Demonstrating CatDB: LLM-based Generation of Data-centric ML Pipelines

Saeed Fathollahzadeh
saeed.fathollahzadeh@concordia.ca
Concordia University
Montreal, Canada

Essam Mansour
essam.mansour@concordia.ca
Concordia University
Montreal, Canada

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

Abstract

AutoML systems automate finding Machine Learning (ML) pipelines but struggle to scale with large datasets due to time-consuming data analysis and complex hyper-parameter search spaces. LLMs (Large Language Models) offer flexibility and scalability for code generation with strong generalization across coding tasks. However, generating data-centric ML pipeline scripts is more challenging, as it requires complex reasoning to align the needs of a dataset with coding tasks, such as data cleaning or feature transformation. Thus, LLMs struggle to generate effective and efficient ML pipelines. This demo paper presents CatDB, which overcomes these challenges by dynamically generating dataset-specific instructions to guide LLMs in generating effective pipelines. CatDB profiles datasets to extract metadata, including refined data catalog information and statistics, and then uses this metadata to break down pipeline generation into instructions of tasks such as data cleaning, transformation, and model training, tailored to specifics of the dataset at hand. This process enables CatDB to leverage LLM coding capabilities more effectively. Our evaluation shows CatDB outperforms existing LLM-based and AutoML systems with up to orders of magnitude faster runtime on large datasets. The audience will experience CatDB's capabilities with commercial and open-source LLMs, using a variety of real datasets, as shown in our demo video and Colab notebook.

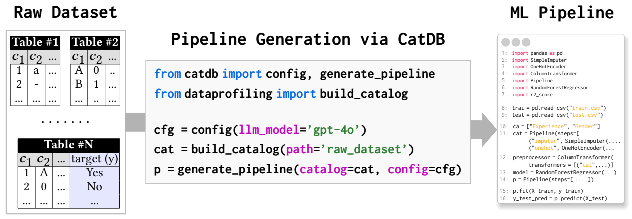
CCS Concepts

• **Information systems** → **Deduplication**; **Data cleaning**; **Data analytics**; **Online analytical processing**; • **Computing methodologies** → *Machine learning*.

Keywords

Data-centric ML Pipelines; Data Catalogs; LLM Code Generation

ACM Reference Format:
Saeed Fathollahzadeh, Essam Mansour, and Matthias Boehm. 2025. Demonstrating CatDB: LLM-based Generation of Data-centric ML Pipelines. 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/10.1145/3722212.3725097



c_1	c_2	...
1	a	...
2	-	...
...	...	...

c_1	c_2	...
A	0	..
B	1	...
...	..	...

c_1	c_2	...	target (y)
1	A	...	Yes
2	0	...	No
...	...	...	...

```
from catdb import config, generate_pipeline
from dataprofiling import build_catalog

cfg = config(llm_model='gpt-4o')
cat = build_catalog(path='raw_dataset')
p = generate_pipeline(catalog=cat, config=cfg)
```

```
import pandas as pd
import SimpleImputer
import OneHotEncoder
import ColumnTransformer
import Pipeline
import RandomForestRegressor
import r2_score

trai = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")

ca = ["Experience", "Gender"]
cat = Pipeline(steps=[
      ("imputer", SimpleImputer(....
      ("onehot", OneHotEncoder(...

preprocessor = ColumnTransformer(
      transformers = [("cat",...)]
model = RandomForestRegressor(...)
p = Pipeline(steps=[ ....])

p.fit(X_train, y_train)
y_test_pred = p.predict(X_test)
```

Figure 1: Data-centric ML Pipeline Generation via CatDB.

1 Introduction

Data-centric machine learning (ML) pipelines build on traditional ML pipelines by emphasizing key data preparation steps, such as cleaning, transformation, and feature engineering. These stages are crucial because the dataset quality often has a greater impact on model accuracy than the choice of algorithm. For example, a data scientist working with a tabular dataset aims to create a pipeline that optimizes performance for tasks like classification or regression. This process involves preparing the dataset through dedicated primitives—such as handling missing values or scaling numerical features—to enhance model performance. Data-centric pipelines are inherently exploratory and require iterative experimentation and a deep understanding of the dataset's quality. Hence, designing high-quality pipelines is computationally intensive and challenging, especially for domain-specific or large datasets. Human-in-the-loop approaches heavily depend on expert data scientists to refine the pipeline, making the process time-consuming and difficult to scale.

