EaSFE: Scalable and Efficient Feature Engineering for Boosting Machine Learning Performance

JIAN CHEN*, YILE CHEN*, and ZHENYA ZHENG, South China University of Technology, China
ZEYI WEN, The Hong Kong University of Science and Technology (Guangzhou), China
YAWEN CHEN†, School of Artificial Intelligence, Henan University, China
JIN HUANG, South China Normal University, China

Feature engineering plays a critical role in machine learning (ML), but existing methods often struggle with high computational cost and limited scalability when applied to large-scale and sparse datasets. In this paper, we propose EaSFE, an efficient and scalable feature engineering framework that unifies feature generation, filtering, and evaluation in an end-to-end manner. EaSFE is designed to efficiently construct and select informative features while explicitly considering computational and memory constraints. To achieve scalability, EaSFE incorporates parallel and distributed execution mechanisms, as well as a chunk-based data processing strategy that enables memory-efficient feature engineering on large datasets. In addition, EaSFE adopts tailored storage and execution strategies to handle high-dimensional sparse data effectively. Extensive experiments on multiple real-world datasets demonstrate that EaSFE consistently improves predictive performance (e.g., 5% accuracy improvement in *poker*) while substantially enhancing efficiency (i.e., over 10x speedup) compared to existing feature engineering methods. In addition, EaSFE is demonstrated to scale to large and sparse datasets, successfully handling datasets with over 119 million training instances and 54 million features.

CCS Concepts: • **Computing methodologies** → **Parallel algorithms**; **Distributed algorithms**; • **Information systems** → **Data mining**.

Additional Key Words and Phrases: Feature engineering, Parallel optimization, Distributed process, Large dataset, Sparse dataset.

1 Introduction

Feature engineering plays an important role in machine learning (ML), helping discover representative features and enhance the predictive accuracy [19, 21, 41]. Meanwhile, feature engineering also benefits deep learning by reducing computational cost and memory usage, while improving interpretability [26, 39]. Figure 1 illustrates the basic process of feature engineering, which involves generating and selecting new features from the original features of a dataset for subsequent training.

Despite substantial progress in feature engineering, existing methods face two major challenges when applied to large datasets: high computational cost and limited scalability. First, feature engineering is computationally

*Both authors contributed equally to this research.
†Corresponding author.

Authors' Contact Information: Jian Chen, ellachen@scut.edu.cn; Yile Chen, jireh.x6@gmail.com, jireh.x6@gmail.com; Zhenya Zheng, zhenya728772024@gmail.com, South China University of Technology, Guangzhou, China; Zeyi Wen, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China, wenzeyi@hkust-gz.edu.cn; Yawen Chen, School of Artificial Intelligence, Henan University, Zhengzhou, China, ywchen@henu.edu.cn; Jin Huang, South China Normal University, Guangzhou, China, huangjin@m.scnu.edu.cn.

ACM 1556-472X/2026/5-ART
https://doi.org/10.1145/3816023

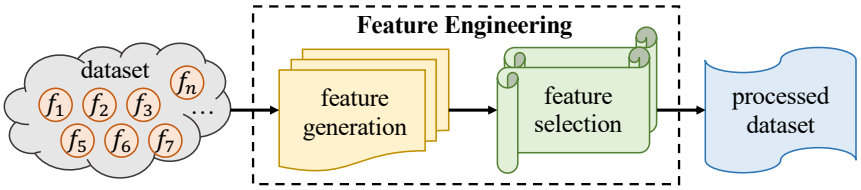


Fig. 1. Key stages in the basic process of feature engineering

expensive [13, 32]. Evaluating a large number of generated features to identify the effective ones demands significant computation and validation, often resulting in considerable time expenditure. Second, the memory overhead associated with managing and processing high-dimensional feature spaces severely limits the scalability of these methods [26]. As a result, conventional approaches struggle to operate efficiently on large datasets, restricting their practical applicability.

To overcome these limitations, we propose **E**fficient **a**nd **S**calable **F**eature **E**ngineering (i.e., EaSFE), an efficient and scalable framework that automates the process of generating and selecting features. EaSFE integrates multiple feature engineering techniques (e.g., aggregation and transformation) to generate a diverse pool of new features. Then these generated features are subsequently filtered using a well-designed evaluation metric to retain only the most relevant ones. To further achieve efficiency and scalability, EaSFE adopts several system-level design choices. (i) Multi-threaded parallelism is employed to accelerate the feature engineering process. (ii) Efficient data loading strategies are introduced to support large-scale datasets, including chunk-based processing and distributed cluster execution. (iii) Special storage strategy is employed in EaSFE to support sparse data processing. Meanwhile, EaSFE also supports the chunking and distributed feature engineering for sparse datasets. These optimizations allow EaSFE to effectively manage large datasets that exceed the memory limits of a single machine, making it suitable for large-scale and sparse feature engineering tasks [17, 41]. Experimental evaluations across various datasets demonstrate that EaSFE outperforms baseline methods in terms of both computational efficiency and predictive accuracy. Moreover, experiments on large-scale datasets further illustrate the scalability of EaSFE. To summarize, our major contributions are listed below.

- We develop EaSFE, which can efficiently complete the feature engineering process by leveraging interaction information among existing features and automatically filtering suitable features.
- We introduce various optimization techniques to enhance the efficiency and scalability of EaSFE, including multi-threaded parallelism, two accelerated strategies (i.e., chunking loading strategy and distributed cluster processing strategy), and a special storage strategy for sparse datasets.
- We experimentally demonstrate that the EaSFE method not only outperforms other approaches in terms of effectiveness (e.g., 5% accuracy improvement) but also achieves high efficiency (i.e., 10x faster than the baselines). Furthermore, EaSFE demonstrates the ability to handle sparse datasets with over 119 million training instances and 54 million features.

2 Background and Related Work

Feature engineering is the process of transforming raw data into new features and selecting the relevant ones to enhance model performance [2, 3]. We summarize the main notations used in this paper as follows: $\mathcal{D}_{tr}$ and $\mathcal{D}_{val}$ denote the training and validation datasets, $\mathcal{F}$ denotes the original feature set, $\mathcal{S}$ represents feature construction operators, and $\mathcal{L}(\cdot)$ denotes the validation loss. Given a training set $\mathcal{D}_{tr}$, validation set $\mathcal{D}_{val}$, original features $\mathcal{F}$, and a set of construction methods $\mathcal{S}$ that generate new features $\mathcal{F}'$, the goal of feature engineering is to find an

optimal subset of generated features that minimizes the validation loss as:

$$\mathcal{F}^* = \arg\min_{\mathcal{F}'' \subseteq \mathcal{F}'} \mathcal{L}(\mathcal{D}_{tr}, \mathcal{D}_{val}, \mathcal{F} \cup \mathcal{F}''), \tag{1}$$

where $\mathcal{F}^*$ denotes the optimal subset of generated features selected from $\mathcal{F}'$, and $\mathcal{L}(\mathcal{D}_{tr}, \mathcal{D}_{val}, \mathcal{F} \cup \mathcal{F}^*)$ represents the model performance when trained on $\mathcal{D}_{tr}$ using the combined feature set of original features and the selected generated features. Feature engineering consists of two main components: feature generation and feature selection.

- **Feature generation**: Early works of feature generation focused on using simple operators to generate new features. For instance, TFC [27] uses an iterative framework to generate and rank feature combinations by information gain, while FCTree [7] adopts a similar approach using decision trees. However, both of them struggle to produce complex, higher-order features, limiting their performance. Recently, some works have offered more advanced strategies. DSM [19] exploits relational data via aggregation but is less effective on a single table. ExploreKit [21] ranks features using external ML models but is computationally expensive. SAFE [31] introduces a scalable, performance-driven search, and Autofeat [15] creates a large nonlinear feature pool, then selects an interpretable subset to enhance linear models. In addition, many researchers have explored more in-depth feature engineering techniques in specific domains, such as time series analysis [34], healthcare models [20], and DNA sequences [36].
- **Feature selection**: Feature selection, a key component of feature engineering, helps reduce dimensionality, eliminate redundancy, and improve model efficiency. Beyond the whole process of feature engineering, many studies have focused specifically on feature selection. Guyon et al. [12] introduced various basic selection methods, including filtering, packing, and embedding. Kira and Rendell [22] proposed a selection method called "relevance" to identify the most relevant features for classification tasks. In addition, Yu and Liu [38] introduced a widely used MRMR approach based on information theory. These methods are considered classical approaches and have served as the foundation for subsequent improvements. More recent advances include graph-based methods like Infinite Feature Selection (IFS) [29], and hybrid techniques combining information gain with genetic algorithms for text data [30]. Additionally, various approaches leveraging evolutionary algorithms and SVMs have been proposed [8, 16].
- **Parallel and distributed feature engineering**: Reducing the high computational cost has become a key research focus in feature engineering, with parallel and distributed computing emerging as a prominent solution [33, 37]. Parallel computing accelerates tasks by processing multiple operations simultaneously [4, 6, 9, 24], while distributed computing scales this further by spreading workloads across multiple nodes, enabling efficient handling of large datasets [5, 11, 14, 18]. These approaches significantly enhance the efficiency and scalability of feature extraction and selection. However, many existing approaches focus on isolated components of feature generation or selection, and often lack an end-to-end system design that jointly considers efficiency, scalability, and resource constraints. More recent data processing frameworks, such as Spark- or Dask-based pipelines [28, 40], also provide generic support for parallel feature processing, though they are not specifically designed for automated feature engineering.

