Learning What Matters: Automated Feature Selection for Learned Performance Models in Parallel Stream Processing

Pratyush Agnihotri
TU Darmstadt and DFKI

Carsten Binnig
TU Darmstadt and DFKI

Manisha Luthra
TU Darmstadt and DFKI

ABSTRACT

Predicting performance in distributed stream processing systems relies on selecting relevant input features, a process traditionally requiring *expert-driven manual tuning*. However, manual selection is inefficient and prone to suboptimal choices. For example, features overly tied to one workload can cause the performance model to mispredict for another workload, affecting the model's generalizability. This paper presents an automated feature selection approach that systematically identifies the most relevant features across workloads (streams and queries), and resource dimensions for learned performance models. We employ feature selection strategies including feature ablation and statistical relevance analysis to evaluate feature importance and distinguish *transferable vs. non-transferable features* to improve generalization. By optimizing the feature set, our approach enhances the accuracy of performance prediction, reduces feature redundancy, and thereby improves parallelism tuning efficiency compared to manual selection. We demonstrate that our approach reduces reliance on manual tuning and training effort by 11× while maintaining robustness in performance models.

VLDB Workshop Reference Format:
Pratyush Agnihotri, Carsten Binnig, and Manisha Luthra. Learning What Matters: Automated Feature Selection for
Learned Performance Models in Parallel Stream Processing. VLDB 2025 Workshop: Applied AI for Database Systems and Applications (AIDB 2025).

1 INTRODUCTION

Performance Models for Parallelism Tuning. Distributed Stream Processing (DSP) has become a cornerstone of modern industries – from e-commerce [20] and finance [15] to social media – for analyzing large-scale data in real time [14]. For instance, Alibaba's retail platform processes on the order of 6 million transactions per second in production [20], and Facebook's streaming analytics handle around 9 GiB of event data per second while maintaining low latency [9]. Achieving such massive throughput demands high degrees of parallelism, i.e., running many operator instances in parallel, to keep up with incoming data rates. Thus, tuning the parallelism of each operator is critical to meet performance targets, but doing so effectively requires *accurate performance models* that predict performance, such as latency and throughput.

Why Automated Feature Selection is Important? A key challenge in building such performance models is selecting the right input features, such as workload characteristics (operator, query, and data stream), or hardware configurations that truly capture the factors affecting performance. *Manual feature selection* [18, 34] is labor-intensive and often suboptimal – engineers may pick features that seem relevant to a specific workload, only to find the model doesn't generalize. In fact, features that are overly tied to one scenario, e.g., a particular filter keyword or threshold, can cause a learned performance model to mispredict when the workload (query plans and data) characteristics or resource configuration change. This motivates the need for an *automated* way to identify *relevant* and *transferable* set of features for a generalizable performance model of DSP.

Research Question and Pitfalls of Existing Work. *RQ: How can we automate feature selection for learned performance models in DSP systems to improve parallelism tuning while ensuring that the models generalize across diverse workloads?* We seek to eliminate the *manual trial-and-error* in choosing model inputs and instead systematically find an optimal feature set that yields accurate performance predictions for different streaming jobs. By answering this question, we aim to enable DSP systems to self-tune their parallelism and resource usage more effectively, without requiring expert intervention for each new workload and resource configuration. Feature selection in machine learning particularly for performance modeling—has been studied in domains such as cloud configuration [7] and compiler optimization [8], but relatively little attention has been given to its role in DSP. We bridge this gap by proposing a systematic, multi-strategy feature selection pipeline that combines ablation analysis and statistical relevance metrics to identify critical and transferable features for DSP.

Our Idea of Automated Feature Selection for DSP. We propose a novel automated feature selection method for learned performance models in DSP systems. Instead of relying on intuition or *one-size-fits-all* feature sets, our approach systematically evaluates the importance of each candidate feature using multiple strategies, especially focusing to each model generalizability. First, we perform *feature ablation* studies: we iteratively remove or mask individual features or groups of related features and observe the impact on the model's prediction accuracy. This reveals which features are critical for predicting performance and which ones contribute little (or even introduce noise). Second, we conduct *statistical relevance analysis*, e.g., correlation and variance analysis, to detect features that are highly redundant or not informative across different workloads. By combining these methods, we pinpoint which input features are consistently useful across various streaming applications – we term these *transferable features* – versus those that are workload-specific or *non-transferable*. We then train the performance model using only the selected feature subset.

Our Results. Optimizing the feature set in this way yields a simpler and more robust model: prediction accuracy improves since spurious features no longer distract the learner, and the model generalizes better to unseen workloads because it is built only on the

Proceedings of the VLDB Endowment. ISSN 2150-8097.

truly relevant, workload-agnostic characteristics. Ultimately, this *automated feature selection* can enhance parallelism tuning in DSP systems: with more reliable performance predictions, the system can make better decisions about scaling operators and allocating resources, achieving lower latency and higher throughput across a range of streaming jobs. Most importantly our evaluations show that the automated feature selection yields upto 11× reduction in training time and improved performance predictions in comparison to manual selection.

Outline. In the remaining paper, we present our approach in Section 2, key findings from our experiments in Section 3, related work in Section 4 and finally conclude in Section 5.

2 OVERVIEW OF AUTOMATED FEATURE SELECTION PIPELINE

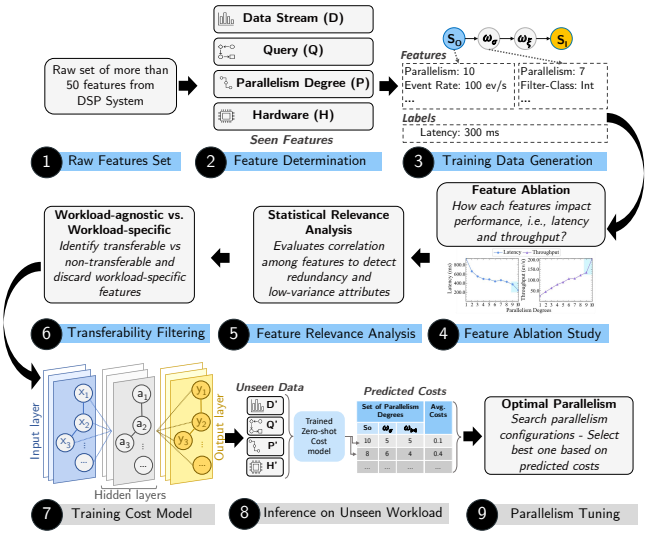

Figure 1: Overview of end-to-end pipeline for automated feature selection and cost modeling in DSP systems. The process begins with feature extraction from stream, query, parallelism, and resource dimensions. Ablation studies and statistical relevance analysis are used to identify transferable features. These are used to train a GNN-based cost model, which is later applied to unseen workloads for performance prediction and parallelism tuning.