AutoML systems automate finding good ML pipelines through model selection and hyper-parameter tuning. However, these systems struggle to scale with large datasets due to time-consuming training and complex hyper-parameter search spaces. In contrast, large language models (LLMs), such as GPT-4 and Gemini-2, offer flexibility and scalability for code generation with strong generalization across coding tasks. Unlike code generation, generating ML pipelines requires in-depth reasoning to align a dataset's needs with tailored coding tasks, such as data cleaning or transformation.

LLMs perform generally very well on popular datasets seen during training. For example, the prompt "generating an ML pipeline for the Titanic dataset" with GPT-4 creates an efficient ML pipeline. However, LLMs struggle with unknown datasets not seen during training, and generate errors due to hallucinations. Some studies have explored using LLMs for model selection and pre-processing in the form of feature engineering. Moreover, these approaches often face significant performance challenges when applied to large datasets with numerous features, which leads to performance degradation. Table 1 qualitatively compares state-of-the-art AutoML systems, such as FLAML [7] and Auto-Sklearn [2], as well as LLM-based systems, such as CAAFE [4], AIDE [6] and AutoGen [8].

Table 1: Comparison of key capabilities of ML Pipeline generators and AutoML. *Dynamic* refers to a system's ability to adjust workflow based on dataset characteristics and ML tasks. *Instinctive* means that LLM generated workflows are applied.

Framework	Full	Verification/	Pre-processing			Model		Pipeline	Cost
	Pipeline	Error Management	Cleaning	Feature Selection & Engineering	Augmentation	Training	Improvement	Optimization	High/Low
AutoML	✗	✗	✗	✗	✗	Fixed Models	Bayes Opt./ Meta-learning/...	✗	N/A
CAAFE [4]	✗	Verification	✗	Feature Engineering	✗	Fixed Model	✗	Multi-Step w/ LLM	High
AIDE [6]	✓	✗	Instinctive	Instinctive	✗	Dynamic	✗	Multi-Step w/ LLM	High
AutoGen [8]	✓	✗	Instinctive	Instinctive	✗	Instinctive	✗	Multi-Step w/ Human+LLM	High
CatDB (our)	✓	✓	Dynamic	Dynamic	Dynamic	Dynamic	Dynamic	Instinctive w/ LLM	Low

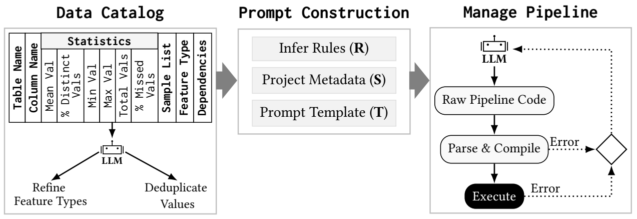


Figure 2: ML Pipeline Generation Workflow in CatDB.

This demo paper introduces CatDB, a Catalog-Driven Builder system powered by LLMs. The pipeline generation with CatDB requires simple API calls, as shown in Figure 1. CatDB improves LLM performance by providing it with dataset-specific instructions and relevant context. CatDB profiles datasets to extract valuable metadata, including refined catalog information and statistics. Using this metadata, CatDB breaks down the generation of pipelines into a sequence of coding tasks related to different stages, such as handling missing values, data cleaning, feature transformations, scaling numerical features, and training a classifier. These tasks are then expressed as customized instructions associated with the extracted metadata. This approach allows CatDB to leverage LLM coding capabilities more effectively. Our evaluation demonstrates that CatDB outperforms existing LLM-based and AutoML systems with up to orders of magnitude faster performance on large datasets.