Despite numerous proposed methods, the application of current feature engineering on large datasets still faces two main challenges. First, feature engineering is computationally expensive [13, 32], both in generating new features through complex operations and in selecting useful features from a vast candidate pool. Second, when dealing with large-scale tasks, fitting the entire dataset into memory during feature engineering becomes a formidable challenge [26]. Therefore, improving the efficiency and scalability of feature engineering is crucial for its application on large datasets.

3 Basic Framework of EaSFE

To address these challenges, we propose EaSFE, an efficient and scalable framework for automated feature engineering, utilizing a combination of carefully designed techniques. In this section, we introduce the basic framework of feature engineering in EaSFE, as shown in Figure 2. First, EaSFE employs three types of methods to generate a new feature set $\mathcal{F}'$ (i.e., Step a). Then, the generated features undergo further feature selection to produce the final feature set $\mathcal{F}^*$ (i.e., Step b). In the following section, we delve into the details of each component.

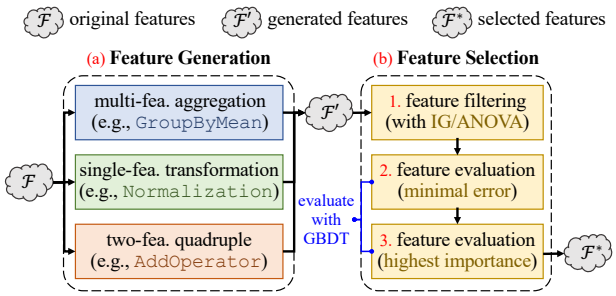


Fig. 2. Framework of the basic EaSFE including diverse feature generation methods (§3.1) and multi-step feature selection (§3.2).

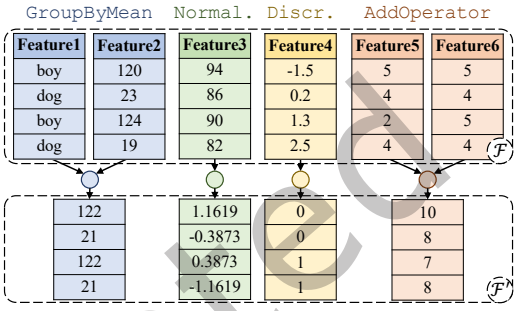


Fig. 3. Examples of 4 generation operators, including: GroupByMean, Normalization, Discretizer, and AddOperator.

3.1 Feature Generation

EaSFE first generates new features using various generation methods. Specifically, EaSFE focuses on two types of features (i.e., categorical and numerical) and employs three types of methods:

- **Multi-feature aggregation**: This method combines categorical and numerical features to generate new ones, using six operations (i.e., GroupByMean, GroupByCount, GroupByMax, GroupByMin, GroupByStd, and GroupByRank), which perform category-based aggregation on numerical features. As the example in Figure 3, GroupByMean groups records by a categorical feature ("Feature1") and computes the mean of a numerical feature ("Feature2") within each group. Similarly, the other methods perform count, maximum, minimum, standard deviation (std), and rank operations on the numeric feature.
- **Single-feature transformation**: The single-feature transformation creates informative new features from individual original features using two operations: Normalization and Discretizer. (1) Normalization: transforms the numerical type feature into a standard normal distribution. (2) Discretizer: divides the numerical feature into a number of bins or intervals.
- **Two-feature quadruple operation**: The two-feature quadruple method creates new features by applying basic arithmetic operations: addition, subtraction, multiplication, and division. In Figure 3, AddOperator adds "Feature5" and "Feature6" to generate the values of the new feature, while the other three methods follow a similar procedure with different arithmetic operations.

EaSFE generates a new feature set $\mathcal{F}'$ with these methods. Additionally, EaSFE can flexibly incorporate extended feature generation methods based on user requirements.

3.2 Feature Selection

After obtaining the generated feature set $\mathcal{F}'$, EaSFE further refines it through feature selection (i.e., Step b in Figure 2) to reduce redundancy and improve quality. Specifically, EaSFE first performs a quick filtering using

information gain (IG) or analysis of variance (ANOVA). After that, EaSFE evaluates the filtered features via an analysis with a GBDT model training. Finally, another GBDT model is trained on the original and retained features, and the new features with the highest importance scores are selected.

Feature Filtering: EaSFE first performs filtering on the generated feature set $\mathcal{F}'$ based on the evaluation scores of each candidate feature X, where $X \in \mathcal{F}'$ denotes a generated feature, Y denotes the target variable, and (x_i, y_j) represent their observed values.

- **IG for classification tasks.** Information gain (IG) measures how much a feature improves data partitioning and is widely used in decision trees for split selection. EaSFE applies IG to quickly filter the generated features. The IG is calculated as follows,

$$IG(Y, X) = E(Y) - E(Y|X), \tag{2}$$

where $E(Y)$ is the entropy of the dataset $\mathcal{D}_{tr}$ before the partitioning, while $E(Y|X)$ denotes the weighted entropy after partitioning using feature X. A higher score indicates that the feature X provides substantial information about the target category, improving the classification performance. Equation 3 provides the calculation method for $E(Y)$,

$$E(Y) = -\sum_{j=1}^{n_y} P(y_j) \log_2 P(y_j), \tag{3}$$

where $P(y_j)$ is the probability of each class y_j in the target category Y, and n_y is the number of distinct classes in Y. Meanwhile, the calculation of $E(Y|X)$ is shown as,

$$\begin{aligned} E(Y|X) &= \sum_{i=1}^{n_x} P(X) E(Y|X) \\ &= -\sum_{i=1}^{n_x} P(x_i) \sum_{j=1}^{n_y} P(y_j|x_i) \log_2 P(y_j|x_i), \end{aligned} \tag{4}$$

where $P(x_i)$ is the probability of each value x_i of the feature X, and $P(y_j|x_i)$ is the conditional probability of y_j given x_i. This computes the weighted entropy of Y across all values of X.

- **ANOVA for regression tasks.** For regression tasks, ANOVA evaluates how feature X influences target Y by dividing data into n_x groups based on X's values. It computes two key measures: the between-group variance $\text{Var}_{\text{between}}$ and the within-group variance $\text{Var}_{\text{within}}$. First, $\text{Var}_{\text{between}}$ captures the extent to which the means of Y differ across groups formed by X, defined as,

$$\text{Var}_{\text{between}} = \frac{1}{n_x} \sum_{i=1}^{n_x} n_i (\bar{y}_i - \bar{y})^2 \tag{5}$$

where $\bar{y}_i$ represents the mean of Y for the i-th group, $\bar{y}$ is the overall mean of Y, and n_i is the number of instances whose feature X is x_i. Then, the within-group variance measures the variability of Y within each group, expressed as,

$$\text{Var}_{\text{within}} = \frac{1}{n - n_x} \sum_{i=1}^{n_x} \sum_{j=1}^{n_i} (y_{i,j} - \bar{y}_i)^2 \tag{6}$$

where $y_{i,j}$ is the individual value of Y in the i-th group and n is the total number of instances in dataset. A much higher $\text{Var}_{\text{between}}$ than $\text{Var}_{\text{within}}$ indicates that feature X creates significant group differences, suggesting a strong correlation with target Y.

Feature Evaluation: After obtaining the filtered features $\mathcal{F}''$, EaSFE conducts a two-step feature evaluation to further select a high-quality feature set, as shown in Algorithm 1.

Algorithm 1: Feature Evaluation Process in EaSFE

Input: $\mathcal{D}$: original dataset, $\mathcal{F}''$: filtered feature set, F: max number of saving features, F'': max number of saving features after feature evaluation.

