SemForest: Semantic-Aware Ontology Generation with Foundation Models

Guohui Guan[†] Sachin Konan[§] Larry Rudolph[¶] Chang Ge[†]
[†]University of Minnesota [§]Princeton University [¶]MIT CSAIL
{gguan, cge}@umn.edu, sk7524@princeton.edu, rudolph@csail.mit.edu

ABSTRACT

Functional Semantic Types (FSTs) enrich column-level semantics by pairing type information with executable logic for data transformation and validation. However, to our best knowledge, the only existing FST generation method relies primarily on name-based merging, resulting in flat, unstructured hierarchies that do not align with real-world semantic structures. We introduce SemForest, a framework that constructs a tree-structured semantic forest of FSTs. SemForest produces the ontology with interpretable semantic meaning by clustering related types in embedding space, and leveraging large language models to organize them into hierarchical trees. The resulting ontology improves interpretability and accelerates semantic retrieval through hierarchical navigation. Experiments on three public data universes demonstrate that SemForest improves retrieval recall while reducing search time compared to the existing baseline.

VLDB Workshop Reference Format:
Guohui Guan, Sachin Konan, Larry Rudolph, and Chang Ge. SemForest: Semantic-Aware Ontology Generation with Foundation Models. VLDB 2025 Workshop: Tabular Data Analysis (TaDA).

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/golden-eggs-lab/semforest.

1 INTRODUCTION

Understanding and organizing the semantics of relational data is crucial for tasks such as automatic join discovery [9, 17], schema matching [29], and large-scale data cleaning [30], and has been widely used and explored such as in the SemTab Challenge [5]. A *semantic type* [23] expresses the real-world meaning of a column beyond its SQL data type [8]; for instance, an `INT` column in a table may represent ages, tax years, or job levels. Recent work has enriched semantic types with executable logic in the form of *Functional Semantic Types* (FSTs) [16], implemented as Python classes that provide transformation and validation methods. These FSTs help automate data onboarding tasks that previously required significant manual effort.

Large language models (LLMs) offer a strong foundation for this process. Trained on diverse corpora, they can reason over schema metadata, example values, and distributional signals to generate context-sensitive type annotations [12]. Their ability to produce executable code [7] also enables the synthesis of functional logic, making them well-suited for generating and organizing FSTs.

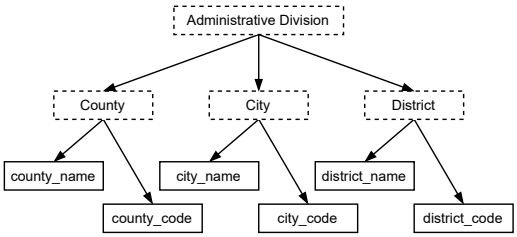

Figure 1: An example semantic tree built from the Kaggle [3] data universe. Constructing details are in § 4.

Despite this promise, to our best knowledge, the only existing FST generation system, FSTO-Gen [16], lacks a semantically structured ontology that reflects how humans organize knowledge. FSTO-Gen relies primarily on name-based merging rather than deeper semantic reasoning, resulting in an ontology with no meaningful groupings or abstractions. For example, in Figure 1, it is desirable to categorize `city_name` and `city_code` under a type such as `City`, and `county_name` and `county_code` under `County`. In FSTO-Gen, however, all four semantic types ({ `city_name, city_code, county_name, county_code`}) remain in a flat collection with no higher-level structure. Introducing a hierarchy, as illustrated, brings semantic organization that enhances interpretability.

To produce FSTs within a semantic-aware structure, we introduce SemForest, which organizes FSTs into a structured organization in the form of a *semantic forest*, leveraging the capabilities of foundation models [6, 26]. Each semantic tree corresponds to a coherent semantic domain, with a hierarchy that reflects real-world semantics. Internal nodes represent broader semantic categories that enhance interpretability and support downstream reasoning. The generated ontology serves as a functional structure that supports concept abstraction and executable transformations, such as type casting and validation.

Our main contributions are as follows:

- We propose a new ontology construction framework for FSTs named SemForest, grounded in real-world semantics (§4).
- We demonstrate that SemForest improves interpretability and retrieval effectiveness in real-world application scenarios (§5).
- We release both the prototype of SemForest and two semantic-aware applications with curated benchmarking datasets [4] to serve as a foundation to future research for the community.

