Pratyush Agnihotri, Boris Koldehofe, Paul Stiegele, Roman Heinrich, Carsten Binnig, Manisha Luthra. ZeroTune: Learned Zero-Shot Cost Model for Parallelism Tuning in Stream Processing. To Appear in Proceedings of the IEEE 40th International Conference on Data Engineering (ICDE), 14 pages

ZeroTune: Learned Zero-Shot Cost Models for Parallelism Tuning in Stream Processing

Pratyush Agnihotri*, Boris Koldehofe†, Paul Stiegele*, Roman Heinrich§, Carsten Binnig*‡, Manisha Luthra*‡
*Technische Universität Darmstadt, †Technische Universität Ilmenau, ‡DFKI Darmstadt, §DHBW Mannheim

***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 $5\times$ 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* [1] or online gaming providers *King* [2] use DSP for their core operations [3]. 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.

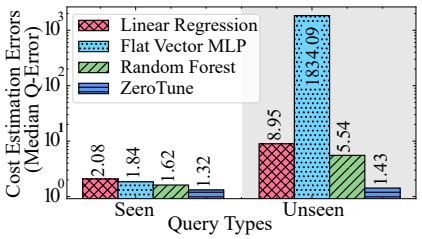


Fig. 1: ZeroTune model offers robust and generalized cost prediction of latency for seen and (even better for) unseen parallel query plans in contrast to other *non-transferable* model architectures [4].

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 [6]. 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[1] cost model. In this work, we propose ZeroTune, that can already *initially* configure parallelism degrees based on cost estimations of parallel

[1]Source code: https://github.com/pratyushagnihotri/ZeroTune

Pratyush Agnihotri, Boris Koldehofe, Paul Stiegele, Roman Heinrich, Carsten Binnig, Manisha Luthra. ZeroTune: Learned Zero-Shot Cost Model for Parallelism Tuning in Stream Processing. To Appear in Proceedings of the IEEE 40th International Conference on Data Engineering (ICDE), 14 pages

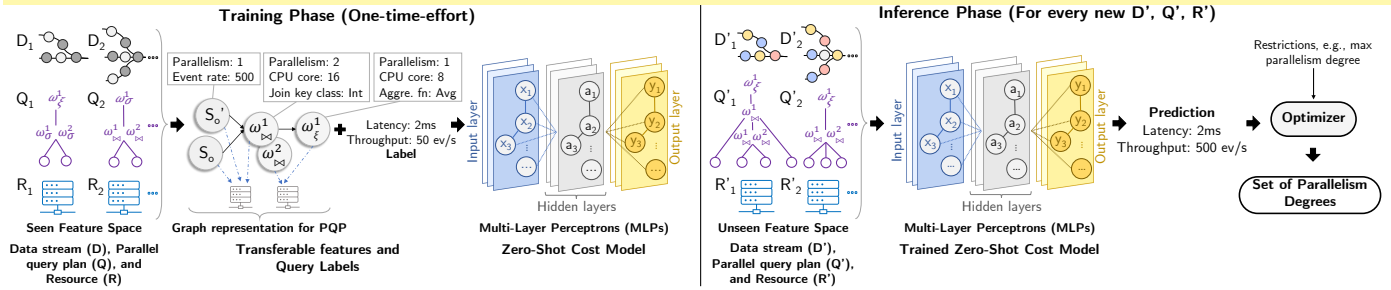


Fig. 2: Overview of ZeroTune for cost prediction that can be used in the parallelism tuning task. We train ZeroTune (see left fig.) using transferable features of a wide range of streaming workloads with different parallelism degrees. Once the model is trained, it can be used in *the inference phase* (see right fig.) to accurately estimate the cost of unseen parallelism degrees without requiring costly retraining of the model. Based on the costs, an optimizer is used to select the parallelism degree that has the minimum cost.

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 "$\leq$" 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 $1000\times$ 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 $4\times$ 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.

Pratyush Agnihotri, Boris Koldehofe, Paul Stiegele, Roman Heinrich, Carsten Binnig, Manisha Luthra. ZeroTune: Learned Zero-Shot Cost Model for Parallelism Tuning in Stream Processing. To Appear in Proceedings of the IEEE 40th International Conference on Data Engineering (ICDE), 14 pages

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 III-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 OptiSample 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[2] 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_{nw}) 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 Featurization

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

[2]A concept of ML where model's parameters are updated using a small amount of labeled data from target metric.