1 Errors $\leftarrow \emptyset$, Ims $\leftarrow \emptyset$;
/* feature evaluation with training errors */
2 $M^{\text{orig}} \leftarrow$ **trainGBDT**$(\mathcal{D}, \mathcal{F}'')$;
3 **for** X_j ***in*** $\mathcal{F}''$ **do**
4 $M_j \leftarrow$ **trainGBDT**$(\mathcal{D}, X_j)$;
5 Errors $\leftarrow$ Errors + **calError**$(\mathcal{D}, M^{\text{orig}}, M_j)$; // Equation 7
6 $\mathcal{F}^* \leftarrow$ **topK**$(\mathcal{F}'', \text{Errors}, F'')$;
/* feature evaluation with feature importance */
7 $M \leftarrow$ **trainGBDT**$(\mathcal{D}, \mathcal{F}^*)$;
8 **for** X_j ***in*** $\mathcal{F}^*$ **do**
9 Ims $\leftarrow$ Ims + **calImportance**$(\mathcal{D}, M, X_j)$; // Equation 8
10 $\mathcal{F}^* \leftarrow$ **topK**$(\mathcal{F}^*, \text{Ims})$;

Output: $\mathcal{F}^*$: selected new features.

(1) **Evaluation with training errors**: EaSFE utilizes a GBDT training strategy to analyze each feature's quality (Lines 2-6). First, EaSFE uses the full feature set to train a GBDT model M^{orig} (Line 2). After that, for each new candidate feature $X_j \in \mathcal{F}''$, a separate GBDT model M_j is trained using only the feature X_j and the target variable Y (Lines 3-4). Then, EaSFE calculates an evaluation score for each feature (Line 5), as:

$$\text{Error}_j = \frac{1}{n}\sum_{i=1}^{n}\left(M_j(x_{i,j}) - \hat{y}_i^{\text{orig}}\right)^2 \tag{7}$$

where $x_{i,j}$ is the value of the feature X_j for the i-th instance, $M_j(\cdot)$ is the prediction of M_j, and $\hat{y}_i^{\text{orig}}$ is the prediction result from the GBDT M^{orig}. Then, the candidate features are ranked based on the errors Error_j, and the top k features with the smallest errors are selected, as $\{X_1, X_2, \ldots, X_k\}$ (Line 6).

(2) **Evaluation with feature importance**: After the evaluation, the selected k features are then used to train a new GBDT model M (Line 7). The feature importance scores $I(X_j)$ are calculated based on their contributions to the model's predictions, typically using metrics such as information gain or the number of splits (Lines 8-9). The final set of features is determined by selecting those with the highest importance scores:

$$I(X_j) = \text{Importance}(X_j) \quad \text{for} \quad j = 1, 2, \ldots, k. \tag{8}$$

The features are then ranked by $I(X_j)$, and the final subset is chosen based on these importance scores (Line 10).

4 Efficient and Scalable EaSFE

The basic EaSFE framework provides an end-to-end feature engineering pipeline. To further improve efficiency and scalability, we enhance EaSFE with several system-level optimizations. As in Figure 4, EaSFE adopts fine-grained parallelism to accelerate feature generation and evaluation. In addition, it incorporates heap-based selection and chunk-based data processing to reduce memory and computational overhead. Moreover, EaSFE devises a distributed cluster processing method and a specific storage strategy to handle large-scale and sparse datasets. The following sections detail each of these enhancements.

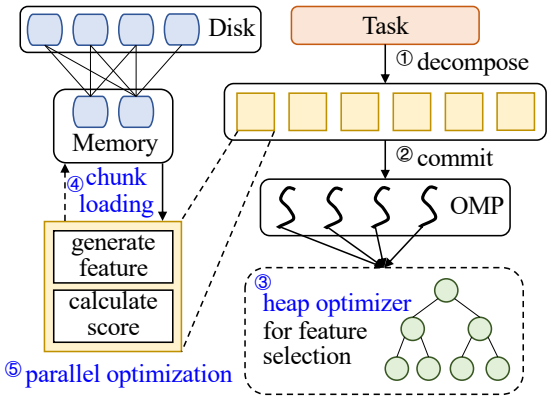


Fig. 4. The parallel optimization design for EaSFE, where OMP means we utilize OpenMP parallelism.

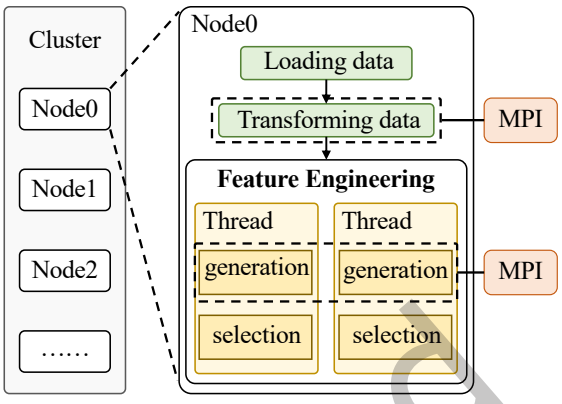


Fig. 5. Design for the large dataset with distributed cluster processing, where MPI is for communication.

4.1 Parallelization and Heap Optimization

To accelerate feature engineering, EaSFE decomposes the process into elemental components and applies fine-grained parallelization. Specifically, each new feature $X_j \in \mathcal{F}' = \{X_1, X_2, \ldots, X_m\}$, where $m = |\mathcal{F}'|$, is treated as the smallest unit of parallel computation, allowing simultaneous execution across multiple processing units. This strategy maximizes resource utilization and significantly reduces runtime. The feature generation process can be expressed as:

$$\mathcal{F}' = \bigcup_{j=1}^{m} \mathcal{G}(X_j) \tag{9}$$

where $\mathcal{G}(X_j)$ represents the generation function for the original feature X_j.

Moreover, we further apply parallelism within individual feature generation methods. New features are derived through transformations or aggregations of original features, such as computing statistical measures. For instance, the mean μ_j and variance σ_j^2 of feature X_j can be parallelized as:

$$\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_{i,j}, \quad \sigma_j^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_{i,j} - \mu_j)^2 \tag{10}$$

where $x_{i,j}$ is the i-th observation of feature X_j and n is the total number of observations. These operations are naturally parallelizable, allowing efficient computation across different features. Additionally, in alignment with the parallelization of feature generation, we integrate feature selection by calculating IG or ANOVA immediately after the generation of each new feature. This integration enables simultaneous feature generation and filtering, ensuring efficient, bottleneck-free execution.

To further enhance efficiency and minimize memory overhead during feature selection, EaSFE uses a heap optimization to keep only the top-k features based on the computed scores (i.e., IG or ANOVA). Unlike traditional methods that compute and store scores for all n features—leading to high memory costs—this heap-based approach dynamically tracks just the best k features ($k \ll n$), cutting memory overhead while preserving selection quality.

4.2 Dataset Chunk Loading

To address the challenge of feature engineering for large-scale datasets that exceed the memory capacity of a single machine, we adopt a dataset chunk loading with multi-buffer processing.

- **Chunk loading:** In EaSFE, the dataset is pre-partitioned by rows into manageable, equally sized chunks that align with the machine's memory constraints. This partitioning strategy balances the workload across processing units and optimizes memory usage during feature engineering. To facilitate efficient data handling, we employ a chunked read-in mechanism that leverages the contiguous in-memory layout of each feature. This structure enables fast binary serialization and deserialization, allowing for rapid, low-overhead data transfers between memory and disk. Consequently, feature data can be efficiently read and written, ensuring smooth memory-disk interaction.
- **Multi-buffer process:** To reduce synchronization overhead in parallel chunk processing, EaSFE introduces a multi-buffer design. Parallel processing of chunked datasets introduces synchronization challenges, as multiple threads often need concurrent access to different data blocks. When only one block resides in memory, threads may compete for access, leading to contention and increased synchronization overhead. To mitigate this, we propose a multi-buffer design, where each buffer holds a distinct data chunk. This allows threads to access separate blocks simultaneously, reducing contention, synchronization overhead, and idle time. Furthermore, the buffer size is configured to respect the memory constraints of the system.

4.3 Distributed Cluster Processing

To support large-scale feature engineering under limited resources, EaSFE adopts a distributed processing approach. The pre-partitioned dataset is divided into chunks (i.e., chunk loading) and distributed across computing nodes in a cluster. Inter-node communication is handled via the Message Passing Interface (MPI), which provides a lightweight and explicit communication model for distributed execution. Compared with higher-level data analytics frameworks such as Spark or Dask, MPI offers finer-grained control over communication and memory management, which aligns well with the resource-aware and system-level optimizations adopted in EaSFE. This choice relies on a static resource allocation model and therefore provides limited runtime elasticity. When dynamic scaling is required, the design of EaSFE does not fundamentally depend on MPI and can be adapted to alternative parallel or scheduling frameworks. Figure 5 illustrates the structure of the distributed processing, highlighting two key operations: "Transforming data" and "Feature generation".