In this section, we present a high-level overview of our automated feature selection pipeline for DSP as presented in Figure 1. The pipeline is designed to identify a compact, transferable set of features that improves the accuracy and generalization of learned performance models, specifically in zero-shot cost model [3, 5] for DSP.

The process begins with a feature ablation study (Section 2.2), where we systematically evaluate the impact of individual features related to workload characteristics (data streams and queries) and resource configurations. In each iteration, we remove one feature from the input set and measure the change in prediction accuracy of our zero-shot cost model. This allows us to distinguish features that are critical for performance estimation from those that are negligible or even detrimental.

Following ablation, we perform statistical relevance analysis (Section 2.3) using multiple feature selection strategies adapted from prior work [28]. This includes computing pairwise correlations, mutual information, and variance-based metrics to identify redundant, highly correlated, or low-variance attributes. The goal is to retain only those features that meaningfully contribute to predictive accuracy and generalize across different workloads. For each feature selection strategy, we record the number of features retained, the computational cost (search time), and the prediction accuracy on a held-out test set. Based on these evaluations, we identify a set of transferable features (Section 2.4)—i.e., features that consistently lead to better generalization across unseen workloads. These features form the input to our learned cost model, while non-transferable features are pruned to reduce complexity and risk of overfitting.

Finally, we train a Graph Neural Network (GNN)-based zero-shot performance model on the selected subset of features and evaluate its generalization ability on previously unseen query workloads [5]. This fully automated feature selection process ensures that our learned model is robust, scalable, and adaptive without the need for manual feature engineering.

2.1 Feature Determination

The automated feature selection pipeline begins with the extraction of a comprehensive set of around 80 raw features (cf. step ① and ② in Figure 1) from real-world executions of DSP workloads a subset presented in Table 1. These features span four essential categories: *data stream characteristics (D)*, *query plans (Q)*, *parallelism degrees (P)*, and *hardware resource profiles (H)* (Table 1). Each training instance is annotated with empirically observed performance metrics—primarily latency and throughput—collected under varying resources and workload configurations.

The primary goal of this stage is to evaluate the influence of individual features on query performance, especially as it relates to parallelism efficiency and system scalability. To that end, we conduct an exploratory analysis that inspects the *distributional properties*, *inter-feature correlations*, and statistical associations of each feature with observed performance metrics. Features that demonstrate high variability between workloads and strong correlation with latency or throughput are designated for further analysis in subsequent selection steps. Beyond raw statistical association, we further interpret the relevance of features in terms of their operational semantics within DSP systems.

2.2 Feature Ablation Study

To systematically assess the contribution of individual features from each feature dimension to performance prediction, we conduct a structured ablation study (cf. step ④ in Figure 1). The goal of this analysis is to identify features that significantly affect the prediction accuracy [24] of the cost model, and to distinguish those that are *critical*, *redundant*, or *workload-specific/-agnostic*.

To evaluate feature importance, we apply a *leave-one-feature-out* strategy [26] at both *fine-grained* and *group levels*. In the *fine-grained* analysis, each feature is individually removed from the

Node	Category	Feature	Description
Logical	operator-parallelism	Parallelism degree	Parallel instances of the operator
	operator-parallelism	Partitioning strategy	Strategy for data distribution (forward, rebalance, hashing)
	operator-parallelism	Grouping number	Number of operators grouped together by DSP
	operator-parallelism	Operator fan-out	Number of downstream operators per node
	operator-parallelism	Operator chaining index	Position of operator in a chained execution group
	data	Tuple width in	Input tuple width
	data	Input tuple fields	Values in input tuples
	data	Tuple data type	Data types in a single tuple
	data	Selectivity	Average selectivity of all instances of a given operator
	data	Event rate	Emitted event rate of the source
	query	Operator type	Type of operator
	query	Filter function	Comparison filter function, e.g., $<, \leq, \geq$
	query	Filter literal value	Filter literal value, e.g., sensor id = 50
	query	Operator name	Name of the operator for the application
	query	Number of filter operators	Number of filter operators for the application
	query	Cardinality hint	Estimated number of unique values for a key
	query	Predicate complexity	Number of clauses in filter condition
	query	Window type	Shifting strategy (tumbling/sliding)
	query	Window policy	Windowing strategy (count/time)
	query	Join key class	Join key data type
	query	Agg. class	Aggregation data type
Physical	resource	CPU cores	Number of processing cores
	resource	CPU frequency	CPU frequency on this instance
	resource	Node identifier	Unique identifier of every instance
	resource	CPU name	CPU name where DSP is running
	resource	OS Type	Operating system type in host machines
	resource	Host name	Given unique identifier or name to host machine

Table 1: Selected subset of determined features for DSP in four dimensions - parallelism, operator, data stream and hardware resources.

input, and the model is retrained to observe changes in prediction accuracy. In the grouped level analysis, we disable entire feature dimensions or categories, such as all operator-related or all parallelism-related features, while keeping the rest of the input constant. This approach reflects real-world deployment scenarios where certain feature sources may be unavailable, and provides insights into the robustness of the model under partial information. For instance, our experimental evaluation (cf. Section 3) shows that excluding parallelism- and resource-level features results in a substantial degradation of prediction accuracy, particularly for unseen query plans and resource configurations. In contrast, removing certain operator-level features—such as filter functions or literal data types has less impact on overall accuracy. These results suggest that parallelism- and system-level features are more generalizable across diverse DSP workloads.

In summary, the ablation study provides a basis for identifying features that are consistently impactful and generalizable. These insights motivate the next phase of our automated feature pipeline, i.e., *statistical relevance analysis*, which more precisely quantifies feature redundancy and selects transferable features using information-theoretic criteria [31].

2.3 Statistical Relevance Analysis

Following the ablation-based evaluation of feature importance, we proceed with a detailed *statistical relevance analysis* (cf. step ⑤ in Figure 1) to further refine DSP feature space. While the ablation study provides initial insights into the predictive utility of individual and grouped features based on model performance, it does not directly account for *inter-feature redundancy*, *sparsity constraints*, or *structural dependencies* inherent in graph-based models that become especially critical when learning compact representations in high-dimensional spaces. To address these limitations and improve the interpretability of our cost model, we conduct a detailed statistical relevance analysis using a diverse set of feature selection strategies [12, 25, 27, 36] spanning wrapper and embedded approaches (see Table 2), specifically tailored to our zero-shot GNN cost model named ZeroTune [5].