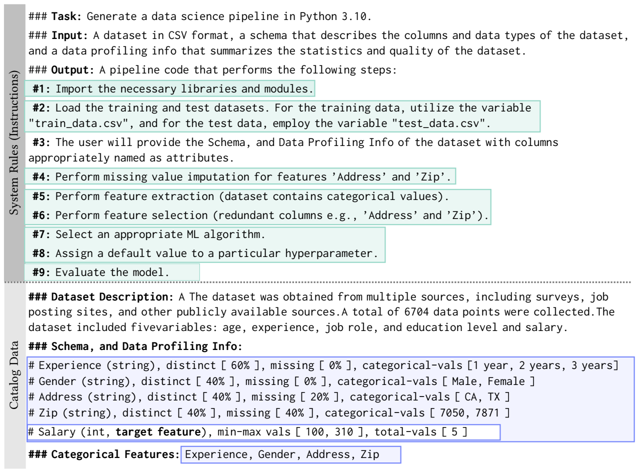
2 CatDB System Overview

CatDB simplifies the task of generating data-centric ML pipelines using LLMs by decomposing the generation into three well-defined sub-problems: data catalog extraction and refinement, LLM prompt construction, and managing the generated pipeline in terms of error correction and execution, as illustrated in Figure 2.

2.1 Building and Refining Data Catalog

If the dataset is not already in the catalog, we first profile it to generate essential metadata. This process includes examining each column for its schema (name and type), distinct and missing values, basic statistics (min, max, median), and feature types (e.g., categorical or list-based). CatDB classifies attributes by type (e.g., string or numeric) and maps them to relevant ML feature types like *Categorical* or *List*. To further enrich the metadata, CatDB utilizes LLMs to infer feature types and improve the accuracy of the classifications. Our refined feature types and cleaned categorical values significantly improve ML model performance in various scenarios.

Our goals for catalog refinement, data cleaning, and the extraction of a refined dataset are: A) **Categorizing Sentence Data Types:** We identify string features as potential categorical candidates, addressing mixed representations and missing values that may hinder accurate classification. CatDB refines values in two


System Rules (Instructions)
Catalog Data

```
### Task: Generate a data science pipeline in Python 3.10.
### Input: A dataset in CSV format, a schema that describes the columns and data types of the dataset,
and a data profiling info that summarizes the statistics and quality of the dataset.
### Output: A pipeline code that performs the following steps:
 #1: Import the necessary libraries and modules.
 #2: Load the training and test datasets. For the training data, utilize the variable
"train_data.csv", and for the test data, employ the variable "test_data.csv".
 #3: The user will provide the Schema, and Data Profiling Info of the dataset with columns
appropriately named as attributes.
 #4: Perform missing value imputation for features 'Address' and 'Zip'.
 #5: Perform feature extraction (dataset contains categorical values).
 #6: Perform feature selection (redundant columns e.g., 'Address' and 'Zip').
 #7: Select an appropriate ML algorithm.
 #8: Assign a default value to a particular hyperparameter.
 #9: Evaluate the model.
### Dataset Description: A The dataset was obtained from multiple sources, including surveys, job
posting sites, and other publicly available sources.A total of 6704 data points were collected.The
dataset included fivevariables: age, experience, job role, and education level and salary.
### Schema, and Data Profiling Info:
# Experience (string), distinct [ 60% ], missing [ 0% ], categorical-vals [1 year, 2 years, 3 years]
# Gender (string), distinct [ 40% ], missing [ 0% ], categorical-vals [ Male, Female ]
# Address (string), distinct [ 40% ], missing [ 20% ], categorical-vals [ CA, TX ]
# Zip (string), distinct [ 40% ], missing [ 40% ], categorical-vals [ 7050, 7871 ]
# Salary (int, target feature), min-max vals [ 100, 310 ], total-vals [ 5 ]
### Categorical Features: Experience, Gender, Address, Zip
```

Figure 3: An Example of a prompt (Instructions and Metadata), constructed by CatDB for a specific dataset.

ways. (1) we separate composite data into distinct features (e.g., splitting an Address attribute into State and Zip) and (2) we split sentence features into categorical elements, converting them into hashed numerical features. Additionally, we utilize LLMs to infer feature types using only attribute names and a small sample set (10 examples in our system). B) **Refining Categorical Data:** We refine semantically equivalent categorical values (e.g., mapping Gender entries like [F, 0, Male] to [Male, Female]). Given the typically small set of distinct values, we submit the entire list to the LLM for generating a mapping of refined to original values.