Pratyush Agnihotri, Boris Koldehofe, Paul Stiegele, Roman Heinrich, Carsten Binnig, Manisha Luthra. ZeroTune: Learned Zero-Shot Cost Model for Parallelism Tuning in Stream Processing. To Appear in Proceedings of the IEEE 40th International Conference on Data Engineering (ICDE), 14 pages

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
	data	Tuple width in	Input tuple width
	data	Tuple width out	Output tuple width
	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
	operator	Operator type	Type of operator
	operator	Filter function	Comparison filter function, e.g., $<, \leq, \geq$
	operator	Filter literal class	Filter literal datatype for comparison, e.g., int
	operator	Window type	Shifting strategy (tumbling/sliding)
	operator	Window policy	Windowing strategy (count/time)
	operator	Window length	Size of the window
	operator	Sliding length	Size of the sliding interval
	operator	Join key class	Join key data type
	operator	Agg. class	Aggregation data type
	operator	Agg. function	Aggregation function, e.g., *min, max, avg*
	operator	Agg. key class	Aggregation key 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	Total memory	Available memory of the node
	resource	Network speed	Network link speed between nodes

TABLE I: Transferable features to derive costs of PQP.

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 of PQP are the ones related to the parallel execution. The most obvious key feature is the *parallelism degree* of individual operators, which profoundly influences the execution cost of PQP in a DSP system. 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 count-based tumbling window 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 increasing parallelism). Intuitively, we saw a decreasing trend for latency and an increasing trend for throughput with the increase in parallelism degrees.

However, another major effect we observed that caused *non-linearity*, in particular, for high parallelism degrees is *operator chaining* (highlighted in blue). DSP systems like Flink offer the capability to group multiple independent operators with the same parallelism degrees. 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 featurization 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 the kind of scheme that is applied. We support forward, rebalance, or hashing schemes [26] in ZeroTune.

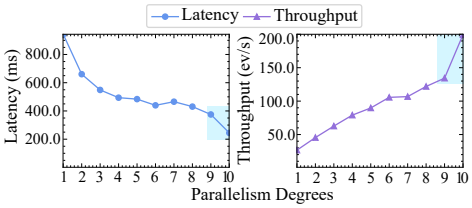


Fig. 3: Effect of selected features parallelism degree and operator grouping on the costs for a linear query. With the increasing parallelism, a decrease in latency and an increase in throughput are observed. For high parallelism degrees, the effect of operator grouping (highlighted in blue) can be observed,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

2) ***Operator-related Features:*** Apart from parallelism-features, we determine other *transferable* features that are data stream and operators-related because of their influence on runtime costs of PQP. 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 PQP. 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 of PQP is the underlying parallel resources used for deployment. By identifying the hardware characteristics specific to parallelism, optimizing

Pratyush Agnihotri, Boris Koldehofe, Paul Stiegele, Roman Heinrich, Carsten Binnig, Manisha Luthra. ZeroTune: Learned Zero-Shot Cost Model for Parallelism Tuning in Stream Processing. To Appear in Proceedings of the IEEE 40th International Conference on Data Engineering (ICDE), 14 pages