Table 2 summarizes the ten strategies evaluated. We deliberately omit *classical filter methods* [33], e.g., mutual information filters or correlation thresholds, as they typically assume vectorized, tabular input and are agnostic to the model architecture. In contrast, our data is graph-structured—spanning query operators, data streams, and resources—and is consumed by a GNN-based cost model. Applying filter methods would require flattening this structure, thereby discarding crucial topological and relational information. Furthermore, to the best of our knowledge, no filter-based selection methods currently exist that are compatible with heterogeneous GNN operating over typed directed acyclic graph (DAG) typical in stream processing pipelines.

Instead, we focus on *wrapper category* [23], which are model-agnostic and rely on the performance of the predictive model itself to guide feature selection. These include *greedy*, *sequential*, and *metaheuristic* search methods, e.g., SFS, SBS, SFFS, NSGA-II, each of which explores the space of feature subsets with different optimization objectives. Such approaches are particularly attractive in our context because they adapt to the underlying GNN architecture and are agnostic to the irregularities in graph data.

We also explore *embedded methods*, which incorporate feature selection directly into the model's training process. Notably, we include two state-of-the-art strategies: $\ell_{2,1}$*-norm regularization*, which encourages row-wise sparsity in learned weight matrices and has been used in structured feature selection in deep models [29, 40]. *Gumbel-Softmax reparameterization*, a differentiable relaxation for discrete feature selection that enables end-to-end training of binary masks [19]. Both approaches are well-suited for deep learning models and can be adapted to GNN without requiring explicit feature enumeration. However, since these methods output soft masks or continuous weight scores, we follow a post-hoc discretization procedure: features are ranked by their learned importance scores, and a grid search is used to determine the top-k subset that maximizes prediction accuracy on validation data.

For GNN-specific feature selection strategy, we implement two variants of *Recursive Feature Elimination (RFE)*: *RFE with GNNExplainer* [37], which estimates feature importance by perturbing node features and measuring the impact on the GNN's output. This method is model-agnostic and suitable for graph-level explanation [39], making it ideal for our use case where feature utility must generalize across diverse query graphs. *RFE with first-layer weights (RFEWI)*, which builds on the intuition that early layers in deep networks tend to learn general, transferable features [38]. By extracting and averaging the absolute magnitudes of weights from the first layer of each node-type encoder in the GNN, we construct global feature rankings, accounting for heterogeneity across different node types, e.g., operators vs. resources.

When features appear in multiple node-type encoders, e.g., Tuple Width used in both filter and aggregation operators, we compute the mean importance score across encoders, yielding a consistent, graph-level feature ranking that respects structural asymmetry.

Overall, this statistical relevance analysis enables us to distill a compact, transferable, and semantically meaningful feature set that

Strategy	Category	Description
Exhaustive Search (ES) [10]	Wrapper (Greedy)	Brute-force evaluation of all possible feature subsets.
Sequential Forward Selection (SFS) [6]	Wrapper (Sequential)	Iteratively adds the feature that improves performance the most.
Sequential Backward Selection (SBS) [17]	Wrapper (Sequential)	Iteratively removes the least important feature.
Sequential Forward Floating Selection (SFFS)[32]	Wrapper (Sequential)	Allows backtracking during forward search to re-evaluate combinations.
Sequential Backward Floating Selection (SBFS)[32]	Wrapper (Sequential)	Similar to SFFS, but starts from the full feature set.
Non-dominated Sorting Genetic Algorithm II (NSGA-II) [11, 36]	Wrapper (Metaheuristic)	Multi-objective evolutionary algorithm optimizing Q-error and feature count.
Recursive Feature Elimination with GNNExplainer (RFE)[16, 37]	Wrapper (GNN-specific)	Uses GNNExplainer to rank and iteratively remove features.
Recursive Feature Elimination with First-Layer Weights (RFEWI) [30]	Wrapper (Model-aware)	Uses early layer weights of GNN to rank features.
$\ell_{2,1}$-norm Regularization [29]	Embedded (Regularization)	Promotes group sparsity in the model; selects features with non-zero weights.
Gumbel-Softmax Feature Selection [1]	Embedded (Reparameterization)	Learns discrete feature masks via Gumbel-Softmax sampling.

Table 2: List of evaluated feature selection strategies, their methodological categories, and descriptions. Wrapper methods use performance feedback from the cost model. Embedded methods integrate selection within model training.

forms the basis for zero-shot model training in the final stage of our pipeline.

2.4 Transferability Filtering

After identifying high-impact and statistically relevant features, we introduce a final filtering step (cf. step ⑥ in Figure 1) to isolate features that are transferable—those that consistently improve prediction accuracy across a wide variety of parallelism, query plans, data streams, and resource configurations. In contrast to features that are only useful in specific scenarios (non-transferable or workload-specific), transferable features [38] contribute to the generalization capability of the learned cost model, a core requirement in the zero-shot based machine learning model [35]. We define a feature as transferable if it satisfies the following properties:

- **P1:** It is retained by multiple feature selection strategies, including both ablation-based and statistical methods.
- **P2:** It consistently contributes to model accuracy on new or unseen workloads, as measured by stable or improved Q-error across test queries not included during training.
- **P3:** It exhibits minimal variance in importance rankings across query types and system architectures, i.e., it is not workload (query or data stream)- or resource-specific in utility.

To assess these properties, we use the subset of selected features from each feature selection strategy and intersect them with those found effective in our *leave-one-out ablation* (cf. Section 2.2) and *statistical relevance analysis* (cf. Section 2.3). We then evaluate their generalization performance across the four main dimensions of variability: (i) parallelism, e.g., parallelism degree, (ii) query plans, e.g., linear, joins, filters, (iii) data characteristics, e.g., event rates, selectivity, and (iv) type of resources and configurations, e.g., homogeneous vs. heterogeneous clusters.

Features such as *parallelism degree*, *operator selectivity*, *event rate*, and *CPU cores* are found to be reliably useful across nearly all configurations. For instance, the parallelism degree directly governs the distribution of workload among operator instances and thus strongly correlates with query latency and throughput across different streaming workloads—making it highly transferable. In contrast, features like filter literal values (e.g., temperature sensor value < 50) tend to have a limited and inconsistent effect on prediction accuracy. Their impact varies significantly depending on the specific query logic and data distribution, and they are often pruned by multiple feature selection strategies. These are classified as *non-transferable* due to their scenario-specific utility and poor generalization in zero-shot settings.