- **Transforming data:** This operation converts the dataset into a format compatible with machine learning algorithms, such as mapping categorical values (e.g., "cat", "dog", "pig") to numerical identifiers. This transformation requires coordinated access to feature values across nodes. To manage memory usage, EaSFE uses MPI to gather data in a feature-wise manner on a designated coordinator node, rather than collecting all features at once. This staged collection reduces peak memory usage and improves scalability.
- **Feature generation:** Feature generation requires access to original feature data across distributed chunks. EaSFE coordinates this process by aggregating the required data on a designated coordinator node, generating new features, and distributing the results back to other nodes. Multi-threading is used to accelerate computation. To ensure correctness during concurrent communication, EaSFE enforces serialized access to MPI communication routines. In addition, each feature is associated with a unique identifier to ensure consistent data exchange across nodes during distributed execution.

4.4 EaSFE for Sparse Datasets

To enhance the generality of EaSFE, we extend support for sparse datasets by adopting a storage strategy that retains only non-zero feature values and their corresponding row indices. This format enables efficient access through simple indexing, with contiguous storage further improving retrieval speed. When a feature is needed, EaSFE dynamically converts its sparse representation to a dense format, releasing it once processing is complete. This approach conserves memory and maintains high system performance.

To further enhance the efficiency, EaSFE assigns a counter to each feature to manage safe multi-threaded access. Threads decrement the counter upon completing their tasks, and the last thread deallocates the feature's resources when the counter reaches zero. This mechanism enables efficient feature reuse, reduces redundant sparse-dense conversions, and ensures memory usage remains within system limits. In addition, we further extend EaSFE to handle large-scale sparse datasets that exceed memory capacity. As detailed in Sections 4.2 and 4.3, our chunk loading and distributed processing strategies are also applied to sparse data, leveraging its sparsity to optimize memory use and computational efficiency.

5 Overview and Complexity Analysis

In this section, we provide an algorithmic overview and complexity analysis of EaSFE. We first describe the overall workflow with its pseudocode, and then analyze its time and space complexity to illustrate the computational cost and scalability characteristics of the proposed framework.

5.1 Overview of EaSFE

We first introduce the key notations and then present the overall procedure using pseudocode.

- Dataset $\mathcal{D} = (d_{i,j}) \in \mathbb{R}^{m \times n}$, where m denotes the number of instances and n denotes the number of features.
- Constructor set $\mathcal{S} = \{S_1, ..., S_K\}$: with K constructors, each generating new features by transforming or combining original ones.
- Set of feature subsets $\hat{\mathcal{F}}$: the collection of candidate q-feature subsets sampled from the original feature set. For notational convenience, $\hat{\mathcal{F}} = \{\hat{\mathcal{F}}_1, \hat{\mathcal{F}}_2, \dots, \hat{\mathcal{F}}_T\}$, where $T \leq \binom{n}{q}$.
- New feature f_t^k: $f_t^k = FC(\hat{\mathcal{F}}_t, S_k)$, where $FC(\cdot)$ indicates that a new feature can be generated using the k-th construction method in $\mathcal{S}$ and the t-th sub-feature set in $\hat{\mathcal{F}}$.

Alg. 2 outlines the complete automatic feature generation and selection process of EaSFE. The algorithm begins by initializing $\mathcal{F}'$ and f_{counts}, where $\mathcal{F}'$ represents the selected feature in the feature selection process and is initially set to null and f_{counts} represents the number of features generated. In the feature generation phase (Lines 2-8), EaSFE first constructs a set $\hat{\mathcal{F}}$ which contains all the possible combinations of selecting q features from dataset $\mathcal{D}$ (Line 4). Thus, $\hat{\mathcal{F}}$ contains $\binom{n}{q}$ different combinations, where n is the feature number of $\mathcal{D}$. On each combination of q features, EaSFE iterates all the construction methods in $\mathcal{S}$ to generate new features (Lines 6-8). With the new features generated, EaSFE proceeds to the feature selection phase (Lines 9-12). This process iterates over all candidate features and selects the F features with the highest score(IG or ANOVA). Besides, we also set an early stop if the number of features in the dataset is particularly large (Line 13). Finally, the features selected are added to the original dataset (Line 14).

5.2 Complexity Analysis

We analyze the time and space complexity of EaSFE by considering the dominant operations in feature generation and selection. Consider a dataset $\mathcal{D}$ with m instances and n features, including n_1 numerical and n_2 categorical features. Let F denote the maximum number of retained features after selection, and Q the maximum number of original features involved in feature construction.

Algorithm 2: Our Proposed EaSFE

Input: $\mathcal{D}$: original dataset, $\mathcal{S}$: set of construction methods, F, F', F'': max number of saving features, features after filtering, and features after feature evaluation, C: number of threads, σ: max number of features generated, Q: max number of features selected from the original dataset.
Output: $\mathcal{D}'$: dataset with selected new features.

$\mathcal{F}' \leftarrow \emptyset$, $f_{counts} \leftarrow 0$; // initialization.
// Feature generation phase.
for $q \leftarrow 1$ **to** Q **do**
  for $k \leftarrow 1$ **to** $|\mathcal{S}|$ **do**
    $\hat{\mathcal{F}} \leftarrow$ **selectFeaturesfromData**$(q, \mathcal{D})$;
    #pragma omp parallel for num_threads(C**)**;
    for $t \leftarrow 1$ **to** $|\hat{\mathcal{F}}|$ **do**
      if *isMatch*$(S_k, \hat{\mathcal{F}}_t)$ **then**
        $f_t^k \leftarrow FC(\hat{\mathcal{F}}_t, S_k)$;
        // Feature filtering phase.
        $\delta \leftarrow$ **calculateScore**$(f_t^k, \mathcal{D})$;
        if $|\mathcal{F}'| < F'$ **then** $\mathcal{F}' \leftarrow \mathcal{F}' + f_t^k$;
        else $f', \delta' \leftarrow$ minimum score in $\mathcal{F}'$;
          if $\delta > \delta'$ **then** $\mathcal{F}' \leftarrow \mathcal{F}' - f' + f_t^k$;
      if $++f_{counts} > \sigma$ **then** Terminating the search;
// Feature evaluation phase.
$\mathcal{F}^* \leftarrow$ **featuteEvaluation**$(\mathcal{D}, \mathcal{F}', F, F'')$, $\mathcal{D}' \leftarrow \mathcal{D} + \mathcal{F}^*$;
return $\mathcal{D}'$;

- The space complexity of EaSFE is $O(m \cdot (n + F))$, which consists of storing the original dataset and a bounded number of selected features. Notably, EaSFE does not materialize all generated features simultaneously; instead, features are generated, evaluated, and filtered in a streaming manner, which effectively controls memory usage even when the potential feature space is large.
- The time complexity of EaSFE can be expressed as $O\Big(\frac{m}{C} \cdot \min\Big(\sigma,\ n_1 + \binom{n_1}{2} + \sum_{q=2}^{Q} n_1 \cdot \binom{n_2}{q-1}\Big)\Big)$, where the term inside the minimum characterizes the upper bound of candidate feature constructions. Specifically, n_1 corresponds to single-feature transformations on numerical features, $\binom{n_1}{2}$ corresponds to two-feature arithmetic operations between numerical features, and $n_1 \cdot \binom{n_2}{q-1}$ accounts for multi-feature aggregation involving one numerical feature and $q-1$ categorical features. The parameter σ denotes the maximum number of features allowed to be generated in practice, which serves as an explicit budget to prevent combinatorial explosion. The factor $\frac{m}{C}$ reflects that feature evaluation is linear in the number of instances and can be effectively accelerated through multi-threaded parallelism with C threads.

Overall, considering both time and space complexity, this analysis shows that the computational cost of EaSFE is bounded by resource-aware feature generation constraints and parallel execution, enabling scalable feature engineering on large-scale datasets.

6 Experimental Study

In this section, we first introduce the experimental setup, including the dataset, baseline methods, and experimental environment. Then, we compare EaSFE with baseline methods across efficiency, effectiveness, and scalability. Following the overall performance comparisons, we further analyze the impact of key optimization components on large-scale and sparse datasets through dedicated experiments in Sections 6.3 and 6.4.

6.1 Experimental Setup

6.1.1 Datasets description. In the experiments, we evaluated EaSFE on 11 widely adopted benchmark datasets that are commonly used in feature engineering and tabular learning studies [10, 41], as listed in Table 1. These datasets cover a range of data scales and characteristics, enabling a comprehensive evaluation of effectiveness. To further assess EaSFE's performance under large-scale and sparse settings, we additionally included 10 public datasets with substantially larger sizes or higher sparsity, as summarized in Table 2. Specifically, *gerCredit*, *vehicleIT*, and *poker* are obtained from Kaggle[1], while *aloi*, *higgs*, *kdd12*, *log1p*, *yearMSD*, and *avazu* are sourced from the LIBSVM repository[2]. The *bankFraud* dataset is taken from a prior study [17].

