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

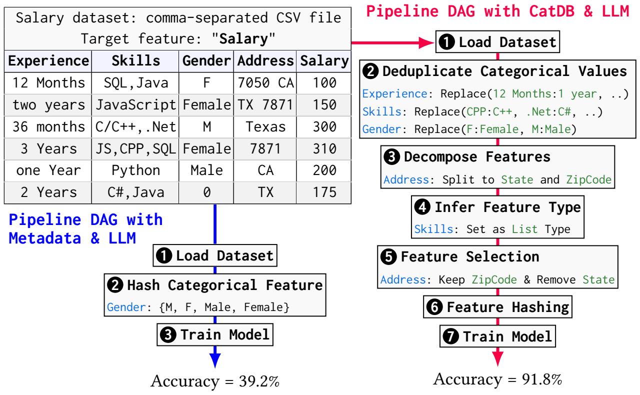
Matthias Boehm
Technische Universität Berlin
matthias.boehm@tu-berlin.de

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.

PVLDB Reference Format:
Saeed Fathollahzadeh, Essam Mansour, and Matthias Boehm. CatDB: Data-catalog-guided, LLM-based Generation of Data-centric ML Pipelines. PVLDB, 18(8): 2639 - 2652, 2025.
doi:10.14778/3742728.3742754

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/CoDS-GCS/CatDB.git.



Experience	Skills	Gender	Address	Salary
12 Months	SQL,Java	F	7050 CA	100
two years	JavaScript	Female	TX 7871	150
36 months	C/C++,.Net	M	Texas	300
3 Years	JS,CPP,SQL	Female	7871	310
one Year	Python	Male	CA	200
2 Years	C#,Java	0	TX	175

Salary dataset: comma-separated CSV file
Target feature: "Salary"
Pipeline DAG with CatDB & LLM
❶ Load Dataset
❷ Deduplicate Categorical Values
Experience: Replace(12 Months:1 year, ..)
Skills: Replace(CPP:C++, .Net:C#, ..)
Gender: Replace(F:Female, M:Male)
❸ Decompose Features
Address: Split to State and ZipCode
❹ Infer Feature Type
Skills: Set as List Type
❺ Feature Selection
Address: Keep ZipCode & Remove State
❻ Feature Hashing
❼ Train Model
Accuracy = 91.8%
Pipeline DAG with Metadata & LLM
❶ Load Dataset
❷ Hash Categorical Feature
Gender: {M, F, Male, Female}
❸ Train Model
Accuracy = 39.2%

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.

1 INTRODUCTION

Modern data-driven applications increasingly rely on machine learning (ML) for cost-effective intelligence augmentation and intelligent infrastructure in various domains, such as health-care, finance, transportation, and production [39]. The traditional data 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 coverage often yields better accuracy than advanced ML models [53] or neural architecture search [60]. Even the original AlexNet paper [45] that won the ImageNet 2012 challenge and re-ignited research on deep neural networks (DNNs) heavily relied on data augmentation. Unfortunately, devising effective data-centric ML pipelines is a labor- and compute-intensive process. For example, tuning data augmentation [22] or data cleaning [74] pipelines requires expensive reinforcement learning or evolutionary algorithms.

Limitations of LLMs and AutoML Tools: LLMs, such as GPT-4 and Gemini-2, are increasingly utilized for tasks like code generation and data wrangling [55]. AutoML tools optimize feature selection and model tuning [24, 27, 46], but struggle to scale with large datasets due to time-intensive data analysis and the complexity of hyper-parameter search spaces. In contrast, LLMs generalize across diverse coding tasks and adapt ML pipelines dynamically. For well-known datasets, such as Titanic, LLMs like GPT-4 and Gemini-2 generate accurate and effective ML pipelines even without explicit dataset characteristics in the prompt. Readers can test this by prompting an LLM with: "Generate a data science pipeline for the Titanic dataset." Since Titanic is a widely used benchmark dataset that LLMs are typically exposed to during training, they can retrieve and apply relevant pre-processing steps, feature engineering, and model choices with high accuracy. However, LLMs struggle

Proceedings of the VLDB Endowment, Vol. 18, No. 8 ISSN 2150-8097.
doi:10.14778/3742728.3742754

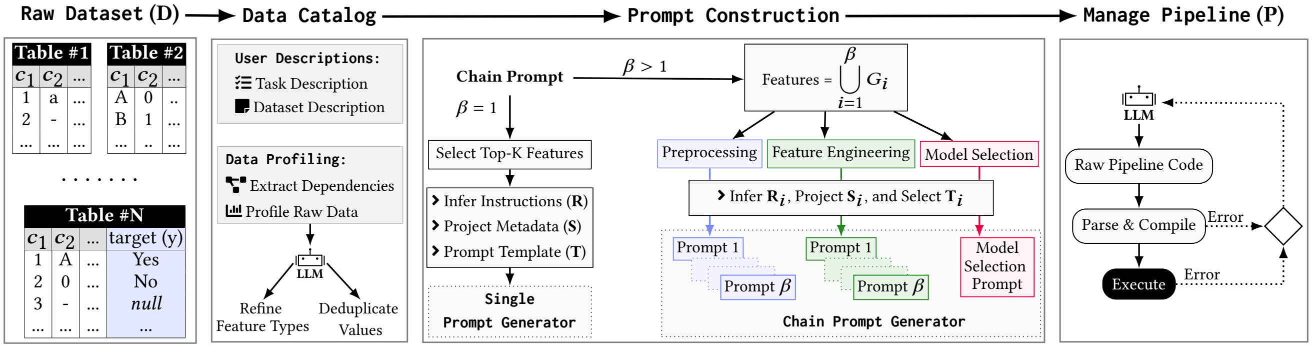


Figure 2: Data-centric ML pipeline generation in CatDB with two variants: Single Prompt Generator (CatDB) and Chain Prompt Generator (CatDB Chain). CatDB Chain is more effective for large datasets to overcome LLM's limited context length.

with unseen datasets due to a lack of prior exposure, as illustrated in Figure 1 with [Metadata-only & LLM]. In such cases, LLMs fail to infer dataset-specific pre-processing steps, hallucinate incorrect features [40], or misinterpret data structures. Unlike standard code generation, ML pipeline creation requires dataset-aware reasoning, which existing LLMs and AI code generation tools [54, 64] are not optimized for. To overcome these limitations, two main approaches are used: fine-tuning and prompt engineering. Fine-tuning LLMs could improve their ability to generate relevant pipelines but is costly and often impractical. A more feasible alternative is structured prompt engineering, where dataset metadata (e.g., feature types, missing value statistics) is incorporated into prompts. A key challenge, however, is dynamically curating dataset-specific instructions that guide the LLM in applying the right data science actions to the given dataset. Refining metadata and curating these instructions effectively remains challenging. By overcoming these challenges, LLMs can offer a scalable approach to automating the generation of data-centric ML pipelines, reducing manual effort.

New Opportunity – Data Catalogs: Data catalogs, such as Gaia-X ecosystem [76], Apache Atlas [11], and Google dataset search [14, 17, 32], can help bridge the gap in overcoming the remaining LLM limitations. In addition to metadata (e.g., schema) and provenance information (e.g., origin), these catalogs also store data characteristics and profiling information [9], which can be used to guide LLMs in tailoring pipelines to the specific needs of each dataset. However, remaining challenges include the need for metadata refinements and dataset-specific instructions that can identify relevant information for specific stages of the pipeline generation.

CatDB Overview: We introduce CatDB, a zero-shot in-context learning (ICL) approach to generate high-performance, data-centric ML pipelines in Python using LLMs. As an ICL approach, CatDB extracts refined data catalog details to craft dataset-specific instructions for data science pipeline stages. By adopting a zero-shot approach, CatDB eliminates the need for task-specific examples. CatDB profiles the input dataset, builds a unified data catalog, and integrates metadata and instructions into LLM prompts, as shown in Figure 2. To avoid manual labeling, CatDB uses LLMs to infer feature types (e.g., categorical or list) over basic types like string and refines metadata by cleaning distinct values. Using this catalog, CatDB splits pipeline creation into coding tasks—such as handling missing values, scaling features, and training classifiers—expressed as tailored instructions tied to the metadata. This approach overcomes LLM limitations and boosts coding efficiency. To handle errors (syntactic and runtime) from LLM hallucinations, CatDB auto-validates and fixes pipelines using a knowledge base of error traces from diverse datasets. The error management improves reliability, scalability, and performance across domains.

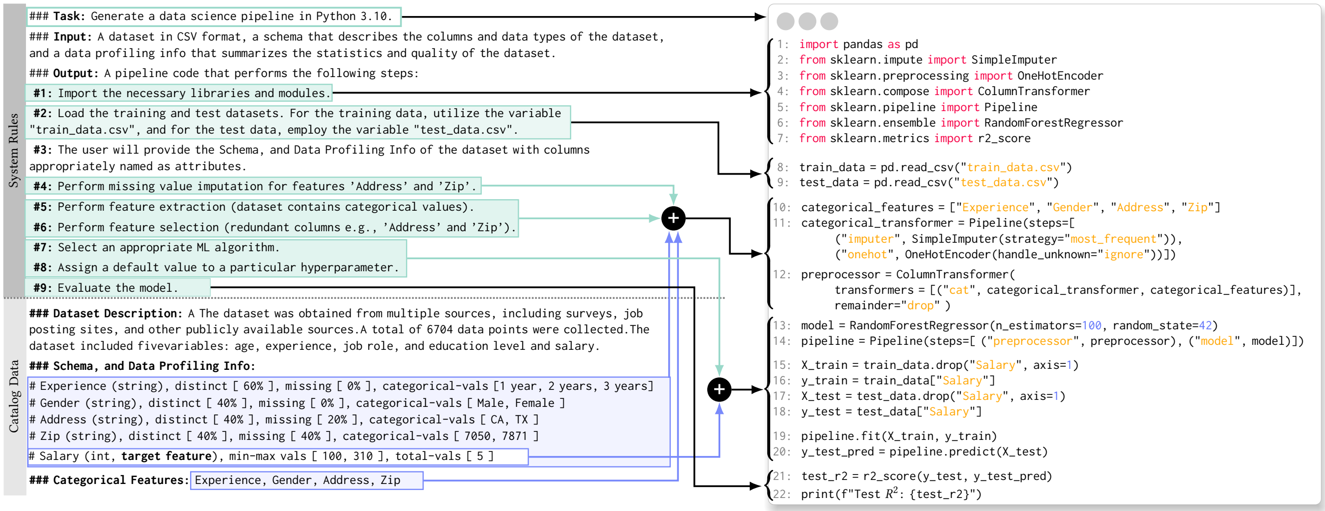
Contributions: Our primary contribution is CatDB, a fully automated system that leverages data catalogs and LLMs to generate effective data-centric ML pipelines. Our key contributions are:

- *Data-Catalog-Guided ML Pipeline Generation:* We present an end-to-end overview of CatDB that integrates data catalogs and LLMs to create data-centric ML pipelines in Section 2.
- *Prompt Construction:* We introduce methods for incorporating data catalog information, refined metadata, and rules into LLM prompts in Section 3.
- *Pipeline Generation:* We outline the process for generating pipelines using LLMs, including error handling, an error traces dataset, and cost analysis, covered in Section 4.
- *Experiments:* We provide results and insights from testing CatDB with 20 diverse datasets (up to 19 tables, 30 million rows, and 478 columns), state-of-the-art baseline systems (CAAFE [37], AIDE [69], AutoGen [82], H2O [46], FLAML [80], and Auto-Sklearn2 [25]) as well as different LLMs (commercial GPT-4o [59] or Gemini-pro-1.5 [77], and open-source Llama-3 [10]) in Section 5. CatDB yields accuracy close to or better than existing LLM-based and AutoML tools, with performance up to orders of magnitude faster on large datasets.

2 CATDB SYSTEM OVERVIEW

CatDB, an zero-shot, in-context learning (ICL) approach, generates detailed dataset-specific instructions to guide LLMs in generating data-centric ML pipelines. CatDB stands for **Cat**alog-**D**riven **B**uilder, profiles the dataset, and refines data catalog information for more precise instructions, as shown in Figure 1, steps 1-7. This approach achieves 91.8% accuracy while eliminating the need for task-specific examples (few-shot learning) across pipeline stages. Figure 3 shows an example constructed prompt and generated pipeline by OpenAI GPT-4o for the Salary dataset.

