# Sentence to Model: Cost-Effective Data Collection LLM Agent

Yael Einy
Tel Aviv University
Israel
yaeleiny@mail.tau.ac.il

Guy Dar
Tel Aviv University
Israel
guy.dar@cs.tau.ac.il

Slava Novgorodov
Tel Aviv University
Israel
slavanov@post.tau.ac.il

Tova Milo
Tel Aviv University
Israel
milo@cs.tau.ac.il

## Abstract

We introduce *Sentence-to-Model*, an automated system that converts natural language queries into tabular datasets and predictive models. The system utilizes LLM agents for planning and active learning to prioritize data collection under budget constraints, such as API costs and rate limits. By integrating resources, it generates datasets and trains machine learning models with minimal human intervention. *Sentence-to-Model* streamlines data collection and enables the refinement of user needs and easy exploration of the data landscape. Our approach highlights the potential of combining LLMs with algorithmic techniques for efficient data science workflows.

## CCS Concepts

• **Information systems** → *Information integration.*

## Keywords

Data Enrichment, Data Integration, Large Language Models

**ACM Reference Format:**
Yael Einy, Guy Dar, Slava Novgorodov, and Tova Milo. 2025. Sentence to Model: Cost-Effective Data Collection LLM Agent. In *Companion of the 2025 International Conference on Management of Data (SIGMOD-Companion '25), June 22–27, 2025, Berlin, Germany*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3722212.3725134

## 1 Introduction

Data collection is the backbone of data science and machine learning. Unfortunately, it requires expertise and reasoning capabilities which classical algorithms struggle to provide. Often, relevant data is available online or inside the organizational internal network, albeit in an unstructured or unorganized way. In recent years, with the advent of large language models (LLMs), new opportunities have been unlocked. To enable LLMs to act autonomously in an external environment, LLM agents [9] emerged. LLM agents use LLMs as a planning faculty and use *tools* – search, retrieval, code execution, and other – as their interface with the external world.

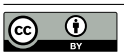
We present *Sentence-to-Model*, a new automated architecture for data collection based on LLM agents. The architecture operates by exploring structured and unstructured sources on the web, in knowledge graphs, and/or by searching in an internal network of an organization. Our system covers a broad family of data collection tasks (see Section 2) and is easily extendable. The user inputs a data collection request (e.g., "*create a dataset of NBA players for predicting their salaries*") and the system outputs two artifacts: the final dataset and a machine learning model trained on the dataset. The proposed architecture allows the user to travel *automagically* from a data science need to the end of a data science process in the minimal human effort possible.

Given a user query, our system invokes a pipeline with four components, as is depicted in Figure 1. *The Explorer* is a multi-agent system that explores the web or an organizational network for sources to scrape information from. Then, the system moves to the enrichment step. As many data sources are incomplete, we need to enrich certain rows to improve prediction. The enrichment step consists of two components. *The Prioritizer* prioritizes the entities and returns an ordering of them, ranked by order of importance. *The Extractor* then uses the prioritized list to choose the entities to augment, processing one entity at a time. After enrichment, in the *Modeling* stage, a model is automatically trained on the dataset.

The system operates under budgetary constraints for three reasons. First, certain websites have rate limits that prevent scrapers from sending unlimited queries. If the website contains one page per entity, we need to choose which entities to extract. Second, the extraction of unstructured data, such as text (for instance, an entity's Wikipedia page), is difficult to streamline. To obtain these features robustly we call an LLM. To keep costs in check, we must set a cap on the number of calls allowed. Third, we strive to minimize time costs. For useful deployment, our system must be optimally responsive and return artifacts in the minimal time possible.

Our system marries the flexible nature of LLMs with algorithmic approaches from data science. Moreover, other LLM systems might also benefit from a similar marriage. Prioritization is important for planning at large, and variants of our approach can be useful for information-seeking agents in other settings.

