Abstract—This paper introduces ZeroTune, a novel cost model for parallel and distributed stream processing that can be used to effectively set initial parallelism degrees of streaming queries. Unlike existing models, which rely majorly on online learning statistics that are non-transferable, context-specific, and require extensive training, ZeroTune proposes data-efficient zero-shot learning techniques that enable very accurate cost predictions without having observed any query deployment. To overcome these challenges, we propose ZeroTune, a graph neural network architecture that can learn from the structural complexity of parallel distributed stream processing systems, enabling them to adapt to unseen workloads and hardware configurations. In our experiments, we show when integrating ZeroTune in a distributed streaming system such as Apache Flink, we can accurately set the degree of parallelism, showing an average speed-up of around 5× in comparison to existing approaches.

Index Terms—Zero-shot cost models, Parallelism tuning

I. INTRODUCTION

Why Parallel Stream Processing? Distributed Stream Processing (DSP) is the key data processing paradigm for analyzing data streams in real time. It is commonly used in a wide range of application domains, such as key players in retail Alibaba or online gaming providers King use DSP for their core operations [1]. Nowadays, many DSP applications are required to process very high volumes of data, e.g., Alibaba needs to process an average of 4 million transactions per second while workload peaks can be even much higher [1]. Such scenarios require data processing over many parallel operator instances to keep up with the high arrival rates of data tuples. Furthermore, such applications impose a wide range of different workload characteristics, raising the challenging problem of parallelism tuning, i.e., setting parallelism degree, dependent on the workload.

Cost Model for Parallelism Tuning. However, to tune the parallelism of DSP operators, it is essential to understand the resulting performance accurately before actually applying the change. The reason is that each change in parallelism might result in reconfiguration of a DSP query requiring expensive relocation and splitting of operators and state to run under a new degree of parallelism. As such, accurate cost models are required that can upfront — before applying a new degree of parallelism — estimate the expected performance metrics such as latency or throughput in a precise manner.

Why is it challenging? Designing an accurate cost model for operator parallelization is, however, a highly challenging task. Many existing solutions for parallelism tuning [5][11] based on machine learning adapts online learning where they directly predict parallelism degrees on-the-fly by monitoring runtime performance of queries. This has three major challenges for a parallel DSP C1: While online learning appears more attractive for scaling decisions in stream processing, it results in highly incorrect initial provisioning of parallel operators. Consequently, the DSP query needs several oscillations until it reaches a stable state and hence has very long convergence times [12]. This iterative nature of trying several parallelism degrees is impractical for real-time applications such as online gaming as it leads to substantial delays.

C2: Because of the nature of online learning, they train the model on a set of context-specific, herein termed as non-transferable features, which poses challenges for its application in machine learning [4]. Such features, like a temperature filter literal of “≤ 27 degrees” may work well for certain data streams (e.g., weather reports) but fail to generalize across varied contexts (e.g., smoke detection data stream). We should instead learn from context-independent characteristics, e.g., filter complexity like how many attributes are specified, to generalize across data contexts and cope with highly changing data environments of streaming workloads.

C3: While initial models achieve generalization, they require an enormous training effort [13][14]. For instance, known models from databases have to be trained on more than 200k queries and 15 different databases to allow generalizability across unseen databases [15][16]. Contrary to databases, DSP workloads and in particular data characteristics of streams cannot be known in advance, so both supporting more variety of workloads and reducing training effort for unseen workloads are highly important to support predictable behavior of parallel operator deployments.

Our proposal: ZeroTune cost model. In this work, we propose ZeroTune that can already initially configure parallelism degrees based on cost estimations of parallel
query plans to avoid costly oscillations from the beginning of query execution. While the proposed model can also be used to readjust parallelism degree at runtime, this is not the focus of this paper. We rather want to show that the cost model provides initial good selections of parallelism degrees. To do this, ZeroTune must be able to understand the dynamics of a DSP system in an offline way, such that it can be transformed across parallel DSP queries and applications without expensive retraining. Therefore, ZeroTune builds on a novel learning paradigm for cost estimation based on the recent advances in transfer learning such as in LLMs [17], here referred to as data-efficient zero-shot learning [18]. ZeroTune’s main goal is to accomplish generalizability and an accurate cost prediction across workloads while reducing the amount of training data.

ZeroTune is trained with so-called transferable features, which have the same semantic meaning across different workloads. For instance, in the filter example above, we take filter operator “<” to learn from the cost (latency and throughput) overhead for this operator based on a datastream-agnostic characteristic.

**Novel parallel graph representation.** For learning and encoding the transferable features, we introduce a novel approach for predicting the costs of parallel query plans (PQP) in a Graph Neural Network (GNN) model architecture. We create a detailed graph model where each node represents key elements of the parallel DSP system in the form of transferable features, such as parallel operator instances, partitioned data streams, and parallel operator mapping on hardware resources. These nodes, defined by their unique attributes and interconnections with other parallel operators, allow the model to understand the complex relationships and interdependencies within the parallel query plan. Through this graph-based approach, our model, ZeroTune, is capable of adapting to unseen parallel operators and hardware, extending its applicability beyond the initial training data. This makes it a versatile model for optimizing parallel query execution in diverse computational settings.