Proceedings of the VLDB Endowment. ISSN 2150-8097.

2 PRELIMINARIES

Data Universe, Product, and Table. We consider a relational data universe $\mathcal{D}$ organized hierarchically: a *universe* consists of multiple *products*, each containing a set of *tables*. A product typically represents a coherent collection curated by the same source or community, where tables may share semantic structure.

Functional Semantic Types (FSTs). An FST is represented as a Python class with a name, metadata (e.g., descriptions, example values), and transformation and validation logic. FSTs are scoped at three levels: table-level (`T-FSTs`), product-level (`P-FSTs`), and universe-level (`G-FSTs`), reflecting increasing semantic generality.

FSTO-Gen. The only existing FST ontology method, FSTO-Gen [16], does so in a bottom-up manner. It first uses an LLM to generate `T-FSTs` from individual tables, based on column-level summaries. Within each product, `T-FSTs` with identical class names are grouped and merged into `P-FSTs`, selecting the most reliable implementation (i.e., the one with the lowest `cast()` error) as representative. Across products, same-named `P-FSTs` are further merged into `G-FSTs` via another round of LLM synthesis, forming universe-level types. Finally, FSTO-Gen adds cross-type cast relations between `G-FSTs` by embedding them and identifying semantically close neighbors, prompting the LLM to generate transformation logic.

Note that this merging process is primarily driven by class name matching. Once an `T-FST` is generated for a column, its corresponding `P-FST` and `G-FST` inherit the same class name. On one hand, this approach constructs an ontology with a representational hierarchy from columns to `T-FSTs`, to `P-FSTs`, to `G-FSTs`, but it lacks meaningful organization within levels. Consequently, the ontology remains flat, with type relationships that do not align with real-world semantic structures. On the other hand, we construct our ontology directly at the `G-FST` level. Since `T-FSTs` and `P-FSTs` are scoped to individual tables and products, they often contain redundant or overlapping semantics when viewed from the universe perspective. In contrast, `G-FSTs` offer a more consolidated view of type semantics at the universe scope. Moreover, building the ontology over `G-FSTs` enables top-down traversal that naturally connects to relevant `P-FSTs`, `T-FSTs`, and underlying columns.

Semantic Trees. Semantic trees have long served as a conceptual tool for encoding structured knowledge in domains such as lexical semantics [21], programming languages [18], and formal logic [24]. In these contexts, they are typically abstract representations used to express hierarchical relationships among concepts or symbols. In our setting, we materialize the concept of semantic trees as a concrete ontology over FSTs. Each node in the tree corresponds to a semantic type, and edges denote specialization relationships from parent to child. However, a single semantic tree is often insufficient to capture the diversity of semantic categories found in real-world data. It is unnatural to force types from different domains into a single unified hierarchy. To accommodate this heterogeneity, we generalize the structure to a semantic forest, which consists of multiple trees, each representing a coherent semantic domain.

3 PROBLEM AND SOLUTION OVERVIEW

Problem Statement. Given a data universe $\mathcal{D}$, a set of FST class templates $\mathcal{T}$, and a language model $\mathcal{M}$, our goal is to generate a set of FSTs F and organize them into a structured ontology in the form of a semantic forest. Formally, we define a process P such that $P(\mathcal{D}, \mathcal{T}, \mathcal{M}) \rightarrow \{F, O\}$, where F is the set of FSTs and O is an ontology structured as a forest of semantic trees.

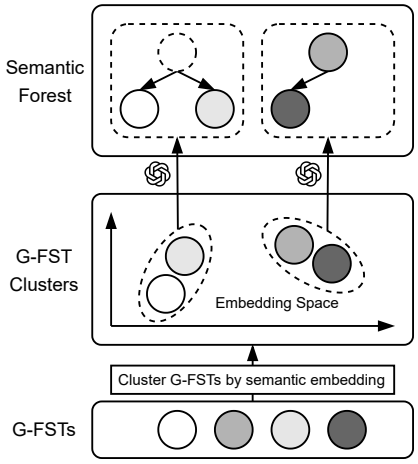

Figure 2: An illustration of semantic forest construction from `G-FSTs` in SemForest. The steps are executed in a bottom-up manner, starting with clustering `G-FSTs` into groups, followed by querying the LLM to construct semantic trees.