Table 1. Dataset properties.

	Dataset	#instance (k)	#feature	#classes	metric
Regression	CA	20.6	8	/	RMSE
	MI	1200	136	/	RMSE
	ME	163	11	/	RMSE
Classification	TE	51	57	2	AUC
	BR	900	58	2	AUC
	DI	102	47	2	AUC
	NO	34.4	118	2	AUC
	VE	98.5	100	2	AUC
	AD	48.8	13	2	AUC
	JA	83.7	54	4	Accuracy
	CO	581	54	7	Accuracy

Table 2. Properties of the 10 additional datasets

Dataset	Regression (sparse)		Classification (dense)				Classification (sparse)			
	log1p	yearMSD	gerCredit	vehicleIT	poker	bankFraud	aloi	higgs	kdd12	avazu
#train instance (k)	16.1	464	1	98.5	1025	4000	108	11000	119705	40429
#test instance (k)	3.3	51.6	/	/	/	/	/	/	29934	4577
#feature	4272227	90	20	100	10	34	128	28	54686452	1000000
#classes	/	/	2	3	10	2	1000	2	2	2
train density (%)	0.1407	100	100	100	100	100	23.9817	92.1057	0.0000201	0.0015
metric	RMSE, R2	RMSE, R2	Acc., F1	Acc., F1	Acc., F1	Acc., F1	Acc., F1	Acc., F1	Acc., F1	Acc., F1

6.1.2 Baseline algorithms. During the experiments, we compared EaSFE with a diverse set of baseline methods that are widely used in feature engineering and tabular learning. Unless otherwise specified, each baseline was implemented using its publicly available library or official implementation, and configured following the settings recommended in the corresponding papers.

- Base: The baseline model used for comparison.
- NN [35]: A deep neural network-based model designed to enhance feature engineering through deep cross networks.

[1]https://www.kaggle.com/datasets
[2]https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

- FCTree [7]: A generalized feature transformation approach utilizing information gain to exclude irrelevant features.
- SAFE [31]: A framework that enhances model robustness by filtering uninformative features using information value.
- AutoFeat [15]: An automated feature engineering tool that generates higher-order polynomial and interaction terms, designed to optimize the feature space for regression models.
- AutoCross [25]: A framework that automates feature crossing and evaluates features based on their impact on linear regression.
- FETCH [23]: A feature engineering pipeline based on a Markov decision process, automating feature generation and selection.
- Featuretools [1]: A robust framework for automating feature engineering, especially effective for time-series and relational data, using aggregation primitives to extract features across tables.
- ExploreKit [21]: An automatic feature generation and selection framework written in Java, employing an iterative approach to generate superior features, albeit with increased execution time.
- OpenFE [41]: An advanced automated feature generation tool that incorporates FeatureBoost and a two-stage pruning algorithm to optimize the feature engineering process.

6.1.3 Implementation details. EaSFE is implemented in C++ and the code is available at [URL omitted due to anonymous reviews], and all experiments related to the feature engineering process are conducted on a Linux operating system equipped with 64 GB of main memory and an Intel(R) Xeon(R) CPU E5-2678 v3 @ 2.50GHz 12-core processor. All reported results are averaged over three independent runs, with detailed configurations provided in the corresponding sections.

6.2 Comparison with Existing Methods

In the experiment, we evaluated the effectiveness and efficiency of EaSFE and all baselines on the benchmark datasets, with results shown in Table 3 and Table 4. We further compared EaSFE against two prominent methods on additional datasets: Featools [19], widely adopted in practice, and ExploreKit [21], noted for its strong ability to boost model performance. The results are presented in Table 6 and Table 7. Since Expkit iteratively generates features, each requiring an evaluator to select the top features, it faces challenges with large-scale datasets like *higgs*, often taking an entire day to complete one iteration. For a fair comparison, we only selected candidate features from Expkit's first iteration and applied the feature selection method from Section 3.2. The same procedure is applied to Featools. Notably, EaSFE used the same construction method as Expkit, yielding identical results post-feature engineering. Since Expkit only supports feature engineering on the training set, we label *log1p* and *yearMSD* as "No" in Table 7 and Figure 6a.

6.2.1 Effectiveness Comparison. As shown in Table 3 and Table 6, EaSFE consistently achieves top performance, notably boosting accuracy by over 5% on the *poker* dataset, highlighting its strong effectiveness in feature engineering.

During the experiment, each method generated up to 2,000 features, retaining half the original feature count—except for *kdd12* and *log1p*, where the feature limit was set to 50 due to high dimensionality. Datasets like *avazu* and *bankFraud* are excluded here and discussed in Section 6.4 due to their chunked format. Meanwhile, we performed resource-constrained hyperparameter optimization (HPO) using Hyperopt[3], with a 4-hour time limit, 2,000 iterations, and decision trees as the evaluation model. For datasets without test sets, 5-fold cross-validation was used. For large datasets (*higgs*, *kdd12*), we sampled 10,000 instances for HPO, maintaining original train/test ratios when applicable. The hyperparameter settings were as follows: "max_depth" ranged from 10 to 1000,

[3]https://github.com/hyperopt/hyperopt

Table 3. Experimental results on baseline datasets with all baselines ("OOT" denotes out-of-time).

Method	RMSE			AUC (%)						Accuracy (%)	
	CA	MI	ME	TE	BR	DI	NO	VE	AD	JA	CO
Base	$0.432_{\pm 0.004}$	$0.744_{\pm 0.0}$	$1128.4_{\pm 1.6}$	$67.1_{\pm 0.2}$	$75.6_{\pm 0.4}$	$73.1_{\pm 0.1}$	$99.6_{\pm 0.0}$	$92.5_{\pm 0.0}$	$92.6_{\pm 0.0}$	$72.1_{\pm 0.2}$	$96.9_{\pm 0.1}$
NN	$0.479_{\pm 0.001}$	$0.750_{\pm 0.0}$	$1413.7_{\pm 2.4}$	$66.1_{\pm 0.2}$	$74.8_{\pm 0.4}$	$71.7_{\pm 0.1}$	$99.2_{\pm 0.0}$	$92.4_{\pm 0.1}$	$92.5_{\pm 0.0}$	$72.0_{\pm 0.1}$	$96.6_{\pm 0.1}$
FCTree	$0.432_{\pm 0.003}$	$0.744_{\pm 0.0}$	$1088.9_{\pm 1.1}$	$67.0_{\pm 0.1}$	$75.0_{\pm 0.4}$	$73.1_{\pm 0.1}$	$99.6_{\pm 0.0}$	$92.6_{\pm 0.0}$	$92.7_{\pm 0.1}$	$71.9_{\pm 0.1}$	$97.1_{\pm 0.0}$
SAFE	×	×	×	$67.3_{\pm 0.1}$	$75.0_{\pm 0.4}$	$73.0_{\pm 0.2}$	$99.6_{\pm 0.0}$	$92.5_{\pm 0.1}$	$92.7_{\pm 0.0}$	×	×
AutoFeat	$0.444_{\pm 0.002}$	$0.744_{\pm 0.0}$	$1172.2_{\pm 2.2}$	$67.2_{\pm 0.2}$	$75.0_{\pm 0.5}$	$73.2_{\pm 0.1}$	$99.6_{\pm 0.0}$	$92.5_{\pm 0.0}$	$91.8_{\pm 0.0}$	$72.1_{\pm 0.1}$	$96.8_{\pm 0.1}$
AutoCross	×	×	×	$65.1_{\pm 0.2}$	$76.5_{\pm 0.1}$	$73.2_{\pm 0.1}$	$99.3_{\pm 0.0}$	$92.1_{\pm 0.0}$	$92.6_{\pm 0.0}$	×	×
FETCH	$0.430_{\pm 0.002}$	OOT	$1130.6_{\pm 1.0}$	$67.3_{\pm 0.2}$	OOT	$73.1_{\pm 0.2}$	$99.6_{\pm 0.0}$	$92.7_{\pm 0.0}$	$92.7_{\pm 0.0}$	$72.0_{\pm 0.1}$	OOT
ExpKit	×	×	×	$64.3_{\pm 0.2}$	$75.1_{\pm 0.2}$	$73.0_{\pm 0.1}$	$97.9_{\pm 0.0}$	$90.8_{\pm 0.0}$	$91.5_{\pm 0.2}$	×	×
Featools	$0.453_{\pm 0.002}$	$0.743_{\pm 0.0}$	$1096.3_{\pm 2.3}$	$66.8_{\pm 0.2}$	$75.2_{\pm 0.3}$	$73.0_{\pm 0.1}$	$99.5_{\pm 0.0}$	$92.5_{\pm 0.0}$	$92.5_{\pm 0.2}$	$71.8_{\pm 0.1}$	$96.1_{\pm 0.1}$
OpenFE	$\mathbf{0.421}_{\pm 0.002}$	$0.738_{\pm 0.0}$	$\mathbf{982.0}_{\pm 1.7}$	$68.0_{\pm 0.2}$	$\mathbf{78.6}_{\pm 0.2}$	$88.8_{\pm 0.2}$	$\mathbf{99.7}_{\pm 0.0}$	$\mathbf{92.8}_{\pm 0.0}$	$\mathbf{92.9}_{\pm 0.0}$	$72.9_{\pm 0.1}$	$97.4_{\pm 0.0}$
EaSFE	$\mathbf{0.421}_{\pm 0.001}$	$\mathbf{0.736}_{\pm 0.0}$	$985.8_{\pm 1.9}$	$\mathbf{68.2}_{\pm 0.2}$	$\mathbf{78.6}_{\pm 0.2}$	$\mathbf{89.1}_{\pm 0.1}$	$\mathbf{99.7}_{\pm 0.0}$	$\mathbf{92.8}_{\pm 0.0}$	$\mathbf{92.9}_{\pm 0.0}$	$\mathbf{73.1}_{\pm 0.1}$	$\mathbf{97.6}_{\pm 0.1}$

