Synergizing Large Language Models and Knowledge-based Reasoning for Interpretable Feature Engineering

Anonymous Author(s)

Abstract

Feature engineering stands as a pivotal step in enhancing the performance of machine learning (ML) models, particularly with tabular data. However, traditional feature engineering methods are often time-consuming and requires case-by-case domain knowledge. In addition, as ML systems become more common, interpretability becomes increasingly important, especially among domain experts. To this end, we propose *ReaGen*, an automated feature engineering (AutoFE) approach that combines knowledge graphs (KGs) with large language models (LLMs) to generate interpretable features. *ReaGen* begins by symbolic *REAsoning* over KG to extract relevant information based on datasets description. Then, it uses an LLM to iteratively *GENerate* meaningful features based on the retrieved information and the datasets description. Finally, to overcome challenges such as hallucinations and handling long contexts typical in LLMs, our model performs logical reasoning on the KG to ensure that the generated features maintain interpretability. *ReaGen* provides Python code for automatic feature generation and detailed explanations of feature utility. It leverages both LLM's internal knowledge and retrieved information from KGs. Experiments on public datasets demonstrate that *ReaGen* significantly improves prediction accuracy while ensuring high interpretability through human-like explanations for each feature. This work highlights the potential of integrating LLMs and KGs in feature engineering, paving the way for interpretable ML models.

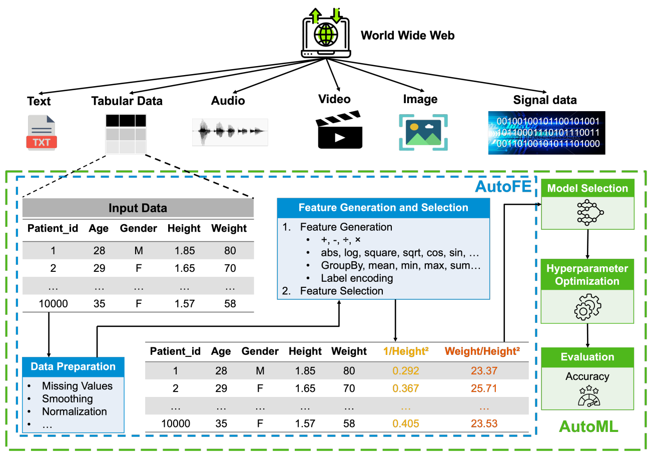
CCS Concepts

• **Computing methodologies** → **Machine learning algorithms**; • **Information systems** → **Semantic web description languages**; **Data mining**.

Keywords

Automated Feature Engineering, Tabular Data, Data Mining, Interpretable Feature Engineering

ACM Reference Format:
Anonymous Author(s). 2018. Synergizing Large Language Models and Knowledge-based Reasoning for Interpretable Feature Engineering. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 16 pages. https://doi.org/XXXXXXX.XXXXXXX



Input Data				
Patient_id	Age	Gender	Height	Weight
1	28	M	1.85	80
2	29	F	1.65	70
…	…	…	…	…
10000	35	F	1.57	58

Patient_id	Age	Gender	Height	Weight	1/Height²	Weight/Height²
1	28	M	1.85	80	0.292	23.37
2	29	F	1.65	70	0.367	25.71
…	…	…	…	…	…	…
10000	35	F	1.57	58	0.405	23.53

Figure 1: An overview of AutoML pipeline on tabular data.

1 Introduction

In today's computational environments, the majority of web applications rely heavily on the analysis and mining of tabular data[1], aka structured data. Over the past decades, ML has demonstrated impressive results with structured data in various domains, such as e-commerce, recommendation, marketing, risk management and more [7, 17, 43]. This success is often attributed to the experience of data scientists who leverage domain knowledge to extract meaningful patterns from data. This crucial yet tedious task is commonly referred to as Feature Engineering (FE). We illustrate in Figure 1, the different steps involved in ML pipeline. Arguably, feature engineering remains a significant bottleneck in the data science workflow, requiring up to 80% of the total time [14], as reported in the State of Data Science report[2]. Given the increasing computational capabilities and the rapid advancements in automated ML (AutoML), AutoFE has received an increasing interest, as it may reduce data scientists workload for quicker decision-making.

1.1 Main Challenges

Recently, several approaches of AutoFE have been proposed [2, 8–10, 12]. However, these approaches have several drawbacks. We categorize these challenges from 4 main perspectives: effectiveness, efficiency, applicability, and interpretability.

1.1.1 Effectiveness. (1) Lack of High-Order Features: AutoFE approaches, particularly learning-based methods, often fail to discover high-order transformations, focusing instead on simple, linear features, which can limit model performance. (2) Missing Feature Interactions: Existing approaches fail to explore interactions between

[1]https://db-engines.com/en/ranking
[2]https://www.anaconda.com/state-of-data-science-report-2023

features, missing out on potentially powerful feature combinations that could improve accuracy.

1.1.2 Interpretability. (1) Lack of Feature Semantics: AutoFE systems often fail to incorporate domain-specific knowledge, leading to features that lack meaning in their context, making them hard to interpret or justify to domain experts. (2) Lack of Feature Interpretability: Complex features generated by advanced methods are often difficult for humans to understand, compromising the transparency of the resulting models. In real-world applications, the ability to interpret the features driving model decisions is crucial.

1.1.3 Efficiency. (1) High Sample Complexity: Many existing methods operate at the level of individual features. While this approach makes the implementation straightforward, it tends to increase the complexity and overall runtime, making it impractical for large-scale applications. (2) Feature Explosion: Many prior works suffer from the feature explosion problem due to the number of candidates, escalating computational costs and raising the risk of overfitting.

1.1.4 Applicability. (1) Scalability: Existing frameworks are often tested on small datasets and struggle to handle larger datasets due to high resource demands. As a result, there is a significant gap between research-level AutoFE frameworks and their ability to handle real-world datasets. (2) Generalization: Features generated by existing approaches, especially reinforcement learning-based (RL) approaches, often fail to perform well on truly unseen datasets, raising concerns about their robustness in real-world applications.

Recently, Large Language Models (LLMs) have demonstrated fruitful progress across many applications, particularly when enhanced with sophisticated prompting strategies such as chain-of-thought (CoT) [37]. These models, trained on extensive web-crawled data across diverse tasks, offer the unique capability of adaptation to new tasks without necessitating task-specific fine-tuning. Consequently, LLMs have the potential to extend traditional AutoML tools by automating additional steps of the data science pipeline, particularly those involving contextual information such as FE. However, several challenges hinder their effective application in FE, including concerns regarding interpretability, possible hallucinations in model responses [4], and difficulties in handling tasks requiring multi-step and context-aware reasoning. These challenges stem from their reliance on internal representations to generate answers, which lack grounding in the external world and restrict their ability to reason reactively or update their knowledge.

1.2 Our proposed solution

To this end, we introduce *ReaGen*, an AutoFE approach that combines LLMs with knowledge-based reasoning to generate interpretable features. *ReaGen* begins by using the dataset's description as input to infer new relationships and relevant information from external KGs, which are then used as additional context. Leveraging the retrieved information and the dataset description, *ReaGen* employs LLMs to automatically generate Python code that creates meaningful features in order to improve the performance of downstream prediction tasks. To ensure feature interpretability and to reduce factual inaccuracies and hallucinations, our model includes a knowledge-based discriminator to discard non-interpretable features. *ReaGen* operates iteratively, providing feedback at each step based on the performance improvement and interpretability of the generated features (details in Section 4). Additionally, it offers human-like explanations for each generated feature, ensuring high interpretability for the end user.

In comparison with existing AutoFE approaches, *ReaGen* offers several advantages:

- **High-Order Features.** *ReaGen* extends beyond the limitations of learning-based methods [2, 9, 44] and approaches relying on pre-defined operators [10, 11], by supporting a broad range of operators. Through its operator-guided prompts, *ReaGen* enables the generation of complex and diverse feature transformations, moving past simple, linear constructions. Additionally, by incorporating external knowledge alongside the LLM's internal knowledge, *ReaGen* uncovers a richer set of high-order transformations, facilitating deeper exploration of feature interactions.
- **Leveraging external data.** *ReaGen* enhances performance by leveraging external knowledge. This capability addresses the challenge of feature semantics by incorporating meaningful domain-specific information, ensuring that generated features are contextually relevant and generalizable.
- **Improved Interpretability.** *ReaGen* enhances feature interpretability by using domain knowledge during feature generation. Unlike traditional AutoFE approaches that ignore context, our approach applies relevant operators to create features that are both meaningful and easier to interpret. Additionally, to avoid inaccuracies common in LLMs, we use a knowledge-based reasoning technique to filter out non-interpretable features, ensuring the generated features are clear and understandable for domain experts.
- **Generalization:** Our approach incorporates a broader range of contextual and external knowledge sources, allowing it to adapt feature engineering to different data environments. *ReaGen* ensures that generated features are robust and applicable across varying datasets, thereby enhancing their generalization capability in real-world applications.

Through extensive experiments on a wide range of datasets, *ReaGen* consistently achieves SOTA performance, significantly outperforming prior work. Additionally, several analytical experiments have demonstrated that our approach not only enhances interpretability but also improves scalability, validating its effectiveness in practical applications.

2 Related work

Existing AutoFE methods can be classified into 03 main classes.

Expansion-Reduction. This technique applies a large set of transformations to generate numerous features followed by feature selection and pruning. For instance, *FICUS* [20] and *TFC* [25] iteratively generate and select features using beam search and information gain, respectively. *Deep Feature Synthesis* [10] and *One Button Machine* [15] explore relational DBs to generate large feature sets. *Cognito* [13] improves search efficiency using BFS and DFS, while *AutoFeat* [8] uses down-sampling to focus searches in smaller regions. *SAFE* [31] enhances scalability with heuristic metrics, and *OpenFE* [40] uses a two-stage advanced pruning process to filter generated features. Despite these optimizations, expansion-reduction suffers from high computational costs, feature space explosion, and lack of convergence guarantees, making it less efficient for large datasets.

Learning-based. These approaches provide an efficient alternative to expansion-reduction by combining feature generation and selection into one process. They generate small batches of features and assess their usefulness through training and evaluation. *ExplorKit* [11] employs a learned ranking function to prioritize features but can be time-consuming. *LFE* [22] uses multi-layer perceptrons for transformation suggestions, but it is limited to classification. To explore more complex features, [12] apply Q-learning [35] on a transformation graph, and [9] use Monte Carlo tree search. *NFS* [2] employs RNNs to generate transformation sequences but incurs high computational costs due to the need for multiple RNNs. On the other hand, *DIFER* [44] uses evolutionary frameworks but often explores limited action spaces, reducing real-world applicability. Additionally, both focus on individual feature impacts, overlooking interactions between features. *FETCH* [16] improves upon this by learning a neural policy to generate feature pipelines tailored to specific datasets. However, learning-based methods still face challenges like heavy training requirements, limited interpretability, and scalability issues for large datasets.

AutoML-based. These approaches aim to reconstruct the feature space of original data, streamlining ML workflow from preprocessing to model selection. For instance, [1] introduced a hierarchical RL framework that classifies features and employs agents to select operations and features, assessing their quality through statistical interaction tests. This method rewards successful feature combinations but is computationally intensive due to the training of multiple agents. Another example is *AutoDS* [32], a web application that automates various ML tasks, allowing data scientists to upload datasets and receive suggestions for ML configurations, preprocessing, algorithm selection, and model training via user-friendly interfaces.

3 Problem Definition

In this work, we address the problem of engineering interpretable features by combining LLMs with knowledge-based reasoning.

3.1 Feature Interpretability

Building a model to generate interpretable features requires a clear definition of interpretability. While literature has acknowledged the importance of feature interpretability [3, 30, 45], there is now clear definition of what constitutes an interpretable feature. Since a formal definition could be elusive, we have sought insights from the field of psychology.

Definition 3.1. *(Interpretability). In general, to interpret means "to explain the meaning" or "to present in understandable terms".*

In psychology, interperetability refers to the ability to understand and make sense of an observation [18]. It involves making psychological theories and concepts comprehensible.

In logics, an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ consists of a nonempty set $\Delta^{\mathcal{I}}$ (the domain) and a function $.^{\mathcal{I}}$ (the interpretation function) that maps every concept to a subset of $\Delta^{\mathcal{I}}$, every role to a relation over $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and every individual to an element of $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.

In the context of ML, "interpretability is the ability to explain or to present in understandable terms to a human".

By incorporating these insights, we propose the following definition of *feature interpretability.*

Definition 3.2. *(Feature Interpretability). It is the ability of domain experts to comprehend and connect the generated features to relevant concepts and entities within their domain knowledge. This implies mapping every new feature to relevant intensional and extensional knowledge within the domain of interest*

From the definition above, we define the following properties that an interpretable feature should meet.