Solution Overview. We propose SemForest, a system that constructs a semantically grounded ontology structured as a forest of hierarchical trees. As illustrated in Figure 2, the core contribution of SemForest lies in organizing `G-FSTs` into a *semantic forest*. This process involves two stages: (1) clustering `G-FSTs` into semantically coherent groups in their embeddings space, and (2) prompting the LLM to build a hierarchical tree structure for each cluster, then assembling these trees into a forest. Each resulting tree corresponds to a coherent semantic domain, with internal nodes capturing increasingly general semantic types. Each tree's hierarchical structure enables top-down semantic navigation, facilitates pruning during retrieval, and improves interpretability by explicitly representing parent–child relationships among types.

4 SEMFOREST PROCESS

A naïve approach to building semantic hierarchies would prompt a language model to directly organize the entire set of `G-FSTs` into structured trees. However, this quickly becomes impractical at scale. Thousands of types introduce input length limitations, and their semantic heterogeneity makes it difficult for the model to produce coherent groupings and hierarchies in a single pass.

To address this, we first group the `G-FSTs` into semantically coherent subsets through clustering. This preprocessing step reduces complexity by constraining the LLM's context to a focused set of related types, making it easier to infer meaningful generalizations. The overall generation process is summarized in Algorithm 1.

4.1 Clustering `G-FSTs` into Semantic Groups

A common approach to clustering, such as k-means [19, 20] or agglomerative clustering [22], requires specifying the number of

Algorithm 1 Semantic forest ontology generation.

Require: Set of G-FSTs $S^{\texttt{G-FST}}$, LLM $\mathcal{M}$
1: **procedure** SemanticForestGen($S^{\texttt{G-FST}}, \mathcal{M}$)
2: Compute embeddings for all $g \in S^{\texttt{G-FST}}$ to obtain matrix E
3: Set max cluster size k, epsilon schedule $\boldsymbol{\varepsilon}$
4: $C \leftarrow$ RecursiveDBSCAN($E, k, \boldsymbol{\varepsilon}$) ▷ Algorithm 2
5: $O \leftarrow \emptyset$ ▷ Initialize semantic forest ontology
6: **for all** cluster $c \in C$ **do**
7: $T_c \leftarrow \mathcal{M}(c)$ ▷ LLM constructs tree for cluster c
8: $O \leftarrow O \cup \{T_c\}$
9: **end for**
10: **return** O ▷ Return semantic forest ontology
11: **end procedure**

clusters in advance. However, in our setting, this is difficult to determine, as semantic types vary widely in both size and granularity. To address these limitations, we adopt a density-based clustering method, DBSCAN [10], which avoids assumptions about cluster count. DBSCAN groups types based on local density, allowing us to discover clusters of varying sizes, which is essential for capturing the heterogeneous structure of semantic categories.

DBSCAN requires a neighborhood radius parameter ε, which controls how close two points must be in the embedding space to be assigned to the same cluster. However, naïvely using a single fixed ε is problematic. A larger ε merges more distant types, producing broad clusters that may overwhelm the LLM due to context window limits and semantic interference. In contrast, a smaller ε yields finer-grained clusters, but the resulting trees are often too small to be meaningful, making them trivial or uninformative. To overcome these limitations, we adopt a recursive DBSCAN strategy that dynamically adjusts ε. This approach balances semantic coherence and cluster size, ensuring that each subset remains tractable for LLM-based hierarchy construction.

To enable clustering, we first embed each G-FST into a dense semantic space (Algorithm 1, Line 2). These embeddings encode semantic similarity, allowing types with related meanings to be grouped based on spatial proximity. Clustering is then performed in the embedding space using a controlled, recursive variant of DBSCAN (Algorithm 2, Appendix B). The process begins with a relatively large neighborhood radius ϵ to identify coarse semantic groupings. If any resulting cluster exceeds a predefined size threshold, it is recursively refined using smaller ϵ values. This continues until all clusters are both size-bounded and semantically coherent. The resulting subsets serve as focused inputs for the next stage, where an LLM constructs semantic hierarchies within each cluster.

