Large Language Models for Automated Feature Engineering

Abstract. Automated feature engineering (AutoFE) aims to liberate data scientists from manual feature construction, which is crucial for improving the performance of machine learning models on tabular data. The semantic information of datasets provides rich context for AutoFE but is exploited in few existing work. In this paper, we introduce AutoFE by Prompting (FEBP), a novel AutoFE approach that leverages large language models (LLMs) to process dataset descriptions and automatically construct features. FEBP iteratively improves its solutions through in-context learning of top-performing examples and is able to semantically explain the constructed features. Experiments on seven public datasets show that FEBP outperforms state-of-the-art AuoFE methods by a significant margin. We also perform ablation study and feature analysis to verify the effect of semantic information and characterize the behavior of LLM-based feature search.

1 Introduction

Tabular data, a form of structured data comprising instances and attributes, have extensive use in numerous domains, e.g., credit assessment, market prediction, and quality control. Classical machine learning models, especially tree-based models [4], have strong performance on tabular datasets of small and medium sizes and high interpretability. Feature engineering is the process of computing new features from feature attributes of a dataset to enhance downstream model performance, which is crucial for classical ML models as it extracts useful information for predicting the target by capturing non-linear relationships. However, feature engineering by hand requires domain expertise and tremendous human labor.

Automated feature engineering (AutoFE) aims to develop high-level models and algorithms to automate the FE process and achieve comparable performance to domain experts. Many existing AutoFE methods, such as DIFER [23] and OpenFE [21], compute and evaluate a large number of features in a trial-and-error manner. While some of these methods learn to optimize the quality of features during AutoFE, they do not utilize prior knowledge to guide feature search. The need to start searching from scratch for new datasets or downstream models hampers their effectiveness and efficiency. Besides, these methods do not explain their solutions and may generate over complex features that affect the interpretability of downstream models.

Most tabular datasets contain descriptions of the dataset and attributes, providing rich context for FE. A feature engineer may consult attribute descriptions to select feature attributes and compute new features that are useful for target prediction. For instance, the *square footage* of a house times the *average housing price per square foot* in neighborhood could be a good predictor of the *market value* of the house. Large language models (LLMs) [14, 2, 13, 15, 16], pretrained on large volumes of text data, excel in natural language processing and encapsulate extensive domain knowledge transferable across datasets. This suggests that given proper instructions, an LLM may process the semantic information of a dataset and utilize its knowledge to perform FE in a similar manner to domain experts. Previously, CAAFE [5] explores this idea by instructing the LLM to generate code in Python, but it is not sufficiently effective in terms of feature search.

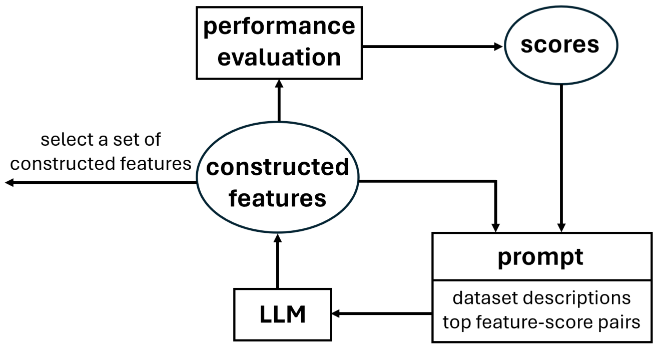


Figure 1. Overview of FEBP

In light of this, we propose a novel AutoFE approach leveraging LLMs for effective and efficient feature engineering, called AutoFE by Prompting (FEBP). As illustrated in Figure 1, we provide the LLM with dataset descriptions and example features represented in canonicalized reverse Polish notation (RPN) and prompt it to construct new features. After evaluating the constructed features, we update in the prompt top-ranked features with evaluation scores and instruct the LLM to construct further features. In this way, the LLM iteratively explores the search space and improves its solutions. The semantic information of dataset descriptions not only informs feature search, but also helps the LLM better understand example features to learn their patterns in-context. Utilizing its domain knowledge, the LLM constructs semantically meaningful features and explains the usefulness of features, enhancing the interpretability of downstream models. Experiments on seven public datasets show that our approach outperforms state-of-the-art baseline methods with statistical significance and achieves over 8% performance gain over three downstream models on average. Furthermore, our ablation study shows that the semantic context of dataset descriptions helps improve the performance.

We summarize our main contributions as follows:

- We propose a novel LLM-based AutoFE approach that exploits the semantic information of datasets and performs adaptive feature search.

- We conduct experimental evaluations of our approach against state-of-the-art baselines using GPT-3.5 and GPT-4.
- We perform analysis on the effect of semantic context on our approach and the behavior of LLM-based feature search.

2 Related Work

2.1 Large Language Models (LLMs)

LLMs are large-scale general-purpose neural networks pretrained on large corpora of raw text data for natural language processing, typically built with transformer-based architectures [17]. Generative LLMs, such as GPT family [14, 2, 13] and LLaMA family [15, 16], are pretrained to successively predict the next token given the input text and can be finetuned using reinforcement learning from human feedback (RLHF) [24]. By this means, they acquire the knowledge about syntax and semantics of human languages and are able to achieve state-of-the-art performance on various tasks like text generation, summarization, and question answering. LLMs can be adapted to specific tasks without changing model parameters through prompt engineering. One approach is to include examples in the prompt for the model to learn in-context, i.e., few-shot learning [2]. Leveraging such capability, an LLM may function as a problem solver [19] that iteratively improves candidate solutions according to the task description and feedback.