**Benefits of ZeroTune.** ZeroTune provides highly accurate cost predictions in comparison to non-transferable representations of ML models as evidenced in Figure 1. More prominently, for unseen query types, it performs 1000× better than existing learned cost models (e.g., using a flat vector in contrast to our proposed graph representation). Furthermore, by training ZeroTune with a novel data-efficient training strategy OptiSample by exploring parallelism degrees derived based on the analytical approaches [12] [19], it achieves generalizability in as little as 5k queries in comparison traditional training strategies which need around 4× more training data as we show later in our evaluations.

Finally, the key contributions of this paper are:

- A zero-shot cost model for parallel query plan predictions that enables generalizability (Section III).
- A novel data-efficient training strategy, OptiSample to train the cost model that enables generalizability in a few amount of training queries and time (Section IV).
- An extensive evaluation that shows the robustness and accuracy of ZeroTune on seen-unseen workloads, across parallelism degrees, unseen operator parameters, data-efficiency in training and speed-ups achieved with the parallelism tuning approach in comparison to other approaches [4] [19] [20] (Section V).

II. Overview of ZeroTune

The main goal of ZeroTune is to provide cost predictions, e.g., for latency and throughput of running parallel stream processing query on distributed and parallel hardware resources without having observed running the query, herein, termed as unseen. In the following, we present the solution (S) overview of our approach towards this along with how we address the aforementioned challenges (C).

C1: Incorrect initial parallelism. To correctly provision the parallelism of operators right from the deployment of the query, we use an offline supervised model as given next.

S1: Figure 2 shows the overall approach where we first train ZeroTune on different data streams, query structures, and heterogeneous resource configurations by selecting a set of transferable features (see C2) in a supervised way (left). Once trained, the ZeroTune is used to correctly infer the costs (latency and throughput) for an unseen query and resource combination (right). Finally, these costs are used by an optimizer to select a parallelism configuration that gives minimum costs. Contrary to existing work, ZeroTune sets the right parallelism degrees from the start while adapting to unseen workloads as we explain next.
C2: Generalizability. One of the key questions of ZeroTune is how it achieves generalizability across workloads of parallel and concurrent operators on diverse hardware configurations. Here, we draw inspiration from how generalizable cost models are designed for databases [13, 15]. There are two main ideas behind this (1) novel feature selection process of so-called transferable features and (2) generic graph representation that maps the query plan to learn from the runtime characteristics. However, the cost model proposed in [15] is tailored towards database operators and cannot reason about the costs of parallel and distributed “streaming” operators. The difficulty therein is to capture and understand the dependencies of parallel resource assignment while taking into account the intricacies of transferability to unseen PQP on multi-core hardware resources.

S2: Transferable features. In our approach, we learn from features like data-partitioning schemes of an operator and CPU cores of the given hardware resource that allows ZeroTune to gain an understanding of the load distribution on the overall costs. Such transferable features represent the attribute space used by ZeroTune about the query operators and their instances with its deployment on hardware resources so that the model can infer relationships between them and transfer knowledge from seen to unseen classes. An example of a non-transferable feature is the name of data sources such as “temperature” instead of event rates, which would tie ZeroTune to a particular data stream of temperature readings. In Section III-B we further detail on how we select those transferable features.

S3: Parallel graph representation. In addition to the transferable features, it is important that the model learns from the structural relationships between the different parallel instances of operators and the partitioning of the data streams. For this, graph representation is a natural choice as it represents both “nodes” as operators with their transferable features and their cross dependencies as “edges” representing the data flow among parallel query operators. Our approach encodes a PQP as a Directed Acyclic Graph (DAG) suitable for GNN. In addition, the operator instance-resource mapping is denoted as an additional edge (dotted lines as shown in Figure 2). Within this structure, every operator or resource node in the PQP derives a hidden state, determined together by the input features transformed by an activation function encoded using a Multi-Layer Perceptron (MLP). As data flows through the graph, we employ a bottom-up message-passing mechanism to update these hidden states. The states are finally aggregated at the root node’s updated hidden state, which then serves as input to a final regression responsible for estimating relevant metrics. This generic graph representation, combined with transferable features, enables us to accurately derive costs for previously unseen PQP during inference. We discuss the challenges and design ideas of this graph mapping in Section II-C.

C3: Enormous training effort. Existing models have to be trained on a broad spectrum of query workloads (200k queries and up to 15 databases) [15]. It is hard to collect such a workload for stream processing, as the data characteristics of streams are known to be dynamic due to their continuous nature. The lack of open stream processing benchmarks makes it even harder to train such large models.

S4: Data-efficient Training. While it is important that the model learns from diverse training data: data stream, PQP and resource combinations; it is practically very costly if not impossible because of the aforementioned reasons. Thus, we propose a hybrid training strategy OptSample that collects workload by varying the data stream, query parameters like window size, and resource combinations from low- to high-end nodes while limiting the exploration to parallelism degrees of the PQP as derived based on the literature [12–19]. By doing this, we aim to achieve a generalizable model with less training data and time as we explain later in Section IV.

III. The ZeroTune Cost Model

As stated above, ZeroTune aims at predicting costs for the execution of unseen PQP on a resource combination not observed. One of the ideas to achieve this generalizability is finding the right set of transferable features that allows us to estimate cost metrics without observing the deployment of the PQP, as we explained next after defining the cost metrics.

A. Metrics for Cost Model

We consider two main cost metrics that are relevant for most streaming applications: end-to-end latency and throughput as defined in requirements for stream processing by Stonebraker et al. [21] but still hold validity in current applications [22–24]. Although we consider these metrics, our model can be fine-tuned for other cost metrics like resource usage and availability [25] depending on the application by simply replacing the final MLP node.

