1 = 0.97 ± 0.00 with balanced precision and recall. CatBoost also performs well with an F1 = 0.93. MLP and RandomForest follow with F1 = 0.90, while KNN performs less with F1 = 0.84. Logistic regression is weak, with F1 = 0.46. In C2 (Table 4.14), where only schemas are present, performance remains high. XGBoost again leads with F1 = 0.95 ± 0.00, CatBoost follows with F1 = 0.90, and MLP and RandomForest remain stable around F1 = 0.86–0.87. KNN drops to F1 = 0.72, and Logistic Regression falls further to F1 = 0.42.

In C3 (Table 4.15), where only instances are used, performance falls sharply. XGBoost still leads but with a lower F1 = 0.57 ± 0.01. MLP achieves F1 = 0.54, while KNN, RandomForest, and CatBoost remain between F1 = 0.44 and 0.48. Logistic Regression is very low at F1 = 0.20.

In the disparate scenarios (D4–D6), the absence of either schema or instance elements makes the task harder. In D4 (Table 4.16), XGBoost keeps a strong performance with F1 = 0.95 ± 0.00, followed by CatBoost (0.91), RandomForest (0.88), and MLP (0.86). KNN and Logistic Regression are weaker with F1 = 0.73 and 0.43.

In D5 (Table 4.17), where schemas are present only on one side, results decrease. XGBoost gives F1 = 0.65 ± 0.01, while MLP, KNN, and RandomForest are close between 0.60 and 0.62. CatBoost drops to F1 = 0.54, and Logistic Regression is almost unusable with F1 = 0.19.

In D6 (Table 4.18), where schemas are removed from one side and only instances remain, this becomes the most difficult setting. XGBoost stays ahead with F1 = 0.59 ± 0.03, while KNN and CatBoost follow with F1 around 0.49. MLP and RandomForest are slightly lower, between F1 = 0.44 and 0.46. Logistic Regression collapses with F1 = 0.10.

We examine the results in depth by dataset category to see how classifiers and integrations adapt to different vocabularies. Tables 4.19–4.24 report results across the five categories.

For Magellan, which contains only seven pairs, results are nearly perfect in all scenarios, with F1 near 1.00 ± 0.00 across all scenarios. The small and controlled vocabulary makes alignment straightforward, and both schema and instance information are consistent.

ChEMBL also shows very strong results. In complete and schema-only scenarios (Table 4.19, Table 4.20), F1 stays around 0.95 ± 0.00, while even in instance-only settings (Table 4.21) the framework reaches F1 = 0.66 ± 0.01. This confirms that the biomedical vocabulary, although

technical, is well captured by embeddings.

TPC-DI follows the same trend. F1 remains very high when schemas are present (around 0.96–0.97), and even in mixed settings (e.g., schema to instance in Table 4.24) the classifier still achieves F1 ≈ 0.86. This reflects the regularity of customer and prospect data, which provides stable vocabularies for embeddings.

Open Data is more difficult. While schema-based scenarios reach F1 above 0.94, instance-only settings are much weaker with F1 $= 0.47 \pm 0.01$ (Table 4.21) and schema-to-instance settings drop further to F1 as low as 0.33 ± 0.02 (Table 4.24). This shows the impact of heterogeneous vocabularies across administrative sources, where unseen tokens in the test data reduce both precision and recall.

Wikidata is the most challenging category. In the best case with schema only, F1 is 0.71 ± 0.09 (Table 4.20), but in instance-only C3 and D3 scenarios, F1 drops to 0.59 ± 0.14 and even 0.41 ± 0.19 (Table 4.21, Table 4.24). The high variance across runs confirms the instability of small-rich vocabularies, where many tokens appear only once.

Discussion. When schema elements are available on both sides, performance is very high, since classifiers can rely on direct vocabulary overlap. Once schemas are missing, as in C3, D5, and D6, performance drops strongly, showing how much the models depend on vocabulary to guide alignment. This confirms that instance-only matching is much more challenging, because vocabularies across training and test data often differ. Even with shuffling, many tokens seen during training are not repeated in test, which explains the lower scores.

Regarding classifiers, boosting methods (XGBoost and CatBoost) are the most robust, since they can balance precision and recall and capture non-linear interactions. XGBoost is clearly the best overall, as it remains strong even when only partial information is available. CatBoost performs well when schemas are present, but loses more in instance-only cases. RandomForest and MLP are stable but weaker, especially when the vocabulary mismatch is high. KNN depends heavily on similarity in the input space, so it struggles when vocabularies change between training and test. Logistic Regression fails in almost all settings: although it keeps a high recall, its very low precision leads to poor F1.

Since the embeddings originate from pre-trained language models, one might question whether the *Valentine* datasets could have been indirectly included in their pre-training corpora. To prevent such overlap, we deliberately restricted our experiments to earlier transformer models—spanning from 2018 (BERT) to 2020 (MiniLM, ALBERT)—which all precede the public release of the Valentine benchmark (2021). This temporal separation ensures that none of the evaluated datasets were seen during pre-training, eliminating potential data leakage and reinforcing the fairness of our evaluation.

Boosting methods are the most resilient in difficult settings, whereas simple linear models are less suited. Across the easier collections (Magellan, ChEMBL, TPC-DI), different embeddings perform similarly because vocabularies are regular; in the harder ones (Open Data, Wikidata), heavier embeddings (BERT, RoBERTa) sometimes yield slightly higher recall while lighter models (DistilBERT, MiniLM) lose ground. Overall, the classifier matters more than the embedding model: gradient-boosted trees (XGBoost, CatBoost) learn and exploit interactions among classical, spectral, and topological meta-features far better than linear models such as linear regression.

In summary, when vocabularies are consistent, the framework reaches near-perfect alignment regardless of embedding. When vocabularies are diverse or inconsistent, performance drops sharply and variance increases, showing that the main limitation is vocabulary mismatch rather than classifier capacity.

RQ2. Efficiency

The objective of this research question is to quantify end-to-end computational cost by breaking down runtime per step (embedding, meta-feature computation, classification) across scenarios, comparing embedding models and classifiers, and identifying effectiveness–efficiency trade-offs to select the best embedding model and classifier.

Table 4.25: Embedding time per model and category for scenario (S1, I1) ↔ (S2, I2). Each cell shows $(\mathcal{S}_1^{\text{gen}}, \mathcal{I}_1^{\text{gen}})|(\mathcal{S}_2^{\text{gen}}, \mathcal{I}_2^{\text{gen}})$ time (seconds).

	ChEMBL (180)	Magellan (7)	OpenData (180)	TPC-DI (180)	Wikidata
ALBERT	8.29 / 8.46	2.73 / 4.35	19.21 / 19.47	7.83 / 5.57	7.30 / 7.25
BART	9.15 / 9.48	2.99 / 4.56	20.40 / 20.60	7.77 / 5.43	7.29 / 7.29
BERT	8.29 / 8.57	2.49 / 3.95	18.68 / 18.61	7.47 / 5.26	6.63 / 6.81
DistilBERT	4.02 / 4.06	1.25 / 2.35	8.98 / 8.92	3.50 / 2.47	3.15 / 3.17
MiniLM	2.23 / 2.24	0.53 / 1.88	4.58 / 4.50	1.77 / 1.19	1.33 / 1.40
RoBERTa	7.17 / 7.39	2.39 / 3.72	16.29 / 16.44	6.53 / 4.56	6.09 / 6.08

Table 4.26: Embedding time per model and category for scenario (S1, ∅) ↔ (S2, ∅). Each cell shows $(\mathcal{S}_1^{\text{gen}}, \mathcal{I}_1^{\text{gen}})|(\mathcal{S}_2^{\text{gen}}, \mathcal{I}_2^{\text{gen}})$ time (seconds).

	ChEMBL (180)	Magellan (7)	OpenData (180)	TPC-DI (180)	Wikidata
ALBERT	8.15 / 8.36	2.57 / 2.57	3.18 / 3.29	7.43 / 5.28	5.06 / 5.07
BART	8.82 / 9.21	2.84 / 2.84	3.40 / 3.63	8.42 / 5.73	5.48 / 5.50
BERT	8.00 / 8.27	2.40 / 2.41	3.10 / 3.17	7.51 / 5.23	4.64 / 4.67
DistilBERT	3.97 / 4.05	1.15 / 1.15	1.43 / 1.45	3.48 / 2.43	2.16 / 2.17
MiniLM	2.21 / 2.28	0.42 / 0.42	0.51 / 0.52	1.84 / 1.23	0.80 / 0.81
RoBERTa	7.09 / 7.35	2.28 / 2.28	2.67 / 2.85	6.51 / 4.51	4.34 / 4.35

Table 4.27: Embedding time per model and category for scenario (∅, I1) ↔ (∅, I2). Each cell shows $(\mathcal{S}_1^{\text{gen}}, \mathcal{I}_1^{\text{gen}})|(\mathcal{S}_2^{\text{gen}}, \mathcal{I}_2^{\text{gen}})$ time (seconds).

	ChEMBL (180)	Magellan (7)	OpenData (180)	TPC-DI (180)	Wikidata
ALBERT	9.33 / 8.98	2.72 / 4.41	18.82 / 19.08	7.89 / 8.13	7.10 / 7.18
BART	10.23 / 9.93	3.00 / 4.75	20.86 / 21.06	8.70 / 9.00	7.83 / 8.02
BERT	10.14 / 9.76	2.46 / 3.79	18.41 / 18.35	7.97 / 8.18	6.62 / 6.98
DistilBERT	4.66 / 4.16	1.24 / 2.34	8.95 / 8.88	4.28 / 4.35	3.35 / 3.39
MiniLM	3.27 / 2.68	0.53 / 1.90	4.62 / 4.55	2.42 / 2.43	1.61 / 1.76
RoBERTa	8.19 / 7.90	2.40 / 3.69	16.25 / 16.42	6.83 / 7.06	6.20 / 6.32

Execution time in our framework arises from three main components: embedding, meta-feature computation, and classification. All reported values are first summed at the dataset level and then averaged per category, ensuring comparability across families of very different sizes (Table 4.1).

Embedding runtimes strongly vary across models and scenarios (Tables 4.25–4.30). Lightweight encoders such as MiniLM and DistilBERT are consistently the fastest, with per-block times as low as 0.5–5 seconds depending on the dataset category. In contrast, full-size models like BERT, RoBERTa, and BART typically require around 7–20 seconds under the same conditions, making

Table 4.28: Embedding time per model and category for scenario (S1, I1) ↔ (S2, ∅). Each cell shows $(\mathcal{S}_1^{\text{gen}}, \mathcal{I}_1^{\text{gen}})|(\mathcal{S}_2^{\text{gen}}, \mathcal{I}_2^{\text{gen}})$ time (seconds).

	ChEMBL (180)	Magellan (7)	OpenData (180)	TPC-DI (180)	Wikidata
ALBERT	9.35 / 8.35	2.74 / 2.58	18.81 / 3.03	7.89 / 5.28	6.77 / 5.06
BART	11.19 / 9.38	3.00 / 2.86	21.55 / 3.31	8.51 / 5.67	7.66 / 5.44
BERT	10.30 / 8.42	2.46 / 2.36	18.71 / 2.86	8.23 / 5.25	6.59 / 4.76
DistilBERT	4.66 / 4.05	1.25 / 1.15	8.95 / 1.33	4.29 / 2.44	3.20 / 2.17
MiniLM	3.26 / 2.24	0.53 / 0.43	4.61 / 0.47	2.38 / 1.20	1.48 / 0.80
RoBERTa	8.20 / 7.33	2.39 / 2.28	16.30 / 2.61	6.85 / 4.49	6.09 / 4.36

Table 4.29: Embedding time per model and category for scenario (S1, I1) ↔ (∅, I2). Each cell shows $(\mathcal{S}_1^{\text{gen}}, \mathcal{I}_1^{\text{gen}})|(\mathcal{S}_2^{\text{gen}}, \mathcal{I}_2^{\text{gen}})$ time (seconds).

	ChEMBL (180)	Magellan (7)	OpenData (180)	TPC-DI (180)	Wikidata
ALBERT	9.32 / 8.97	2.69 / 4.36	18.77 / 19.03	7.86 / 8.10	7.05 / 7.14
BART	10.37 / 10.10	3.01 / 4.66	21.10 / 21.29	8.41 / 8.68	7.68 / 7.69
BERT	10.24 / 9.85	2.45 / 3.84	18.58 / 18.50	7.97 / 8.18	6.70 / 7.00
DistilBERT	4.65 / 4.15	1.24 / 2.36	8.96 / 8.90	4.28 / 4.35	3.35 / 3.40
MiniLM	3.33 / 2.75	0.54 / 1.93	4.61 / 4.53	2.41 / 2.44	1.66 / 1.81
RoBERTa	8.18 / 7.89	2.38 / 3.71	16.27 / 16.45	6.84 / 7.06	6.32 / 6.32

Table 4.30: Embedding time per model and category for scenario (S1, ∅) ↔ (∅, I2). Each cell shows $(\mathcal{S}_1^{\text{gen}}, \mathcal{I}_1^{\text{gen}})|(\mathcal{S}_2^{\text{gen}}, \mathcal{I}_2^{\text{gen}})$ time (seconds).

	ChEMBL (180)	Magellan (7)	OpenData (180)	TPC-DI (180)	Wikidata
ALBERT	7.89 / 9.03	2.59 / 4.35	3.25 / 19.06	7.46 / 8.14	6.75 / 7.17
BART	8.51 / 10.45	2.89 / 4.78	3.50 / 22.01	8.54 / 9.06	7.45 / 8.21
BERT	7.65 / 9.78	2.33 / 3.83	3.16 / 18.85	7.46 / 8.41	6.46 / 7.21
DistilBERT	3.86 / 4.15	1.15 / 2.41	1.45 / 8.90	3.49 / 4.35	2.94 / 3.40
MiniLM	2.15 / 2.72	0.43 / 1.95	0.52 / 4.53	1.87 / 2.45	1.17 / 1.80
RoBERTa	6.86 / 7.94	2.27 / 3.77	2.72 / 16.50	6.54 / 7.10	5.83 / 6.36

Table 4.31: Average meta-feature computation time (s) per embedding model and category, averaged across all datasets for scenario (S1, I1) ↔ (S2, I2).

	ChEMBL	Magellan	OpenData	TPC-DI	Wikidata
ALBERT	905.92 ± 393.80	105.22 ± 61.57	4252.48 ± 1858.30	843.58 ± 369.91	1083.81 ± 963.98
BART	913.54 ± 399.62	107.65 ± 63.82	4280.64 ± 1855.47	839.50 ± 367.58	1086.20 ± 960.00
BERT	912.36 ± 400.40	105.19 ± 62.44	4272.54 ± 1864.34	839.52 ± 367.35	1076.37 ± 958.85
DistilBERT	908.67 ± 397.01	103.51 ± 60.23	4264.11 ± 1867.86	840.53 ± 367.51	1083.67 ± 957.08
MiniLM	892.69 ± 393.47	102.97 ± 59.06	4207.99 ± 1837.58	818.70 ± 358.35	1046.20 ± 927.29
RoBERTa	905.15 ± 393.90	105.04 ± 62.59	4249.75 ± 1849.35	834.05 ± 364.44	1077.07 ± 962.87

Table 4.32: Average meta-feature computation time (s) per embedding model and category, averaged across all datasets for scenario (S1, ∅) ↔ (S2, ∅).

	ChEMBL	Magellan	OpenData	TPC-DI	Wikidata
ALBERT	913.87 ± 397.85	107.40 ± 64.75	4469.28 ± 1938.58	845.92 ± 376.02	727.09 ± 303.23
BART	907.78 ± 396.33	107.13 ± 64.37	4512.55 ± 1971.07	849.27 ± 375.78	728.04 ± 298.95
BERT	903.45 ± 394.92	106.52 ± 63.84	4545.07 ± 2024.31	842.55 ± 364.81	728.08 ± 310.63
DistilBERT	903.44 ± 393.69	106.62 ± 64.27	4515.79 ± 1974.00	839.55 ± 366.40	730.72 ± 306.24
MiniLM	896.83 ± 395.45	105.28 ± 61.40	4454.19 ± 1943.15	824.32 ± 361.99	722.61 ± 296.68
RoBERTa	914.58 ± 396.01	107.39 ± 64.19	4496.05 ± 1951.50	840.42 ± 368.49	723.76 ± 312.96

Table 4.33: Average meta-feature computation time (s) per embedding model and category, averaged across all datasets for scenario (∅, I1) ↔ (∅, I2).

	ChEMBL	Magellan	OpenData	TPC-DI	Wikidata
ALBERT	894.90 ± 390.18	114.93 ± 67.96	4831.26 ± 1915.94	806.45 ± 349.96	741.97 ± 411.07
BART	893.14 ± 387.86	114.99 ± 68.25	4843.15 ± 1923.17	807.13 ± 349.59	750.66 ± 420.72
BERT	889.13 ± 386.42	115.38 ± 69.40	4846.72 ± 1927.72	815.33 ± 353.73	758.91 ± 429.24
DistilBERT	893.88 ± 387.70	114.75 ± 68.00	4833.08 ± 1919.95	810.73 ± 351.58	752.35 ± 418.18
MiniLM	881.85 ± 386.74	113.37 ± 68.87	4797.09 ± 1910.43	794.97 ± 343.20	728.65 ± 412.84
RoBERTa	895.33 ± 387.06	115.83 ± 67.84	4847.18 ± 1921.46	811.03 ± 351.24	738.07 ± 407.12

Table 4.34: Average meta-feature computation time (s) per embedding model and category, averaged across all datasets for scenario (S1, I1) ↔ (S2, ∅).

	ChEMBL	Magellan	OpenData	TPC-DI	Wikidata
ALBERT	915.16 ± 392.65	107.10 ± 63.92	4447.85 ± 1947.59	822.78 ± 357.40	761.61 ± 353.04
BART	909.16 ± 395.14	105.74 ± 64.50	4420.25 ± 1929.10	828.96 ± 359.31	753.54 ± 319.17
BERT	906.45 ± 393.59	105.44 ± 68.03	4424.40 ± 1932.93	820.85 ± 355.83	740.26 ± 313.63
DistilBERT	905.56 ± 391.00	106.08 ± 64.90	4432.66 ± 1939.25	830.45 ± 363.86	749.23 ± 321.00
MiniLM	896.69 ± 391.07	104.58 ± 64.99	4372.92 ± 1913.08	804.84 ± 350.45	720.42 ± 294.63
RoBERTa	923.97 ± 404.07	106.05 ± 64.69	4388.76 ± 1914.68	821.81 ± 357.48	740.35 ± 312.01

Table 4.35: Average meta-feature computation time (s) per embedding model and category, averaged across all datasets for scenario (S1, I1) ↔ (∅, I2).

	ChEMBL	Magellan	OpenData	TPC-DI	Wikidata
ALBERT	894.79 ± 388.09	106.31 ± 62.32	4311.98 ± 1903.45	815.83 ± 353.08	673.04 ± 312.99
BART	896.53 ± 388.99	106.05 ± 61.72	4305.77 ± 1874.76	811.56 ± 350.09	683.93 ± 320.16
BERT	893.42 ± 388.34	106.28 ± 62.04	4314.14 ± 1883.03	811.78 ± 350.53	686.56 ± 317.66
DistilBERT	891.40 ± 396.97	105.75 ± 61.75	4303.58 ± 1871.69	814.32 ± 351.98	681.24 ± 320.11
MiniLM	886.81 ± 390.90	104.60 ± 59.51	4269.73 ± 1872.34	801.72 ± 347.64	667.10 ± 286.17
RoBERTa	891.14 ± 386.48	106.75 ± 61.98	4277.28 ± 1861.24	807.73 ± 349.82	685.26 ± 314.85

Table 4.36: Average meta-feature computation time (s) per embedding model and category, averaged across all datasets for scenario (S1, ∅) ↔ (∅, I2).

	ChEMBL	Magellan	OpenData	TPC-DI	Wikidata
ALBERT	5682.61 ± 2493.69	644.04 ± 390.29	28289.24 ± 12501.94	4777.25 ± 2061.14	3527.49 ± 2325.09
BART	3664.84 ± 1584.36	431.35 ± 269.74	17903.56 ± 7712.81	3378.73 ± 1450.26	2759.49 ± 1364.37
BERT	4474.56 ± 1967.60	512.40 ± 313.41	22995.58 ± 10176.01	4076.94 ± 1766.40	3355.61 ± 1619.46
DistilBERT	4875.94 ± 2089.35	542.59 ± 311.72	23703.00 ± 10415.04	4431.58 ± 1898.39	3498.79 ± 1724.62
MiniLM	871.78 ± 390.20	101.88 ± 60.39	4313.89 ± 1870.45	799.33 ± 346.49	645.65 ± 302.70
RoBERTa	3726.21 ± 1640.68	420.73 ± 249.40	19339.59 ± 8820.29	3449.93 ± 1490.96	2718.17 ± 1297.72

Table 4.37: Training and testing time (in seconds, mean ± std) per classifier and embedding model for scenario (S1, I1) ↔ (S2, I2).