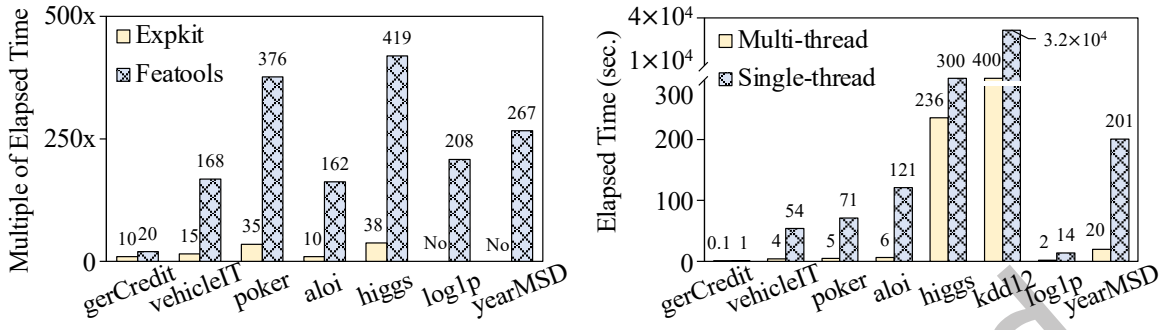
Table 4. Elapsed time results on baseline datasets with all baselines.

Method	CA	MI	ME	TE	BR	DI	NO	VE	AD	JA	CO
NN	318.6	94544.2	644.5	554.1	41014.5	615.1	995.1	8457.5	540.7	4411.6	18441.5
FCTree	139.8	81254.9	394.1	354.9	20104.6	341.5	689.4	4587.6	390.4	2541.1	9874.1
SAFE	×	×	×	312.5	374.2	57.1	64.1	604.1	35.1	×	×
AutoFeat	12.4	1254.1	1141.2	1874.1	38415.1	2148.2	2987.4	39847.1	741.1	21451.2	67412.5
AutoCross	×	×	×	6142.2	65414.1	7412.4	6845.5	6987.2	651.3	×	×
FETCH	5847.2	-	65479.0	9745.2	-	14125.7	19504.1	84147.4	1456.5	30254.8	-
ExpKit	×	×	×	20758.0	-	41516.0	14001.5	40091.4	10582.5	×	×
Featools	34.9	1470.8	43.6	336.0	2292.8	509.6	53.5	629.2	245.7	172.6	1576.6
OpenFE	5.1	1860.4	60.8	126.5	1988.8	185.4	264.9	184.0	34.1	324.8	2400.2
EaSFE	**1.7**	**742.5**	**17.4**	**11.0**	**115.8**	**13.4**	**35.5**	**58.8**	**6.4**	**119.7**	**320.0**

"min_samples_split" ranged from 2 to 10, "min_samples_leaf" ranged from 1 to 10, and "max_leaf_nodes" ranged from 10 to 1000.

6.2.2 Efficiency Comparison. Tables 4 and 7 show the feature engineering time for EaSFE and baseline methods. "OOM" indicates an out-of-memory error, and "No" means the method doesn't support the dataset. All methods used 20 threads, with settings consistent with Section 6.2.1. Data I/O time was excluded. To highlight EaSFE's efficiency, Figure 6a presents the time ratio of Expkit and Featools compared to EaSFE. EaSFE is about 10× faster than Expkit and hundreds of times faster than Featools—for example, 419× faster on *higgs*. We also measured cache misses to explore efficiency differences. As shown in Table 8, EaSFE has significantly fewer cache misses than Expkit, which contributes to its performance.

6.2.3 Scalability Comparison. To evaluate the scalability of EaSFE, we conducted experiments under limited computational resources using a virtual machine with 6GB of memory. Table 5 presents the performance of each method across different feature dimension scales—[10%, 20%, 40%, 60%, 80%, 100%] for *log1p*, and [0.001‰, 0.01‰, 0.1‰, 1‰, 1%, 100%] for *kdd12*. As shown in the table, EaSFE consistently demonstrates superior scalability compared to the baseline methods.



(a) Comparison on baselines

(b) Comparison on parallelism

Fig. 6. (a) Comparative analysis of elapsed time: Expkit and Featools vs EaSFE. (b) Comparison of EaSFE efficiency between single-thread and multi-thread.

Table 5. Scalability on *log1p* and *kdd12*.

Method	log1p						kdd12					
	10%	20%	40%	60%	80%	100%	0.001‰	0.01‰	0.1‰	1‰	1%	100%
NN	√	×	×	×	×	×	√	×	×	×	×	×
FCTree	√	×	×	×	×	×	√	×	×	×	×	×
SAFE	√	√	√	√	√	×	√	√	×	×	×	×
AutoFeat	√	√	×	×	×	×	√	×	×	×	×	×
AutoCross	√	×	×	×	×	×	√	×	×	×	×	×
FETCH	√	×	×	×	×	×	√	×	×	×	×	×
ExpKit	√	×	×	×	×	×	√	×	×	×	×	×
Featools	√	√	√	√	√	√	√	√	√	√	√	×
OpenFE	√	√	×	×	×	×	√	×	×	×	×	×
EaSFE	√	√	√	√	√	√	√	√	√	√	√	√

6.3 Impact of Parallel Optimization

To analyze the impact of the parallel optimization component in EaSFE, we compare its execution efficiency under single-threaded and multi-threaded settings (20 threads), while keeping all other components unchanged. As shown in Figure 6b, enabling multi-threaded execution leads to a substantial efficiency improvement, achieving approximately a 10x speedup over the single-threaded configuration. This result demonstrates that the parallel optimization component plays a critical role in accelerating EaSFE, highlighting its effectiveness and scalability for computationally intensive feature engineering tasks.

Table 6. Accuracy and F1 compare with baselines.

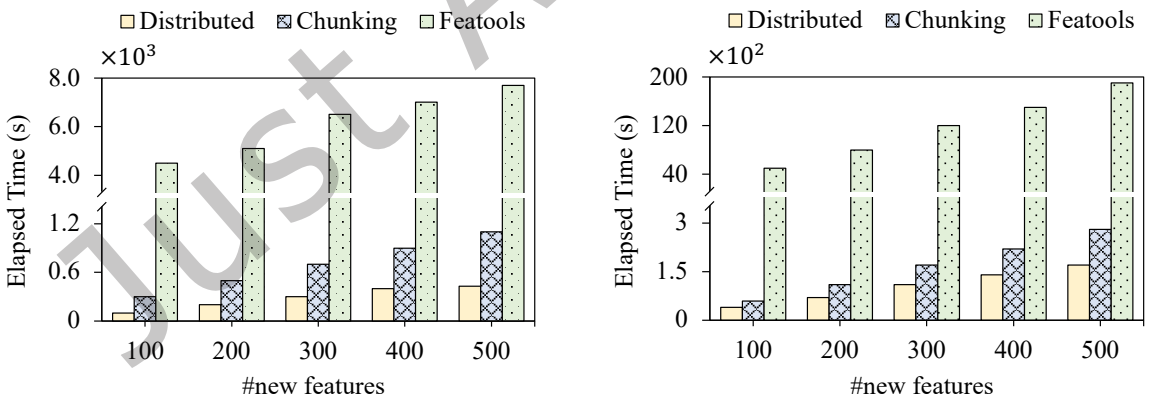
Dataset	Accuracy (%)			F1 (%)		
	Original	EaSFE	Featools	Original	EaSFE	Featools
gerCredit	73.80	**74.10**	72.70	66.38	**65.35**	64.35
vehicleIT	79.65	**79.89**	78.75	78.56	**78.89**	78.07
poker	57.13	**62.68**	56.37	11.80	**14.38**	0.1127
aloi	67.46	**67.52**	67.49	67.45	**67.49**	67.48
higgs	63.07	**64.14**	64.07	63.07	**64.11**	64.07
kdd12	95.56	95.56	OOM	48.86	48.86	OOM
avazu	98.95	**100.00**	99.10	98.01	**100.00**	98.99
bankFraud	98.68	**98.70**	98.60	53.63	**54.11**	51.12
	RMSE			R2 (%)		
log1p	0.383	**0.381**	0.383	49.20	**49.77**	49.33
yearMSD	10.24	**10.22**	10.23	11.01	**11.27**	11.12

Table 7. Elapsed time compare with Expkit and Featools.