Definition 1: End-to-end latency ($L$): It represents the time interval starting from the production of the first data tuple from the data source ($S_o$) until when the output of the query result is delivered to the data sink ($S_i$). In other words, it is the sum of processing latency ($L_{proc}$) of each operator in the processing pipeline, network latency ($L_{net}$) of data transmission from a data source, within operators ($\omega \in \Omega$) to the data sink, and input ($L_{in}$) and output ($L_{out}$) latency of reading and writing data to and from external systems like IoT data sources. As operators may be allocated to resources at different locations, network latency is also considered.

Definition 2: Throughput ($T$): The second metric of our cost model is throughput, measured as the number of data records processed by the DSP system per unit of time [23].

B. Transferable Featureization

We define transferable features as those characteristics related to query, data stream, and hardware learned for the cost prediction task of PQP that can be effectively applied or transferred to cost prediction of an unseen query, data stream...
TABLE I: Transferable features to derive costs of PDP and hardware. In Table I and the following text, we define the selection of transferable features in three categories.

1) Operator Parallelism-related Features: The most important set of features for cost prediction are the ones related to the parallel execution. The most obvious feature is the parallelism degree of individual operators, which profoundly influences the execution cost of parallel processing systems. To gain a deeper understanding of the influence of parallelism degree, we conducted a micro-benchmark illustrated in Figure 3, where we observed how parallelism degree influences the cost metrics latency and throughput. We conducted this experiment on Apache Flink with a context-based job stream query where all other parameters besides parallelism degree were kept deterministic. Here, the input event rate is meant to achieve maximum utilization of the cluster while ensuring there is no backpressure (with parallelism degree equal to one)

Further, another major effect we observed that caused non-linearity, in particular, for high parallelism degrees is operator chaining (highlighted in blue). The DSP systems like Flink offer the capability to group multiple independent operators with the same parallelism degree. This grouping allows them to be placed on a single computation unit or physical node, resulting in enhanced performance by reducing interprocess communication overhead and effectively utilizing available processing cores. This specifically led to a significant reduction in latency and, consequently, growth in throughput. Therefore, our featureization approach takes into account both individual operators and grouping number that indicates the number of operators that are grouped together on nodes during query execution.

Further, the costs of a parallel query plan are influenced by partitioning schemes used by the operators as they dictate how the load is distributed among the different parallel instances of an operator. Thus, the partitioning strategy feature specifies kind of scheme that is applied. We support forward, rebalance, or hashing schemes in ZeroTune.

Fig. 3: Effect of selected features parallelism degree and operator grouping on the costs for a linear query. With the increasing parallelism, a decrease in latency and an increase in throughput are observed. For high parallelism degrees, the effect of operator grouping (highlighted in blue) can be observed, which leads to a sudden improvement in the costs.

2) Operator-related Features: Apart from parallelism-related features, we determine other transferable features that are data stream and operators-related because of their influence on runtime costs of PDP. For example, in the context of the query (step 0 in Figure 4), for the window operator we consider window types (tumbling or sliding window), window policy (time-based or count-based), window length, sliding length (for sliding window). Similarly, the features for the aggregation operator are the aggregation function and data type of the aggregate literal; features for the filter operator are the filter function, the data type of filter literal, and average selectivity, respectively, for each operator type.

Further, to reason about the performance of the given data model, we consider data stream-related features. In DSP, the data being processed by operators within a query is highly dynamic and differs a lot with time. Unlike traditional databases, where the data and its distribution are known in advance, DSP operates on streaming data with limited visibility before query execution. To optimize parallelism in DSP, we focus on transferable attributes of the data stream that can be generalized across different PDP. These attributes majorly include the event rate (data arrival rate) and tuple width (size of data elements). Since the given event rate dictates the processing required by the operator and hence the parallelism, by leveraging these transferable data attributes, we can effectively adapt the parallelism of operators to the dynamic nature of streaming data.

Another important data-related feature is the selectivity of operators, which refers to the proportion of data meeting specific conditions within an operator. Here, we consider the average selectivity of all the parallel instances of an operator (see next section), as different instances may be allocated to different physical nodes, resulting in varying performance impacts. For instance, a simple filter-aggregate query runs on two different data streams with the same input rate but different data characteristics. Without selectivity as an input feature, a cost model cannot derive the cost of the aggregate since it cannot differentiate between the computational complexity for both streams. Similar observations have been made in cost models for query runtime in DBMS [15].

3) Resource-related Features: Finally, a last crucial aspect to consider for cost estimation is the underlying parallel resources used for deployment. By identifying the hardware characteristics specific to parallelism, optimizing
the performance of DSP applications on different hardware configurations is possible. We identify fixed hardware specifications such as the number of processing cores and CPU speed as representative transferable features for the hardware types. All these features have a direct correlation to achievable parallelism and the overall cost in terms of latency and throughput. For instance, the available number of processing cores in a physical node directly impacts the level of parallelism that can be achieved for an operator. With more processing cores, a higher level of parallelism can be achieved, resulting in fast computation time, low processing latency, and high throughput.

C. ZeroTune model architecture

In Figure 4, we present ZeroTune’s model architecture. First, we define the training and inference of ZeroTune, including our novel parallel graph representation. In the end, we show how an optimizer based on cost predictions can be used for parallelism tuning.

