SemPipes: Optimizable Semantic Data Operators for Tabular Machine Learning Pipelines

NHR4CES Community Workshop 2026, May 12

Olga Ovcharenko Matthias Boehm Sebastian Schelter

DEEM Lab
RESPONSIBLE DATA ENGINEERING RESEARCH

BIFOLD

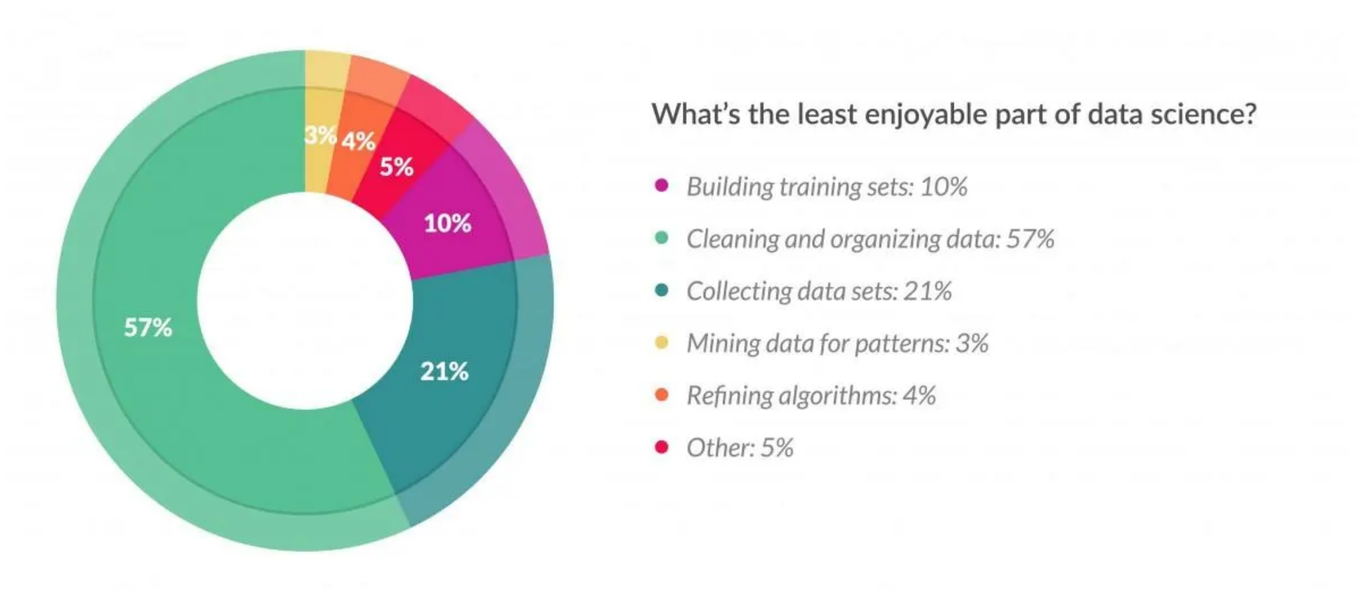
Technische Universität Berlin

About Me

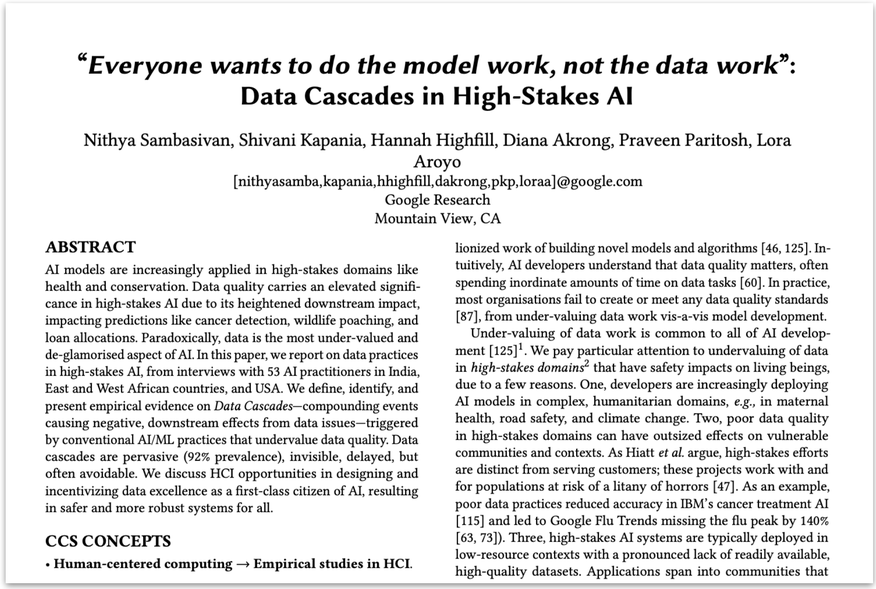
- Second year PhD student at DEEM Lab RESPONSIBLE DATA ENGINEERING RESEARCH, supervised by Prof. Schelter
- Studied before at ETH zürich and TU Graz

- Working on semantic operators in the context of ML and data management
- Prior experience on applied ML for genomics and development of ML systems internals

Data Preparation: The Least Enjoyable Part of Data Science



https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/



"*Everyone wants to do the model work, not the data work*": Data Cascades in High-Stakes AI

Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Paritosh, Lora Aroyo
[nithyasamba,kapania,hhighfill,dakrong,pkp,loraa]@google.com
Google Research
Mountain View, CA

ABSTRACT

AI models are increasingly applied in high-stakes domains like health and conservation. Data quality carries an elevated significance in high-stakes AI due to its heightened downstream impact, impacting predictions like cancer detection, wildlife poaching, and loan allocations. Paradoxically, data is the most under-valued and de-glamorised aspect of AI. In this paper, we report on data practices in high-stakes AI, from interviews with 53 AI practitioners in India, East and West African countries, and USA. We define, identify, and present empirical evidence on *Data Cascades*—compounding events causing negative, downstream effects from data issues—triggered by conventional AI/ML practices that undervalue data quality. Data cascades are pervasive (92% prevalence), invisible, delayed, but often avoidable. We discuss HCI opportunities in designing and incentivizing data excellence as a first-class citizen of AI, resulting in safer and more robust systems for all.

CCS CONCEPTS

• **Human-centered computing → Empirical studies in HCI.**

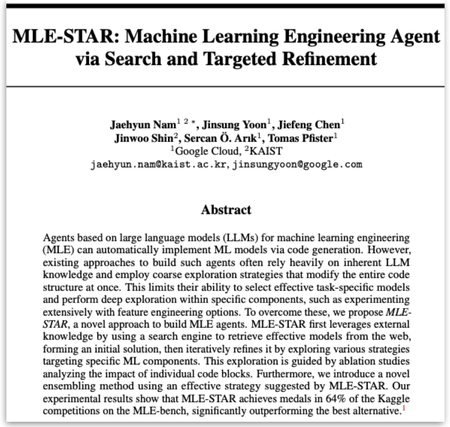
lionized work of building novel models and algorithms [46, 125]. Intuitively, AI developers understand that data quality matters, often spending inordinate amounts of time on data tasks [60]. In practice, most organisations fail to create or meet any data quality standards [87], from under-valuing data work vis-a-vis model development.

Under-valuing of data work is common to all of AI development [125][1]. We pay particular attention to undervaluing of data in *high-stakes domains*[2] that have safety impacts on living beings, due to a few reasons. One, developers are increasingly deploying AI models in complex, humanitarian domains, *e.g.*, in maternal health, road safety, and climate change. Two, poor data quality in high-stakes domains can have outsized effects on vulnerable communities and contexts. As Hiatt *et al.* argue, high-stakes efforts are distinct from serving customers; these projects work with and for populations at risk of a litany of horrors [47]. As an example, poor data practices reduced accuracy in IBM's cancer treatment AI [115] and led to Google Flu Trends missing the flu peak by 140% [63, 73]). Three, high-stakes AI systems are typically deployed in low-resource contexts with a pronounced lack of readily available, high-quality datasets. Applications span into communities that

"*Everyone wants to do the model work, not the data work*": Data Cascades in High-Stakes AI, CHI'21

ML Engineering Agents Will Write All Pipelines Instead of Us!