We also quantify transferability using a *cross-validation* over workload types, where each held-out fold represents a new, unseen workload and resource configurations. A feature is marked as *non-transferable* if its inclusion does not improve prediction accuracy (i.e., reducing Q-error) across a majority of folds, or if it introduces performance instability (e.g., higher variance in predictions). By filtering out *non-transferable features*, we reduce model complexity and improve its ability to generalize in the zero-shot regime. The resulting feature set serves as the final input to our zero-shot model training pipeline [5].

3 EXPERIMENT RESULTS

In this section, we present the effectiveness of our automated feature selection pipeline for zero-shot DSP performance modeling. Specifically, we aim to answer the following questions:

- **Exp. 1: Comparing feature selection strategies and manually selected features.** Which feature selection strategies (e.g., RFE, GumbelSoftmax, $\ell_{2,1}$-norm) or manual selection yield the best trade-off between model accuracy, search time, and feature compactness?
- **Exp. 2: Importance of transferable vs. non-transferable features.** What is the predictive performance when using only transferable features versus only non-transferable features?
- **Exp. 3: Model compactness and training efficiency.** How does feature selection reduce model complexity and training overhead without sacrificing accuracy?

Experiment Setup and Metrics. We adapt the experimental setup as used in ZeroTune [5], deploying our experiments on the CloudLab [13] testbed. The underlying stream processing system is Apache Flink v1.16, with task orchestration managed via Kubernetes. The system is evaluated on a heterogeneous mix of homogeneous (Ho) and heterogeneous (He) cluster configurations as summarized in Table 3. These include diverse CPU, memory, and network configurations, enabling a comprehensive assessment across realistic deployment scenarios. The model is trained and validated on seen clusters, while generalization is tested on a mix of both seen and previously unseen cluster types. Our dataset is derived from 24,000 synthetic DSP queries encompassing linear pipelines and 2- to 3-way window joins. Each query is annotated with observed latency and throughput across various configurations of event rates, parallelism degrees, window parameters, and

Parameters	Unit	Training Range (Seen)	Testing Range (Unseen)
Event rate	ev/sec.	100, 200, 400, 500, 700, 1k, 2k, 3k, 5k, 10k, 20k, 50k, 100k, 250k, 500k, 1m	50, 75, 150, 300, 450, 600, 850, 1.5k, 4k, 7.5k, 15k, 35k, 175k, 375k, 750k, 1.5m, 2m, 3m, 4m
Tuple width and data type	values	[1 - 5] x [str., doubles, int]	[6 - 15] x [str., double, int]
Window length	tuples	5, 10, 25, 50, 75, 100	2, 3, 4, 7, 17, 37, 62, 82, 150, 200, 250, 300, 350, 400
Window duration	ms	250, 500, 1k, 2k, 3k	50, 100, 150, 200, 325, 750, 1.5k, 2.5k, 4k, 5k, 6k, 7k, 8k, 9k, 10k
Sliding length	ratio	[0.3, 0.4, 0.5, 0.6, 0.7] x Window length	
Cluster type	-	ms510, rs620	c6420, c8220x, c8220, dss7500, c6320, rs625
Network link speed	Gbps	1, 10	
Number of workers	-	2, 4, 6	3, 8, 10
Parallel query structures	-	Linear, 2-way join, 3-way join	2-chained filters, 3-chained filters, 4-way join, 5-way join, 6-way join
Benchmark queries	-	-	Spike detection, Smart-grid (local), Smart-grid (global)
Operator type	-	Source, Filter, Window-join, Window-Aggregation	
Parallelism degree categories	-	$1 \leq XS < 8, 8 \leq S < 16, 16 \leq M < 32,$ $32 \leq L < 64, 64 \leq XL < 128$	
CloudLab cluster type	-	Ho: m510, He: rs620	Ho: c6420, He: c8220x, c8220, dss7500, c6320, rs6525

Table 3: Training and inference parameter ranges [2, 4, 5] for feature selection strategies. Underlined values denote extrapolation scenarios used to evaluate model generalization beyond the training datasets. "Ho" and "He" denote homogeneous and heterogeneous cluster types [13].

cluster resources. The training dataset comprises 80% of the queries, with the remaining split equally into validation and test sets. To evaluate generalization, we use a distinct set of unseen queries, including real-world benchmarks such as smart-grid and spike detection workloads (cf. Table 3).

Implementation and Feature Selection Integration. We integrate our feature selection pipeline with the ZeroTune cost model by augmenting the training stage to include selected subsets of features obtained through our automated pipeline. This enables end-to-end evaluation of the impact of different feature sets on model performance. During training, we evaluate the model with features selected via wrapper and embedded strategies (cf. Section 2.3), while the test phase remains fixed in input dimensions to ensure fair comparisons. Unlike ZeroTune, which relies on expert-curated transferable features, our model dynamically selects them from a larger candidate pool.

Evaluation Metrics. We use the Q-error metric [24] to quantify prediction accuracy. For a given predicted cost $\hat{c}$ and the true observed cost c, Q-error is defined as:

$$\text{Q-error}(c, \hat{c}) = \max\left(\frac{c}{\hat{c}}, \frac{\hat{c}}{c}\right) \geq 1$$

Q-error provides a robust, scale-invariant measure of prediction deviation and is particularly suitable for evaluating models across heterogeneous workloads. We report both the median and tail, i.e., 50th and 95th percentiles of Q-error over the unseen test set to characterize accuracy under varying operational conditions.

3.1 Comparing Feature Selection Strategies

In this experiment, we evaluate and differentiate the performance of ten distinct feature selection strategies (cf. Section 2.3) by analyzing their trade-offs across three key criteria: (i) *model accuracy*, measured by median Q-error (cf. Figure 2a); (ii) *search time*, measured in hours (cf. Figure 2b); and (iii) *feature compactness*, defined as the number of selected features (cf. Figure 2c). These strategies span a diverse spectrum—ranging from brute-force wrappers and sequential heuristics to model-aware explainability methods, metaheuristics, and embedded regularization-based methods.

Comparison of Feature Selection Stategies. Figure 2 highlights that $\ell_{2,1}$*-norm* regularization yields the best overall trade-off, achieving the lowest Q-error 1.30, selecting the most compact subset 15 features, and requiring minimal search time 5.5 hours. This method promotes group sparsity by regularizing feature weights during training, thereby automatically retaining only informative, *transferable features*. For instance, features like *tuple width*, *window length*, or *event rate*, which correlate strongly with latency and throughput across queries and hardware, are consistently preserved. At the same time, *non-transferable* features, such as *node ID* or *literal constants*, are pruned. Similarly, *Gumbel-Softmax* strategy, another embedded approach, also demonstrates strong performance. It produces similarly compact feature sets with competitive accuracy and slightly higher but still efficient runtime. Its differentiable binary masking mechanism enables end-to-end optimization through reparameterization. However, its sensitivity to *hyperparameters*, such as sampling temperature, may introduce variance and reduce reproducibility across setups.

