SemPiper: Interactive Code Synthesis for Semantic Operators in Machine Learning Pipelines

Olga Ovcharenko
BIFOLD & TU Berlin
ovcharenko@tu-berlin.de

Luciano Duarte
BIFOLD & TU Berlin
duarte.castineira@tu-berlin.de

Sebastian Schelter
BIFOLD & TU Berlin
schelter@tu-berlin.de

ABSTRACT

Machine learning (ML) pipelines require extensive data preparation, feature engineering, and integration across heterogeneous sources, making them tedious and error-prone to develop. While large language models (LLMs) have recently shown promise for assisting programming tasks, chat-based interfaces provide limited control over pipeline behavior and often produce code that is difficult to optimize or integrate into production systems. We demonstrate SemPipes, a novel programming model that extends ML pipelines with declarative, LLM-powered semantic data operators. SemPipes allows developers to specify high-level natural language instructions for data-centric operations, while seamlessly combining these operators with arbitrary Python code from standard data science libraries. For the semantic operators, it synthesizes specialized implementations at pipeline training time, conditioned on dataset characteristics and pipeline context, enabling the flexible yet controlled integration of LLM capabilities. We demonstrate SemPipes through SemPiper, an interactive interface that visualizes computational graphs of the pipelines, synthesized operator implementations, and optimization trajectories produced by an evolutionary search procedure. Attendees can explore three end-to-end scenarios, modify pipelines, inspect generated code, and observe how semantic operators are synthesized and iteratively optimized. The demonstration highlights how declarative semantic operators enable controllable, optimizable, and practical integration of LLMs into ML pipeline development.

PVLDB Reference Format:
Olga Ovcharenko, Luciano Duarte, and Sebastian Schelter. SemPiper: Interactive Code Synthesis for Semantic Operators in ML Pipelines. PVLDB, 19(12): 4538 - 4541, 2026.
doi:10.14778/3827998.3828060

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://olgaovcharenko.github.io/sempiper/.

1 INTRODUCTION

Large language models (LLMs) are reshaping software development through advances in code synthesis. Approaches range from AI-assistants for programming [1] to fully autonomous program generation [3, 5]. Despite their promise, these approaches raise concerns about control, safety, and unintended behavior [9], and may even reduce developer productivity [2]. A key limitation is that chat-based interaction only provides coarse control over the development process and produces code that is difficult to optimize.

ML Pipelines. The outlined issues are particularly important for iterative, data-centric, and repetitive machine learning (ML) pipeline development, which is central to ML research and practice [14]. In real-world settings, such pipelines operate on complex datasets originating from heterogeneous sources—including data lakes, warehouses, and log files, and must be joined, cleaned, augmented, and encoded with *ML pipelines* [12]. These pipelines encompass tasks such as prediction, integration, debugging, and postprocessing, combine complex logic across multiple libraries [13], and are costly to execute in production [15]. Building tabular ML pipelines is tedious and error-prone, requiring substantial domain expertise and engineering effort. Industry surveys identify data preparation as the most time-consuming and least enjoyable aspect of data science.

SemPipes: Extending ML Pipelines with Semantic Data Operators. Recent work in semantic query processing points to a promising direction: integrating LLM-powered operators into data processing systems. In this paradigm, SQL is extended with semantic operators that invoke LLMs directly within database engines, enabling queries over multimodal and unstructured data [6–8, 11]. Inspired by this line of work, we recently introduced SemPipes [10], a novel programming model that extends tabular ML pipelines with declarative, LLM-powered *semantic data operators* (`SemOps`). SemPipes pipelines are written in Python and selectively delegate data-centric operations to context-aware, optimizable semantic operators that generate task-specific Python code (Section 2). Following a declarative paradigm, SemPipes separates *what* should be computed from *how* it is executed: developers provide high-level natural language instructions for selected operations, while the system synthesizes their concrete implementations. During training, SemPipes uses LLMs to generate custom implementations for SemOps, conditioned on data characteristics, user instructions, and pipeline context. Therefore, SemPipes only requires a small number of LLM calls at training time, proportional to the number of semantic operators, and eliminates the reliance on LLMs during inference. This design enables semi-automated pipeline construction while preserving the ability to iteratively develop a pipeline in interactive notebook environments. Once defined, the operator implementations in a SemPipes pipeline can be further refined through LLM-based code mutation guided by evolutionary search to improve end-to-end predictive performance. In contrast to systems like LOTUS [11] and Palimpzest [8], which rely on LLM-based data processing, SemPipes leverages code generation.