2.2 Automated Feature Engineering (AutoFE)

Automated feature engineering computes new features for the input data and augments or replaces portions of the existing features, with the aim to enhance the performance of downstream models. Common AutoFE approaches include expansion-reduction [7, 6, 21], genetic algorithms [22], and reinforcement learning [9, 10]. DIFER [23] utilizes neural networks to learn the quality of constructed features and optimize features in the embedding space. OpenFE [21] proposes a feature boost algorithm to speedup feature evaluation. Nonetheless, these approaches do not exploit the semantic information of datasets, which affects their performance and the interpretability of solutions.

2.3 AutoFE with Domain Knowledge

The benefits of incorporating domain knowledge in AutoFE are twofold: (1) reducing the cost of learning an AutoFE model, especially feature evaluation overhead; (2) improving the effectiveness of AutoFE. Previous work espousing this idea takes different approaches. One approach is to transfer the knowledge from past AutoFE experience. LFE [12] represents features with quantile sketches that are transferable across datasets, and inputs them to a transformation recommendation model. FETCH [10] is an RL-based AutoFE framework that takes tabular data as the state and is generalizable to new data. E-AFE [18] pretrains a feature evaluator to efficiently learn its RL-based AutoFE model. Another approach is to exploit the semantic information of datasets. KAFE [3] leverages knowledge graphs to identify semantically informative features relevant to the prediction task. CAAFE [5] manipulates Pandas data frames using the code produced from the LLM based on dataset descriptions. Our work also exploits the domain knowledge of LLMs, but we adopt a compact form of feature representations with pre-defined transformation operators. Our approach reduces the search space and helps the LLM learn the patterns of useful features, leading to stronger and more robust performance.

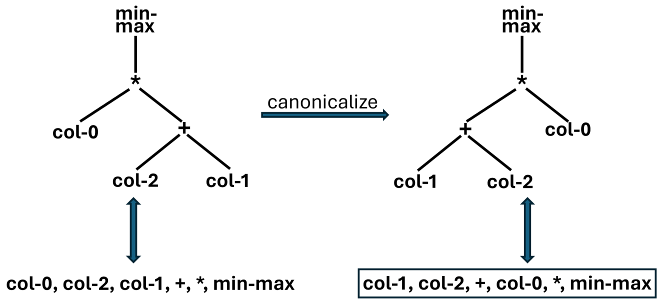


Figure 2. RPN, expression tree, and canonicalization

3 Notations

We denote a tabular dataset as $D = \langle X, \mathbf{y} \rangle$, where $X = \{\mathbf{x}_1, \dots, \mathbf{x}_d\}$ is the set of raw features with $\mathbf{x}_i \in \mathbb{R}^n$ for $i = 1, \dots, d$ and $\mathbf{y} \in \mathbb{R}^n$ is the target. A new feature $\tilde{\mathbf{x}} = t(\mathbf{x}_{j_1}, \dots, \mathbf{x}_{j_o})$ can be constructed through the transformation of existing features via some operator $t \in \mathbb{R}^n \times \dots \times \mathbb{R}^n \to \mathbb{R}^n$ of arity o. Given a set of transformation operators T, we define the feature space X^T recursively as: for any $\tilde{\mathbf{x}} \in X^T$, either $\tilde{\mathbf{x}} \in X$; or $\exists t \in T$, s.t., $\tilde{\mathbf{x}} = t(\tilde{\mathbf{x}}_{j_1}, \dots, \tilde{\mathbf{x}}_{j_o})$, where $\tilde{\mathbf{x}}_{j_1}, \dots, \tilde{\mathbf{x}}_{j_o} \in X^T$.

We define the order of a feature $\tilde{\mathbf{x}} \in X^T$ as:

$$\alpha(\tilde{\mathbf{x}}) = \begin{cases} 0 & \text{if } \tilde{\mathbf{x}} \in X, \\ 1 + \max_j \alpha(\tilde{\mathbf{x}}_j) & \text{if } \tilde{\mathbf{x}} = t(\tilde{\mathbf{x}}_{j_1}, \dots, \tilde{\mathbf{x}}_{j_o}) \text{ for some } t \in T. \end{cases} \quad (1)$$

The constrained feature space by an upper limit on the order k is denoted as $X^T_k = \{\tilde{\mathbf{x}} \in X^T \mid \alpha(\tilde{\mathbf{x}}) \leq k\}$.

We denote the performance evaluation score of a downstream machine learning model M on the dataset as $\mathcal{E}_M(X, \mathbf{y})$. The goal of AutoFE is to construct a set of features $\tilde{X}^*$ to add to the dataset such that the model performance is optimized, formally:

$$\tilde{X}^* = \arg \max_{\emptyset \neq \tilde{X} \subseteq X^T \setminus X} \mathcal{E}_M(X \cup \tilde{X}, \mathbf{y}). \quad (2)$$

We can parse any feature $\tilde{\mathbf{x}} \in X^T$ to an expression tree, where leaf nodes are raw features and internal nodes are operators [23]. For features that include commutative operators like addition and multiplication, the expression tree is not unique since the children of commutative operators are unordered. We adopt a canonicalization scheme for ordering the children so that the expression tree becomes unique: we arrange operator nodes before feature nodes and lexicographically sort nodes within each of the two groups. We then represent the feature with the post-order traversal string of the canonicalized expression tree, a.k.a., reverse Polish notation (RPN). Figure 2 illustrates an example. The feature corresponding to an RPN string f is denoted as $\tilde{\mathbf{x}}_f$; the set of features corresponding to a set of RPN strings F is denoted as $\tilde{X}_F$.