Dataset		Time (s)		
		EaSFE	Expkit	Featools
classification	gerCredit	0.1	1	2
	vehicleIT	4	60	670
	poker	5	176	1,881
	aloi	6	62	973
	higgs	236	8,930	98,880
	kdd12	4,244	OOM	OOM
chunking	avazu	1,721	No	162,324
	bankFraud	623	No	61,412
regression	log1p	2	No	416
	yearMSD	20	No	5,348

Table 8. Comparison of cache miss value with Expkit.

Dataset	EaSFE	Expkit
gerCredit	178,893	179,089,102
vehicleIT	362,463,449	6,237,267,756
poker	559,668,300	10,635,280,560
aloi	327,394,525	7,602,924,454
higgs	37,483,461,404	841,236,475,961



(a) Comparison on *avazu* dataset

(b) Comparison on *bankFraud* dataset

Fig. 7. Comparison of elapsed time for Distributed, Chunking, and Featools on the *avazu* and *bankFraud* datasets.

6.4 Large Dataset Experiment

To analyze the impact of EaSFE's large-scale and sparse data handling components, we conducted a series of experiments on the *bankFraud* and *avazu* datasets. These experiments focus on evaluating the effectiveness and efficiency of chunk loading and distributed execution under large-scale and sparse settings. For *bankFraud*, we used four consistent chunks (1,000,000 instances, 34 features each), excluding two with differing feature dimensions. The *avazu* dataset includes two training (14,596,137 and 25,832,830 instances) and two testing chunks (1,719,304 and 2,858,160 instances). EaSFE supports large-scale processing via chunk loading and distributed cluster execution. As both yield consistent results and Expkit lacks support for large datasets, our experiments focus on comparing the performance with the original dataset and Featools. We also assessed the efficiency of EaSFE's two modes versus Featools. Due to hardware constraints, our experimental cluster consists of two nodes, each configured as described in Section 6.1. Nevertheless, the distributed design of EaSFE is general and can be extended to a larger number of nodes.

6.4.1 Effectiveness Comparison. Table 6 presents a comparison of Accuracy and F1 scores on the *bankFraud* and *avazu* datasets. In our experiments, we used the same experimental setup, sampling methods, and HPO process as described in Section 6.2.1. However, unlike previous experiments, the *avazu* and *bankFraud* datasets consist of multiple chunks. Therefore, for each block, we randomly sampled a subset, applied HPO, and then combined the results of all chunks to obtain the final evaluation. The results show that EaSFE maintains strong predictive performance across large-scale datasets, indicating that the introduced scalability components do not degrade effectiveness.

6.4.2 Efficiency Comparison. In Figure 7, we compare the efficiency of different large-scale processing modes in EaSFE, including distributed execution and chunk loading, against Featools. The Distributed approach utilizes a cluster comprising two nodes, as described earlier in this section. For Chunking, we set the buffer size to 2, and we used the same cluster configuration for Featools as that for Distributed. For consistency, we set the number of retained features to 50 for the *avazu* and to half of the original feature set for *bankFraud*.

The results indicate that the Distributed approach is significantly faster than Chunking, with the performance gap widening as the dataset size increases. This difference highlights the trade-off between disk I/O overhead in chunk loading and inter-node communication in distributed execution. However, compared to Featools, the Chunking approach shows more than an order-of-magnitude improvement in efficiency.

6.4.3 Impact of the number of buffers on chunk loading. To further isolate the effect of the multi-buffer optimization in chunk loading, we evaluate the impact of varying the number of buffers on execution efficiency, as shown in Figure 8. The x-axis represents the number of buffers, and we set the maximum number of features generated to 2,000 in our experiment. As depicted in the graph, the time cost decreases as the number of buffers increases, and there is a corresponding relationship between the speed improvement and the number of buffers. If two buffers are used, the speed improvement is close to two times, which shows the impact of the number of buffers on the efficiency.

6.4.4 Chunk loading on limited computational resources. To evaluate the effectiveness of the chunk loading component under constrained computational resources, we ran experiments on a Linux virtual machine with 6GB RAM and a 4-core Intel i7-8550U (1.80 GHz), without swap. Using *avazu* (606M non-zero training elements), full loading requires 4.8GB, excluding additional memory for targets, structures, and dense features—leading to an "out of memory" error. However, by using chunk loading on the largest *avazu* chunk (387M elements, 3GB), EaSFE completed the experiment successfully with peak memory usage around 5GB, demonstrating that the chunk loading component enables EaSFE to operate under strict memory constraints.

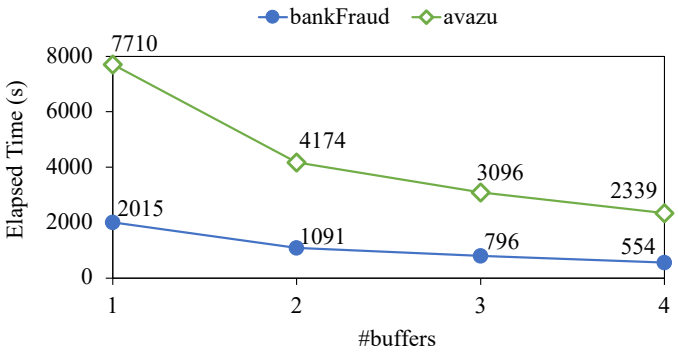


Fig. 8. Comparison of elapsed time for chunk loading of large datasets under different numbers of buffers.

6.5 Summary of Experimental Results

In summary, we draw the following main conclusions based on the experimental studies:

- Compared to the baseline methods, EaSFE reduces computational cost by approximately 10x and improves predictive performance (e.g., 5% accuracy improvement in *poker*), while also demonstrating better scalability, especially in handling large and sparse datasets.
- EaSFE exhibits robust parallel capability, achieving about 10x speedup in multi-threaded setups compared to single-threaded execution, demonstrating the effectiveness of parallelization in large-scale tasks.
- The introduced techniques (i.e., chunk loading, distributed processing, and storage strategy) significantly enhance EaSFE's scalability, especially for large-scale and sparse datasets, and efficiently adapt to training scenarios with limited resources. This component shows a clear advantage in memory usage and processing efficiency.

7 Conclusion

In this paper, we introduce an efficient and scalable feature engineering framework, namely EaSFE, which leverages various methods to generate a large number of new features and can select the effective ones during the generation process. Additionally, EaSFE incorporates several techniques to improve efficiency and scalability, including: (1) feature-level multiple-threaded parallelism; (2) efficient data loading via chunk loading and distributed cluster processing; and (3) a specialized storage strategy for sparse data. The experimental results across various datasets demonstrate that EaSFE significantly enhances the model performance (e.g., 5% accuracy improvement), surpassing other methods by several orders of magnitude in terms of efficiency (e.g., 10x speedup than existing methods). Meanwhile, the experiments on large-scale datasets further underscore the scalability of EaSFE.

Acknowledgments

This research was funded by the National Natural Science Foundation of China (Grant No. 62376099), the Postdoctoral Fellowship Program of CPSF (Grant No. GZC20251042) , the China Postdoctoral Science Foundation (Grant No. 2024M760785), and Key Research Project Plan of Higher Education Institutions in Henan (Grant No. 26A520002).

References

[1] 2023. [n. d.]. Featuretools: An Open Source Python Framework for Automated Feature Engineering. https://www.featuretools.com/.