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828060

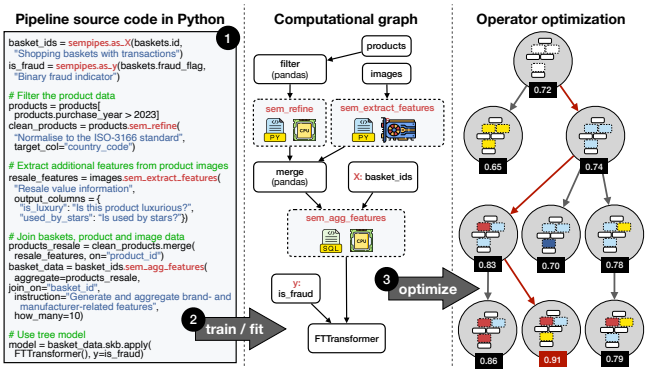

Figure 1: High-level overview of SemPipes. SemPipes extends standard Python ML pipelines with semantic data operators that delegate selected data-centric tasks to LLMs. During training, it synthesizes operator implementations from data, natural language instructions, and pipeline context, and can further optimize them on a validation set using evolutionary search guided by downstream model performance.

Demonstration Details. We present SemPiper, an interactive demo that "*plays* SemPipes" and exposes its internals through a dedicated user interface. We demonstrate SemPipes across three end-to-end ML scenarios with respect to fraud detection in e-commerce, integration of multimodal art data and prediction of house prices based on tables and images (Section 3). Each scenario emphasizes different types of semantic data operators, ranging from semantic feature generation and extraction to data cleaning and refinement. For each scenario, we present the pipeline code, its corresponding computational graph, and the integrated SemOps. Through the interactive SemPiper interface, attendees can observe how operator implementations are synthesized at training time with SemPipes, inspect the generated Python code, and examine how each implementation is conditioned on dataset characteristics and natural language instructions. Attendees can further modify pipeline code, adjust instructions, or even implement their own scenario. In addition to interactive code synthesis for a pipeline, we also showcase the evolutionary optimizer for SemOps in SemPipes. For that, SemPiper visualizes different optimization trajectories, including a search tree of candidate implementations, along with their validation performance and the evolutionary search procedure that selects improved variants. We provide the web-based user interface for our demonstration SemPiper, with all example pipelines and datasets, at https://olgaovcharenko.github.io/sempiper.

2 SYSTEM OVERVIEW

We briefly introduce SemPipes [10].

Overview. Figure 1 provides a brief overview of SemPipes: ❶ ML pipelines are defined in Python using standard data science libraries, including pandas for tabular data manipulation, NumPy for numerical computation, scikit-learn for feature encoding, and models compatible with the scikit-learn ecosystem. SemPipes extends these pipelines with *semantic data operators*, which *delegate selected data-centric operations to LLMs*. ❷ During training, when a pipeline is fitted to the training data, SemPipes *synthesizes custom implementations for the semantic operators* based on data, prompt, and pipeline context. The synthesized code may execute locally on the CPU or leverage a pretrained model on a local GPU. ❸ Once a pipeline is defined, *the implementations of its semantic operators can be tuned on a validation set*. SemPipes performs evolutionary search under a given search policy (e.g., Monte Carlo tree search), using downstream validation performance as the fitness function. During search, operator implementations are iteratively mutated via reflective prompting using prior performance scores and a tree-structured memory.