Notation: Given a tabular dataset $\mathcal{D}$ (of categorical or numerical features) and an ML-task (such as classification or regression), we aim to generate a data-centric ML pipeline P that maximizes the



```
### 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
```

```python
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score

train_data = pd.read_csv("train_data.csv")
test_data = pd.read_csv("test_data.csv")

categorical_features = ["Experience", "Gender", "Address", "Zip"]
categorical_transformer = Pipeline(steps=[
      ("imputer", SimpleImputer(strategy="most_frequent")),
      ("onehot", OneHotEncoder(handle_unknown="ignore"))])

preprocessor = ColumnTransformer(
      transformers = [("cat", categorical_transformer, categorical_features)],
      remainder="drop" )

model = RandomForestRegressor(n_estimators=100, random_state=42)
pipeline = Pipeline(steps=[ ("preprocessor", preprocessor), ("model", model)])

X_train = train_data.drop("Salary", axis=1)
y_train = train_data["Salary"]
X_test = test_data.drop("Salary", axis=1)
y_test = test_data["Salary"]

pipeline.fit(X_train, y_train)
y_test_pred = pipeline.predict(X_test)

test_r2 = r2_score(y_test, y_test_pred)
print(f"Test $R^2$: {test_r2}")
```

Figure 3: An example CatDB-generated prompt (instructions and refined metadata) for the Salary dataset, along with the resulting pipeline. Our dataset-specific instructions guide the LLM in generating code based on our refined metadata analysis.

task-specific accuracy on validation data. In detail, the dataset $\mathcal{D} = \{(\mathbf{X}_i, \mathbf{y}_i)\}_{i=1}^{n}$ comprises rows of d-dimensional feature vectors $\mathbf{X}_i$ and corresponding labels $\mathbf{y}_i$. For classification, $\mathbf{y}_i$ is categorical (either binary or multi-class), whereas for regression, $\mathbf{y}_i$ is numeric. The column names are $C = \{c_1, c_2, ..., c_d\}$ of type string.

We split the problem of *data-catalog-guided, LLM-based generation of data-centric ML pipelines* into three sub-problems to facilitate debuggability and extensibility, as shown in Figure 2.

Data Profiling: If metadata is unavailable, CatDB profiles the dataset to obtain basic metadata. For every column, we extract the *schema* (column name and data type), *number of distinct values*, *number of missing values*, *basic statistics* (e.g., min/max, median), and *feature types* (e.g., categorical, list). Additionally, we leverage LLMs to refine and verify feature types and clean categorical values.

Prompt Construction: CatDB generates LLM prompts by combining data catalog information in various ways (Table 1). We create structured prompts for *data pre-processing*, *feature engineering*, *model selection*, and *hyper-parameter tuning* using relevant metadata. Each prompt includes an instruction for a coding task in the form of *rule* messages (**R**) to guide the output and *schema* messages (**S**) with catalog details. These messages are integrated into an LLM-specific prompt template **T**, which can be a single prompt ($\beta = 1$) or a sequence ($\beta > 1$). CatDB is LLM-agnostic, with the aim of generating high-quality pipelines while minimizing hallucinations. Well-structured prompts improve consistency across LLMs, ensuring semantically similar outputs with modest variability.

Pipeline Generation: The LLM-generated Python ML pipeline requires refinement, validation, and execution. To ensure semantic correctness, CatDB uses dataset-specific instructions, guiding the LLM to perform well-defined coding tasks such as data cleaning (e.g., handling missing values) and data transformation (e.g., scaling numerical features) based on dataset metadata. This strategy improves the effectiveness of the generated pipeline. To further enhance reliability, a dedicated error management component detects and corrects syntactic and runtime errors, ensuring that the generated pipeline is both meaningful and executable.

Table 1: Metadata Combinations.

Data Profiling	Combinations of Data Profiling Items										
Items	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11
Schema	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Distinct Value Count		✓				✓	✓				✓
Missing Value Frequency			✓			✓		✓	✓		✓
Basic Statistics				✓			✓	✓		✓	✓
Categorical Values					✓				✓	✓	✓
User Description (optional)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

User API: Finally, we provide users of CatDB with a programmatic way to interact with the system's data catalog and functionality. This API consists of a function that accepts specific parameters to retrieve or manipulate data, allowing users to control the data access and processing according to their needs.

```
md = catdb_collect(M) /* collect metadata */
llm = LLM (model, clinet_url, config) /* config LLM */
P = catdb_pipgen(md,llm)
/* P.code: source code of generated pipeline */
/* P.results: outputs of pipeline's execution */
```

3 CATALOG AND PROMPT CONSTRUCTION

In this section, we describe the key algorithms for data catalog metadata collection, refinement, and projection, as well as efficient rule extraction, and the overall prompt construction algorithm. We also provide examples to convey the underlying intuitions.

3.1 Data Catalog

Data catalog information is a rich source for constructing informative prompts. Algorithm 1 presents the pseudo-code for the offline extraction of metadata from datasets. We pass the dataset D and an optional parameter τ_1 that specifies the number of samples to store in the data catalog for each column. The sample helps avoid redundant data access during prompt construction.

Column Metadata: We first extract the columns (Line 1) and initialize a dictionary for column metadata (Line 2). Subsequently, we iterate through all columns (Line 3) and extract their data type,

Algorithm 1 PROFILING($\mathcal{D}, \tau_1$)

```
Input: Dataset $\mathcal{D}$, Number of Samples $\tau_1$
Output: Data Profiling $\mathcal{P}$
 1: cols ← GETCOLUMNS($\mathcal{D}$)                              // get dataset columns.
 2: $\mathcal{P}$ ← [dict()]$_{1\times|\text{cols}|}$
 3: for $c \in cols$ do                                           // iterate over dataset columns.
 4:     $\mathcal{P}[c]_{\text{dataType, isCategorical}}$ ← GETCOLUMNTYPE($\mathcal{D}, c$)
 5:     $\mathcal{P}[c]_{\text{distinctionPercentage}}$ ← GETDISTINCTIONPERCENTAGE($\mathcal{D}, c$)
 6:     $\mathcal{P}[c]_{\text{missingPercentage}}$ ← GETMISSINGPERCENTAGE($\mathcal{D}, c$)
 7:     $\mathcal{P}[c]_{\text{IDs}}$ ← GEINCLUSIONDEPENDENCIES($\mathcal{D}, c$)
 8:     $\mathcal{P}[c]_{\text{similarities}}$ ← GETSIMILARITIES($\mathcal{D}, c$)
 9:     $\mathcal{P}[c]_{\text{correlations}}$ ← GETCORRELATIONS($\mathcal{D}, c$)
10:     $\mathcal{P}[c]_{\text{samples}}$ ← GETSAMPLES($\mathcal{D}, c, \tau_1$)
11:     $\mathcal{P}[c]_{\text{statistics}}$ ← GETNUMERICALSTATISTICVALUES($\mathcal{D}, c$)
12: return $\mathcal{P}$
```

feature types, distinct and missing values percentages (Lines 4-6). Extracting column dependencies are complex tasks and hence, our system adopts a simpler approach. We create column embeddings (i.e., vectors of length 300) and use these embedding to extract metadata like inclusion dependencies, similarities, and column correlations (Lines 7-9). This approach yields faster processing (a few seconds) with minor degradation in accuracy. For categorical columns, we store all unique values as column samples (Line 10), and for non-categorical and numerical columns, we randomly select τ_1 values as column samples. Finally, we calculate statistics (e.g., min/max and median) only for non-categorical columns (Line 11).

3.2 Data Catalog Refinements

CatDB profiles datasets to collect attribute types (e.g., string, number) and categorizes them as ML feature types (e.g., *Categorical* and *List*). We leverage LLMs to infer feature types and enhance the collected metadata. Our refined feature types and cleaned values significantly enhance ML model accuracy in many cases. Figure 4 illustrates the workflow of our catalog refinement, data cleaning, and extraction of a refined dataset, which we will discuss below.

Categorizing Sentence Data Types: We first identify a list of string feature as candidates for categorical feature types. These candidates might include values in different representations, and both mixed data and missing values could have hindered accurate type identification. CatDB refines two types of values. First, we separate, refine, and represent composite data as separate features. For example, the Address attribute in Figure 1 is a mix of zip codes and states, and thus, split into State and Zip. Second, we identify and split sentence features into individual values, each hashed to a new numerical feature. For example, the Skills attribute in Figure 1 is identified as a list feature and split into new features in Figure 5. Additionally, LLMs can infer feature types with just the attribute name and a few samples (10 in our system). We observed that the refined values exhibit fewer distinct values, which facilitates the identification as categorical. For example, the Experience attribute in Figure 5 was transformed into a short list of categorical values.

Refining Categorical Data: Additionally, we refine categorical values that are semantically equivalent (e.g., Figure 1 Gender attribute). The list of distinct items of a categorical feature is usually small, and thus, we submit the entire list to the LLM to obtain a mapping of refined to original values. In case of many distinct items, we perform this process batch-wise for robustness.

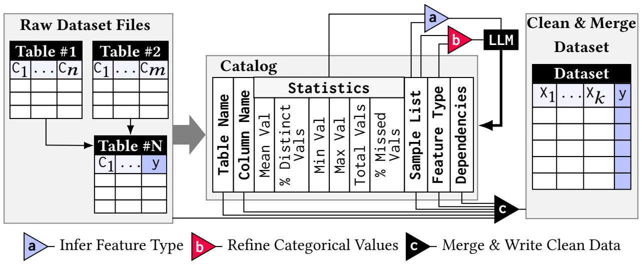
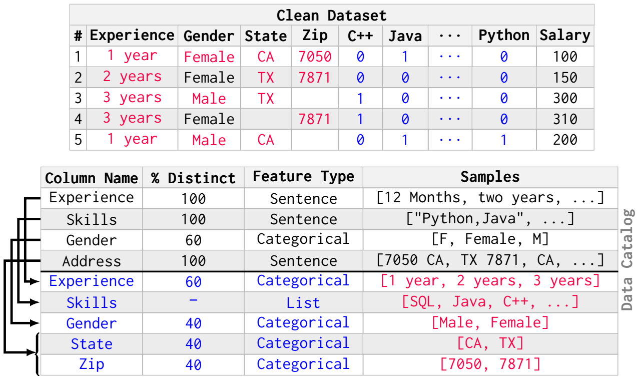


Figure 4: Data Catalog Refinements & Data Preparation.



Clean Dataset									
#	Experience	Gender	State	Zip	C++	Java	· · ·	Python	Salary
1	1 year	Female	CA	7050	0	1	· · ·	0	100
2	2 years	Female	TX	7871	0	0	· · ·	0	150
3	3 years	Male	TX		1	0	· · ·	0	300
4	3 years	Female		7871	1	0	· · ·	0	310
5	1 year	Male	CA		0	1	· · ·	1	200

Column Name	% Distinct	Feature Type	Samples
Experience	100	Sentence	[12 Months, two years, ...]
Skills	100	Sentence	["Python,Java", ...]
Gender	60	Categorical	[F, Female, M]
Address	100	Sentence	[7050 CA, TX 7871, CA, ...]
Experience	60	Categorical	[1 year, 2 years, 3 years]
Skills	–	List	[SQL, Java, C++, ...]
Gender	40	Categorical	[Male, Female]
State	40	Categorical	[CA, TX]
Zip	40	Categorical	[7050, 7871]

Figure 5: Example of Data Catalog Update & Data Cleaning.

Materializing Prepared Data: After completing the refinement process, we update and overwrite the input dataset. In detail, we apply the mapping of categorical features values and join multi-table datasets into a single table. In this stage, we are not considering missing value imputation, outlier removal, and feature selection, which we consider during pipeline generation.

3.3 Metadata Projection and Rule Definition

With the metadata at hand, we can construct prompts to guide LLMs in generating pipelines. Our prompts consist of (1) the *schema and metadata* (we filter and project relevant metadata), and (2) *rules* (we define LLM tasks as rules based on data characteristics). Algorithm 2 sketches this metadata extraction and rules definition.