4 Proposed Method

In this section, we propose a novel iterative AutoFE approach leveraging LLMs, particularly, *GPT* models [14, 2, 13]. We call our approach AutoFE by Prompting (FEBP). The main idea is to provide the LLM with descriptive information of the dataset in the prompt and guide it to search for effective features using examples.

Our prompt primarily consists of:

1. A meta description of the dataset;

2. A list of indexed attributes of the dataset, with attribute types, value ranges, and descriptions;
3. Lists of transformation operators with descriptions, grouped by the arity;
4. A list of example features with performance evaluation scores ranked in the ascending order;
5. An output template of features and explanations.

The descriptions of the dataset, features, and the target provide contextual information necessary for the LLM to understand the dataset and apply domain knowledge. We include descriptions of transformation operators as they help the LLM parse feature strings in RPN syntax and construct syntactically valid feature strings. The value ranges of attributes are useful for selecting appropriate transformations to apply on features, e.g., min-max normalization when the scale is too large. The template not only formats the output but also instructs the LLM to reason about the usefulness of proposed features and make semantic explanations. Additionally, we append a constraint instruction asking the LLM to use no more than a certain number of operators, which reduces the search space and regularizes the solutions. A full prompt is presented in Figure 3. It may be helpful to include other attribute statistics in the prompt, e.g., mean, standard deviation, and skewness. The examination of their effects is left for future work.

We initialize the prompt with k simple random features in the constrained feature space $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_k \in X_2^T$ in canonicalized RPNs as seeds, without performance evaluation. Our rationale is to let the LLM start search from a small feature space, where it is easier to identify basic patterns of promising features. We ask the LLM to propose m new feature strings in each feature construction iteration. For each feature string f, we check whether it is valid and not duplicate with previously evaluated features. If both criteria are met, we evaluate the performance score of adding this feature to the dataset $s = \mathcal{E}_M(X \cup \{\tilde{\mathbf{x}}_f\}, \mathbf{y})$ through cross validation on the training data and add $\langle f, s\rangle$ to the candidate set F_{cand}. When f is among the top-k candidate features in terms of the score s, we update examples in the prompt with the top-k feature-score pairs $\langle f', s'\rangle \in F_{cand}$ ranked in the ascending order, taking incremental performance scores $s' - \mathcal{E}_M(X, \mathbf{y})$ from the baseline. We then use the updated prompt to instruct the LLM to further propose features. Once feature construction completes, we successively add candidate features to the dataset from the best to the worst. The optimal number of features to add is determined based on validation performance, which takes feature interactions into account.

Methodologically, we instruct the LLM to act as a problem solver [19] in our approach. Analogous to genetic algorithms [11, 22] that produce new solutions through recombinations and mutations on existing solutions with high fitness, we maintain a pool of top-performing candidate solutions as examples. By learning examples and scores in-context [2], the LLM is able to recognize the patterns of promising features and propose new features that are likely to be useful. It may, for instance, make analogies to, modify, or combine some of the example features. We expect that the beginning of the search is more exploratory due to diversity in initial examples. As iteration goes on, the LLM learns to exploit promising feature space, so the search becomes more focused and would eventually converge. In addition, the dataset descriptions serve as a prior that guides the selection of feature attributes and operators, improving the effectiveness of feature search. The sampling temperature of the LLM can be tuned to balance between exploration and exploitation. A high temperature encourages new solutions to be different from the examples; while a low temperature prefers small changes to examples.

Algorithm 1: AutoFE by Prompting

Input : Dataset $D = \langle X, \mathbf{y}\rangle$ and model M
Output: A set of feature strings F
1 Initialize prompt P with dataset descriptions and example features
2 $F_{cand} \leftarrow \emptyset$
3 **repeat** `// feature construction`
4 Ask the LLM to propose m feature strings using prompt P
5 **for** *each proposed feature string* f **do**
6 **if** f *is valid and* $f \notin F_{cand}$ **then**
7 Evaluate cross validation performance score $s = \mathcal{E}_M(X \cup \{\tilde{\mathbf{x}}_f\}, \mathbf{y})$ on training data
8 $F_{cand} \leftarrow F_{cand} \cup \{\langle f, s\rangle\}$
9 Replace in prompt P existing $\langle \bar{f}, \bar{s}\rangle$ with top-k $\langle f', s'\rangle \in F_{cand}$ on s'
10 **end**
11 **end**
12 **until** *maximum number of iterations*
13 **for** $n \leftarrow 1$ **to** $|F_{cand}|$ **do** `// feature selection`
14 Select top-n feature strings F_n in F_{cand} on s
15 Evaluate performance score $\mathcal{E}_M(X \cup \tilde{X}_{F_n}, \mathbf{y})$ on validation data
16 **end**
17 return F_n with the maximum validation performance score

Algorithm 1 summarizes our proposed method. The cost of query to the LLM in line 4 scales linearly with the number of features in the dataset and the number of examples k in the prompt, but remains constant across feature construction iterations. The computation cost of feature evaluation in line 7 also remains constant. Feature evaluations in line 7 and lines 13-16 are parallelizable. Figure 4 shows an example output, where the LLM proposes a new feature in RPN and explains its usefulness from the semantic perspective.