Table 1: Semantic data operators in SemPipes.

Category	Semantic Operators
Data Cleaning	`sem_fillna`, `sem_clean`, `sem_refine`
Feature Extraction	`sem_extract_features`, `sem_gen_features`,
& Generation	`sem_agg_features`
Data Augmentation	`sem_augment`
Parameterization	`sem_select`, `sem_choose`

Implementation. SemPipes builds on skrub DataOps pipelines,[1] a lightweight abstraction for multi-table ML pipelines over dataframes. DataOps represents pipelines as a computational graph of data operations and dependencies, enabling lazy execution and pipeline rewriting. Unlike standard scikit-learn pipelines that are restricted to a linear sequence of single-table transformations, the DataOps abstraction elevates the entire pipeline, including multi-table dataframe operations, into a unified scikit-learn-style predictor with `fit` and `predict` methods. SemPipes integrates semantic data operators as scikit-learn-style estimators within the skrub computational graph.

Operator-Specific Code Synthesis at Training Time. SemPipes synthesizes operator implementations at training time, conditioned on input data characteristics, natural language instructions, and pipeline context, including the downstream task and model type. Table 1 shows the complete list of `SemOps`. The synthesized `SemOps` code varies by operator type and user instruction: for feature generation, SemPipes produces CPU-based pandas code; for multimodal feature extraction, `SemOps` leverage pretrained models on a local GPU, e.g., HuggingFace models in zero-shot mode. This code synthesis approach only requires a few LLM calls at training time, whose number is linear in the number of semantic operators. During inference, the generated operator code is executed without any further LLM involvement. The operator-centric design enables fine-grained validation of synthesized code. Beyond syntax checking via parsing, SemPipes executes each operator on a data sample and verifies operator-specific output constraints.

Optimizing the `SemOps` in an ML Pipeline. In practice, improving the data operations of a tabular ML pipeline is an iterative, manual process: data scientists select a target operation (e.g., feature generation), rewrite its code, potentially integrate external libraries, and rerun the pipeline on a validation set to assess performance improvements. SemPipes automates this workflow through LLM-based mutation of synthesized operator code, guided by tree-structured evolutionary search (e.g., Monte Carlo Tree Search). During this optimization, the system generates a set of candidate implementations by mutating the current code, integrates each candidate into the pipeline, and evaluates the generated pipeline variant on a validation set. The search procedure prioritizes promising code variants based on their impact on downstream model performance.

[1] https://skrub-data.org/stable/data_ops.html

3 DEMONSTRATION DETAILS

SemPiper demonstrates SemPipes through three scenarios, each centered on a ML pipeline highlighting certain semantic data operators. We provide the source code, datasets, and the web-based SemPiper interface. We show each scenario as detailed in Figure 2:

(1) We introduce ❶ the scenario, the pipeline code, ❸ its corresponding computational graph, and semantic data operators.
(2) Attendees ❷ execute the pipeline with SemPipes to interactively synthesize its SemOps code by an LLM during training. Next, through the SemPiper interface, attendees inspect the ❹ generated Python code for each SemOp and how it is conditioned on data characteristics and natural language instructions.
(3) Then, attendees can modify the natural language instructions of the SemOps and rewrite the pipeline code to experiment with synthesizing new or improved operator implementations.
(4) Finally, ❺ we showcase the pipeline optimizer by visualizing candidate code for each SemOp, with their validation performance and the evolutionary search procedure that selects improved variants. ❻ Attendees can compare operator code, trace performance improvements, and inspect the cost, LLM calls, and time used for the whole pipeline and separate operators.

Interactive Code Synthesis. We detail our three scenarios.