Schema and Metadata (S): Through the dataset schema and column metadata, we guide the LLM. We also encode the column names, which can enable the LLM to find correlations similar to schema matching. If column names are not available, we add more meaningful metadata, such as data type, feature type, statistics, and samples. Furthermore, we include the number of missing values and distinct values for choosing relevant pre-processing steps (see Lines 2-6). As a basis for choosing the top-K columns as part of feature engineering, we also encode column dependencies as metadata. Finally, we specify the label y as the target column (Line 6).

Rule Definition (R): We guide the LLM via rules with two goals. First, we aim to *reduce errors and improve performance.* CatDB employs a knowledge base to identify sources of errors and poor model performance. Accordingly, we provide the LLM with a list of rules for more directed search. Second, we aim for an *open-ended task guidance* without dictating specific steps (e.g., XGBoost as the ML model). Instead, we guide the LLM towards considering certain primitives, which allows better options. There are four types of essential system rules that can be inferred from a data catalog:

Algorithm 2 METADATAANDRULES(C)

```
Input: Columns $C$(contains catalog data)
Output: Schema & Metadata $\mathbf{S}$, Rules $\mathbf{R}$
 1: // a) Extracting Schema and Metadata for Saving in S
 2: for $c \in C$ do                              // iterate over dataset columns
 3:     $\mathbf{S}[c] \leftarrow \{\text{cat}[c]_{\text{name, dataType, isCategorical, statistics, samples}}\}$
 4:     $\mathbf{S}[c] \leftarrow \bigcup\{\text{cat}[c]_{\text{distinctionPercentage, missingPercentage, IDs}}\}$
 5:     if c ∈ {target column} then
 6:         $\mathbf{S}[c] \leftarrow \bigcup\{\text{"target column"}\}$
 7: // b) Generating Rules for Saving in R
 8: if $C$ ∈ {has missing values} then
 9:     $\mathbf{R}^{\text{preprocessing}} \leftarrow$ GETMISSINGVALUEIMPUTATIONRULE($C$)
10: $\mathbf{R}^{\text{preprocessing}} \leftarrow \bigcup$ GETDATAAUGMENTATIONRULE($C$)
11: if $C$ ∈ {labels are imbalance} then
12:     $\mathbf{R}^{\text{preprocessing}} \leftarrow \bigcup$ GETREBALANCINGRULE($C$)
13: $\mathbf{R}^{\text{fe-engineering}} \leftarrow \bigcup$ GETFEATUREEXTRACTIONRULE($C$)
14: $\mathbf{R}^{\text{fe-engineering}} \leftarrow \bigcup$ GETFEATURESELECTIONRULE($C$)
15: $\mathbf{R}^{\text{model-selection}} \leftarrow \bigcup$ GETMODELSELECTIONRULE($C$)
16: return $\mathbf{S}, \mathbf{R}$
```

- *Data Preparation* rules describe feature refinements (e.g., for missing values, feature normalization, outlier removal). A generated pipeline will select appropriate methods (e.g., `most-frequent` value for categorical features).
- *Feature Dependency* rules encode the value extraction from features or their relationships (e.g., for feature selection).
- *Feature Filter* rules remove features with low relevance or correlation, which is useful for multi-table data with missing values and unnecessary features after consolidation.
- *Data Augmentation* rules rely on catalog statistics and target features. In small or imbalanced datasets, we guide LLMs to add data augmentation before training (Lines 8-15).

3.4 Overall Prompt Construction

Putting it all together, we now describe the overall prompt construction algorithm (see Algorithm 3) and its remaining limitations.

Initialization: The algorithm is invoked with the dataset $\mathcal{D}$, the LLM $\mathcal{M}$, and two optional parameters α and β that specify the number of top-K columns impacting the target and a number of chain prompts. We load the data catalog information in Line 1, remove any unnecessary columns in Line 2, and also remove empty and constant columns, as well as columns containing few non-null values (e.g., columns with values in less than 2% of rows).

Metadata Projection: Furthermore, we select the top α features (Line 3) if the user specifies this parameter. In order to do so, we encode column dependencies as metadata. CatDB projects metadata, focusing on patterns crucial for pipeline generation. The metadata is organized in the following order: 1) categorical, 2) features highly correlated with the target but with missing values, 3) sentence, 4) numerical, and 5) boolean features. The top-K algorithm prioritizes categorical features and, if space permits, selects from other groups. We found that categorical metadata is vital for pipeline generation because the feature types identified can differ from those in the data catalog. For instance, a feature with 7 distinct integer values might be identified as numerical and a candidate for normalization. However, the data catalog may highlight it as a categorical feature suitable for feature hashing and one-hot encoding. Other feature groups can be generalized with brief prompts (e.g., features `v1-v20` are numerical with 5% missing values), allowing pipeline pre-processing to recognize them as specified in the data catalog.

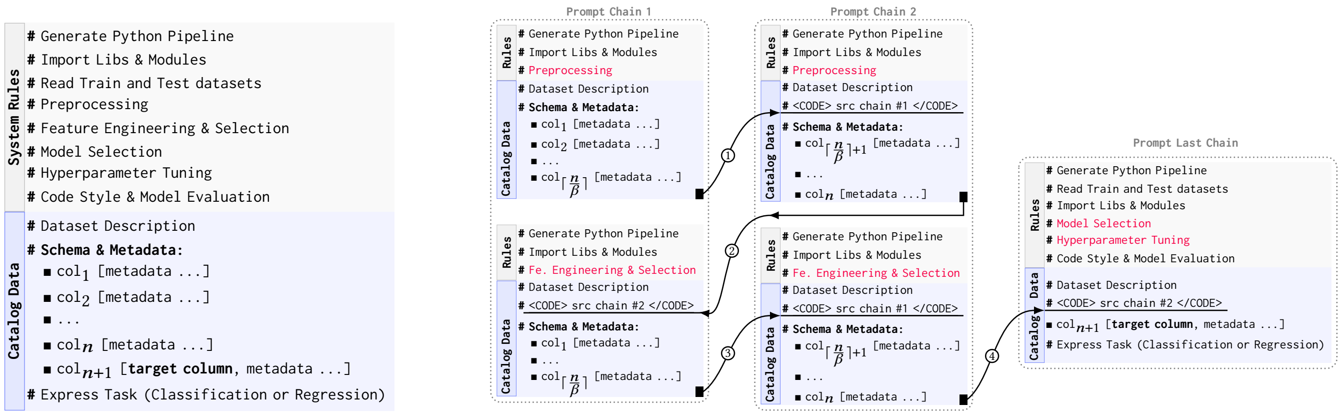
Algorithm 3 PROMPT($\mathcal{D}, \mathcal{M}, \alpha, \beta$)

```
Input: Dataset $\mathcal{D}$, LLM $\mathcal{M}$, Top-K Columns $\alpha$, Chains $\beta$
Output: Prompt(s) $\mathcal{P}$
 1: c ← LOADDATACATALOG($\mathcal{D}$)            // load datasets's data catalog.
 2: c ← CLEANDATACATALOG(c)           // remove unnecessary columns.
 3: c ← SELECTTOPKCOLUMNS(c, $\mathcal{M}, \alpha$)          // select Top-K columns.
 4: if $\beta == 1$ then // all columns' metadata & rules together (CatDB).
 5:     S, R ← METADATAANDRULES(c)    // get S and R for selected cols.
 6:     $\mathcal{P} \leftarrow$ FORMATPROMPT($S, R, \mathcal{M}$)         // format prompt question.
 7: else                          // split tasks, S, and R for CatDB Chain.
 8:     $\mathcal{P} \leftarrow [\,]_{1\times(\beta+1)}$;       $k \leftarrow \lceil \frac{|c|}{\beta} \rceil$
 9:     for $i \in \{1, \cdots, \beta\}$ do             // iterating over column chunks.
10:         S, R ← METADATAANDRULES($c[(i-1) \times k : min(i \times k, |c|)]$)
11:         $\mathcal{P}[i]^{\text{preprocessing}} \leftarrow$ FORMATPROMPT($S, R^{\text{preprocessing}}, \mathcal{M}$)
12:         $\mathcal{P}[i]^{\text{fe-engineering}} \leftarrow$ FORMATPROMPT($S, R^{\text{fe-engineering}}, \mathcal{M}$)
13:     S′ ← <CODE>preprocessing & fe-engineering</CODE>
14:     R′ ← METADATAANDRULES$^{\text{model-selection}}$(c)
15:     $\mathcal{P}[\beta+1]^{\text{model-selection}} \leftarrow$ FORMATPROMPT($S', R', \mathcal{M}$)
16: return $\mathcal{P}$
```

CatDB Default: The user can specify the number of chains via β: if $\beta = 1$, we will construct a single prompt without consideration of LLM limitations. By default, we combine **S** and **R** to form the final prompt, which is returned in Lines 4-6.

CatDB Chain: For $\beta > 1$ (called CatDB Chain), we split the data catalog information into β chunks, each containing k columns (see Line 8). We construct pre-processing and feature-engineering prompts for each chunk (Lines 9-12), but only one model selection prompt (Line 15). The model selection prompt includes the results of pre-processing and feature engineering tasks (generated Python code) as well as the rules for model selection based on the target column (Lines 13-14). Figure 6 shows the prompt templates of CatDB ($\beta = 1$) and CatDB Chain ($\beta = 2$). In CatDB Chain, we first submit all pre-processing prompts, followed by feature engineering prompts. In each chain, we append the previous result to the new prompt. This strategy incrementally updates the pipeline based on the prompt rules and metadata. We adopted this strategy for two reasons: First, a chunking of columns may lose column dependencies. By appending the source of the pipeline, the LLM can interpret the source and successfully distinguish the column dependencies. Second, we verify each pipeline step independently, simplifying error detection and correction. However, this approach increases token costs as each prompt includes the context.

Handling Prompt Limitations: Despite successful construction of prompts, CatDB faces a systemic limitation of LLM message sizes. If datasets are small and the final prompt fits the LLM's limitations, the invocation only requires a single prompt ($\beta = 1$). However, for large datasets with many features, the prompt exceeds the LLM's maximum token limit. To overcome this limitation, we introduce two approaches. First, we reduce the number of features via the parameter α, which trades model performance due to feature selection with substantially reduced costs. Second, CatDB Chain divides the metadata into smaller subtasks, projects metadata, and chains the results ($\beta > 1$). While this strategy requires multiple LLM invocations (increased time and cost), it is effective for much larger datasets and provides robust error management.


System Rules
# Generate Python Pipeline
# Import Libs & Modules
# Read Train and Test datasets
# Preprocessing
# Feature Engineering & Selection
# Model Selection
# Hyperparameter Tuning
# Code Style & Model Evaluation
Catalog Data
# Dataset Description
# Schema & Metadata:
■ col$_1$ [metadata ...]
■ col$_2$ [metadata ...]
■ ...
■ col$_n$ [metadata ...]
■ col$_{n+1}$ [target column, metadata ...]
# Express Task (Classification or Regression)
Prompt Chain 1
Rules
# Generate Python Pipeline
# Import Libs & Modules
# Preprocessing
Catalog Data
# Dataset Description
# Schema & Metadata:
■ col$_1$ [metadata ...]
■ col$_2$ [metadata ...]
■ ...
■ col$_{\lceil \frac{n}{\beta} \rceil}$ [metadata ...]
Rules
# Generate Python Pipeline
# Import Libs & Modules
# Fe. Engineering & Selection
Catalog Data
# Dataset Description
# <CODE> src chain #2 </CODE>
# Schema & Metadata:
■ col$_1$ [metadata ...]
■ ...
■ col$_{\lceil \frac{n}{\beta} \rceil}$ [metadata ...]
Prompt Chain 2
Rules
# Generate Python Pipeline
# Import Libs & Modules
# Preprocessing
Catalog Data
# Dataset Description
# <CODE> src chain #1 </CODE>
# Schema & Metadata:
■ col$_{\lceil \frac{n}{\beta} \rceil+1}$ [metadata ...]
■ ...
■ col$_n$ [metadata ...]
Rules
# Generate Python Pipeline
# Import Libs & Modules
# Fe. Engineering & Selection
Catalog Data
# Dataset Description
# <CODE> src chain #1 </CODE>
# Schema & Metadata:
■ col$_{\lceil \frac{n}{\beta} \rceil+1}$ [metadata ...]
■ ...
■ col$_n$ [metadata ...]
1
2
3
4
Prompt Last Chain
Rules
# Generate Python Pipeline
# Read Train and Test datasets
# Import Libs & Modules
# Model Selection
# Hyperparameter Tuning
# Code Style & Model Evaluation
Catalog Data
# Dataset Description
# <CODE> src chain #2 </CODE>
■ col$_{n+1}$ [target column, metadata ...]
# Express Task (Classification or Regression)

(a) CatDB Prompt Template ($\beta = 1$).