1. **Readable:** A feature is readable when it is expressed in clear and human-friendly language that is easily understood by domain experts, avoiding obscure codes and acronyms.
2. **Meaningful:** Features must refer to real-world entities that experts can reason about, depending on their context and expertise, excluding overly complex engineered features that lack intuitive understanding.
3. **Traceable:** Features should have a clear connection to their original data sources, allowing users to track their lineage and ensuring transparency in feature creation.
4. **Useful:** Features must have a meaningful relationship with the target variable, contributing to the model's performance by reflecting genuine patterns rather than spurious correlations, thereby enhancing model generalization.

The quality of ML models strongly depends on the quality of input features. Even simple and interpretable models, like regression, become difficult to understand with non-interpretable features. When ML models are deployed in real-world applications, their predictions are communicated through the language of their features. Whether through feature importance, visualizations of decision boundaries, or examples that represent how decisions are made, features serve as the foundation for explaining the model's behavior to users. If the features are difficult to understand, any attempts to explain the model would fail.

3.2 Feature Engineering

Given a predictive problem on a tabular dataset $D = (X, y)$ consisting of: (i) a set of features $X = \{x_1, .., x_p\} \in \mathbb{R}^{n \times p}$, where n and p are the number of instances and features respectively; (ii) a target vector y; (iii) an applicable ML algorithm L (e.g. XGBoost); and (iv) a corresponding cross-validation performance metric $\mathcal{P}$ (e.g. F1-score). We define a FE pipeline $\mathcal{T} = \{t_1, ..., t_m\}$ as a sequence of m transformations applied to X. The set of generated features from X using $\mathcal{T}$ is denoted as $\hat{X}_{\mathcal{T}}$. Based on definition 3.2, we define $\mathcal{I}_{KG} : \mathcal{X} \rightarrow \{0, 1\}$ as an interpretability function that returns 1 if a feature is interpretable based on domain knowledge, and 0 otherwise.

The goal of AutoFE is to find the optimal FE pipeline $\mathcal{T}$ that generates $\hat{X}_{\mathcal{T}}$ which maximizes the performance $\mathcal{P}(L(\hat{X}_{\mathcal{T}}, y))$ for a given algorithm L and a metric $\mathcal{P}$, as shown in Equation 1:

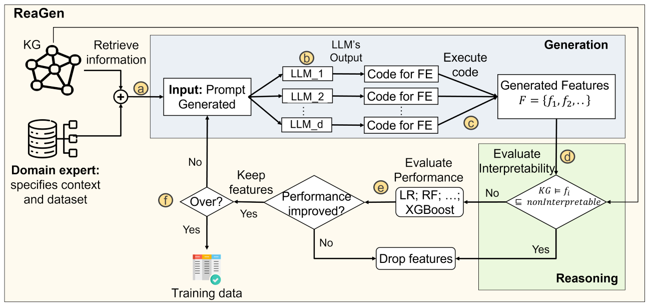


Figure 2: An overview of *ReaGen* architecture.

$$\mathcal{T} = \arg\max_{\mathcal{T}} \mathcal{P}(L(\hat{X}_{\mathcal{T}}, y)) \quad s.t. \quad \prod_{\hat{x_i} \in \hat{X}_{\mathcal{T}}} \mathcal{I}_{KG}(\hat{x_i}) = 1 \quad \mathcal{I}_{KG}(\hat{x_i}) \in \{0, 1\}, \forall \hat{x_i} \in \hat{X}_{\mathcal{T}} \tag{1}$$

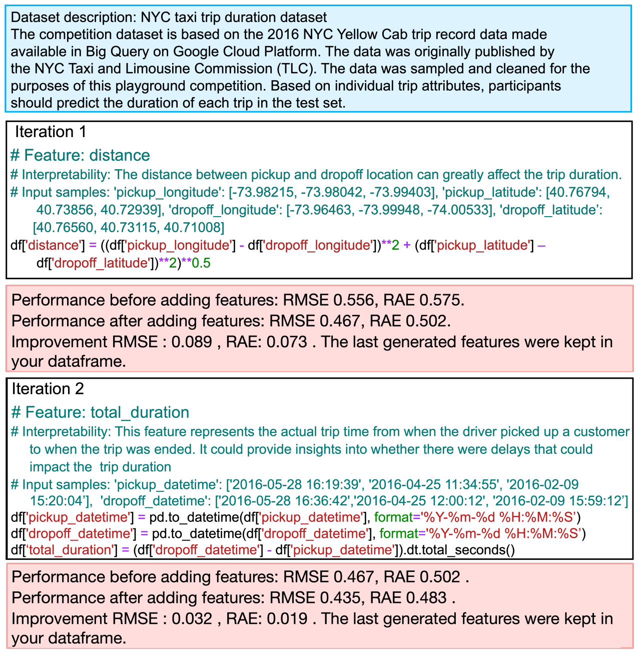
Note that *ReaGen* aims at generating features that maximize performance while ensuring that the generated features remain, as far as possible, close to the concepts known by domain experts.

4 Proposed approach

In this section, we present *ReaGen*, an AutoFE approach that combines the use of LLMs and knowledge-based reasoning techniques to engineer interpretable features, as outlined in Figure 2. *ReaGen* bridges the gap between ML and symbolic AI, offering a promising direction for interpretable AutoML tools.

4.1 Overview

ReaGen follows an iterative search process that takes a dataset, its description, and, if available, external knowledge sources as inputs. Initially, it leverages the external knowledge to extract additional information (see Figure 2.a). This knowledge can come in various forms, such as ontologies, KGs, and metadata. Using a predefined template, *ReaGen* generates a prompt that includes the dataset description, the FE task, and the information retrieved from the external knowledge (illustrated in Figure 2.b, with further details in Section 4.2). This generated prompt is then used to query multiple LLMs to identify the optimal set of transformations needed to create new features, as depicted in Figure 2.c (Section 4.3). After the new features are generated, their interpretability is assessed in relation to the concepts and entities within the domain knowledge embedded in the KG, using a logic-based reasoning algorithm. Features that lack relevance to the domain knowledge are discarded, ensuring that only relevant and interpretable features are retained, as shown in Figure 2.d (details in Section 4.4). Following the interpretability assessment, the resulting dataset is evaluated using a downstream model and a performance metric to gauge the effectiveness of the newly generated features for the given task (see Figure 2.e).


Dataset description: NYC taxi trip duration dataset
The competition dataset is based on the 2016 NYC Yellow Cab trip record data made available in Big Query on Google Cloud Platform. The data was originally published by the NYC Taxi and Limousine Commission (TLC). The data was sampled and cleaned for the purposes of this playground competition. Based on individual trip attributes, participants should predict the duration of each trip in the test set.

```
Iteration 1
# Feature: distance
# Interpretability: The distance between pickup and dropoff location can greatly affect the trip duration.
# Input samples: 'pickup_longitude': [-73.98215, -73.98042, -73.99403], 'pickup_latitude': [40.76794,
    40.73856, 40.72939], 'dropoff_longitude': [-73.96463, -73.99948, -74.00533], 'dropoff_latitude':
   [40.76560, 40.73115, 40.71008]
df['distance'] = ((df['pickup_longitude'] - df['dropoff_longitude'])**2 + (df['pickup_latitude'] –
    df['dropoff_latitude'])**2)**0.5
```

Performance before adding features: RMSE 0.556, RAE 0.575.
Performance after adding features: RMSE 0.467, RAE 0.502.
Improvement RMSE : 0.089 , RAE: 0.073 . The last generated features were kept in your dataframe.

```
Iteration 2
# Feature: total_duration
# Interpretability: This feature represents the actual trip time from when the driver picked up a customer
   to when the trip was ended. It could provide insights into whether there were delays that could
   impact the  trip duration
# Input samples: 'pickup_datetime': ['2016-05-28 16:19:39', '2016-04-25 11:34:55', '2016-02-09
   15:20:04'],  'dropoff_datetime': ['2016-05-28 16:36:42','2016-04-25 12:00:12', '2016-02-09 15:59:12']
df['pickup_datetime'] = pd.to_datetime(df['pickup_datetime'], format='%Y-%m-%d %H:%M:%S')
df['dropoff_datetime'] = pd.to_datetime(df['dropoff_datetime'], format='%Y-%m-%d %H:%M:%S')
df['total_duration'] = (df['dropoff_datetime'] - df['pickup_datetime']).dt.total_seconds()
```

Performance before adding features: RMSE 0.467, RAE 0.502 .
Performance after adding features: RMSE 0.435, RAE 0.483 .
Improvement RMSE : 0.032 , RAE: 0.019 . The last generated features were kept in your dataframe.

Figure 3: Application of *ReaGen* on the NYC taxi trip duration dataset. The dataset description provided is highlighted in blue, and the model performance evaluation is shown in red. The LLM-generated Python code follows a template from the prompt, resulting in an RMSE improvement of 0.121 over two iterations on the validation set.

This iterative process continues until a predefined budget is reached (e.g., number of iterations), as shown in Figure 2.f. Consequently, *ReaGen* helps improve the quality of the generated features and facilitates the automatic construction of interpretable and high-performing models. Figure 3 shows a simplified version of one such run on the *NYC taxi trip duration* dataset.

4.2 Prompting LLMs for AutoFE

To leverage LLMs for AutoFE, *ReaGen* uses a template to create a prompt that guides the generation of interpretable features. Template-based prompting is a common technique to improve the quality of LLM responses [33]. Essentially, the LLMs are instructed to propose meaningful features for a prediction task, justifying their usefulness, and to drop unnecessary features. Additionally, they are tasked to provide Python code to automatically generate and drop these features. This prompt incorporates the following elements.

4.2.1 Dataset Description. Dataset Information: This includes a description of the dataset and the target variable to be predicted, along with feature names and their context, data types (e.g., float, int, categorical), and, where applicable, the domains of categorical features. For open-source datasets from platforms like Kaggle [3] and OpenML[4], much of this information can often be extracted automatically. Our model enhances this description by providing a

[3]https://www.kaggle.com/
[4]https://www.openml.org/

summary of descriptive statistics, such as the percentage of missing values, minimum and maximum values, and the count of unique values, along with 10 random records from the dataset. This contextual information offers insights into the dataset's scale and feature encoding, enabling LLMs to generate semantically meaningful features and process them effectively.

4.2.2 *External Knowledge.*

Users can provide external knowledge bases, such as libraries or publicly available ontologies, to enrich the dataset with additional information. For example, these resources can be used to derive relevant context, such as population density from a city feature or weather data based on a city and a date. This additional information adds context and enriches the prompt, allowing for the generation of context-aware features.

4.2.3 *Feature Generation.*

LLMs are prompted to generate new features by applying a variety of transformations. We categorize these transformations into three types: (i) Arithmetic functions; which can be unary or binary functions, (ii) Aggregation functions (e.g., GroupByThenAvg), which are applied based on the feature type (numerical or categorical), (iii) Customized functions, which generate new features that cannot be derived from the previous transformations, such as extracting Day, Month, Year, Is_Weekend, or Is_Rush_Hour from a date feature. More details on these transformations are provided in Appendix A.

We used two different prompt strategies, depending on the type of transformations: *proposal strategy* and *sampling strategy* [38, 39]. The former is used for unary functions, where LLMs are prompted to suggest all relevant transformations for a given feature along with confidence levels (high, medium, low). *ReaGen* then selects the transformations with high confidence. This approach is most effective when the search space is small. For the *sampling strategy*, LLMs are prompted to provide candidate features for a specific transformation. This is particularly useful when the number of possible operands for a transformation is large. For example, a *GroupByThenAgg* transformation requires both a grouping feature and a context feature (an aggregation column) along with an aggregation function. In datasets with many categorical features, even one grouping column can generate various feature candidates. In such cases, LLMs are tasked with identifying the most relevant aggregation columns and functions for the selected groupby column.

To maintain readability and traceability, LLMs are prompted to name the generated features in the format *Function_Name(operands)*, where *Function_Name* is the name of the proposed transformation (e.g., *GroupByThenAvg*), and *operands* is the list of features involved. For example, applying *GroupByThenAvg* on the features *Revenue* and *sex* would result in a new feature named *GroupByThenAvg(Revenue, sex)*, representing the average revenue grouped by sex.

4.2.4 *Expected Output.*

A template for the desired output is provided using Chain-of-Thought (CoT) prompting, as described in [37], which includes intermediate reasoning steps. The output consists of several key components: the proposed feature's name, a description of the feature, an explanation of its relevance to the task, the names and examples of the features used, ensuring traceability, and the Python code to generate or remove the feature.

4.2.5 *Error Handling.*

LLMs are also used to handle exceptions. If the generated code produces an error, the error is passed back to the LLMs in the next iteration, instructing them to correct it.

The datasets used in our experiments are described in Appendix E, and the fully-formed prompt is shown in Figure 9 in Appendix D. We found that this prompt design enables *ReaGen* to generate interpretable and meaningful features.