(a) Fraud Detection on Multi-Table Retail Data. The first scenario is a fraud detection pipeline that trains a binary classifier to identify fraudulent shopping baskets. The pipeline uses two input tables: `baskets`, which contains fraud labels, and `products`, which describes the actual products purchased within each basket. The ML pipeline performs several preprocessing steps using popular Python libraries such as NLTK and unidecode for textual normalization and SciPy for price statistics. These steps are combined with semantic data operators, followed by a tree-based CatBoost classifier.

Demonstrated Semantic Operators. This scenario highlights four semantic operators. First, `sem_fillna` imputes missing product manufacturer values by inferring them from attributes such as the title and description. Second, `sem_extract_features` extracts additional information from product brand names (`is_luxury_brand` and `is_known_brand`, and `brand_risk_score`) indicating the potential brand's association with fraud. Third, `sem_gen_features` synthesizes additional features. Finally, `sem_agg_features` aggregates the product-level data into basket-level features. The basket-level features are used to train the tree-based fraud detection model.

Exemplary Synthesized Code. With Google's `gemini-2.5-flash` as backing LLM, SemPipes synthesizes operator code. First, `sem_fillna` evaluates several imputation strategies and selects `KNNImputer` as the best-performing model. Second, `sem_extract_features` applies `facebook/bart-large-mnli`, a zero-shot classification model, and `google/flan-t5-base`, a text generation model, to assign risk scores. Next, `sem_gen_features` calculates `avg_brand_cash_price` and `brand_frequency`. Finally, `sem_agg_features` computes aggregate features such as the total cash price per basket.

(b) Cultural Origin Prediction for Multimodal Museum Catalog Data. The second scenario demonstrates a museum artifact analytics pipeline that combines tabular data with unstructured textual descriptions to label the artworks from the Metropolitan Museum of Art with their cultural origin [4]. The pipeline operates on the `artworks` table containing attributes describing a piece of art, together with free-text descriptions. We first enrich the table with lightweight natural language processing features extracted using the spaCy library, including named entities (people, locations, cultural groups, and dates), noun phrases, and adjective density. Next, the pipeline applies several semantic data operators to transform the noisy museum metadata into structured features. In the end, the data is vectorized and used to train an FT-Transformer classifier, an adaptation of the Transformer architecture for tabular data.

Demonstrated Semantic Operators. This scenario evolves around three semantic operators. First, `sem_extract_features` converts heterogeneous date strings into a structured temporal representation. Second, `sem_gen_features` enriches the dataset with additional predictive features. Finally, `sem_refine` standardizes the raw museum object names to address inconsistencies and high-cardinality values to produce cleaner categorical representations.

Exemplary Synthesized Code. With Google's `gemini-2.5-flash` as backing LLM, SemPipes synthesizes operator code as follows. First, `sem_extract_features` extracts the date values `year_start`, `start_is_bce`, `year_end`, and `end_is_bce` by combining a HuggingFace model `google/flan-t5-small` with rule-based parsing. The synthesized code prompts the model to extract structured information from the raw date string, producing JSON fields such as century and start/end years. These outputs are then refined using regular expressions and deterministic rules to enforce precise interval conventions for centuries and BCE/CE ranges. Second, `sem_gen_features` removes irrelevant columns and creates new features from existing attributes, including text length features, temporal features about artwork duration, and features such as `has_country_info` and `has_artist_nationality_info`. Finally, `sem_refine` standardizes the raw museum object names using regular expressions and correspondence dictionaries.

(c) Price Prediction on Multimodal Real Estate Data. The third scenario is a house price prediction pipeline that combines tabular and image data to estimate the price of houses in California. The pipeline consumes three tables: `facts`, containing house-level attributes such as price, square footage, and room counts; `cities`, containing location; and `images`, containing image metadata and file paths. The regression task predicts house selling price. After joining the tables, the pipeline performs several manual preprocessing and feature engineering steps. The enriched dataset is then vectorized using the `TableVectorizer` from the skrub library and consumed by the tabular foundation model `TabPFNRegressor`. Finally, predicted prices are combined with the original data and analyzed using DuckDB to compute city-level statistics.