Classifier	**Metric**	albert	bart	bert	distilbert	minilm	roberta	**Classifier Mean**
RandomForest	Train Time	226.13 ± 2.37	225.76 ± 2.81	227.01 ± 5.22	226.75 ± 3.83	227.16 ± 2.77	227.85 ± 5.22	226.78 ± 3.70
	Test Time	2.49 ± 0.16	2.47 ± 0.13	2.48 ± 0.16	2.43 ± 0.11	2.51 ± 0.16	2.48 ± 0.14	2.48 ± 0.14
LogisticRegression	Train Time	80.45 ± 17.61	80.56 ± 17.81	81.15 ± 18.60	81.26 ± 18.58	87.78 ± 20.48	81.11 ± 18.41	82.05 ± 18.58
	Test Time	0.02 ± 0.00	0.02 ± 0.00	0.02 ± 0.00	0.02 ± 0.00	0.02 ± 0.00	0.02 ± 0.00	0.02 ± 0.00
KNN	Train Time	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00	0.18 ± 0.00	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00
	Test Time	105.62 ± 0.72	105.97 ± 0.13	106.19 ± 2.57	105.48 ± 0.77	105.66 ± 0.54	105.89 ± 0.42	105.80 ± 0.86
MLP	Train Time	134.12 ± 20.10	134.55 ± 19.75	134.95 ± 20.02	134.94 ± 20.18	143.08 ± 20.87	135.12 ± 20.19	136.13 ± 20.19
	Test Time	0.21 ± 0.15	0.21 ± 0.16	0.21 ± 0.15	0.21 ± 0.15	0.25 ± 0.16	0.21 ± 0.15	0.22 ± 0.15
CatBoost	Train Time	63.42 ± 1.16	62.36 ± 1.03	62.75 ± 1.32	62.05 ± 1.12	62.65 ± 1.27	62.11 ± 1.10	62.56 ± 1.17
	Test Time	0.14 ± 0.01	0.13 ± 0.01	0.14 ± 0.01	0.15 ± 0.02	0.14 ± 0.01	0.14 ± 0.02	0.14 ± 0.01
XGBoost	Train Time	15.49 ± 0.15	15.43 ± 0.20	15.42 ± 0.21	15.40 ± 0.21	15.47 ± 0.26	15.43 ± 0.27	15.44 ± 0.22
	Test Time	0.38 ± 0.00	0.38 ± 0.01	0.38 ± 0.01	0.38 ± 0.00	0.38 ± 0.01	0.38 ± 0.01	0.38 ± 0.01
Embedding Mean	Train Time	86.64 ± 6.90	86.47 ± 6.94	86.91 ± 7.56	86.76 ± 7.32	89.39 ± 7.61	86.97 ± 7.53	–
	Test Time	18.14 ± 0.17	18.20 ± 0.07	18.24 ± 0.48	18.11 ± 0.18	18.16 ± 0.15	18.19 ± 0.12	–

Table 4.38: Training and testing time (in seconds, mean ± std) per classifier and embedding model for scenario (S1, ∅) ↔ (S2, ∅).

Classifier	**Metric**	albert	bart	bert	distilbert	minilm	roberta	**Classifier Mean**
RandomForest	Train Time	246.96 ± 2.93	248.03 ± 4.13	247.95 ± 3.55	246.77 ± 2.70	247.20 ± 2.39	248.57 ± 3.87	247.58 ± 3.26
	Test Time	2.25 ± 0.06	2.27 ± 0.07	2.26 ± 0.06	2.25 ± 0.04	2.25 ± 0.07	2.30 ± 0.09	2.26 ± 0.06
LogisticRegression	Train Time	58.68 ± 8.62	59.91 ± 11.19	56.47 ± 8.22	56.66 ± 8.42	58.65 ± 8.88	57.41 ± 7.82	57.96 ± 8.86
	Test Time	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00
KNN	Train Time	0.15 ± 0.00	0.17 ± 0.04	0.15 ± 0.00	0.15 ± 0.00	0.16 ± 0.00	0.17 ± 0.03	0.16 ± 0.01
	Test Time	104.59 ± 0.48	104.55 ± 0.44	104.74 ± 0.43	104.61 ± 0.40	104.75 ± 0.45	105.24 ± 2.15	104.75 ± 0.72
MLP	Train Time	159.85 ± 16.46	160.51 ± 16.98	160.52 ± 16.72	160.72 ± 16.61	155.19 ± 26.45	160.92 ± 16.78	159.62 ± 18.34
	Test Time	0.11 ± 0.01	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.01	0.11 ± 0.01	0.11 ± 0.01	0.11 ± 0.00
CatBoost	Train Time	60.56 ± 1.61	60.41 ± 1.59	60.40 ± 1.58	60.35 ± 1.64	60.56 ± 1.51	60.55 ± 1.68	60.47 ± 1.60
	Test Time	0.14 ± 0.01	0.14 ± 0.01	0.14 ± 0.01	0.14 ± 0.01	0.14 ± 0.01	0.14 ± 0.01	0.14 ± 0.01
XGBoost	Train Time	15.68 ± 0.32	15.77 ± 0.46	15.64 ± 0.20	16.18 ± 1.66	16.03 ± 1.09	15.67 ± 0.17	15.83 ± 0.65
	Test Time	0.40 ± 0.00	0.40 ± 0.00	0.40 ± 0.00	0.40 ± 0.00	0.40 ± 0.00	0.40 ± 0.00	0.40 ± 0.00
Embedding Mean	Train Time	90.31 ± 4.99	90.80 ± 5.73	90.19 ± 5.05	90.14 ± 5.17	89.63 ± 6.72	90.55 ± 5.06	–
	Test Time	17.92 ± 0.09	17.91 ± 0.09	17.94 ± 0.08	17.92 ± 0.08	17.94 ± 0.09	18.03 ± 0.38	–

Table 4.39: Training and testing time (in seconds, mean ± std) per classifier and embedding model for scenario (∅, I1) ↔ (∅, I2).

Classifier	**Metric**	albert	bart	bert	distilbert	minilm	roberta	**Classifier Mean**
RandomForest	Train Time	331.12 ± 8.41	329.66 ± 7.16	331.02 ± 12.12	327.00 ± 4.95	327.86 ± 6.52	328.57 ± 8.08	329.21 ± 7.87
	Test Time	3.93 ± 0.45	3.95 ± 0.51	3.76 ± 0.23	3.72 ± 0.17	3.73 ± 0.20	3.82 ± 0.23	3.82 ± 0.30
LogisticRegression	Train Time	145.05 ± 60.70	136.55 ± 47.03	141.35 ± 62.94	138.29 ± 63.17	110.79 ± 14.32	129.19 ± 36.48	133.54 ± 47.44
	Test Time	0.02 ± 0.01	0.02 ± 0.00	0.02 ± 0.00	0.02 ± 0.00	0.02 ± 0.00	0.02 ± 0.00	0.02 ± 0.00
KNN	Train Time	0.16 ± 0.03	0.17 ± 0.03	0.16 ± 0.02	0.16 ± 0.01	0.16 ± 0.00	0.16 ± 0.00	0.16 ± 0.01
	Test Time	89.50 ± 4.76	88.55 ± 3.54	89.04 ± 4.97	90.11 ± 5.29	88.23 ± 2.92	91.16 ± 5.34	89.43 ± 4.47
MLP	Train Time	227.84 ± 55.83	224.55 ± 56.07	223.93 ± 54.62	223.60 ± 53.25	235.47 ± 41.59	224.89 ± 51.74	226.71 ± 52.19
	Test Time	0.12 ± 0.01	0.12 ± 0.01	0.12 ± 0.01	0.12 ± 0.00	0.12 ± 0.01	0.12 ± 0.01	0.12 ± 0.01
CatBoost	Train Time	69.73 ± 2.50	69.35 ± 2.11	68.63 ± 0.60	68.62 ± 0.67	68.75 ± 0.69	68.80 ± 0.68	68.98 ± 1.21
	Test Time	0.16 ± 0.02	0.15 ± 0.01	0.15 ± 0.01	0.15 ± 0.01	0.15 ± 0.01	0.16 ± 0.02	0.15 ± 0.01
XGBoost	Train Time	15.40 ± 0.59	15.57 ± 0.44	15.37 ± 0.36	15.42 ± 0.35	15.32 ± 0.23	15.40 ± 0.27	15.41 ± 0.37
	Test Time	0.39 ± 0.02	0.39 ± 0.02	0.38 ± 0.01	0.39 ± 0.01	0.38 ± 0.01	0.39 ± 0.01	0.39 ± 0.01
Embedding Mean	Train Time	131.55 ± 21.34	129.31 ± 18.81	130.08 ± 21.78	128.85 ± 20.40	126.39 ± 10.56	127.84 ± 16.21	–
	Test Time	15.69 ± 0.88	15.53 ± 0.68	15.58 ± 0.87	15.75 ± 0.91	15.44 ± 0.52	15.94 ± 0.94	–

Table 4.40: Training and testing time (in seconds, mean ± std) per classifier and embedding model for scenario (S1, I1) ↔ (S2, ∅).

Classifier	Metric	albert	bart	bert	distilbert	minilm	roberta	Classifier Mean
RandomForest	Train Time	240.39 ± 2.64	240.95 ± 2.29	241.62 ± 2.74	243.23 ± 4.82	242.01 ± 3.02	242.88 ± 3.90	241.85 ± 3.24
	Test Time	2.34 ± 0.13	2.36 ± 0.08	2.40 ± 0.14	2.39 ± 0.14	2.42 ± 0.15	2.40 ± 0.15	2.39 ± 0.13
LogisticRegression	Train Time	53.89 ± 4.13	53.93 ± 4.60	57.73 ± 13.10	60.79 ± 14.81	59.54 ± 15.19	58.16 ± 14.79	57.34 ± 11.10
	Test Time	0.02 ± 0.00	0.02 ± 0.00	0.02 ± 0.00	0.01 ± 0.00	0.02 ± 0.00	0.01 ± 0.00	0.02 ± 0.00
KNN	Train Time	0.16 ± 0.00	0.16 ± 0.00	0.16 ± 0.00	0.16 ± 0.01	0.16 ± 0.02	0.16 ± 0.02	0.16 ± 0.01
	Test Time	104.99 ± 0.77	105.31 ± 2.85	105.20 ± 0.54	104.90 ± 0.62	105.15 ± 0.59	105.17 ± 0.58	105.12 ± 0.99
MLP	Train Time	155.35 ± 17.50	156.53 ± 17.26	157.02 ± 17.40	156.74 ± 16.96	158.47 ± 17.39	158.04 ± 16.53	157.03 ± 17.17
	Test Time	0.12 ± 0.01	0.11 ± 0.01	0.12 ± 0.01	0.12 ± 0.01	0.11 ± 0.01	0.12 ± 0.01	0.12 ± 0.01
CatBoost	Train Time	61.19 ± 1.05	61.12 ± 0.98	61.54 ± 0.88	61.72 ± 0.91	61.38 ± 1.15	61.44 ± 1.00	61.40 ± 0.99
	Test Time	0.14 ± 0.01	0.13 ± 0.01	0.13 ± 0.01	0.14 ± 0.01	0.14 ± 0.01	0.14 ± 0.01	0.14 ± 0.01
XGBoost	Train Time	15.41 ± 0.49	15.23 ± 0.22	15.32 ± 0.30	15.35 ± 0.25	15.28 ± 0.21	15.38 ± 0.31	15.33 ± 0.30
	Test Time	0.40 ± 0.00	0.40 ± 0.00	0.40 ± 0.01	0.40 ± 0.01	0.40 ± 0.01	0.40 ± 0.01	0.40 ± 0.01
Embedding Mean	Train Time	87.73 ± 4.30	87.99 ± 4.23	88.90 ± 5.74	89.66 ± 6.29	89.47 ± 6.16	89.35 ± 6.09	–
	Test Time	18.00 ± 0.16	18.06 ± 0.49	18.05 ± 0.12	17.99 ± 0.13	18.04 ± 0.13	18.04 ± 0.13	–

Table 4.41: Training and testing time (in seconds, mean ± std) per classifier and embedding model for scenario (S1, I1) ↔ (∅, I2).

Classifier	Metric	albert	bart	bert	distilbert	minilm	roberta	Classifier Mean
RandomForest	Train Time	298.17 ± 3.67	297.53 ± 3.57	298.16 ± 2.49	297.23 ± 3.23	297.96 ± 1.39	297.70 ± 2.52	297.79 ± 2.81
	Test Time	3.53 ± 0.08	3.53 ± 0.07	3.53 ± 0.07	3.58 ± 0.11	3.49 ± 0.07	3.50 ± 0.08	3.53 ± 0.08
LogisticRegression	Train Time	91.05 ± 7.22	91.80 ± 6.97	91.33 ± 7.32	97.49 ± 11.80	96.64 ± 7.70	91.03 ± 7.25	93.22 ± 8.04
	Test Time	0.02 ± 0.01	0.02 ± 0.01	0.02 ± 0.01	0.02 ± 0.01	0.02 ± 0.00	0.02 ± 0.00	0.02 ± 0.01
KNN	Train Time	0.16 ± 0.00	0.17 ± 0.02	0.16 ± 0.00	0.18 ± 0.00	0.16 ± 0.00	0.16 ± 0.00	0.16 ± 0.00
	Test Time	87.42 ± 1.26	86.87 ± 0.49	87.15 ± 1.05	85.86 ± 0.92	87.59 ± 1.67	87.60 ± 2.12	87.08 ± 1.25
MLP	Train Time	273.31 ± 24.31	271.95 ± 23.21	272.94 ± 24.17	216.04 ± 56.42	272.63 ± 33.58	273.20 ± 25.34	263.35 ± 31.17
	Test Time	0.12 ± 0.02	0.12 ± 0.01	0.12 ± 0.01	0.14 ± 0.08	0.11 ± 0.01	0.12 ± 0.01	0.12 ± 0.02
CatBoost	Train Time	68.82 ± 0.69	68.37 ± 0.33	68.84 ± 0.73	67.99 ± 0.20	68.88 ± 0.60	68.42 ± 0.20	68.56 ± 0.46
	Test Time	0.15 ± 0.02	0.15 ± 0.02	0.15 ± 0.01	0.15 ± 0.01	0.15 ± 0.01	0.14 ± 0.00	0.15 ± 0.01
XGBoost	Train Time	14.80 ± 0.20	14.99 ± 0.53	14.75 ± 0.21	14.81 ± 0.27	14.86 ± 0.21	14.83 ± 0.21	14.84 ± 0.27
	Test Time	0.39 ± 0.01	0.39 ± 0.00	0.39 ± 0.01	0.38 ± 0.01	0.39 ± 0.01	0.39 ± 0.00	0.39 ± 0.01
Embedding Mean	Train Time	124.39 ± 6.01	124.14 ± 5.77	124.36 ± 5.82	115.62 ± 11.99	125.19 ± 7.25	124.22 ± 5.92	–
	Test Time	15.27 ± 0.23	15.18 ± 0.10	15.23 ± 0.19	15.02 ± 0.19	15.29 ± 0.29	15.29 ± 0.37	–

Table 4.42: Training and testing time (in seconds, mean ± std) per classifier and embedding model for scenario (S1, ∅) ↔ (∅, I2).

Classifier	Metric	albert	bart	bert	distilbert	minilm	roberta	Classifier Mean
RandomForest	Train Time	394.17 ± 4.96	393.06 ± 5.96	393.01 ± 5.72	396.17 ± 4.30	395.15 ± 4.17	383.51 ± 3.72	392.51 ± 4.81
	Test Time	1.85 ± 0.08	1.81 ± 0.07	1.82 ± 0.12	1.81 ± 0.06	1.93 ± 0.05	1.75 ± 0.05	1.83 ± 0.07
LogisticRegression	Train Time	172.28 ± 16.01	215.76 ± 28.10	134.82 ± 14.08	119.44 ± 12.53	188.46 ± 11.18	158.67 ± 11.82	164.91 ± 15.62
	Test Time	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00
KNN	Train Time	0.21 ± 0.06	0.21 ± 0.05	0.21 ± 0.05	0.19 ± 0.02	0.19 ± 0.02	0.19 ± 0.02	0.20 ± 0.03
	Test Time	39.95 ± 0.36	39.97 ± 0.36	40.02 ± 0.38	39.97 ± 0.31	39.88 ± 0.33	39.99 ± 0.28	39.96 ± 0.34
MLP	Train Time	321.88 ± 67.69	318.43 ± 44.97	292.75 ± 47.07	319.64 ± 47.69	293.90 ± 54.89	315.83 ± 51.22	310.40 ± 52.26
	Test Time	0.06 ± 0.03	0.06 ± 0.01	0.06 ± 0.04	0.06 ± 0.00	0.06 ± 0.00	0.06 ± 0.00	0.06 ± 0.01
CatBoost	Train Time	48.21 ± 0.24	47.78 ± 0.22	48.22 ± 0.92	47.75 ± 0.18	47.89 ± 0.20	47.86 ± 0.25	47.95 ± 0.34
	Test Time	0.06 ± 0.00	0.06 ± 0.00	0.07 ± 0.09	0.06 ± 0.00	0.06 ± 0.00	0.06 ± 0.00	0.06 ± 0.02
XGBoost	Train Time	7.59 ± 0.24	7.57 ± 0.20	8.38 ± 5.50	7.49 ± 0.42	7.55 ± 0.16	7.51 ± 0.18	7.68 ± 1.12
	Test Time	0.12 ± 0.01	0.12 ± 0.01	0.12 ± 0.01	0.12 ± 0.01	0.12 ± 0.01	0.13 ± 0.01	0.12 ± 0.01
Embedding Mean	Train Time	157.39 ± 14.87	163.80 ± 13.25	146.23 ± 12.22	148.45 ± 10.86	155.52 ± 11.77	152.26 ± 11.20	–
	Test Time	7.01 ± 0.08	7.01 ± 0.07	7.02 ± 0.11	7.00 ± 0.06	7.01 ± 0.07	7.00 ± 0.06	–

them clearly slower than MiniLM and DistilBERT. Even for the largest encoders, the embedding step remains below a few tens of seconds per dataset. MiniLM (6 layers, hidden size 384, ∼22M params) and DistilBERT (6 layers, ∼66M params) are fastest; BERT/RoBERTa/BART (12 layers, hidden size 768, >100M params) are slower, while ALBERT benefits from parameter sharing (intermediate runtime).

ALBERT is intermediate: despite having the same depth as BERT (Table 4.3), its parameter-sharing reduces cost, though it remains slower than MiniLM. Across datasets, data sources such as TPC-DI and Open Data demand more embedding time than ones like Magellan, consistent with their larger schema and tuple ranges (Table 4.1).

The meta-feature extraction step emerges as the most time-consuming part of the pipeline. Execution times in Tables 4.31–4.36 show that Open Data and TPC-DI regularly exceed several thousand seconds, sometimes reaching beyond 20,000 seconds, depending on the scenario and embedding model. This stage dominates runtime because it requires exhaustive pairwise meta-feature computation between datasets. As shown in Algorithm 1, for each attribute pair the framework computes a wide spectrum of meta-features, including classical distances, string-based similarities, spectral metrics, and even topological meta-features. Since this process is repeated for all attribute pairs, the cost grows quadratically with schema size. Meta-features are computed pairwise across attributes, which drives the cost: e.g., with 51 attributes (OpenData) per table there are $51 \times 51 = 2601$ pairs, each requiring $h = 112$ meta-features, including spectral and topological meta-features (matrix decompositions, persistence diagrams). The meta-feature computation can be parallelized because each attribute pair is processed independently.

Importantly, execution times remain high regardless of whether embeddings are lightweight or full-size, indicating that optimization of meta-feature computation is central for scalability.

Training and testing times for classifiers are modest compared to embedding and meta-feature computation (Tables 4.37–4.42). Boosting methods (XGBoost, CatBoost) are both accurate and efficient, with training completed within tens of seconds and testing under a second. RandomForest and MLP require longer training, often exceeding 200 seconds, while KNN shifts cost to the testing phase, where exhaustive distance computations make it the slowest method for inference (∼90–105 seconds). Logistic Regression is consistently the fastest in testing, but its low accuracy (RQ1 - 4.3.2) makes the efficiency gains irrelevant in practice.

Discussion. The results reveal distinct efficiency patterns. Embedding costs scale with model complexity: MiniLM and DistilBERT provide competitive runtimes, whereas BERT, RoBERTa, and BART impose significantly higher costs (two to four times slower). Given their limited performance advantage (RQ1-4.3.2), lightweight encoders offer the best trade-off between accuracy and efficiency.

The calculation of meta-features is the most time-consuming in terms of total execution time. On large datasets (TPC-DI, Open Data), it overwhelms both embedding and classification, whereas on smaller datasets it remains moderate. This indicates that improving or approximating meta-feature computation is more impactful for scalability than reducing embedding costs alone.

Classification adds comparatively little overhead. Boosting methods balance effectiveness and speed most effectively, whereas RandomForest and MLP are slower with no accuracy benefit, and KNN suffers from high inference cost. Logistic Regression remains efficient in testing but unusable in practice due to poor predictive quality.

RQ3. Meta-Feature Importance

We fix XGBoost as the classifier, since it was identified as both the most effective and efficient in RQ1 and RQ2. As in all experiments, results are averaged per dataset pair using 10 random seeds

Table 4.43: Precision, Recall, and F1-score per family and embedding model for scenario (S1, I1) ↔ (S2, I2).