(b) CatDB Chain Prompt Template ($\beta = 2$).

Figure 6: CatDB and CatDB Chain (Two Chains as Examples) with Ordering of Submitted Prompts.

4 PIPELINE GENERATION AND VALIDATION

CatDB generates an initial pipeline by submitting the constructed prompt to the LLM and validating the output. Our validation includes a syntax check for correctness and a runtime check on a local dataset. This mechanism facilitates efficient error management, and we also conduct code analysis to identify and refine any missing steps in the pipeline (see again, Figure 3 for an example).

4.1 Overall Pipeline Generation

We aim to generate a high-quality, data-centric ML pipeline (no errors, accurate, and fast) with only few LLM interactions. Accordingly, we generate *stateless* prompts—that are self-contained—which simplifies the re-submission of prompts for individual tasks. Moreover, we encode catalog information, such as file formats, delimiters, and statistics (e.g., ratio of missing values) to guide the LLM. This metadata is more concise than example records.

Algorithm Description: Algorithm 4 shows the overall CatDB algorithm with a single prompt. For CatDB Chain, the entire algorithm is repeated for each chain, passing the result of a chain to the next. We first construct the prompt in Line 3 and submit it to the LLM in Line 2 for obtaining the pipeline code. In Lines 3-15, we attempt to debug and refine the generated pipeline by incorporating both local and LLM systems. The maximum number of attempts is limited by τ_2 to ensure termination and control costs. Lines 7-15 systematically check for the different error types, where we aim to locally fix errors (Line 8), but fall back to LLM-based error handling for unknown errors (Lines 10-14). Finally, we refine or fix problematic parts of the pipeline, and return the pipeline.

Example 1 (Pipeline Generation). *Figure 3 illustrates a combination of rules and metadata that guides the LLM in adding step-specific code. The file path and format of the dataset allows the LLM to generate a CSV reader in Lines 8-9. For pre-processing (Lines 10-12), the pipeline spends no time extracting columns with missing values because we already know and encode the number of data points and the ratio of not-null values into the prompt. Similarly, for feature engineering, we guide the LLM with a list of categorical columns as well as their distinct item ratios and values. The pipeline comprises an* `OneHotEncoder` *and removes unnecessary columns after feature engineering. The prompt mentions the target feature and asks for training a regressor in Lines 15-18. Finally, our rules guided the LLM to pick a* `RandomForestRegressor` *with fixed hyper-parameters.*

Algorithm 4 PIPEGEN($\mathcal{D}, \mathcal{M}, \alpha, \tau_2$)

```
Input: Dataset $\mathcal{D}$, LLM $\mathcal{M}$, Top-K Columns $\alpha$, Maximum Attempts $\tau_2$
Output: Pipeline Source Code $SRC$
 1: p ← PROMPT($\mathcal{D}, \mathcal{M}, \alpha, \beta = 1$)        // dataset prompt Algorithm 3.
 2: src ← SUBMITPROMPTTOLLM(p, $\mathcal{M}$)
 3: for $i \in \tau_2$ do                                  // iterate to fix pipeline error.
 4:     err ← PARSEANDEXECUTE(src)
 5:     if err = ∅ then
 6:         break
 7:     else if err ∈ {Knowledge-base Error} then
 8:         src ← FIXKNOWLEDGEBASEERORRANDAPPLYPATCH(src, err)
 9:     else if err ∈ {Syntax Error} then
10:         p ′ ← ERRORPROMPT($\mathcal{D}, \mathcal{M}$, src, err)   // syntax error prompt.
11:         src ← SUBMITPROMPTTOLLM(p′, $\mathcal{M}$)
12:     else
13:         cat ← GETCATALOGDATA($\mathcal{D}$, err)  // filter & project metadata.
14:         p ′ ← ERRORPROMPT($\mathcal{D}, \mathcal{M}$, src, err, cat)   // run. err. prompt.
15:         src ← SUBMITPROMPTTOLLM(p′, $\mathcal{M}$)
16: if ¬VERIFYPIPELINECODE(src) then
17:     src ← HANDCRAFTPIPELINE(src)
18: return src
```

Algorithm Cost Analysis: Finally, consider a pipeline prompt P_p of rules and metadata, and an error prompt P_e of pipeline code and the error. The incurred costs with a single-prompt CatDB are:

$$C(P_p, P_e, \gamma, \tau_2) = \gamma \mathcal{L}(P_p) + \sum_{i=1}^{\gamma} \sum_{j=1}^{\tau_2} \mathcal{L}(P_{e_{ij}}), \qquad (1)$$

where $\mathcal{L}(x)$ is the number of tokens, γ is the number of LLM interactions, and τ_2 is the maximum number of LLM error correction attempts. $P_{e_{ij}}$ (iteration i, attempt j) depends on the size of the error message. Furthermore, the costs of CatDB Chain are:

$$C_{chain} = C(P_m, P_e, \gamma, \tau_2) + \sum_{p \in \{P_d, P_f\}} \sum_{i=1}^{\beta} C(p_i, P_{e\,i}, \gamma, \tau_2), \qquad (2)$$

where P_d, P_f, and P_m denote the individual pre-processing, feature engineering, and model selection prompts.

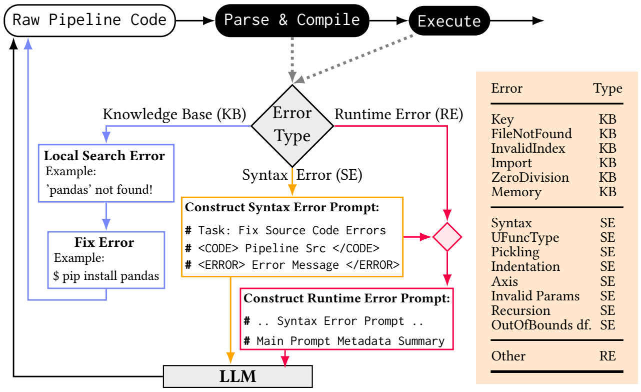


Figure 7: Error Management & Classification of Error Types.

4.2 Validation and Pipeline Error Management

Although we guide the LLM with metadata to generate correct and effective pipelines, errors are generally unavoidable. Randomization means even fixed prompts can yield varying errors. To tackle this issue, CatDB comprises a dedicated error management component (see Figure 7), which we describe in detail below.

Types of Errors: Analyzing request logs from various LLMs, we identified 23 types of errors (Figure 8 shows their relative occurrence frequencies), categorized into three groups: (i) Environment & Package Errors: Pipelines run in a basic, pre-installed environment. The CatDB Knowledge Base (KB) API manages six error types, such as missing packages, which it resolves by installing dependencies and re-executing the pipeline. (ii) Syntax & Parse Errors (SE): Using the `ast` [3] library, we parse and execute pipelines, automatically handling issues like missing imports, uncommented text, and indentation. If unresolved, we resubmit the pipeline to the LLM. These errors occur in <3% of cases and are typically fixed in one iteration. (iii) Runtime & Semantic Errors (RE): Pipelines are tested on sample data. Most errors (85%) arise from missing metadata or unavailable data. LLM assistance, incorporating catalog details (e.g., column types), resolves these typically within four iterations.

Error Correction: The CatDB KB API provides a cost-effective and locally executable solution. However, both SE and RE may necessitate additional LLM interactions. Instead of our initial prompt templates, we use dedicated prompt templates for error correction. As shown in Figure 7, these templates combine (1) the source code of the erroneous pipeline (in `<CODE>` tags), (2) the error message with line numbers (in `<ERROR>` tags), and (3) a summary of the original prompt (including metadata relevant to the errors solely for RE). Table 2 shows the error distributions of our substantial error traces we collected across various datasets, pipelines, and LLMs over an extended system development period. Most errors are related to RE and highlight the need to improve initial prompts and enhance KB API handling to reduce generation costs.

Table 2: Error Distributions of Error Trace Dataset.

LLM	Total Requests	KB [%]	SE [%]	RE [%]
Llama3.1-70b	20,868	2.464	2.907	94.629
Gemini-1.5 pro	10,041	21.213	2.092	76.695

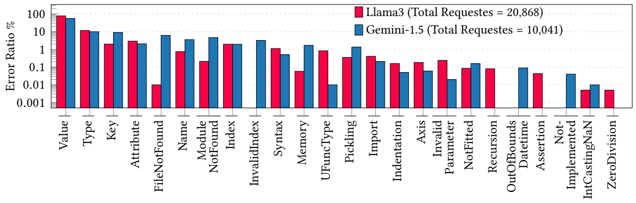


Figure 8: Ratio and Distribution of Errors.

Guarantees: CatDB gives the guarantee that there are no silent errors, unknowingly corrupting accuracy. Combining syntax checks and runtime checks on local validation data, we verify the functionality and accuracy of the generated ML pipeline. In case of errors, the combination of a local knowledge base of errors&solutions as well as iterative LLM interactions for error correction have proven very effective. By adding rare remaining errors to the knowledge base, any manual error correction became the exception.

4.3 System Limitations

CatDB leverages LLMs to generate ML pipelines tailored to specific datasets, but there are remaining limitations that constitute interesting directions for future work. First, we support basic data cleaning tasks such as the handling of null values, duplicate values, and outliers, but there are also many unsupported pre-processing steps (e.g., entity resolution and data augmentation from data lakes). Second, CatDB is designed for supervised learning tasks where we leverage the accuracy on the validation data as a signal. Future extensions should include pre-processing and unsupervised tasks for tabular, time-series, and image data. Third, we do not yet enforce library constraints on pipeline generation. Organizations may have restrictions on certain libraries, and thus, we should enforce lists of allowed/disallowed libraries for compliance.

5 EXPERIMENTS

We study our CatDB framework on a variety of real-world datasets with different data characteristics, examining both existing and non-existing pipelines, and different LLMs. The primary insights are that: (1) CatDB generates pipelines with competitive runtime, cost, and evaluation performance, and (2) CatDB is robust with regard to errors and large datasets with many features.

5.1 Experimental Setting

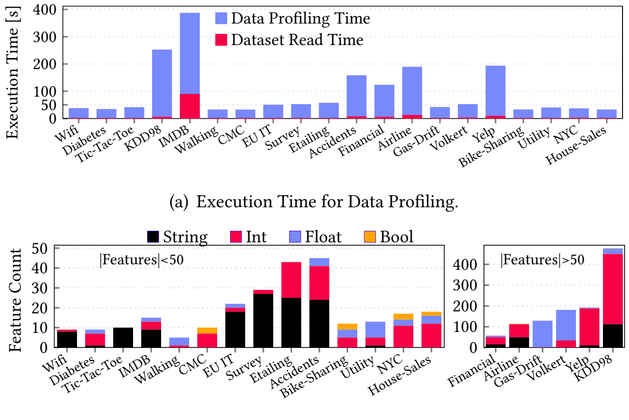
HW/SW Environment: We ran all experiments on a server node (VM) with an Intel Core CPU (with 32 vcores) and 148 GB of DDR4 RAM. The software stack consisted of Ubuntu 22.04, OpenJDK 11 (for Java baselines), and Python 3.10 (for Python baselines).

Implementation Details: The entire CatDB system is implemented in Python and utilizes the data profiling sub-system of KGLiDS [35]. For generating pipeline code, we use the OpenAI API [6] for GPT-4o, the Groq cloud [5] service for Llama3.1-70b, and Google AI Studio [4] for Gemini-1.5-pro. The latency of LLMs was negligible, lasting only a few seconds (analyzed via dedicated tools [2]). We further utilize an automatic method for extracting required packages and creating local environments.

Datasets: Table 3 shows the real-world datasets used for all of our experiments. These datasets encompass three task types (binary/multi-class classification, and regression), single/multi-table datasets, and different data characteristics. We divided all datasets into 70/30 train and test sets. For CatDB, we profile the datasets,

Table 3: Used Datasets and their Data Characteristics.