The artifacts can be used as-is if the output satisfies the user. Otherwise, the method can serve to test data hypotheses. The accelerated data collection process opens up new opportunities for data scientists to perform *data collection exploration* – a symbiotic human-machine process of back-and-forth experimentation, which helps identify the appropriate compromise between data needs and available data. We can map out where human effort is still required and/or refine user needs. Finally, the system can provide a summary




*SIGMOD-Companion '25, June 22–27, 2025, Berlin, Germany*

ACM ISBN 979-8-4007-1564-8/2025/06
https://doi.org/10.1145/3722212.3725134

of artifacts, keep track of data provenance, and propose pointers for further exploration by manual or automated work.

*Demonstration Scenario.* Our demonstration will showcase the application of our system for creating a tabular dataset and training an XGBoost model. The audience will act as end-users, experiencing the process from querying the system to analyzing results. The demonstration consists of two parts: (1) querying the system with predefined or custom queries, such as *"predict All-Star Award recipients among NBA players"*, and (2) interacting with a dashboard to explore the generated dataset, statistical summaries, and the model's performance metrics.

**Algorithm 1** Planner

**Require:** User query of data collection task $Q$
**Ensure:** Initial table $\mathcal{T}$ and actionable insights $\mathcal{I}$
1: Initialize data source and insight registries $D, I \leftarrow \emptyset$
2: Initialize search history with user query $H \leftarrow \{Q\}$
3: **while not** *exploration_complete*(H) **do**
4: Refine plan and return tool and query: $T, q \leftarrow \textsc{Plan}(H)$
5: Execute search and get source candidates $C = T(q)$
6: Get $(H', D', I') \leftarrow \textsc{Worker}(C)$
7: Update search history and registries $H, D, I$ by appending $H', D', I'$.
8: **end while**
9: Process $H, I$ and decide on final set of insights $\mathcal{I}$
10: Produce table $\mathcal{T}$ by joining or choosing a dataset in $D$
11: **return** Consolidated table $\mathcal{T}$ and insights $\mathcal{I}$

**Algorithm 2** Worker

**Require:** Source candidates to explore $C$ from Planner
**Ensure:** Summary $\mathcal{S} = (H', D', I')$ of search journey
1: **for** each candidate source $d \in C$ **do**
2: Navigate links and explore related content from $d$
3: **if** $d$ is batch-processable **then**
4: Generate and run extraction script $\textsc{Extract}(d)$
5: Append to $D'$ a tuple of the source's path, description, and extracted dataset (`path`, `desc`, `dataset`)
6: **else**
7: Append to $I'$ an entry (`path`, `description`)
8: **end if**
9: **end for**
10: Summarize search journey and save to $H'$
11: **return** Summary $\mathcal{S} = (H', D', I')$

## 2 Setup

*Input Query.* The query specifies a tabular data collection task in natural language. For concreteness, we concentrate on the family of data collection tasks that can be described by a 2-tuple $(D, T)$, where $D$ is a domain of entities (e.g., *"Basketball players"*), and $T$ is a target feature (e.g., *"salary"*). Each entity is represented by exactly one row in the dataset. The query is stated in natural language rather than a 2-tuple for user convenience at no cost (as the system operates in natural language regardless). This category contains a large part of the natural tabular data collection tasks. Filtering criteria can be absorbed into the entity description (e.g., *"Basketball players born after 1980"*). Target transformations can be absorbed into the target description (*"Player's weight-to-height ratio"*). Our system is easily adaptable to data collection tasks beyond this scope.