1) Training and Inference: The zero-shot model is trained in an offline supervised manner using the queries depicted in a graph representation and encoding of the aforementioned transferable features in logical and physical nodes using a CNN model (step 0 and 1). Each graph node serves in CNN as a multi-layer perceptron (MLP) with the physical and logical nodes embedded into the input and hidden layers in a bottom-up manner (of the PQP) based on the proposed graph representation (step 2). The neural message passing takes place between the operator nodes (data-flow graph depicted in black edge) followed by between physical nodes (orange edge) and lastly on the operator-resource mapping (green edge) to receive the aggregated information on the sink where the predictions are read out (step 3). For example, given query in the figure, the hidden state of the node representing filter operator instance 1 ($\omega^1_1$) is obtained by feeding the feature vector $x_1$ (containing transferable features of data source) into an MLP which is shared by all the upstream nodes (grey nodes). The input and output of the hidden states are combined along the data flow and physical operator mapping using the message passing scheme as illustrated in the order of step 3 in Figure 4. This is important for the model to learn the runtime behavior of each instance of parallel operators on distinct hardware resources and vice versa to derive precise costs on them. Followed by message passing, the final output node ($y_1$) predicts latency or throughput at the inference (step 4).

2) Parallel Graph Representation: A key challenge in the graph representation of PQP is to encode the attribute space in the graph without increasing complexity and affecting the performance of the cost model. To achieve this, we explored two possibilities to encode the previously defined transferable features in our graph representation: (1) encode each operator instance and its features as a separate graph node or (2) encode only distinct operators separately while parallel operators together with parallelism-related features.

However, the problem with the first (1) design option is that the transferable features that the model learns about each parallel node (operator instance) are almost the same (if not entirely), which leads to many duplicate nodes in the graph with redundant information, not to mention the added complexity to the model (with parallel nodes and edges). For example, assuming maximum achievable parallelism on a machine is $\text{max}_P = 64$, then for a linear query, there will be 64 instances of filter operator and correspondingly 64 edges from single source ($\omega^P_{S_0} = 1$) to filter instances. Then there will be twice 128 edges from each filter to window aggregation ($\omega^P_{I_x} = 2$) which would lead to $64 \times \text{filter}$ and $64 \times \text{window aggregate}$, leading to an incredible amount
of 4096 edges for neural message passing. This leads to a highly complex neural model with hardly any new information per operator instance node. Therefore, instead, we opt for the second (2) approach to encode a parallel operator individually as a graph node with their corresponding logical and physical transferable features due to its simplicity and limited information gain in method (1) in comparison to its complexity. To encode transferable features of multiple instances in a single graph we aggregate information such as individual selectivities as average selectivities of all instances. For this, we conducted a microbenchmark where we analyzed individual selectivities of parallel instances for different partitioning schemes that resulted in very similar that support our decision.

Another challenge is to capture the resource mapping for each operator instance when we aggregate their transferable feature characteristics into a single graph node as discussed above. Although we aggregate information like selectivity, we maintain edge information for each parallel operator instance (e.g., $\omega_i$, here $i$=op. instance) as shown in the table on the operator-resource mapping in Figure 3 step 2. This is done to ensure that the model can reason about the cost overhead of each resource and instance type during the inference.

3) **Parallelism tuning with ZeroTune**: ZeroTune cost model can be combined with an optimizer as described in this section to determine parallelism such that the overall costs of query execution are minimal. In the following, we formally define the optimizer in a combinatorial optimization problem with the combined cost objective.

The goal of ZeroTune is to find a set of parallelism degrees $P_i$ for each operator $\omega_i \in \Omega$ in a given parallel query plan $PQP$ where the inferred costs $C$ for query execution on a selected set of resources $R$ are minimum. For enumerating parallelism degrees, we propose a novel OptiSample strategy as discussed in Section IV. We define the combined objective function for the **parallelism tuning problem** considering the two cost metrics (1) latency ($C_L$) and (2) throughput ($C_T$). Here, $C_L$ and $C_T$ are the normalized latency and throughput costs within the range $[0, 1]$, with 0 being the best and 1 being the worst. In normalization, we negate throughput because we want to maximize it, and the weight factor $w_t$ determines the relative importance of the two costs. Thus, in the parallelism tuning problem, we aim to minimize a weighted sum of latency and throughput costs to allow the selection of parallelism degrees with the lowest cost on the given hardware resources:

$$C = \arg \min_{C_L, C_T} \left[ w_t \cdot C_L + (1 - w_t) \cdot C_T \right]$$

s.t. for each operator $\omega_i$, the parallelism degree $P_i$ is always an integer $P_i \in \mathbb{Z}$ and must be more than equal to 1, i.e., $P_i \geq 1$, as well as maximum parallelism degree cannot exceed the total number of cores $n_{core}$ of all the given physical resources $R$, i.e., $P_{max_p} \leq n_{core}$.

IV. **Data-efficient Training**

A key aspect for generalization is that we train ZeroTune cost model with a broad spectrum of workload and physical resource characteristics to ensure that it can generalize to unseen workload and resources. This leads to the question **how to enumerate different PQP and its assignment on resources** for generalization. Prior work has shown that random sampling of training range leads to generalization for workloads and resources. However, random enumeration of parallelism degrees will result in noisy query plans that the learned model may not benefit from. For instance, setting parallelism degrees lower for upstream operators and higher for operators downstream in the query graph could lead to backpressure, adversely impacting query performance.