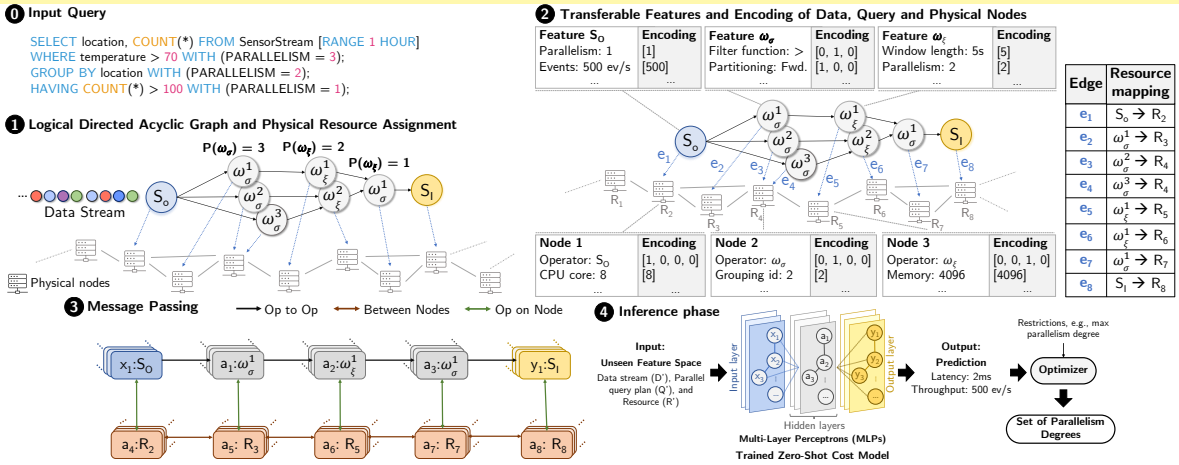


Fig. 4: ZeroTune model architecture detailed view on the training, inference, and optimizer for parallelism recommendation. For a given arbitrary PQP, e.g., step ⓪ to get locations with high temperature, ZeroTune ① transforms it into a physical execution plan dictating the resource assignment. The plan is enumerated for different parallelism degrees based on the proposed strategy with the deployment on physical resources. Step ② shows how the query is represented as a graph structure with different operators and resources as node types and their transferable feature representation that allows generalization across unseen workloads. ③ Afterwards, the resulting feature vectors of every operator, including its features, are encoded into a node-specific MLP to obtain a hidden state which is propagated through the graph to the sink using the specified order in the graph of the query. The final MLP is used to predict the cost of query execution. ④ Finally, the optimizer uses the *what-if* costs of the parallel query plan to determine the optimal set of parallelism degrees.

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 GNN model (step ⓪ and ①). Each graph node serves in GNN 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 (ω_{σ}^{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 $max_P = 64$, then for a linear query, there will be 64 instances of filter operator and correspondingly 64 edges from single source ($\omega_{So}^{P} = 1$) to filter instances. Then there will be twice 128 edges from each filter to window aggregation ($\omega_{\xi}^{P} = 2$) which would lead to 64× filter and 64× window aggregate, leading to an incredible amount

Pratyush Agnihotri, Boris Koldehofe, Paul Stiegele, Roman Heinrich, Carsten Binnig, Manisha Luthra. ZeroTune: Learned Zero-Shot Cost Model for Parallelism Tuning in Stream Processing. To Appear in Proceedings of the IEEE 40th International Conference on Data Engineering (ICDE), 14 pages

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., ω^i_σ, here i=op. instance) as shown in the table on the operator-resource mapping in Figure 4 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 wt 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:

$$C = \underset{C_L, C_T}{\arg\min}[wt \cdot C_L + (1 - wt) \cdot C_T] \quad (1)$$

s.t. for each operator ω_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 [15]. 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 $N \cdot |p^{|\omega|}_{max_P}|$ 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 [20], 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 (ω_i) and downstream (ω_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 [12, 19] 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 ω_{So} as an input and returns parallelism degrees of downstream operators as an output. The source operator (ω_{So}) has no upstream operator, and the algorithm will finish traversing when the downstream operator (ω_j) of the current (upstream) operator (ω_i) is data sink (S_I). 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 [27] (Lines 3-4 and Equations (3) to (5) defined below) and in the next step estimate the output rate of the current operator based on this

Pratyush Agnihotri, Boris Koldehofe, Paul Stiegele, Roman Heinrich, Carsten Binnig, Manisha Luthra. ZeroTune: Learned Zero-Shot Cost Model for Parallelism Tuning in Stream Processing. To Appear in Proceedings of the IEEE 40th International Conference on Data Engineering (ICDE), 14 pages