- LLM-based agents for the end-to-end machine learning engineering using code generation
- **Promises**
  - Automatic implementation of ML pipelines via code generation
  - State-of-the-art results on multiple machine learning engineering benchmarks
  - End-to-end ML automation across diverse data modalities with minimal human intervention
  - Comprehensive, LLM-based system for generating effective, error-free, and efficient data-centric ML pipelines



MLE-STAR: Machine Learning Engineering Agent via Search and Targeted Refinement

Jaehyun Nam[1 2 *], Jinsung Yoon[1], Jiefeng Chen[1]
Jinwoo Shin[2], Sercan Ö. Arık[1], Tomas Pfister[1]
[1]Google Cloud, [2]KAIST
jaehyun.nam@kaist.ac.kr, jinsungyoon@google.com

Abstract

Agents based on large language models (LLMs) for machine learning engineering (MLE) can automatically implement ML models via code generation. However, existing approaches to build such agents often rely heavily on inherent LLM knowledge and employ coarse exploration strategies that modify the entire code structure at once. This limits their ability to select effective task-specific models and perform deep exploration within specific components, such as experimenting extensively with feature engineering options. To overcome these, we propose *MLE-STAR*, a novel approach to build MLE agents. MLE-STAR first leverages external knowledge by using a search engine to retrieve effective models from the web, forming an initial solution, then iteratively refines it by exploring various strategies targeting specific ML components. This exploration is guided by ablation studies analyzing the impact of individual code blocks. Furthermore, we introduce a novel ensembling method using an effective strategy suggested by MLE-STAR. Our experimental results show that MLE-STAR achieves medals in 64% of the Kaggle competitions on the MLE-bench, significantly outperforming the best alternative.[1]

MLE-STAR: Machine Learning Engineering Agent via Search and Targeted Refinement, NeurIPS'25

MLZero: A Multi-Agent System for End-to-end Machine Learning Automation

Haoyang Fang, Boran Han, Nick Erickson, Xiyuan Zhang, Su Zhou, Anirudh Dagar, Jiani Zhang, Ali Caner Turkmen, Cuixiong Hu, Huzefa Rangwala, Ying Nian Wu, Bernie Wang, George Karypis

Amazon Web Services

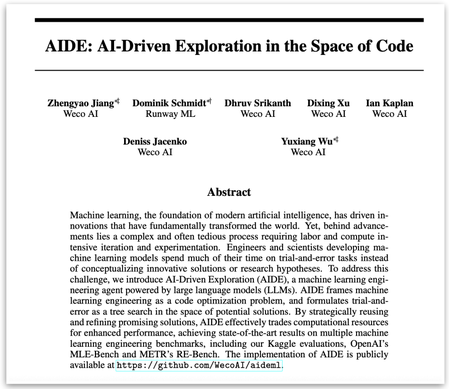
{haoyfang,boranhan,neerick,xiyuanz,suzhou,anidagar}@amazon.com
{zhajiani,atturkm,tonyhu,rhuzefa,wunyin,yuyawang,gkarypis}@amazon.com

Website: https://project-mlzero.github.io/
GitHub: https://github.com/autogluon/autogluon-assistant

Abstract

Existing AutoML systems have advanced the automation of machine learning (ML); however, they still require substantial manual configuration and expert input, particularly when handling multimodal data. We introduce MLZero, a novel multi-agent framework powered by Large Language Models (LLMs) that enables end-to-end ML automation across diverse data modalities with minimal human intervention. A cognitive perception module is first employed, transforming raw multimodal inputs into perceptual context that effectively guides the subsequent workflow. To address key limitations of LLMs, such as hallucinated code generation and outdated API knowledge, we enhance the iterative code generation process with semantic and episodic memory. MLZero demonstrates superior performance on MLE-Bench Lite, outperforming all competitors in both success rate and solution quality, securing six gold medals. Additionally, when evaluated on our Multimodal AutoML Agent Benchmark, which includes 25 more challenging tasks spanning diverse data modalities, MLZero outperforms the competing methods by a large margin with a success rate of 0.92 (+263.6%) and an average rank of 2.28. Our approach maintains its robust effectiveness even with a compact 8B LLM, outperforming full-size systems from existing solutions.

MLZero: A Multi-Agent System for End-to-end Machine Learning Automation, NeurIPS'25



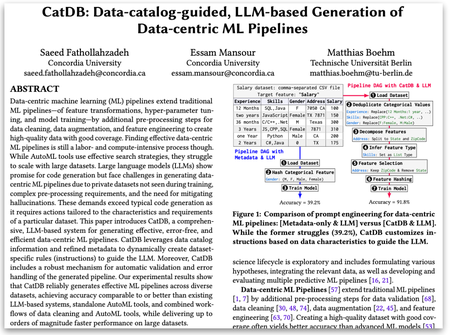
AIDE: AI-Driven Exploration in the Space of Code

Zhengyao Jiang[◁] Weco AI
Dominik Schmidt[◁] Runway ML
Dhruv Srikanth Weco AI
Dixing Xu Weco AI
Ian Kaplan Weco AI
Deniss Jacenko Weco AI
Yuxiang Wu[◁] Weco AI

Abstract

Machine learning, the foundation of modern artificial intelligence, has driven innovations that have fundamentally transformed the world. Yet, behind advancements lies a complex and often tedious process requiring labor and compute intensive iteration and experimentation. Engineers and scientists developing machine learning models spend much of their time on trial-and-error tasks instead of conceptualizing innovative solutions or research hypotheses. To address this challenge, we introduce AI-Driven Exploration (AIDE), a machine learning engineering agent powered by large language models (LLMs). AIDE frames machine learning engineering as a code optimization problem, and formulates trial-and-error as a tree search in the space of potential solutions. By strategically reusing and refining promising solutions, AIDE effectively trades computational resources for enhanced performance, achieving state-of-the-art results on multiple machine learning engineering benchmarks, including our Kaggle evaluations, OpenAI's MLE-Bench and METR's RE-Bench. The implementation of AIDE is publicly available at https://github.com/WecoAI/aideml.

AIDE: AI-Driven Exploration in the Space of Code



CatDB: Data-catalog-guided, LLM-based Generation of Data-centric ML Pipelines

Saeed Fathollahzadeh
Concordia University
saeed.fathollahzadeh@concordia.ca

Essam Mansour
Concordia University
essam.mansour@concordia.ca

Matthias Boehm
Technische Universität Berlin
matthias.boehm@tu-berlin.de

ABSTRACT

Data-centric machine learning (ML) pipelines extend traditional ML pipelines—of feature transformations, hyper-parameter tuning, and model training—by additional pre-processing steps for data cleaning, data augmentation, and feature engineering to create high-quality data with good coverage. Finding effective data-centric ML pipelines is still a labor- and compute-intensive process though. While AutoML tools use effective search strategies, they struggle to scale with large datasets. Large language models (LLMs) show promise for code generation but face challenges in generating data-centric ML pipelines due to private datasets not seen during training, complex pre-processing requirements, and the need for mitigating hallucinations. These demands exceed typical code generation as it requires actions tailored to the characteristics and requirements of a particular dataset. This paper introduces CatDB, a comprehensive, LLM-based system for generating effective, error-free, and efficient data-centric ML pipelines. CatDB leverages data catalog information and refined metadata to dynamically create dataset-specific rules (instructions) to guide the LLM. Moreover, CatDB includes a robust mechanism for automatic validation and error handling of the generated pipeline. Our experimental results show that CatDB reliably generates effective ML pipelines across diverse datasets, achieving accuracy comparable to or better than existing LLM-based systems, standalone AutoML tools, and combined workflows of data cleaning and AutoML tools, while delivering up to orders of magnitude faster performance on large datasets.