4.3 Feature Generator

The prompt described above is used by multiple LLMs to propose transformations for creating new features (Figure 2.c). Each LLM i produces Python code to generate features for the current dataset D. To mitigate risks associated with the automatic execution of AI-generated code, such as malicious code, we established an allowlist of permitted transformations. The generated code is parsed and checked against this allowlist. If a transformation is not allowed, *ReaGen* raises an exception and passes it to the next LLM $i + 1$. Similarly, if the proposed operands are invalid or the code generates an error, the issue is passed to another LLM for resolution.

Once validated, the code from each LLM is executed on a copy of dataset D, resulting in a new dataset D^i. These datasets are then combined using a union operation, creating an updated dataset D'. Additionally, the LLMs provide explanations for each proposed feature, using both internal knowledge and retrieved external information to justify its utility. In cases where the proposed transformations cannot generate suitable features due to missing data, the LLMs are tasked with suggesting potential data sources to assist users in generating the necessary features.

4.4 Knowledge-based discriminator

Traditional approaches often combine transformations to generate new features, testing their effectiveness without considering the interpretability of the results. In contrast, *ReaGen* emphasizes interpretability by integrating domain knowledge into the feature generation process. Initially, we attempted to use LLMs to generate interpretable features with high predictive power, without additional interpretability assessments. However, we discovered that LLM-generated features were not consistently interpretable by our standards. While LLMs can suggest features that are related to domain knowledge, they often produce arbitrary combinations of transformations that lack a clear semantic connection to the domain. For example, LLMs may suggest adding or subtracting raw features with different units of measurement or representing different quantities, which domain experts find unreasonsable. Even when such features improve model performance, they fail to meet the interpretability criteria necessary for experts to trust and understand the outputs. Moreover, we observed that LLMs sometimes handle specific types of features, like dates, in unintelligent ways. For instance, in a dataset of monthly warehouse inventory (stocks), LLMs incorrectly aggregated stock values over a quarter by summing or averaging them, rather than selecting the "last value," which is the appropriate aggregation. Similarly, LLMs struggle with dynamic units like prices, often incorrectly suggesting that different currencies (e.g., euros and dollars) be summed, which lacks contextual meaning. When features involve multiple layers of aggregation (e.g., over time and location), LLMs frequently fail to respect the correct order. Through collaboration with domain experts, we identified several such inconsistencies in LLM-generated features.

To address these shortcomings, we introduced an additional interpretability check for features proposed by LLMs. To reduce factual inaccuracies typical in LLMs [27, 42], we designed a knowledge-based discriminator to assess the semantic distance between proposed features and domain concepts embedded in KGs or knowledge bases. We began by constructing a KG[5] using a description logics-based language. The KG leverages two types of knowledge: (i) Domain-agnostic knowledge which includes knowledge about *units of measurement* and the class *non-interpretable*; (ii) Domain-specific knowledge using several publicly available Knowledge sources from various domains, further details in Appendix B. Then, we applied a reasoning algorithm to filter out non-interpretable features. Using the Hermit reasoner [29], we checked if a feature $x' \in D'$ could be subsumed under the concept of *non-interpretable* (e.i., $KG \models x' \sqsubset$ *non-interpretable*). Features identified as non-interpretable were removed (Figure 2.d). If a feature lacked sufficient information, its unit was used as a secondary discriminator ($KG \models Units(u)$), and features with unknown units were discarded. This process ensures that the generated features and transformations are aligned with domain knowledge and interpretable to domain experts. As a result, *ReaGen* not only enhances trust in the model's outputs but also reduces bias.

4.5 Performance Evaluation

Post-interpretability assessment, the current dataset D and its extension D' are split into training and validation sets, respectively designated as D_{train}, D_{valid}, D'_{train} and D'_{valid}. An ML algorithm, L, is then trained on D'_{train} and evaluated on D'_{valid} to obtain its performance $\mathcal{P}'$. If $\mathcal{P}'$ exceeds the performance $\mathcal{P}$ obtained in the previous iteration using D_{train} and D_{valid}, the new features are retained and D_{train} and D_{valid} are updated to D'_{train} and D'_{valid}. Otherwise, the new features are rejected and D_{train} and D_{valid} remain unchanged as shown in Figure 2.e.

Given that evaluation relies heavily on cross-validation results from downstream tasks, the focus on performance metrics can substantially increase the time required for validating proposed transformations. By leveraging our discriminator to pre-filter non-interpretable features, we address this challenge. These features are discarded early, removing the need for unnecessary model evaluations. This reduces wasted computation, optimizing time complexity and boosting the overall efficiency of the process.

5 Experiments

5.1 Experimental setup

5.1.1 Datasets. We use 30 public datasets collected from the Internet, including 18 academic datasets and 10 large-scale industrial datasets, all available on Kaggle, OpenML, and UCIrvine[6]. Since LLMs are trained on public data, including potentially these datasets if released before their knowledge cutoff, evaluating on these datasets could introduce bias. To address this, we used a mix of datasets: those released before September 2021, potentially known to GPT-3.5-turbo and GPT-4, and newer datasets from Kaggle competitions released after September 2021, which are less likely to have been accessed by these models, since they are only accessible through specific agreements.

5.1.2 AutoFE Methods. We compared *ReaGen* with several methods. (1) Base: The original dataset without FE; (2) Random: Apply random transformations on row features; (3) DFS [10] for which we set its $max\_depth = 3$; (4) AutoFeat [8]: a python library for AutoFE; (5) mCAFE [9]: uses Monte Carlo Search; (6) NFS [2]:uses neural search for FE; (7) DIFER [44]: an evolutionary framework; and (8) OpenFE [41]. Additionally, we campared *ReaGen* with two AutoML methods. (9) AutoSklearn [6]: A popular AutoML framework based on Bayesian optimization; and (10) AutoGluon [5]: A famous AutoML framework developed by Amazon Inc. All methods were used with default settings and limited to 50 new features.

5.1.3 Evaluation metrics. We used the metrics 1 - rae (relative absolute error) [28] and F1-score, for regression and classification tasks respectively. Both evaluation metrics are that the higher the score, the better the performance. 5-fold cross-validation protocol is used to evaluate the effectiveness of the generated features.

5.1.4 Evaluation setup. The datasets are loaded into memory as Pandas dataframes [21]. We use OpenAI's GPT-3.5-turbo and GPT-4 [23] as LLMs in our architecture to generate Python code, that is automatically executed. To avoid information leakage and overfitting, we combined the three-way split and cross validation techniques with most datasets. First, the dataset is separated into three distinct parts: training, validation, and testing sets. Then we apply cross-validation on the training set. Performance is measured on the current dataset with 5 random validation splits using Random Forest [34], while the testing sets remain unseen until the final experimental evaluation. Features are retained if their average improvement in evaluation metrics over 5 splits is positive. The selected features are then evaluated using 5 commonly used ML models: Random Forest (RF), Decision Tree (DT), Logistic/Linear Regression (LR), Support Vector Machines (SVM), and XGBoost (XGB). All parameters are set to default values in scikit-learn [24].

5.2 Effectiveness of *ReaGen*

We report in Table 1 the evaluation results on 18 datasets (due to lack of space) compared to AutoFE methods. *ReaGen* outperforms the methods in most cases (16/18 datasets). Compared to raw data, the features generated by *ReaGen* can improve the performance by an average of 19.73%. Compared with *OpenFE*, *NFS* and *mCAFE*, *ReaGen* achieves an average improvement of 3.83%, 5.36% and 6.84% respectively.

The superior performance of *ReaGen* can be attributed to several key factors: (i) Leveraging External Knowledge: Our model uses domain knowledge to generate features grounded in domain understanding and that better explain the target variable. This external knowledge enriches the dataset with contextual features and metadata, which goes beyond the capabilities of baselines that primarily rely on raw data transformations. (ii) Leveraging LLMs: By utilizing the capabilities of LLMs, our approach generates semantically meaningful features based on dataset descriptions. This aligns with the core concept of feature engineering, which involves using domain

[5]https://cutt.ly/bex8C2Cc
[6]https://archive.ics.uci.edu/

Table 1: Comparing the effectiveness of *ReaGen* with the baselines. Inst. is short for Intstances, Feat. is short for Features.

Datasets	Task	#Inst./#Feat	Base	*ReaGen*	Expansion-Reduction				Learning-based Methods			AutoML Approaches	
					Random	DFS	AutoFeat	OpenFE	NFS	mCAFE	DIFER	AutoSklearn	AutoGluon
Fertility	C	100 / 9	0.870	**0.932**	0.795	0.790	0.890	0.920	0.916	0.856	0.880	0.840	0.880
Hipatitis	C	155 / 19	0.832	**0.972**	0.845	0.850	0.868	0.932	0.870	0.877	0.883	0.810	0.780
Megawatt1	C	253 /37	0.870	**0.928**	0.815	0.890	0.889	0.902	0.920	0.896	0.910	0.885	0.885
Credit-a	C	690 / 15	0.840	**0.900**	0.705	0.821	0.859	0.867	0.866	0.846	0.863	0.864	0.842
Diabetes	C	768 / 8	0.740	**0.853**	0.670	0.737	0.767	0.842	0.786	0.813	0.798	0.801	0.788
Wine Quality Red	C	999 / 12	0.531	**0.707**	0.506	0.547	0.524	0.594	0.584	0.674	0.582	0.580	0.572
German	C	1001 / 24	0.742	**0.827**	0.655	0.780	0.796	0.813	0.805	0.795	0.777	0.746	0.750
SVMGuide3	C	1243 / 21	0.740	**0.859**	0.721	0.711	0.789	**0.858**	0.856	0.829	0.834	0.807	0.798
Spam Base	C	4601 / 57	0.939	**0.956**	0.939	0.919	0.943	0.952	0.939	0.937	0.942	0.925	0.900
Wine Quality White	C	4900 / 12	0.494	**0.607**	0.504	0.488	0.502	0.569	0.516	0.502	0.515	0.537	0.525
Home Credit Default Risk	C	30000 / 25	0.797	**0.837**	0.789	0.802	0.806	0.810	0.799	0.801	0.810	0.820	0.821
Amazon Employee	C	32769 / 9	0.712	0.932	0.740	0.744	0.739	0.909	**0.945**	0.897	0.909	0.947	0.949
Higgs Boson	C	50000 / 28	0.718	**0.739**	0.699	0.682	0.468	0.730	0.731	**0.739**	**0.738**	0.716	0.709
Openml_637	R	500 / 50	0.516	**0.700**	0.511	0.510	0.576	0.680	0.537	0.543	0.600	0.641	0.674
Openml_620	R	1000 / 25	0.630	0.748	0.608	0.652	0.657	0.669	0.694	0.643	0.726	0.720	**0.776**
Openml_618	R	1000 / 50	0.428	**0.740**	0.411	0.411	0.632	0.732	0.640	0.738	0.660	0.720	**0.740**
Airfoil	R	1503 / 5	0.753	**0.808**	0.752	0.771	0.595	0.789	0.696	0.616	0.624	0.516	0.510
Bikeshare DC	R	10886 / 11	0.393	**0.988**	0.381	0.693	0.849	0.981	0.974	0.906	0.981	0.981	0.967

Table 2: Comparison of *ReaGen* with learning-based approaches on large-scale datasets. *T* is the runtime in minutes.

Dataset	Task	#Inst./#Feat	Base	NFS	DIFER	ReaGen	T_{NFS}	T_{DIFER}	T_{ReaGen}
AP Ovary	C	275/10936	0.829	0.836	0.850	**0.859**	100	112	**21**
Gisette	C	6000/5000	0.946	0.950	0.954	**0.956**	203	153	**25**
Medical Appointment	C	110527/13	0.492	0.650	0.799	**0.855**	503	120	**24**
Accelerometer	C	153000/5	0.702	**0.705**	**0.704**	**0.704**	567	491	**24**
Medical Charges	C	163065/11	0.883	0.893	0.897	**0.899**	501	597	**35**
Covtype	C	581012/55	0.943	0.967	0.964	**0.973**	> 3000	2756	**55**
Poker Hand	C	1025010/11	0.711	0.890	**0.940**	0.928	> 3000	> 3000	**72**

Table 3: Evaluating *ReaGen* in Kaggle competitions.

Competition	#Inst./#Feat	Metric	NFS	DIFER	*ReaGen*
Restaurant Revenue	137/42	RMSE ↓	0.347	0.398	**0.300**
House Prices	1461/80	RMSE↓	0.141	0.135	**0.130**
Tabular Playground Series	250000/102	RMSE↓	7.90	7.92	**0.788**
NYC Taxi Ride Duration	2702376	AUC↑	0.551	0.563	**0.581**

knowledge to create features that enhance model performance. (iii) High-Order Feature Generation: *ReaGen*'s ability to generate high-order features plays a crucial role in its performance. In fact, as the maximum order of features increases, the accuracy of *ReaGen* improves (as shown in Appendix C.2). This indicates the effectiveness of generating high-order features, as the composition of transformations is crucial for discovering complex relationships between features that simpler methods may miss. (vi) Diverse Transformations: Unlike most baselines that rely on basic arithmetic functions, *ReaGen* utilizes a wider range of transformations, including aggregations and logical operators on binary features. This diversity allows the model to generate more promising features, ultimately leading to better performance. Our experiments highlighted that aggregation functions, in particular, were effective in discovering patterns over time periods and space vectors, giving *ReaGen* an advantage.