ID	Dataset	#Tables	*n* (nrow)	*m* (ncol)	Dataset Type	#Classes
1	Wifi	1	98	9	Binary	2
2	Diabetes	1	768	9	Binary	2
3	Tic-Tac-Toe	1	958	10	Binary	2
4	IMDB	7	30,530,313	15	Binary	2
5	KDD98	1	82,318	478	Binary	2
6	Walking	1	149,332	5	Multiclass	22
7	CMC	1	1,473	10	Multiclass	3
8	EU IT	1	1,253	23	Multiclass	148
9	Survey	1	2,778	29	Multiclass	9
10	Etailing	1	439	44	Multiclass	5
11	Accidents	3	954,036	46	Multiclass	6
12	Financial	8	552,017	62	Multiclass	4
13	Airline	19	445,827	115	Multiclass	3
14	Gas-Drift	1	13,910	129	Multiclass	6
15	Volkert	1	58,310	181	Multiclass	10
16	Yelp	4	229,907	194	Multiclass	9
17	Bike-Sharing	1	17,379	12	Regression	869
18	Utility	1	4,574	13	Regression	95
19	NYC	1	581,835	17	Regression	1,811
20	House-Sales	1	21,613	18	Regression	4,028



(a) Execution Time for Data Profiling.

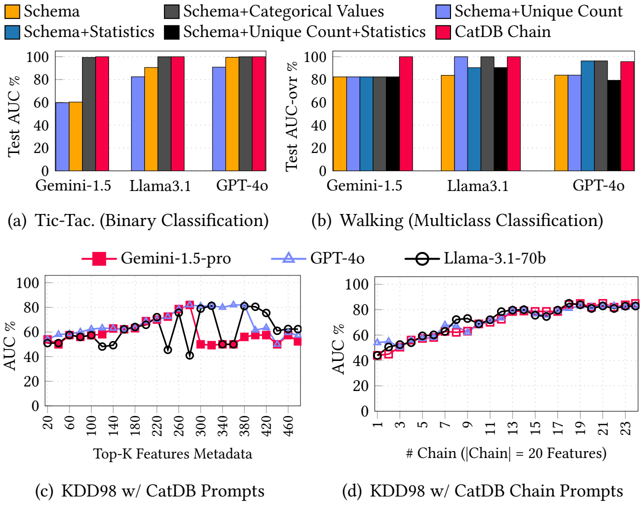
(b) Data Type Distribution.

Figure 9: Data Profiling and Data Type Distribution.

extract their metadata, and store this metadata in our data catalog. Figure 9(a) shows the runtime of this offline data profiling. Our system takes ≈ 6 *min* for large datasets and < 50*s* for small ones. Figure 9(b) further shows the distribution of data types. We observe a good mix of numerical, textual, and categorical features.

Baseline Comparisons: We compare our CatDB framework with state-of-the-art systems in three settings:

- *LLM-based Baselines:* AIDE [69] and AutoGen [82] are state-of-the-art, end-to-end LLM-based solution generators for ML tasks. Additionally, CAAFE [37] a semi-automated ML task generation system, uses LLMs for automatic feature engineering in small tabular datasets. While CAAFE and AIDE only support the OpenAI LLMs, we extended their functionality to support Llama and Gemini, as well as other models beyond TabPFN [36] (e.g., RandomForest) for CAAFE.
- *AutoML Tools:* We compare CatDB with AutoML systems designed for tabular data. We prepared all datasets in a way to be compatible with the following AutoML tools: AutoGluon [24], H2O [46], Flaml [80], Auto-Sklearn (for regression) [27], and Auto-Sklearn 2.0 (for classification) [25].



(a) Tic-Tac. (Binary Classification)

(b) Walking (Multiclass Classification)

(c) KDD98 w/ CatDB Prompts

(d) KDD98 w/ CatDB Chain Prompts

Figure 10: Metadata Impact on Pipeline Performance.

- *AutoML-based Workflows:* Since AutoML primarily focuses on model selection, we added two pre-processing steps: data cleaning w/ SAGA [74] and Learn2Clean (L2C) [15], and data augmentation w/ ADASYN [33] for classification and Imbalanced Learning Regression [83]. The pre-processed datasets were then submitted to the AutoML tools. Pre-processing was only done on the training set, while model performance was evaluated on the unaltered test sets.

5.2 Prompt Construction and Metadata Impact

To study the impact of metadata, we compose different configurations as shown in Table 1, and evaluate the pipeline accuracy on three different datasets and task types. Figure 10 shows the performance of CatDB with gradually increasing metadata.

Pipeline Quality: Figure 10 shows micro benchmark results for the impact of metadata and the quantity of these meta data. First, more metadata does not always improve pipeline quality. Simple schema metadata (feature names and types) often matches or outperforms schemas with additional statistics (Figures 10(a), 10(c)). CatDB carefully combines metadata and instructions to enhance performance, as excessive metadata can lead LLMs to ignore tasks. When prompts lack specific instructions tied to relevant metadata (e.g., handling missing values), LLMs may default to arbitrary pre-processing, such as random imputation. Second, LLMs struggle with overly large metadata inputs. In Figure 10(c), we vary K for the top-K feature metadata selection. Exceeding 260 features caused very large prompts and led to ignored rules. As shown in Figure 10(d), CatDB Chain mitigates this issue by splitting tasks and feature metadata, ensuring effective pipeline generation. Third, due to randomness, there is some variation in performance even with a fixed prompt for different meta data configurations, but CatDB Chain achieved consistently high performance compared to all other configurations. Fourth, fixed metadata is only effective for homogeneous datasets (e.g., numerical-only with basic statistics) but falls short for heterogeneous datasets with mixed feature types. CatDB's metadata selection adapts to dataset characteristics, leading to >20% performance gains over Gemini configurations

Table 4: Catalog Refinement and Data Cleaning (6 Datasets Refined: 1-14 Columns, LLM = Gemini-1.5).

Method	EU IT														Wifi				Etailing						Survey		Utility	Yelp	
	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#1	#2	#3	#4	#1	#2	#3	#4	#5	#6	#1	#2	#1	#1	#2
Original	563	256	148	64	119	53	59	46	24	11	48	12	15	4	69	5	4	2	256	121	71	90	134	14	1008 (sentence)	9	199	2060	61
CatDB	100	43	45	32	95	39	47	36	16	3	43	6	9	3	15	4	3	1	18	17	31	66	127	10	160 (categorical)	8	132	512	55

Table 5: Performance Comparison of Six Cleaning Datasets (LLM = Gemini-1.5, ✶: CatDB Config, ✢: CatDB Chain Config).

Dataset	CatDB/CatDB Chain				CAAFE				AIDE		AutoGen		AutoML						AutoML w/ Workflow						
	Original		Refined		TabPFN		R.Forest						H2O		Flaml		Autogluon		Clean	H2O		Flaml		Autogluon	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Method	Train	Test	Train	Test	Train	Test
EU IT✶	**100.0**	39.2	**100.0**	**91.8**	0.1	N/A	0.1	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	90.5	44.8	L2C	N/A	N/A	N/A	N/A	89.9	32.6
Wifi✶	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	56.5	79.8	61.5	N/A	N/A	64.2	64.2	84.3	61.0	84.9	73.7	90.6	61.0	SAGA	99.9	60.6	98.4	64.2	97.9	65.8
Etailing✶	**100.0**	68.5	**100.0**	**99.9**	99.8	71.0	79.6	69.9	**100.0**	69.7	**100.0**	63.4	**100.0**	53.1	79.4	64.9	96.9	67.7	L2C	99.9	64.0	**100.0**	65.2	98.5	72.4
Survey✶	**100.0**	92.2	**100.0**	97.1	99.5	97.7	88.5	88.5	99.7	90.6	**100.0**	**98.1**	84.9	84.3	98.8	**98.1**	99.2	93.4	SAGA	83.0	75.3	99.3	88.2	99.1	85.6
Utility✶	99.8	98.8	99.8	98.8	Dosen't support				99.8	99.0	98.8	98.8	N/A	N/A	**99.9**	99.0	**99.9**	**99.5**	SAGA	40.6	4.6	No trained models			
Yelp✢	**99.9**	66.8	**99.9**	**98.2**	Out of Mem.		65.8	65.5	N//A	N/A	76.2	75.8	92.5	77.2	85.5	76.7	84.4	81.9	L2C	81.1	64.6	90.0	70.1	96.8	89.0

using the same metadata (Figures 10(a) and 10(b)). Finally, despite significant performance differences across LLM models and tasks, CatDB Chain consistently yields high accuracy.

5.3 Catalog Refinement and Data Cleaning

In order to evaluate the data catalog refinement and initial cleaning, we perform dedicated micro-benchmarks.

Number of Distinct Items: Table 4 shows the LLM-based catalog refinement, with 1 to 14 updates. We ran these experiments with all LLMs, observed negligible differences, and thus, report the Gemini results. We distinguish categorical and list features, where lists are highlighted with gray background. For instance, in Yelp column #1, values such as "Golf,Roofing,Movers", "Movers,Taxis" and "Taxis,Golf" were joined and treated as categorical values. Our system identifies the feature type as a list, extracts unique values ({Golf,Roofing,Movers,Taxis}) and applies k-hot encoding. For other columns (not highlighted), CatDB deduplicates the values, where we see a systematic reduction of distinct items.

Accuracy Impact: Table 5 compares the accuracy of original and refined versions on six datasets, including a comparison with workflows of state-of-the-art data cleaning and AutoML tools. CatDB consistently improves the performance, particularly on datasets with data quality issues, achieving up to 52% gains. First, on EU IT, categorical features caused L2C to fail due to the absence of continuous columns. The target feature had semantically identical but differently formatted duplicates, leading to imbalanced labels and incorrect upsampling. AutoGluon performed better on the original data, while AIDE, AutoGen, and CAAFE removed missing-value features, reducing sample size and impairing model building. Second, on Etailing, performance was initially comparable across methods, as all relied on basic LLM decisions. Our cleaning (catalog refinement) step improved accuracy by 30% due to eliminating duplicate values correlated with the target feature. Third, on Yelp, hashed days and timestamps were misinterpreted as missing values. SAGA and L2C applied imputation, distorting the distribution, while LLM-based methods failed to recognize list features, leading to poor performance. For Wifi, Survey, and Utility, improvements stemmed from better categorical value handling and deduplication. In Wifi, CatDB refined a highly correlated categorical feature and removed a constant-value feature, while in Survey, a feature was transformed from a sentence to a categorical feature. Overall, CatDB shows very good and robust train and test accuracy.

Table 6: Runtime Comparison of Six Cleaning Datasets.

Dataset	CatDB [s]		CAAFE [s]		AIDE	AutoGen	Cleaning + Aug.
	Original	Refined	TabPFN	R.Forest	[s]	[s]	[s]
EU IT	4.5	1.4	132.2	238.4	N/A	N/A	203.6 + 1.6
Wifi	0.8	0.8	86.30	89.5	N/A	48.92	200.4 + 5.2
Etailing	1.6	1.7	178.9	98	40	245.6	330 + 1.6
Survey	1.8	1.4	297.4	114.7	40	39.2	2,086.8 + 5.5
Utility	4.2	3.8	N/A	N/A	40	29.9	863.6 + 2,755.1
Yelp	584.4	2,477.4	N/A	1,029.5	N/A	323	41,400.6 + 83.8

Costs and Runtime: CatDB shows major cost and runtime differences to other LLM-based baselines. CatDB's costs are influenced by three factors: First, CatDB's uses a two-step process for refining categorical values and integrating data catalog information, which adds overhead. Second, CatDB provides only moderately detailed metadata to the LLM, whereas CAAFE includes schema and 10 samples per feature, which incurs higher costs for datasets with many features. AIDE and AutoGen rely on concise, human-generated descriptions, but this approach shifts effort to human oversight. Third, CatDB boosts the pipeline quality by incorporating error-handling mechanisms, which further increases cost. In contrast, AIDE and AutoGen resubmit prompts, while CAAFE skips feature engineering when errors occur. Finally, Table 6 compares pipeline execution times (excluding generation time) across six datasets. CatDB substantially outperforms CAAFE by leveraging cleaned data and avoiding computationally expensive pipeline primitives. On Yelp as an exception, one-hot encoding increased data size and processing time. Overall, CatDB outperforms all baselines, on almost all datasets by more than an order of magnitude.