The transformation operators we adopt include:

- Unary transformations: logarithm, reciprocal, square root, and min-max normalization;
- Binary transformations: addition, subtraction, multiplication, division, and modulo.

When computing min-max normalization, we take the minimum and maximum from the training data. Other transformations only require information from a single row of the table. Hence, all these transformation operations can be performed instance by instance on test examples.

5 Experiments

5.1 Experimental Setup

We benchmark on seven public datasets from Kaggle[1] and UCI repository[2] with descriptive information of the dataset and attributes, listed in Table 1. Each dataset is randomly split into training, validation, and test sets with the ratio $16:4:5$. The downstream models we evaluate include linear models (Lasso regression for regression and logistic regression for classification), Random Forest [1], and

[1] https://www.kaggle.com
[2] https://archive.ics.uci.edu

Dataset description:
This dataset contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan from April 2005 to September 2005.
Dataset contains the following columns:
col-0 (int) [10000, 800000]: LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementary credit
col-1 (category) {1, 2}: SEX: Gender (1=male, 2=female)
col-2 (category) {0, 1, 2, 3, 4, 5, 6}: EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)
col-3 (category) {0, 1, 2, 3}: MARRIAGE: Marital status (1=married, 2=single, 3=others)
col-4 (int) [21, 79]: AGE: Age in years
col-5 (category) {-2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8}: PAY_0: Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=payment delay for two months, ... 8=payment delay for eight months, 9=payment delay for nine months and above)
...
col-23 (category) {0, 1}: default.payment.next.month: Default payment (1=yes, 0=no)
We have the following unary operators:
log: taking the log of the absolute value
sqrt_abs: taking the square root of the absolute value
min_max: min-max normalization
reciprocal: taking the reciprocal
We have the following binary operators:
+: summing two columns
−: subtracting two columns
∗: multiplying two columns
/: taking the division of two columns
mod_column: taking the modulo of two columns
Feature strings are reverse Polish notation (RPN) expressions that operate on the columns of our dataset. Each feature string constructs an extra column that is useful for the downstream model random forest to predict the target col-23. The model will be trained on the dataset with the constructed columns and evaluated on a holdout set. The best columns will be selected.
Below are feature strings arranged in ascending order based on their performance scores. Higher scores are better.
Feature
col-17,col-21,*,col-20,+,sqrt_abs
Score
0.0011
...
Feature
col-4,col-6,*,col-12,col-16,-,sqrt_abs,*
Score
0.0014
Give me a new feature string that is different from all strings above and has a higher score. Use no more than five operators. Make sure all columns and operators exist and do not include the target column. Follow the syntax of RPN.
Output format:
Feature

Usefulness
(Explanation why this adds useful real world knowledge to predict the target according to dataset description)

Figure 3. Example prompt

Feature
col-4,col-5,/,col-6,*,col-15,*,col-18,*,min_max

Usefulness
This feature calculates the ratio of age to repayment status in August 2005, multiplied by the amount of bill statement in July 2005, and previous payment in August 2005, then normalizes the result. This feature captures the relationship between age, repayment behavior, bill amounts, and previous payments, which can provide insights into how these factors influence the likelihood of default payment.

Figure 4. Example output of the LLM

LightGBM [8]. On linear models, we target encode categorical features. We tune model parameters using randomized search both prior to and post AutoFE (see details in our code), because the model may need to be reconfigured to accommodate the extra features. We evaluate regression performance with $1-(relative\ absolute\ error)$ and classification performance with accuracy. For both metrics, a higher score indicates better performance.

We compare FEBP against the following state-of-the-art AutoFE methods:

Table 1. Datasets used in our experiments, where R stands for regression and C stands for classification (the number of classes is shown in bracket).

Name	Task	# Samples	# Features
Airfoil (AF)	R	1503	5
Boston Housing (BH)	R	506	13
Bikeshare (BS)	R	731	10
Wine Quality Red (WQR)	R	1599	11
AIDS Clinical Trials (ACT)	C[2]	2139	23
Credit Default (CD)	C[2]	30000	23
German Credit (GC)	C[2]	1000	20

- DIFER[3] [23]: A neural network-based method that optimizes features in the embedding space.
- OpenFE[4] [21]: An expansion-reduction method that evaluates and ranks first-order features using a feature boost algorithm.
- CAAFE[5] [5]: An LLM-based method that produces Python code based on dataset descriptions.

[3] https://github.com/PasaLab/DIFER
[4] https://github.com/IIIS-Li-Group/OpenFE
[5] https://github.com/automl/CAAFE

We adopt *GPT-3.5 Turbo*[6] and *GPT-4*[7] as LLMs in our evaluation. For FEBP, we include $k = 10$ example features in the prompt and set the temperature of GPT models to 1. We prompt the LLM to construct $m = 1$ feature in each query for accurate control of feature construction and reducing the number of feature evaluations. We perform feature selection each time 10 new features are add to the candidate set, terminate when there are 200 candidate features in total, and then select the best subset of features based on validation performance. For CAAFE [5], we set the number of iterations to 20. Further increasing this limit is likely to cause failures due to the context window of GPT models. Other parameters of baseline methods are initialized as per the corresponding papers. We report the results from five repeated runs unless stated otherwise.