In addition, to evaluate the robustness of our model, we used a range of classifiers, including XGBoost, SVM, and Decision Tree. The results, shown in Table 8 in Appendix C.1, indicate that features generated by *ReaGen* consistently outperform those from other techniques across different ML models, showing a superior robustness.

5.3 Efficiency and Scalability

5.3.1 Efficiency. The execution time of *ReaGen* depends on the dataset size, prediction task, and ML model used for evaluation. For a 5-fold cross-validation, using Random Forest with 10 iterations, the running time varies from 6 minutes to 70 minutes depending on the dataset. For smaller datasets like *German credit* dataset, it required around 5:43 minutes, while larger datasets like the *Poker Hand* dataset took approximately 70 minutes. On average, 65% of the time is spent on code generation, over 30% on interpretability evaluation, and less than 5% on evaluating the generated features. We compare the runtime of different methods on a benchmark datasets, and show the results in Table 2. One can see that *ReaGen* is significantly more efficient than baselines. *ReaGen* only runs a few minutes to generate features on most medium-scale datasets.

5.3.2 Scalability. To demonstrate *ReaGen*'s performance on large-scale industrial datasets, we conducted comparative experiments using 7 such datasets. The results, shown in Table 2, reveal that *ReaGen* required minimal computation time across all datasets and still managed to achieve significant performance improvements.

5.3.3 Generalization. : One of the challenges in FE is the risk of overfitting, which can lead to a significant drop in performance on unseen test data. To address this, we used k-fold cross-validation to improve generalization ability. To further validate *ReaGen*'s generalizability, we conducted an additional experiment by participating in several Kaggle competitions. These competitions provide training datasets and unlabeled test sets, with final evaluations performed by Kaggle itself to prevent artificial predictions. The results, presented in Table 3, demonstrate *ReaGen*'s strong generalization capabilities.

Table 4: Illustrations of features generated by *ReaGen*. Using dataset descriptions, *ReaGen* automatically generates meaningful features through feature combinations, binning, removing unnecessary and non-interpretable features.

Description	Generated code

Retrieved information (NYC taxi trip duration dataset)	# Feature: precipitation_average
	# Interpretability: The amount of precipitation on the day of pickup could influence the trip duration due to varying road conditions and traffic flow. When it rains the roads can become slick and visibility can be reduced, which can lead to slower speeds and longer travel times.
	# Input samples: 'pickup_datetime': ['2016-05-28', '2016-04-25', '2016-02-09'], 'precipitation': ['0.04', '0.00', '0.10']
	df = df.assign(pickup_date=df['pickup_datetime'].dt.date).merge(weather[['date', 'precipitation']], left_on='pickup_date', right_on='date', how='left')
	weather_avg = weather.groupby('date')['precipitation'].mean().reset_index().rename (columns={'precipitation':'precipitation_avg'})
	df = df.merge(weather_avg, left_on='pickup_date', right_on='date', how='left')

Binning (NYC taxi trip duration dataset)	# Feature: Pickup time_of_day (Morning/Afternoon/Evening/Night) # Interpretability: Time of Day split into 4 broad timeslots can be useful in determining trip durations due to rush hours. # Input samples: 'pickup_datetime': ['2016-01-01 06:02:06', '2016-01-02 14:30:45', '2016-01-03 21:52:58'] df['time_of_day'] = pd.cut(df['pickup_datetime'].dt.hour, bins=[0,6,12,18,24], labels=['Night','Morning','Afternoon','Evening'])
Meaning -ful features (Medical appointment dataset)	# Feature: days_scheduled _appointment # Interpretability: This feature calculates the number of days between the scheduled day and the appointment day. This could be useful as patients who schedule their appointments far in advance may be more likely to forget or not show up. # Input samples: 'ScheduledDay': ['2016-05-12T07:47:18Z', '2016-04-28T07:17:50Z', '2016-04-11T07:56:14Z'], 'AppointmentDay': ['2016-05-12T00:00:00Z', '2016-05-03T00:00:00Z', '2016-04-29T00:00:00Z'] df['ScheduledDay'] = pd.to_datetime(df['ScheduledDay']).dt.date df['AppointmentDay'] = pd.to_datetime(df['AppointmentDay']).dt.date df['days_scheduled_appointment'] = (df['AppointmentDay'] - df['ScheduledDay']).dt.days

Remove features (German credit dataset)	# Feature Selection: Drop 'foreign_worker'
	# Explanation: The feature is dropped because it has a very low mean (0.035), indicating that the vast majority of samples are not foreign workers. Therefore, this feature is unlikely to provide much useful information for the classification task.
	df.drop(columns=['foreign_worker'], inplace=True)

Drop non-interpret -able features (Kidney stone urine dataset)	# Feature: gravity_osmo_difference
	# Interpretability: This feature represents the difference between the specific gravity and osmolarity of the urine. It could be useful in identifying the balance between these two factors, which might be related to the formation of kidney stones.
	# Input samples: 'gravity': [1.011, 1.014, 1.029], 'osmo': [200.0, 300.0, 400.0]
	df['gravity_osmo_differance'] = df['gravity'] - df['osmo']
	
	# The feature was dropped from the dataframe
	# Explanation: It is subsumed from class 'non_interpretable'. We cannot subtract 'specific gravity' and 'osmolarity' because they are different types of measurements with different units.
	df.drop(columns=['gravity_osmo_differance '], inplace=True)

5.4 Feature Importance

Our work aims at generating useful and interpretable features for domain experts. In this experiment, we use SHAP (SHapley Additive exPlanations) [19] to compare the importance of *ReaGen*'s generated features with raw features. We start by creating a new dataset combining the *n* raw features with the top-ranked *n* features generated by *ReaGen*. We then use SHAP to score feature importance in the model's predictions. Figure 4 showcases the results for 4 different datasets. Notably, the features generated by *ReaGen* (shown in blue) are more important compared to raw features (shown in orange) across all datasets, thereby validating the effectiveness of our model in generating meaningful features.

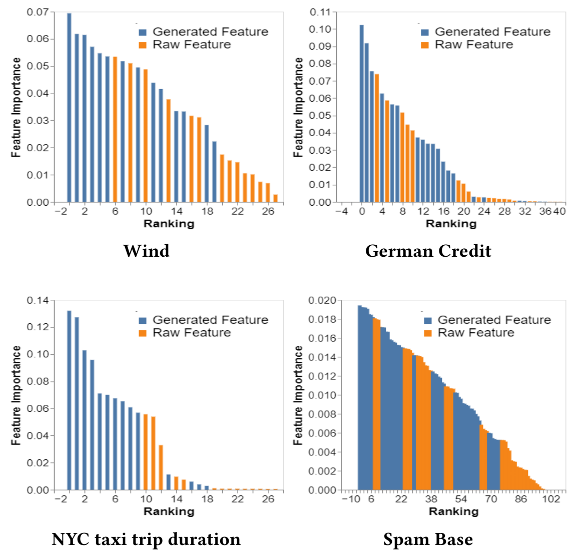


Figure 4: Comparing feature importance of raw features (in orange) and generated features with *ReaGen* (in blue).

Furthermore, in Table 4, we illustrate examples of *ReaGen*'s generated features. These features are readable, easily traceable to raw data, and semantically relevant, allowing domain experts to align them with their knowledge for easier interpretation. For instance, the feature *distance* in Figure 3 is clear and interpretable, allowing experts to understand what is being referred to and which raw features were used. This feature is meaningful due to its direct correlation with the target variable, given the evident relationship between pickup-to-dropoff distance and taxi ride duration. In addition, the logical reasoner maintains interpretability by excluding features that violate rules in the KG. For example, the feature *gravity_osmo_difference* in Table 4 was discarded because subtracting features with different units results in a non-interpretable feature. These explanations provide transparency. Further experiments on feature interpretability are presented in Appendix C.3.

6 Conclusion

Our work presents a novel approach that combines the use of knowledge-based reasoning with large language models to automate feature engineering for structured data. By incorporating external knowledge, *ReaGen* enhances the factual accuracy and grounding on LLMs, thereby reducing the occurrence of hallucinated thoughts. Our experiments demonstrate the effectiveness of our approach and the interpretability of the generated features. As generative models continue to evolve, we expect that the performance of *ReaGen* will further improve. In this work, we bridge the gap between ML and symbolic AI, highlighting the importance of incorporating domain knowledge and context-aware solutions in AutoML tools. Specifically, we showcase the use of LLMs to automate FE. We also foresee the potential of LLMs to automate other steps in the data science workflow, such as model selection for tabular data.

References

[1] Ehtesamul Azim, Dongjie Wang, Kunpeng Liu, Wei Zhang, and Yanjie Fu. 2024. Feature Interaction Aware Automated Data Representation Transformation. In *Proceedings of the 2024 SIAM International Conference on Data Mining, SDM 2024, Houston, TX, USA, April 18-20, 2024*, Shashi Shekhar, Vagelis Papalexakis, Jing Gao, Zhe Jiang, and Matteo Riondato (Eds.). SIAM, 878–886. https://doi.org/10.1137/1.9781611978032.100

[2] Xiangning Chen, Qingwei Lin, Chuan Luo, Xudong Li, Hongyu Zhang, Yong Xu, Yingnong Dang, Kaixin Sui, Xu Zhang, Bo Qiao, et al. 2019. Neural feature search: A neural architecture for automated feature engineering. In *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 71–80.

[3] Minje Choi, Luca Maria Aiello, Krisztián Zsolt Varga, and Daniele Quercia. 2020. Ten Social Dimensions of Conversations and Relationships. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, Yennun Huang, Irwin King, Tie-Yan Liu, and Maarten van Steen (Eds.). ACM / IW3C2, 1514–1525. https://doi.org/10.1145/3366423.3380224

[4] Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. 2023. Chain-of-Verification Reduces Hallucination in Large Language Models. *CoRR* abs/2309.11495 (2023). https://doi.org/10.48550/ARXIV.2309.11495 arXiv:2309.11495

[5] 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). arXiv:2003.06505 https://arxiv.org/abs/2003.06505

[6] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. 2019. Auto-sklearn: Efficient and Robust Automated Machine Learning. In *Automated Machine Learning - Methods, Systems, Challenges*, Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (Eds.). Springer, 113–134. https://doi.org/10.1007/978-3-030-05318-5_6

[7] Tu Gu, Kaiyu Feng, Gao Cong, Cheng Long, Zheng Wang, and Sheng Wang. 2023. The RLR-Tree: A Reinforcement Learning Based R-Tree for Spatial Data. *Proc. ACM Manag. Data* 1, 1 (2023), 63:1–63:26.

[8] Franziska Horn, Robert Pack, and Michael Rieger. 2019. The autofeat python library for automated feature engineering and selection. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 111–120.

[9] Yiran Huang, Yexu Zhou, Michael Hefenbrock, Till Riedel, Likun Fang, and Michael Beigl. 2022. Automatic Feature Engineering Through Monte Carlo Tree Search. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 581–598.

[10] James Max Kanter and Kalyan Veeramachaneni. 2015. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE international conference on data science and advanced analytics (DSAA)*. IEEE, 1–10.

[11] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. 2016. Explorekit: Automatic feature generation and selection. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 979–984.

[12] Udayan Khurana, Horst Samulowitz, and Deepak Turaga. 2018. Feature engineering for predictive modeling using reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[13] Udayan Khurana, Deepak Turaga, Horst Samulowitz, and Srinivasan Parthasrathy. 2016. Cognito: Automated feature engineering for supervised learning. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*. IEEE, 1304–1307.

[14] Hoang Thanh Lam, Beat Buesser, Hong Min, Tran Ngoc Minh, Martin Wistuba, Udayan Khurana, Gregory Bramble, Theodoros Salonidis, Dakuo Wang, and Horst Samulowitz. 2021. Automated Data Science for Relational Data. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. 2689–2692. https://doi.org/10.1109/ICDE51399.2021.00305

[15] Hoang Thanh Lam, Johann-Michael Thiebaut, Mathieu Sinn, Bei Chen, Tiep Mai, and Oznur Alkan. 2017. One button machine for automating feature engineering in relational databases. *arXiv preprint arXiv:1706.00327* (2017).

[16] Liyao Li, Haobo Wang, Liangyu Zha, Qingyi Huang, Sai Wu, Gang Chen, and Junbo Zhao. 2023. Learning a Data-Driven Policy Network for Pre-Training Automated Feature Engineering. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/forum?id=688hNNMigVX

[17] Yiming Li, Yanyan Shen, and Lei Chen. 2022. Camel: Managing Data for Efficient Stream Learning. In *Proceedings of the 2022 International Conference on Management of Data*. 1271–1285.

[18] Tania Lombrozo. 2006. The structure and function of explanations. *Trends in cognitive sciences* 10, 10 (2006), 464–470.