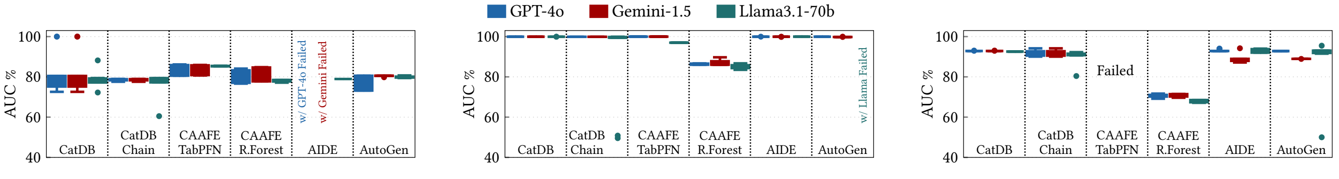
5.4 Pipeline Performance with 10 Iterations

To study CatDB's and CatDB Chain's pipeline generation, we generate and execute ML pipelines for three datasets using three LLMs with 10 iterations of prompt executions.

Quality of Generated Pipelines: We evaluate pipeline quality over 10 iterations to account for randomness, even with LLM temperature set to zero. Figure 11 compares our method with LLM-based baselines in terms of AUC scores. The CAAFE framework—which uses a fixed pre-processing stage, LLM-driven feature engineering, and a fixed classifier (TabPFN)—performs well with minimal variance (Figures 11(a) and 11(b)). However, CAAFE struggles with high-dimensional feature spaces, such as the Volkert dataset (Figure 11(c)), and its performance drops when replacing TabPFN with RandomForest as a model with better scalability. The AIDE

Table 7: Performance Comparison of 8 Datasets (Iteration = 1, OOM: Out of Memory, TO: Time Out). L2C Preprocessing: Decimal Scale Normalization (DS), Exact Duplicate (ED), Approximate Duplicate (AD), Inter Quartile Range (IQR), Local Outlier Factor (LOF), Expectation-Maximization (EM) and MEDIAN Imputations, DROP Sample.

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

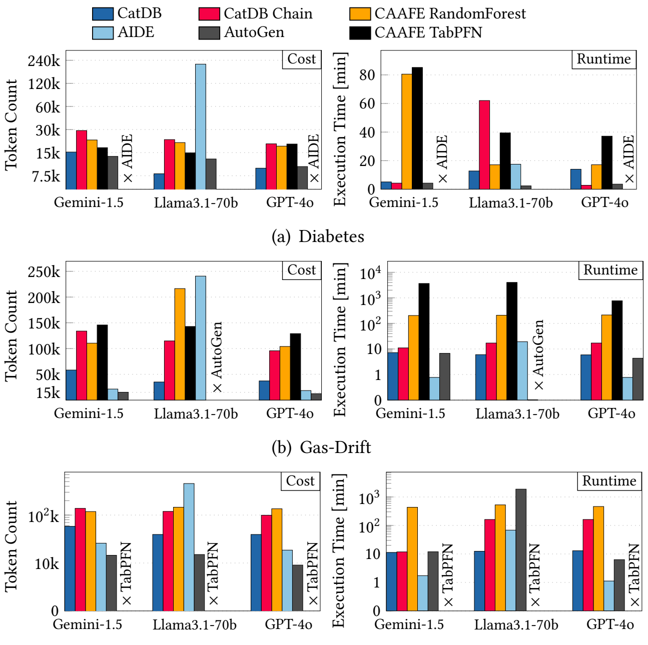


(a) Diabetes (**CatDB Chain** Selected)

(b) Gas-Drift (**CatDB** Selected)

(c) Volkert (**CatDB Chain** Selected)

Figure 11: Performance Comparison of Three Datasets w/ LLM-based Baselines (10 iterations).



(a) Diabetes

(b) Gas-Drift

(c) Volkert

Figure 12: Cost and Runtime Comparison of Three Datasets and Different LLMs (10 iterations).

framework shows similar performance but lacks stability across LLMs, failing on the Diabetes dataset (Figure 11(a)) and underperforming on Volkert (Figure 11(c)) with Gemini. AutoGen also faces instability, requiring human intervention and failing to generate a pipeline for the Gas-Drift dataset after 15 attempts with the Llama model. In contrast, CatDB delivers comparable or better performance, albeit with higher variance, due to LLM-based pre-processing, model selection, and independent interactions.

Costs and Runtime of Generated Pipelines: Figure 12 (Cost part) shows the tokens consumed, where CatDB is more cost-efficient than CatDB Chain, and both outperforming CAAFE. CatDB's cost stems from output size (1-2K tokens), while CAAFE is dominated by input tokens. AIDE's concise metadata and task descriptions reduce initial costs but increase with iterative resubmissions when errors occur. Its efficiency depends on the LLM: successful generation minimizes cost (e.g., Figures 12(b) and 12(c) with Gemini), while failures lead to more tokens (e.g., Figures 12(a) and 12(b) with Llama). AutoGen's approach mirrors CatDB, while human-assisted frameworks consume fewer tokens (Figures 12(a) and 12(c)). Figure 12 (Runtime part) compares total runtime for pipeline generation and execution. On the small Diabetes dataset (Figure 12(a)), CatDB and CatDB Chain achieve an 8x to 14x speedup over CAAFE, with an even larger gap for datasets with many features (Figures 12(b) and 12(c)) due to pre-processing before feature engineering and model construction. Compared to AIDE

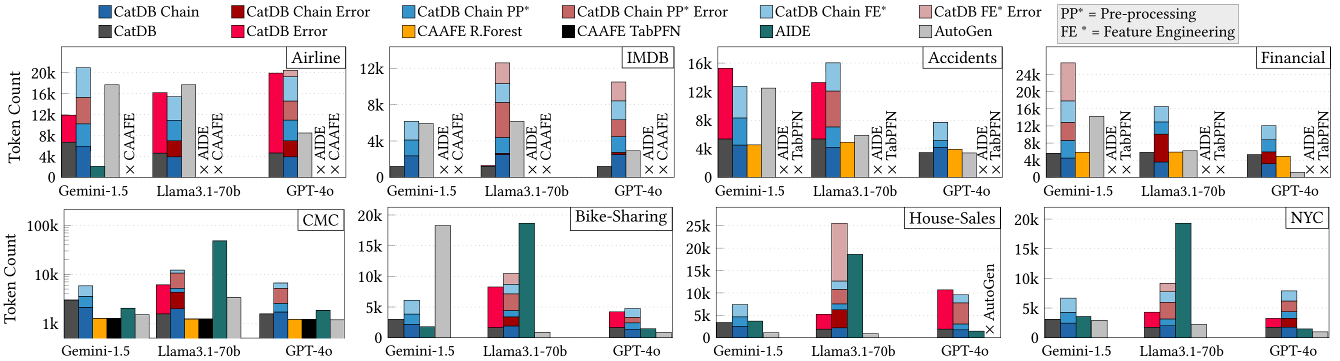
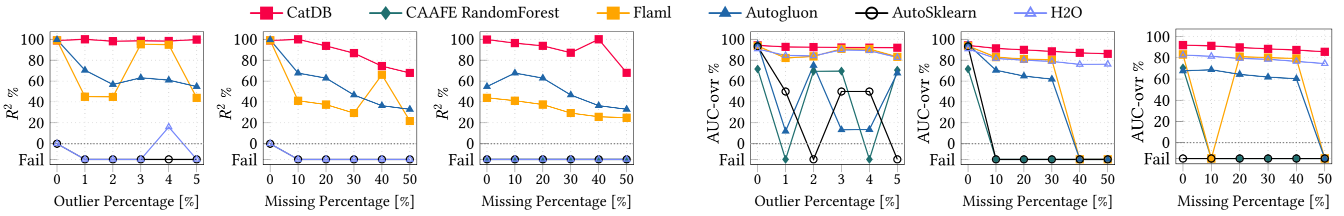


Figure 13: Cost Comparison of 8 Datasets and Different LLMs (1 iteration).



(a) Utility Outlier (b) Utility Missing Value (c) Utility MV + Outlier=5% (d) Volkert Outlier (e) Volkert Missing Value (f) Volkert MV + Outlier=5%

Figure 14: End-to-End Experiments with Outlier and Missing Value Injection (Gemini-1.5).

and AutoGen, CatDB generates more detailed instructions and error handling, increasing processing time. In contrast to AIDE and AutoGen—whose runtime variability depends on the LLM performance—CatDB shows consistently moderate end-to-end runtime and much smaller pipeline runtime.

5.5 Pipeline Performance with Single Iteration

We now compare CatDB with both LLM-based baselines and AutoML tools, for a single LLM repetition but up to 15 error correction iterations. The time budget of AutoML tools (which we run for multiple repetitions) was set to the measured CatDB runtime.

Quality of Generated Pipelines: Table 7 shows the results for all classification (binary/multi-class) and regression tasks. Overall, we see reliable performance of CatDB and CatDB Chain, even compared to many 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. CatDB Chain yields generally better performance for larger datasets—where the task splitting ensures all tasks are represented properly (e.g., CatDB missed feature engineering for CMC) and a reduced number of errors. In contrast, CAAFE TabPFN failed on large datasets, and many AutoML tools ran into out-of-memory errors or timeouts.

Costs of Generated Pipelines: We further reevaluate the token consumption including initial prompts and error handling across ten datasets. For CAAFE, AIDE, and AutoGen, only the total token count was considered. Figure 13 shows that CatDB and CAAFE have comparable costs, while CatDB Chain sometimes incurs higher costs. Error management costs for CatDB and CatDB Chain vary significantly across LLMs, with higher costs for regression tasks and multi-table datasets. Most of CatDB Chain's costs are due to error management, especially with the Llama model, which struggled to maintain the system conversation but eventually converged.

Table 8: Runtime of 8 Datasets Across Different LLMs [s].

Baseline	Gemini-1.5			Llama3.1-70b			GPT-4o		
	Fail	AVG	SUM	Fail	AVG	SUM	Fail	AVG	SUM
CatDB	0	7.5	60.1	0	7.4	58.9	0	19.6	156.5
CatDB Chain	0	8.1	65.1	0	7.4	59.5	0	14.5	116.1
CAAFE TabPFN	7	~~7.4~~	~~7.4~~	7	~~1.3~~	~~1.3~~	7	~~7.4~~	~~7.4~~
CAAFE R.Forest	5	~~34.2~~	~~102.7~~	5	~~22.7~~	~~68.0~~	5	~~22.6~~	~~67.9~~
AIDE	3	~~0.8~~	~~4.2~~	4	~~17.3~~	~~69.3~~	4	~~0.7~~	~~2.7~~
AutoGen	0	11.8	94.4	0	7.4	58.9	1	~~48.7~~	~~340.7~~

End-to-end Generation Runtime: Table 8 shows the end-to-end runtime across eight datasets and various LLMs, including the number of failed datasets (Fail), the average (AVG) and total (SUM) runtimes for successful datasets (where the runtimes are only comparable without failures). For CatDB, the reported runtime includes data loading, catalog refinement, metadata projection, rule definition, pipeline generation, error management, and execution. CAAFE succeeded on one small dataset with TabPFN and on three with RandomForest but failed on larger datasets after four days. Compared to the ten repetitions, CAAFE's single data preprocessing step dominates the runtime here. The AIDE and AutoGen baselines lack pre-processing and error management, and their runtime heavily dependent on LLMs. Llama-generated pipelines often defaulted to naïve grid search, which substantially increases the runtimes. These frameworks also spent more time retrying requests (AIDE up to 20 times, AutoGen up to 15) and executing grid search pipelines. In contrast, both CatDB and CatDB Chain, successfully run across all LLMs, datasets, and tasks. Additionally, CatDB's multi-threading rules further improved the runtime efficiency. Overall, CatDB achieves the best trade-off of good end-to-end runtime, robustness in terms of rare failures, very fast pipeline runtime, and high accuracy even with a single iteration.