In contrast, *Exhaustive Search (ES)*, while theoretically optimal by evaluating all possible subsets, is computationally prohibitive in practice. Even after restricting the search space using feature groupings, e.g., freezing operator features while searching over data features, the runtime exceeded 232 hours. Despite its relatively low Q-error 1.35, the marginal gain over embedded approaches does not justify the extreme computational burden. This demonstrates the infeasibility of exhaustive methods in high-dimensional feature spaces common in stream processing.

NSGA-II, a multi-objective genetic algorithm, achieves a favorable balance between accuracy 1.32 and compactness by optimizing for both Q-error and feature count. However, its stochastic population-based nature incurs a substantial search time of 159 hours, and results may vary across runs. Although suitable for Pareto-optimal exploration, such costs limit its applicability in time-constrained deployments.

Sequential *heuristics—SFS, SBS, SFFS, and SBFS—demonstrate* moderate trade-offs. Their greedy selection behavior results in runtimes of $39 - 142$ hours but prone to local optima. For instance, *SFS* tends to repeatedly select highly correlated features such as CPU cores and CPU frequency, leading to redundancy. Floating variants *SFFS, SBFS* improve accuracy slightly by permitting backtracking but do not significantly reduce feature counts or runtimes.

The *RFEWI* strategy, which ranks features based on GNN first-layer weight magnitudes, performs worst overall. It yields the highest Q-error 1.55 and retains a large number of features (21), indicating that shallow signals from early layers are insufficient for capturing complex, multi-hop interactions in the graph structure. In comparison, *RFE with GNNExplainer* performs better across all axes, as it derives feature importance from end-to-end saliency scores, although it remains more tied to the training distribution and is less interpretable.

In summary, *embedded strategies*, particularly $\ell_{2,1}$*-norm* regularization, offer the best balance of *accuracy*, *compactness*, and *computational* efficiency. As reinforced in Table 4, such approaches are

(a) Prediction accuracy (in Median Q-error) **(b) Search time (in Hours)** **(c) # of features selected**

Figure 2: Comparison of feature selection strategies across three dimensions: (a) prediction accuracy, (b) search time required, and (c) number of selected features. Embedded strategies such as $\ell_{2,1}$-norm and Gumbel-Softmax achieve high accuracy while minimizing search time and feature count. In contrast, Exhaustive Search (ES), while accurate, is computationally prohibitive. Sequential wrappers (SFS, SBS, SFFS, SBFS) incur moderate time and yield larger feature sets with varying accuracy.

more scalable, generalizable, and interpretable than wrapper-based methods, making them better suited for learned cost models in distributed stream processing environments. While wrappers and metaheuristics can provide marginal accuracy gains, their high computational cost and poor reproducibility limit their practicality for real-world or time-sensitive scenarios.

Feature Selection by Strategies vs. Manual Selection. To complement our quantitative comparison, we provide a qualitative assessment of feature selection strategies by examining a representative subset of the features they selected from all set of features in comparison to manually selected features. In Figure 3 represents which features were chosen by each strategy, focusing on a selection of semantically meaningful attributes from the full feature space. The figure reveals that *embedded strategies*, particularly *$\ell_{2,1}$-norm* regularization and *Gumbel-Softmax*, consistently select a compact and high-quality subset of features in comparison to manually selected features. These primarily include *transferable features* such as *parallelism degree*, *event rate*, and *CPU cores*, which are known to exhibit strong generalization across heterogeneous workloads and resources. Their selections align with the low Q-error and compactness observed in prior experiments (cf. Figure 2), confirming that sparsity-inducing regularization and differentiable selection mechanisms are effective at capturing core performance drivers.

In contrast, *wrapper-based methods* such as *RFEWI* and *sequential heuristics*, e.g., SFS, SBS, often select a broader and more redundant set of features, including less transferable ones such as window policy or node ID. This behavior suggests limited ability to prune irrelevant features, which explains their comparatively higher prediction error and reduced generalizability under distributional shifts. Notably, *Exhaustive Search* and *NSGA-II* strategies cover a broader spectrum of features, capturing both relevant and marginally useful signals due to their large search space and multi-objective optimization design. While this can improve accuracy in controlled scenarios, it also increases computational cost and risks overfitting, especially when applied to high-dimensional or evolving environments. Overall, the feature-level analysis substantiates the superior selectivity and generalization of embedded strategies. It also reinforces our earlier conclusion that principled sparsity mechanisms are more effective than heuristic or brute-force approaches for constructing compact and transferable representations in learned cost models.

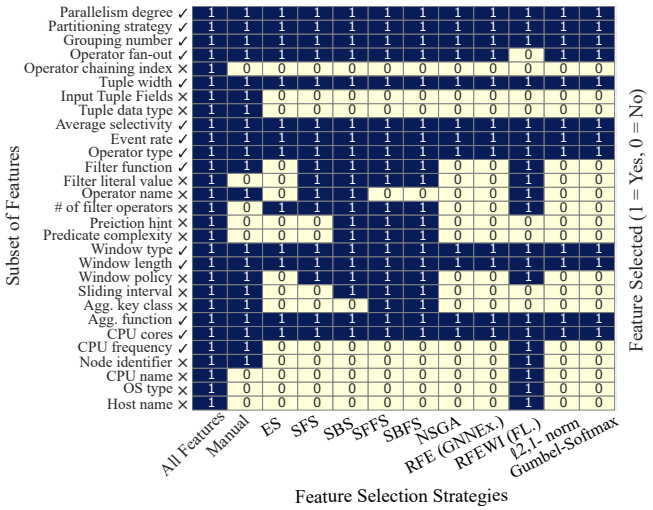

Figure 3: Representative subset of features from the total set of features selected by different strategies vs. all and manually selected features (baselines). Colored cells indicate inclusion of a feature by a strategy. Embedded methods (e.g., $\ell_{2,1}$-norm, Gumbel-Softmax) favor compact, transferable features, while some wrappers (e.g., SBS, RFEWI) include often select noisy, workload-dependent attributes. A cell marked as 1 (colored) indicates that the feature is selected. The features are annotated (✓) indicating transferable and (×) indicating non-transferable features.