[19] Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. *Advances in neural information processing systems* 30 (2017).

[20] Shaul Markovitch and Dan Rosenstein. 2002. Feature Generation Using General Constructor Functions. *Mach. Learn.* 49, 1 (2002), 59–98. https://doi.org/10.1023/A:1014046307775

[21] Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference 2010 (SciPy 2010), Austin, Texas, June 28 - July 3, 2010*, Stéfan van der Walt and Jarrod Millman (Eds.). scipy.org, 56–61. https://doi.org/10.25080/MAJORA-92BF1922-00A

[22] Fatemeh Nargesian, Horst Samulowitz, Udayan Khurana, Elias B Khalil, and Deepak S Turaga. 2017. Learning Feature Engineering for Classification.. In *Ijcai*. 2529–2535.

[23] OpenAI. 2023. GPT-4 Technical Report. *CoRR* abs/2303.08774 (2023). https://doi.org/10.48550/ARXIV.2303.08774 arXiv:2303.08774

[24] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830.

[25] Selwyn Piramuthu and Riyaz T. Sikora. 2009. Iterative feature construction for improving inductive learning algorithms. *Expert Syst. Appl.* 36, 2 (2009), 3401–3406. https://doi.org/10.1016/J.ESWA.2008.02.010

[26] Vinay Uday Prabhu and Abeba Birhane. 2020. Large image datasets: A pyrrhic win for computer vision? *CoRR* abs/2006.16923 (2020). arXiv:2006.16923 https://arxiv.org/abs/2006.16923

[27] Vipula Rawte, Amit P. Sheth, and Amitava Das. 2023. A Survey of Hallucination in Large Foundation Models. *CoRR* abs/2309.05922 (2023). https://doi.org/10.48550/ARXIV.2309.05922 arXiv:2309.05922

[28] Maxim Vladimirovich Shcherbakov, Adriaan Brebels, Nataliya Lvovna Shcherbakova, Anton Pavlovich Tyukov, Timur Alexandrovich Janovsky, Valeriy Anatol'evich Kamaev, et al. 2013. A survey of forecast error measures. *World applied sciences journal* 24, 24 (2013), 171–176.

[29] Robert DC Shearer, Boris Motik, and Ian Horrocks. 2008. Hermit: A highly-efficient OWL reasoner.. In *Owled*, Vol. 432. 91.

[30] Ying Sheng, Sandeep Tata, James B. Wendt, Jing Xie, Qi Zhao, and Marc Najork. 2018. Anatomy of a Privacy-Safe Large-Scale Information Extraction System Over Email. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, Yike Guo and Faisal Farooq (Eds.). ACM, 734–743. https://doi.org/10.1145/3219819.3219901

[31] Qitao Shi, Ya-Lin Zhang, Longfei Li, Xinxing Yang, Meng Li, and Jun Zhou. 2020. Safe: Scalable automatic feature engineering framework for industrial tasks. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1645–1656.

[32] Dakuo Wang, Josh Andres, Justin D. Weisz, Erick Oduor, and Casey Dugan. 2021. AutoDS: Towards Human-Centered Automation of Data Science. In *CHI '21: CHI Conference on Human Factors in Computing Systems, Virtual Event / Yokohama, Japan, May 8-13, 2021*, Yoshifumi Kitamura, Aaron Quigley, Katherine Isbister, Takeo Igarashi, Pernille Bjørn, and Steven Mark Drucker (Eds.). ACM, 79:1–79:12. https://doi.org/10.1145/3411764.3445526

[33] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/pdf?id=1PL1NIMMrw

[34] Yan Wang, Huaiqing Wu, and Dan Nettleton. 2023. Stability of Random Forests and Coverage of Random-Forest Prediction Intervals. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/6452474601429509f3035dc81c233226-Abstract-Conference.html

[35] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8 (1992), 279–292.

[36] Hilde J. P. Weerts, Florian Pfisterer, Matthias Feurer, Katharina Eggensperger, Edward Bergman, Noor H. Awad, Joaquin Vanschoren, Mykola Pechenizkiy, Bernd Bischl, and Frank Hutter. 2024. Can Fairness be Automated? Guidelines and Opportunities for Fairness-aware AutoML. *J. Artif. Intell. Res.* 79 (2024), 639–677. https://doi.org/10.1613/JAIR.1.14747

[37] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[38] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html

[39] Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, and Guoyin Wang. 2023. Instruction Tuning for Large Language Models: A Survey. *CoRR* abs/2308.10792 (2023).

https://doi.org/10.48550/ARXIV.2308.10792 arXiv:2308.10792

[40] Tianping Zhang, Zheyu Aqa Zhang, Zhiyuan Fan, Haoyan Luo, Fengyuan Liu, Qian Liu, Wei Cao, and Li Jian. 2023. OpenFE: Automated Feature Generation with Expert-level Performance. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA (Proceedings of Machine Learning Research, Vol. 202)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 41880–41901. https://proceedings.mlr.press/v202/zhang23ay.html

[41] Tianping Zhang, Zheyu Aqa Zhang, Zhiyuan Fan, Haoyan Luo, Fengyuan Liu, Qian Liu, Wei Cao, and Li Jian. 2023. OpenFE: Automated Feature Generation with Expert-level Performance. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA (Proceedings of Machine Learning Research, Vol. 202)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 41880–41901. https://proceedings.mlr.press/v202/zhang23ay.html

[42] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, Longyue Wang, Anh Tuan Luu, Wei Bi, Freda Shi, and Shuming Shi. 2023. Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models. *CoRR* abs/2309.01219 (2023). https://doi.org/10.48550/ARXIV.2309.01219 arXiv:2309.01219

[43] Yuhao Zhang, Frank Mcquillan, Nandish Jayaram, Nikhil Kak, Ekta Khanna, Orhan Kislal, Domino Valdano, and Arun Kumar. 2021. Distributed Deep Learning on Data Systems: A Comparative Analysis of Approaches. *Proc. VLDB Endow.* 14, 10 (2021), 1769–1782.

[44] Guanghui Zhu, Zhuoer Xu, Chunfeng Yuan, and Yihua Huang. 2022. DIFER: differentiable automated feature engineering. In *International Conference on Automated Machine Learning*. PMLR, 17–1.

[45] Alexandra Zytek, Ignacio Arnaldo, Dongyu Liu, Laure Berti-Equille, and Kalyan Veeramachaneni. 2022. The need for interpretable features: motivation and taxonomy. *ACM SIGKDD Explorations Newsletter* 24, 1 (2022), 1–13.

Table 5: Arithmetic functions

Unary	Binary
Log, Abs, square, sqrt,	$+$, $-$, $\div$, $\times$
Sin, Cos, Tanh, sigmoid,	$\wedge$, $\vee$
min-max normlaization, residual	
Residual, One-Hot-Encoding	

Table 6: Aggregation functions: N refers to Numerical and C refers to Categorical

N × N	N × C	C × C
min, max	GroupByThenMin, GroupByThenMax	Combine
	GroupByThenStd, GroupByThenMedian	CombineThenFrequency
	GroupByThenSum, GroupByThenMean	GroupByThenCountDistinct

A Feature Transformations

In our work, we used three (03) categories of transformations: arithmetic functions, aggregation functions and customized operators.

1. Arithmetic functions act on features and are divided into two categories: unary and binary. Unary transformations act on a single feature, while binary transformations operate on two features (Table 5).
2. Aggregation functions are categorized based on the type of features they work with (i.e, numerical or categorical), as shown in Table 6. For instance, the GroupByThenMean operator requires a categorical and a numerical feature, while *Min* operates on two numerical features.
3. Customized functions: we designed some peculiar functions to extract important information from dates such as *Day, Month, Year, Season, Is_Weekend, Is_Holiday, Is_Rush_Hour*...etc, *Duration* to calculate the duration between two date features, and *Haversine* to calculate the distance between two points given their longitudes and latitudes.

In public datasets, features are classified as numerical, categorical, or ordinal. The key distinction between ordinal and categorical features is that ordinal features have a defined order of categories (e.g., "Age"). When applying feature transformations, ordinal features can be treated as both numerical and categorical. For example, we can compute *GroupByThenMax(Gender, Age)*, which gives the average age for each gender, or *GroupByThenMean(Age,Income)*, which provides the average income for different age groups. When feature names are anonymized, we consider string-based features as categorical, discrete features (with less than 30 unique values) as ordinal, while continuous features are considered numerical.

B Knowledge representation

Since we defined feature interpretability based on domain knowledge, for our experiments, we constructed a comprehensive knowledge graph (KG[7]) that captures information across multiple domains using a description logic-based language. The KG, displayed in Figure 5, contains two types of knowledge:

[7]https://cutt.ly/bex8C2Cc

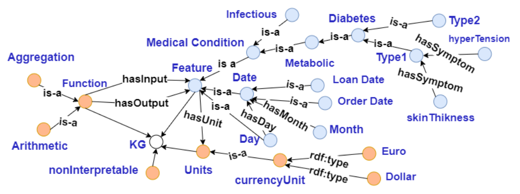


Figure 5: A sample of the KG.

1. Domain-agnostic knowledge includes the classes *unitsOfMeasurement*, *Function* and *nonInterpretable*. The former provides a broad range of measures and quantities from various domains, such as the International System of Units (e.g. Physics, Geometry, the International System of Units); The second class defines the vocabulary and semantics of transformations that we described in the previous section; *nonInterpretable* contains concepts and individuals that are considered as non-interpretable for domain experts. In addition, we used SWRL (Semantic Web Rule Language) to define specific rules to determine whether a feature is interpretable. We show an example below. For instance, the first rule states that adding two features with different units would result in a non-interpretable feature and that periodic inventory totals are not summable. Similar explanations could be given to the other rules.

(1) $Feature(?x) \wedge Feature(?y) \wedge Feature(?z) \wedge Addition(?f)$
$\wedge\ hasUnit(?x, ?u) \wedge hasUnit(?y, ?v) \wedge Different(?u, ?v) \wedge$
$hasInput(?f, ?x) \wedge hasInput(?f, ?y) \wedge hasOutput(?f, ?z)$
$\rightarrow nonInterpretable(?z)$

(2) $aggregationSum(?f) \wedge Stock(?x) \wedge hasInput(?f, ?x) \wedge$
$Feature(?z) \wedge hasOutput(?f, ?z) \rightarrow nonInterpretable(?z)$

(3) $Addition(?f) \wedge Temperature(?x) \wedge hasInput(?f, ?x) \wedge$
$Feature(?z) \wedge hasOutput(?f, ?z) \rightarrow nonInterpretable(?z)$

2. Domain-specific knowledge which is represented by the class *Feature*. This class covers concepts from various domain applications, such as Healthcare, Banking, Retail, E-commerce, Finance, and others. The knowledge graph is also linked to public knowledge graphs and ontologies from different domains, (e.g., DBpedia[8], INSEE[9]). This part of the knowledge can be expanded over time to cover additional application areas using knowledge base integration tools (e.g. RDFLib [10], Apache Jena [11]).

These concepts and rules form the TBox and ABox of our knowledge base. We display in Table 7 an example of such TBox.

ReaGen exploits the KG to generate new domain-specific features through symbolic reasoning. More specifically, we use HermiT [29], a DL-based reasoner, to extract additional knowledge. Due to space

[8]https://mappings.dbpedia.org/
[9]https://rdf.insee.fr/geo/index.html
[10]https://rdflib.readthedocs.io/en/stable/
[11]https://jena.apache.org/

Table 7: Sample of the TBox.

Function	$\sqsubseteq$	$\geq 1\ hasInput.(\forall hasUnit.Units) \sqcap \geq 1\ hasOutput.(\forall hasUnit.Units)$
Arithmetic	$\sqsubseteq$	Function $\sqcap \leq 2\ hasInput \sqcap \leq 1\ hasOutput$
Addition	$\sqsubseteq$	Arithmetic $\sqcap \geq 2\ hasInput.(\forall hasUnit.Unit) \sqcap \leq 1\ hasOutput.(\forall hasUnit.Unit)$
Cos	$\sqsubseteq$	$\leq 1\ hasInput.(\forall hasUnit.angleU) \sqcap \leq 1\ hasOutput.Double$
Sin	$\sqsubseteq$	$\leq 1\ hasInput.(\forall hasUnit.angleU) \sqcap \leq 1\ hasOutput.Double$
...		
$\perp$	$\sqsupseteq$	Feature $\sqcap$ Function
Date	$\sqsubseteq$	$\exists hasDay.Day \sqcap \exists hasMonth.Month \sqcap \exists hasYear.Year$
Location	$\sqsubseteq$	$\exists hasCountry.Country \sqcap \exists hasCity.City$
Energy	$\sqsubseteq$	$\exists hasMass.Mass \sqcap \exists hasVelocity.Velocity$
LoanApproval	$\sqsubseteq$	Customer $\sqcap \exists hasIncome.Income \sqcap \exists hasGoodCreditScore.Score$
PremiumInsured	$\sqsubseteq$	Customer $\sqcap \exists hasGoodHealth.Health \exists hasAge.Age \sqcap Age(young)$
...		

limitation, we display in Table 7 an overview of the TBox used in our experiments. For instance, if our dataset has a feature *Date*, we can use the following rule to infer three new features: *Day*, *Month*, and *Year*. We also can infer *Population Total* from *Location* or from *City* features. Furthermore, we use the relationships between entities in the KG to extract new features, by looking at the relations where a feature participates as a subject or an object, i.e. incoming and outgoing edges in the KG. For example, using the feature *City* and relation *locatedIn*, we can generate features for the entities that are located in this city, such as *Universities*, *Companies* or even important *Events*. These features may have a significant impact on the ML model.