4.2 Semantic Tree Construction

While clustering addresses the scalability challenge, the next problem is structural: how to create meaningful hierarchies within each subset of G-FSTs. Although types in a cluster are semantically related, they often lack explicit parent–child relationships and exist at similar levels of abstraction. As a result, naïvely linking them into a tree would create shallow or incoherent hierarchies that fail to reflect real-world semantic organization.

To overcome this limitation, we leverage LLMs, which can propose abstractions not directly observable in the input. Specifically, we prompt the LLM to synthesize internal nodes that represent abstract semantic categories missing from the original type set. For example, given types like `city_name` and `city_code`, the LLM may introduce a higher-level node such as `City`, thereby establishing a meaningful structure over the cluster.

We construct these hierarchies by providing the LLM with the list of G-FSTs in each cluster and asking it to organize them into a semantic tree (Algorithm 1, Lines 7). The resulting tree includes the original G-FSTs as leaf nodes and synthesized or reused types as internal nodes. These trees are then assembled into a semantic forest (Line 10), which forms the structured ontology.

5 EVALUATION

We evaluate using two semantic-aware applications, namely, joinability detection and concatenation recovery, to demonstrate how the semantic forest improves both retrieval accuracy and efficiency.

5.1 Evaluation Setup

Datasets. We evaluate across three datasets spanning both general-purpose and domain-specific settings. Table 3 and Table 4 in Appendix A summarize dataset characteristics and ontology statistics generated by SemForest and FSTO-Gen.

Baseline. We compare SemForest against FSTO-Gen [16].

Evaluation Tasks. We consider two semantic retrieval tasks:
1) Joinability: Given a column C, identify columns X in the data universe such that X and C share the same semantic type (e.g., both are `Country`) and contain related values, making them joinable.
2) Concatenation: Given a column C, identify two columns $\{X_1, X_2\}$ in the data universe from which C can be derived through concatenation (e.g., `year_month` from `year` and `month`).

Benchmark Construction. There are previous works [17] evaluating joinability tasks, but to our best knowledge, their data and tasks are not publicly available. To enable recall-based evaluation with explicit ground-truth, we construct our own retrieval benchmarks and release it on GitHub [4].

We construct benchmarks without relying on manually labeled ground-truth by generating retrieval targets through data transformations. Specifically, we apply operations such as semantic-preserving value edits to existing columns to create new ones. This approach allows us to identify which columns serve as ground-truth targets for evaluation. For datasets with ground-truth, one approach is to use the semantic type alone as the retrieval signal. However, having the same semantic type does not guarantee joinability. For example, `temperature_celsius` and `temperature_fahrenheit` may share the same type but are not directly joinable.

For the joinability task, we sample columns from the data universe and apply a set of nine semantic-preserving transformations, such as value translation and reverse translation for categorical values, or numeric normalization for numerical values, to create variants. These variants, along with the original, are inserted back into the universe. The goal is to retrieve all variants of a column given the original as the query. For the concatenation task, we

Table 1: Comparison of joinability across 3 universes. We report recall (higher is better), search time, and the number of semantic types accessed during retrieval (lower is better).

Metric	Kaggle		Harvard		BiodivTab	
	FSTO.	SemForest	FSTO.	SemForest	FSTO.	SemForest
Recall	0.5833	**0.6792**	0.2600	**0.6200**	0.7788	**0.8924**
Time (ms)	2,594.19	**603.68**	217.73	**108.27**	478.30	**136.95**
# Types	2,529	**994**	1,662	**1,227**	705	**286**

Table 2: Comparison of concatenation across 3 universes. We report recall (higher is better), search time, and the number of semantic types accessed during retrieval (lower is better).

Metric	Kaggle		Harvard		BiodivTab	
	FSTO.	SemForest	FSTO.	SemForest	FSTO.	SemForest
Recall	0.3546	**0.4362**	0.1000	**0.3611**	0.5725	**0.6267**
Time (ms)	1,957.41	**437.14**	143.22	**68.40**	500.36	**163.07**
# Types	2,529	**1,000**	1,662	**1,204**	705	**318**

sample pairs of columns, concatenate them to form a synthetic column, and use it as a query to retrieve the original source columns. In both settings, ground-truth targets are explicitly constructed and inserted into the universe, enabling recall-based evaluation of ontology-guided retrieval.