Moreover, exhaustive enumeration, even for a single query like a 2-way join, is impractical due to a vast number of combinations. For instance, when we use $N$ physical resources to place the query for the 2-way join with $9$ operators with arbitrary parallelism degrees $P = \{p_1, \ldots, p_{max_p}\}$ will result into $N \cdot |P_{max_p}|$ combinations, i.e., with nine operators and parallelism degrees: $1, 2, \ldots, 20$ results into $512 bn$ combinations of queries with $N = 1$ resource. To address this challenge, we propose OptiSample, an optimized sampling method that efficiently balances exploratory diversity (related to parallel query workload and resources) as well as analytical precision of parallelism degrees to generate a representative and manageable subset of these combinations.

**OptiSample Training Strategy.** Unlike the random approach, the OptiSample strategy offers a systematic way to determine the right number of processing instances (parallelism degree $P$) for each operator in the operator graph $G$ to collect realistic training data. To achieve this, the strategy utilizes the characteristics of workload (query and data stream) and physical resources, such as input event rate ($In_{ER}$), operator selectivity ($sel$), output rate ($Out_{ER}$) and the number of cores ($n_{core}$) to enumerate parallelism degrees for both upstream ($\omega_i$) and downstream ($\omega_j$) operators in the operator graph. The main intuition is that higher input rates and selectivities require higher parallelism degrees because of the computation required to process the input rate of events leading to better handling of backpressure. This is inspired by existing scaling controllers of known data-parallel DSP systems that select parallelism degrees based on these parameters.

In Algorithm 1 Lines 1-12, we provide an explanation of the OptiSample strategy that assigns parallelism degrees to operators in a bottom-up fashion. The OptiSample strategy takes the input event rate $In_{ER}(\omega_{So})$ of the source operator $\omega_{So}$ as an input and returns parallelism degrees of downstream operators as an output. The source operator ($\omega_{So}$) has no upstream operator, and the algorithm will finish traversing when the downstream operator ($\omega_j$) of the current (upstream) operator ($\omega_i$) is data sink ($S_j$). The parallelism for the source operator is based on the event rate specified by the application, so we assume it is known and assign the parallelism for the downstream operators. For each operator $\omega \in \Omega$, we use estimated selectivities and Equations (3) to (5) defined below and in the next step estimate the output rate of the current operator based on this.
The documents distributed by this server have been provided by the contributing authors as a means to ensure timely dissemination of scholarly and technical work on a non-commercial basis. Copyright and all rights therein are maintained by the authors or by other copyright holders, notwithstanding that they have offered their works here electronically. It is understood that all persons copying this information will adhere to the terms and constraints invoked by each author’s copyright. These works may not be reposted without the explicit permission of the copyright holder.
the flat vector as Flat Vector MLP. Finally, we train a random vector as a flat vector and train a linear model inspired by [4].

The idea of this baseline is to represent features like the number of different operator types, their selectivity, and parallelism degree (our addition) to other non-transferable architectures like flat vector representation from DBMS. The idea of this baseline is to measure the relative deviation of the true cost \( c \) (latency and throughput) and its prediction \( c' \), to assess \( \text{ZERO\textsc{TUNE}} \)’s accuracy and generalization. The model’s accuracy is evaluated in terms of q-error for latency and throughput, using OptiSample training procedure. Our experiments show q-error being mostly close to 1, indicating \( \text{ZERO\textsc{TUNE}} \)'s model's accuracy in cost predictions and its generalizability across unseen scenarios. For the evaluation of the optimizer, we used weighted cost (cf. Equation 1) and mean speed-up calculated as a fraction of improvement in latency or throughput in comparison to the baselines.

**Baselines:** Due to the lack of baselines that can provide generalizable cost predictions for DSP, we compared the \( \text{ZERO\textsc{TUNE}} \) to other non-transferable architectures like flat vector representation from DBMS. The idea of this baseline is to represent features like the number of different operator types, their selectivity, and parallelism degree (our addition) as a flat vector and train a linear model inspired by [4]. We represent this linear model as Linear Regression and the extension to deep neural network with MLPs that learn from the flat vector as Flat Vector MLP. Finally, we train a random forest model with the flat vector. Moreover, we compare against a greedy approach [20] and Dhalion [19], which incorporates analytical derivation of parallelism degrees, in terms of mean speed-ups, and weighted sum (Equation 1).

**A. Exp. 1: Accuracy on Seen-Unseen Workload**

In the first experiment, we evaluate the \( \text{ZERO\textsc{TUNE}} \) model’s prediction accuracy on seen query structures, i.e., within the range, followed by its generalization on unseen query structures and benchmarks as presented in Table III. We compare on measuring errors in estimating both latency and throughput using the median and 95th percentile q-errors as performance metrics, with 1.0 being a perfect estimate. We trained and evaluated our \( \text{ZERO\textsc{TUNE}} \) model in this experiment using the proposed OptiSample enumeration strategy to set the parallelism degrees and compared with different model architectures using flat vector [4] in Figure 5. In later experiments, we compare the two training strategies, random and OptiSample in Section V-D. In the following, we discuss the performance of \( \text{ZERO\textsc{TUNE}} \) on seen-unseen workloads.

1) **Seen Workload:** Table IV presents the study for the PQP having query structure - linear, 2-way, and 3-way joins that are within the range of the seen test set (cf. Table III). In addition, we included the “overall” accuracy across all query structures. As, we can observe consistently highly accurate cost predictions both for latency and throughput, showing the efficiency of \( \text{ZERO\textsc{TUNE}} \) trained using the proposed data-efficient OptiSample training strategy. Moreover, \( \text{ZERO\textsc{TUNE}} \) consistently outperformed all model architecture baselines incorporating flat vector representation, which do not learn from the structural information of the query graph that \( \text{ZERO\textsc{TUNE}} \) learns from (cf. Figure 5). In the remaining evaluations, we use the “overall” model training on all query types unless otherwise specified.