C Additional results

C.1 Performance on different models

This experiment aims to answer the question: Is *ReaGen* robust across various machine learning models? We conducted the experiment on 10 different datasets from classification regression tasks. To evaluate robustness of our model, we used a range of classifiers, including Random Forest (RF), XGBoost (XGB), Logistic Regression for classification tasks and Linear Regression for regression tasks (LR), SVM (SVM), and Decision Tree (DT). Performance was measured using the same evaluation metrics of the previous experiments, i.e, *F1-score* and *1-rae*. The results, shown in Table 8, indicate that features generated by *ReaGen* consistently outperform those from other techniques across different ML models, showing a superior robustness of our model.

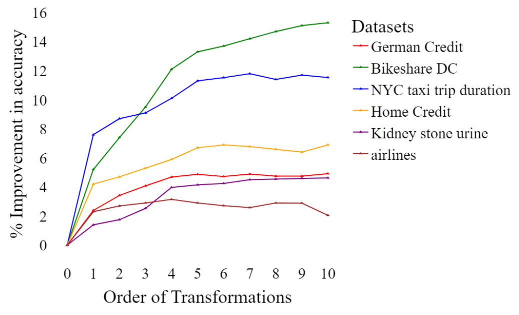


Figure 6: Effect of high-order features on accuracy improvement.

C.2 Effect of high-order features

Figure 6 illustrates the impact of generating high-order features, which involve composing multiple transformations to create a feature, on the improvement of prediction accuracy. Remarkably, the ability of *ReaGen* to generate high-order features significantly contributes to its performance. Specifically, we demonstrate that as the maximum order of features increases, the accuracy of *ReaGen* also improves. This observation underscores the effectiveness of generating high-order features, as the composition of transformations is pivotal for unveiling intricate relationships between features. In essence, the ability to synthesize complex features enhances the model's predictive power by capturing insightful patterns within the data. However, high-order features could become more challenging for domain expert to understand. So in practice, it would be reasonable to set an order of 2 to 3 to have a good trade-off between model's accuracy and feature interpretability.

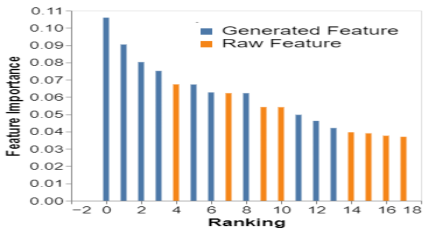
C.3 Feature Interpretability

C.3.1 Feature Importance. In this experiment, we use SHAP (SHapley Additive exPlanations) [19] to compare the importance of *ReaGen*'s generated features with raw features. We start by creating a new dataset combining the n raw features with the top-ranked n features generated by *ReaGen*. We then use SHAP to score feature importance in the model's predictions. Figure 7 showcases the results for 9 different datasets. Notably, the features generated by *ReaGen* (shown in blue) are more important compared to raw features (shown in orange) across all datasets, thereby validating the effectiveness of our model in generating interpretable and relevant features.

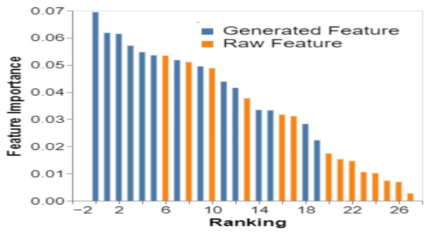
C.3.2 Model-agnostic Interpretability. Despite the agreement on the importance of feature interpretability, traditional AutoFE methods often neglect interpretability and focus on performance. To this end, in our work, we focus on feature interpretability for domain experts. We use in this experiments SHAP to quantify the contribution of input features to the model prediction, i.e., to attribute the predictions to specific parts of the input features. For a more in-depth analysis, we display in Figure 8 the top 10 features generated by our model across three different datasets from different domains. It can be seen that, compared to the baselines, the features are easily readable and trackable, enabling users to understand their meaning, and more interpretable, aligning with the domain knowledge.

Table 8: Comparing the effectiveness of *ReaGen* with AutoFE methods on benchmark datasets. ± indicates the standard deviation across 5 splits.

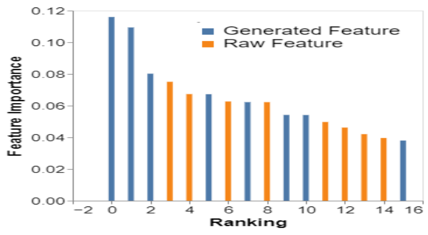
Dataset	ALG	Base	DFS	AutoFeat	NFS	OpenFE	mCAFE	ReaGen	Dataset	ALG	Base	DFS	AutoFeat	NFS	OpenFE	mCAFE	ReaGen
Diabetes	RF	$0.740_{\pm.04}$	$0.737_{\pm.05}$	$0.767_{\pm.06}$	$0.786_{\pm.10}$	$0.842_{\pm.05}$	$0.813_{\pm.04}$	**0.853**$_{\pm.03}$	**Medical appoin -tment**	RF	$0.491_{\pm.05}$	$0.499_{\pm.11}$	$0.790_{\pm.05}$	$0.650_{\pm.02}$	$0.824_{\pm.04}$	$0.786_{\pm.05}$	**0.853**$_{\pm.01}$
	DT	$0.732_{\pm.05}$	$0.732_{\pm.08}$	$0.741_{\pm.02}$	$0.770_{\pm.05}$	$0.830_{\pm.0}$	$0.798_{\pm.05}$	**0.849**$_{\pm.02}$		DT	$0.478_{\pm.01}$	$0.487_{\pm.05}$	$0.780_{\pm.05}$	$0.634_{\pm.05}$	$0.824_{\pm.02}$	$0.753_{\pm.04}$	**0.851**$_{\pm.03}$
	LR	$0.753_{\pm.05}$	$0.748_{\pm.05}$	$0.780_{\pm.08}$	$0.792_{\pm.05}$	$0.837_{\pm.04}$	$0.819_{\pm.06}$	**0.857**$_{\pm.01}$		LR	$0.502_{\pm.10}$	$0.517_{\pm.08}$	$0.803_{\pm.02}$	$0.702_{\pm.04}$	$0.839_{\pm.02}$	$0.753_{\pm.08}$	**0.870**$_{\pm.05}$
	SVM	$0.742_{\pm.05}$	$0.719_{\pm.04}$	$0.756_{\pm.05}$	$0.762_{\pm.05}$	$0.827_{\pm.05}$	$0.788_{\pm.05}$	**0.850**$_{\pm.04}$		SVM	$0.480_{\pm.07}$	$0.491_{\pm.04}$	$0.782_{\pm.07}$	$0.641_{\pm.08}$	$0.819_{\pm.05}$	$0.764_{\pm.10}$	**0.861**$_{\pm.03}$
	XGB	$0.755_{\pm.06}$	$0.750_{\pm.04}$	$0.788_{\pm.06}$	$0.792_{\pm.05}$	$0.823_{\pm.04}$	$0.819_{\pm.05}$	**0.863**$_{\pm.01}$		XGB	$0.503_{\pm.04}$	$0.515_{\pm.04}$	$0.801_{\pm.04}$	$0.713_{\pm.05}$	$0.860_{\pm.03}$	$0.838_{\pm.08}$	**0.892**$_{\pm.03}$
German Credit	RF	$0.742_{\pm.04}$	$0.780_{\pm.08}$	$0.796_{\pm.05}$	$0.805_{\pm.06}$	$0.813_{\pm.01}$	$0.795_{\pm.04}$	**0.827**$_{\pm.04}$	**Credit Default Risk**	RF	$0.797_{\pm.01}$	$0.802_{\pm.01}$	$0.806_{\pm.01}$	$0.799_{\pm.04}$	$0.810_{\pm.03}$	$0.801_{\pm.04}$	**0.837**$_{\pm.07}$
	DT	$0.720_{\pm.10}$	$0.722_{\pm.02}$	$0.752_{\pm.05}$	$0.791_{\pm.08}$	$0.803_{\pm.05}$	$0.765_{\pm.06}$	**0.809**$_{\pm.03}$		DT	$0.782_{\pm.08}$	$0.789_{\pm.01}$	$0.7.92_{\pm.01}$	$0.792_{\pm.01}$	$0.802_{\pm.03}$	$0.789_{\pm.02}$	**0.822**$_{\pm.04}$
	LR	$0.770_{\pm.05}$	$0.792_{\pm.01}$	$0.807_{\pm.06}$	$0.808_{\pm.09}$	$0.812_{\pm.01}$	$0.782_{\pm.01}$	**0.825**$_{\pm.05}$		LR	$0.801_{\pm.09}$	$0.804_{\pm.10}$	$0.805_{\pm.07}$	$0.807_{\pm.06}$	$0.811_{\pm.06}$	$0.805_{\pm.06}$	**0.829**$_{\pm.01}$
	SVM	$0.755_{\pm.01}$	$0.779_{\pm.01}$	$0.794_{\pm.01}$	$0.811_{\pm.01}$	$0.809_{\pm.03}$	$0.792_{\pm.05}$	**0.817**$_{\pm.03}$		SVM	$0.789_{\pm.05}$	$0.793_{\pm.01}$	$0.790_{\pm.03}$	$0.801_{\pm.0}$	$0.806_{\pm.10}$	$0.803_{\pm.05}$	**0.818**$_{\pm.04}$
	XGB	$0.790_{\pm.07}$	$0.795_{\pm.07}$	$0.801_{\pm.08}$	$0.814_{\pm.01}$	$0.815_{\pm.11}$	$0.808_{\pm.03}$	**0.832**$_{\pm.03}$		XGB	$0.802_{\pm.04}$	$0.804_{\pm.04}$	$0.811_{\pm.09}$	$0.819_{\pm.01}$	$0.814_{\pm.07}$	$0.809_{\pm.02}$	**0.825**$_{\pm.02}$
Kidney Stone	RF	$0.504_{\pm.01}$	$0.550_{\pm.01}$	$0.578_{\pm.01}$	$0.620_{\pm.10}$	$0.662_{\pm.11}$	$0.632_{\pm.07}$	**0.683**$_{\pm.02}$	**Openml _681**	RF	$0.428_{\pm.04}$	$0.411_{\pm.02}$	$0.632_{\pm.05}$	$0.640_{\pm.01}$	$0.732_{\pm.07}$	$0.738_{\pm.01}$	**0.740**$_{\pm.01}$
	DT	$0.495_{\pm.01}$	$0.559_{\pm.11}$	$0.562_{\pm.01}$	$0.613_{\pm.03}$	$0.649_{\pm.04}$	$0.607_{\pm.09}$	**0.672**$_{\pm.02}$		DT	$0.425_{\pm.01}$	$0.408_{\pm.05}$	$0.630_{\pm.08}$	$0.629_{\pm.07}$	$0.715_{\pm.02}$	$0.725_{\pm.10}$	**0.729**$_{\pm.02}$
	LR	$0.529_{\pm.01}$	$0.590_{\pm.09}$	$0.605_{\pm.05}$	$0.619_{\pm.02}$	$0.667_{\pm.11}$	$0.609_{\pm.07}$	**0.693**$_{\pm.07}$		LR	$0.432_{\pm.07}$	$0.412_{\pm.05}$	$0.645_{\pm.05}$	$0.642_{\pm.04}$	$0.736_{\pm.09}$	$0.740_{\pm.06}$	**0.749**$_{\pm.03}$
	SVM	$0.480_{\pm.07}$	$0.508_{\pm.05}$	$0.593_{\pm.09}$	$0.621_{\pm.05}$	$0.615_{\pm.01}$	$0.599_{\pm.21}$	**0.667**$_{\pm.02}$		SVM	$0.423_{\pm.05}$	$0.405_{\pm.01}$	$0.635_{\pm.05}$	$0.632_{\pm.01}$	**0.735**$_{\pm.04}$	$0.720_{\pm.07}$	$0.733_{\pm.01}$
	XGB	$0.509_{\pm.05}$	$0.593_{\pm.05}$	$0.608_{\pm.10}$	$0.658_{\pm.01}$	$0.684_{\pm.05}$	$0.661_{\pm.08}$	**0.698**$_{\pm.04}$		XGB	$0.430_{\pm.08}$	$0.413_{\pm.07}$	$0.642_{\pm.02}$	$0.645_{\pm.04}$	$0.739_{\pm.08}$	$0.735_{\pm.07}$	**0.743**$_{\pm.02}$
Amazon Employee	RF	$0.712_{\pm.09}$	$0.744_{\pm.07}$	$0.739_{\pm.02}$	**0.945**$_{\pm.11}$	$0.909_{\pm.01}$	$0.897_{\pm.02}$	$0.932_{\pm.01}$	**Bikeshare DC**	RF	$0.393_{\pm.07}$	$0.693_{\pm.15}$	$0.849_{\pm.04}$	$0.974_{\pm.02}$	$0.981_{\pm.09}$	$0.906_{\pm.07}$	**0.988**$_{\pm.04}$
	DT	$0.701_{\pm.02}$	$0.738_{\pm.10}$	$0.732_{\pm.02}$	**0.932**$_{\pm.07}$	$0.903_{\pm.02}$	$0.878_{\pm.14}$	$0.928_{\pm.02}$		DT	$0.378_{\pm.05}$	$0.682_{\pm.02}$	$0.840_{\pm.09}$	$0.965_{\pm.02}$	$0.959_{\pm.01}$	$0.896_{\pm.10}$	**0.975**$_{\pm.02}$
	LR	$0.715_{\pm.11}$	$0.750_{\pm.05}$	$0.742_{\pm.05}$	$0.935_{\pm.02}$	$0.913_{\pm.05}$	$0.905_{\pm.04}$	**0.941**$_{\pm.05}$		LR	$0.401_{\pm.02}$	$0.702_{\pm.02}$	$0.853_{\pm.02}$	$0.975_{\pm.02}$	$0.981_{\pm.15}$	$0.921_{\pm.05}$	**0.990**$_{\pm.01}$
	SVM	$0.710_{\pm.05}$	$0.740_{\pm.09}$	$0.735_{\pm.02}$	**0.936**$_{\pm.02}$	$0.910_{\pm.05}$	$0.906_{\pm.07}$	$0.928_{\pm.04}$		SVM	$0.380_{\pm.14}$	$0.695_{\pm.04}$	$0.845_{\pm.01}$	$0.978_{\pm.06}$	$0.978_{\pm.03}$	$0.922_{\pm.21}$	**0.982**$_{\pm.02}$
	XGB	$0.719_{\pm.04}$	$0.755_{\pm.02}$	$0.745_{\pm.15}$	**0.945**$_{\pm.09}$	$0.915_{\pm.02}$	$0.908_{\pm.02}$	**0.945**$_{\pm.01}$		XGB	$0.405_{\pm.06}$	$0.712_{\pm.07}$	$0.855_{\pm.03}$	$0.980_{\pm.03}$	$0.982_{\pm.05}$	$0.932_{\pm.06}$	**0.993**$_{\pm.03}$
airlines	RF	$0.620_{\pm.03}$	$0.612_{\pm.06}$	$0.618_{\pm.01}$	$0.621_{\pm.03}$	**0.628**$_{\pm.06}$	$0.622_{\pm.06}$	$0.625_{\pm.02}$	**NYC Taxi Ride**	RF	$0.425_{\pm.05}$	$0.505_{\pm.03}$	-	$0.551_{\pm.01}$	$0.501_{\pm.03}$	$0.446_{\pm.07}$	**0.589**$_{\pm.01}$
	DT	$0.604_{\pm.15}$	$0.601_{\pm.07}$	$0.604_{\pm.02}$	$0.615_{\pm.03}$	$0.618_{\pm.03}$	**0.619**$_{\pm.07}$	**0.619**$_{\pm.09}$		DT	$0.419_{\pm.05}$	$0.489_{\pm.03}$	-	$0.539_{\pm.23}$	$0.487_{\pm.15}$	$0.426_{\pm.01}$	**0.580**$_{\pm.03}$
	LR	$0.622_{\pm.05}$	$0.621_{\pm.11}$	$0.617_{\pm.09}$	$0.634_{\pm.11}$	**0.642**$_{\pm.03}$	$0.635_{\pm.07}$	$0.639_{\pm.15}$		LR	$0.430_{\pm.17}$	$0.513_{\pm.02}$	$0.458_{\pm.07}$	$0.559_{\pm.03}$	$0.515_{\pm.01}$	$0.452_{\pm.07}$	**0.604**$_{\pm.03}$
	SVM	$0.611_{\pm.09}$	$0.609_{\pm.03}$	$0.615_{\pm.08}$	**0.626**$_{\pm.07}$	$0.624_{\pm.03}$	$0.622_{\pm.08}$	**0.626**$_{\pm.01}$		SVM	$0.423_{\pm.08}$	$0.499_{\pm.04}$	-	$0.545_{\pm.08}$	$0.492_{\pm.03}$	$0.459_{\pm.17}$	**0.589**$_{\pm.02}$
	XGB	$0.625_{\pm.03}$	$0.626_{\pm.07}$	$0.607_{\pm.07}$	**0.639**$_{\pm.03}$	$0.638_{\pm.03}$	$0.635_{\pm.17}$	$0.635_{\pm.01}$		XGB	$0.432_{\pm.07}$	$0.515_{\pm.03}$	-	$0.562_{\pm.07}$	$0.523_{\pm.03}$	$0.486_{\pm.02}$	**0.613**$_{\pm.01}$