Retrieval Methods. FSTO-Gen exhaustively scans all semantic types in the ontology. In contrast, SemForest leverages its tree-structured ontology to perform a two-stage retrieval: it first identifies the most relevant trees via root-level embeddings, and then searches within the selected tree using its internal hierarchy. This hierarchical strategy significantly reduces both search time and the number of types accessed per query.

For both tasks, we first query the ontology to retrieve candidate columns and then use an LLM to identify the correct matches. Since the LLM-based verification step is identical across methods, our evaluation focuses on the ontology-driven retrieval stage. For a fair comparison, we restrict evaluation to columns for which both SemForest and FSTO-Gen successfully produce semantic types, using their intersection as the query set.

Evaluation Metrics. We report *recall* over the ground-truth columns associated with each query, as well as query runtime and the number of semantic types accessed during the retrieval process. We do not report *precision* because, in the retrieved columns, there may be additional correct matches that are not labeled, making precision unreliable to evaluate.

Implementation Details. We implement SemForest in Python 3.11, use OpenAI's API to access the `gpt-4o-mini` model, and OpenAI's `text-embedding-3-small` model to encode G-FSTs into vectors. Prompts used to query the LLM are publicly available [4]. For FSTO-Gen, we adopt the original implementation provided by the authors [16]. All experiments are conducted on a server equipped with dual Intel Xeon Gold 5218 CPUs and 512 GB of RAM. Reported results are averaged over 10 independent runs.

5.2 Evaluation Results

Experiment 1: Identifying Joinable Columns. Table 1 reports recall, query time, and the number of FSTs accessed during retrieval across the Kaggle, Harvard, and BiodivTab universes.

SemForest outperforms FSTO-Gen on all three metrics. Higher recall shows that SemForest retrieves more ground-truth columns, demonstrating improved accuracy. It also achieves lower query time and accesses fewer semantic types by leveraging its hierarchical structure for efficient retrieval. Its two-stage search strategy, where it first identifies relevant trees and then selects branches within them, enables effective pruning of the search space.

Experiment 2: Identifying Concatenated Columns. Table 2 presents results on the concatenation benchmark. SemForest again achieves higher recall while reducing both query time and the number of accessed semantic types. These improvements reflect its ability to efficiently identify semantically related column pairs by navigating the semantic forest structure.

Overall, these results highlight the benefits of the semantic forest in supporting accurate, efficient semantic-aware column retrieval.

6 RELATED WORK

Semantic Type Annotation in Tables. Early methods [15, 25, 28] framed column typing as a supervised classification task over fixed taxonomies, using deep learning models or context cues. Though effective for common types, these approaches struggled with domain-specific or rare types and required predefined label sets. LLMs have enabled zero-shot semantic type annotation by inferring column types from values or descriptions without task-specific training [11]. Unlike traditional classifiers, LLMs can flexibly handle custom or fine-grained types. Recent work [12] explored prompting LLMs for various table-related tasks, laying the groundwork for richer ontology construction: instead of predefining a taxonomy, one can leverage the LLM's general knowledge to dynamically populate ontologies with types and their relationships.

Ontology Construction and Hierarchical Semantics. Prior work has explored automatic ontology construction from text (e.g., Hearst patterns [14], Probase [27]) and from web tables [13]. These systems demonstrated that large-scale, data-driven taxonomies are feasible, but they typically focused on general lexical or encyclopedic knowledge and lacked support for functional data types or executable logic. FSTO-Gen [16] introduced a new paradigm by generating Functional Semantic Types that pair column semantics with executable logic (e.g., normalization, validation). Its ontology is constructed in a bottom-up manner and lacks the higher-order groupings needed for interpretability and efficient retrieval.

7 CONCLUSION

Understanding functional semantics in relational data is key to building scalable, intelligent systems. We introduced SemForest, which constructs a tree-structured ontology over FSTs by combining semantic embeddings with LLM-based abstraction. By clustering and organizing types into a semantic forest, SemForest enables interpretable, semantically aware navigation. Experiments across diverse datasets show improved accuracy and efficiency in column retrieval, demonstrating its potential for data integration.

REFERENCES

[1] [n.d.]. BiodivTab Dataset. https://zenodo.org/records/5584180.

[2] [n.d.]. Harvard Dataset. https://github.com/twosigma/functional_semantic_types/blob/main/assets/harvard/harvard_dataset_data_df.csv.