5.2 *Performance Comparison*

Table 2 compares the performance of all methods. While there is no single method that dominates all cases, our method FEBP achieves the best overall performance and the lowest mean rank. FEBP yields over 8% improvement over baseline model performance on average, with over 22% gain on linear models and $\sim$3% gain on both Random Forest and LightGBM. The greater performance gain on linear models is because Random Forest and LightGBM are able to model complex relationships themselves. Our paired t-tests show that the performance margin of FEBP over other AutoFE methods except DIFER is statistically significant with p-value < 0.001. FEBP is considerably more efficient in the sense that it evaluates only 200 candidate features, whereas DIFER evaluates over 1000 candidate features.

We also note that the performance of FEBP and CAAFE using GPT-4 is not significantly different from using GPT-3.5. Using GPT-4 yields better performance on linear models but slightly worse performance on Random Forest. We speculate this may be because the stronger in-context learning capability of GPT-4 is more likely to cause overfitting on the training data.

5.3 *Effect of Semantic Context*

To verify the effect of semantic context on our method, we compare the full version with a blinded version where the descriptions of dataset and operators are removed. From Table 3, the performance of blinded version degrades. Our paired t-tests show that the performance margin is statistically significant with p-value $< 1e^{-4}$. Despite the blinded version performs reasonably well on linear models, the performance decline is more evident on Random Forest and LightGBM, probably because the construction of non-semantically meaningful features overfits the training data.

We also report the number of LLM responses to understand the efficiency of feature construction. From Table 3, the incorporation of semantic context improves the efficiency of GPT-3.5 but decreases the efficiency of GPT-4. One possible reason is that the semantic information injects some bias that causes GPT-4 to reproduce similar responses.

5.4 *Performance Analysis*

We analyze the behavior of FEBP for further insights. Here we report the experimental results on ACT, BH, GC, and WQR datasets with linear models from ten repeated runs using *GPT-3.5 Turbo*.

[6] https://platform.openai.com/docs/models/gpt-3-5
[7] https://platform.openai.com/docs/models/gpt-4-turbo-and-gpt-4

Feature Learning

We investigate the cross validation score on training data across feature construction iterations. Figure 5 shows that the score tends to increase with the number of iterations. This validates that FEBP is able to improve the quality of constructed features through in-context learning of top-ranked examples and scores during feature search.

Feature Complexity

We investigate the order of candidate features across feature construction iterations. Figure 6 shows that the feature order increases faster in early iterations and then becomes more stable. On the one hand, we see that FEBP learns to explore into more complex features in promising feature space. On the other hand, our constraint instruction regularizes the solutions to avoid over complex features.

Feature Divergence

We analyze the divergence of a new candidate features from previous features during feature construction. We compute the edit distance between canonicalized feature expression trees using the algorithm in [20] and normalize it by the sum of nodes in the two trees. Figure 7 shows the mean normalized tree edit distance between the current feature and previous five features across iterations. The declining trend we observe indicates that feature search is converging.

Feature Construction Efficiency

We investigate the number of LLM responses to construct a new candidate feature. Figure 8 shows that there is an increasing trend with iterations, meaning that more responses are dropped. This is because it becomes more difficult to construct a non-duplicate feature and also syntactical errors are more likely to occur when features gets more complex. Overall, the increase is quite small, so FEBP is scalable to a large number of iterations.

6 Conclusion and Future Work

In this paper, we propose a novel LLM-based AutoFE approach for effective and efficient feature engineering that exploits the semantic information of datasets. We provide the LLM with dataset descriptions and example features in RPN and prompt it to construct new features. The LLM iteratively explores the search space and improves its solutions. Experiments show that our approach outperforms state-of-the-art baseline methods with statistical significance and the semantic context of dataset descriptions helps improve the performance. We characterize the behavior of LLM-based feature search in our analysis. For future work, we consider introducing adaptive techniques to the prompt design, such as automated prompt engineering.

Ethics Statement

All datasets used in this work are public and free of personal information. Datasets are for research purpose only. Our usage of GPT models complies with the terms and conditions of OpenAI.

Table 2. Summary of experimental results. For each compared method, the left and right columns show the results without and with parameter tuning post AutoFE, respectively. The best results are highlighted in boldface, and the second best results are underlined.