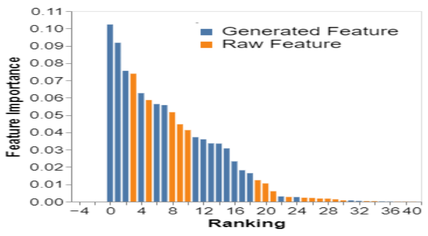
Amazon Employee



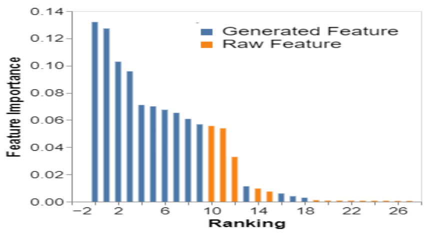
Wind



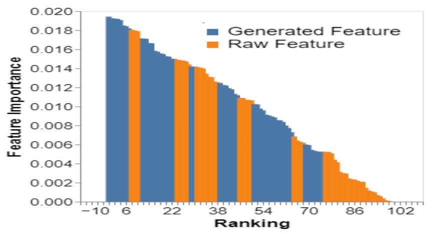
Diabete



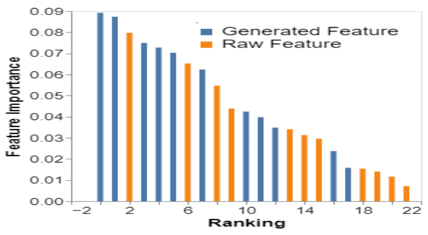
German Credit



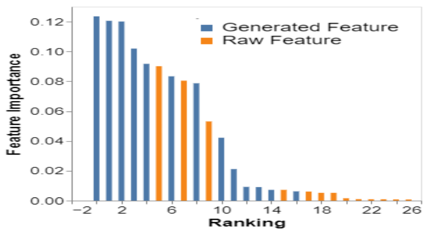
NYC taxi trip duration



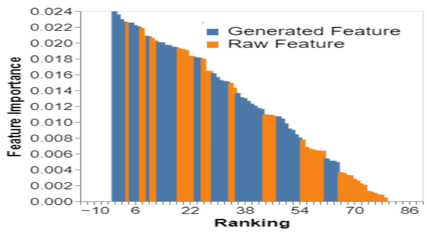
Spam Base



Bikeshare DC

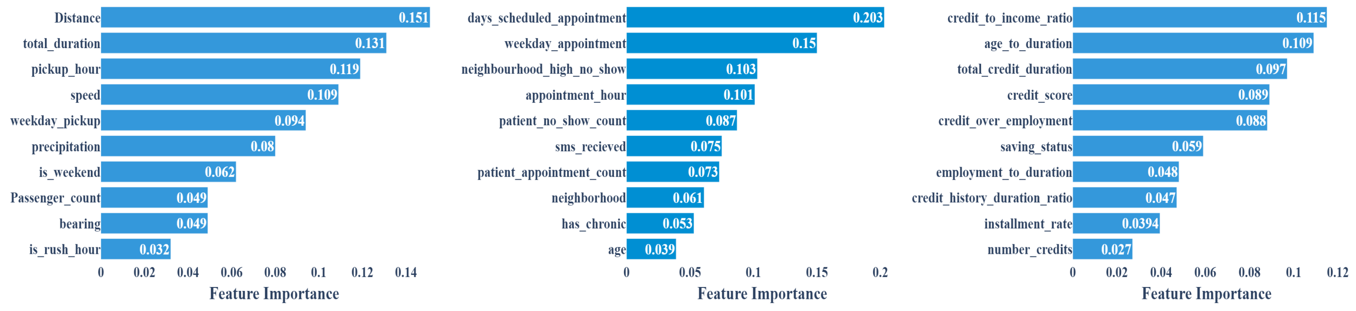


Medical Appointment



Ailerons

Figure 7: Comparing feature importance of raw features (in orange) and generated features with *ReaGen* (in blue).



(a) NYC Taxi Ride Duration **(b) Medical Appointment** **(c) German Credit**

Figure 8: Top-10 features of *ReaGen* on 3 different datasets.

For instance, in the *NYC Taxi Ride* dataset, our model generated 10 out of the top 10 features, including crucial indicators like *Distance* and *Duration* between pickup and dropoff locations and times, directly correlating with taxi ride duration. Moreover, it incorporates temporal aggregations such as *Is_Rush_Hour* and *Is_Weekend*, reflecting their significant impact on traffic conditions.

In the *Medical Appointment* dataset, *ReaGen* generates insightful features such as *Duration* in days between appointment scheduling and the actual appointment, indicating a correlation with appointment no-shows. Additionally, it captures factors like *Weekday*. A thorough analysis of this dataset showed that patients have a higher tendency to miss their appointments on weekdays rather than on weekends or holidays.

D Full LLM Prompt

We show in Figure 9 the full prompt generated by our model for NYC taxi trip duration dataset. The corresponding code is a response of the LLM to this prompt.

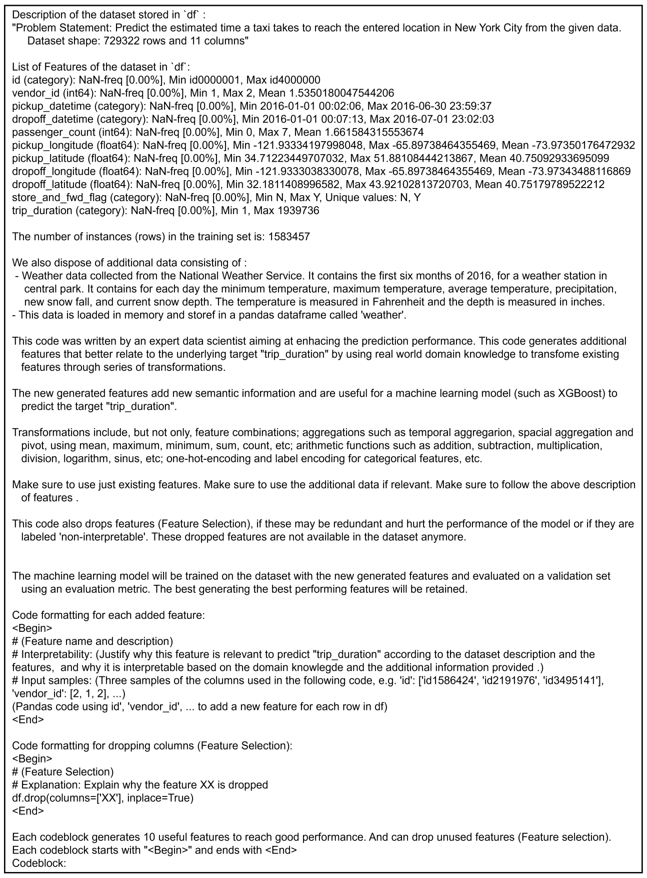
E Datasets Description

Our model leverage datasets description to automatically generate a prompt for feature generation. The dataset descriptions used in our experiments were extracted from the datasets respective source. We show in Figures 10 - 18 the description of some of the datasets used in our experiments.