Family	**Metric**	ALBERT	BART	BERT	DistilBERT	MiniLM	RoBERTa	**Family Mean**
Topology	Precision	0.75 ± 0.09	0.75 ± 0.09	0.75 ± 0.09	0.75 ± 0.09	0.75 ± 0.09	0.75 ± 0.09	0.75 ± 0.08
	Recall	0.48 ± 0.05	0.48 ± 0.05	0.48 ± 0.05	0.48 ± 0.05	0.48 ± 0.05	0.48 ± 0.05	0.48 ± 0.05
	F1	0.59 ± 0.06	0.59 ± 0.06	0.59 ± 0.06	0.59 ± 0.06	0.59 ± 0.06	0.59 ± 0.06	0.59 ± 0.06
Spectral	Precision	0.48 ± 0.10	0.48 ± 0.10	0.48 ± 0.10	0.48 ± 0.10	0.49 ± 0.10	0.48 ± 0.10	0.48 ± 0.10
	Recall	0.34 ± 0.08	0.34 ± 0.08	0.34 ± 0.08	0.34 ± 0.08	0.34 ± 0.08	0.34 ± 0.08	0.34 ± 0.08
	F1	0.40 ± 0.09	0.40 ± 0.09	0.40 ± 0.09	0.40 ± 0.09	0.40 ± 0.09	0.40 ± 0.09	0.40 ± 0.09
Classical	Precision	0.90 ± 0.02	0.90 ± 0.02	0.90 ± 0.02	0.90 ± 0.02	0.90 ± 0.02	0.90 ± 0.02	0.90 ± 0.02
	Recall	0.95 ± 0.03	0.95 ± 0.03	0.95 ± 0.03	0.95 ± 0.03	0.95 ± 0.03	0.95 ± 0.03	0.95 ± 0.03
	F1	0.93 ± 0.02	0.93 ± 0.02	0.93 ± 0.02	0.93 ± 0.02	0.93 ± 0.02	0.93 ± 0.02	0.93 ± 0.02
Topology+Spectral	Precision	0.85 ± 0.00	0.85 ± 0.00	0.85 ± 0.00	0.85 ± 0.00	0.85 ± 0.00	0.85 ± 0.00	0.85 ± 0.00
	Recall	0.57 ± 0.00	0.57 ± 0.00	0.57 ± 0.00	0.57 ± 0.00	0.57 ± 0.00	0.57 ± 0.00	0.57 ± 0.00
	F1	0.68 ± 0.00	0.68 ± 0.00	0.68 ± 0.00	0.68 ± 0.00	0.68 ± 0.00	0.68 ± 0.00	0.68 ± 0.00
Topology+Classical	Precision	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00
	Recall	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00
	F1	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00
Spectral+Classical	Precision	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00
	Recall	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00
	F1	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00
Topology+Spectral+Classical	Precision	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00
	Recall	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00
	F1	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00

Table 4.44: Precision, Recall, and F1-score per family and embedding model for scenario (S1, ∅) ↔ (S2, ∅).

Family	**Metric**	ALBERT	BART	BERT	DistilBERT	MiniLM	RoBERTa	**Family Mean**
Topology	Precision	0.67 ± 0.16	0.67 ± 0.16	0.67 ± 0.16	0.67 ± 0.16	0.67 ± 0.16	0.67 ± 0.16	0.67 ± 0.16
	Recall	0.20 ± 0.04	0.20 ± 0.04	0.20 ± 0.04	0.20 ± 0.04	0.20 ± 0.04	0.20 ± 0.04	0.20 ± 0.04
	F1	0.31 ± 0.07	0.31 ± 0.07	0.31 ± 0.07	0.31 ± 0.07	0.31 ± 0.07	0.31 ± 0.07	0.31 ± 0.06
Spectral	Precision	0.37 ± 0.19	0.37 ± 0.19	0.37 ± 0.19	0.37 ± 0.19	0.37 ± 0.19	0.37 ± 0.19	0.37 ± 0.18
	Recall	0.10 ± 0.05	0.10 ± 0.05	0.10 ± 0.05	0.10 ± 0.05	0.10 ± 0.05	0.10 ± 0.05	0.10 ± 0.04
	F1	0.15 ± 0.07	0.15 ± 0.07	0.15 ± 0.07	0.15 ± 0.07	0.15 ± 0.07	0.15 ± 0.07	0.15 ± 0.07
Classical	Precision	0.86 ± 0.04	0.86 ± 0.04	0.86 ± 0.04	0.86 ± 0.04	0.86 ± 0.04	0.86 ± 0.04	0.86 ± 0.04
	Recall	0.95 ± 0.03	0.95 ± 0.03	0.95 ± 0.03	0.95 ± 0.03	0.95 ± 0.03	0.95 ± 0.03	0.95 ± 0.03
	F1	0.90 ± 0.04	0.90 ± 0.04	0.90 ± 0.04	0.90 ± 0.04	0.90 ± 0.04	0.90 ± 0.04	0.90 ± 0.03
Topology+Spectral	Precision	0.81 ± 0.00	0.81 ± 0.00	0.81 ± 0.00	0.81 ± 0.00	0.81 ± 0.00	0.81 ± 0.00	0.81 ± 0.00
	Recall	0.25 ± 0.00	0.25 ± 0.00	0.25 ± 0.00	0.25 ± 0.00	0.25 ± 0.00	0.25 ± 0.00	0.25 ± 0.00
	F1	0.38 ± 0.00	0.38 ± 0.00	0.38 ± 0.00	0.38 ± 0.00	0.38 ± 0.00	0.38 ± 0.00	0.38 ± 0.00
Topology+Classical	Precision	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00
	Recall	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00
	F1	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00
Spectral+Classical	Precision	0.92 ± 0.00	0.92 ± 0.00	0.92 ± 0.00	0.92 ± 0.00	0.92 ± 0.00	0.92 ± 0.00	0.92 ± 0.00
	Recall	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00
	F1	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00
Topology+Spectral+Classical	Precision	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00
	Recall	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00
	F1	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00

Table 4.45: Precision, Recall, and F1-score per family and embedding model for scenario (∅, I1) ↔ (∅, I2).

Family	**Metric**	ALBERT	BART	BERT	DistilBERT	MiniLM	RoBERTa	**Family Mean**
Topology	Precision	0.40 ± 0.04	0.40 ± 0.04	0.40 ± 0.04	0.40 ± 0.04	0.40 ± 0.04	0.40 ± 0.04	0.40 ± 0.04
	Recall	0.35 ± 0.04	0.35 ± 0.04	0.35 ± 0.04	0.35 ± 0.04	0.35 ± 0.04	0.35 ± 0.04	0.35 ± 0.04
	F1	0.37 ± 0.04	0.37 ± 0.04	0.37 ± 0.04	0.37 ± 0.04	0.37 ± 0.04	0.37 ± 0.04	0.37 ± 0.04
Spectral	Precision	0.28 ± 0.04	0.28 ± 0.04	0.28 ± 0.04	0.28 ± 0.04	0.28 ± 0.04	0.28 ± 0.04	0.28 ± 0.04
	Recall	0.28 ± 0.02	0.28 ± 0.02	0.28 ± 0.02	0.28 ± 0.02	0.28 ± 0.02	0.28 ± 0.02	0.28 ± 0.02
	F1	0.28 ± 0.03	0.28 ± 0.03	0.28 ± 0.03	0.28 ± 0.03	0.28 ± 0.03	0.28 ± 0.03	0.28 ± 0.03
Classical	Precision	0.24 ± 0.02	0.24 ± 0.02	0.24 ± 0.02	0.24 ± 0.02	0.24 ± 0.02	0.24 ± 0.02	0.24 ± 0.02
	Recall	0.56 ± 0.01	0.56 ± 0.01	0.56 ± 0.01	0.56 ± 0.01	0.56 ± 0.01	0.56 ± 0.01	0.56 ± 0.01
	F1	0.33 ± 0.02	0.33 ± 0.02	0.33 ± 0.02	0.33 ± 0.02	0.33 ± 0.02	0.33 ± 0.02	0.33 ± 0.02
Topology+Spectral	Precision	0.50 ± 0.00	0.50 ± 0.00	0.50 ± 0.00	0.50 ± 0.00	0.50 ± 0.00	0.50 ± 0.00	0.50 ± 0.00
	Recall	0.44 ± 0.00	0.44 ± 0.00	0.44 ± 0.00	0.44 ± 0.00	0.44 ± 0.00	0.44 ± 0.00	0.44 ± 0.00
	F1	0.47 ± 0.00	0.47 ± 0.00	0.47 ± 0.00	0.47 ± 0.00	0.47 ± 0.00	0.47 ± 0.00	0.47 ± 0.00
Topology+Classical	Precision	0.47 ± 0.00	0.47 ± 0.00	0.47 ± 0.00	0.47 ± 0.00	0.47 ± 0.00	0.47 ± 0.00	0.47 ± 0.00
	Recall	0.57 ± 0.00	0.57 ± 0.00	0.57 ± 0.00	0.57 ± 0.00	0.57 ± 0.00	0.57 ± 0.00	0.57 ± 0.00
	F1	0.51 ± 0.00	0.51 ± 0.00	0.51 ± 0.00	0.51 ± 0.00	0.51 ± 0.00	0.51 ± 0.00	0.51 ± 0.00
Spectral+Classical	Precision	0.49 ± 0.00	0.49 ± 0.00	0.49 ± 0.00	0.49 ± 0.00	0.49 ± 0.00	0.49 ± 0.00	0.49 ± 0.00
	Recall	0.65 ± 0.00	0.65 ± 0.00	0.65 ± 0.00	0.65 ± 0.00	0.65 ± 0.00	0.65 ± 0.00	0.65 ± 0.00
	F1	0.56 ± 0.00	0.56 ± 0.00	0.56 ± 0.00	0.56 ± 0.00	0.56 ± 0.00	0.56 ± 0.00	0.56 ± 0.00
Topology+Spectral+Classical	Precision	0.55 ± 0.00	0.55 ± 0.00	0.55 ± 0.00	0.55 ± 0.00	0.55 ± 0.00	0.55 ± 0.00	0.55 ± 0.00
	Recall	0.62 ± 0.00	0.62 ± 0.00	0.62 ± 0.00	0.62 ± 0.00	0.62 ± 0.00	0.62 ± 0.00	0.62 ± 0.00
	F1	0.59 ± 0.00	0.59 ± 0.00	0.59 ± 0.00	0.59 ± 0.00	0.59 ± 0.00	0.59 ± 0.00	0.59 ± 0.00

Table 4.46: Precision, Recall, and F1-score per family and embedding model for scenario (S1, I1) ↔ (S2, ∅).

Family	**Metric**	ALBERT	BART	BERT	DistilBERT	MiniLM	RoBERTa	**Family Mean**
Topology	Precision	0.56 ± 0.28	0.56 ± 0.28	0.56 ± 0.28	0.56 ± 0.28	0.56 ± 0.28	0.56 ± 0.28	0.56 ± 0.27
	Recall	0.14 ± 0.07	0.14 ± 0.07	0.14 ± 0.07	0.14 ± 0.07	0.14 ± 0.07	0.14 ± 0.07	0.14 ± 0.07
	F1	0.22 ± 0.11	0.22 ± 0.11	0.22 ± 0.11	0.22 ± 0.11	0.22 ± 0.11	0.22 ± 0.11	0.22 ± 0.11
Spectral	Precision	0.34 ± 0.17	0.34 ± 0.17	0.34 ± 0.17	0.34 ± 0.17	0.34 ± 0.21	0.34 ± 0.17	0.34 ± 0.17
	Recall	0.10 ± 0.04	0.10 ± 0.04	0.10 ± 0.04	0.10 ± 0.04	0.10 ± 0.05	0.10 ± 0.04	0.10 ± 0.05
	F1	0.15 ± 0.07	0.15 ± 0.07	0.15 ± 0.07	0.15 ± 0.07	0.16 ± 0.09	0.15 ± 0.07	0.15 ± 0.07
Classical	Precision	0.89 ± 0.02	0.89 ± 0.02	0.89 ± 0.02	0.89 ± 0.02	0.89 ± 0.02	0.89 ± 0.02	0.89 ± 0.02
	Recall	0.93 ± 0.05	0.93 ± 0.05	0.93 ± 0.05	0.93 ± 0.05	0.93 ± 0.05	0.93 ± 0.05	0.93 ± 0.05
	F1	0.91 ± 0.03	0.91 ± 0.03	0.91 ± 0.03	0.91 ± 0.03	0.91 ± 0.03	0.91 ± 0.03	0.91 ± 0.03
Topology+Spectral	Precision	0.86 ± 0.00	0.86 ± 0.00	0.86 ± 0.00	0.86 ± 0.00	0.86 ± 0.00	0.86 ± 0.00	0.86 ± 0.00
	Recall	0.22 ± 0.00	0.22 ± 0.00	0.22 ± 0.00	0.22 ± 0.00	0.21 ± 0.00	0.22 ± 0.00	0.22 ± 0.00
	F1	0.35 ± 0.00	0.35 ± 0.00	0.35 ± 0.00	0.35 ± 0.00	0.34 ± 0.00	0.35 ± 0.00	0.35 ± 0.00
Topology+Classical	Precision	0.92 ± 0.00	0.92 ± 0.00	0.92 ± 0.00	0.92 ± 0.00	0.92 ± 0.00	0.92 ± 0.00	0.92 ± 0.00
	Recall	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00
	F1	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00
Spectral+Classical	Precision	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00
	Recall	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00
	F1	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00
Topology+Spectral+Classical	Precision	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00
	Recall	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00
	F1	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00

Table 4.47: Precision, Recall, and F1-score per family and embedding model for scenario (S1, I1) ↔ (∅, I2).

Family	**Metric**	ALBERT	BART	BERT	DistilBERT	MiniLM	RoBERTa	**Family Mean**
Topology	Precision	0.56 ± 0.04	0.56 ± 0.04	0.56 ± 0.04	0.55 ± 0.06	0.56 ± 0.04	0.56 ± 0.04	0.56 ± 0.05
	Recall	0.45 ± 0.04	0.45 ± 0.04	0.45 ± 0.04	0.46 ± 0.05	0.45 ± 0.04	0.45 ± 0.04	0.45 ± 0.04
	F1	0.50 ± 0.04	0.50 ± 0.04	0.50 ± 0.04	0.50 ± 0.05	0.50 ± 0.04	0.50 ± 0.04	0.50 ± 0.04
Spectral	Precision	0.37 ± 0.05	0.37 ± 0.05	0.37 ± 0.05	0.39 ± 0.03	0.37 ± 0.05	0.37 ± 0.05	0.37 ± 0.05
	Recall	0.41 ± 0.05	0.41 ± 0.05	0.41 ± 0.05	0.41 ± 0.04	0.41 ± 0.05	0.41 ± 0.05	0.41 ± 0.04
	F1	0.39 ± 0.05	0.39 ± 0.05	0.39 ± 0.05	0.40 ± 0.03	0.39 ± 0.05	0.39 ± 0.05	0.39 ± 0.05
Classical	Precision	0.48 ± 0.02	0.48 ± 0.02	0.48 ± 0.02	0.37 ± 0.15	0.48 ± 0.02	0.48 ± 0.02	0.46 ± 0.07
	Recall	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.64 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.62 ± 0.01
	F1	0.54 ± 0.02	0.54 ± 0.02	0.54 ± 0.02	0.45 ± 0.12	0.54 ± 0.02	0.54 ± 0.02	0.52 ± 0.06
Topology+Spectral	Precision	0.64 ± 0.00	0.64 ± 0.00	0.64 ± 0.00	0.62 ± 0.00	0.63 ± 0.00	0.64 ± 0.00	0.64 ± 0.01
	Recall	0.55 ± 0.00	0.55 ± 0.00	0.55 ± 0.00	0.54 ± 0.00	0.55 ± 0.00	0.55 ± 0.00	0.55 ± 0.00
	F1	0.59 ± 0.00	0.59 ± 0.00	0.59 ± 0.00	0.58 ± 0.00	0.59 ± 0.00	0.59 ± 0.00	0.59 ± 0.00
Topology+Classical	Precision	0.64 ± 0.00	0.64 ± 0.00	0.64 ± 0.00	0.63 ± 0.00	0.64 ± 0.00	0.64 ± 0.00	0.63 ± 0.00
	Recall	0.67 ± 0.00	0.67 ± 0.00	0.67 ± 0.00	0.67 ± 0.00	0.67 ± 0.00	0.67 ± 0.00	0.67 ± 0.00
	F1	0.65 ± 0.00	0.65 ± 0.00	0.65 ± 0.00	0.65 ± 0.00	0.65 ± 0.00	0.65 ± 0.00	0.65 ± 0.00
Spectral+Classical	Precision	0.59 ± 0.00	0.59 ± 0.00	0.59 ± 0.00	0.58 ± 0.00	0.59 ± 0.00	0.59 ± 0.00	0.59 ± 0.00
	Recall	0.72 ± 0.00	0.72 ± 0.00	0.72 ± 0.00	0.73 ± 0.00	0.72 ± 0.00	0.72 ± 0.00	0.72 ± 0.00
	F1	0.65 ± 0.00	0.65 ± 0.00	0.65 ± 0.00	0.65 ± 0.00	0.64 ± 0.00	0.65 ± 0.00	0.65 ± 0.00
Topology+Spectral+Classical	Precision	0.65 ± 0.00	0.65 ± 0.00	0.65 ± 0.00	0.64 ± 0.00	0.66 ± 0.00	0.65 ± 0.00	0.65 ± 0.01
	Recall	0.69 ± 0.00	0.69 ± 0.00	0.69 ± 0.00	0.70 ± 0.00	0.69 ± 0.00	0.69 ± 0.00	0.69 ± 0.00
	F1	0.67 ± 0.00	0.67 ± 0.00	0.67 ± 0.00	0.67 ± 0.00	0.67 ± 0.00	0.67 ± 0.00	0.67 ± 0.00

Table 4.48: Precision, Recall, and F1-score per family and embedding model for scenario (S1, ∅) ↔ (∅, I2).

Family	**Metric**	ALBERT	BART	BERT	DistilBERT	MiniLM	RoBERTa	**Family Mean**
Topology	Precision	0.55 ± 0.08	0.58 ± 0.15	0.57 ± 0.14	0.66 ± 0.13	0.53 ± 0.18	0.54 ± 0.18	0.57 ± 0.15
	Recall	0.29 ± 0.10	0.27 ± 0.10	0.28 ± 0.11	0.30 ± 0.10	0.13 ± 0.05	0.25 ± 0.15	0.25 ± 0.12
	F1	0.37 ± 0.11	0.37 ± 0.13	0.37 ± 0.13	0.41 ± 0.12	0.21 ± 0.07	0.33 ± 0.18	0.35 ± 0.14
Spectral	Precision	0.46 ± 0.06	0.55 ± 0.06	0.54 ± 0.04	0.51 ± 0.07	0.34 ± 0.06	0.50 ± 0.08	0.48 ± 0.09
	Recall	0.30 ± 0.06	0.34 ± 0.04	0.31 ± 0.06	0.33 ± 0.08	0.18 ± 0.06	0.29 ± 0.08	0.29 ± 0.08
	F1	0.36 ± 0.06	0.42 ± 0.05	0.39 ± 0.06	0.40 ± 0.08	0.23 ± 0.07	0.36 ± 0.09	0.36 ± 0.09
Classical	Precision	0.45 ± 0.06	0.46 ± 0.07	0.49 ± 0.07	0.48 ± 0.07	0.44 ± 0.04	0.46 ± 0.09	0.46 ± 0.07
	Recall	0.33 ± 0.05	0.33 ± 0.04	0.33 ± 0.05	0.33 ± 0.07	0.29 ± 0.03	0.35 ± 0.09	0.33 ± 0.06
	F1	0.38 ± 0.05	0.38 ± 0.05	0.39 ± 0.06	0.39 ± 0.08	0.35 ± 0.03	0.40 ± 0.10	0.38 ± 0.07
Topology+Spectral	Precision	0.69 ± 0.00	0.78 ± 0.00	0.71 ± 0.00	0.78 ± 0.00	0.73 ± 0.00	0.73 ± 0.00	0.74 ± 0.03
	Recall	0.41 ± 0.00	0.40 ± 0.00	0.41 ± 0.00	0.42 ± 0.00	0.21 ± 0.00	0.41 ± 0.00	0.38 ± 0.08
	F1	0.51 ± 0.00	0.53 ± 0.00	0.52 ± 0.00	0.55 ± 0.00	0.33 ± 0.00	0.53 ± 0.00	0.49 ± 0.08
Topology+Classical	Precision	0.63 ± 0.00	0.69 ± 0.00	0.67 ± 0.00	0.69 ± 0.00	0.59 ± 0.00	0.66 ± 0.00	0.66 ± 0.03
	Recall	0.50 ± 0.00	0.50 ± 0.00	0.50 ± 0.00	0.52 ± 0.00	0.37 ± 0.00	0.55 ± 0.00	0.49 ± 0.06
	F1	0.56 ± 0.00	0.58 ± 0.00	0.57 ± 0.00	0.59 ± 0.00	0.45 ± 0.00	0.60 ± 0.00	0.56 ± 0.05
Spectral+Classical	Precision	0.63 ± 0.00	0.64 ± 0.00	0.63 ± 0.00	0.67 ± 0.00	0.56 ± 0.00	0.62 ± 0.00	0.63 ± 0.03
	Recall	0.52 ± 0.00	0.55 ± 0.00	0.53 ± 0.00	0.57 ± 0.00	0.43 ± 0.00	0.58 ± 0.00	0.53 ± 0.05
	F1	0.57 ± 0.00	0.59 ± 0.00	0.58 ± 0.00	0.62 ± 0.00	0.49 ± 0.00	0.60 ± 0.00	0.57 ± 0.04
Topology+Spectral+Classical	Precision	0.65 ± 0.00	0.68 ± 0.00	0.68 ± 0.00	0.71 ± 0.00	0.61 ± 0.00	0.67 ± 0.00	0.67 ± 0.03
	Recall	0.53 ± 0.00	0.53 ± 0.00	0.53 ± 0.00	0.53 ± 0.00	0.40 ± 0.00	0.57 ± 0.00	0.52 ± 0.05
	F1	0.58 ± 0.00	0.59 ± 0.00	0.60 ± 0.00	0.61 ± 0.00	0.48 ± 0.00	0.62 ± 0.00	0.58 ± 0.05

and 10 repeated runs per seed, with values reported as mean ± std. Tables 4.43–4.48 present precision, recall, and F1 for each meta-feature family (Classical, Spectral, Topological) and their combinations across scenarios. A clear trend emerges in the compliant schema-available settings (C1–C2): Classical meta-features alone already deliver high precision and recall, and when paired with Topological or Spectral meta-features, performance becomes nearly perfect, with F1 consistently above 0.94. In contrast, scenarios (C3, D5, D6) reveal a dependency on meta-feature diversity. Neither single families nor two-family combinations suffice; the best performance arises only when all three families (Classical, Spectral, and Topological) are combined.