3.2 Transferable vs. Non-Transferable Features

We further investigate the role of *transferable* and *non-transferable* features in predictive performance by isolating their impact. Specifically, we train two separate models: one using only transferable features, e.g., operator semantics, data selectivity, parallelism, and

Strategy	Type	Accuracy	Search Time	Compactness	Reasoning
Exhaustive Search (ES)	Brute-force wrapper	✓ Highest	× Very High	✓ Optimal (theoretically)	Evaluates all feature subsets. Although optimal, it is computationally infeasible for large spaces and prone to overfitting on small datasets.
SFS / SBS / SFFS / SBFS	Sequential wrappers	✓ Moderate to High	~ Moderate	~ Often Redundant	Greedy heuristics that may converge to suboptimal solutions. SFFS/SBFS allow limited backtracking to improve coverage at the cost of additional runtime.
NSGA-II	Metaheuristic wrapper	✓ High	~ High	✓ Pareto-optimal	Effectively balances accuracy and compactness using evolutionary search. Well-suited for multi-objective optimization, but computationally expensive and non-deterministic.
RFE (GNNExplainer)	Model-specific wrapper	✓ High	~ Medium	✓ Good	Utilizes model explainability to rank features, aligning well with internal model behavior. May inherit training data bias.
RFEWI (First-layer)	Model-aware wrapper	~ Moderate	✓ Fast	~ Coarse	Relies on initial-layer GNN weights, making it fast but often insufficient for capturing complex interactions and deeper semantics.
$\ell_{2,1}$-norm Regularization	Embedded	✓ High	✓ Fast	✓ Sparse	Promotes group sparsity through regularization, leading to compact and transferable feature sets. Efficient and generalizable.
Gumbel-Softmax	Embedded	✓ High	~ Medium	✓ Sparse	Learns discrete masks via differentiable relaxation. Balances accuracy and sparsity, but is sensitive to tuning.

Table 4: Comparison of feature selection strategies based on accuracy, search time, and compactness.

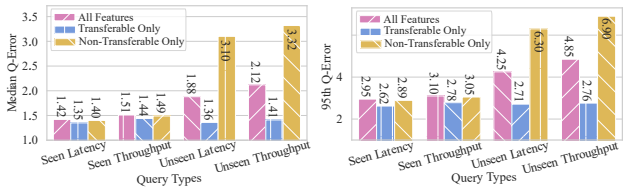

Figure 4: Median and 95th percentile Q-error across query types when training with transferable-only, non-transferable-only, and all features. Transferable features match or outperform full models on unseen data.

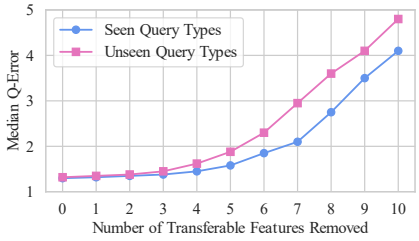

Figure 5: Effect of feature removal on model accuracy. As top-ranked transferable features are removed, the Q-error remains stable initially but increases sharply after a critical threshold, demonstrating that a small subset of features carries most predictive power.

another using only non-transferable features, e.g., literal constants, hardware IDs, filter conditions. We then evaluate both models' accuracy across seen and unseen query workloads and resource configurations, as presented in Figure 4. The results show that the *transferable-only* model achieves predictive accuracy comparable to—or better than—the *full feature model* on unseen workloads. For instance, on 95th percentile Q-error in unseen latency and throughput prediction, the transferable model significantly outperforms the non-transferable-only model. Similar trends are evident for median Q-error, confirming that transferable features are both high-signal and generalizable.

These findings align with prior observations in Experiments 1. In Experiment 1 (cf. Figure 2), selection strategies like $\ell_{2,1}$*-norm*, *Gumbel-Softmax*, and *RFE-GNNExplainer* excelled by favoring these features—offering compact yet accurate models. This experiment consolidates this evidence, showing that transferable features alone suffice for robust generalization under distribution shifts in workload or system configuration. In contrast, the *non-transferable*-only model, while effective on seen data, suffers significant degradation on unseen setups. This highlights its susceptibility to overfitting—capturing spurious correlations in context-specific features like node IDs, literal constants, or memory values that lack general utility. Such features often act as shortcuts that fail when deployed in new or unseen environments.

In summary, this experiment emphasizes the foundational role of transferable features in building accurate and generalizable learned cost models for distributed stream processing. These features not only improve performance and reduce variance but also streamline feature selection, leading to faster training and better scalability—particularly valuable in dynamic, heterogeneous systems.

3.3 Model Compactness and Training Efficiency

Model Compactness. To understand the sensitivity of the cost model to individual features, we analyze the impact of systematically removing transferable features in decreasing order of their importance (cf. Figure 5). The plot tracks the median Q-error as the top-10 ranked subset of transferable features are removed one-by-one, comparing prediction accuracy on both seen and unseen queries. Initially, we observe that the removal of the least important features induces only marginal increases in prediction error. For instance, eliminating the bottom five features raises the median Q-error from approximately 1.3 to just 1.6 for seen queries, and

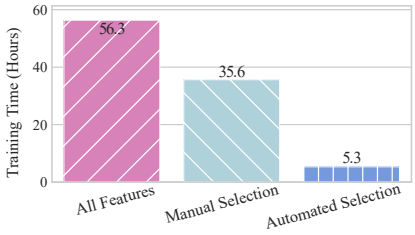

Figure 6: Comparison of model training time across three strategies—All Features, Manual Selection, and Automated Selection, highlighting the efficiency gains of automated selection. The automated model achieves up to 11× reduction in training time while maintaining predictive accuracy.

from 1.4 to 1.7 for unseen queries. This behavior confirms that a large fraction of low-impact features can be discarded without significantly affecting accuracy—indicating that the model is robust to feature sparsification and that many features are redundant.

However, as higher-ranked features are removed, the Q-error begins to rise steeply. When the most important features are excluded, e.g., after removing more than seven, the median Q-error spikes sharply, reaching 3.1 for seen and over 4.0 for unseen queries. This inflection—often referred to as a *knee* in the error curve—highlights the existence of a small but critical subset of transferable features that contribute disproportionately to model performance. These findings underscore two important insights: First, cost models benefit from aggressively pruning irrelevant or weakly informative features, which simplifies model training and improves generalization. Second, preserving top-ranked transferable features is essential for maintaining accuracy, particularly in scenarios involving distribution shifts such as unseen queries or resource configurations.

Furthermore, this experiment aligns with earlier findings from Experiment 2 (cf. Section 3.2), where models trained solely on transferable features achieved strong generalization.