2.2 Prompt Construction

CatDB analyzes the metadata (e.g., column names, data types, statistics, and dependencies) to automatically generate dataset-specific instructions (rules). These rules break down the pipeline generation into specific coding tasks tailored to subsets of the dataset, as illustrated in Figure 3. Thus, the constructed prompt guides LLMs in generating a more efficient pipeline. These rules are related to:

- *Data Preparation:* Selecting appropriate techniques for handling missing values, normalization, and outlier removal.
- *Feature Dependency:* Identifying relevant features, extracting distinct values, and handling dependencies.
- *Feature Filter:* Removing irrelevant or redundant features.
- *Data Augmentation:* Generating synthetic data to address small or imbalanced datasets.

By combining metadata-driven prompts and rules, we guide LLMs to produce more accurate and efficient ML pipelines. In detail, the LLM prompts comprise *rule* messages (**R**) for guidance and *schema* messages (**S**) for context from the catalog, forming a template (**T**).

2.3 Pipeline Generation

The third step involves correcting, refining, and executing the Python ML pipeline generated by the LLM. Our goal is to apply the pipeline on the provided dataset and assess its performance. This process also requires handling various errors, including syntactic, semantic, and runtime issues.

Validation and Error Management: While we guide the LLM with metadata, errors are inevitable due to the inherent randomness of LLMs. To address this, CatDB includes a dedicated error management component. We categorize errors into three groups:

- *Environment and Package Errors:* We handle missing packages and out-of-memory issues with predefined strategies.
- *Syntax and Parse Errors:* We automatically fix common syntax errors or resubmit the pipeline to the LLM for correction.
- *Runtime and Semantic Errors:* We run pipelines on a sample, identify errors, and iteratively refine the pipeline with LLM assistance (up to τ times) and domain knowledge.

3 Demonstration Scenarios

To demonstrate the utility of CatDB in ML applications, we will guide attendees through the following steps for using CatDB. We describe intended user interactions through Python code in six steps (see Figure 4), using the Adult dataset as a running example.

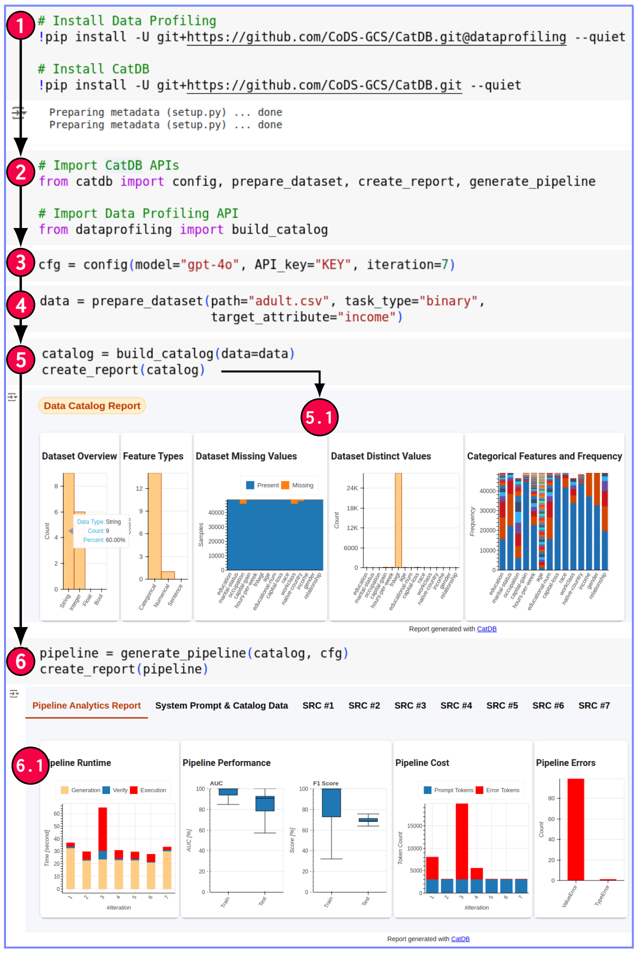
Step 1. Installation and Prerequisites: We initially install CatDB, from our open-source GitHub repository that provides a Python API. No external libraries are required. CatDB is not a data profiling framework but utilizes outputs from profiling systems, such as the profiling system of KGLiDS [3].