[2] Mohammad Zoynul Abedin, Petr Hajek, Taimur Sharif, Md Shahriare Satu, and Md Imran Khan. 2023. Modelling Bank Customer Behaviour using Feature Engineering and Classification Techniques. Research in International Business and Finance 65 (2023), 101913.
[3] Sjoerd Boeschoten, Cagatay Catal, Bedir Tekinerdogan, Arjen Lommen, and Marco Blokland. 2023. The Automation of the Development of Classification Models and Improvement of Model Quality using Feature Engineering Techniques. Expert Systems with Applications (ESWA) 213 (2023), 118912.
[4] Kaizhi Chen, Chenjun Lin, Shangping Zhong, and Longkun Guo. 2014. A Parallel SRM Feature Extraction Algorithm for Steganalysis Based on GPU Architecture. In 2014 Sixth International Symposium on Parallel Architectures, Algorithms and Programming. 178–182. doi:10.1109/PAAP.2014.36
[5] Maximilian Christ, Andreas W Kempa-Liehr, and Michael Feindt. 2016. Distributed and Parallel Time Series Feature Extraction for Industrial Big Data Applications. arXiv preprint arXiv:1610.07717 (2016).
[6] Ayça Deniz and Hakan Ezgi Kiziloz. 2020. Parallel Multiobjective Feature Selection for Binary Classification. In 2020 5th International Conference on Computer Science and Engineering (UBMK). 1–5. doi:10.1109/UBMK50275.2020.9219383
[7] Wei Fan, Erheng Zhong, Jing Peng, Olivier Verscheure, Kun Zhang, Jiangtao Ren, Rong Yan, and Qiang Yang. 2010. Generalized and Heuristic-free Feature Construction for Improved Accuracy. In Proceedings of the 2010 SIAM International Conference on Data Mining (SDM). SIAM, 629–640.
[8] Yuling Fan, Baihua Chen, Weiqin Huang, Jinghua Liu, Wei Weng, and Weiyao Lan. 2022. Multi-label Feature Selection based on Label Correlations and Feature Redundancy. Knowledge-Based Systems 241 (2022), 108256.
[9] Geoffrey C Fox, Roy D Williams, and Guiseppe C Messina. 2014. Parallel Computing Works! Elsevier.
[10] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. 2023. Revisiting Deep Learning Models for Tabular Data. arXiv:2106.11959 [cs.LG] https://arxiv.org/abs/2106.11959
[11] Richard L Graham, Timothy S Woodall, and Jeffrey M Squyres. 2006. Open MPI: A Flexible High Performance MPI. In Parallel Processing and Applied Mathematics: 6th International Conference, PPAM 2005, Poznań, Poland, September 11-14, 2005, Revised Selected Papers 6. Springer, 228–239.
[12] Isabelle M Guyon, Andr&#xe, and Elisseeff. 2003. An Introduction to Variable and Feature Selection. The Journal of Machine Learning Research (JMLR) (2003).
[13] Marwa Hammami, Slim Bechikh, Chih-Cheng Hung, and Lamjed Ben Said. 2018. A Multi-Objective Hybrid Filter-Wrapper Evolutionary Approach for Feature Construction on High-dimensional Data. In 2018 IEEE Congress on Evolutionary Computation (CEC). IEEE, 1–8.
[14] Victoria J Hodge, Simon O'Keefe, and Jim Austin. 2016. Hadoop Neural Network for Parallel and Distributed Feature Selection. Neural Networks 78, C (2016), 24–35.
[15] Franziska Horn, Robert Pack, and Michael Rieger. 2020. The AutoFeat Python Library for Automated Feature Engineering and Selection. In Machine Learning and Knowledge Discovery in Databases: International Workshops of ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part I. Springer, 111–120.
[16] Gang Hu, Bo Du, Xiaofeng Wang, and Guo Wei. 2022. An Enhanced Black Widow Optimization Algorithm for Feature Selection. Knowledge-Based Systems 235 (2022), 107638.
[17] Sérgio Jesus, José Pombal, Duarte Alves, André Cruz, Pedro Saleiro, Rita Ribeiro, João Gama, and Pedro Bizarro. 2022. Turning the Tables: Biased, Imbalanced, Dynamic Tabular Datasets for ML Evaluation. Advances in Neural Information Processing Systems (NeurIPS) 35 (2022), 33563–33575.
[18] Guiyuan Jiang, Guiling Zhang, and Dakun Zhang. 2010. A Distributed Dynamic Parallel Algorithm for SIFT Feature Extraction. In 2010 3rd International Symposium on Parallel Architectures, Algorithms and Programming. 381–385. doi:10.1109/PAAP.2010.58
[19] 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.
[20] Aditya M Kashyap, Delip Rao, Mary Regina Boland, Li Shen, and Chris Callison-Burch. 2025. Predicting Explainable Dementia Types with LLM-aided Feature Engineering. Bioinformatics (2025), btaf156.
[21] 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.
[22] K. Kira and L. A. Rendell. 1992. A Practical Approach to Feature Selection. Morgan Kaufmann Publishers Inc. (1992).
[23] 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).
[24] Wang Lu, Wu Zhigang, and Cai Zixing. 2012. Research on Parallel Algorithm of Salient Features Extraction for Nature Scene Recognition. In 2012 International Conference on Computer Distributed Control and Intelligent Environmental Monitoring. 564–567. doi:10.1109/CDCIEM.2012.140
[25] Yuanfei Luo, Mengshuo Wang, Hao Zhou, Quanming Yao, Wei-Wei Tu, Yuqiang Chen, Wenyuan Dai, and Qiang Yang. 2019. Autocross: Automatic Feature Crossing for Tabular Data in Real-World Applications. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 1936–1945.

[26] Fatemeh Nargesian, Horst Samulowitz, Udayan Khurana, Elias B Khalil, and Deepak S Turaga. 2017. Learning Feature Engineering for Classification.. In Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, Vol. 17. 2529–2535.
[27] Selwyn Piramuthu and Riyaz T Sikora. 2009. Iterative Feature Construction for Improving Inductive Learning Algorithms. Expert Systems with Applications (ESWA) 36, 2 (2009), 3401–3406.
[28] Matthew Rocklin. 2015. Dask: Parallel Computation with Blocked algorithms and Task Scheduling. In SciPy. https://api.semanticscholar.org/CorpusID:63554230
[29] G. Roffo, S. Melzi, and M. Cristani. 2016. Infinite Feature Selection. In IEEE International Conference on Computer Vision (ICCV), 2015.
[30] L. Shang. [n. d.]. A Feature Selection Method Based on Information Gain and Genetic Algorithm. In 2012 International Conference on Computer Science and Electronics Engineering.
[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] Idheba Mohamad Ali Omer Swesi and Azuraliza Abu Bakar. 2019. Feature Clustering for PSO-based Feature Construction on High-Dimensional Data. Journal of Information and Communication Technology 18, 4 (2019), 439–472.
[33] Ioannis G Tsoulos. 2022. QFC: A Parallel Software Tool for Feature Construction, Based on grammatical evolution. Algorithms 15, 8 (2022), 295.
[34] Can Wang, Mitra Baratchi, Thomas Bäck, Holger H Hoos, Steffen Limmer, and Markus Olhofer. 2022. Towards Time-series Feature Engineering in Automated Machine Learning for Multi-step-ahead Forecasting. Engineering Proceedings 18, 1 (2022), 17.
[35] Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed Chi. 2021. Dcn v2: Improved Deep & Cross Network and Practical Lessons for Web-scale Learning to Rank Systems. In Proceedings of the Web Conference 2021. 1785–1797.
[36] Ting Wang, Yunpeng Cui, Tan Sun, Huan Li, Chao Wang, Ying Hou, Mo Wang, Li Chen, and Jinming Wu. 2025. A Feature Engineering Method for Whole-Genome DNA Sequence with Nucleotide Resolution. International Journal of Molecular Sciences 26, 5 (2025), 2281.
[37] Qi Xiong, Xinman Zhang, Wen-Feng Wang, and Yuhong Gu. 2020. A Parallel Algorithm Framework for Feature Extraction of EEG Signals on MPI. Computational and Mathematical Methods in Medicine 2020 (2020).
[38] L. Yu and H. Liu. 2004. Efficient Feature Selection via Analysis of Relevance and Redundancy. The Journal of Machine Learning Research (JMLR) (2004).
[39] Kyung Keun Yun, Sang Won Yoon, and Daehan Won. 2021. Prediction of Stock Price Direction using a Hybrid GA-XGBoost Algorithm with a Three-stage Feature Engineering Process. Expert Systems with Applications (ESWA) 186 (2021), 115716.
[40] Matei A. Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In USENIX Workshop on Hot Topics in Cloud Computing. https://api.semanticscholar.org/CorpusID:263897801
[41] Tianping Zhang, Zheyu Zhang, Zhiyuan Fan, Haoyan Luo, Fengyuan Liu, Qian Liu, Wei Cao, and Jian Li. 2023. OpenFE: Automated Feature Generation with Expert-level Performance. In Proceedings of the 40th International Conference on Machine Learning (ICML). 41880–41901.