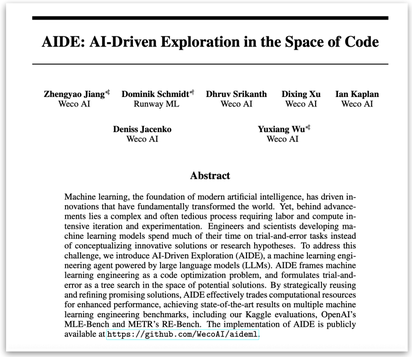
Figure 1: Comparison of prompt engineering for data-centric ML pipelines: [Metadata-only & LLM] versus [CatDB & LLM]. While the former struggles (39.2%), CatDB customizes instructions based on data characteristics to guide the LLM.

science lifecycle is exploratory and includes formulating various hypotheses, integrating the relevant data, as well as developing and evaluating multiple predictive ML pipelines [16, 21].

Data-centric ML Pipelines [57] extend traditional ML pipelines [1, 7] by additional pre-processing steps for data validation [68], data cleaning [30, 48, 74], data augmentation [22, 43], and feature engineering [63, 70]. Creating a high-quality dataset with good coverage often yields better accuracy than advanced ML models [53].

CatDB: Data-catalog-guided, LLM-based Generation of Data-centric ML Pipelines, VLDB'24

Will ML Engineering Agents Write All Pipelines Instead of Us?



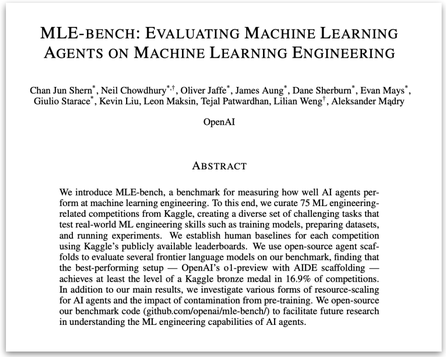
AIDE: AI-Driven Exploration in the Space of Code

Zhengyao Jiang[◁] Weco AI · Dominik Schmidt[◁] Runway ML · Dhruv Srikanth Weco AI · Dixing Xu Weco AI · Ian Kaplan Weco AI · Denis Jacenko Weco AI · Yuxiang Wu[◁] Weco AI

Abstract

Machine learning, the foundation of modern artificial intelligence, has driven innovations that have fundamentally transformed the world. Yet, behind advancements lies a complex and often tedious process requiring labor and compute intensive iteration and experimentation. Engineers and scientists developing machine learning models spend much of their time on trial-and-error tasks instead of conceptualizing innovative solutions or research hypotheses. To address this challenge, we introduce AI-Driven Exploration (AIDE), a machine learning engineering agent powered by large language models (LLMs). AIDE frames machine learning engineering as a code optimization problem, and formulates trial-and-error as a tree search in the space of potential solutions. By strategically reusing and refining promising solutions, AIDE effectively trades computational resources for enhanced performance, achieving state-of-the-art results on multiple machine learning engineering benchmarks, including our Kaggle evaluations, OpenAI's MLE-Bench and METR's RE-Bench. The implementation of AIDE is publicly available at `https://github.com/WecoAI/aideml`.

AIDE: AI-Driven Exploration in the Space of Code



MLE-BENCH: EVALUATING MACHINE LEARNING AGENTS ON MACHINE LEARNING ENGINEERING

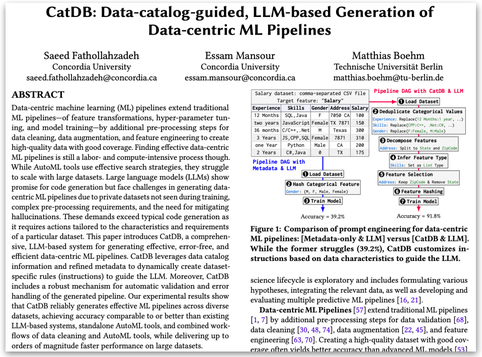
Chan Jun Shern*, Neil Chowdhury*,†, Oliver Jaffe*, James Aung*, Dane Sherburn*, Evan Mays*, Giulio Starace*, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng†, Aleksander Mądry

OpenAI

ABSTRACT

We introduce MLE-bench, a benchmark for measuring how well AI agents perform at machine learning engineering. To this end, we curate 75 ML engineering-related competitions from Kaggle, creating a diverse set of challenging tasks that test real-world ML engineering skills such as training models, preparing datasets, and running experiments. We establish human baselines for each competition using Kaggle's publicly available leaderboards. We use open-source agent scaffolds to evaluate several frontier language models on our benchmark, finding that the best-performing setup — OpenAI's o1-preview with AIDE scaffolding — achieves at least the level of a Kaggle bronze medal in 16.9% of competitions. In addition to our main results, we investigate various forms of resource-scaling for AI agents and the impact of contamination from pre-training. We open-source our benchmark code (github.com/openai/mle-bench/) to facilitate future research in understanding the ML engineering capabilities of AI agents.

MLE-Bench: Evaluating Machine Learning Agents on Machine Learning Engineering, ICLR'25



CatDB: Data-catalog-guided, LLM-based Generation of Data-centric ML Pipelines

Saeed Fathollahzadeh Concordia University saeed.fathollahzadeh@concordia.ca · Essam Mansour Concordia University essam.mansour@concordia.ca · Matthias Boehm Technische Universität Berlin matthias.boehm@tu-berlin.de

ABSTRACT

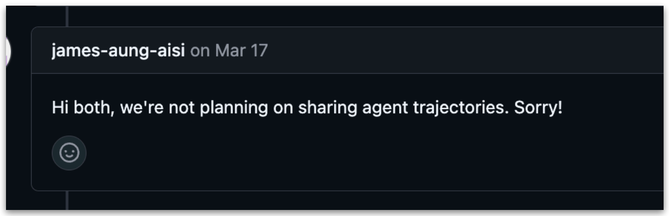
Data-centric machine learning (ML) pipelines extend traditional ML pipelines—of feature transformations, hyper-parameter tuning, and model training—by additional pre-processing steps for data cleaning, data augmentation, and feature engineering to create high-quality data with good coverage. Finding effective data-centric ML pipelines is still a labor- and compute-intensive process though. While AutoML tools use effective search strategies, they struggle to scale with large datasets. Large language models (LLMs) show promise for code generation but face challenges in generating data-centric ML pipelines due to private datasets not seen during training, complex pre-processing requirements, and the need for mitigating hallucinations. These demands exceed typical code generation as it requires actions tailored to the characteristics and requirements of a particular dataset. This paper introduces CatDB, a comprehensive, LLM-based system for generating effective, error-free, and efficient data-centric ML pipelines. CatDB leverages data catalog information and refined metadata to dynamically create dataset-specific rules (instructions) to guide the LLM. Moreover, CatDB includes a robust mechanism for automatic validation and error handling of the generated pipeline. Our experimental results show that CatDB reliably generates effective ML pipelines across diverse datasets, achieving accuracy comparable to or better than existing LLM-based systems, standalone AutoML tools, and combined workflows of data cleaning and AutoML tools, while delivering up to orders of magnitude faster performance on large datasets.

Figure 1: Comparison of prompt engineering for data-centric ML pipelines: [Metadata-only & LLM] versus [CatDB & LLM]. While the former struggles (39.2%), CatDB customizes instructions based on data characteristics to guide the LLM.

science lifecycle is exploratory and includes formulating various hypotheses, integrating the relevant data, as well as developing and evaluating multiple predictive ML pipelines [16, 21].