[3] [n.d.]. Kaggle Dataset. https://github.com/twosigma/functional_semantic_types/blob/main/assets/kaggle/kaggle_dataset_data_df.csv.

[4] [n.d.]. SemForest Prototype. https://github.com/golden-eggs-lab/semforest.

[5] [n.d.]. SemTab: Semantic Web Challenge on Tabular Data to Knowledge Graph Matching. https://www.cs.ox.ac.uk/isg/challenges/sem-tab/.

[6] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ B. Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen Creel, Jared Quincy Davis, Dorottya Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren E. Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, and et al. 2021. On the Opportunities and Risks of Foundation Models. *CoRR* abs/2108.07258 (2021).

[7] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[8] E. F. Codd. 1970. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM* 13, 6 (1970), 377–387.

[9] Tianji Cong, James Gale, Jason Frantz, H. V. Jagadish, and Çagatay Demiralp. 2023. WarpGate: A Semantic Join Discovery System for Cloud Data Warehouses. In *13th Conference on Innovative Data Systems Research, CIDR 2023, Amsterdam, The Netherlands, January 8-11, 2023*. www.cidrdb.org.

[10] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *(KDD)*. AAAI Press, 226–231.

[11] Benjamin Feuer, Yurong Liu, Chinmay Hegde, and Juliana Freire. 2023. ArcheType: A Novel Framework for Open-Source Column Type Annotation using Large Language Models. *CoRR* abs/2310.18208 (2023). https://doi.org/10.48550/arXiv.2310.18208

[12] Heng Gong, Yawei Sun, Xiaocheng Feng, Bing Qin, Wei Bi, Xiaojiang Liu, and Ting Liu. 2020. TableGPT: Few-shot Table-to-Text Generation with Table Structure Reconstruction and Content Matching. In *COLING*. International Committee on Computational Linguistics, 1978–1988.

[13] Rahul Gupta, Alon Y. Halevy, Xuezhi Wang, Steven Euijong Whang, and Fei Wu. 2014. Biperpedia: An Ontology for Search Applications. *Proc. VLDB Endow.* 7, 7 (2014), 505–516. http://www.vldb.org/pvldb/vol7/p505-gupta.pdf

[14] Marti A. Hearst. 1992. Automatic Acquisition of Hyponyms from Large Text Corpora. In *14th International Conference on Computational Linguistics, COLING 1992, Nantes, France, August 23-28, 1992*. 539–545.

[15] Madelon Hulsebos, Kevin Zeng Hu, Michiel A. Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César A. Hidalgo. 2019. Sherlock: A Deep Learning Approach to Semantic Data Type Detection. In *KDD*. ACM, 1500–1508.

[16] Sachin Konan, Larry Rudolph, and Scott Affens. 2024. Automating the Generation of a Functional Semantic Types Ontology with Foundational Models. In *NAACL*. Association for Computational Linguistics, 248–265.

[17] Christos Koutras, Jiani Zhang, Xiao Qin, Chuan Lei, Vasileios Ioannidis, Christos Faloutsos, George Karypis, and Asterios Katsifodimos. 2024. OmniMatch: Effective Self-Supervised Any-Join Discovery in Tabular Data Repositories. *CoRR* abs/2403.07653 (2024).

[18] S Kundu. 1986. Tree resolution and generalized semantic tree. In *Proceedings of the ACM SIGART International Symposium on Methodologies for Intelligent Systems* (Knoxville, Tennessee, USA) *(ISMIS '86)*. Association for Computing Machinery, New York, NY, USA, 270–278.

[19] Stuart P. Lloyd. 1982. Least squares quantization in PCM. *IEEE Trans. Inf. Theory* 28, 2 (1982), 129–136.

[20] James MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, Vol. 5. University of California press, 281–298.

[21] Andriy Mnih and Geoffrey E. Hinton. 2008. A Scalable Hierarchical Distributed Language Model. In *NeurIPS*. Curran Associates, Inc., 1081–1088.

[22] Fionn Murtagh and Pedro Contreras. 2017. Algorithms for hierarchical clustering: an overview, II. *WIREs Data Mining Knowl. Discov.* 7, 6 (2017).

[23] Joan Peckham and Fred J. Maryanski. 1988. Semantic Data Models. *ACM Comput. Surv.* 20, 3 (1988), 153–189.