**Algorithm 3** Extractor

**Require:** Prioritized list of entities $E = \{e_1, e_2, \ldots, e_n\}$, insights $\mathcal{I}$, initial table $\mathcal{T}$, budget $B \in \mathbb{N}$
**Ensure:** Updated table $\mathcal{T}$
1: Set remaining budget $b \leftarrow B$
2: **for** each entity $e_i \in E$ **and** $b > 0$ **do**
3: Select and execute extraction tactic:
- **Entangled Sources**: Extract features using `path` and `description`
- **Search Tools**: Search for answer for specific question on the entity
- **LLM Knowledge**: Use large language models as a fallback
4: Update $\mathcal{T}$ with results from $e_i$
5: Incorporate new insights from the task execution: $I' \leftarrow I' \cup \{I(e_i)\}$
6: Decrement budget: $b \leftarrow b - \texttt{cost}(e_i)$
7: **end for**
8: **return** updated table $\mathcal{T}$

*Data Sources.* Data sources are divided into two broad categories. Batched Sources are sources where one page (one web page, one SPARQL query, etc.) contains features for many entities *simultaneously*. For example, we can generate a DBPedia [2] query that returns a table with all NBA players and their features. Other examples include a single webpage that contains a table with multiple entities. These sources are the most economical, as features for many entities can be extracted at once. When the Explorer finds batched sources, it will extract them right away. To avoid relying on the LLM's ability to extract features directly, the LLM generates a scraping code tailored for the source. In the case of a knowledge graph, the LLM generates the SPARQL query.

Entangled Sources, on the other hand, assume "meta-structure". In entangled sources, information is spread across multiple pages, and each entity requires access to a separate page. For example, IMDB contains one page per movie. The Explorer does not extract directly from entangled sources, as this is costly. Instead, it keeps track of them and leaves them for the Extractor component, which will refer to specific pages in the source according to prioritization.

## 3 Technical Overview

### 3.1 Explorer

The Explorer uses a multi-agent setup to explore knowledge graphs, web resources, or an organizational network for sources to scrape information. The flexibility of natural language allows agents to adapt easily to different environments, as opposed to traditional software. The Explorer has access to three (types of) tools. *Search*

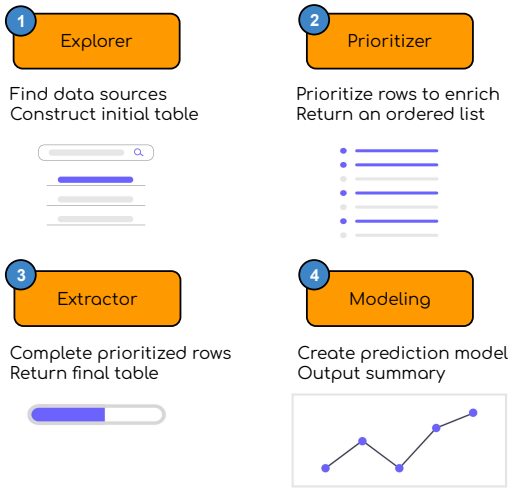



**Figure 1: System Overview**

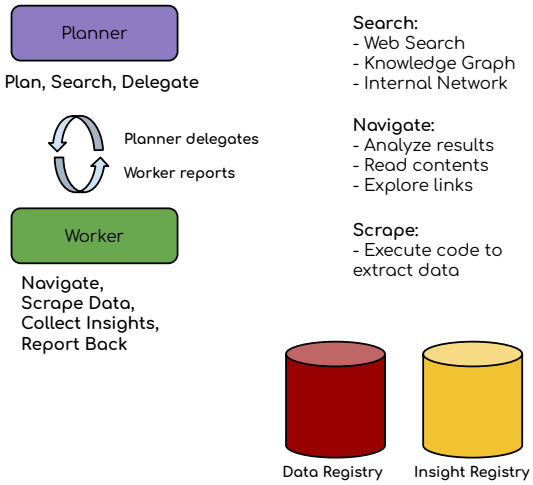



**Figure 2: Explorer Overview**

*Tools*, including web search (natural language keywords), knowledge graph search (SPARQL queries), and/or an internal network search engine. The next set of tools is *Navigation*. Under navigation, we put the abilities of the agent to interact with the content it retrieved. The agent can read the contents of a webpage and/or navigate to another (linked or related) webpage. *Scraping* – a restricted Python interpreter is provided to the system to execute scraping scripts. The LLM will not extract the information but will write a scraping script, as this process is more reliant.