Data-centric ML Pipelines [57] extend traditional ML pipelines [1, 7] by additional pre-processing steps for data validation [68], data cleaning [30, 48, 74], data augmentation [22, 45], and feature engineering [63, 70]. Creating a high-quality dataset with good cov-

CatDB: Data-catalog-guided, LLM-based Generation of Data-centric ML Pipelines, VLDB'24



Intransparent Experiments

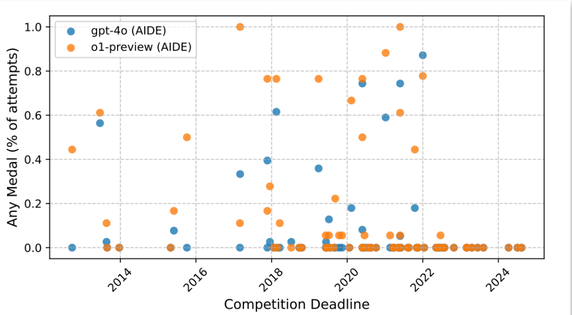


Figure 9: The percentage of attempts where models achieved any medal on each competition, plotted against each competition's end date. We find that models score medals on competitions between 2013 to 2022, but struggle to score medals on the more recent (and often more challenging) competitions after 2022.

```
# Combine train and test data for preprocessing
combined_data = pd.concat([train_data, test_data], ignore_index=True)

# Preprocessing: Fill missing values and one-hot encode categorical features
for col in combined_data.columns:
    if combined_data[col].dtype == "object":
        combined_data[col] = combined_data[col].fillna(combined_data[col].mode()[0])
        le = LabelEncoder()
```

Data Leakage

Performance Drop for Tasks Published after LLM Training Cutoff Date

What Would Be Nice To Have?

- An easy and robust way to control which parts of the pipeline
  - To write **by hand**
  - To generate & optimize **via an LLM**

- A runtime system that synthesizes & optimizes the code mimicking manual optimization by a data scientist
  - Users can **declaratively** specify the operations to delegate to LLMs
  - LLM generates **implementations tailored** to the users **pipeline**

Declarative Programming with Semantic Operators

- **Core idea:** Users **embed semantic operators with NL instructions** into their **code** to define **what** to compute, the underlying system decides **how** to compute it
  - LLMs in SQL operators

```
SELECT COUNT(*)
FROM Cars
WHERE AI.IF(pic, 'the picture shows a red car');
```

SemBench: A Benchmark for Semantic Query Processing Engines, VLDB'26

```
def paper_digest(research_interest: str, baseline: str):
  return papers_df\
    .sem_search("abstracts", research_interest, K=100)\
    .sem_filter(f"the paper {{abstracts}} claim to
     outperform {baseline}")\
    .sem_agg(f"Write a digest summarizing {{abstracts}}
     and their relevance to {research_interest}")
```

Semantic Operators and Their Optimization: Enabling LLM-Based Data Processing with Accuracy Guarantees in LOTUS, VLDB'24

  - Modular AI software automating the optimization of prompt construction

```
import dspy
from dspy.datasets import HotPotQA

dspy.configure(lm=dspy.LM("openai/gpt-4o-mini"))

def search_wikipedia(query: str) -> list[str]:
    results = dspy.ColBERTv2(url="http://20.102.90.50:2017/wiki17_abstracts")(query, k=3)
    return [x["text"] for x in results]

trainset = [x.with_inputs('question') for x in HotPotQA(train_seed=2024, train_size=500).train]
react = dspy.ReAct("question -> answer", tools=[search_wikipedia])

tp = dspy.MIPROv2(metric=dspy.evaluate.answer_exact_match, auto="light", num_threads=24)
optimized_react = tp.compile(react, trainset=trainset)
```

DSPy: Compiling Declarative Language Model Calls into State-of-the-Art Pipelines, ICLR'24

SemPipes

- **SemPipes** introduces **AI-assisted primitives** for writing (tabular) ML pipelines
- A novel declarative programming model that integrates **LLM-powered optimizable semantic data operators**
  - A new execution model: Training-time code synthesis, inference-time execution
  - Leverages skrub DataOps pipelines, a lightweight abstraction for multi-table ML pipelines

SemPipes

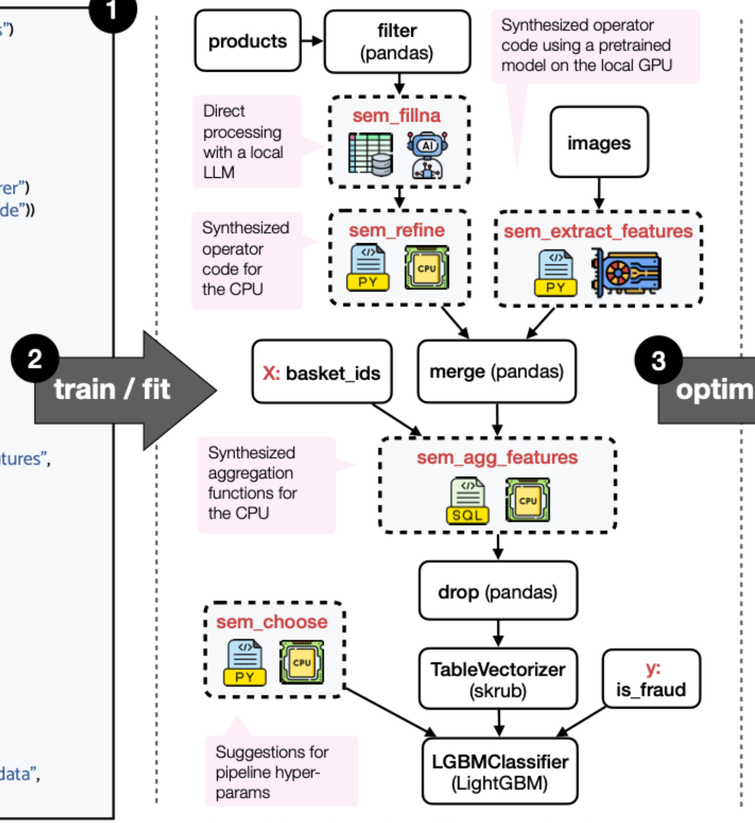
- **SemPipes synthesizes custom operator implementations** based on data characteristics, user instructions in natural language, and pipeline context

Pipeline source code in Python

```
basket_ids = sempipes.as_X(baskets.id, "Shopping baskets with transactions")
is_fraud = sempipes.as_y(baskets.fraud_flag, "Binary fraud indicator")

# Filter and clean the product data
products = products[products.basket_id.isin(basket_ids.id)]
products = products[products.purchase_year > 2023]
clean_products = (products
  .sem_fillna("Infer from product-related attributes", target_col="manufacturer")
  .sem_refine("Normalise to the ISO-3166 standard", target_col="country_code"))

# Extract additional features from product images
resale_features = images.sem_extract_features("Resale value information",
  output_columns = {"is_luxury_product": "Is this product luxurious?",
    "is_used_by_stars": "Is this product used by movie or pop stars?"})

# Join baskets, product and image data, generate aggregated features
products_resale = clean_products.merge(resale_features, on="product_id")
basket_data = basket_ids.sem_agg_features(
  aggregate=products_resale, join_on="basket_id",
  instruction="Generate and aggregate brand- and manufacturer-related features",
  how_many=10)

# Encode features via skrub and sklearn
basket_data = basket_data.drop(columns=["id", "basket_id"])
encoded_baskets = basket_data.skb.apply(
  skrub.TableVectorizer(),
  low_cardinality=OrdinalEncoder(),
  high_cardinality=skrub.GapEncoder())

# Use tree model for learning with a large hyperparameter range
model = encoded_baskets.skb.apply_with_sem_choose(
  lgbm.LGBMClassifier(), y=is_fraud,
  learning_rate=skrub.choose_from([0.1, 0.01, 0.001]),
  choices=sem_choose(num_leaves="Range appropriate for the processed data",
                     max_depth="Range compatible with num_leaves"))
```