Discussion. These results underline the complementary nature of the three meta-feature families. Classical meta-features are generally the most discriminative, driving high performance especially when schema information is available. Their combination with Topological or Spectral meta-features further stabilizes predictions, yielding near-perfect alignment under schema-available conditions. However, in schema-scarce scenarios dominated by instances (C3, D5 and D6), performance drops if any family is omitted. Here, only the joint use of all three families ensures robust results, demonstrating that Topological and Spectral meta-features play a critical compensatory role when schema information is absent. Overall, the findings confirm that Classical meta-features are the strongest individual contributors, but the full tri-family design is essential to maintain stability and effectiveness across all scenarios.

RQ4. Meta-Feature Selection

The objective of this research question is to assess whether automatically selecting a subset of meta-features can improve effectiveness or stability—by comparing performance with and without selection and identifying a reduced meta-space that preserves effectiveness while reducing redundancy and improving efficiency.

Meta-feature abbreviations. Each abbreviation concatenates three parts in the form `<core><op?><scope>`:

- **Core tag** — e.g., `Cos`, `NESum`, `h1_count`.
- **Operator tag** (topology only, optional) — e.g., `B`, `W`, `E0`, `E1`.
- **Scope tag** (mandatory) — `_A1`, `_A2` for individual embeddings; `_a1`, `_a2` for local point clouds; `_a1a2`, `_A1A2` for pairwise comparisons.

Full definitions appear in Table 4.49.

Feature selection protocol. Before selection, we drop any pair of meta-features with absolute Pearson correlation > 0.90. We then compare four strategies—Boruta, SelectFromModel, and backward stepwise regression with and without correlation filtering—run 100 times (10-fold CV $\times$ 10 repetitions) with XGBoost and aggregated over the **551 datasets**. All selection outcomes are reported on the website, while only the backward stepwise regression without correlation method is shown here, since it produces the smallest and most stable meta-space. Backward stepwise starts from the full meta-space and iteratively removes the least informative features by minimizing the Akaike Information Criterion (AIC).

This criterion balances model fit and complexity by penalizing the number of parameters. At each iteration, the feature whose removal most decreases AIC is discarded until no further improvement is possible. The resulting subset minimizes information loss while avoiding overfitting, leading to a compact yet expressive meta-space.

Table 4.49: Meta-feature abbreviations

Core tags	
Classical	Cos (Cosine), Eucl (Euclidean), Mink (Minkowski), Cheb (Chebyshev), Canb (Canberra), Pear (Pearson), Spea (Spearman), *syntactic similarity*: len (length), eq (exact equality), eqcf (case-fold equality), in (substring containment), lev (Levenshtein distance), levS (Levenshtein similarity), dosa (Damerau–OSA distance), dosaS (Damerau–OSA similarity), lcsL (LCS length), lcsR (LCS ratio), cprefR (common prefix ratio), csuffR (common suffix ratio), jacTok (Jaccard on tokens), diceTok (Dice on tokens), jacBG (Jaccard on bigrams), jacTG (Jaccard on trigrams), cosBG (Cosine on bigrams), cosTG (Cosine on trigrams), jaro (Jaro), JW (Jaro–Winkler)
Spectral	SR (Stable Rank), RM (RankMe), aRQ (Alpha–ReQ), NES (NESum), SC (SelfCluster)
Topological	*grounds*: cos (Cosine), euc (Euclidean), man (Manhattan), *invariants*: h0c (H_0 count), h1c (H_1 count), h0ml (H_0 max life), h1ml (H_1 max life)
Operator tags (topological only)	
B	bottleneck $\|\cdot\|_\infty$
W	Wasserstein ($p = 2$)
E0	persistence entropy on H_0
E1	persistence entropy on H_1
Scope tags	
$\_A1$, $\_A2$	whole embedding matrices $E_{\mathcal{CS}_1}$ or $E_{\mathcal{CS}_2}$
$\_a1$, $\_a2$	local point clouds P_1^i or P_2^j
$\_E1E2$	pairwise metrics on ($E_{\mathcal{CS}_1}$,$E_{\mathcal{CS}_2}$) (remplace $\_A1A2$)
$\_P1P2$	pairwise metrics on (P_1^i, P_2^j) (remplace _a1a2)
$\_a$, $\_b$	(syntactic) individual strings a or b
$\_ab$	(syntactic) pairwise metric on (a, b)
$\_a\_in\_b$, $\_b\_in\_a$	(syntactic) directionnels pour `in` : $a \subset b$ ou $b \subset a$

Table 4.51: Precision, Recall, and F1-score (mean ± std) per classifier and embedding model for scenario (S1, I1) ↔ (S2, I2)— after feature selection

Classifier	**Metric**	albert	bart	bert	distilbert	minilm	roberta	**Classifier Mean**
RandomForest	Precision	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	Recall	0.82 ± 0.01	0.82 ± 0.01	0.82 ± 0.01	0.82 ± 0.01	0.83 ± 0.01	0.82 ± 0.01	0.82 ± 0.01
	F1	0.90 ± 0.00	0.90 ± 0.00	0.90 ± 0.00	0.90 ± 0.00	0.90 ± 0.00	0.90 ± 0.00	0.90 ± 0.00
LogisticRegression	Precision	0.27 ± 0.00	0.27 ± 0.00	0.27 ± 0.00	0.27 ± 0.00	0.27 ± 0.00	0.27 ± 0.00	0.27 ± 0.00
	Recall	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00
	F1	0.42 ± 0.00	0.42 ± 0.00	0.42 ± 0.00	0.42 ± 0.00	0.42 ± 0.00	0.42 ± 0.00	0.42 ± 0.00
KNN	Precision	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00
	Recall	0.74 ± 0.00	0.74 ± 0.00	0.74 ± 0.00	0.74 ± 0.00	0.74 ± 0.00	0.74 ± 0.00	0.74 ± 0.00
	F1	0.84 ± 0.00	0.84 ± 0.00	0.84 ± 0.00	0.84 ± 0.00	0.84 ± 0.00	0.84 ± 0.00	0.84 ± 0.00
MLP	Precision	0.91 ± 0.02	0.91 ± 0.02	0.92 ± 0.02	0.92 ± 0.02	0.91 ± 0.01	0.92 ± 0.02	0.91 ± 0.02
	Recall	0.89 ± 0.02	0.89 ± 0.02	0.89 ± 0.02	0.89 ± 0.02	0.89 ± 0.01	0.89 ± 0.02	0.89 ± 0.02
	F1	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01
CatBoost	Precision	0.88 ± 0.01	0.88 ± 0.01	0.88 ± 0.01	0.88 ± 0.01	0.87 ± 0.02	0.88 ± 0.01	0.88 ± 0.01
	Recall	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00
	F1	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01
XGBoost	Precision	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00
	Recall	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00
	F1	0.96 ± 0.00	0.96 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00

Table 4.52: Precision, Recall, and F1-score (mean ± std) per classifier and embedding model for scenario (S1, ∅) ↔ (S2, ∅)— after feature selection

Classifier	**Metric**	albert	bart	bert	distilbert	minilm	roberta	**Classifier Mean**
RandomForest	Precision	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00
	Recall	0.78 ± 0.01	0.78 ± 0.01	0.78 ± 0.01	0.78 ± 0.01	0.78 ± 0.01	0.78 ± 0.01	0.78 ± 0.01
	F1	0.87 ± 0.00	0.87 ± 0.00	0.87 ± 0.00	0.87 ± 0.00	0.87 ± 0.00	0.87 ± 0.00	0.87 ± 0.00
LogisticRegression	Precision	0.25 ± 0.00	0.25 ± 0.00	0.25 ± 0.00	0.25 ± 0.00	0.25 ± 0.00	0.25 ± 0.00	0.25 ± 0.00
	Recall	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00
	F1	0.39 ± 0.00	0.39 ± 0.00	0.39 ± 0.00	0.39 ± 0.00	0.39 ± 0.00	0.39 ± 0.00	0.39 ± 0.00
KNN	Precision	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01
	Recall	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01
	F1	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01
MLP	Precision	0.89 ± 0.02	0.89 ± 0.02	0.89 ± 0.02	0.89 ± 0.02	0.88 ± 0.02	0.89 ± 0.02	0.89 ± 0.02
	Recall	0.86 ± 0.03	0.86 ± 0.03	0.86 ± 0.03	0.86 ± 0.03	0.87 ± 0.02	0.86 ± 0.03	0.86 ± 0.02
	F1	0.87 ± 0.01	0.87 ± 0.01	0.87 ± 0.01	0.87 ± 0.01	0.87 ± 0.01	0.87 ± 0.01	0.87 ± 0.01
CatBoost	Precision	0.81 ± 0.01	0.81 ± 0.01	0.81 ± 0.01	0.81 ± 0.01	0.81 ± 0.01	0.81 ± 0.01	0.81 ± 0.01
	Recall	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00
	F1	0.89 ± 0.01	0.89 ± 0.01	0.89 ± 0.01	0.89 ± 0.01	0.89 ± 0.01	0.89 ± 0.01	0.89 ± 0.01
XGBoost	Precision	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01
	Recall	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00
	F1	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00

Table 4.53: Precision, Recall, and F1-score (mean ± std) per classifier and embedding model for scenario (∅, I1) ↔ (∅, I2)— after feature selection

Classifier	**Metric**	albert	bart	bert	distilbert	minilm	roberta	**Classifier Mean**
RandomForest	Precision	0.91 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01
	Recall	0.26 ± 0.01	0.26 ± 0.00	0.26 ± 0.00	0.26 ± 0.00	0.26 ± 0.00	0.26 ± 0.00	0.26 ± 0.00
	F1	0.41 ± 0.01	0.41 ± 0.00	0.41 ± 0.00	0.41 ± 0.00	0.41 ± 0.00	0.41 ± 0.00	0.41 ± 0.00
LogisticRegression	Precision	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00
	Recall	0.82 ± 0.01	0.82 ± 0.00	0.82 ± 0.00	0.82 ± 0.00	0.82 ± 0.00	0.82 ± 0.00	0.82 ± 0.00
	F1	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00
KNN	Precision	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01
	Recall	0.29 ± 0.01	0.29 ± 0.01	0.29 ± 0.01	0.29 ± 0.01	0.29 ± 0.01	0.29 ± 0.01	0.29 ± 0.01
	F1	0.43 ± 0.01	0.43 ± 0.01	0.43 ± 0.01	0.43 ± 0.01	0.43 ± 0.01	0.43 ± 0.01	0.43 ± 0.01
MLP	Precision	0.74 ± 0.03	0.74 ± 0.03	0.74 ± 0.03	0.74 ± 0.03	0.75 ± 0.01	0.74 ± 0.03	0.74 ± 0.02
	Recall	0.41 ± 0.03	0.41 ± 0.03	0.41 ± 0.03	0.41 ± 0.03	0.40 ± 0.02	0.41 ± 0.03	0.41 ± 0.03
	F1	0.53 ± 0.02	0.53 ± 0.02	0.53 ± 0.02	0.53 ± 0.02	0.52 ± 0.02	0.53 ± 0.02	0.53 ± 0.02
CatBoost	Precision	0.28 ± 0.00	0.28 ± 0.00	0.28 ± 0.00	0.28 ± 0.00	0.28 ± 0.00	0.28 ± 0.00	0.28 ± 0.00
	Recall	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01
	F1	0.41 ± 0.00	0.41 ± 0.00	0.41 ± 0.00	0.41 ± 0.00	0.41 ± 0.00	0.41 ± 0.00	0.41 ± 0.00
XGBoost	Precision	0.50 ± 0.01	0.50 ± 0.01	0.50 ± 0.01	0.50 ± 0.01	0.50 ± 0.01	0.50 ± 0.01	0.50 ± 0.01
	Recall	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01
	F1	0.55 ± 0.01	0.55 ± 0.01	0.55 ± 0.01	0.55 ± 0.01	0.55 ± 0.01	0.55 ± 0.01	0.55 ± 0.01

Table 4.54: Precision, Recall, and F1-score (mean ± std) per classifier and embedding model for scenario (S1, I1) ↔ (S2, ∅)— after feature selection

Classifier	**Metric**	albert	bart	bert	distilbert	minilm	roberta	**Classifier Mean**
RandomForest	Precision	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00
	Recall	0.80 ± 0.00	0.80 ± 0.00	0.80 ± 0.00	0.80 ± 0.00	0.80 ± 0.00	0.80 ± 0.00	0.80 ± 0.00
	F1	0.88 ± 0.00	0.88 ± 0.00	0.88 ± 0.00	0.88 ± 0.00	0.88 ± 0.00	0.88 ± 0.00	0.88 ± 0.00
LogisticRegression	Precision	0.26 ± 0.00	0.26 ± 0.00	0.26 ± 0.00	0.26 ± 0.00	0.26 ± 0.00	0.26 ± 0.00	0.26 ± 0.00
	Recall	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00
	F1	0.40 ± 0.00	0.40 ± 0.00	0.40 ± 0.00	0.40 ± 0.00	0.40 ± 0.00	0.40 ± 0.00	0.40 ± 0.00
KNN	Precision	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01
	Recall	0.60 ± 0.01	0.60 ± 0.01	0.60 ± 0.01	0.60 ± 0.01	0.61 ± 0.00	0.61 ± 0.00	0.61 ± 0.00
	F1	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.00	0.74 ± 0.00	0.74 ± 0.01
MLP	Precision	0.86 ± 0.04	0.86 ± 0.04	0.86 ± 0.04	0.86 ± 0.04	0.85 ± 0.04	0.85 ± 0.04	0.86 ± 0.04
	Recall	0.88 ± 0.03	0.88 ± 0.03	0.88 ± 0.03	0.88 ± 0.03	0.89 ± 0.03	0.89 ± 0.03	0.88 ± 0.03
	F1	0.87 ± 0.01	0.87 ± 0.01	0.87 ± 0.01	0.87 ± 0.01	0.87 ± 0.01	0.87 ± 0.01	0.87 ± 0.01
CatBoost	Precision	0.83 ± 0.01	0.83 ± 0.01	0.83 ± 0.01	0.83 ± 0.01	0.83 ± 0.01	0.83 ± 0.01	0.83 ± 0.01
	Recall	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00
	F1	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.00	0.90 ± 0.01	0.90 ± 0.01
XGBoost	Precision	0.92 ± 0.00	0.92 ± 0.00	0.92 ± 0.00	0.92 ± 0.00	0.92 ± 0.00	0.92 ± 0.00	0.92 ± 0.00
	Recall	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00
	F1	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.94 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.94 ± 0.00

Table 4.55: Precision, Recall, and F1-score (mean ± std) per classifier and embedding model for scenario (S1, I1) ↔ (∅, I2)— after feature selection

Classifier	**Metric**	albert	bart	bert	distilbert	minilm	roberta	**Classifier Mean**
RandomForest	Precision	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.00	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01
	Recall	0.40 ± 0.01	0.40 ± 0.01	0.40 ± 0.01	0.40 ± 0.01	0.40 ± 0.01	0.40 ± 0.01	0.40 ± 0.01
	F1	0.56 ± 0.01	0.56 ± 0.01	0.57 ± 0.01	0.57 ± 0.00	0.57 ± 0.01	0.57 ± 0.01	0.56 ± 0.01
LogisticRegression	Precision	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00
	Recall	0.81 ± 0.01	0.81 ± 0.01	0.81 ± 0.01	0.82 ± 0.00	0.81 ± 0.01	0.81 ± 0.01	0.81 ± 0.01
	F1	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00
KNN	Precision	0.88 ± 0.00	0.88 ± 0.00	0.88 ± 0.00	0.87 ± 0.01	0.88 ± 0.00	0.88 ± 0.00	0.88 ± 0.00
	Recall	0.45 ± 0.01	0.45 ± 0.01	0.45 ± 0.01	0.46 ± 0.00	0.45 ± 0.01	0.45 ± 0.01	0.45 ± 0.01
	F1	0.60 ± 0.01	0.60 ± 0.01	0.60 ± 0.01	0.60 ± 0.00	0.60 ± 0.01	0.60 ± 0.01	0.60 ± 0.01
MLP	Precision	0.73 ± 0.06	0.73 ± 0.06	0.72 ± 0.06	0.78 ± 0.03	0.72 ± 0.06	0.72 ± 0.06	0.73 ± 0.05
	Recall	0.52 ± 0.01	0.52 ± 0.01	0.53 ± 0.01	0.51 ± 0.03	0.53 ± 0.01	0.53 ± 0.01	0.52 ± 0.01
	F1	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.02	0.61 ± 0.01	0.61 ± 0.02	0.61 ± 0.02	0.61 ± 0.02
CatBoost	Precision	0.38 ± 0.01	0.37 ± 0.01	0.37 ± 0.01	0.37 ± 0.01	0.37 ± 0.01	0.37 ± 0.01	0.37 ± 0.01
	Recall	0.78 ± 0.01	0.78 ± 0.01	0.78 ± 0.01	0.78 ± 0.01	0.78 ± 0.01	0.78 ± 0.01	0.78 ± 0.01
	F1	0.51 ± 0.01	0.51 ± 0.01	0.51 ± 0.01	0.50 ± 0.01	0.51 ± 0.01	0.51 ± 0.01	0.50 ± 0.01
XGBoost	Precision	0.60 ± 0.02	0.60 ± 0.02	0.60 ± 0.02	0.59 ± 0.01	0.60 ± 0.02	0.60 ± 0.02	0.60 ± 0.02
	Recall	0.68 ± 0.01	0.67 ± 0.01	0.67 ± 0.01	0.67 ± 0.01	0.67 ± 0.01	0.67 ± 0.01	0.67 ± 0.01
	F1	0.64 ± 0.01	0.63 ± 0.01	0.63 ± 0.01	0.63 ± 0.01	0.63 ± 0.01	0.63 ± 0.01	0.63 ± 0.01

Table 4.56: Precision, Recall, and F1-score (mean ± std) per classifier and embedding model for scenario (S1, ∅) ↔ (∅, I2)— after feature selection

Classifier	**Metric**	albert	bart	bert	distilbert	minilm	roberta	**Classifier Mean**
RandomForest	Precision	0.97 ± 0.01	0.98 ± 0.00	0.97 ± 0.01	0.97 ± 0.00	0.97 ± 0.01	0.97 ± 0.01	0.97 ± 0.01
	Recall	0.22 ± 0.01	0.24 ± 0.01	0.23 ± 0.01	0.26 ± 0.01	0.18 ± 0.01	0.25 ± 0.01	0.23 ± 0.01
	F1	0.36 ± 0.01	0.39 ± 0.01	0.37 ± 0.01	0.41 ± 0.01	0.30 ± 0.01	0.40 ± 0.01	0.37 ± 0.01
LogisticRegression	Precision	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00
	Recall	0.65 ± 0.01	0.65 ± 0.01	0.64 ± 0.01	0.64 ± 0.01	0.59 ± 0.01	0.65 ± 0.01	0.64 ± 0.01
	F1	0.10 ± 0.00	0.10 ± 0.00	0.09 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.00
KNN	Precision	0.81 ± 0.01	0.84 ± 0.01	0.82 ± 0.02	0.78 ± 0.01	0.79 ± 0.02	0.82 ± 0.01	0.81 ± 0.01
	Recall	0.25 ± 0.01	0.30 ± 0.01	0.27 ± 0.01	0.27 ± 0.01	0.23 ± 0.01	0.31 ± 0.01	0.27 ± 0.01
	F1	0.39 ± 0.01	0.44 ± 0.01	0.41 ± 0.01	0.40 ± 0.01	0.36 ± 0.01	0.45 ± 0.01	0.41 ± 0.01
MLP	Precision	0.71 ± 0.03	0.72 ± 0.06	0.71 ± 0.08	0.72 ± 0.05	0.69 ± 0.07	0.72 ± 0.04	0.71 ± 0.05
	Recall	0.21 ± 0.02	0.32 ± 0.04	0.24 ± 0.03	0.26 ± 0.03	0.20 ± 0.02	0.28 ± 0.02	0.25 ± 0.03
	F1	0.33 ± 0.02	0.44 ± 0.03	0.36 ± 0.02	0.38 ± 0.03	0.31 ± 0.02	0.41 ± 0.03	0.37 ± 0.02
CatBoost	Precision	0.25 ± 0.00	0.34 ± 0.01	0.23 ± 0.00	0.25 ± 0.00	0.20 ± 0.00	0.25 ± 0.00	0.25 ± 0.00
	Recall	0.66 ± 0.01	0.74 ± 0.01	0.62 ± 0.01	0.64 ± 0.01	0.59 ± 0.01	0.64 ± 0.01	0.65 ± 0.01
	F1	0.36 ± 0.00	0.47 ± 0.01	0.33 ± 0.00	0.36 ± 0.00	0.29 ± 0.00	0.36 ± 0.00	0.36 ± 0.00
XGBoost	Precision	0.59 ± 0.01	0.64 ± 0.01	0.60 ± 0.02	0.63 ± 0.01	0.55 ± 0.02	0.63 ± 0.01	0.61 ± 0.01
	Recall	0.46 ± 0.01	0.52 ± 0.01	0.44 ± 0.01	0.47 ± 0.01	0.36 ± 0.01	0.48 ± 0.01	0.46 ± 0.01
	F1	0.51 ± 0.01	0.57 ± 0.01	0.51 ± 0.01	0.54 ± 0.01	0.44 ± 0.01	0.55 ± 0.01	0.52 ± 0.01