Step 2. Required APIs: The `config` and `generate_pipeline` APIs from CatDB, along with `build_catalog` from KGLiDS, are essential for generating end-to-end data-centric ML pipeline. End-users can utilize the `prepare_dataset` API to split (into train, test, and validation sets) and materialize their data. Furthermore, CatDB features a powerful `create_report` API that generates and visualizes data catalog information and pipeline generation outputs. This API is highly valuable for debugging and gaining a comprehensive dataset overview. The `build_catalog` API can be changed if the user decides to utilize a different data profiling framework.

Step 3. LLM Configuration: We support online LLMs (OpenAI, Google Gemini, and Graq) that provide services. Therefore, the end-user only needs to pass the model name, and CatDB will identify the source of the services. Naturally, the API key is required for cost management and authorization. As mentioned throughout the paper, LLMs are prone to hallucinations. To address and mitigate hallucinations, to reduce LLM randomization, and to provide multiple options, we can generate more pipelines by passing a number of iterations (`iteration=7` in Figure 4). While LLMs have numerous configuration parameters (such as temperature, and maximum number of tokens) that are hidden from the high-level API for simplicity, end-users can modify the CatDB default settings.

Step 4. Dataset Preparation: As part of an ML pipeline, the end-user must split the dataset, indicate the task type (binary/multi-class classification or regression), and specify the target feature. Our optional dataset preparation API simplifies this process.

Step 5. Build Data Catalog: We profile a given raw input dataset via our `create_report` API that computes and visualizes five main



```
# Install Data Profiling
!pip install -U git+https://github.com/CoDS-GCS/CatDB.git@dataprofiling --quiet

# Install CatDB
!pip install -U git+https://github.com/CoDS-GCS/CatDB.git --quiet

  Preparing metadata (setup.py) ... done
  Preparing metadata (setup.py) ... done

# Import CatDB APIs
from catdb import config, prepare_dataset, create_report, generate_pipeline

# Import Data Profiling API
from dataprofiling import build_catalog

cfg = config(model="gpt-4o", API_key="KEY", iteration=7)

data = prepare_dataset(path="adult.csv", task_type="binary",
                       target_attribute="income")

catalog = build_catalog(data=data)
create_report(catalog)

pipeline = generate_pipeline(catalog, cfg)
create_report(pipeline)
```

Figure 4: CatDB Data-Centric ML Pipeline Generation Demonstration (CoLab Link).

diagrams (Step 5.1), and we materialize all this information as extended catalog metadata: 1) *Dataset Overview & Distinct Values:* This diagram presents the column data types and their ratios. For example, the Adult dataset contains nine columns of data type string, representing 60% of the total columns. Additionally, the distinct values diagram shows the number of unique values per column. 2) *Feature Types:* The second analysis in CatDB identifies feature types (Categorical, Numerical, and Sentence). Identifying these types informs for data pre-processing and substantially impacts model training. We have extended the KGLiDS framework to incorporate feature type identification, utilizing the dataset overview statistics for this purpose. 3) *Missing Value Report:* This diagram visualizes data density within each feature. The missing value ratio can inform about a need for missing value imputation or feature reduction during pre-processing. 4) *Categorical Feature Materialization:* We compute and visualize the categorical features and the frequency of each unique item within individual columns. This frequency information serves two purposes: selecting appropriate feature transformations (e.g., one-hot encoding, binning), and for classification, checking for class imbalance.