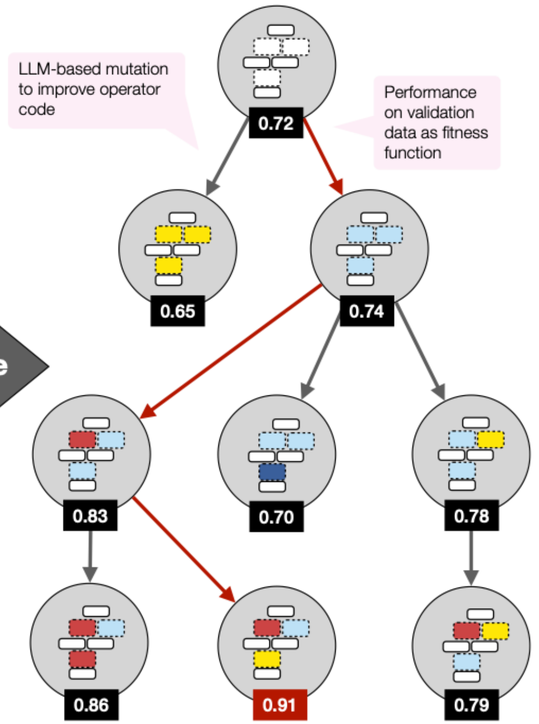
ML pipeline combining code from common data science libraries with **semantic data operators from SemPipes**.

Computational graph of the pipeline with synthesized operator code



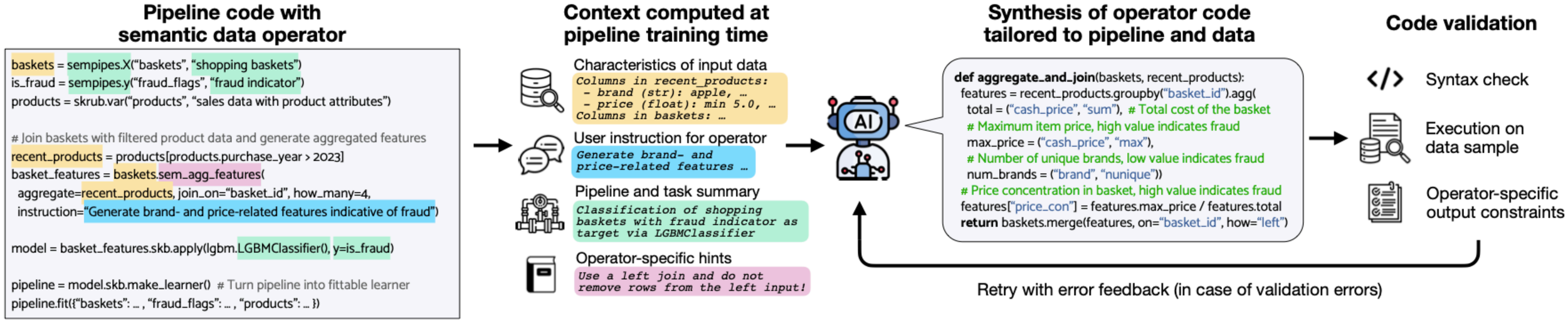
At training time, **SemPipes synthesizes custom operator implementations** based on data characteristics, user instructions and the pipeline context.

Evolutionary search to tune synthesized operator code



SemPipes can **automatically tune the synthesized operator code via evolutionary search** on a validation set.

SemPipes

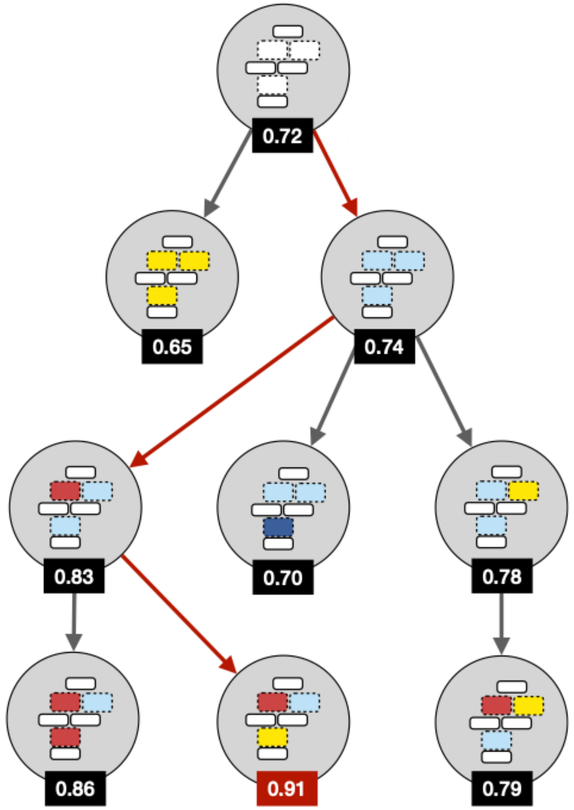


- **Training**
  - Synthesize operator code based on input data characteristics, pipeline context & prompt
  - Validate by executing code on a sample & check operator-specific output constraints
- **Inference**
  - Reuse synthesized code

SemPipes Optimizer

- **Refines** the LLM-synthesized implementations and **iteratively improves** generated code and predictive performance

- LLM suggests a better implementation based on history, predictive score on validation data, and previous “good" implementations
  - **Monte Carlo tree search** to improve the **each operator**
  - Select which operator to optimize in each round using **multi-armed bandits** approach
  - Evaluate improvements & **predictive performance** on the **validation set**



SemPipes Operators

- **Data cleaning & augmentation** operators

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

Introduced semantic operators

- **Logical:** Fill missing values
- **Physical:** Select one of 5 imputation models suggested by LLM

```
products = products.sem_fillna(
    target_column="make",
    nl_prompt="Infer the manufacturer from relevant product-related attributes like title or description.",
    impute_with_existing_values_only=True,
)
```

- **Logical:** Refine column's unique values using parametric knowledge
- **Physical:** Create correspondence dictionaries

```
cleaned_countries_ref = countries_ref.sem_refine(
    nl_prompt="Make sure that all values are in the ISO 3166-1 alpha-2 two-letter uppercase format",
    target_column="country_code",
    refine_with_existing_values_only=False,
).skb.eval()
```

- **Logical:** Data augmentation (rows)
- **Physical:** Augment data using Synthetic Data Vault (SDV) Python library

```
salaries_augmented = salaries.sem_augment(
    nl_prompt="Augment data to improve 'current_annual_salary' prediction.",
    number_of_rows_to_generate=2000,
    name="augment_salaries",
    generate_via_code=False,
)
```

SemPipes Operators

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

Introduced semantic operators

- **Feature extraction** operators

- **Logical:** Generate features from existing columns
- **Physical:** Combine/remove existing columns

```
movie_stats = movie_stats.sem_gen_features(
    nl_prompt="""
        Create additional features that could help predict the box office revenue of a movie.
        Consider aspects like genre, production details, cast, crew, and any other relevant information
        that could influence a movie's financial success. Some of the attributes are in JSON format,
        so you might need to parse them to extract useful information.
    """,
    name="additional_movie_features",
    how_many=25,
)
```

- **Logical:** Aggregate columns and calculate new aggregated columns
- **Physical:** Left join with custom aggregations

```
aggregated = baskets.sem_agg_features(
    products,
    left_on="ID",
    right_on="basket_ID",
    nl_prompt="""
        Generate product features that are indicative of potentially fraudulent baskets, make it easy to distinguish
        anomalous baskets from regular ones! It might be helpful to combine different product statistics.
        """,
    name="basket_agg_features",
    how_many=1,
)
```