Table 4.57: Precision, Recall, and F1-score (mean ± std) per classifier and embedding model for scenario (S1, I1) ↔ (S2, I2) – (80 meta-features)

Classifier	**Metric**	albert	bart	bert	distilbert	minilm	roberta	**Classifier Mean**
RandomForest	Precision	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00
	Recall	0.82 ± 0.01	0.82 ± 0.01	0.82 ± 0.01	0.82 ± 0.01	0.82 ± 0.01	0.82 ± 0.01	0.82 ± 0.01
	F1	0.90 ± 0.00	0.90 ± 0.00	0.90 ± 0.00	0.90 ± 0.00	0.90 ± 0.00	0.90 ± 0.00	0.90 ± 0.00
LogisticRegression	Precision	0.29 ± 0.00	0.29 ± 0.00	0.29 ± 0.00	0.29 ± 0.00	0.29 ± 0.00	0.29 ± 0.00	0.29 ± 0.00
	Recall	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00
	F1	0.44 ± 0.00	0.44 ± 0.00	0.44 ± 0.00	0.44 ± 0.00	0.44 ± 0.00	0.44 ± 0.00	0.44 ± 0.00
KNN	Precision	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00
	Recall	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01	0.74 ± 0.01
	F1	0.84 ± 0.01	0.84 ± 0.01	0.84 ± 0.01	0.84 ± 0.01	0.84 ± 0.01	0.84 ± 0.01	0.84 ± 0.01
MLP	Precision	0.92 ± 0.01	0.92 ± 0.01	0.92 ± 0.01	0.92 ± 0.01	0.92 ± 0.01	0.92 ± 0.01	0.92 ± 0.01
	Recall	0.88 ± 0.02	0.88 ± 0.02	0.88 ± 0.02	0.88 ± 0.02	0.88 ± 0.01	0.88 ± 0.02	0.88 ± 0.02
	F1	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.00	0.90 ± 0.01	0.90 ± 0.01
CatBoost	Precision	0.87 ± 0.02	0.87 ± 0.02	0.87 ± 0.02	0.87 ± 0.02	0.87 ± 0.03	0.87 ± 0.02	0.87 ± 0.02
	Recall	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00
	F1	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	0.92 ± 0.02	0.93 ± 0.01	0.92 ± 0.01
XGBoost	Precision	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00
	Recall	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00
	F1	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00

Table 4.58: Precision, Recall, and F1-score (mean ± std) per classifier and embedding model for scenario (S1, ∅) ↔ (S2, ∅) – (80 meta-features)

Classifier	**Metric**	albert	bart	bert	distilbert	minilm	roberta	**Classifier Mean**
RandomForest	Precision	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00
	Recall	0.77 ± 0.01	0.77 ± 0.01	0.77 ± 0.01	0.77 ± 0.01	0.77 ± 0.01	0.77 ± 0.01	0.77 ± 0.01
	F1	0.86 ± 0.00	0.86 ± 0.00	0.86 ± 0.00	0.86 ± 0.00	0.86 ± 0.00	0.86 ± 0.00	0.86 ± 0.00
LogisticRegression	Precision	0.27 ± 0.00	0.27 ± 0.00	0.27 ± 0.00	0.27 ± 0.00	0.27 ± 0.00	0.27 ± 0.00	0.27 ± 0.00
	Recall	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00
	F1	0.42 ± 0.00	0.42 ± 0.00	0.42 ± 0.00	0.42 ± 0.00	0.42 ± 0.00	0.42 ± 0.00	0.42 ± 0.00
KNN	Precision	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01
	Recall	0.57 ± 0.01	0.57 ± 0.01	0.57 ± 0.01	0.57 ± 0.01	0.57 ± 0.01	0.57 ± 0.01	0.57 ± 0.01
	F1	0.71 ± 0.01	0.71 ± 0.01	0.71 ± 0.01	0.71 ± 0.01	0.71 ± 0.01	0.71 ± 0.01	0.71 ± 0.01
MLP	Precision	0.87 ± 0.01	0.87 ± 0.01	0.87 ± 0.01	0.87 ± 0.01	0.86 ± 0.02	0.87 ± 0.01	0.87 ± 0.02
	Recall	0.84 ± 0.04	0.84 ± 0.04	0.84 ± 0.04	0.84 ± 0.04	0.86 ± 0.03	0.84 ± 0.04	0.84 ± 0.04
	F1	0.85 ± 0.02	0.85 ± 0.02	0.85 ± 0.02	0.85 ± 0.02	0.86 ± 0.01	0.85 ± 0.02	0.86 ± 0.02
CatBoost	Precision	0.82 ± 0.03	0.82 ± 0.03	0.82 ± 0.03	0.82 ± 0.03	0.82 ± 0.03	0.82 ± 0.03	0.82 ± 0.03
	Recall	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00
	F1	0.90 ± 0.02	0.90 ± 0.02	0.90 ± 0.02	0.90 ± 0.02	0.90 ± 0.02	0.90 ± 0.02	0.90 ± 0.02
XGBoost	Precision	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00	0.93 ± 0.00
	Recall	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00
	F1	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00

Table 4.59: Precision, Recall, and F1-score (mean ± std) per classifier and embedding model for scenario (∅, I1) ↔ (∅, I2) – (80 meta-features)

Classifier	**Metric**	albert	bart	bert	distilbert	minilm	roberta	**Classifier Mean**
RandomForest	Precision	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01
	Recall	0.28 ± 0.00	0.28 ± 0.00	0.28 ± 0.00	0.28 ± 0.00	0.28 ± 0.00	0.28 ± 0.00	0.28 ± 0.00
	F1	0.42 ± 0.01	0.42 ± 0.01	0.42 ± 0.01	0.42 ± 0.01	0.42 ± 0.01	0.42 ± 0.01	0.42 ± 0.01
LogisticRegression	Precision	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00
	Recall	0.82 ± 0.01	0.82 ± 0.01	0.82 ± 0.01	0.82 ± 0.01	0.82 ± 0.01	0.82 ± 0.01	0.82 ± 0.01
	F1	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00
KNN	Precision	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01
	Recall	0.32 ± 0.01	0.32 ± 0.01	0.32 ± 0.01	0.32 ± 0.01	0.32 ± 0.01	0.32 ± 0.01	0.32 ± 0.01
	F1	0.46 ± 0.01	0.46 ± 0.01	0.46 ± 0.01	0.46 ± 0.01	0.46 ± 0.01	0.46 ± 0.01	0.46 ± 0.01
MLP	Precision	0.66 ± 0.05	0.66 ± 0.05	0.66 ± 0.05	0.66 ± 0.05	0.67 ± 0.04	0.66 ± 0.05	0.66 ± 0.05
	Recall	0.45 ± 0.02	0.45 ± 0.02	0.45 ± 0.02	0.45 ± 0.02	0.44 ± 0.02	0.45 ± 0.02	0.45 ± 0.02
	F1	0.53 ± 0.01	0.53 ± 0.01	0.53 ± 0.01	0.53 ± 0.01	0.53 ± 0.01	0.53 ± 0.01	0.53 ± 0.01
CatBoost	Precision	0.31 ± 0.01	0.31 ± 0.01	0.31 ± 0.01	0.31 ± 0.01	0.31 ± 0.01	0.31 ± 0.01	0.31 ± 0.01
	Recall	0.73 ± 0.01	0.73 ± 0.01	0.73 ± 0.01	0.73 ± 0.01	0.73 ± 0.01	0.73 ± 0.01	0.73 ± 0.01
	F1	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01
XGBoost	Precision	0.52 ± 0.01	0.52 ± 0.01	0.52 ± 0.01	0.52 ± 0.01	0.52 ± 0.01	0.52 ± 0.01	0.52 ± 0.01
	Recall	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01
	F1	0.56 ± 0.01	0.56 ± 0.01	0.56 ± 0.01	0.56 ± 0.01	0.56 ± 0.01	0.56 ± 0.01	0.56 ± 0.01

Table 4.60: Precision, Recall, and F1-score (mean ± std) per classifier and embedding model for scenario (S1, I1) ↔ (S2, ∅) – (80 meta-features)

Classifier	**Metric**	albert	bart	bert	distilbert	minilm	roberta	**Classifier Mean**
RandomForest	Precision	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.98 ± 0.00	0.99 ± 0.00	0.98 ± 0.00
	Recall	0.79 ± 0.01	0.79 ± 0.01	0.79 ± 0.01	0.79 ± 0.01	0.79 ± 0.01	0.79 ± 0.01	0.79 ± 0.01
	F1	0.87 ± 0.00	0.87 ± 0.00	0.87 ± 0.00	0.87 ± 0.00	0.87 ± 0.00	0.87 ± 0.00	0.87 ± 0.00
LogisticRegression	Precision	0.28 ± 0.00	0.28 ± 0.00	0.28 ± 0.00	0.28 ± 0.00	0.28 ± 0.00	0.28 ± 0.00	0.28 ± 0.00
	Recall	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00
	F1	0.43 ± 0.00	0.43 ± 0.00	0.43 ± 0.00	0.43 ± 0.00	0.43 ± 0.00	0.43 ± 0.00	0.43 ± 0.00
KNN	Precision	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01	0.95 ± 0.01
	Recall	0.57 ± 0.01	0.57 ± 0.01	0.57 ± 0.01	0.57 ± 0.01	0.57 ± 0.01	0.57 ± 0.01	0.57 ± 0.01
	F1	0.71 ± 0.01	0.71 ± 0.01	0.71 ± 0.01	0.71 ± 0.01	0.71 ± 0.01	0.71 ± 0.01	0.71 ± 0.01
MLP	Precision	0.88 ± 0.01	0.88 ± 0.01	0.88 ± 0.01	0.88 ± 0.01	0.88 ± 0.01	0.88 ± 0.01	0.88 ± 0.01
	Recall	0.84 ± 0.03	0.84 ± 0.03	0.84 ± 0.03	0.84 ± 0.03	0.85 ± 0.03	0.84 ± 0.03	0.84 ± 0.03
	F1	0.86 ± 0.01	0.86 ± 0.01	0.86 ± 0.01	0.86 ± 0.01	0.86 ± 0.01	0.86 ± 0.01	0.86 ± 0.01
CatBoost	Precision	0.84 ± 0.02	0.84 ± 0.02	0.84 ± 0.02	0.84 ± 0.02	0.84 ± 0.02	0.84 ± 0.02	0.84 ± 0.02
	Recall	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00
	F1	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.91 ± 0.01
XGBoost	Precision	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01	0.93 ± 0.01
	Recall	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00
	F1	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00	0.95 ± 0.00

Table 4.61: Precision, Recall, and F1-score (mean ± std) per classifier and embedding model for scenario (S1, I1) ↔ (∅, I2) – (80 meta-features)

Classifier	**Metric**	albert	bart	bert	distilbert	minilm	roberta	**Classifier Mean**
RandomForest	Precision	0.96 ± 0.00	0.96 ± 0.00	0.96 ± 0.00	0.95 ± 0.00	0.96 ± 0.01	0.96 ± 0.00	0.96 ± 0.00
	Recall	0.42 ± 0.01	0.42 ± 0.01	0.42 ± 0.01	0.42 ± 0.01	0.42 ± 0.01	0.42 ± 0.01	0.42 ± 0.01
	F1	0.58 ± 0.01	0.58 ± 0.01	0.58 ± 0.01	0.59 ± 0.01	0.58 ± 0.01	0.58 ± 0.01	0.58 ± 0.01
LogisticRegression	Precision	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00
	Recall	0.82 ± 0.01	0.82 ± 0.01	0.82 ± 0.01	0.83 ± 0.01	0.82 ± 0.01	0.82 ± 0.01	0.82 ± 0.01
	F1	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00
KNN	Precision	0.88 ± 0.01	0.88 ± 0.01	0.88 ± 0.01	0.88 ± 0.01	0.88 ± 0.01	0.88 ± 0.01	0.88 ± 0.01
	Recall	0.46 ± 0.01	0.46 ± 0.01	0.46 ± 0.01	0.46 ± 0.01	0.46 ± 0.01	0.46 ± 0.01	0.46 ± 0.01
	F1	0.60 ± 0.01	0.60 ± 0.01	0.60 ± 0.01	0.60 ± 0.01	0.60 ± 0.01	0.60 ± 0.01	0.60 ± 0.01
MLP	Precision	0.73 ± 0.06	0.73 ± 0.06	0.73 ± 0.06	0.68 ± 0.04	0.71 ± 0.05	0.73 ± 0.06	0.72 ± 0.05
	Recall	0.53 ± 0.02	0.53 ± 0.02	0.53 ± 0.02	0.55 ± 0.02	0.54 ± 0.03	0.53 ± 0.02	0.53 ± 0.02
	F1	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01	0.61 ± 0.01
CatBoost	Precision	0.40 ± 0.01	0.40 ± 0.01	0.40 ± 0.01	0.40 ± 0.01	0.40 ± 0.01	0.40 ± 0.01	0.40 ± 0.01
	Recall	0.77 ± 0.01	0.77 ± 0.01	0.77 ± 0.01	0.78 ± 0.01	0.77 ± 0.01	0.77 ± 0.01	0.77 ± 0.01
	F1	0.53 ± 0.01	0.53 ± 0.01	0.53 ± 0.01	0.52 ± 0.01	0.53 ± 0.01	0.53 ± 0.01	0.53 ± 0.01
XGBoost	Precision	0.63 ± 0.01	0.63 ± 0.01	0.63 ± 0.01	0.62 ± 0.01	0.63 ± 0.01	0.63 ± 0.01	0.62 ± 0.01
	Recall	0.67 ± 0.01	0.67 ± 0.01	0.67 ± 0.01	0.68 ± 0.01	0.67 ± 0.01	0.67 ± 0.01	0.67 ± 0.01
	F1	0.65 ± 0.01	0.65 ± 0.01	0.65 ± 0.01	0.65 ± 0.01	0.65 ± 0.01	0.65 ± 0.01	0.65 ± 0.01

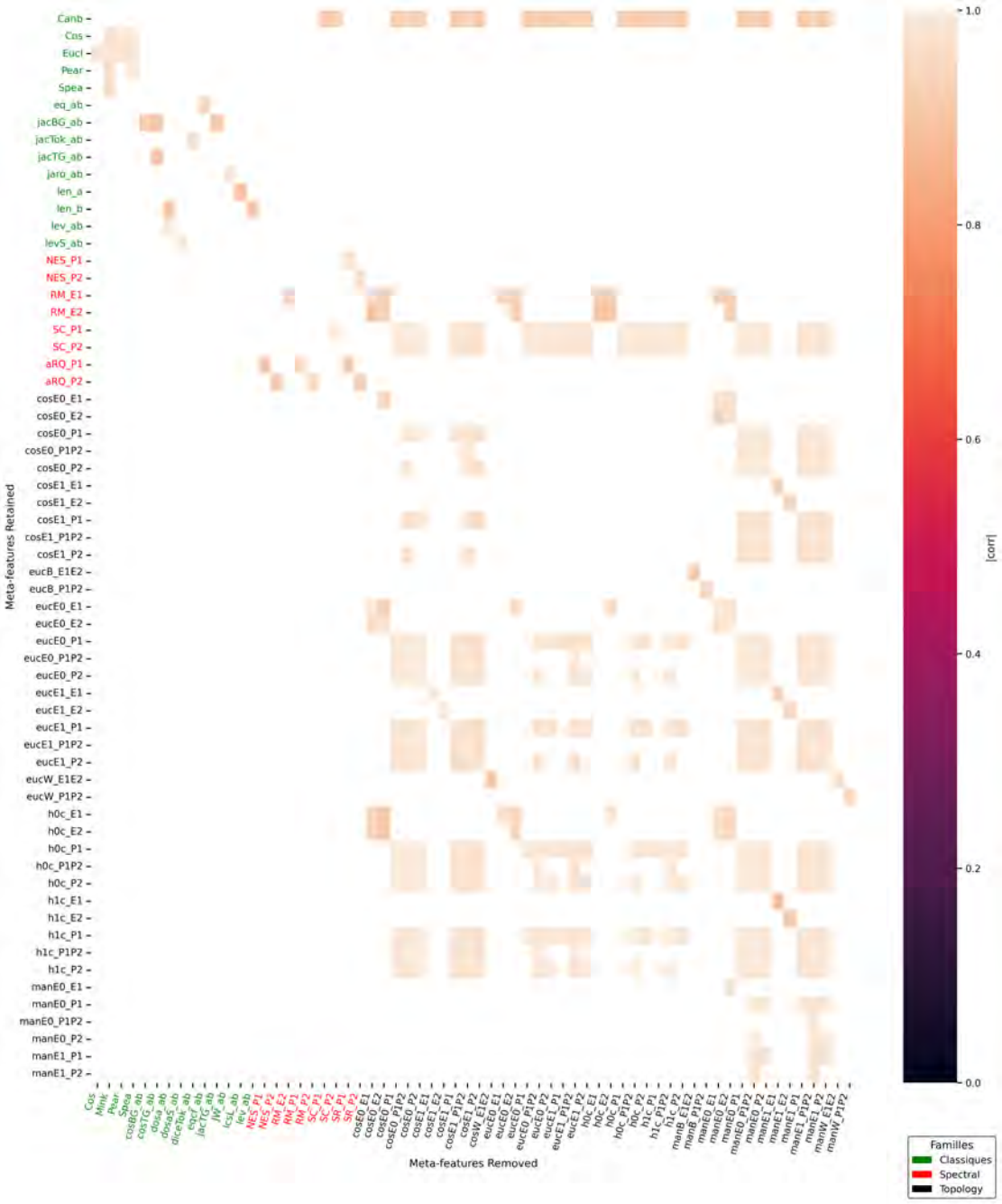


Figure 4.5: Heatmap of pairwise correlations ($\geq$ 0.9) between retained and removed meta-features.

We start from 112 meta-features in three families: Classical, Spectral, and Topological. Before selection, many meta-features were highly redundant, with correlations often exceeding 0.9. We apply backward stepwise regression independently on each dataset pair, repeating the process 10

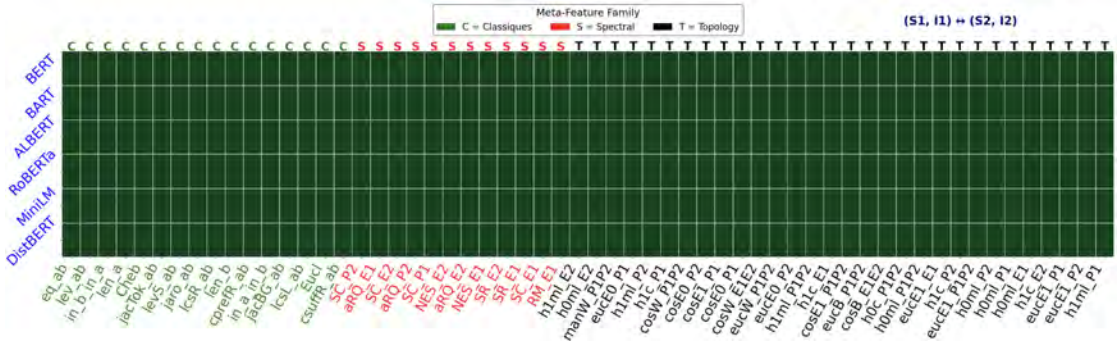


Figure 4.6: Feature selection results – Scenario (S1, I1) ↔ (S2, I2).