[24] Peter Smith. 2020. *An Introduction to Formal Logic* (2 ed.). Cambridge University Press.

[25] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çagatay Demiralp, Chen Chen, and Wang-Chiew Tan. 2022. Annotating Columns with Pre-trained Language Models. In *SIGMOD*. ACM, 1493–1503.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NeurIPS*. 5998–6008.

[27] Wentao Wu, Hongsong Li, Haixun Wang, and Kenny Qili Zhu. 2012. Probase: a probabilistic taxonomy for text understanding. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*. ACM, 481–492.

[28] Dan Zhang, Yoshihiko Suhara, Jinfeng Li, Madelon Hulsebos, Çagatay Demiralp, and Wang-Chiew Tan. 2020. Sato: Contextual Semantic Type Detection in Tables. *Proc. VLDB Endow.* 13, 11 (2020), 1835–1848.

[29] Yu Zhang, Di Mei, Haozheng Luo, Chenwei Xu, and Richard Tzong-Han Tsai. 2025. Smutf: Schema matching using generative tags and hybrid features. *Information Systems* (2025), 102570.

[30] Xuanhe Zhou, Chengliang Chai, Guoliang Li, and Ji Sun. 2022. Database Meets Artificial Intelligence: A Survey. *IEEE Trans. Knowl. Data Eng.* 34, 3 (2022), 1096–1116.

Table 3: Dataset statistics for the three data universes used in our experiments. We report the domain, availability of ground-truth labels, and the number of columns, tables, and data products in each universe.

Universe	Domain	Ground-truth	# Cols.	# Tabs.	# Data Prods.
Kaggle [3]	General	False	8,837	715	239
Harvard [2]	General	False	6,980	494	12
BiodivTab [1]	Biomedical	True	883	45	45

Table 4: Ontology construction statistics for FSTO-Gen and SemForest across all data universes. We report the number of annotated columns and generated semantic types at different levels (T-FST, P-FST, G-FST), as well as the number of semantic trees produced (if applicable).

Universe	Ontology	# Annot. Cols.	# T-FSTs	# P-FSTs	# G-FSTs	# Sem. Trees
Kaggle [3]	FSTO-Gen	7,861	5,285	3,775	2,622	–
	SemForest	8,198	–	4,254	3,241	1,485
Harvard [2]	FSTO-Gen	6,451	3,410	2,747	2,477	–
	SemForest	5,372	–	3,497	3,375	1,640
BiodivTab [1]	FSTO-Gen	784	663	663	274	–
	SemForest	805	–	802	300	80

Algorithm 2 Recursive DBSCAN clustering.

Require: Embedding matrix E, max cluster size k, epsilon schedule $\boldsymbol{\varepsilon}$

1: **procedure** RecursiveDBSCAN(E, k, $\boldsymbol{\varepsilon}$)
2: **if** $\boldsymbol{\varepsilon}$ is empty **then**
3: **return** {indices(E)} ▷ Return as one cluster
4: **end if**
5: $\varepsilon \leftarrow$ first($\boldsymbol{\varepsilon}$)
6: Apply DBSCAN on E with parameter ε
7: Initialize $C \leftarrow \emptyset$ ▷ Cluster result set
8: **for all** cluster c from DBSCAN **do**
9: **if** $|c| > k$ **then**
10: $E_c \leftarrow E[c]$
11: $C \leftarrow C \cup$ RecursiveDBSCAN(E_c, k, $\boldsymbol{\varepsilon}[2:]$)
12: **else**
13: $C \leftarrow C \cup \{c\}$
14: **end if**
15: **end for**
16: **return** C
17: **end procedure**

A DATASET AND ONTOLOGY STATISTICS

Table 3 and Table 4 present key statistics for the three dataset universes used in our experiments, Kaggle, Harvard, and BiodivTab, along with the corresponding ontologies constructed by FSTO-Gen and SemForest. Table 3 summarizes dataset-level characteristics, including domain, availability of ground-truth labels, and the number of columns, tables, and data products. Table 4 reports ontology-level metrics, such as the number of annotated columns and the number of generated FSTs at different levels of abstraction (T-FSTs, P-FSTs, and G-FSTs), as well as the number of semantic trees.