Model	Dataset	Raw	DIFER		OpenFE		CAAFE				FEBP			
							GPT-3.5		GPT-4		GPT-3.5		GPT-4	
Linear Model	AF	0.3474	0.5870	0.6090	0.4300	0.4303	0.4011	0.4016	0.4376	0.4378	0.6612	0.6616	**0.6649**	0.6647
	BH	0.3776	0.5013	0.4994	0.3900	0.3880	0.4788	0.4765	0.4503	0.4506	0.4995	0.5025	0.5184	**0.5289**
	BS	1.0000	-	-	-	-	-	-	-	-	-	-	-	-
	WQR	0.2696	0.2475	0.2630	0.2713	0.2736	0.2742	0.2757	**0.2776**	0.2776	0.2722	0.2745	0.2713	0.2748
	ACT	0.8505	0.8715	**0.8799**	0.8729	0.8729	0.8519	0.8514	0.8565	0.8570	0.8729	0.8794	0.8766	0.8762
	CD	0.8267	0.8273	0.8280	0.8265	0.8268	0.8265	0.8267	0.8238	0.8238	0.8282	0.8282	0.8288	**0.8288**
	GC	0.7100	0.7140	0.7420	0.7320	0.7280	0.7350	0.7330	0.7210	0.7210	0.7570	0.7460	**0.7590**	0.7420
Random Forest	AF	0.7677	0.7650	0.7786	0.7579	0.7682	0.7711	0.7693	0.7696	0.7720	0.7709	**0.7787**	0.7681	0.7749
	BH	0.5426	**0.5718**	0.5701	0.5658	0.5620	0.5556	0.5556	0.5512	0.5492	0.5549	0.5533	0.5543	0.5522
	BS	0.9446	0.9865	0.9871	0.9901	0.9901	**0.9916**	**0.9916**	0.9818	0.9816	0.9873	0.9881	0.9845	0.9848
	WQR	0.3662	0.3838	0.3832	0.3753	0.3729	0.3718	0.3718	0.3693	0.3693	**0.3862**	0.3845	0.3810	0.3810
	ACT	0.8808	0.8897	0.8897	0.8832	0.8841	0.8827	0.8855	0.8827	0.8827	**0.8925**	0.8921	0.8893	0.8864
	CD	0.8293	0.8285	0.8291	0.8287	0.8285	0.8291	0.8289	0.8294	0.8287	0.8295	0.8294	**0.8295**	0.8276
	GC	0.7450	0.7550	0.7500	0.7650	0.7570	**0.7690**	0.7620	0.7660	0.7630	0.7640	0.7620	0.7680	0.7680
Light-GBM	AF	0.8375	0.8285	0.8411	0.8188	0.8244	0.8364	0.8348	**0.8430**	0.8426	0.8311	0.8392	0.8366	0.8395
	BH	0.5537	0.5607	0.5636	**0.5693**	0.5618	0.5540	0.5571	0.5478	0.5501	0.5619	0.5644	0.5642	0.5595
	BS	0.9429	0.9763	0.9786	0.9751	0.9797	0.9555	0.9565	0.9449	0.9487	0.9737	0.9754	0.9801	**0.9813**
	WQR	0.3825	0.4145	**0.4182**	0.3898	0.3884	0.4131	0.4035	0.3902	0.3952	0.4118	0.4171	0.4021	0.4042
	ACT	0.8832	0.8794	0.8827	0.8808	0.8799	0.8822	0.8860	0.8827	0.8818	0.8888	**0.8925**	0.8902	**0.8925**
	CD	0.8300	0.8283	0.8277	0.8293	0.8287	0.8296	0.8298	0.8301	0.8294	0.8301	0.8297	**0.8303**	0.8294
	GC	0.7250	0.7650	0.7600	0.7550	0.7700	0.7490	0.7550	0.7450	0.7720	0.7680	0.7720	**0.7760**	0.7700
Mean		0.6806	0.7091	0.7140	0.6953	0.6958	0.6979	0.6976	0.6950	0.6967	0.7171	0.7185	**0.7187**	0.7183
Mean Rank		10.95	7.90	5.50	8.35	8.23	7.55	7.58	8.50	8.65	4.88	**3.70**	4.25	4.98

Table 3. Performance comparison of FEBP with and without semantic blinding. For each compared setting, the left and middle columns show the results without and with parameter tuning post AutoFE, respectively, and the right column shows the number of LLM responses. The results where the default version outperforms the blinded version are highlighted in boldface.

Model	Dataset	Raw	GPT-3.5						GPT-4					
			Blinding			Default			Blinding			Default		
Linear Model	AF	0.3474	0.6613	0.6602	450.0	0.6612	**0.6616**	481.2	0.6678	0.6672	275.0	0.6649	0.6647	371.4
	BH	0.3776	0.4678	0.4794	438.0	**0.4995**	**0.5025**	**378.6**	0.4869	0.4996	295.6	**0.5184**	**0.5289**	335.4
	BS	1.0000	-	-	-	-	-	-	-	-	-	-	-	-
	WQR	0.2696	0.2643	0.2733	442.8	**0.2722**	**0.2745**	**328.4**	0.2645	0.2702	244.6	**0.2713**	**0.2748**	312.6
	ACT	0.8505	0.8790	0.8799	442.8	0.8729	0.8794	**372.2**	0.8720	0.8729	238.8	**0.8766**	**0.8762**	377.4
	CD	0.8267	0.8283	0.8283	454.8	0.8282	0.8282	**342.0**	0.8282	0.8289	238.2	**0.8288**	0.8288	250.4
	GC	0.7100	0.7460	0.7390	432.2	**0.7570**	**0.7460**	**379.0**	0.7430	0.7410	231.2	**0.7590**	**0.7420**	310.6
Random Forest	AF	0.7677	0.7644	0.7743	425.2	**0.7709**	**0.7787**	473.8	0.7610	0.7690	274.2	**0.7681**	**0.7749**	314.2
	BH	0.5426	0.5483	0.5483	479.2	**0.5549**	**0.5533**	**374.4**	0.5507	0.5491	238.4	**0.5543**	**0.5522**	278.6
	BS	0.9446	0.9628	0.9628	510.0	**0.9873**	**0.9881**	**386.8**	0.9535	0.9543	247.4	**0.9845**	**0.9848**	255.0
	WQR	0.3662	0.3749	0.3738	461.4	**0.3862**	**0.3845**	**362.6**	0.3666	0.3674	253.0	**0.3810**	**0.3810**	283.2
	ACT	0.8808	0.8864	0.8841	475.8	**0.8925**	**0.8921**	**357.6**	0.8874	0.8841	222.4	**0.8893**	**0.8864**	424.0
	CD	0.8293	0.8283	0.8282	497.0	**0.8295**	**0.8294**	**432.2**	0.8291	0.8286	375.2	**0.8295**	0.8276	304.0
	GC	0.7450	0.7630	0.7580	459.2	**0.7640**	**0.7620**	**368.2**	0.7510	0.7490	229.6	**0.7680**	**0.7680**	471.8
Light-GBM	AF	0.8375	0.8304	0.8356	479.6	**0.8311**	**0.8392**	**464.2**	0.8185	0.8266	284.6	**0.8366**	**0.8395**	360.6
	BH	0.5537	0.5503	0.5467	490.8	**0.5619**	**0.5644**	**455.0**	0.5500	0.5609	238.4	**0.5642**	0.5595	345.6
	BS	0.9429	0.9693	0.9691	480.2	**0.9737**	**0.9754**	**414.0**	0.9539	0.9536	312.6	**0.9801**	**0.9813**	**236.8**
	WQR	0.3825	0.4087	0.4151	493.0	**0.4118**	**0.4171**	**322.8**	0.4057	0.4050	246.8	0.4021	0.4042	293.6
	ACT	0.8832	0.8864	0.8883	513.0	**0.8888**	**0.8925**	**367.4**	0.8813	0.8748	229.0	**0.8902**	**0.8925**	359.6
	CD	0.8300	0.8284	0.8292	490.8	**0.8301**	**0.8297**	**440.8**	0.8295	0.8299	218.6	**0.8303**	0.8294	371.2
	GC	0.7250	0.7620	0.7620	482.4	**0.7680**	**0.7720**	**376.6**	0.7550	0.7550	225.0	**0.7760**	**0.7700**	382.2
Mean		0.6806	0.7105	0.7118	469.9	**0.7171**	**0.7185**	**393.9**	0.7078	0.7092	255.9	**0.7187**	**0.7183**	331.9