Step 6. Pipeline Generation: CatDB generates pipelines with data catalog and LLM configurations, then runs the pipelines and

Table 2: Performance Comparison of 8 Datasets (Iteration = 1, TO: Time Out, OOM: Out of Memory)

Dataset	Metric	LLM	CatDB		CAAFE		AIDE	AutoGen	AutoML				AutoML w/ Cleaning & Augmentation			
			Single	Chain	TabPFN	R.Forest			A.Sklearn	H2O	FLAML	Autogluon	A.Sklearn	H2O	FLAML	Autogluon
Airline	AUC-ovr	GPT-4o	**100.0**	**100.0**	OOM		N/A	**100.0**	OOM	**100.0**	TO	TO	No Training Model			
		Gemini-1.5	**100.0**	**100.0**			**100.0**	**100.0**	OOM	**100.0**	TO	TO				
		Llama3.1-70b	**100.0**	**100.0**			N/A	99.0	OOM	**100.0**	TO	TO				
IMDB	AUC	GPT-4o	98.62	98.58	OOM		N/A	**100.0**	OOM	**100.0**	**100.0**	99.99	OOM	99.99	99.99	TO
		Gemini-1.5	99.98	99.98			N/A	**100.0**	OOM	**100.0**	99.99	97.5	OOM	99.99	99.99	TO
		Llama3.1-70b	97.18	96.38			N/A	50.0	OOM	**100.0**	99.99	99.99	OOM	99.99	99.75	TO
Accidents	AUC-ovr	GPT-4o	94.21	95.01	OOM	84.62	TO	94.44	OOM	93.9	93.94	**97.34**	OOM	91.49	93.12	94.95
		Gemini-1.5	94.2	94.21	OOM	84.25	TO	96.54	OOM	90.96	95.36	**97.17**	OOM	84.35	93.35	94.91
		Llama3.1-70b	95.0	95.18	OOM	84.02	TO	93.83	OOM	92.92	94.95	**97.35**	OOM	93.09	92.93	94.72
Financial	AUC-ovr	GPT-4o	**100.0**	99.9	OOM	85.24	TO	**100.0**	OOM	**100.0**	**100.0**	99.99	OOM	99.01	**100.0**	**100.0**
		Gemini-1.5	**100.0**	**100.0**	OOM	84.9	TO	**100.0**	OOM	**100.0**	**100.0**	99.99	OOM	**100.0**	**100.0**	**100.0**
		Llama3.1-70b	**100.0**	**100.0**	OOM	85.94	TO	**100.0**	OOM	**100.0**	**100.0**	99.99	OOM	99.01	**100.0**	**100.0**
CMC	AUC-ovr	GPT-4o	68.66	73.81	73.13	67.91	71.56	71.61	TO	**76.39**	**76.39**	27.22	TO	74.15	73.62	72.81
		Gemini-1.5	68.84	74.33	73.13	67.95	72.02	71.61	TO	75.17	**77.23**	25.71	TO	74.43	75.65	71.11
		Llama3.1-70b	73.15	71.03	75.7	74.29	71.56	71.53	TO	**75.96**	75.89	27.22	TO	75.71	74.24	71.11
Bike-Sharing	R^2	GPT-4o	76.83	86.89	Doesn't support		39.46	39.46	TO	< 0	90.97	60.0	TO	< 0	92.27	**92.3**
		Gemini-1.5	92.06	92.12			94.27	93.55	**94.32**	< 0	93.82	64.29	93.17	< 0	93.29	93.18
		Llama3.1-70b	79.54	88.04			93.47	93.5	**94.39**	< 0	93.82	72.15	93.17	< 0	92.81	92.15
House-Sales	R^2	GPT-4o	87.48	86.34	Doesn't support		75.41	N/A	**89.81**	15.68	87.99	86.69	83.57	26.5	77.7	84.19
		Gemini-1.5	**99.99**	87.98			75.41	87.9	89.86	TO	89.94	90.34	84.07	15.91	77.7	83.6
		Llama3.1-70b	87.96	89.0			87.87	87.9	89.89	TO	89.39	**90.36**	84.13	TO	83.62	84.19
NYC	R^2	GPT-4o	48.58	55.17	Doesn't support		32.64	**68.74**	10.14	TO	56.38	43.25	9.56	31.56	62.85	41.24
		Gemini-1.5	67.22	65.53			**69.25**	68.74	62.41	TO	68.82	45.16	33.72	TO	68.91	51.97
		Llama3.1-70b	48.58	48.58			68.71	68.75	56.67	35.28	65.88	45.56	57.04	31.57	**69.42**	55.59