Demonstrated Semantic Operators. This scenario highlights three semantic operators. First, `sem_clean` detects and corrects anomalies in the `sqft` column. Second, `sem_extract_features` derives features from house images, focusing on the house exterior and using domain knowledge about housing and geographic context in California. Finally, `sem_gen_features` synthesizes predictive attributes by generating property-, city-, and location-level features.

Exemplary Synthesized Code. With Google's `gemini-2.5-flash` as backing LLM, SemPipes synthesizes operator implementations

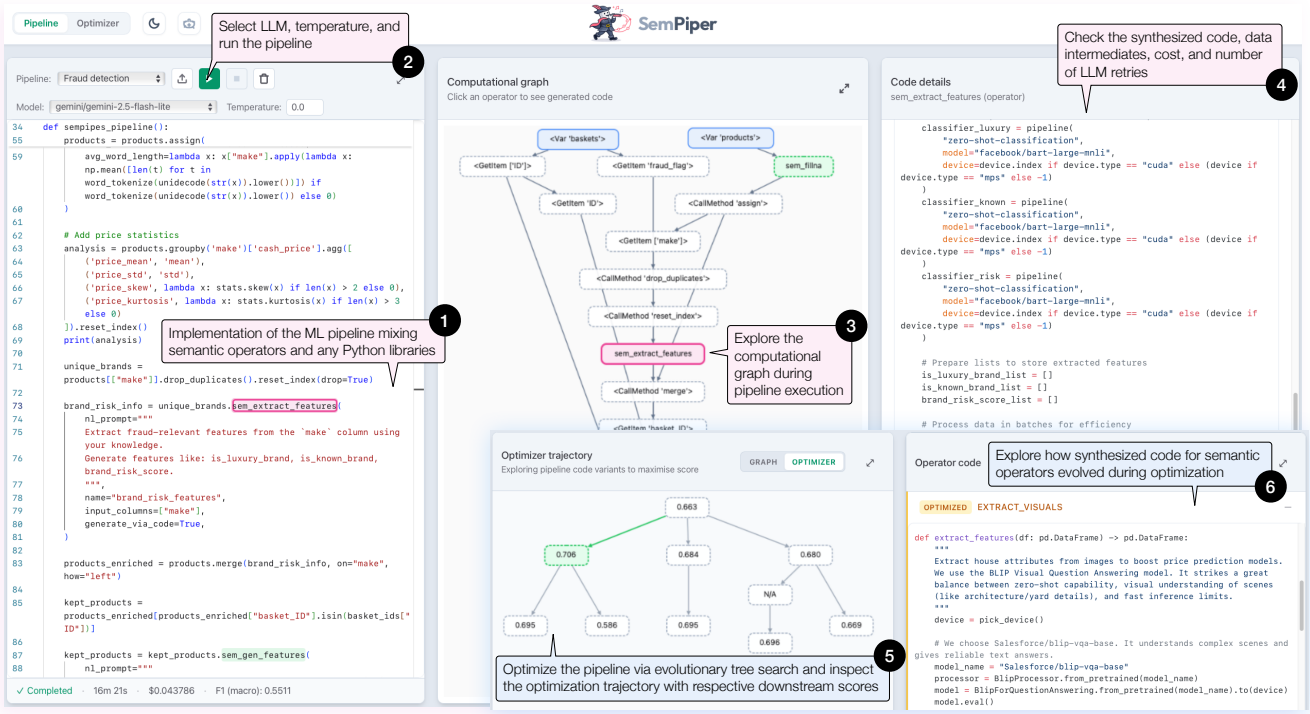

Figure 2: Web-based user interface of SemPiper: In the *Pipeline* tab, ❶ attendees load one of three provided pipelines or implement their own using any Python libraries. Next, ❷ attendees select an LLM and temperature and execute the pipeline. ❸ During execution, the computational graph of the pipeline can be interactively explored. ❹ Subsequently, attendees can deeply inspect the synthesized operator code, data intermediates, model performance, runtime, and cost of the pipeline. In the *Optimizer* tab, ❺ attendees explore evolutionarily generated and evaluated pipeline variations, including ❻ their evolved operator code and performance across optimizer iterations.