B RECURSIVE DBSCAN CLUSTERING ALGORITHM

Algorithm 2 outlines the recursive clustering procedure used to organize semantically related G-FSTs into coherent groups. The algorithm operates on the embedding matrix E of all G-FSTs, and takes as input a maximum cluster size k and a schedule of ϵ values $\boldsymbol{\varepsilon}$ for the DBSCAN algorithm.

The procedure begins by applying DBSCAN with the first value in the $\boldsymbol{\varepsilon}$ schedule, a list of ϵ values ordered from large to small (Line 5–6). If the resulting cluster exceeds the size threshold k (Line 9), the algorithm recursively applies DBSCAN to that cluster using the next (smaller) ϵ in the schedule (Line 11). This process continues until all clusters are size-bounded and semantically tight.

If the current $\boldsymbol{\varepsilon}$ schedule is exhausted, the remaining set is returned as a single cluster (Line 3). Otherwise, clusters that satisfy the size constraint are directly added to the result set (Line 14). The recursion ensures progressively finer semantic partitions while avoiding excessive fragmentation or loss of structure.

This controlled, recursive process enables the system to construct semantically compact clusters that serve as input units for downstream semantic tree construction.

C ALGORITHM COMPLEXITY ANALYSIS

For Algorithm 1, computing embeddings for n G-FSTs requires $O(n)$ calls to the embedding model and $O(n \cdot d)$ time and space to store the resulting embedding matrix, where d is the embedding dimension. The recursive clustering step (Line 4) calls Algorithm 2, which applies DBSCAN iteratively to clusters of varying sizes. Although each individual call to DBSCAN operates on a subset of the data, in the worst case when clustering fails to partition the input, the total cost can reach $O(n^2)$. In practice, recursive splitting significantly reduces the size of each subproblem, keeping the actual runtime much lower. Tree construction using the LLM (Line 7) is applied to clusters of bounded size k, and therefore scales linearly with the number of clusters.

D MORE EXPERIMENTS

Experiment 3: Effectiveness of Recursive Clustering. Table 5 compares the impact of different DBSCAN ϵ values with our recursive clustering across three data universes. We report the number of resulting clusters (i.e., semantic trees), the maximum cluster size, and the coverage, defined as the fraction of G-FSTs that were successfully integrated into a final semantic tree after LLM processing.

We observe that using a fixed ϵ introduces a trade-off between semantic cohesion and LLM compatibility. When ϵ is small (e.g., 0.1), clustering produces many tiny clusters (e.g., over 3,000 for both Kaggle and Harvard), each containing only a few types. This results in high coverage (near 100%) because LLMs can easily process small sets, but the resulting semantic trees are overly fragmented and semantically shallow, each tree contains, on average, fewer than 3 types, limiting their utility as an ontology. In contrast, a large ϵ (e.g., 0.6) yields a few coarse clusters containing thousands of types. These exceed the LLM's coherence limits. As a result,

Table 5: Comparison of DBSCAN ε settings and our recursive clustering strategy for semantic forest construction. We report the number of resulting clusters, maximum cluster size (number of types), and type coverage (percentage of G-FSTs included in trees) across all three universes.

DBSCAN ε	Kaggle			Harvard			BiodivTab		
	# clusters	max cluster	coverage	# clusters	max cluster	coverage	# clusters	max cluster	coverage
0.1	3,185	3	99.94%	3,299	5	99.97%	289	4	100.00%
0.35	1,193	1,165	65.26%	1,613	866	76.00%	101	37	97.33%
0.6	7	3,235	1.51%	13	3,362	1.84%	2	298	48.00%
Recursive	1,485	29	98.36%	1,640	29	98.37%	80	29	96.33%

coverage drops significantly (e.g., just 1.51% for Kaggle), since the LLM cannot consistently integrate such large sets into valid trees.

Our recursive clustering method addresses this trade-off by applying DBSCAN iteratively with a decreasing ϵ schedule and enforcing a maximum cluster size of 30. This strikes a balance: clusters are semantically meaningful yet small enough for the LLM to process effectively. As shown, recursive clustering yields high coverage across all datasets (above 95%) while maintaining manageable cluster sizes (e.g., max 29 types). The resulting trees are both structurally coherent and semantically rich, leading to a more interpretable and functionally useful ontology.