2) **Unseen Workloads:** To evaluate generalizability, we increased the complexity of the parallel query structures to ones that are not encountered during the training phase, e.g., 2-4 chained filters, 4-6 way joins, as shown in Table III.
TABLE IV: Q-errors (median, and 95th) of cost prediction of seen and unseen parallel query structures with synthetic dataset and public benchmarks. The proposed ZERO-TUNE model trained data efficiently provides very accurate costs both within the range and across unseen workloads and public benchmarks.

<table>
<thead>
<tr>
<th>Query Type</th>
<th>1 Increasing</th>
<th>2 Increasing</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Regression</td>
<td>1.21 1.82 1.24 2.31</td>
<td>1.37 1.82 1.62 2.05</td>
<td>1.30 1.35 1.37 1.82</td>
</tr>
<tr>
<td>Flat Vector MLP</td>
<td>1.38 1.89 1.72 2.70</td>
<td>1.38 1.89 1.72 2.70</td>
<td>1.38 1.89 1.72 2.70</td>
</tr>
<tr>
<td>Random Forest</td>
<td>1.30 1.35 1.37 1.82</td>
<td>1.30 1.35 1.37 1.82</td>
<td>1.30 1.35 1.37 1.82</td>
</tr>
<tr>
<td>ZeroTune</td>
<td>1.22 1.40 1.35 2.50</td>
<td>1.22 1.40 1.35 2.50</td>
<td>1.22 1.40 1.35 2.50</td>
</tr>
<tr>
<td>Spike Detection</td>
<td>1.25 1.36 1.37 1.96</td>
<td>1.25 1.36 1.37 1.96</td>
<td>1.25 1.36 1.37 1.96</td>
</tr>
<tr>
<td>Smart-grid (local)</td>
<td>1.33 2.00 1.64 3.81</td>
<td>1.33 2.00 1.64 3.81</td>
<td>1.33 2.00 1.64 3.81</td>
</tr>
<tr>
<td>Smart-grid (global)</td>
<td>1.44 1.60 1.52 2.40</td>
<td>1.44 1.60 1.52 2.40</td>
<td>1.44 1.60 1.52 2.40</td>
</tr>
</tbody>
</table>

Fig. 5: ZERO-TUNE model offers robust and generalized cost prediction for seen (white) and unseen (shaded) parallel query plans in contrast to other model architectures. Generating 200 test queries per query type for assessment. The results in Table IV 2 show that ZERO-TUNE’s accuracy and adaptability in predicting accurate cost both median and tail errors (95th percentile) for unseen parallel query structures. While the model excels in simple structures, e.g., 2-way joins, there are an increasing trend in q-errors as the complexity of the unseen parallel query structures increases (e.g., 6-way joins), especially for throughput prediction, as throughput increases dramatically, especially with the combinations of high parallelism degrees.

Few-shot: To further improve q-errors for such complex unseen parallel query plans, we used an ML technique called few-shot learning [30], where ZERO-TUNE is trained using a 500 training examples of complex join structures. Figure 6 reaffirms that the model remains robust to essential attributes by coping with the dynamic nature of DSP.

Furthermore, ZERO-TUNE consistently outperformed flat vector representation baselines also for unseen query structures due to the parallel graph-encoding method that effectively captures query complexities, demonstrating superior performance even with novel query structures (cf. Figure 5).

3) Unseen Queries from Public Benchmarks: In addition to synthetic queries, we evaluated ZERO-TUNE model using two public streaming benchmark queries: spike detection and smart-grid [24] [31] [32]. The spike detection benchmark [32] analyzes sensor data stream to identify spikes against a 2s moving average value. The smart-grid benchmark aims to predict energy consumption loads using data from smart plugs in private households. Here, energy consumption is calculated at local and global levels using two queries and a 10s sliding window with a 3s slide.

GENERATING 500 TEST QUERIES PER QUERY TYPE FOR ASSESSMENT. THE RESULTS IN TABLE IV 2 SHOW THAT ZERO-TUNE’S ACCURACY AND ADAPTABILITY IN PREDICTING ACCURATE COST BOTH MEDIAN AND TAIL ERRORS (95TH PERCENTILE) FOR UNSEEN PARALLEL QUERY STRUCTURES. WHILE THE MODEL EXCELS IN SIMPLE STRUCTURES, E.G., 2-WAY JOINS, THERE ARE AN INCREASING TREND IN Q-ERRORS AS THE COMPLEXITY OF THE UNSEEN PARALLEL QUERY STRUCTURES INCREASES (E.G., 6-WAY JOINS), ESPECIALLY FOR THROUGHPUT PREDICTION, AS THROUGHPUT INCREASES DRAMATICALLY, ESPECIALLY WITH THE COMBINATIONS OF HIGH PARALLELISM DEGREES.

Few-shot: To further improve q-errors for such complex unseen parallel query plans, we used an ML technique called few-shot learning [30], where ZERO-TUNE is trained using a 500 training examples of complex join structures. Figure 6 reaffirms that the model remains robust to essential attributes by coping with the dynamic nature of DSP.

Furthermore, ZERO-TUNE consistently outperformed flat vector representation baselines also for unseen query structures due to the parallel graph-encoding method that effectively captures query complexities, demonstrating superior performance even with novel query structures (cf. Figure 5).