in this pipeline as follows. The code for `sem_clean` cleans the `sqft` column by removing outliers through IQR-based capping. The synthesized implementation of the `sem_extract_features` operator applies the popular visual question answering model `Salesforce/blip-vqa-base` to extract image-based attributes. Finally, `sem_gen_features` synthesizes four features (`total_rooms`, `sqft_per_bath`, `city_avg_sqft`, and `city_avg_bed`), which capture both property-level characteristics and local housing context.

Evolutionary Optimization of Semantic Operator Code. The final part of the demonstration centers on an interactive optimizer interface for exploring SemPipes' evolutionary tree search runs. The interface visualizes the search process under different strategies, such as Monte Carlo Tree Search, showing how candidate operator implementations are generated and evaluated over time. Each node in the search tree exposes the synthesized code and its predictive performance on the validation set, allowing attendees to trace how the optimizer navigates the search space and discovers on best-performing pipeline variants. For example, for a house price prediction task, attendees can observe how the optimizer leverages the `Salesforce/blip-vqa-base` visual question answering model and refines which features to extract. The best-performing pipeline extracts semantically rich features such as `number_of_stories`, `landscape_quality`, `architectural_style`, `garage_capacity`, and `driveway_size`, whereas weaker variants relied on coarser features like `has_visible_garage` and `exterior_condition`.

REFERENCES

[1] Anthropic. 2025. Claude Code: AI-powered coding assistant. https://www.claude.com/product/claude-code.
[2] Becker et al. 2025. Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity. arXiv:2507.09089
[3] Chan et al. MLE-bench: Evaluating Machine Learning Agents on Machine Learning Engineering. *ICLR'25*
[4] Duarte, et al. 2026. ArtiFact: A Large-Scale Multi-Modal Cultural Heritage Dataset. *VLDB Workshop: Novel Optimizations for Visionary AI Systems (NOVAS)*
[5] Fang et al. MLZero: A Multi-Agent System for End-to-end ML Automation. *NeurIPS'25*
[6] Google. 2025. Generative AI overview in BigQuery. https://docs.cloud.google.com/bigquery/docs/generative-ai-overview
[7] Liskowski et al. The Cortex AISQL Engine for Unstructured Data. *SIGMOD'26*
[8] Liu et al. Palimpzest: Optimizing AI-Powered Analytics with Declarative Query Processing. *CIDR'25*
[9] Nolan. 2025. AI-Powered Coding Tool Wiped Out a Software Company's Database in 'Catastrophic Failure'. https://fortune.com/2025/07/23/ai-coding-tool-replit-wiped-database-called-it-a-catastrophic-failure/.
[10] Ovcharenko et al. SemPipes – Optimizable Semantic Data Operators for Tabular Machine Learning Pipelines. arXiv:2602.05134
[11] Patel et al. Semantic Operators and Their Optimization: Enabling LLM-Based Data Processing with Accuracy Guarantees in LOTUS. *VLDB'25*
[12] Polyzotis et al. Data Management Challenges in Production ML. *SIGMOD'17*
[13] Psallidas et al. Data Science Through the Looking Glass: Analysis of Millions of GitHub Notebooks and ML.NET Pipelines. *SIGMOD Record'22*
[14] Breugel et al. Tabular Foundation Models Should Be a Research Priority. *ICML'24*
[15] Xin, et al. 2021. Production machine learning pipelines: Empirical analysis and optimization opportunities. *SIGMOD'21*