- **Logical:** Extract features from text/images/ audio
- **Physical:** Pre-trained models, regexes

```
labeled_tweets = tweets.sem_extract_features(
    nl_prompt="Extract features from textual tweets that could help predict toxicity.",
    input_columns=["text"],
    name="tweet_features",
    output_columns=output_columns,
    generate_via_code=True,
    print_code_to_console=True,
)
```

SemPipes

- Selected results of **SemPipes** operators
  - Semantic operators improve and simplify ML pipelines written by humans and agents

Data-Centric Task	Metric	Pipeline	Origin: Expert	Origin: Agent	Original Pipeline	w/ SemOps & Instructions	w/ Optimized SemOps
Extraction of clinically informed features	F1 Score ↑	`micromodels`	✓		0.665 ± 0.01	0.692 ± 0.02	**0.729** ± 0.00
		`swe-micromodels`		✓	0.690 ± 0.00	0.692 ± 0.02	**0.729** ± 0.00
Blocking for entity resolution	Recall ↑	`rutgers`	✓		0.238 ± 0.00	0.240 ± 0.02	**0.255** ± 0.01
		`sustech`	✓		**0.322** ± 0.00	0.320 ± 0.00	0.321 ± 0.00
		`aide-sigmod`		✓	0.043 ± 0.00	0.163 ± 0.03	**0.165** ± 0.03
		`baseline-sigmod`	✓		0.099 ± 0.00	0.141 ± 0.02	**0.173** ± 0.05
Feature engineering (scrabble rating)	RMSE ↓	`kaggle-scrabble`	✓		176.1 ± 24.7	149.2 ± 24.8	**144.4** ± 18.3
		`aide-scrabble`		✓	255.0 ± 24.9	195.2 ± 21.9	**186.7** ± 21.1
		`swe-scrabble`		✓	228.5 ± 12.9	210.1 ± 43.0	**187.8** ± 21.4
Data annotation	Accuracy ↑	`hibug`	✓		0.598 ± 0.05	0.758 ± 0.15	**0.766** ± 0.16
Data augmentation for fairness	AUROC ↑	`sivep`	✓		0.653 ± 0.03	0.682 ± 0.00	**0.710** ± 0.04
		`swe-fair`		✓	0.650 ± 0.04	0.682 ± 0.02	**0.710** ± 0.04

SemPipes Operator Evaluation

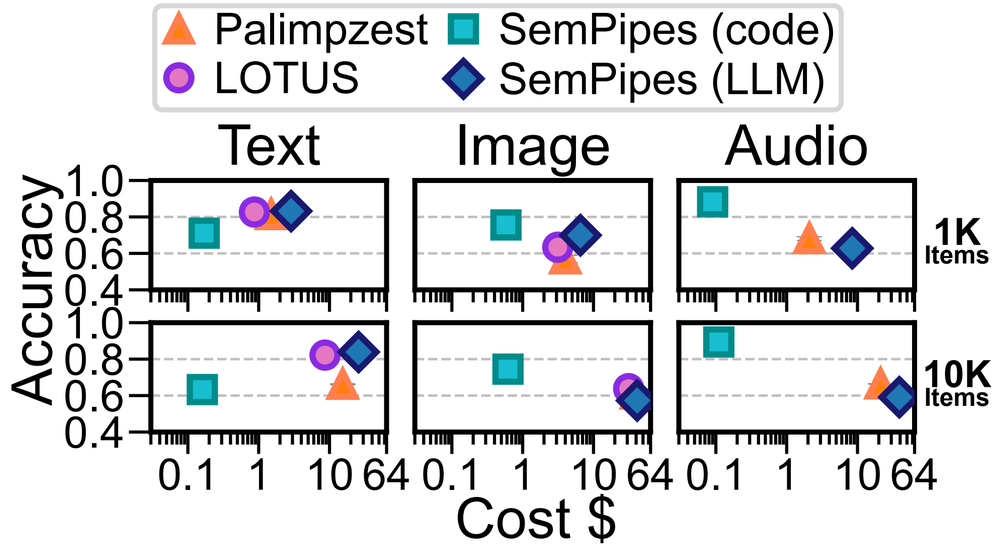
- **sem_gen_features** for the feature generation from existing data/columns
- **SemPipes** is on par with LLM feature engineering methods like CAAFE (NeurIPS'24)
  - Larger context such as data characteristics and pipeline & task summary

	Dataset	No FE	CAAFE	SemPipes
RF	balance-scale	0.685 ± 0.13	0.737 ± 0.19	**0.884** ± 0.23
	tic-tac-toe	0.807 ± 0.07	0.878 ± 0.11	**0.929** ± 0.08
	airlines	0.629 ± 0.03	**0.631** ± 0.03	0.625 ± 0.03
TabPFN	balance-scale	0.848 ± 0.25	0.863 ± 0.24	**0.938** ± 0.11
	tic-tac-toe	0.959 ± 0.02	0.945 ± 0.07	**0.997** ± 0.00
	airlines	0.647 ± 0.03	0.647 ± 0.03	**0.650** ± 0.03

AUROC scores without feature engineering (No FE), and for automated feature engineering on three datasets with Random Forest (RF) and TabPFN. **SemPipes** uses **sem_gen_features**

SemPipes Operator Evaluation

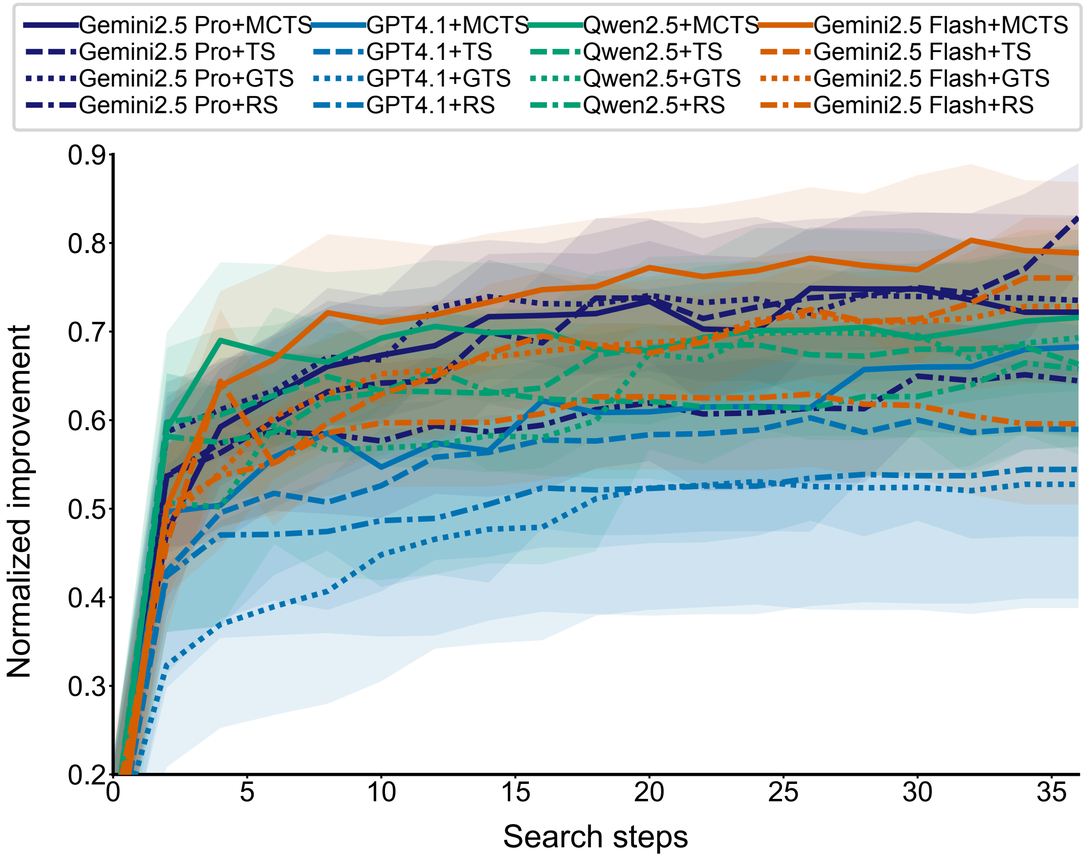
- **sem_extract_features** for zero-shot feature extraction from multi-modal data
- **SemPipes** outperforms direct LLM data processing for specialized data
  - Text - financial clause classification (DeBERTa)
  - Image - pneumonia detection in X-Rays (BiomedCLIP)
  - Audio - environmental sound extraction (CLAP)