3) Unseen Queries from Public Benchmarks: In addition to synthetic queries, we evaluated ZERO-TUNE model using two public streaming benchmark queries: spike detection and smart-grid [24] [31] [32]. The spike detection benchmark [32] analyzes sensor data stream to identify spikes against a 2s moving average value. The smart-grid benchmark aims to predict energy consumption loads using data from smart plugs in private households. Here, energy consumption is calculated at local and global levels using two queries and a 10s sliding window with a 3s slide.
In this section, we evaluate the generalization ability of ZeroTune model to predict cost across various workload parameters. To achieve this, we interpolate and extrapolate in the range presented in Table III Figure 8 illustrates the impact of several key parameters, including tuple width, window configurations, and amount of available workers in a cluster, which are selected, because of their direct relation to the parallelism degrees and the costs. For instance, the event rate is a natural fit due to the backpressure effect, but also window configurations because of the parallel processing of data stream comprising different key-based windows. We use the model trained on linear, 2-, and 3-way join query structures to evaluate model on inter- and extrapolation ranges.

**Tuple Widths.** To evaluate the model’s accuracy, we extrapolate the impact of tuple widths, potentially benefiting from higher parallelism degrees, and tested the ZeroTune model from 6 – 15 tuple widths. We used at least 165 queries per tuple width, ensuring an equal distribution between linear queries, 2- and 3-way join queries. In Section V-B, the model demonstrates stable performance and generalization to unseen tuple widths (shaded), suggesting its effective learning of the correlations between tuple width and cost in parallel query structures.

**Event Rates.** In Section V-B (log scale on the x-axis), we analyzed the ZeroTune model’s ability to inter- and extrapolate event rates beyond the training range (cf Table III). The model shows high accuracy in predicting costs for varying event rates, both low and high event rates within (white) and outside (shaded) the training range. Its proficiency in handling higher event rates is due to its learned understanding of the system’s processing limitations and backpressure effects at full hardware capacity. However, at very low event rates, the q-error values slightly increase, likely due to the model’s limited exposure to such low-utilization scenarios, where less data processing leads to resource under-utilization. Overall, the ZeroTune demonstrates very good performance and generalization for both seen and unseen event rates in terms of predicting latency and throughput.

**Window Durations (Time-based).** Section V-B (log scale on the x-axis), shows the ZeroTune’s ability to accurately predict latency and throughput across varying parallelism categories. Notably, the ZeroTune model again exhibits reasonably accurate predictions across all parallelism categories with relatively small variations, suggesting that the model can effectively capture the performance patterns and generalize well to unseen query plans within this template structure. However, a slight decrease is noted for more complex parallel query plans, especially in throughput for 5-way, and 6-way join, as per the q-error trends observed in Table IV. To address this, we enhanced performance using an additional dataset of joins (500 queries). Using few-shot learning as shown in Figure 7d, it significantly amplifies the prediction accuracy in both median and tail errors in comparison to zero-shot learning and adapts more effectively to each parallelism category XS and XL, aligned with the overall results (cf. Figure 6).

C. Exp. 3: Generalization for Unseen Parameters

After analyzing the ZeroTune model’s generalization ability to unseen parameters, we evaluated the model’s performance on unseen configurations of heterogeneous and homogeneous resources and the impact of parallelism degrees, analyzed the model’s ability to handle unseen query structures, and compared the performance and generalization to ZeroTune model trained on the same configurations. The results indicate that the ZeroTune model consistently delivers accurate cost predictions also for unseen homogeneous and heterogeneous hardware and (d) comparison of unseen query benchmark.

![Graphs showing median q-error for seen query structures, median q-error for unseen query benchmark, median q-error for unseen homogeneous and heterogeneous resources, and median q-error for unseen query structures.](image)
predict costs over a range of window durations (cf. Table III). The model initially shows slightly high median q-errors for smaller unseen windows but overall, it exhibits comparable good performance as the duration increases. As small windows lead to rapid data processing and high data turnover, it challenges the model to accurately capture the performance characteristics and predict the associated costs when the model has not seen smaller windows at all (the first seen range is 250ms). In contrast, longer windows allow more data accumulation before processing, aiding the models to better understand the patterns of the system, thereby higher accuracy in estimating costs for longer windows. However, as the window increases towards the end of the unseen range, variations in the accuracy of the model are observed, possibly due to fewer training examples. Overall, the model generalizes very well for longer windows in particular and provides reasonably well predictions for small windows.

**Window Lengths (Count-based).** Like window duration, window length also influences the cost of PQP with distinct impacts for time-based and count-based windows on throughput. Time-based windows typically maintain constant throughput, unaffected by incoming event rates, while count-based windows’ throughput is directly related to the input rate of the operator. For instance, a tumbling count-based window with a window length of 10 will reduce the outgoing event rate to 10% of the incoming event rate, making throughput prediction harder for unseen window sizes for count-based windows. Section V-B (note log scale on the x-axis) shows the ZeroTune model’s accuracy and generalization predicting costs for seen and unseen window lengths. However, extremely low and higher unseen window length shows a slight increase in q-error (but only for throughput) due to the reasons mentioned before.

**Amount of Workers.** The number of workers or nodes in the DSP system plays a critical influence on cost. For instance, smaller cluster sizes may have limited parallelism due to fewer available workers, while larger cluster sizes can handle a higher number of parallel tasks. In the evaluation results shown in Section V-B, we show that the model consistently gives accurate cost predictions for varying cluster sizes, allowing it to generate parallel query plans with increased parallelism as the cluster size grows.