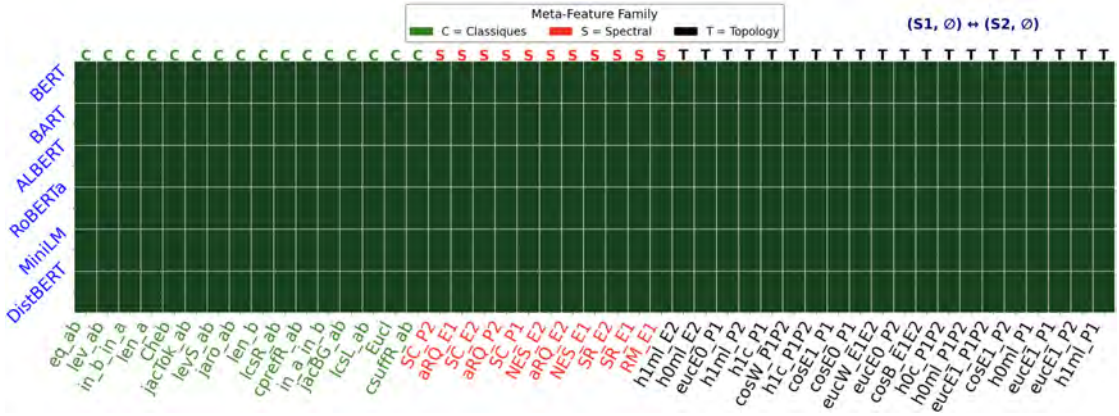


Figure 4.7: Feature selection results – Scenario (S1, ∅) ↔ (S2, ∅).

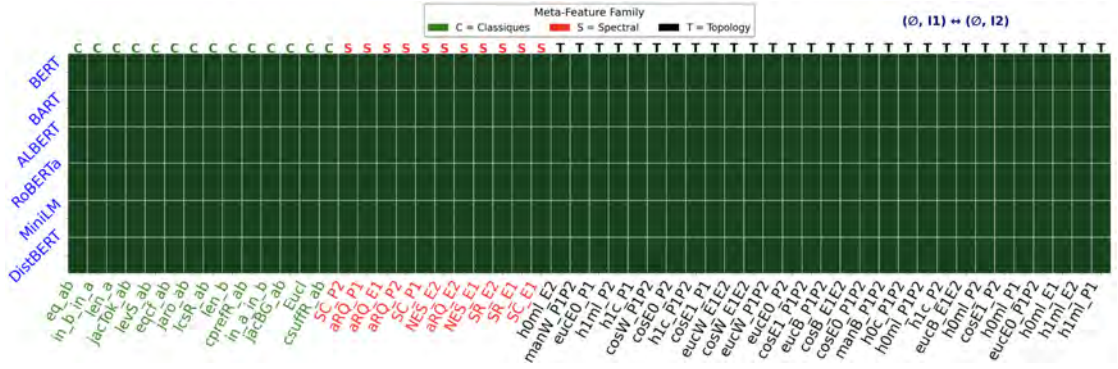


Figure 4.8: Feature selection results – Scenario (∅, I1) ↔ (∅, I2).

times.

Figure 4.5 shows the averaged high-correlation structure (≥ 0.9) among retained features across scenarios and models. Classical features display the expected clustering of string-based similarities; Spectral features (e.g., NESum variants) are highly self-correlated, reflecting sensitivity to global embedding stability. Topological features exhibit very strong internal redun-

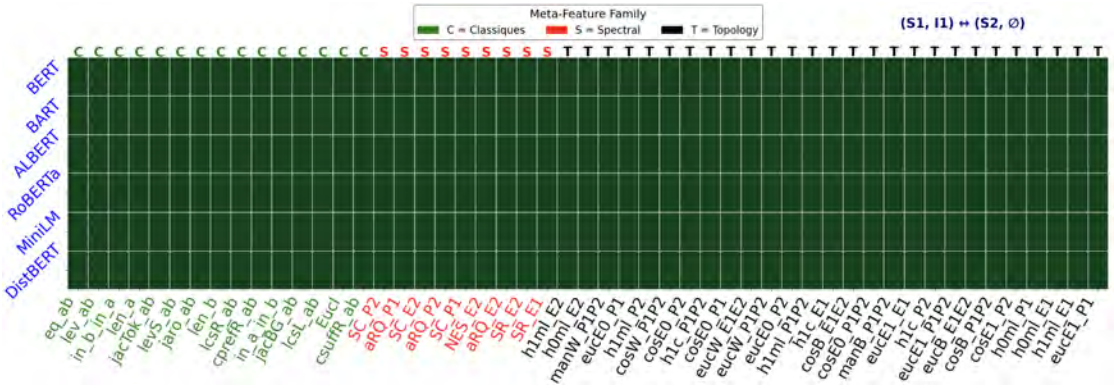


Figure 4.9: Feature selection results – Scenario (S1, I1) ↔ (S2, ∅).

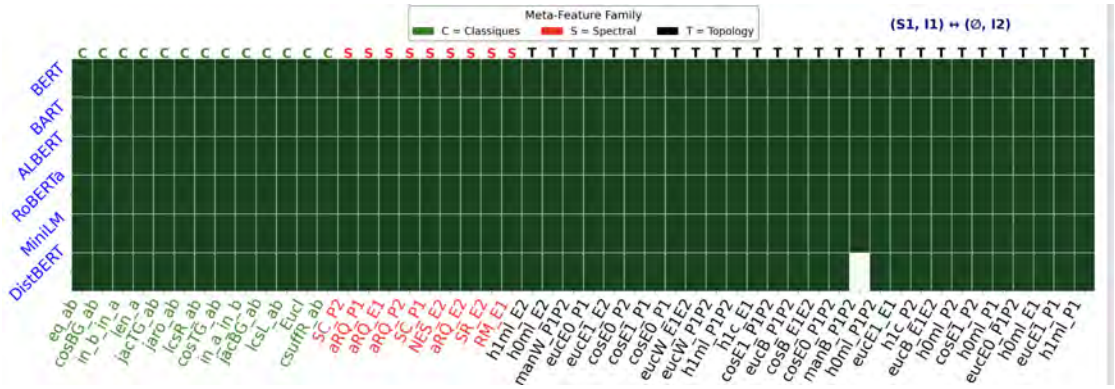


Figure 4.10: Feature selection results – Scenario (S1, I1) ↔ (∅, I2) (schema-only → instance-only).

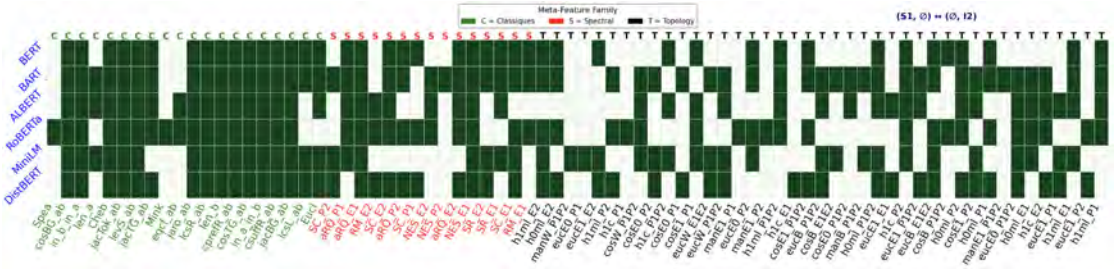


Figure 4.11: Feature selection results – Scenario (S1, ∅) ↔ (∅, I2).

dancy, especially among H_0 counts and persistence entropy: for instance, `h0_count_a1` and `h0_count_a2` correlate almost perfectly ($r \approx 1.0$), and entropies with different ground metrics (Euclidean vs. Manhattan on H_0) exceed 0.999. Some Topological invariants also overlap with Spectral descriptors such as NESum, indicating related aspects of geometry and stability.

Figures 4.6–4.11 summarize retained subsets across scenarios and embedding models. In schema-available settings (C1, C2, D4), roughly half the features are pruned (typically 43–55), with most retained in the Classical family. In instance-available settings (C3, D5, D6), more

Table 4.62: Precision, Recall, and F1-score (mean ± std) per classifier and embedding model for scenario (S1, ∅) ↔ (∅, I2) – (80 meta-features)

Classifier	Metric	albert	bart	bert	distilbert	minilm	roberta	Classifier Mean
RandomForest	Precision	0.98 ± 0.01	0.98 ± 0.01	0.98 ± 0.00	0.98 ± 0.01	0.98 ± 0.00	0.98 ± 0.01	0.98 ± 0.00
	Recall	0.24 ± 0.01	0.26 ± 0.01	0.26 ± 0.01	0.29 ± 0.01	0.17 ± 0.01	0.28 ± 0.01	0.25 ± 0.01
	F1	0.39 ± 0.01	0.41 ± 0.01	0.42 ± 0.01	0.45 ± 0.01	0.29 ± 0.01	0.43 ± 0.01	0.40 ± 0.01
LogisticRegression	Precision	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00
	Recall	0.65 ± 0.01	0.65 ± 0.01	0.64 ± 0.01	0.64 ± 0.01	0.60 ± 0.01	0.65 ± 0.01	0.64 ± 0.01
	F1	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.00
KNN	Precision	0.84 ± 0.01	0.86 ± 0.01	0.86 ± 0.01	0.83 ± 0.02	0.81 ± 0.01	0.85 ± 0.01	0.84 ± 0.01
	Recall	0.26 ± 0.01	0.34 ± 0.01	0.32 ± 0.01	0.33 ± 0.01	0.26 ± 0.01	0.32 ± 0.01	0.30 ± 0.01
	F1	0.40 ± 0.01	0.48 ± 0.01	0.47 ± 0.01	0.47 ± 0.01	0.39 ± 0.01	0.46 ± 0.01	0.45 ± 0.01
MLP	Precision	0.62 ± 0.06	0.65 ± 0.05	0.67 ± 0.04	0.67 ± 0.07	0.61 ± 0.05	0.64 ± 0.05	0.64 ± 0.05
	Recall	0.32 ± 0.03	0.36 ± 0.02	0.35 ± 0.02	0.37 ± 0.02	0.25 ± 0.02	0.35 ± 0.04	0.33 ± 0.02
	F1	0.42 ± 0.02	0.46 ± 0.02	0.46 ± 0.01	0.47 ± 0.01	0.35 ± 0.02	0.45 ± 0.03	0.43 ± 0.02
CatBoost	Precision	0.32 ± 0.01	0.35 ± 0.01	0.35 ± 0.01	0.37 ± 0.01	0.23 ± 0.00	0.37 ± 0.01	0.33 ± 0.01
	Recall	0.69 ± 0.01	0.72 ± 0.01	0.75 ± 0.01	0.75 ± 0.01	0.59 ± 0.01	0.76 ± 0.01	0.71 ± 0.01
	F1	0.44 ± 0.01	0.47 ± 0.01	0.48 ± 0.01	0.49 ± 0.01	0.33 ± 0.00	0.50 ± 0.01	0.45 ± 0.01
XGBoost	Precision	0.64 ± 0.02	0.65 ± 0.01	0.64 ± 0.01	0.67 ± 0.01	0.59 ± 0.02	0.64 ± 0.02	0.64 ± 0.01
	Recall	0.49 ± 0.01	0.51 ± 0.01	0.52 ± 0.01	0.52 ± 0.01	0.37 ± 0.01	0.55 ± 0.01	0.49 ± 0.01
	F1	0.55 ± 0.01	0.57 ± 0.01	0.58 ± 0.01	0.59 ± 0.01	0.45 ± 0.01	0.59 ± 0.01	0.55 ± 0.01

than 55 are kept, as Spectral and Topological features become necessary when schema overlap is missing. Across all scenarios and embeddings, the global union of retained features totals 80 unique meta-features; about 32 ($\sim 29\%$) are never selected.

Effectiveness with 80 selected features. Tables 4.57–4.62 report precision, recall, and F1 across classifiers and embeddings for the six scenarios using the 80 selected features. In schema-available cases (C1, C2), boosted models dominate: XGBoost approaches ceiling (F1 $\approx$ 0.95–0.97), CatBoost is close (F1 $\approx$ 0.90), and MLP/Random Forest are competitive (0.86–0.90). KNN remains acceptable (0.71–0.84), while Logistic Regression collapses due to high recall ($\approx$ 0.95) and very low precision ($\approx$ 0.27–0.29), yielding F1 barely above 0.40.

Instance-only alignment (C3) is the most challenging: F1 ranges 0.20–0.56 depending on the classifier; XGBoost reaches 0.56, MLP 0.53, RF/CatBoost/KNN mid-0.40s, and Logistic Regression 0.20. Despite high precision in some models (e.g., RF 0.90, KNN 0.85), recall remains low (0.28–0.32), reflecting the difficulty without schema anchors—underscoring the need for Spectral and Topological cues when Classical similarities are insufficient.

Asymmetric scenarios show the same trend. In D4 (schema+instances vs. schema-only), performance is close to schema-available: XGBoost 0.95, CatBoost 0.91, MLP 0.86, RF 0.87; KNN drops to 0.71, Logistic Regression to 0.43. D5 (schema+instances vs. instance-only) is lower but workable: XGBoost 0.65, MLP/KNN $\approx$ 0.60–0.61, RF/CatBoost 0.53–0.58, Logistic Regression 0.19. D6 (schema-only vs. instance-only) is most fragile: XGBoost $\approx$ 0.55, others < 0.45, Logistic Regression $\approx$ 0.10. These results indicate reduced robustness under asymmetry, with Spectral and Topological features helping prevent collapse. Importantly, these scores match those with the full 112-feature space, showing that the 80-feature space preserves effectiveness while improving the efficiency for about 30%.

Tables 4.51–4.56 report precision, recall, and F1 (mean ± std) for all classifiers and embeddings in every scenario; they are obtained from meta-feature spaces selected per embedding and per scenario. Globally, we recover the same effectiveness as with the meta-space (80 and full 112 meta-features): rankings of classifiers and the relative difficulty of scenarios are unchanged, with no meaningful degradation in performance.

Table 4.63: Average Precision, Recall, and F1-score (± std) per baseline for scenario (S1, I1) ↔ (S2, I2).

Method	Precision	Recall	F1-score
Distribution	0.29 ± 0.29	0.23 ± 0.28	0.23 ± 0.25
Cupid	0.46 ± 0.40	0.37 ± 0.42	0.34 ± 0.37
ISResMat	0.44 ± 0.26	0.71 ± 0.25	0.50 ± 0.25
Coma	0.74 ± 0.33	0.48 ± 0.30	0.51 ± 0.25
SimilarityFlooding	0.49 ± 0.35	0.69 ± 0.38	0.53 ± 0.35
Coma++	0.79 ± 0.32	0.56 ± 0.27	0.59 ± 0.23
Magneto	0.50 ± 0.31	0.90 ± 0.13	0.58 ± 0.29
MagnetoFT	0.51 ± 0.31	0.90 ± 0.15	0.59 ± 0.30
MagnetoGPT	0.69 ± 0.34	0.89 ± 0.20	0.72 ± 0.30
MagnetoFTGPT	0.68 ± 0.34	0.89 ± 0.21	0.72 ± 0.31
MetaSpace-80	$\mathbf{0.96 \pm 0.00}$	$\mathbf{0.97 \pm 0.00}$	$\mathbf{0.97 \pm 0.00}$

Discussion. The selection analysis confirms the central role of Classical meta-features. String-based similarities such as Jaro, Levenshtein, LCS, Jaccard, cosine n-grams, and prefix/suffix checks are consistently retained, especially in schema-available settings. Spectral and Topological meta-features are less frequently selected when schemas are available, but they are crucial in instance-only scenarios (C3, D5, D6), where they stabilize alignment in the absence of schema information. Taken together, the retained subsets show that while Classical meta-features form the backbone of the framework, Spectral and Topological ones provide robustness in harder cases.

The global union of 80 features can be viewed as a candidate reduced meta-space: a portable set that covers all scenarios without needing the full 112 features. This compact design cuts nearly half of the original feature space while keeping effectiveness intact, and demonstrates how redundancy reduction can improve efficiency without sacrificing stability. We reduced the meta-space to 80 meta-features. The 32 removed are not necessarily useless: being highly correlated with retained ones, they could re-appear in alternative, partially overlapping subsets. That said, pruning by a fixed correlation threshold ($|\rho| > 0.9$) is debatable—some discarded meta-features might add value through non-linear interactions and yield an even more compact or better-performing set. Likewise, our regression-based selection is linear and may under-capture the non-linear structure we observed (e.g., where Logistic Regression underperforms). These choices likely limit optimality; exploring alternative, non-linear selection strategies and subset searches is a promising direction for future work.

Overall, the findings show that the three families remain complementary, but a carefully pruned subset of meta-features is sufficient for robust performance across all scenarios.

RQ5. Baseline

The objective of this research question is to determine how our framework compares to established benchmarks in supervised pattern matching by comparing F1 scores, precision, and recall across all scenarios.
To ensure a fair comparison, all baselines receive identical inputs as our method per scenario. Results are averaged over all Valentine datasets and reported as mean ± std.

We compare our framework to established baselines— Cupid, Coma/Coma++, distribution-based, Magneto (and variants), Similarity Flooding, and ISResMat. Tables 4.63–4.68 report Precision, Recall, and F1 for the six scenarios.

Table 4.64: Average Precision, Recall, and F1-score (± std) per baseline for scenario (S1, ∅) ↔ (S2, ∅).

Method	Precision	Recall	F1-score
Distribution	0.12 ± 0.26	0.09 ± 0.23	0.09 ± 0.22
Cupid	0.41 ± 0.39	0.33 ± 0.40	0.31 ± 0.36
ISResMat	0.47 ± 0.26	0.67 ± 0.25	0.51 ± 0.25
Coma	0.73 ± 0.34	0.47 ± 0.31	0.51 ± 0.27
SimilarityFlooding	0.48 ± 0.36	0.64 ± 0.41	0.50 ± 0.36
Coma++	0.63 ± 0.37	0.48 ± 0.34	0.47 ± 0.29
Magneto	0.53 ± 0.29	0.84 ± 0.19	0.60 ± 0.28
MagnetoFT	0.53 ± 0.29	0.84 ± 0.20	0.60 ± 0.28
MagnetoGPT	0.53 ± 0.29	0.84 ± 0.20	0.60 ± 0.28
MagnetoFTGPT	0.53 ± 0.29	0.84 ± 0.20	0.60 ± 0.28
MetaSpace-80	$\mathbf{0.93 \pm 0.01}$	$\mathbf{0.97 \pm 0.00}$	$\mathbf{0.95 \pm 0.00}$

Table 4.65: Average Precision, Recall, and F1-score (± std) per baseline for scenario (∅, I1) ↔ (∅, I2).

Method	Precision	Recall	F1-score
Distribution	0.56 ± 0.35	0.44 ± 0.32	0.45 ± 0.29
Cupid	0.52 ± 0.36	0.61 ± 0.26	0.49 ± 0.27
ISResMat	0.47 ± 0.26	0.67 ± 0.25	0.51 ± 0.25
Coma	0.40 ± 0.28	0.28 ± 0.30	0.27 ± 0.20
SimilarityFlooding	0.18 ± 0.17	0.26 ± 0.27	0.19 ± 0.17
Coma++	0.55 ± 0.31	0.37 ± 0.29	0.38 ± 0.22
Magneto	0.53 ± 0.29	0.84 ± 0.20	0.60 ± 0.28
MagnetoFT	0.53 ± 0.29	0.83 ± 0.20	0.60 ± 0.27
MagnetoGPT	0.53 ± 0.29	$\mathbf{0.84 \pm 0.19}$	$\mathbf{0.60 \pm 0.27}$
MagnetoFTGPT	0.53 ± 0.29	0.83 ± 0.20	0.60 ± 0.28
MetaSpace-80	$\mathbf{0.53 \pm 0.01}$	0.61 ± 0.01	0.57 ± 0.01

Table 4.66: Average Precision, Recall, and F1-score (± std) per baseline for scenario (S1, I1) ↔ (S2, ∅).

Method	Precision	Recall	F1-score
Distribution	0.04 ± 0.13	0.01 ± 0.05	0.02 ± 0.06
Cupid	0.37 ± 0.39	0.22 ± 0.31	0.24 ± 0.30
ISResMat	0.47 ± 0.26	0.66 ± 0.25	0.52 ± 0.25
Coma	0.72 ± 0.35	0.47 ± 0.31	0.50 ± 0.27
SimilarityFlooding	0.46 ± 0.36	0.64 ± 0.41	0.48 ± 0.36
Coma++	0.64 ± 0.36	0.42 ± 0.32	0.44 ± 0.28
Magneto	0.51 ± 0.30	0.85 ± 0.19	0.58 ± 0.29
MagnetoFT	0.51 ± 0.30	0.85 ± 0.19	0.58 ± 0.29
MagnetoGPT	0.51 ± 0.30	0.85 ± 0.19	0.58 ± 0.29
MagnetoFTGPT	0.51 ± 0.30	0.85 ± 0.19	0.58 ± 0.29
MetaSpace-80	$\mathbf{0.93 \pm 0.01}$	$\mathbf{0.97 \pm 0.01}$	$\mathbf{0.95 \pm 0.00}$

Table 4.67: Average Precision, Recall, and F1-score (± std) per baseline for scenario (S1, I1) ↔ (∅, I2).