References

[1] L. Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

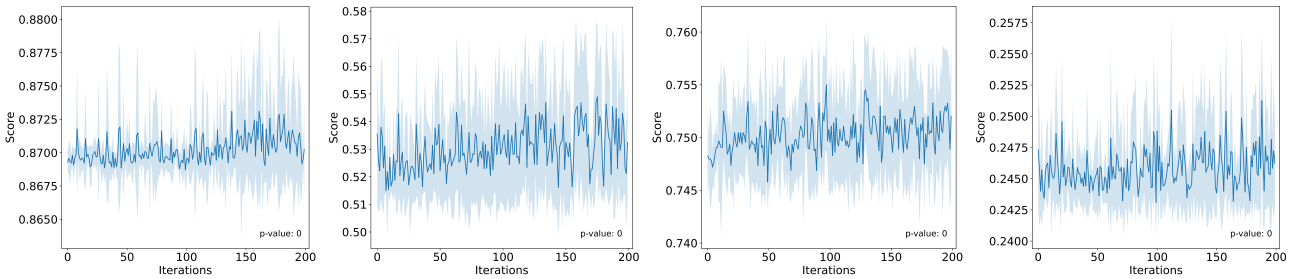
[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[3] S. Galhotra, U. Khurana, O. Hassanzadeh, K. Srinivas, and H. Samulowitz. Kafe: Automated feature enhancement for predictive modeling using external knowledge, 2019. URL https://kr2ml.github.io/2019/papers/KR2ML_2019_paper_17.pdf.

[4] L. Grinsztajn, E. Oyallon, and G. Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL https://openreview.net/forum?id=Fp7__phQszn.

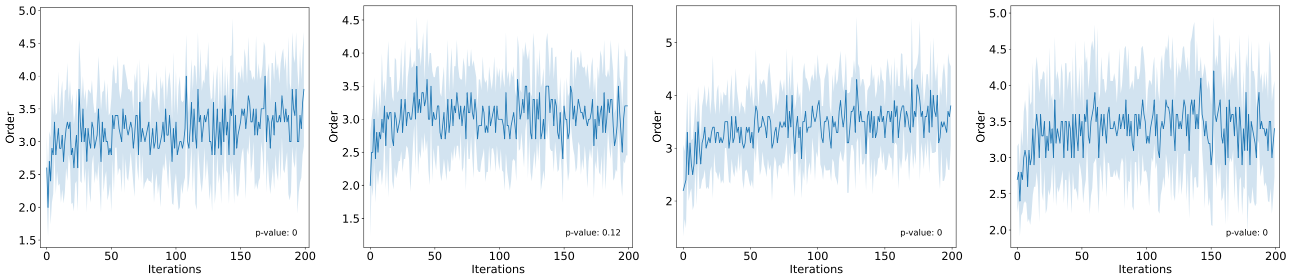
[5] N. Hollmann, S. Müller, and F. Hutter. Llms for semi-automated data science: Introducing caafe for context-aware automated feature engineering, 2023. URL https://arxiv.org/abs/2305.03403.

[6] F. Horn, R. Pack, and M. Rieger. The autofeat python library for automated feature engineering and selection. In *Machine Learning and Knowledge Discovery in Databases*, pages 111–120, Cham, 2020.