Training efficiency. To further assess the impact of feature selection on model complexity and training overhead, we compare the training time excluding search time, required by three configurations: (i) full feature set (no selection), (ii) manually selected features, and (iii) features selected via our automated pipeline. As shown in Figure 6, the model trained on all available features requires over 56 hours to converge, while the automated selection achieves near-optimal accuracy with only 5.3 hours of training—an 11 reduction in training time. Manual selection, although better than the full model, still requires approximately 35.6 hours, largely due to redundant or less informative features included based on intuition.

This stark contrast demonstrates the effectiveness of automated feature selection in improving training efficiency. The reduction stems from two factors: (a) smaller feature sets that reduce input dimensionality and model size, and (b) prioritization of high-signal, transferable features that enable faster convergence. For instance, with only 15 features, the automated pipeline yields better generalization and significantly reduces learning time. In contrast, training with all features imposes both computational and statistical burdens—longer runtime and increased overfitting risk—due to the inclusion of context-specific or noisy features. Manual feature engineering, while helpful, is limited by human bias and lacks systematic assessment of feature relevance. Thus, this experiment confirms that automated feature selection not only yields compact and generalizable models but also offers considerable computational savings, making it ideal for real-time or resource-constrained stream processing systems.

4 RELATED WORK

In this section, we summarize existing work based on recent advances in feature selection for machine learning on (1) graph-structured data and (2) performance modeling. To the best of our knowledge, our work is the first to propose and evaluate a comprehensive multi-strategy feature selection framework specifically tailored to GNN-based cost models in stream processing.

Feature Engineering in Machine Learning. Selecting relevant features is a well-studied problem in machine learning. Classic feature selection [12, 25, 27, 36] include *filter methods*, (e.g., using statistical tests or information gain), *wrapper methods* (searching for the best feature subset via cross-validation), and *embedded methods* (where feature selection is part of model training). In general, reducing feature sets can mitigate overfitting and the curse of dimensionality, leading to simpler, more generalizable models. Our work shares this motivation but is applied in a novel context of stream processing performance models with multiple feature dimensions. Unlike standard feature selection, our DSP scenario involves streaming workloads (query plans and data streams from different applications) and resource characteristics for query plan execution and deployment, requiring careful consideration of what constitutes a *feature*. For instance query graph properties are unique to this domain. There is emerging work on feature selection for data stream classification, but our focus is on features for performance prediction in stream processing systems, which to our knowledge has not been explicitly addressed before. Furthermore, traditional approaches in stream processing often assume hand-crafted features that reflect intuition about operator characteristics, data rates, and resource metrics. While effective to an extent, these methods fail to scale across diverse workloads or adapt to evolving system behaviors.

Feature Selection for Performance Modeling. Feature selection in machine learning—particularly for performance modeling—has been studied in domains such as compiler optimization [8] and cloud configuration [7], but relatively little attention has been given to its role in DSP. We bridge this gap by proposing a systematic, multi-strategy feature selection pipeline that combines ablation analysis and statistical relevance metrics to identify critical and transferable features for DSP.

Research Gap. As our model architecture builds on GNNs, feature selection for GNNs is a relatively underexplored topic. While *filter-based* methods [22, 33] are widely used in traditional tabular domains [25], their application to GNNs is non-trivial due to the need to preserve structural dependencies. Recent embedded and *wrapper methods* [23], such as GNNExplainer [37] and GSAT [21], have shown promise in interpreting node-level importance. Our work leverages such methods—along with architectural insights from GNN literature—to evaluate feature relevance in the context of

DSP workload prediction. Notably, we extend existing methods (e.g., RFE with GNNExplainer [37], $\ell_{2,1}$-norm [29], and GumbelSoftmax regularization [19]) to systematically rank and select features for graph-based cost models. While prior work has explored learned models for stream processing and the utility of GNNs for structured prediction tasks, few efforts have tackled the problem of *automating feature selection for zero-shot performance modeling in DSP systems.*

5 CONCLUSION

In this paper, we presented a novel automated feature selection pipeline for performance modeling in DSP systems for parallelism tuning. Our approach systematically evaluates a diverse set of features across query, data, parallelism, and hardware dimensions, using a combination of ablation studies and statistical relevance analysis to identify those that are transferable across workloads. By isolating generalizable features and eliminating redundant or scenario-specific ones, we enable the training of robust zero-shot cost models that generalize effectively to unseen queries and deployment configurations. Our evaluation demonstrates that using only the selected transferable features improves both the predictive accuracy and generalization capability of learned cost models, while significantly reducing feature redundancy and training complexity. These improvements also translate into better decisions for parallelism tuning, ultimately enhancing the throughput and latency trade-offs in DSP systems.

While our study focused on feature selection for GNN-based zero-shot cost models, several promising directions remain for future exploration. First, we plan to investigate adaptive feature selection approaches that dynamically adjust the feature subset during model retraining in evolving workloads. Second, the integration of temporal patterns in streaming data—captured via time-series analysis or recurrent neural architectures—could further enrich the feature space and benefit prediction tasks. Finally, extending the proposed pipeline to support multi-objective optimization, e.g., balancing accuracy with energy efficiency or cost constraints, represents a valuable direction for practical deployments in resource-constrained environments.

ACKNOWLEDGMENTS

This work has been supported by the LOEWE program (Reference III 5 - 519/05.00.003-(0005)), etaGPT project under grant number 03EN4107, Athene Young Investigator Programme and hessian.AI at TU Darmstadt, as well as DFKI Darmstadt.

REFERENCES

[1] Deepak Bhaskar Acharya and Huaming Zhang. 2020. Feature selection and extraction for graph neural networks. In *Proceedings of the 2020 ACM southeast conference*. 252–255.

[2] Pratyush Agnihotri and Carsten Binnig. 2025. Demonstrating PDSP-Bench: A Benchmarking System for Parallel and Distributed Stream Processing. In *Companion of the 2025 International Conference on Management of Data*. 7–10.

[3] Pratyush Agnihotri, Boris Koldehofe, Carsten Binnig, and Manisha Luthra. 2023. Zero-Shot Cost Models for Parallel Stream Processing. In *Proceedings of the Sixth International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. 1–5.

[4] Pratyush Agnihotri, Boris Koldehofe, Roman Heinrich, Carsten Binnig, and Manisha Luthra. 2025. PDSP-Bench: A Benchmarking System for Parallel and Distributed Stream Processing. In *Performance Evaluation and Benchmarking: Traditional to Big Data to Internet of Things*, Raghunath Nambiar and Meikel Poess (Eds.). Springer International Publishing, 1–23. https://arxiv.org/abs/2504.10704

[5] Pratyush Agnihotri, Boris Koldehofe, Paul Stiegele, Roman Heinrich, Carsten Binnig, and Manisha Luthra. 2024. ZeroTune: Learned Zero-Shot Cost Models for Parallelism Tuning in Stream Processing.. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 2040–2053.