Zero-shot feature extraction via direct data processing and code generation

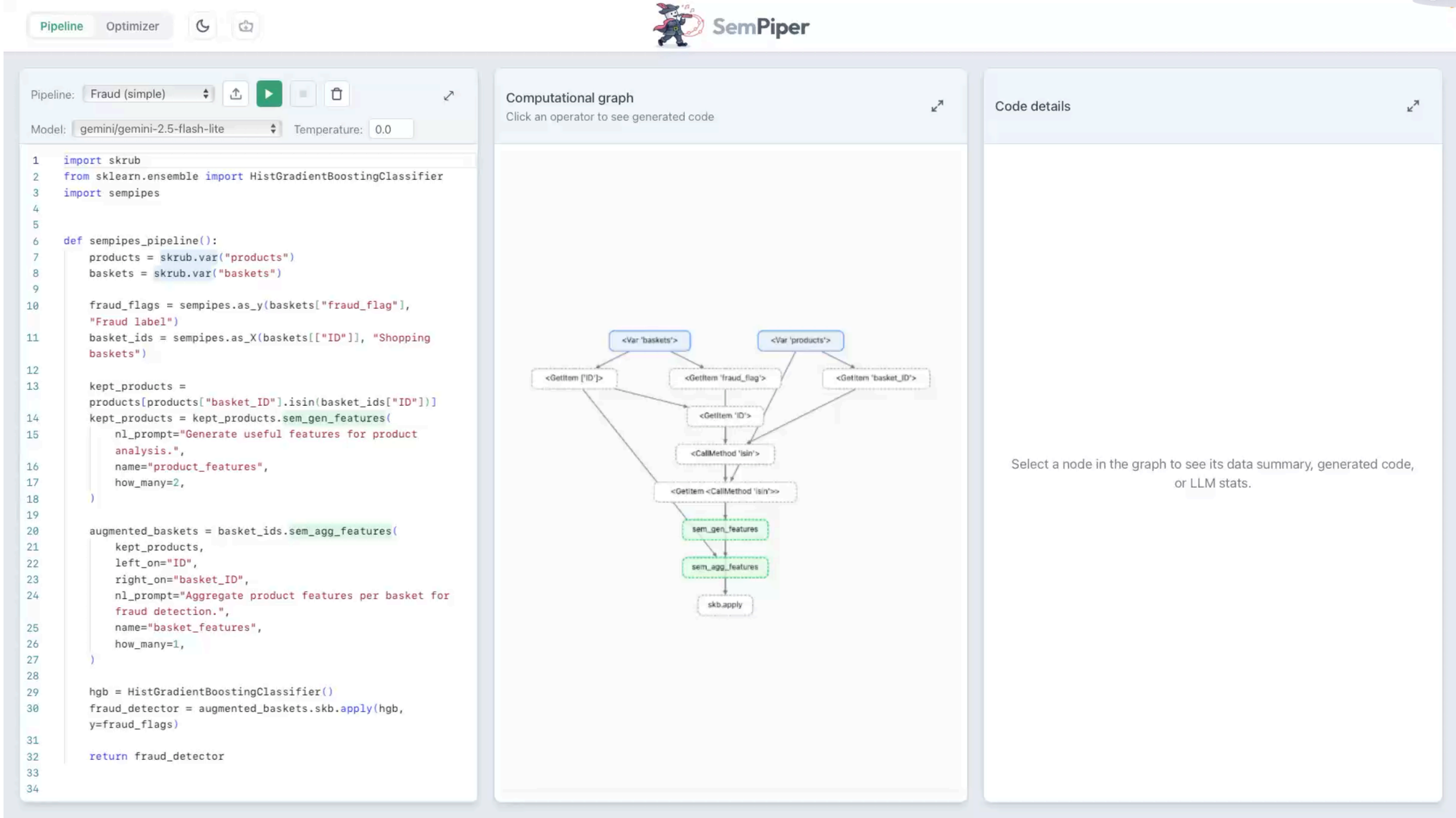
Smaller vs Larger Models

- **SemPipes** contextualization allows to achieve competitive performance with smaller models



Optimization effectiveness of LLMs for code synthesis in semantic data operators. Search policies include Monte Carlo tree search (MCTS), truncation selection (TS), greedy tree search (GTS) or random search (RS)

SemPiper “plays” SemPipes

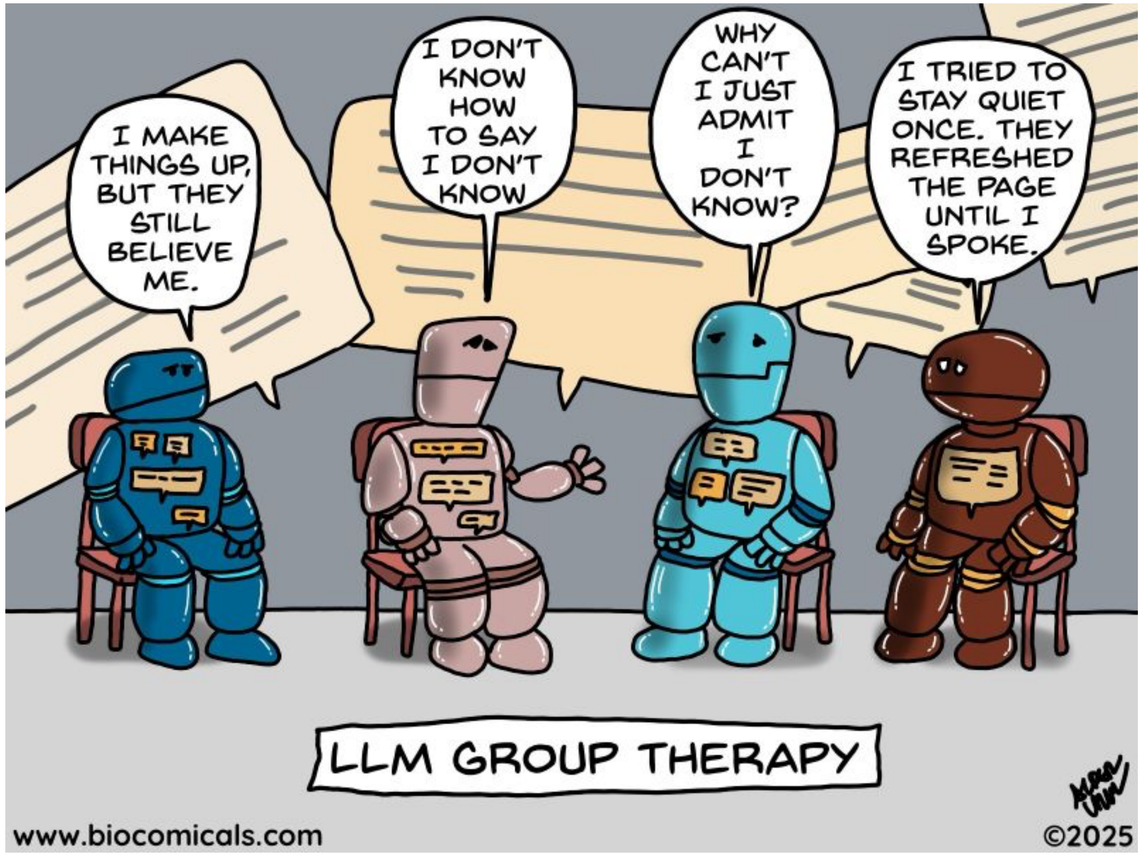


Conclusions

- Delegate data-heavy, time-consuming tasks to LLMs via code generation
- Utilize existing resources such as pre-trained HuggingFace models
- **Limitations**
  - Security risks if **SemPipes** is run in a not security-hardened sandbox
  - Limited support of videos, time series, and documents
- **Future work**
  - **SemAgent**: Generate end-to-end pipelines using **SemPipes** operators
    - Localize tasks within a pipeline and evolve each task separately