The Explorer's architecture consists of two agents: The *Planner* and its *Workers*. The Planner chooses which actions to take (where and what to search). The Worker analyzes the results and sends a summary of its exploration to the Planner. The summary is appended to the Planner's history, which based upon the exploration history, outputs a new query. The separation of concerns between the Planner and the Workers has two benefits: (a) It keeps the Planner's history short by containing only summaries of explorations; (b) It allows the Workers to focus on their specific task, which renders them less prone to distraction. Additionally, the Planner can spawn multiple Workers in parallel for better efficiency. Schematic algorithms are provided in Algorithms 1 & 2 is provided. A birdseye view of the Explorer is given in Figure 2.

## 3.2 Enrichment

The enrichment process consists of two components with complementary roles. A prioritization algorithm that decides on an ordering of entities and an extraction agent that executes feature extraction based on the prioritized list.

*The Prioritizer* is a prioritization algorithm that decides which entities are most urgent to explore further. Our algorithm is a variant of the method presented in [6]. In this paper, we will not elaborate on the algorithm. In broad strokes, the algorithm trains a decision-tree model and uses the model's uncertainty about its prediction to quantify importance. The more uncertain the model is about an entity, the higher its urgency. The process incorporates an online feedback loop to refine the prioritization dynamically.

*The Extractor* follows the prioritized list to choose which entities to augment first. It uses a best-effort approach to extract features for one entity at a time, beginning from the most prioritized entities, going down the list until its set budget is exhausted. It has a list of strategies for handling the extraction of features, which may differ by entity. The algorithm for the Extractor is given in Algorithm 3.

Once the enrichment stage is done, an automatic machine learning pipeline is run on the final dataset. The model splits the data into train and validation sets and trains an XGBoost model on the training set and validates its performance on the validation dataset.

# 4 Additional Modules

Our proposed system supports extra modules that will not be presented as part of the demo. However, they are worth mentioning and are planned to be incorporated in future versions.

*Prediction Needs.* In this demo, we take a *dataset* question and output a prediction model. A further step is to start from a *particular* question (*"what will be Michael Jordan's salary next year?"*) and transform the problem into a dataset problem behind the scenes.

*Prediction via Generalization.* When data for a task is unavailable, we may cast the problem as a subset of a general problem. For example, to predict the election results in Narnia, train on data for other election races, and predict on the features of Narnia. All steps, including generalization strategies, can be automated with LLMs.

*The Waze of Data Collection.* We will request permission from users to track the journeys taken by Sentence-to-Model on their requests. By (manually and/or automatically) analyzing the crowd-sourced data, we will identify common pitfalls and strategies that we will incorporate into the system's design.

# 5 Related Work

Large language models (LLMs) are used in data science to automate tasks such as feature engineering. Frameworks like [5, 7] employ LLMs to generate features for tabular data learning, relying on

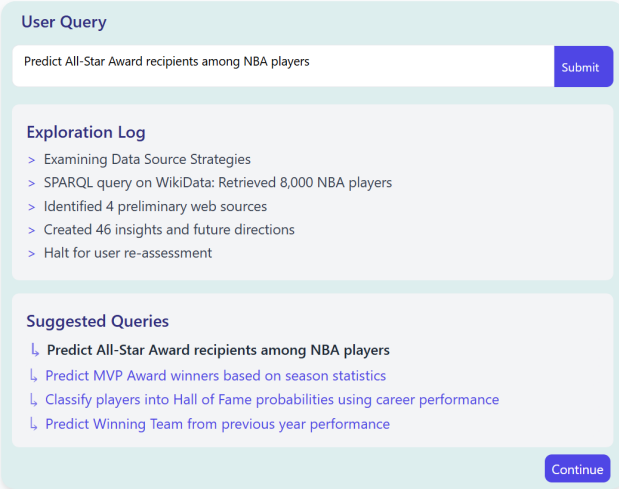



**Figure 3: User Interface**


**Figure 4: Results Dashboard**

LLM world knowledge and the existing features. However, these methods treat LLMs as standalone tools. Our approach integrates LLMs with external tools for reliable data collection iteratively, enabling cost-effective automated end-to-end data science.