F Broader Impact Statement

F.1 Limitations

ReaGen has some limitations. Its effectiveness relies on the quality of the dataset descriptions. If users provide inaccurate descriptions, the performance may drop significantly. However, in real world applications, this is less likely to be an issue since detailed descriptions of the data are typically available and of a good quality. In addition, handling datasets with a large number of features can result in very large prompts, which can be challenging for LLMs to process effectively. Likewise, incorporating additional information from external knowledge bases can increase the size of the prompts, especially when dealing with substantial amounts of knowledge. Finally, the use of LLMs involves broader societal impacts and ethical considerations, as detailed bellow.



```
Description of the dataset stored in `df` :
"Problem Statement: Predict the estimated time a taxi takes to reach the entered location in New York City from the given data.
    Dataset shape: 729322 rows and 11 columns"

List of Features of the dataset in `df`:
id (category): NaN-freq [0.00%], Min id0000001, Max id4000000
vendor_id (int64): NaN-freq [0.00%], Min 1, Max 2, Mean 1.5350180047544206
pickup_datetime (category): NaN-freq [0.00%], Min 2016-01-01 00:02:06, Max 2016-06-30 23:59:37
dropoff_datetime (category): NaN-freq [0.00%], Min 2016-01-01 00:07:13, Max 2016-07-01 23:02:03
passenger_count (int64): NaN-freq [0.00%], Min 0, Max 7, Mean 1.661584315553674
pickup_longitude (float64): NaN-freq [0.00%], Min -121.93334197998048, Max -65.89738464355469, Mean -73.97350176472932
pickup_latitude (float64): NaN-freq [0.00%], Min 34.71223449707032, Max 51.88108444213867, Mean 40.75092933695099
dropoff_longitude (float64): NaN-freq [0.00%], Min -121.9333038330078, Max -65.89738464355469, Mean -73.97343488116869
dropoff_latitude (float64): NaN-freq [0.00%], Min 32.1811408996582, Max 43.92102813720703, Mean 40.75179789522212
store_and_fwd_flag (category): NaN-freq [0.00%], Min N, Max Y, Unique values: N, Y
trip_duration (category): NaN-freq [0.00%], Min 1, Max 1939736

The number of instances (rows) in the training set is: 1583457

We also dispose of additional data consisting of :
 - Weather data collected from the National Weather Service. It contains the first six months of 2016, for a weather station in
   central park. It contains for each day the minimum temperature, maximum temperature, average temperature, precipitation,
   new snow fall, and current snow depth. The temperature is measured in Fahrenheit and the depth is measured in inches.
- This data is loaded in memory and storef in a pandas dataframe called 'weather'.

This code was written by an expert data scientist aiming at enhacing the prediction performance. This code generates additional
  features that better relate to the underlying target "trip_duration" by using real world domain knowledge to transfome existing
  features through series of transformations.

The new generated features add new semantic information and are useful for a machine learning model (such as XGBoost) to
  predict the target "trip_duration".

Transformations include, but not only, feature combinations; aggregations such as temporal aggregarion, spacial aggregation and
  pivot, using mean, maximum, minimum, sum, count, etc; arithmetic functions such as addition, subtraction, multiplication,
  division, logarithm, sinus, etc; one-hot-encoding and label encoding for categorical features, etc.

Make sure to use just existing features. Make sure to use the additional data if relevant. Make sure to follow the above description
  of features .

This code also drops features (Feature Selection), if these may be redundant and hurt the performance of the model or if they are
  labeled 'non-interpretable'. These dropped features are not available in the dataset anymore.


The machine learning model will be trained on the dataset with the new generated features and evaluated on a validation set
  using an evaluation metric. The best generating the best performing features will be retained.

Code formatting for each added feature:
<Begin>
# (Feature name and description)
# Interpretability: (Justify why this feature is relevant to predict "trip_duration" according to the dataset description and the
features,  and why it is interpretable based on the domain knowlegde and the additional information provided .)
# Input samples: (Three samples of the columns used in the following code, e.g. 'id': ['id1586424', 'id2191976', 'id3495141'],
'vendor_id': [2, 1, 2], ...)
(Pandas code using id', 'vendor_id', ... to add a new feature for each row in df)
<End>

Code formatting for dropping columns (Feature Selection):
<Begin>
# (Feature Selection)
# Explanation: Explain why the feature XX is dropped
df.drop(columns=['XX'], inplace=True)
<End>

Each codeblock generates 10 useful features to reach good performance. And can drop unused features (Feature selection).
Each codeblock starts with "<Begin>" and ends with <End>
Codeblock:
```

Figure 9: Full LLM Prompt for the NYC taxi trip duration dataset.

F.2 Social Impact of automated feature engineering

Traditional feature engineering is typically carried out manually by data scientists, consuming significant time and effort. Automating

Context: Over the past decade, bicycle-sharing systems have been growing in number and popularity in cities across the world. Bicycle-sharing systems allow users to rent bicycles for short trips, typically 30 minutes or less. With the latest technologies, it is easy for a user of the system to access a dock within the system to unlock or return bicycles. These technologies also provide a wealth of data that can be used to explore how these bike-sharing systems are used. The data consists of BikeShare information for three large cities in the US - New York City, Chicago, and Washington, DC.
Content: Sample Data
1. 'tripduration', '839'
2. 'starttime', '1/1/2016 00:09:55'
3. 'stoptime', '1/1/2016 00:23:54'
4. 'start station id', '532'
5. 'start station name', 'S 5 Pl & S 4 St'
6. 'start station latitude', '40.710451'
7. 'start station longitude', '-73.960876'
8. 'end station id', '401'
9. 'end station name', 'Allen St & Rivington St'
10. 'end station latitude', '40.72019576'
11. 'end station longitude', '-73.98997825'
12. 'bikeid', '17109'
13. 'usertype', 'Customer'
14. 'birth year', ''
15. 'gender', '0'

Figure 10: Description of Bikeshare DC dataset.

Problem Statement: Predict the estimated time a taxi takes to reach the entered location in New York City from the given data.
Dataset shape: 729322 rows and 11 columns

Figure 11: Description of NYC taxi trip duration dataset.

Dataset description: Medical appointments show/no-show: 110.527 medical appointments its 14 associated variables (characteristics). The most important one if the patient show-up or no-show to the appointment.
Data Dictionary
1. PatientId : Identification of a patient
2. AppointmentID : identification of each appointment
3. Gender : Male or Female.
4. Scheduled_time : The day of the actuall appointment, when they have to visit the doctor.
5. Appointment_time : The day someone called or registered the appointment, this is before appointment of course.
6. Age: How old is the patient.
7. Neighbourhood : Where the appointment takes place.
8. Scholarship : True of False .
9. Hipertension : True or False
10. Diabetes : True or False
11. Alcoholism : True or False
12. Handcap True or False
13. SMS_received: 1 or more messages sent to the patient.
14. No-show : True or False.

Figure 12: Description of medical appointment no-show dataset.

Dataset description: Kidney Stone
This dataset can be used to predict the presence of kidney stones based on urine analysis.
The 79 urine specimens, were analyzed in an effort to determine if certain physical characteristics of the urine might be related to the formation of calcium oxalate crystals.
The six physical characteristics of the urine are:
- (1) specific gravity, the density of the urine relative to water;
- (2) pH, the negative logarithm of the hydrogen ion;
- (3) osmolarity (mOsm), a unit used in biology and medicine but not in physical chemistry. Osmolarity is proportional to the concentration of molecules in solution;
- (4) conductivity (mMho milliMho). One Mho is one reciprocal Ohm. Conductivity is proportional to the concentration of charged ions in solution;
- (5) urea concentration in millimoles per litre;
- (6) calcium concentration (CALC) in millimolesllitre.

The data is obtained from 'Physical Characteristics of Urines With and Without Crystals'.

Figure 13: Description of Kidney Stone dataset.

this process can greatly reduce the workload of data scientists, enabling them to make faster decisions at lower costs. In this context, *ReaGen* streamlines the process of data preparation and feature extraction, reducing the time and effort required for data scientists to build and refine models. This increased efficiency allows organizations to analyze larger datasets and iterate on models more quickly, ultimately leading to greater productivity. *ReaGen* also empowers data scientists to explore a wider range of feature combinations and

Dataset description: A Credit Card Dataset for Machine Learning!
Context
Credit score cards are a common risk control method in the financial industry. It uses personal information and data submitted by credit card applicants to predict the probability of future defaults and credit card borrowings. The bank is able to decide whether to issue a credit card to the applicant. Credit scores can objectively quantify the magnitude of risk.

Generally speaking, credit score cards are based on historical data. Once encountering large economic fluctuations. Past models may lose their original predictive power. Logistic model is a common method for credit scoring. Because Logistic is suitable for binary classification tasks and can calculate the coefficients of each feature. In order to facilitate understanding and operation, the score card will multiply the logistic regression coefficient by a certain value (such as 100) and round it.

At present, with the development of machine learning algorithms. More predictive methods such as Boosting, Random Forest, and Support Vector Machines have been introduced into credit card scoring. However, these methods often do not have good transparency. It may be difficult to provide customers and regulators with a reason for rejection or acceptance.

Figure 14: Description of Home Credit Default Risk dataset.

German Credit dataset
This dataset classifies people described by a set of attributes as good or bad credit risks . This dataset comes with a cost matrix : “‘
Good Bad (predicted)
Good 0 1 (actual)
Bad 5 0 “‘
It is worse to class a customer as good when they are bad (5) , than it is to class a customer as bad when they are good (1) .
Attribute description
1. Status of existing checking account , in Deutsche Mark
2. Duration in months
3. Credit history (credits taken , paid back duly , delays , critical accounts)
4. Purpose of the credit (car , television ,...)
5. Credit amount
6. Status of savings account / bonds , in Deutsche Mark
7. Present employment , in number of years .
8. Installment rate in percentage of disposable income
9. Personal status (married , single ,...) and sex
10. Other debtors / guarantors
11. Present residence since X years
12. Property (e. g. real estate)
13. Age in years
14. Other installment plans (banks , stores)
15. Housing (rent , own ,...)
16. Number of existing credits at this bank
17. Job
18. Number of people being liable to provide maintenance for
19. Telephone (yes , no)
20. Foreign worker (yes , no)

Figure 15: Description of German Credit dataset.

** Breast Cancer Wisconsin (Original) Data Set .** Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass . They describe characteristics of the cell nuclei present in the image . The target feature records the prognosis (malignant or benign) .

Figure 16: Description of Breast cancer-w dataset.

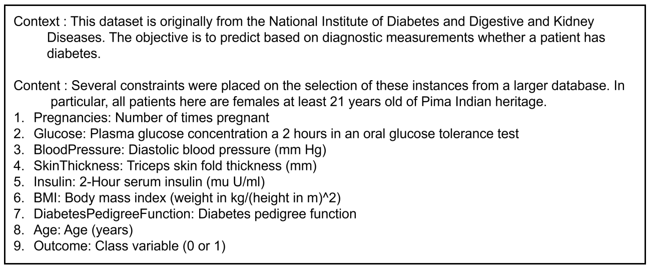
Airlines Dataset Inspired in the regression dataset from Elena Ikonomovska . The task is to predict whether a given flight will be delayed , given the information of the scheduled departure .

Figure 17: Description of airlines dataset.

transformations, uncovering valuable insights and patterns that may not be immediately apparent.

F.3 Interpretable Machine Learning

As the use of advanced AI techniques becomes more common, understanding and interpreting their results becomes crucial. Our approach focuses on feature interpretability for domain experts, using knowledge-based reasoning mechanisms. Our main goal is to simplify automate the feature engineering process by generating simple code and provide clear explanations about the proposed features using large language models. To reduce factual inaccuracies and hallucinations, typical in LLMs [27, 42], we combine LLMs with knowledge-based reasoning mechanisms, ensuring that the generated features and transformations are not only accurate, but also understandable to domain experts. Our focus on interpretability



Context : This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict based on diagnostic measurements whether a patient has diabetes.

Content : Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

1. Pregnancies: Number of times pregnant
2. Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. BloodPressure: Diastolic blood pressure (mm Hg)
4. SkinThickness: Triceps skin fold thickness (mm)
5. Insulin: 2-Hour serum insulin (mu U/ml)
6. BMI: Body mass index (weight in kg/(height in m)^2)
7. DiabetesPedigreeFunction: Diabetes pedigree function
8. Age: Age (years)
9. Outcome: Class variable (0 or 1)

Figure 18: Description of diabetes dataset.

helps bridge the gap between complex ML methods and symbolic AI, enabling collaboration between AI systems and experts and allowing them to confidently understand and validate the results. Additionally, by explaining the reasoning behind each generation, we enhance transparency and build trust in the AI-driven decision-making process.

F.4 Ethical Considerations

In light of observed ethical concerns replicated by AI technologies, including data privacy, bias, and transparency, it's crucial to address the potential bias implications within our approach, *ReaGen*, which uses GPT-3.5-turbo and GPT-4 trained on web-crawled data embedding societal biases. Notably, several studies have exemplified the presence of such biases in web-crawled data [26, 36].

Given this, when using data containing demographic or discriminative information, caution is advised in employing our approach. We advocate for meticulous scrutiny of generated features to mitigate these biases. Nevertheless, it is important to note that through the integration of domain knowledge and logical reasoning techniques, we aim to ensure that selected features are not only accurate but also understandable and interpretable by domain experts due to the way our method is set up, i.e. the features retained by our model are interpretable based on the semantics embedded in domain knowledge, and relevant leading to improvement in cross-validation. Additionally, in our approach, we allow for the possibility of introducing 'human in the loop', seeking explicit confirmation from experts before executing code especially when dealing with high-risk and critical domains.