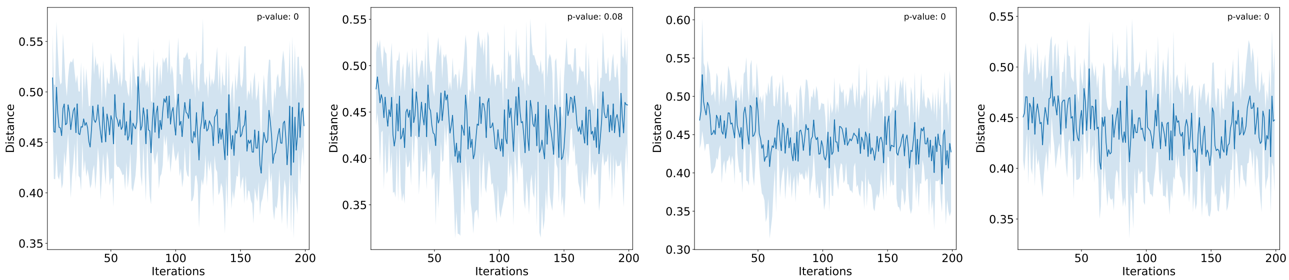
(a) ACT (b) BH (c) GC (d) WQR

Figure 5. The cross validation score on training data across iterations. The shaded regions represent standard deviations. We show the p-value of OLS regression.



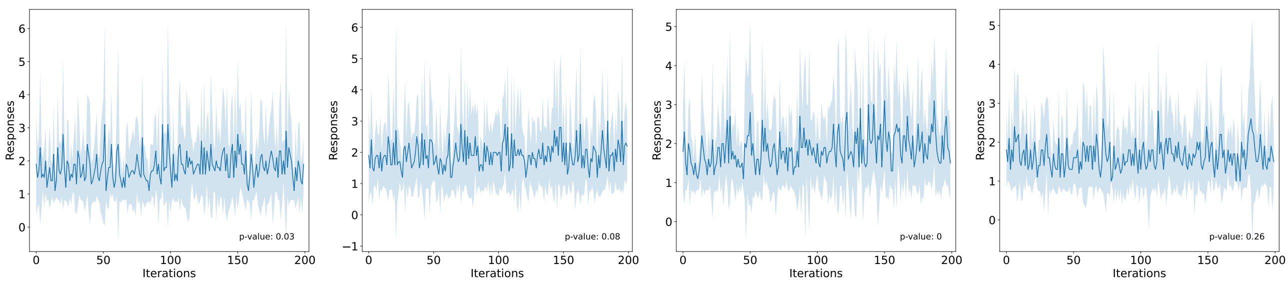
(a) ACT (b) BH (c) GC (d) WQR

Figure 6. The order of candidate features across iterations. The shaded regions represent standard deviations. We show the p-value of OLS regression.



(a) ACT (b) BH (c) GC (d) WQR

Figure 7. The mean normalized tree edit distance between a new candidate feature and previous five features across iterations. The shaded regions represent standard deviations. We show the p-value of OLS regression.



(a) ACT (b) BH (c) GC (d) WQR

Figure 8. The number of LLM responses to construct a new candidate feature across iterations. The shaded regions represent standard deviations. We show the p-value of OLS regression.

Springer International Publishing. ISBN 978-3-030-43823-4.

[7] J. M. Kanter and K. Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE international conference on data science and advanced analytics (DSAA)*, pages 1–10. IEEE, 2015.

[8] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[9] U. Khurana, H. Samulowitz, and D. Turaga. Feature engineering for predictive modeling using reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[10] L. Li, H. Wang, L. Zha, Q. Huang, S. Wu, G. Chen, and J. Zhao. Learning a data-driven policy network for pre-training automated feature engineering. In *The Eleventh International Conference on Learning Representations*, 2023.

[11] K. Man, K. Tang, and S. Kwong. Genetic algorithms: concepts and applications [in engineering design]. *IEEE Transactions on Industrial Electronics*, 43(5):519–534, 1996. doi: 10.1109/41.538609.

[12] F. Nargesian, H. Samulowitz, U. Khurana, E. B. Khalil, and D. S. Turaga. Learning feature engineering for classification. In *Ijcai*, volume 17, pages 2529–2535, 2017.

[13] OpenAI. Gpt-4 technical report, 2023. URL https://cdn.openai.com/papers/gpt-4.pdf.

[14] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners, 2019.

[15] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. Llama: Open and efficient foundation language models, 2023. URL https://arxiv.org/abs/2302.13971.

[16] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[18] K. Wang, P. Wang, and C. Xu. Toward efficient automated feature engineering. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 1625–1637, 2023. doi: 10.1109/ICDE55515.2023.00128.

[19] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen. Large language models as optimizers, 2023. URL https://arxiv.org/abs/2309.03409.

[20] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18:1245–1262, 12 1989. doi: 10.1137/0218082.

[21] T. Zhang, Z. Zhang, Z. Fan, H. Luo, F. Liu, Q. Liu, W. Cao, and J. Li. Openfe: automated feature generation with expert-level performance. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23, 2023.

[22] G. Zhu, S. Jiang, X. Guo, C. Yuan, and Y. Huang. Evolutionary automated feature engineering. In *Pacific Rim International Conference on Artificial Intelligence*, pages 574–586. Springer, 2022.

[23] G. Zhu, Z. Xu, C. Yuan, and Y. Huang. Difer: Differentiable automated feature engineering. In *Proceedings of the First International Conference on Automated Machine Learning*, volume 188 of *Proceedings of Machine Learning Research*, pages 17/1–17. PMLR, 25–27 Jul 2022. URL https://proceedings.mlr.press/v188/zhu22a.html.

[24] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.