SemPipes research prototype

Thanks



SemPipes

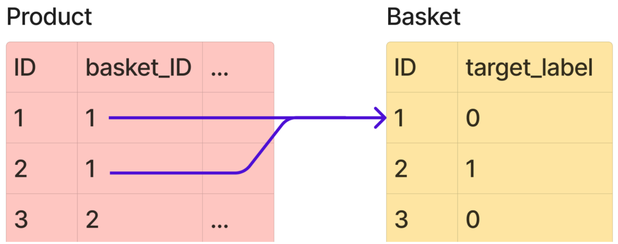
Contact

Skrub & DataOps

- **Eases preprocessing** & **feature engineering** for tabular ML and bridges between dataframe libraries (pandas, polars) and ML libraries (sklearn, scipy)
- Allows constructions of pipelines that can be **fully tuned**, **inspected**, or **applied** to new data
- **DataOps**
  - Special objects that encapsulate operations on data (applying operators, calling methods)
  - Replayable on new data and fitting/predicting with a pipeline w/o any code rewrites
  - Combinable into a DataOps plan, a directed acyclic graph (DAG) of operations

```
>>> import skrub
>>> a = skrub.var("a")
>>> b = skrub.var("b")
>>> c = a + b
>>> c
<BinOp: add>
>>> c.skb.eval({"a": 10, "b": 6})
16
```

A Very Simple Example Pipeline



- Credit fraud dataset with products and baskets tables
- **Task:** Predict if a basket is fraudulent (binary classification, highly unbalanced classes)

```
def pipeline() -> skrub.DataOp:
    products = skrub.var("products")
    baskets = skrub.var("data")
    labels = skrub.var("labels")

    baskets = baskets.skb.mark_as_X().skb.set_description("Potentially fraudulent shopping baskets of products")
    labels = labels.skb.mark_as_y().skb.set_description(
        "Flag indicating whether the basket is fraudulent (1) or not (0)"
    )

    aggregated = baskets.sem_agg_features(
        products,
        left_on="ID",
        right_on="basket_ID",
        nl_prompt="""
            Generate product features that are indicative of potentially fraudulent baskets, make it easy to distinguish
            anomalous baskets from regular ones! It might be helpful to combine different product statistics.
            """,
        name="basket_agg_features",
        how_many=1,
    )

    encoded = aggregated.skb.apply(TableVectorizer())
    return encoded.skb.apply(HistGradientBoostingClassifier(random_state=0), y=labels)
```

A Very Simple Example Pipeline, SemPipes Generated Code

- Left join to aggregate features with **SemPipes sem_agg_features**

```
def _sem_agg_join(left_join_column, left_df, right_join_column, right_df):

    # Aggregate required features at the basket_ID level
    agg_df = right_df.groupby(right_join_column).agg(
        total_basket_cash_price=('cash_price', 'sum'),
        total_basket_product_units=('Nbr_of_prod_purchas', 'sum'),
        # This feature counts the number of distinct 'item' categories purchased within each basket.
        num_unique_items_in_basket=('item', 'nunique'),
        # This feature captures the maximum cash price of any single product entry within a given basket.
        max_item_cash_price_in_basket=('cash_price', 'max'),
        # This feature counts how many items within a basket are categorized as service or fulfilment.
        num_service_fulfilment_items_in_basket=('is_service_or_fulfilment', 'sum'),
        # This feature calculates the standard deviation of `cash_price` for all items within a specific basket.
        basket_cash_price_std=('cash_price', 'std'),
        # Description: This feature counts the number of distinct 'make' (manufacturer/brand) types present within each basket.
        num_unique_makes_in_basket=('make', 'nunique'),
        # This feature calculates the minimum cash price of any product line within a given basket.
        min_item_cash_price_in_basket=('cash_price', 'min'),
        # This feature calculates the average cash price across all *line items* within a basket, providing a central tendency of
      item values.
        mean_item_cash_price_in_basket=('cash_price', 'mean')
    ).reset_index()

    # Description: This feature calculates the average cash value contributed by each individual product unit within a basket.
    agg_df['basket_item_value_per_product_unit'] = agg_df['total_basket_cash_price'] / agg_df['total_basket_product_units']
    agg_df.loc[agg_df['total_basket_product_units'] == 0, 'basket_item_value_per_product_unit'] = 0

    # Description: This feature calculates the ratio of the maximum single item's cash price to the total cash price of the entire
      basket.
    agg_df['basket_price_concentration_max'] = agg_df['max_item_cash_price_in_basket'] / agg_df['total_basket_cash_price']
    agg_df.loc[agg_df['total_basket_cash_price'] == 0, 'basket_price_concentration_max'] = 0

    # Fill NaNs for basket_cash_price_std (occurs when a basket has only one item) with 0.
    # A single item basket has no variation in price, so its standard deviation is 0.
    agg_df['basket_cash_price_std'] = agg_df['basket_cash_price_std'].fillna(0)

    # Drop the intermediate sum columns used for calculations, as they are not the final desired features
    agg_df = agg_df.drop(columns=['total_basket_cash_price', 'total_basket_product_units'])

    # Rename the right_join_column in agg_df to match left_join_column in left_df for merging
    agg_df = agg_df.rename(columns={right_join_column: left_join_column})

    # Left join the aggregated features back to the main dataframe
    left_df = left_df.merge(agg_df, on=left_join_column, how='left')

    return left_df
```

A Very Simple Example Pipeline, SemPipes Generated Code

```
import pandas as pd
import numpy as np

def _sem_agg_join(left_join_column, left_df, right_join_column, right_df):
    agg_df = right_df.groupby(right_join_column).agg(
        total_basket_cash_price=('cash_price', 'sum'), total_basket_product_units=('Nbr_of_prod_purchas', 'sum'),
         num_unique_items_in_basket=('item', 'nunique'), max_item_cash_price_in_basket=('cash_price', 'max'),
         num_service_fulfilment_items_in_basket=('is_service_or_fulfilment', 'sum'), basket_cash_price_std=('cash_price', 'std'),
         min_item_cash_price_in_basket=('cash_price', 'min'), mean_item_cash_price_in_basket=('cash_price', 'mean')
    ).reset_index()
```

Aggregate product-level features into basket-level features

```
    agg_df['basket_item_value_per_product_unit'] = agg_df['total_basket_cash_price'] / agg_df['total_basket_product_units']
    agg_df.loc[agg_df['total_basket_product_units'] == 0, 'basket_item_value_per_product_unit'] = 0

    agg_df['basket_price_concentration_max'] = agg_df['max_item_cash_price_in_basket'] / agg_df['total_basket_cash_price']
    agg_df.loc[agg_df['total_basket_cash_price'] == 0, 'basket_price_concentration_max'] = 0

    agg_df['basket_cash_price_std'] = agg_df['basket_cash_price_std'].fillna(0)
```

Fill missing values

```
    agg_df = agg_df.drop(columns=['total_basket_cash_price', 'total_basket_product_units'])
    agg_df = agg_df.rename(columns={right_join_column: left_join_column})
```

Clean up

```
    left_df = left_df.merge(agg_df, on=left_join_column, how='left')
```

Left join

```
    return left_df
```