[6] David W Aha and Richard L Bankert. 1995. A comparative evaluation of sequential feature selection algorithms. In *Pre-proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*. PMLR, 1–7.

[7] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. 2017. {CherryPick}: Adaptively unearthing the best cloud configurations for big data analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 469–482.

[8] Amir H Ashouri, William Killian, John Cavazos, Gianluca Palermo, and Cristina Silvano. 2018. A survey on compiler autotuning using machine learning. *ACM Computing Surveys (CSUR)* 51, 5 (2018), 1–42.

[9] Guoqiang Jerry Chen, Janet L Wiener, Shridhar Iyer, Anshul Jaiswal, Ran Lei, Nikhil Simha, Wei Wang, Kevin Wilfong, Tim Williamson, and Serhat Yilmaz. 2016. Realtime data processing at facebook. In *Proceedings of the International Conference on Management of Data*. 1087–1098.

[10] Manoranjan Dash and Huan Liu. 2003. Consistency-based search in feature selection. *Artificial intelligence* 151, 1-2 (2003), 155–176.

[11] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[12] Justin Doak. 1992. An evaluation of feature selection methods and their application to computer security. *Techninal Report CSE-92-18* (1992).

[13] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 1–14.

[14] Data Flair. 2016. *Apache Flink Use Cases – Real life case studies of Apache Flink*. [Online; accessed 20-05-2022].

[15] Sebastian Frischbier, Mario Paic, Alexander Echler, and Christian Roth. 2019. Managing the Complexity of Processing Financial Data at Scale - An Experience Report. In *Complex Systems Design and Management*. Springer International Publishing, 14–26.

[16] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. 2002. Gene selection for cancer classification using support vector machines. *Machine learning* 46 (2002), 389–422.

[17] Amin Ul Haq, Jianping Li, Muhammad Hammad Memon, Muhammad Hunain Memon, Jalaluddin Khan, and Syeda Munazza Marium. 2019. Heart disease prediction system using model of machine learning and sequential backward selection algorithm for features selection. In *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*. IEEE, 1–4.

[18] Md Rashedul Islam, Aklima Akter Lima, Sujoy Chandra Das, Muhammad Firoz Mridha, Akibur Rahman Prodeep, and Yutaka Watanobe. 2022. A comprehensive survey on the process, methods, evaluation, and challenges of feature selection. *IEEE Access* 10 (2022), 99595–99632.

[19] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* (2016).

[20] Xiaowei Jiang. 2021. Blink: How Alibaba Uses Apache Flink. https://www.ververica.com/blog/blink-flink-alibaba-search. [Online; accessed 27-05-2022].

[21] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. 2020. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 66–74.

[22] Alan Jović, Karla Brkić, and Nikola Bogunović. 2015. A review of feature selection methods with applications. In *2015 38th international convention on information and communication technology, electronics and microelectronics (MIPRO)*. Ieee, 1200–1205.

[23] Ron Kohavi and George H John. 1997. Wrappers for feature subset selection. *Artificial intelligence* 97, 1-2 (1997), 273–324.

[24] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? 9, 3 (2015), 204–215.

[25] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P Trevino, Jiliang Tang, and Huan Liu. 2017. Feature selection: A data perspective. *ACM computing surveys (CSUR)* 50, 6 (2017), 1–45.

[26] Jianguo Liu, Neil Danait, Shawn Hu, and Sayon Sengupta. 2013. A leave-one-feature-out wrapper method for feature selection in data classification. In *2013 6th International Conference on Biomedical Engineering and Informatics*. IEEE, 656–660.

[27] Luis Carlos Molina, Lluís Belanche, and Àngela Nebot. 2002. Feature selection algorithms: A survey and experimental evaluation. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings*. IEEE, 306–313.

[28] Felix Neutatz, Felix Biessmann, and Ziawasch Abedjan. 2021. Enforcing Constraints for Machine Learning Systems via Declarative Feature Selection: An

Experimental Study. In *Proceedings of the 2021 International Conference on Management of Data*. 1345–1358.

[29] Feiping Nie, Heng Huang, Xiao Cai, and Chris Ding. 2010. Efficient and robust feature selection via joint $\ell_{2,1}$-norms minimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems - Volume 2*. 1813–1821.

[30] Cheng Peng, Xinyu Wu, Wen Yuan, Xinran Zhang, Yu Zhang, and Ying Li. 2019. MGRFE: Multilayer recursive feature elimination based on an embedded genetic algorithm for cancer classification. *IEEE/ACM transactions on computational biology and bioinformatics* 18, 2 (2019), 621–632.

[31] Jose C Principe, Dongxin Xu, Qun Zhao, and John W Fisher. 2000. Learning from examples with information theoretic criteria. *Journal of VLSI signal processing systems for signal, image and video technology* 26 (2000), 61–77.

[32] Pavel Pudil, Jana Novovičová, and Josef Kittler. 1994. Floating search methods in feature selection. *Pattern recognition letters* 15, 11 (1994), 1119–1125.

[33] Noelia Sánchez-Maroño, Amparo Alonso-Betanzos, and María Tombilla-Sanromán. 2007. Filter methods for feature selection–a comparative study. In *International Conference on Intelligent Data Engineering and Automated Learning*. Springer, 178–187.

[34] Jiliang Tang, Salem Alelyani, and Huan Liu. 2014. Feature selection for classification: A review. *Data classification: Algorithms and applications* (2014), 37.

[35] Yongqin Xian, Christoph H Lampert, Bernt Schiele, and Zeynep Akata. 2018. Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly. *IEEE transactions on pattern analysis and machine intelligence* 41, 9 (2018), 2251–2265.

[36] Bing Xue, Mengjie Zhang, Will N Browne, and Xin Yao. 2015. A survey on evolutionary computation approaches to feature selection. *IEEE Transactions on evolutionary computation* 20, 4 (2015), 606–626.

[37] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems* 32 (2019).

[38] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? *Advances in neural information processing systems* 27 (2014).

[39] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. 2020. Xgnn: Towards model-level explanations of graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 430–438.

[40] Xiaotong Yuan, Ping Li, Tong Zhang, Qingshan Liu, and Guangcan Liu. 2016. Learning Additive Exponential Family Graphical Models via $\ell_{\{2,1\}}$-norm Regularized M-Estimation. In *Advances in Neural Information Processing Systems*, Vol. 29.