5.6 Data-centric ML Pipelines

Finally, we study the impact of generating data-centric ML pipelines, compared to AutoML tools (without pre-processing in end-to-end experiments). AIDE and AutoGen were excluded due to frequent failures and the need for human-generated descriptions and interventions. We use the Utility (regression) and Volkert (classification) datasets and introduce outliers and missing values. Figure 14 shows the impact of pre-processing primitives on model quality. Increasing the ratio of outliers (from 0% to 5%), CatDB maintains good performance, whereas all AutoML tools deteriorate for more than 1% data corruptions. In contrast, both CatDB and some AutoML tools (Flaml and Autogluon) handle missing values quite well in regression (see Figure 14(b)). In classification (see Figures 14(e) and 14(f)), most AutoML tools and CAAFE deteriorate. We further generate mixed errors (Figure 14(c) and 14(f)) where most AutoML tools show again low performance. Generating data-centric ML pipelines with CatDB robustly yields high prediction quality.

6 RELATED WORK

Our work is related to data catalogs, LLM-guided artifact generation, data preparation for ML, and AutoML tools, which we survey here.

Data Catalogs and Refinements: Data catalogs are essential for data governance, research data management (according to FAIR data principles [81]), and data markets; with examples like Gaia-X data catalogs [76], Apache Atlas [11], and Google dataset search [14, 17, 32]. These catalogs store basic metadata, detailed data provenance [20, 61], and data profiles [9]. KGLiDS [35] interlinks data catalogs with abstract pipeline scripts and offers GNN-based automation for tasks like data cleaning and transformation, but it lacks LLM support for complete pipeline generation. Some catalogs also include index structures, embeddings, and data summaries for dataset discovery and augmentation [67]. Most state-of-the-art systems [8, 13, 24] rely on statistical and syntactic properties, which can lead to incorrect feature type assignments in noisy data. Recent approaches [71] improve feature type inference and deduplication by training ML models on manually labeled data [72], but still require manual labeling. In contrast, we leverage LLMs to infer feature types and refine metadata, demonstrating improved performance across diverse datasets and utilizing this information for guiding LLM artifact generation.

LLM-guided Generation: LLMs like GPT [18], Gemini, and Llama are extensively used to generate artifacts such as source code [31, 38], queries [29], data wrangling programs [55], data analysis pipelines, and visualizations, as well as for question answering [51]. This generation process allows for materialization, scrutiny, and correction before deployment. Typically, LLMs are integrated into data management systems by serializing data entries and predicting masked tokens. Recently, GIDCL [84] combines LLMs with Graph Neural Networks to improve data cleaning, whereas SMARTFEAT [50] uses LLMs to generate new features from contextual information and external knowledge. CAAFE [37] focuses on feature engineering but struggles to handle large datasets. In contrast, CatDB generates scalable, end-to-end data-centric ML pipelines.

Data Preparation for ML: Data preparation methods in terms of data validation [68], cleaning [30, 48, 74], and augmentation [22, 45] as well as feature engineering [63, 70] are crucial for high quality, data-centric ML pipelines. First, data validation [12, 68, 78] summarizes data characteristics and validates if expectations are satisfied through constraints. Data visualization tools, such as Facets [65] and TFDV [23, 62], further allow the semi-manual identification of errors and anomalies. Second, data cleaning involves two main tasks: error detection and correction. Tools like ActiveClean [44], CPClean [41], and Learn2Clean [15] focus on specific models or error types, such as missing values. Li et al. explored the impact of basic data cleaning techniques on ML model accuracy [48]. Other methods include evolutionary algorithms for finding effective data cleaning pipelines [74] and feature engineering with fairness constraints [56, 66]. Third, data enrichment adds features to boost model quality. Tools like FeatNavigator [49], SOS [43], and SANTOS [42] automate the addition of relevant features from relational tables by evaluating their importance for the downstream ML task. Some frameworks use reinforcement learning for data augmentation [22] of additional synthetic data points, derived from a small labeled dataset. Although these standalone methods are very effective, they cannot be seamlessly composed due to system boundary crossing and specialized internal enumeration procedures. In contrast, CatDB first refines data catalog information (e.g., removing duplicate categories) with iterative LLM interactions, and then guides the LLM through metadata and instructions to generate appropriate data-centric ML pipelines. We run the LLM-generated lightweight ML pipelines without additional iterative search procedures, yielding efficient and scalable execution plans.

AutoML Tools: Automating the process of model and feature selection through AutoML tools is a well-studied problem [19, 26, 28, 47, 85]. Examples include Auto-Weka [79], Auto-Sklearn [25], Auto-Keras, TPOT [58], H2O-AutoML [46], TuPaQ [75], Alpine Meadow [73], KGpip [34], and AlphaD3M [52]. Most of these tools focus primarily on model selection though. Recently, dedicated frameworks aim to find data augmentation pipelines via reinforcement learning [22], data cleaning pipelines through evolutionary algorithms [74], and feature engineering decisions under fairness and other constraints [56, 66]. In contrast, CatDB generates data-centric ML pipelines in a data-catalog-guided and LLM-based manner.

7 CONCLUSIONS

To summarize, we introduced CatDB as a holistic system for the data-catalog-guided, LLM-based generation of effective and efficient data-centric ML pipelines. Key technical contributions are techniques for incorporating data catalog information and rules into the LLM prompts, data catalog refinements, prompt chaining, and error handling. We draw two major conclusions. First, the LLM-based generation of data-centric ML pipelines with metadata from data catalogs yields competitive pipeline accuracy with more efficient pipelines compared to specialized tools for data cleaning as well as AutoML tools. Second, dedicated pipeline validation and error handling with a knowledge base of common errors and LLM specifics, ensures reliable and trustworthy pipelines. Interesting future work includes (1) a more fine-grained, task-specific encoding of metadata into prompts, (2) generating high-performance and scalable ML pipelines (e.g., distributed or specialized accelerators), and (3) LLM-based reasoning agents for natural language insights and automated explanations in data-centric ML pipelines.

REFERENCES

[1] 2021. State of Data Science and Machine Learning. https://www.kaggle.com/kaggle-survey-2021

[2] 2024. Artifici Alanalysis. https://artificialanalysis.ai/models/mixtral-8x7b-instruct/providers#summary

[3] 2024. ast: Abstract Syntax Trees. https://docs.python.org/3/library/ast.html

[4] 2024. Google AI Studio. https://aistudio.google.com/

[5] 2024. Groq Cloud. https://console.groq.com/

[6] 2024. OpenAI. https://platform.openai.com/

[7] 2024. Scikit-learn: Machine Learning in Python. https://scikit-learn.org/stable/

[8] 2024. TransmogrifAI: Automated Machine Learning for Structured Data. https://github.com/salesforce/TransmogrifAI

[9] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. 2017. Data Profiling: A Tutorial. In *SIGMOD*. 1747–1751. https://doi.org/10.1145/3035918.3054772

[10] AI@Meta. 2024. Llama 3 Model Card. (2024). https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md

[11] Apache Atlas. 2020. Open Metadata Management and Governance. https://atlas.apache.org/.

[12] Florian Bachinger, Lisa Ehrlinger, Gabriel Kronberger, and Wolfram Wöß. 2024. Data Validation Utilizing Expert Knowledge and Shape Constraints. *ACM J. Data Inf. Qual.* 16, 2 (2024), 13:1–13:27. https://doi.org/10.1145/3661826

[13] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, Chiu Yuen Koo, Lukasz Lew, Clemens Mewald, Akshay Naresh Modi, Neoklis Polyzotis, Sukriti Ramesh, Sudip Roy, Steven Euijong Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, and Martin Zinkevich. 2017. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In *SIGKDD*. 1387–1395. https://doi.org/10.1145/3097983.3098021

[14] Omar Benjelloun, Shiyu Chen, and Natasha F. Noy. 2020. Google Dataset Search by the Numbers. In *ISWC*, Vol. 12507. 667–682. https://doi.org/10.1007/978-3-030-62466-8_41

[15] Laure Berti-Équille. 2019. Learn2Clean: Optimizing the Sequence of Tasks for Web Data Preparation. 2580–2586. https://doi.org/10.1145/3308558.3313602

[16] Matthias Boehm, Iulian Antonov, Sebastian Baunsgaard, Mark Dokter, Robert Ginthör, Kevin Innerebner, Florijan Klezin, Stefanie N. Lindstaedt, Arnab Phani, Benjamin Rath, Berthold Reinwald, Shafaq Siddiqui, and Sebastian Benjamin Wrede. 2020. SystemDS: A Declarative Machine Learning System for the End-to-End Data Science Lifecycle. In *CIDR*. http://cidrdb.org/cidr2020/papers/p22-boehm-cidr20.pdf

[17] Dan Brickley, Matthew Burgess, and Natasha F. Noy. 2019. Google Dataset Search: Building a search engine for datasets in an open Web ecosystem. In *WWW*. 1365–1375. https://doi.org/10.1145/3308558.3313685

[18] Tom B. Brown et al. 2020. Language Models are Few-Shot Learners. In *NeurIPS*. https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[19] Girish Chandrashekar and Ferat Sahin. 2014. A survey on feature selection methods. *Computers & Electrical Engineering* 40, 1 (2014), 16–28. https://doi.org/10.1016/J.COMPELECENG.2013.11.024

[20] James Cheney, Laura Chiticariu, and Wang Chiew Tan. 2009. Provenance in Databases: Why, How, and Where. *Found. Trends Databases* 1, 4 (2009), 379–474. https://doi.org/10.1561/1900000006

[21] Jeffrey Cohen, Brian Dolan, Mark Dunlap, Joseph M. Hellerstein, and Caleb Welton. 2009. MAD Skills: New Analysis Practices for Big Data. *PVLDB* 2, 2 (2009), 1481–1492. https://doi.org/10.14778/1687553.1687576

[22] Ekin D. Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. 2019. AutoAugment: Learning Augmentation Strategies From Data. In *CVPR*. 113–123. https://doi.org/10.1109/CVPR.2019.00020

[23] Mike Dreves, Gene Huang, Zhuo Peng, Neoklis Polyzotis, Evan Rosen, and Paul Suganthan G. C. 2021. Validating Data and Models in Continuous ML Pipelines. *IEEE Data Eng. Bull.* 44, 1 (2021), 42–50. http://sites.computer.org/debull/A21mar/p42.pdf

[24] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander J. Smola. 2020. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. *CoRR* abs/2003.06505 (2020). https://arxiv.org/abs/2003.06505

[25] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2022. Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning. *J. Mach. Learn. Res.* 23 (2022). http://jmlr.org/papers/v23/21-0992.html

[26] Matthias Feurer and Frank Hutter. 2019. Hyperparameter optimization. *Automated machine learning: Methods, systems, challenges* (2019), 3–33.

[27] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and Robust Automated Machine Learning. In *NeurIPS*. 2962–2970. https://proceedings.neurips.cc/paper/2015/hash/11d0e6287202fced83f79975ec59a3a6-Abstract.html

[28] George Forman. 2003. An Extensive Empirical Study of Feature Selection Metrics for Text Classification. *Journal of Machine Learning Research* 3 (2003), 1289–1305. http://jmlr.org/papers/v3/forman03a.html

[29] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *CoRR* (2023). https://doi.org/10.48550/ARXIV.2308.15363

[30] Shubha Guha, Falaah Arif Khan, Julia Stoyanovich, and Sebastian Schelter. 2023. Automated Data Cleaning Can Hurt Fairness in Machine Learning-based Decision Making. In *ICDE*. 3747–3754. https://doi.org/10.1109/ICDE55515.2023.00303

[31] Chunxi Guo, Zhiliang Tian, Jintao Tang, Pancheng Wang, Zhihua Wen, Kang Yang, and Ting Wang. 2023. Prompting GPT-3.5 for Text-to-SQL with De-semanticization and Skeleton Retrieval *(Lecture Notes in Computer Science)*. Springer, 262–274. https://doi.org/10.1007/978-981-99-7022-3_23

[32] Alon Y. Halevy, Flip Korn, Natalya Fridman Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. 2016. Goods: Organizing Google's Datasets. In *SIGMOD*. 795–806. https://doi.org/10.1145/2882903.2903730

[33] Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li. 2008. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In *IJCNN*. 1322–1328. https://doi.org/10.1109/IJCNN.2008.4633969

[34] Mossad Helali, Essam Mansour, Ibrahim Abdelaziz, and et al. 2022. A Scalable AutoML Approach Based on Graph Neural Networks. *PVLDB* 15, 11 (2022). https://www.vldb.org/pvldb/vol15/p2428-helali.pdf

[35] Mossad Helali, Niki Monjazeb, Shubham Vashisth, Philippe Carrier, Ahmed Helal, Antonio Cavalcante, Khaled Ammar, Katja Hose, and Essam Mansour. 2024. KGLiDS: A Platform for Semantic Abstraction, Linking, and Automation of Data Science. In *ICDE*.