automatically fixes runtime and syntax errors. The CatDB pipeline reports have three main sections: **6.1.) Analytics Report Tab:** i) *Pipeline Runtime Report:* Expresses LLM latency, pipeline validation, and pipeline execution times per iteration. This capability allows the identification of slow pipelines. ii) *Pipeline Performance:* Examines pipeline performance on both training and test datasets. For each task, two different metrics are offered and visualized as box-plots (binary classification: AUC and F1-score, multi-class classification: AUC-ovr and Log-loss, regression: R2-score and RMSE). iii) *Pipeline Cost:* Reports the pipeline cost, which is the number of tokens used by a pipeline prompt and an error prompt to generate the ML pipeline and fix runtime and syntax errors. iv) *Error Categorization and Frequency:* Shows the frequency and types of errors in different diagrams. **6.2.) Prompt Visualization Tab:** Visualizes the constructed prompts submitted to the LLM in the `System Prompt & Catalog Data` tabs, demonstrating all this information in a hierarchical tree structure. This hierarchical tree primarily shows which data from the data catalog is projected, filtered, and combined to form a the submitted LLM prompt.

4 Experiments

Settings. We developed two variants: CatDB, which uses a single prompt, and CatDB Chain, which uses a chain of prompts for pre-processing, feature engineering, and model selection. We evaluate our system on 8 real-world datasets across classification and regression tasks, and compare it with two groups of baselines: *LLM-based Pipelines* where we compare with AIDE [6], AutoGen [8], and CAAFE [4] (which leverage LLMs for feature engineering and pipeline generation), as well as *AutoML Systems* where we compare with AutoML tools for tabular data including AutoGluon [1], H2O [5], FLAML [7], and Auto-Sklearn [2].

Quality of Generated Pipelines: Table 2 shows the results for all classification (binary/multi-class) and regression tasks. Overall, we see reliable performance of CatDB and CatDB Chain compared to the state-of-the-art systems. Every row shows the test AUC/R^2 for a dataset/LLM pair. CatDB and CatDB Chain achieve a majority of top rankings. Here, CatDB Chain yields generally better performance for larger datasets, where the task splitting ensures all tasks are represented (e.g., CatDB missed feature engineering for CMC), reduces errors, and achieves very good accuracy. In contrast, CAAFE TabPFN failed on large datasets, and many AutoML tools ran into out-of-memory errors or timeouts.

Runtime of Generated Pipelines: CatDB's runtime—which includes data loading, metadata projection, rule definition, pipeline generation, validation, error management, and execution—is fast across all datasets. CAAFE succeeded with smaller datasets but failed on larger ones after 4 days, due to data pre-processing dominating its runtime. AIDE and AutoGen often spent excessive time retrying requests and executing naive grid search pipelines. In contrast, CatDB and CatDB Chain executed successfully on all datasets, with performance improvements by guiding LLMs to only generate necessary primitives and parallelized code.

5 Conclusions

CatDB guides LLMs with data catalog information, data-specific instructions, and error handling. This approach enables CatDB to leverage LLM coding capabilities more effectively to generate robust and efficient data-centric ML pipelines at moderate costs.

References

[1] Nick Erickson and et al. 2020. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. *CoRR* abs/2003.06505 (2020). https://arxiv.org/abs/2003.06505
[2] Matthias Feurer and et al. 2015. Efficient and Robust Automated Machine Learning. In *NeurIPS 28*. 2962–2970.
[3] Mossad Helali and et al. 2024. KGLiDS: A Platform for Semantic Abstraction, Linking, and Automation of Data Science. In *ICDE*.
[4] Noah Hollmann and et al. 2023. Large Language Models for Automated Data Science: Introducing CAAFE for Context-Aware Automated Feature Engineering. In *NeurIPS*. https://arxiv.org/pdf/2305.03403
[5] Erin LeDell and et al. 2020. H2o automl: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, Vol. 2020. ICML.
[6] Dominik Schmidt and et al. 2024. AIDE: Human-Level Performance in Data Science Competitions. https://www.weco.ai/blog/technical-report
[7] Chi Wang and et al. 2021. FLAML: A Fast and Lightweight AutoML Library. In *Proceedings of Machine Learning and Systems (MLSys)*. mlsys.org.
[8] Qingyun Wu and et al. 2024. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. In *ICLR 2024 Workshop on LLM Agents*.