**Exp. 4: Data-efficient Training**

In Figure 9, we compare the accuracy of our ZeroTune models trained with random data: ZT-Random and OptiSample.

---

The documents distributed by this server have been provided by the contributing authors as a means to ensure timely dissemination of scholarly and technical work on a non-commercial basis. Copyright and all rights therein are maintained by the authors or by other copyright holders, notwithstanding that they have offered their works here electronically. It is understood that all persons copying this information will adhere to the terms and constraints invoked by each author’s copyright. These works may not be reposted without the explicit permission of the copyright holder.
We analyze previous work for parallelism tuning in [DSP] systems as (1) non-learned and (2) learned approaches.

**Non-learned Approaches:** Modern [DSP] systems like Apache Flink [26], Heron [33], Storm [34] and Spark [35] offer parallelism degree as a manual knob to initially tune based on the workload and quality requirements. However, manual tuning is known to be unreliable and susceptible to changes in the workload. To tackle this issue, researchers have intensively applied various online analytical techniques to monitor system performance [54-57] and determine the optimal parallelism degree. For instance, monitoring-based heuristic approaches [12, 19, 53-54] extensively use parameters such as observed rate, processing rate, CPU utilization, etc., to determine optimal parallelism degree. Similarly, other mathematical optimization approaches like [12, 43], ILP [25], game theory [44], queueing-theory [45-47] have been proposed for parallelism tuning. Some works have used analytical approaches for resource estimation [48, 49] and parallelism adjustment [47, 50]. While these approaches provide some level of semi-automation for parallelism tuning, they cannot be used to determine initial optimal parallelism because they rely on query runtime observations that make them unsuitable. This results in sub-optimal decision-making as well as high monitoring overhead in converging to an optimal parallelism degree.

**Learned Approaches:** Recently, there has been interest in developing more learned approaches for parallelism tuning in [DSP] systems [5, 8-11, 51, 52]. For instance, some works have explored parallelism using stream partitioning prediction [11] that is an orthogonal direction. An alternative prominent technique that has gained popularity is incorporating elasticity into the [DSP] system by allocating or deallocating resources to accommodate accelerating workloads [5, 10]. There has been interest in automated parallelism tuning in recent work [6, 53]. Although these studies have shown substantial performance improvements using machine learning, they train on non-transferable features and hence lack generalization across unseen workloads, as done in this work.

## VII. Conclusion and Future Work

In this paper, we propose ZeroTune, a novel data-efficient zero-shot learned approach for performance prediction of [PQP] in [DSP] systems. We propose novel transferable features related to [PQP] and a parallel graph representation to enable generalization for unseen workloads and resources. Our experiments show that ZeroTune achieves an average speed-up of around 5× in recommending optimal parallelism degrees compared to existing approaches and outperforms analytical methods by accurately predicting query performance for different parallelism degrees. Moreover, by training ZeroTune using a novel data-efficient training strategy OptiSample, we significantly reduce the training effort by 4× for generalization.

**Acknowledgements**

This research is funded by the DFG as part of project C2 within the Collaborative Research Center (CRC) 1053–MAKI; BMBF and the state of Hesse as part of the NHR Program and HMWK cluster project 3AI (The Third Wave of AI). We also want to thank hessian.AI at TU Darmstadt and DFKI Darmstadt.

**Fig. 10:** (a) Mean speed-up using parallelism tuning by ZeroTune compared to a greedy heuristic [20], showing high speed-up in latency and throughput. (b) Comparative performance of query types with parallelism degrees determined by ZeroTune versus Dhalion [19]. The plot shows weighted average runtimes, with ZeroTune surpassing Dhalion in both seen and unseen queries.

**Fig. 11:** Feature ablation when using only (1) operator-, (2) parallelism- and resource-related features, and (3) all features for latency cost prediction.

compared the performance of our model with Dhalion [19], the well-known and widely used traditional (non-learned) parallelism tuning algorithm for Heron [53]. Dhalion is a state-of-the-art auto-scaling controller with wide acceptance both in academia and industry (Twitter Heron), making it a suitable comparison. Moreover, ZeroTune outperforming the non-transferable representations like flat-vector in the previous experiments encouraged us to look into these well-established auto-scalers.

The results in Figure 10, indicating the weighted average cost (cf. Equation 1), shows that Dhalion performs relatively well for similar and simple query structures, which is the design focus, its effectiveness declines with increasing complexity of parallel queries. In contrast, our ZeroTune model consistently determines cost-effective parallelism degrees for both seen and unseen queries, regardless of the query complexity, outperforming Dhalion’s tuning algorithm.

**F. Exp. 6: Ablation Study**

We perform a feature ablation to identify and quantify the contribution of each transferable feature to the generalization performance of the model.

Figure 11 shows that while operator-specific features such as average selectivity, enrich the model’s understanding of data processing, they do not alone significantly improve the accuracy of predictions for parallel query plans. However, integrating these with parallelism-specific features such as parallelism degrees, enhances model performance by linking data processing characteristics with parallel execution efficiency and resource utilization, thereby significantly improving the model’s efficiency and generalization to predict costs accurately for both seen and unseen parallel query plans.

**VI. RELATED WORK**

We analyze previous work for parallelism tuning in [DSP] systems as (1) non-learned and (2) learned approaches.
REFERENCES