LLM agents integrate language models with tool-based interactions, enabling autonomous task execution. ReAct [9] uses an LLM to decide on tool calls based on the outputs of previous calls. Plan-and-Solve [8] uses the LLM to devise a plan ahead of time to improve reasoning.

Works in AutoML for data science have largely neglected data collection [3]. Most works assume a closed-world use case and therefore can usually avoid open-ended exploration. Symphony [4] translates keywords into neural embeddings in order to search over a given data store. Evaporate [1] studies how an LLM can extract tabular data from unstructured sources in a pre-defined data lake. However, the process is designed by humans. It usually requires some pre-defined setup. Here, we seek to automate the exploration and design decisions from the problem statement alone.

## 6 Demonstration Scenario

To illustrate the application of our approach, we present a demonstration centered on creating a tabular dataset and an XGBoost model. The conference audience will play the role of a user who will benefit from the end-to-end experience of our system, starting from the initial query to the generated dataset and the trained model. The demonstration consists of two primary components: (1) a user querying the system and exploring alternative queries, and (2) an interactive dashboard for analyzing the generated dataset and model performance.

In the first part, the user initiates a query, such as *"predict All-Star Award recipients among NBA players"*. We will have a set of pre-defined queries to speed up the demonstration, but the audience will be able to write their own queries. Figure 3 provides a schematic view of this process. We can see a screenshot of an HTML page that captures the user input along with a concise log of key milestones. Additionally, the interface suggests alternative sibling queries that might be more suitable for further exploration, from a data availability point of view. The user can either proceed with the original query or select one of the alternatives.

Figure 4 depicts a dashboard that offers an overview of the generated output. This includes a "peek mode" view of the tabular dataset, general statistical summaries, and a birdseye visualization of the XGBoost model's performance metrics. The audience can interact with the dashboard to examine the data, interpret the model's outputs, and identify insights. Finally, the generated model can be exported as-is for future deployment in the real-world environment.

During the demonstration, we will explain the mechanisms under the hood of the system and through these components, we plan to highlights the workflow of constructing a data-driven model while facilitating user engagement and exploration.

## References

[1] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2023. Language models enable simple systems for generating structured views of heterogeneous data lakes. *arXiv preprint arXiv:2304.09433* (2023).

[2] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. DBpedia: a nucleus for a web of open data. In *Proceedings of the 6th International The Semantic Web and 2nd Asian Conference on Asian Semantic Web Conference*. Berlin, Heidelberg, 722–735.

[3] Rafael Barbudo, Sebastián Ventura, and José Raúl Romero. 2023. Eight years of AutoML: categorisation, review and trends. *Knowledge and Information Systems* 65, 12 (2023), 5097–5149.

[4] Zui Chen, Zihui Gu, Lei Cao, Ju Fan, Samuel Madden, and Nan Tang. 2023. Symphony: Towards Natural Language Query Answering over Multi-modal Data Lakes.. In *CIDR*.

[5] Yael Einy, Tova Milo, and Slava Novgorodov. 2024. Cost-Effective LLM Utilization for Machine Learning Tasks over Tabular Data. In *GUIDE-AI*. Association for Computing Machinery, New York, NY, USA, 45–49. https://doi.org/10.1145/3665601.3669848

[6] Aviv Hadar, Tova Milo, and Kathy Razmadze. 2025. Datamap-Driven Tabular Coreset Selection for Classifier Training. *VLDB* (2025).

[7] Noah Hollmann, Samuel Müller, and Frank Hutter. 2023. Large Language Models for Automated Data Science: Introducing CAAFE for Context-Aware Automated Feature Engineering. In *NeurIPS*.

[8] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. In *Annual Meeting of the Association for Computational Linguistics*.

[9] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. ReAct: Synergizing Reasoning and Acting in Language Models. *ArXiv* abs/2210.03629 (2022).