Method	Precision	Recall	F1-score
Distribution	0.38 ± 0.30	0.30 ± 0.30	0.30 ± 0.25
Cupid	0.30 ± 0.21	0.29 ± 0.32	0.23 ± 0.15
ISResMat	0.48 ± 0.26	0.67 ± 0.24	0.52 ± 0.24
Coma	0.34 ± 0.26	0.20 ± 0.28	0.19 ± 0.17
SimilarityFlooding	0.12 ± 0.12	0.18 ± 0.24	0.13 ± 0.12
Coma++	0.51 ± 0.31	0.32 ± 0.29	0.34 ± 0.22
Magneto	0.53 ± 0.29	0.84 ± 0.19	0.60 ± 0.27
MagnetoFT	0.52 ± 0.29	0.83 ± 0.20	0.59 ± 0.28
MagnetoGPT	0.53 ± 0.29	0.84 ± 0.19	0.61 ± 0.27
MagnetoFTGPT	0.53 ± 0.29	0.83 ± 0.20	0.60 ± 0.28
MetaSpace-80	$\mathbf{0.63 \pm 0.02}$	$\mathbf{0.67 \pm 0.01}$	$\mathbf{0.65 \pm 0.01}$

Table 4.68: Average Precision, Recall, and F1-score (± std) per baseline for scenario (S1, ∅) ↔ (∅, I2).

Method	Precision	Recall	F1-score
Distribution	0.04 ± 0.13	0.01 ± 0.04	0.02 ± 0.06
Cupid	0.23 ± 0.22	0.17 ± 0.25	0.16 ± 0.16
ISResMat	0.47 ± 0.26	0.67 ± 0.24	0.51 ± 0.25
Coma	0.35 ± 0.27	0.20 ± 0.28	0.20 ± 0.18
SimilarityFlooding	0.12 ± 0.12	0.18 ± 0.26	0.12 ± 0.12
Coma++	0.31 ± 0.27	0.18 ± 0.25	0.18 ± 0.18
Magneto	0.52 ± 0.29	$\mathbf{0.84 \pm 0.19}$	0.59 ± 0.28
MagnetoFT	0.53 ± 0.29	0.83 ± 0.20	0.60 ± 0.28
MagnetoGPT	0.52 ± 0.29	0.84 ± 0.20	0.60 ± 0.28
MagnetoFTGPT	0.53 ± 0.29	0.84 ± 0.20	0.60 ± 0.28
MetaSpace-80	$\mathbf{0.68 \pm 0.03}$	0.59 ± 0.03	$\mathbf{0.64 \pm 0.03}$

Across the six scenarios, METASPACE-80 leads in five out of six, with clear margins over the best MAGNETO variant (retriever–reranker configurations combining zero-shot/fine-tuned SLM-Small Language Model- retrievers with either a bipartite or LLM reranker): C1: +0.25 (0.97 vs 0.72, Table 4.63); C2: +0.35 (0.95 vs ∼0.60, Table 4.64); D4: +0.37 (0.95 vs ∼0.58, Table 4.66); D5: +0.04 (0.65 vs ∼0.61, Table 4.67); D6: +0.04 (0.64 vs ∼0.60, Table 4.68). The lone exception is C3, where MAGNETO variants slightly outperform METASPACE-80 (–0.03; 0.57 vs ∼0.60, Table 4.65). Beyond average F1, METASPACE-80 is markedly more stable (std ≈ 0–0.03) than MAGNETO (std ≈ 0.27–0.30), making it the more reliable choice overall.

Discussion. MAGNETO variants primarily exploit instance-level information via an SLM retriever and, for some variants, an LLM reranker. In our results, LLM-reranked MAGNETO (*Magneto–GPT*) is competitive in C1 (F1 = 0.72), takes the lead in C3 (F1 = 0.60), and otherwise remains around 0.58–0.61 in C2 and D4–D6—i.e., a marked peak when the schema and instances are available, followed by consistent results in other scenarios. Its higher variability (std ≈ 0.28) suggests sensitivity to dataset shifts. ISRESMAT is stable at F1 ≈ 0.51, COMA/COMA++ (labels/constraints) are competitive in C1 (F1 = 0.51/0.59) but deteriorate when schema information is missing (C1→D6: 0.51 → 0.20, 0.59 → 0.18). CUPID and the DISTRIBUTION baseline underperform overall, yet benefit from instances (ΔF1 C1→C3 = +0.15 and +0.22). SIMILARITY FLOODING performs well on C1 (= 0.53) but drops in instance-only or asymmetric settings (C3 = 0.19, D6 = 0.12).
These results indicate that a supervised meta-space offers the best effectiveness–stability trade-off overall, making METASPACE-80 the preferred matching method across Valentine scenarios (+34.7% in C1, +58.3% in C2, +63.8% in D4, +6.6% in D5, +6.7% in D6; −5.0% in C3) versus the best MAGNETO variant, with markedly lower variability (std ≈ 0–0.03 vs. 0.27–0.30, about 90% lower).

4.3.3 Conclusion

This section assessed the supervised pipeline, where schema-only and instance-only sources are enriched with LLMs and then projected into a meta-space that combines *classical*, *spectral*, and *topological* meta-features. These meta-features jointly capture semantic, syntactic, and contextual aspects of the meta-space. Together, they provide a geometric characterization through which attribute correspondences can be learned more robustly than with classical meta-features alone.

The experimental results highlight both the strengths and boundaries of this approach. In scenarios where schema information is available on both sides (C1 and C2), boosted learners, and in particular XGBoost, reach near-ceiling performance (F1 ≈ 0.95–0.97) with balanced precision and recall across heterogeneous families such as *Magellan*, *ChEMBL*, and *TPC-DI*. Even without embedding fine-tuning, the meta-learner captures non-linear interactions between meta-features. As schema information diminishes (C3 and disparate scenarios), performance declines (best F1 ≈ 0.57–0.65), reflecting the difficulty of relying only on instances under domain shift, most noticeably in highly heterogeneous datasets like *OpenData* and *Wikidata*. Still, the contribution of spectral and topological meta-features becomes particularly evident in these settings: they stabilize alignment when classical meta-features are weakened, preventing collapse in effectiveness.

Beyond raw results, the analysis of meta-feature selection confirms the discriminative value of the meta-space. From an initial set of 112 meta-features, backward stepwise selection consistently prunes 54–69 per scenario, yielding compact subsets of about 80 meta-features that remain effective across scenarios. Classical meta-features dominate when schemas are present, but spec-

tral and topological meta-features play a central role in schema-sparse or instance-only settings, offering a safeguard against distributional variability. In short, the supervised pipeline embodies a logic of refinement through learning: it leverages annotations to train a meta-learner that transforms meta-features into precise alignment predictions. At the same time, it underscores the need for further work on meta-feature efficiency, enrichment quality control, and domain adaptation, especially for diverse data domains.

4.4 Conclusion

This chapter presented an extensive evaluation of the proposed framework across six scenarios and five heterogeneous dataset categories, demonstrating how enrichment, embeddings, and alignment can be orchestrated into two complementary pipelines. The unsupervised pipeline has shown that schema-only and instance-only alignment, one of the most challenging cases in data integration, can be approached effectively by combining LLM-enrichment with a set of schema and instance matchers. Ablation studies revealed that not all matchers contribute equally, but the integration of type, rule, schema, instance, and schema–instance matchers provides robustness across noisy and heterogeneous conditions. In the Valentine benchmark, GPT-3.5 proved the most effective for enrichment, and GPT embeddings achieved the best mean F1 and NDCG, outperforming traditional string-based baselines. While F1 scores remain constrained by the inherent asymmetry of schema-only with instance-only data alignment, the results demonstrate that informed matcher composition coupled with enrichment can yield practical solutions where conventional methods fail.

The supervised pipeline, by contrast, capitalizes on the availability of labels to construct a meta-space unifying classical, spectral, and topological meta-features. This meta-space allows boosted learners to achieve near-perfect performance when schemas are present (F1 $\approx$ 0.95–0.97 in C1 and C2) and to remain competitive in schema-only or instance-only scenarios (F1 $\approx$ 0.57–0.65). Meta-feature selection confirmed the compactness and discriminative power of the meta-space, with classical meta-features dominating in schema-available cases and spectral and topological meta-features stabilizing performance under heterogeneity. These results highlight the originality of introducing a meta-space that integrates multiple families of meta-features into schema matching, offering a more holistic characterization than classical meta-features alone.

Together, the two pipelines articulate complementary logics. The unsupervised pipeline prioritizes adaptability and robustness in low-resource settings, operating without supervision yet maintaining effectiveness through matcher diversity and enrichment. The supervised pipeline, in contrast, refines alignment decisions through label-driven learning, transforming meta-features into precise alignment predictions when annotations are available. Their joint evaluation shows not only that both approaches are feasible but also that they are synergistic: unsupervised alignment extends applicability to annotation-less domains, while supervised alignment offers ceiling-level precision where annotations exist. Limitations remain—particularly around enrichment fidelity, potential LLM prior knowledge, and data confidentiality—but the framework provides a solid foundation for further advances, including privacy-preserving enrichment, adaptive weighting of matchers, and cost-aware meta-feature selection.

Chapter 5

Conclusion

5.1 Context and Problem Statement

The work presented in this thesis was motivated by a central challenge in data matching: how to align heterogeneous tabular datasets when information is incomplete or unbalanced across sources. As explained in the introduction, modern data ecosystems often produce tables that are either *schema-only*, exposing column names but no instances, or *instance-only*, exposing raw values but no schema. Such conditions occur frequently in data lakes, open government platforms, enterprise repositories, and scientific databases, and they are made more complex by differences in naming conventions, formats, and semantics.

Classical schema matching methods are not designed for these situations, since they usually rely on both schema and instance information being available. As a result, real-world matching tasks often remain unsolved when one modality is missing. The central hypothesis of this thesis was that context can be generated and completed, and that this enrichment, combined with embeddings and machine learning, can provide a robust foundation for schema alignment in both compliant and disparate settings.

5.2 Summary of Contributions and Results

To test this hypothesis, we introduced a unified framework structured into two complementary pipelines: an unsupervised pipeline, based on enrichment and matcher composition, and a supervised pipeline, based on embeddings and meta-space learning. Both pipelines follow the same principle: generate missing schema or instances with large language models, encode enriched attributes into dense embeddings, and then align sources using either matcher aggregation or a trained classifier.

Six Alignment Scenarios. As introduced in Chapter 1, schema matching tasks for tabular datasets can be classified into six scenarios, three compliant and three disparate:

- **C1:** Complete $(\mathcal{S}_1, \mathcal{I}_1) \leftrightarrow$ Complete $(\mathcal{S}_2, \mathcal{I}_2)$
- **C2:** Schema-only $(\mathcal{S}_1, \varnothing) \leftrightarrow$ Schema-only $(\mathcal{S}_2, \varnothing)$
- **C3:** Instance-only $(\varnothing, \mathcal{I}_1) \leftrightarrow$ Instance-only $(\varnothing, \mathcal{I}_2)$
- **D4:** Complete $(\mathcal{S}_1, \mathcal{I}_1) \leftrightarrow$ Schema-only $(\mathcal{S}_2, \varnothing)$

- **D5:** Complete $(\mathcal{S}_1, \mathcal{I}_1) \leftrightarrow$ Instance-only $(\varnothing, \mathcal{I}_2)$
- **D6:** Schema-only $(\mathcal{S}_1, \varnothing) \leftrightarrow$ Instance-only $(\varnothing, \mathcal{I}_2)$

Our framework was designed to address all six cases. The unsupervised pipeline focused on the hardest configuration (D6), while the supervised pipeline covered the full range (C1-C3 and D4-6).

Unsupervised pipeline. The unsupervised pipeline addressed alignment by combining multiple matchers (Type, Rule, Schema, Instance, and Schema-Instance) each producing similarity matrices, which were aggregated through a decision process. LLM-based enrichment played a central role, producing synthetic schema for instance-only sources and synthetic instances for schema-only sources. Experiments showed that this pipeline is effective and robust in heterogeneous conditions. In the Valentine benchmark, GPT-3.5 produced the most useful auxiliary information, and GPT embeddings achieved the best mean F1 and NDCG, outperforming traditional string-based methods (+34.7% in C1, +58.3% in C2, +63.8% in D4, +6.6% in D5, +6.7% in D6; −5.0% in C3) versus the best MAGNETO variant, with markedly lower variability (std $\approx$ 0–0.03 vs. 0.27–0.30, about 90% lower. Ablation studies highlighted that the Instance and Schema-Instance matchers contributed the most to the final similarity matrix, while others had smaller but still valuable roles. Although performance remains limited by the asymmetry of schema-only and instance-only inputs, the pipeline proved that alignment is possible even when conventional methods fail.

Supervised pipeline. The supervised pipeline used labels to refine alignment through a meta-learning strategy. Enriched attributes were embedded and compared through three families of meta-features: *classical*, *spectral*, and *topological*. These meta-features captured semantic, syntactic, and contextual information from the geometry of the embeddings, forming a rich meta-space for learning. A meta-learner, trained on these meta-features, predicted correspondences with high accuracy (F1 $\pm$ std: C1 0.97 ± 0.00, C2 0.95 ± 0.00, C3 0.57 ± 0.01, D4 0.95 ± 0.00, D5 0.65 ± 0.01, D6 0.64 ± 0.03).

Results confirmed the effectiveness of this framework. In schema-rich scenarios (C1 and C2), boosted learners-especially XGBoost-achieved near-ceiling results, with F1 scores between 0.95 and 0.97 across *Magellan*, *ChEMBL*, and *TPC-DI*. When schema information was missing (C3, D4-D6), performance decreased (best F1 between 0.57 and 0.65), but spectral and topological meta-features played a key role in stabilizing predictions when classical similarity weakened. Feature selection further showed that from 112 original meta-features, compact subsets of around 80 were sufficient to retain discriminative effectiveness. This confirmed the originality and value of combining classical, spectral, and topological meta-features into a meta-space.

Complementary logics. The two pipelines follow different but complementary logics. The unsupervised pipeline emphasizes adaptability and robustness in low-resource conditions, making it suitable for annotation-scarce domains. The supervised pipeline uses labels to refine alignment, transforming geometric meta-features into precise predictions with ceiling-level performance. Together, they provide a flexible methodology capable of addressing both the most difficult disparate scenario (D6) and the most favorable compliant scenario (C1).

5.3 Limitations and Future Work

Short-term Future Work

We will systematically study the two prompts (supervised and unsupervised) and quantify their impact by defining and applying quality metrics for generated data that capture semantics, syntax, context, and distance from the original data [95]. For the embeddings phase, we will move beyond zero-shot approaches to explore fine-tuning—especially for Wikidata and Open Data categories across diverse scenarios—and strengthen this stage on which both frameworks rely.

In the unsupervised framework leveraging matchers, to reduce bias and noise while preserving information from specialized matchers, we will replace uniform weighting with data- and matcher-aware weights that reflect each component's actual contribution. In the supervised framework leveraging Meta-space, we will revisit our handling of correlated meta-features, incorporate execution time into feature selection to favor cheaper meta-features, and adopt selection methods aligned with non-linear classifiers, given the poor performance of linear regression.

Finally, to mitigate potential leakage from LLM training, we will evaluate on private datasets that LLMs have never seen.

Long-term Future Work

We will extend alignment from two-source (disparate or compliant) settings to multi-source inputs ("holistic alignment"), explicitly including both scenarios within this framework [102]. Beyond tabular data, we will extend alignment to other data types such as ontologies and JSON Schemas. As longer-term work, we will tackle matching across two document collections (e.g., MongoDB) where each instance may have its own structure. The objective is to align both the collections and their sub-schemas—a more challenging case that couples schema and instance-level variability.

5.4 Final Remarks

This thesis proposed a framework that advances schema matching under disparity and incompleteness. By combining LLM-based enrichment, embedding-based representation, and two complementary alignment steps, it demonstrated that both robustness and precision are possible under heterogeneous conditions. The unsupervised pipeline showed that alignment can be achieved without labels, while the supervised pipeline achieved very high precision when labels were available, due to a meta-space that combines classical, spectral, and topological meta-features.

At the same time, the work highlighted important challenges for future research, especially in quality control, confidentiality, and adaptation to new data paradigms. The perspectives outlined suggest a path toward more general and reliable solutions, where schema matching evolves into a broader alignment problem that integrates multiple sources and multiple modalities.

Schema matching remains a cornerstone of data integration. As data grows in scale, variety, and complexity, the ability to align incomplete and heterogeneous sources will be central to building reliable analytic and knowledge systems. The contributions of this thesis demonstrate that enrichment, embeddings, and meta-features together provide a strong foundation for the next generation of schema matching frameworks.

Bibliography

[1] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. Profiling relational data: A survey. *The VLDB Journal*, 24(4):557–581, 2015.

[2] Dogu Araci. Finbert: Financial sentiment analysis with pre-trained language models. *arXiv preprint arXiv:1908.10063*, 2019.

[3] Md. Asif-Ur-Rahman, Bayzid Ashik Hossain, Michael Bewong, Md Zahidul Islam, Yanchang Zhao, Jeremy Groves, and Rory Judith. A semi-automated hybrid schema matching framework for vegetation data integration. *Expert Systems with Applications*, 229:120405, 2023.

[4] David Aumueller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. Schema and ontology matching with coma++. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data (SIGMOD '05)*, pages 906–908. ACM, 2005.

[5] Hamed Babaei Giglou, Jennifer D'Souza, Felix Engel, and Sören Auer. Llms4om: Matching ontologies with large language models. In *The Semantic Web: ESWC 2024 Satellite Events*, volume 15344 of *Lecture Notes in Computer Science*, pages 25–35. Springer, 2024.

[6] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.

[7] Alexander Bilke and Felix Naumann. Schema matching using duplicates. In *21st International Conference on Data Engineering (ICDE'05)*, pages 69–80. IEEE, 2005.

[8] Olivier Bodenreider. The unified medical language system (umls): integrating biomedical terminology. *Nucleic Acids Research*, 32(Database issue):D267–D270, 2004.

[9] Alex Bogatu, Alvaro AA Fernandes, Norman W Paton, and Nikolaos Konstantinou. Dataset discovery in data lakes. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020.

[10] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

[11] Christopher Buss, Mahdis Safari, Arash Termehchy, Stefan Lee, and David Maier. Towards scalable schema mapping using large language models. *arXiv preprint arXiv:2505.24716*, 2025.

[12] Michael J Cafarella, Alon Halevy, and Nodira Khoussainova. Data integration for the relational web. *Proceedings of the VLDB Endowment*, 2(1):1090–1101, 2009.

[13] Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: Exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, 1(1):538–549, 2008.

[14] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. Creating embeddings of heterogeneous relational datasets for data integration tasks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1335–1349, 2020.

[15] Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009.

[16] Silvana Castano and Valeria De Antonellis. Global viewing of heterogeneous data sources. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):277–297, 2001.

[17] Raul Castro Fernandez, Ziawasch Abedjan, et al. Aurum: A data discovery system. In *IEEE ICDE*, pages 1001–1012, 2018.

[18] Raul Castro Fernandez and Samuel Madden. Termite: A system for tunneling through heterogeneous data. In *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, pages 1–8, 2019.

[19] Frédéric Chazal and Bertrand Michel. An introduction to topological data analysis: Fundamental and practical aspects for data scientists. *Frontiers in Artificial Intelligence*, 4:667963, 2021. Souvent confondu avec un livre CUP ; ceci est l'article de synthèse révisé (meilleure citation).

[20] Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles Sutton. Colnet: Embedding the semantics of web tables for column type prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2975–2982, 2019. arXiv:1811.01304.

[21] Xingyu Chen, Sanjib Das, and Jeff Heflin. Prompting llms for schema matching: The alt-gen benchmark. In *Proceedings of the VLDB 2024 TaDA Workshop*, 2024. Workshop paper; DOI non disponible.

[22] Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David Karger. Arda: Automatic relational data augmentation for machine learning, 2020.

[23] Christina Christodoulakis, Eric B Munson, Moshe Gabel, Angela Demke Brown, and Renée J Miller. Pytheas: Pattern-based table discovery in csv files. *Proceedings of the VLDB Endowment*, 13(11):2075–2089, 2020.

[24] Vassilis Christophides, Vasilis Efthymiou, and Kostas Stefanidis. *Entity Resolution in the Web of Data*. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool, 2021.

[25] Xu Chu, Ihab F Ilyas, Sanjay Krishnan, and Jiannan Wang. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 international conference on management of data*, pages 2201–2206, 2016.

[26] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IJCAI-2003 Workshop on Information Integration on the Web (IIWeb-03)*, 2003.

[27] Isabel F Cruz, Flavio Palandri Antonelli, and Cosmin Stroe. Agreementmaker: Efficient matching for large real-world schemas and ontologies. *Proceedings of the VLDB Endowment*, 2(2):1586–1589, 2009.

[28] Fred J. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

[29] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. Finding related tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD '12)*, pages 817–828, 2012.

[30] Li Deng, Shuo Zhang, and Krisztian Balog. Table2vec: Neural word and entity embeddings for table population and retrieval. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '19)*, pages 1029–1032, 2019.

[31] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. Turl: Table understanding through representation learning. *Proceedings of the VLDB Endowment*, 14(3):307–319, 2021.

[32] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[33] Robin Dhamankar, Yoonkyong Lee, AnHai Doan, Alon Halevy, and Pedro Domingos. imap: Discovering complex semantic matches between database schemas. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 383–394, 2004.

[34] Lee R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[35] Hong-Hai Do, Sergey Melnik, and Erhard Rahm. Comparison of schema matching evaluations. In *Net. ObjectDays: International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World*, pages 221–237. Springer, 2002.