[36] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. 2023. TabPFN: A Transformer That Solves Small Tabular Classification Problems in a Second. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=cp5PvcI6w8_

[37] 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*. https://arxiv.org/pdf/2305.03403

[38] Abhinav Jain, Chima Adiole, Swarat Chaudhuri, Thomas W. Reps, and Chris Jermaine. 2023. Tuning Models of Code with Compiler-Generated Reinforcement Learning Feedback. *CoRR* (2023). https://doi.org/10.48550/ARXIV.2305.18341

[39] Michael I. Jordan. 2018. SysML: Perspectives and Challenges. In *MLSys*.

[40] Adam Tauman Kalai and Santosh S. Vempala. 2023. Calibrated Language Models Must Hallucinate. *CoRR* abs/2311.14648 (2023). https://doi.org/10.48550/ARXIV.2311.14648

[41] Bojan Karlas, Peng Li, Renzhi Wu, Nezihe Merve Gürel, Xu Chu, Wentao Wu, and Ce Zhang. 2020. Nearest Neighbor Classifiers over Incomplete Information: From Certain Answers to Certain Predictions. *PVLDB* 14, 3 (2020), 255–267. https://doi.org/10.5555/3430915.3442426

[42] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J. Miller, and Mirek Riedewald. 2023. SANTOS: Relationship-based Semantic Table Union Search. *SIGMOD* 1, 1 (2023), 9:1–9:25. https://doi.org/10.1145/3588689

[43] Jayoung Kim, Chaejeong Lee, Yehjin Shin, Sewon Park, Minjung Kim, Noseong Park, and Jihoon Cho. 2022. SOS: Score-based Oversampling for Tabular Data. In *SIGKDD*. 762–772. https://doi.org/10.1145/3534678.3539454

[44] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J. Franklin, and Ken Goldberg. 2016. ActiveClean: Interactive Data Cleaning For Statistical Modeling. *PVLDB* 9, 12 (2016), 948–959. http://www.vldb.org/pvldb/vol9/p948-krishnan.pdf

[45] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NeurIPS*. 1106–1114. https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

[46] Erin LeDell and Sebastien Poirier. 2020. H2o automl: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, Vol. 2020. ICML.

[47] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P. Trevino, Jiliang Tang, and Huan Liu. 2018. Feature Selection: A Data Perspective. *Comput. Surveys* 50, 6 (2018), 94:1–94:45. https://doi.org/10.1145/3136625

[48] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. 2021. CleanML: A Study for Evaluating the Impact of Data Cleaning on ML Classification Tasks. In *ICDE*. 13–24. https://doi.org/10.1109/ICDE51399.2021.00009

[49] Jiaming Liang, Chuan Lei, Xiao Qin, Jiani Zhang, Asterios Katsifodimos, Christos Faloutsos, and Huzefa Rangwala. 2024. FeatNavigator: Automatic Feature Augmentation on Tabular Data. *CoRR* abs/2406.09534 (2024). https://doi.org/10.48550/ARXIV.2406.09534

[50] Yin Lin, Bolin Ding, H. V. Jagadish, and Jingren Zhou. 2024. SMARTFEAT: Efficient Feature Construction through Feature-Level Foundation Model Interactions. In *CIDR*. www.cidrdb.org. https://www.cidrdb.org/cidr2024/papers/p72-lin.pdf

[51] Yiming Lin, Madelon Hulsebos, Ruiying Ma, Shreya Shankar, Sepanta Zeighami, Aditya G. Parameswaran, and Eugene Wu. 2024. Towards Accurate and Efficient Document Analytics with Large Language Models. *CoRR* abs/2405.04674 (2024). https://doi.org/10.48550/ARXIV.2405.04674

[52] Roque Lopez, Raoni Lourenço, Rémi Rampin, Sonia Castelo, Aécio S. R. Santos, Jorge Henrique Piazentin Ono, Cláudio T. Silva, and Juliana Freire. 2023. AlphaD3M: An Open-Source AutoML Library for Multiple ML Tasks. In *International Conference on Automated Machine Learning (PMLR)*, Vol. 224. 22/1–22. https://proceedings.mlr.press/v224/lopez23a.html

[53] Mark Mazumder et al. 2022. DataPerf: Benchmarks for Data-Centric AI Development. *CoRR* abs/2207.10062 (2022). https://doi.org/10.48550/ARXIV.2207.10062

[54] Rohan Mukherjee, Chris Jermaine, and Swarat Chaudhuri. 2020. Searching a Database of Source Codes Using Contextualized Code Search. *PVLDB* (2020), 1765–1778. https://doi.org/10.14778/3401960.3401972

[55] Avanika Narayan, Ines Chami, Laurel J. Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? *PVLDB* 16, 4 (2022), 738–746. https://doi.org/10.14778/3574245.3574258

[56] Felix Neutatz, Marius Lindauer, and Ziawasch Abedjan. 2024. AutoML in heavily constrained applications. *VLDB J.* 33, 4 (2024), 957–979. https://doi.org/10.1007/S00778-023-00820-1

[57] Luis Oala et al. 2023. DMLR: Data-centric Machine Learning Research - Past, Present and Future. *CoRR* abs/2311.13028 (2023). https://doi.org/10.48550/ARXIV.2311.13028

[58] Randal S. Olson and Jason H. Moore. 2016. TPOT: A Tree-based Pipeline Optimization Tool for Automating Machine Learning, Vol. 64. JMLR.org. http://proceedings.mlr.press/v64/olson_tpot_2016.html

[59] OpenAI. 2024. GPT-4o. (2024). https://openai.com/index/hello-gpt-4o/

[60] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. 2018. Efficient Neural Architecture Search via Parameter Sharing. In *ICML*, Vol. 80. 4092–4101. http://proceedings.mlr.press/v80/pham18a.html

[61] Arnab Phani, Benjamin Rath, and Matthias Boehm. 2021. LIMA: Fine-grained Lineage Tracing and Reuse in Machine Learning Systems. In *SIGMOD*. 1426–1439. https://doi.org/10.1145/3448016.3452788

[62] Neoklis Polyzotis, Martin Zinkevich, Sudip Roy, Eric Breck, and Steven Whang. 2019. Data Validation for Machine Learning. In *MLSys*, Vol. 1. 334–347. https://proceedings.mlsys.org/paper_files/paper/2019/file/928f1160e52192e3e0017fb63ab65391-Paper.pdf

[63] Danrui Qi, Jinglin Peng, Yongjun He, and Jiannan Wang. 2024. Auto-FP: An Experimental Study of Automated Feature Preprocessing for Tabular Data. 129–142. https://doi.org/10.48786/EDBT.2024.12

[64] Nikitha Rao, Jason Tsay, Kiran Kate, Vincent J. Hellendoorn, and Martin Hirzel. 2023. AI for Low-Code for AI. *CoRR* abs/2305.20015 (2023). https://doi.org/10.48550/ARXIV.2305.20015

[65] Sergey Redyuk, Zoi Kaoudi, Volker Markl, and Sebastian Schelter. 2021. Automating Data Quality Validation for Dynamic Data Ingestion. In *EDBT*. 61–72. https://doi.org/10.5441/002/EDBT.2021.07

[66] Ricardo Salazar, Felix Neutatz, and Ziawasch Abedjan. 2021. Automated Feature Engineering for Algorithmic Fairness. *PVLDB* 14, 9 (2021), 1694–1702. https://doi.org/10.14778/3461535.3463474

[67] Aécio S. R. Santos, Aline Bessa, Fernando Chirigati, Christopher Musco, and Juliana Freire. 2021. Correlation Sketches for Approximate Join-Correlation Queries. In *SIGMOD*. 1531–1544. https://doi.org/10.1145/3448016.3458456

[68] Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Bießmann, and Andreas Grafberger. 2018. Automating Large-Scale Data Quality Verification. *PVLDB* 11, 12 (2018), 1781–1794. https://doi.org/10.14778/3229863.3229867

[69] Dominik Schmidt, Yuxiang Wu, and Zhengyao Jiang. 2024. AIDE: Human-Level Performance in Data Science Competitions. https://www.weco.ai/blog/technical-report

[70] Vraj Shah, Jonathan Lacanlale, Premanand Kumar, Kevin Yang, and Arun Kumar. 2021. Towards Benchmarking Feature Type Inference for AutoML Platforms. In *SIGMOD*. 1584–1596. https://doi.org/10.1145/3448016.3457274

[71] Vraj Shah, Jonathan Lacanlale, Premanand Kumar, Kevin Yang, and Arun Kumar. 2021. Towards Benchmarking Feature Type Inference for AutoML Platforms. In *SIGMOD*. 1584–1596. https://doi.org/10.1145/3448016.3457274

[72] Vraj Shah, Thomas J. Parashos, and Arun Kumar. 2024. How do Categorical Duplicates Affect ML? A New Benchmark and Empirical Analyses. *PVLDB* 17, 6 (2024), 1391–1404. https://www.vldb.org/pvldb/vol17/p1391-shah.pdf

[73] Zeyuan Shang, Emanuel Zgraggen, Benedetto Buratti, Ferdinand Kossmann, Philipp Eichmann, Yeounoh Chung, Carsten Binnig, Eli Upfal, and Tim Kraska. 2019. Democratizing Data Science through Interactive Curation of ML Pipelines. In *SIGMOD*. 1171–1188. https://doi.org/10.1145/3299869.3319863

[74] Shafaq Siddiqi, Roman Kern, and Matthias Boehm. 2023. SAGA: A Scalable Framework for Optimizing Data Cleaning Pipelines for Machine Learning Applications. *Proc. ACM Manag. Data* 1, 3 (2023), 218:1–218:26. https://doi.org/10.1145/3617338

[75] Evan R. Sparks, Ameet Talwalkar, Daniel Haas, Michael J. Franklin, Michael I. Jordan, and Tim Kraska. 2015. Automating model search for large scale machine learning. In *SoCC*. 368–380. https://doi.org/10.1145/2806777.2806945

[76] Hubert Tardieu. 2022. Role of Gaia-X in the European Data Space Ecosystem. In *Designing Data Spaces: The Ecosystem Approach to Competitive Advantage*. 41–59. https://doi.org/10.1007/978-3-030-93975-5_4

[77] Gemini Team. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. https://doi.org/10.48550/arXiv.2403.05530

[78] Saravanan Thirumuruganathan, Nan Tang, Mourad Ouzzani, and AnHai Doan. 2020. Data Curation with Deep Learning. In *EDBT*. 277–286. https://doi.org/10.5441/002/EDBT.2020.25

[79] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In *SIGKDD*. 847–855. https://doi.org/10.1145/2487575.2487629

[80] Chi Wang, Qingyun Wu, Markus Weimer, and Erkang Zhu. 2021. FLAML: A Fast and Lightweight AutoML Library. In *MLSys*. https://proceedings.mlsys.org/paper/2021/hash/92cc227532d17e56e07902b254dfad10-Abstract.html

[81] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. 2016. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific data* 3 (2016). https://www.nature.com/articles/sdata201618

[82] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2024. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*. https://openreview.net/forum?id=uAjxFFing2

[83] Wenglei Wu, Nicholas Kunz, and Paula Branco. 2022. Imbalancedlearningregression-a python package to tackle the imbalanced regression problem. In *ECML PKDD*. 645–648.

[84] Mengyi Yan, Yaoshu Wang, Yue Wang, Xiaoye Miao, and Jianxin Li. 2024. GIDCL: A Graph-Enhanced Interpretable Data Cleaning Framework with Large Language Models. *SIGMOD* 2, 6 (2024), 236:1–236:29. https://doi.org/10.1145/3698811

[85] Li Yang and Abdallah Shami. 2020. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing* 415 (2020), 295–316. https://www.sciencedirect.com/science/article/pii/S0925231220311693