[36] Hong Hai Do and Erhard Rahm. Coma: A system for flexible combination of schema matching approaches. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB 2002)*, pages 610–621. Morgan Kaufmann, 2002.

[37] AnHai Doan, Pedro Domingos, and Alon Halevy. Learning to match the schemas of data sources: A multistrategy learning approach. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB 2001)*, pages 279–290, 2001.

[38] AnHai Doan, Pedro Domingos, and Alon Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data (SIGMOD '01)*, pages 509–520. ACM, 2001.

[39] Anhai Doan, Pedro Domingos, and Alon Halevy. Learning to match the schemas of data sources: A multistrategy approach. *Machine Learning*, 50(3):279–301, 2003.

[40] AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of Data Integration*. Elsevier, 2012.

[41] Xin Luna Dong and Divesh Srivastava. *Big Data Integration*. Number 3 in Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2015.

[42] Kawin Ethayarajh. How contextual are contextualized word representations? comparing the geometry of bert, elmo, and gpt-2. In *Proceedings of EMNLP-IJCNLP 2019*, pages 55–65. Association for Computational Linguistics, 2019.

[43] Yuyang Fang et al. Llms for tabular data: Challenges, methods, and opportunities. *Proceedings of the 30th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2024. Entrée ambiguë : aucune notice DOI/ACM correspondant exactement à ce titre/auteurs n'a été trouvée. Voir à titre lié : *Large Language Models (LLMs) on Tabular Data: Prediction, Generation, and Understanding – A Survey*, arXiv:2402.17944.

[44] Daniel Faria, Cátia Pesquita, Emanuel Santos, Isabel F. Cruz, and Francisco M. Couto. The agreementmakerlight ontology matching system. In *Proceedings of OTM Confederated International Conferences "On the Move to Meaningful Internet Systems" (OTM 2013)*, volume 8185 of *Lecture Notes in Computer Science*, pages 527–541. Springer, 2013.

[45] Raul Castro Fernandez, Essam Mansour, et al. Seeping semantics: Linking datasets using word embeddings for data discovery. In *IEEE ICDE*, 2018.

[46] Guillem Garrido, Nils Reimers, and Iryna Gurevych. Rankme: Assessing the quality of text embeddings via effective rank. *arXiv preprint arXiv:2210.02885*, 2022.

[47] Lise Getoor and Ashwin Machanavajjhala. Entity resolution: theory, practice & open challenges. *Proceedings of the VLDB Endowment*, 5(12):2018–2019, 2012.

[48] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. S-match: An algorithm and an implementation of semantic matching. In *The Semantic Web: Research and Applications (ESWS 2004)*, volume 3053 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2004.

[49] Alon Halevy, Flip Korn, Natalya F Noy, et al. Goods: Organizing google's datasets. In *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data (SIGMOD '16)*, pages 795–806. ACM, 2016.

[50] Benjamin Hättasch, Michael Truong-Ngoc, Andreas Schmidt, and Carsten Binnig. It's ai match: A two-step approach for schema matching using embeddings. In *Proceedings of the 2nd Workshop on Applied AI for Database Systems and Applications (AIDB)*, 2022.

[51] Yeye He, Kris Ganjam, and Xu Chu. Sema-join: Joining semantically-related tables using big table corpora. *Proceedings of the VLDB Endowment*, 8(12):1358–1369, 2015.

[52] Yuan He, Jiaoyan Chen, Denvar Antonyrajah, and Ian Horrocks. Bertmap: A bert-based ontology alignment system. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI)*, 2022.

[53] Sven Hertling and Heiko Paulheim. Olala: Ontology matching with large language models. In *Proceedings of the 12th Knowledge Capture Conference (K-CAP '23)*, pages 131–139. ACM, 2023.

[54] Christoph Hofer, Roland Kwitt, Marc Niethammer, and Andreas Uhl. Deep learning with topological signatures. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

[55] Paul Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.

[56] Matthew A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989.

[57] Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. Logmap: Logic-based and scalable ontology matching. In *The Semantic Web – ISWC 2011, Proceedings, Part I (Lecture Notes in Computer Science, vol. 7031)*, pages 273–288, 2011.

[58] Alexandros Kalousis and Sven Hertling. Deepalignment: Unsupervised ontology matching with refined word vectors. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 857–872, 2018.

[59] Jaewoo Kang and Jeffrey F Naughton. On schema matching with opaque column names and data values. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 205–216, 2003.

[60] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J Miller, and Mirek Riedewald. Santos: Relationship-based semantic table union search. *Proceedings of the ACM on Management of Data*, 1(1):1–25, 2023.

[61] Nour Elhouda Kired, Franck Ravat, Jiefu Song, and Olivier Teste. Embedding-based data matching for disparate data sources. In *Proc. 26th Int. Conf. on Big Data Analytics and Knowledge Discovery (DAWAK)*, pages 66–71, Naples, Italy, August 2024. HAL: hal-04612345.

[62] Nour Elhouda Kired, Franck Ravat, Jiefu Song, and Olivier Teste. Alignment of schema-only and instance-only data sources using large language models. In *Research Challenges in Information Science, RCIS 2025, Proceedings, Part II (Lecture Notes in Business Information Processing)*, pages 67–85, 2025. HAL: hal-05091439.

[63] Nour Elhouda Kired, Franck Ravat, Jiefu Song, and Olivier Teste. Alignment of schema-only and instance-only data sources using large language models. In *Proc. BDA 2025*, 2025. Programme et infos : https://bda2025.sciencesconf.org/.

[64] Pradap Konda, Sanjay Subramanian Seshadri, Elan Segarra, Brent Hueth, and AnHai Doan. Executing entity matching end to end: A case study on magellan. *Proceedings of the 22nd International Conference on Extending Database Technology (EDBT)*, pages 489–500, 2019.

[65] Christos Koutras. *Tabular Data Understanding and Integration*. PhD thesis, TU Delft, 2024.

[66] Christos Koutras, Kyriakos Psarakis, George Siachamis, Andra Ionescu, Marios Fragkoulis, Angela Bonifati, and Asterios Katsifodimos. Valentine in action: Matching tabular data at scale. *Proceedings of the VLDB Endowment*, 14(12):2871–2874, 2021.

[67] Amir Laadhar, Faïza Ghozzi, Imen Megdiche, Franck Ravat, Olivier Teste, et al. Pomap++ results for oaei 2019: Fully automated machine learning approach for ontology matching. In *Proceedings of the 14th International Workshop on Ontology Matching (OM@ISWC 2019)*, volume 2536 of *CEUR Workshop Proceedings*, pages 169–174, Auckland, New Zealand, oct 2019. CEUR-WS.org. HAL: hal-05091439.

[68] Amir Laadhar, Faiza Ghozzi, Imen Megdiche, Franck Ravat, Olivier Teste, and Faiez Gargouri. Partitioning and local matching learning of large biomedical ontologies. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (SAC 2019)*, pages 2285–2292, 2019.

[69] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Chan Ho So, and Jaewoo Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, 2020.

[70] Oliver Lehmberg and Christian Bizer. Stitching web tables for improving matching quality. *Proceedings of the VLDB Endowment*, 10(11):1502–1513, 2017.

[71] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[72] Wen-Syan Li and Chris Clifton. Semint: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data & Knowledge Engineering*, 33(1):49–84, 2000.

[73] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. Ditto: A transformer-based entity matching framework. *arXiv preprint arXiv:2004.00584*, 2020.

[74] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and searching web tables using entities, types and relationships. *Proceedings of the VLDB Endowment*, 3(1-2):1338–1347, 2010.

[75] Yurong Liu, Eduardo Pena, Aécio Santos, Eden Wu, and Juliana Freire. Magneto: Combining small and large language models for schema matching. *arXiv preprint arXiv:2412.08194*, 2024.

[76] Yurong Liu, Aécio Santos, Eduardo H. M. Pena, Roque Lopez, Eden Wu, and Juliana Freire. Enhancing biomedical schema matching with llm-based training data generation. *arXiv preprint*, 2024. Preprint; version OpenReview disponible.

[77] Jayant Madhavan, Philip A Bernstein, AnHai Doan, and Alon Halevy. Corpus-based schema matching. In *21st International Conference on Data Engineering (ICDE'05)*, pages 57–68. IEEE, 2005.

[78] Jayant Madhavan, Philip A Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB 2001)*, pages 49–58, 2001.

[79] Sabine Massmann, Salvatore Raunich, David Aumüller, Patrick Arnold, and Erhard Rahm. Evolution of the coma match system. In *Proceedings of the 6th International Workshop on Ontology Matching (OM 2011)*, volume 814 of *CEUR Workshop Proceedings*, 2011.

[80] Imen Megdiche, Olivier Teste, and Cassia Trojahn. An extensible linear approach for holistic ontology matching. In *The Semantic Web – ISWC 2016, Part I (Lecture Notes in Computer Science)*, volume 9981, pages 393–410. Springer, 2016. Corrige la mention OTM 2015; publication finale ISWC 2016 (clé conservée).

[81] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the 18th IEEE International Conference on Data Engineering (ICDE 2002)*, pages 117–128. IEEE, 2002.

[82] George A. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[83] Renée J Miller, Mauricio A Hernández, Laura M Haas, Lingling Yan, CT Howard Ho, Ronald Fagin, and Lucian Popa. The clio project: Managing heterogeneity. *ACM SIGMOD Record*, 30(1):78–83, 2001.

[84] Alvaro E. Monge and Charles Elkan. The field matching problem: Algorithms and applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 267–270. AAAI Press, 1996.

[85] Jiaqi Mu, Suma Bhat, and Pramod Viswanath. All-but-the-top: Simple and effective postprocessing for word representations. In *Proceedings of the International Conference on Learning Representations (ICLR) — Workshop Track*, 2018. arXiv:1702.01417.

[86] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)*, pages 19–34. ACM, 2018.

[87] Fatemeh Nargesian, Erkang Zhu, Renée J Miller, Ken Q Pu, and Patricia C Arocena. Data lake management: Challenges and opportunities. *Proceedings of the VLDB Endowment*, 12(12):1986–1989, 2019.

[88] Fatemeh Nargesian, Erkang Zhu, Ken Q Pu, and Renée J Miller. Table union search on open data. In *VLDB*, 2018.

[89] Hossam Nassif and AnHai Doan. Learning schema matching with dumas. In *Proceedings of the IEEE 26th International Conference on Data Engineering (ICDE '10) Workshops/Forums*, 2010. Aucun DOI officiel disponible; version atelier/fora ICDE.

[90] Felix Naumann, Heiko Fahling, Melanie Herschel, Bernhard Haupt, and Manfred Weis. Evaluation of schema matching approaches. In *Proceedings of the 2002 International Workshop on Web and Databases (WebDB 2002)*, pages 29–40, 2002. Workshop WebDB; DOI non attribué (pas de DOI officiel disponible).

[91] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[92] Duc Thanh Ngo, Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. Yam++: A multi-strategy ontology matching system. In *Journal on Data Semantics IV (Lecture Notes in Computer Science)*, pages 160–184. Springer, 2005.

[93] Thanh Trung Nguyen, Huu Nguyen, Quoc Viet Hung Nguyen, and Karl Aberer. Adaptive data augmentation for evolving classification. In *2014 IEEE International Conference on Data Mining (ICDM)*, pages 639–648. IEEE, 2014.

[94] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. Blocking and filtering techniques for entity resolution: A survey. *ACM Computing Surveys (CSUR)*, 53(2):1–42, 2020.

[95] Simone Papicchio, Paolo Papotti, and Luca Cagliero. Qatch: Automatic evaluation of sql-centric tasks on proprietary data. *ACM Transactions on Intelligent Systems and Technology*, 16(2):45:1–45:26, 2025.

[96] Marcel Parciak, Brecht Vandevoort, Frank Neven, Liesbet M. Peeters, and Stijn Vansummeren. Schema matching with large language models: An experimental study. *arXiv preprint arXiv:2407.11852*, 2024.

[97] José A. Perea. Topological time series analysis. *Notices of the American Mathematical Society*, 66(5):686–694, 2019.

[98] Fotis Psallidas, Yiwen Zhu, Bojan Karlas, Jordan Henkel, Matteo Interlandi, Subru Krishnan, Brian Kroth, Venkatesh Emani, Wentao Wu, Ce Zhang, et al. Data science through the looking glass: Analysis of millions of github notebooks and ml.net pipelines. *ACM SIGMOD Record*, 51(2):30–37, 2022.

[99] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[100] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.

[101] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4):3–13, 2000.

[102] Philippe Roussille, Imen Megdiche, Olivier Teste, and Cássia Trojahn. Boosting holistic ontology matching: Generating graph clique-based relaxed reference alignments for holistic evaluation. In *Knowledge Engineering and Knowledge Management (EKAW 2018), LNCS*, pages 355–369, 2018. Lecture Notes in Computer Science, vol. 11313. Publisher: Springer.

[103] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[104] Gerard Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[105] Nabeel Seedat and Mihaela van der Schaar. Matchmaker: Self-improving large language model programs for schema matching. *arXiv preprint arXiv:2410.24105*, 2024.

[106] Lee M. Seversky, Steven Davis, and Matthew Berger. Time-series topological data analysis: New data and opportunities. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 59–67. IEEE, 2016.

[107] Eitam Sheetrit, Menachem Brief, Moshik Mishaeli, and Oren Elisha. Rematch: Retrieval enhanced schema matching with llms. *arXiv preprint arXiv:2403.01567*, 2024.

[108] Roee Shraga, Avigdor Gal, and Haggai Roitman. Adnev: Cross-domain schema matching using deep similarity matrix adjustment and evaluation. *Proceedings of the VLDB Endowment*, 13(9):1401–1415, 2020.

[109] Pavel Shvaiko and Jérôme Euzenat. A survey of schema-based matching approaches. In *Journal on Data Semantics IV*, pages 146–171. Springer, 2005.

[110] Pavel Shvaiko and Jérôme Euzenat. Ontology matching: State of the art and future challenges. *IEEE Transactions on Knowledge and Data Engineering*, 25(1):158–176, 2011.

[111] Ravid Shwartz-Ziv and Amitai Armon. Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90, 2022.

[112] Michael Stonebraker, Daniel Bruckner, Ihab F Ilyas, et al. Data curation at scale: The data tamer system. In *CIDR*, 2013.

[113] Fabian M. Suchanek, Serge Abiteboul, and Pierre Senellart. Paris: Probabilistic alignment of relations, instances, and schema. *Proceedings of the VLDB Endowment*, 5(3):157–168, 2011.

[114] Andreas Thor and Erhard Rahm. Moma: A mapping-based object matching system. In *Proceedings of the 3rd Biennial Conference on Innovative Data Systems Research (CIDR 2007)*, pages 247–258, 2007.

[115] Anton Tsitsulin, Enver Kacupaj, Achim Rettinger, and Elena Demidova. Unsupervised evaluation metrics and learning criteria for schema matching. In *Proceedings of the 2023 ACM SIGMOD International Conference on Management of Data*, 2023.

[116] Esko Ukkonen. Approximate string-matching with q-grams and maximal matches. *Theoretical Computer Science*, 92(1):191–211, 1992.

[117] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

[118] S. Wang et al. Llmatch: A unified schema matching framework with large language models. *arXiv preprint arXiv:2507.10897*, 2025.

[119] William E. Winkler. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. Technical report, U.S. Bureau of the Census, 1990.

[120] Jinghui Xu, Zhanxing Zhu, Hongyu Ren, and Shuiwang Ji. Understanding and improving layer normalization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[121] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. Infogather: Entity augmentation and attribute discovery by holistic matching with web tables. In *ACM SIGMOD*, pages 97–108, 2012.

[122] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. Tabert: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020)*, pages 8413–8426. Association for Computational Linguistics, 2020.

[123] Jing Zhang, Bonggun Shin, Jinho D. Choi, and Joyce C. Ho. SMAT: An Attention-based Deep Learning Solution to the Automation of Schema Matching. In *Proceedings of the 25th European Conference on Advances in Databases and Information Systems*, ADBIS'21, pages 260–274, 2021.

[124] Meihui Zhang and Kaushik Chakrabarti. Infogather+: Semantic matching and annotation of numeric and time-varying attributes in web tables. In *ACM SIGMOD*, pages 145–156, 2013.

[125] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, et al. Automatic discovery of attributes in relational databases. In *ACM SIGMOD*, pages 109–120, 2011.

[126] Shuo Zhang and Krisztian Balog. Entitables: Smart assistance for entity-focused tables. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17)*, pages 255–264. ACM, 2017.

[127] Yi Zhang and Zachary G Ives. Finding related tables in data lakes for interactive data science. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20)*, pages 1951–1966. ACM, 2020.

[128] Yu Zhang, Mei Di, Haozheng Luo, Chenwei Xu, and Richard Tzong-Han Tsai. Smutf: Schema matching using generative tags and hybrid features. *arXiv preprint arXiv:2402.01685*, 2025. Dernière révision: May 3, 2025 (v3).

[129] Zixuan Zhao and Raul Castro Fernandez. Leva: Boosting machine learning performance with relational embedding data augmentation. In *Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data*, pages 1504–1517, 2022.

[130] Afra Zomorodian and Gunnar Carlsson. Computing persistent homology. *Discrete & Computational Geometry*, 33(2):249–274, 2005.

RÉPUBLIQUE FRANÇAISE
Liberté Égalité Fraternité

Université de Toulouse

Titre : Alignement de sources de données disparates à l'aide de grands modèles de langage

Mots clés : Appariement de données, Données tabulaires disparates, Génération d'informations auxiliaires, Apprentissage automatique, Méta-espace

Résumé : Cette thèse s'inscrit dans le domaine de l'intégration de données, et plus particulièrement dans l'alignement de schémas de données tabulaires hétérogènes et incomplets. Dans des environnements tels que les lacs de données, les systèmes fédérés ou les bases multi-modèles, les sources de données présentent souvent des disparités : certaines ne disposent que d'un schéma (colonnes sans instances), d'autres uniquement d'instances (valeurs sans attributs), ou encore des versions partielles et bruitées. Ces situations compliquent considérablement les tâches classiques de correspondance de schémas, qui supposent habituellement des sources complètes et symétriques.
Pour relever ce défi, nous introduisons le concept de génération de contexte, consistant à produire automatiquement les éléments manquants (par exemple, générer des instances plausibles à partir d'un schéma, ou inférer des noms de colonnes à partir de valeurs). Grâce aux avancées des modèles de langage de grande taille (LLMs), il devient possible de synthétiser des contextes réalistes et pertinents, puis de les exploiter dans des processus de mise en correspondance.
La contribution principale de cette recherche est le développement d'un cadre unifié de correspondance de schémas capable de fonctionner dans des scénarios variés, qu'ils soient conformes (sources complètes, uniquement schéma, uniquement instances) ou disparates (sources partiellement ou asymétriquement spécifiées). Deux approches sont proposées :
1. une approche non supervisée, combinant plusieurs algorithmes de correspondance enrichis par contexte généré,
2. une approche supervisée, qui exploite des représentations vectorielles (embeddings) et un ensemble riche de méta-caractéristiques pour entraîner un classifieur de correspondances.
Les résultats expérimentaux, obtenus notamment sur le benchmark Valentine et sur des jeux de données réels, montrent que nos méthodes atteignent ou dépassent l'état de l'art, y compris dans les cas les plus difficiles de sources disparates. Nous mettons également à disposition une implémentation open source, ainsi que les ressources expérimentales utilisées.
En résumé, cette thèse propose des solutions innovantes pour améliorer la résilience et l'autonomie des systèmes d'intégration de données, en s'appuyant sur la génération et l'exploitation de contexte pour dépasser les limites des approches traditionnelles.

Title: aligning disparate data sources using large language models

Key words: Data Matching, Disparate Tabular Data., Auxiliary Information Generation, Machine Learning, Meta Space

Abstract: This thesis addresses the challenge of schema matching for heterogeneous and incomplete tabular datasets, a recurring problem in data lakes, federated systems, and multi-model environments. Real-world data sources are often structurally disparate: some provide schema definitions without instance data, others contain raw values without headers, while many present partial, noisy, or inconsistent information. Such conditions significantly compromise traditional schema matching methods, which typically assume well-structured and symmetric sources.
To overcome these challenges, we propose the notion of context generation, i.e., automatically synthesizing missing elements to enrich datasets before alignment. Examples include generating representative values for schema-only tables, or inferring attribute names from instance-only tables. Recent advances in large language models (LLMs) make this generative reasoning feasible, enabling more robust comparisons even under severe data disparities.
Building on this idea, we develop a unified schema matching framework that handles both compliant (complete, schema-only, instance-only) and disparate (asymmetric or incomplete) scenarios. Our contributions include:
1. an unsupervised matching framework that ensembles multiple matchers enriched by generated context, and
2. a supervised approach that embeds enriched attributes into a common semantic space, extracts a comprehensive set of meta-features, and trains a classifier for accurate correspondence prediction.
Extensive experiments on the Valentine benchmark and additional real-world datasets demonstrate that our methods achieve state-of-the-art performance, including in previously intractable scenarios such as schema-only vs. instance-only matching. Moreover, we release an open-source implementation, datasets, and evaluation resources to support reproducibility and further research.
In summary, this thesis advances schema matching by leveraging context generation with LLMs, enabling more resilient, autonomous, and accurate data integration strategies.