Algorithm 1: OptiSample Strategy for Training.

```
input : input source event rate In_ER(ω_So)
        Operator graph comprising set of operators ω ∈ Ω
output: Parallelism degree P(ω)
1  forall ω_i ∈ Ω do
2      if ω_i is not ω_So then
3          ω_j = ω_i.downstream() ;
4          CurrentSelectivity(ω_i) =
             EstSelectivity(ω_i) Definitions 4 to 6;
5          Out_ER(ω_i) =
             CurrentSelectivity(ω_i) · In_ER(ω_i) Definition 3;
6          In_ER(ω_j) = Out_ER(ω_i);
7          if ω_i.upstream() is ω_So then
8              P(ω_i) =
                 EstParallelism(ω_i, In_ER(ω_So)) Definition 7;
9          else
10             P(ω_i) = EstParallelism(ω_i, In_ER(ω_i)) Definition 8;
11     else
12         In_ER(ω_So) = ComputeSourceER(ω_So)
```

selectivity (Line 5). The output rate of the current operator is the input rate of the downstream operator (Line 6).

Later, based on the input rate and hence the selectivity of the upstream operator, the parallelism of the current operator is computed (Lines 8–10). In case the current operator is a source operator (Line 11), then its input rate will be equivalent to that of the source (Line 12) which is used in assigning parallelism of an operator whose predecessor is a source operator (Line 7). In the following, we formally present the definitions required for the above algorithm.

Definition 3: Output rate of an arbitrary operator ω: The output rate $Out_{ER}(\omega)$ is determined based on the input rate it receives from the upstream operator ($Out_{ER}(\omega_i)$) or source ($In_{ER}(\omega_{So})$) when there is no operator together with the selectivity. Formally, given an upstream ω_i and downstream operator ω_j the output rates are:

$$Out_{ER}(\omega_i) = In_{ER}(\omega_i) \cdot \text{sel}(\omega_i), \text{ where } \omega_i = (\omega_\sigma, \omega_{\bowtie}, \omega_\xi), \quad (2)$$

$$Out_{ER}(\omega_j) = \begin{cases} Out_{ER}(\omega_i) \cdot \text{sel}(\omega_j) & \text{, if } \omega_i = (\omega_\sigma, \omega_{\bowtie}, \omega_\xi) \\ In_{ER}(\omega_{So}) \cdot \text{sel}(\omega_j) & \text{, if } \omega_i = \omega_{So}. \end{cases}$$

Definition 4: Selectivity of filter operator ω_σ: The selectivity $sel(\omega_\sigma)$ of the filter operator ω_σ is determined as the fraction of the number of filtered tuples $|f_{\omega_\sigma}(D_{In})|$ that satisfy the filter function to the total number of input tuples $|D_{In}|$ from input data stream D, as given by the equation:

$$sel(\omega_\sigma) = \frac{|f_{\omega_\sigma}(D_{In})|}{|D_{In}|}, \quad \text{with } 0 \leq sel(\omega_\sigma) \leq 1. \quad (3)$$

Definition 5: Selectivity of the operator $\omega_{\bowtie}$: The selectivity $\text{sel}(\omega_{\bowtie})$ of the join operator $\omega_{\bowtie}$ is determined based on the resulting join partners by joining tuples from two windows $|W_{D1_{In}} \bowtie W_{D2_{In}}|$ on the cartesian product of all the resulting join tuples in the input windows $|W_{D1_{In}}|$ and $|W_{D2_{In}}|$ of the join input relations from windows $W_{D1_{In}}$ and $W_{D2_{In}}$ from two input streams D_1 and D_2, formally:

$$sel(\omega_{\bowtie}) = \frac{|W_{D1_{In}} \bowtie W_{D2_{In}}|}{|W_{D1_{In}}| \times |W_{D2_{In}}|}, \quad \text{with } 0 \leq sel(\omega_{\bowtie}) \leq 1. \quad (4)$$

Definition 6: Selectivity of aggregation operator ω_ξ: The selectivity $\text{sel}(\omega_\xi)$ of the window aggregate operator ω_ξ is determined based on the number of distinct group-by tuples in the window of the input stream $W_{D_{In}}$ on the total number of tuples in the window $|W_{D_{In}}|$, as given by the equation:

$$sel(\omega_\xi) = \frac{|\textit{group-by } (W_{D_{In}})|}{|W_{D_{In}}|}, \quad \text{with } 0 \leq sel(\omega_\xi) \leq 1. \quad (5)$$

Based on the estimated selectivities, input and output rates, we define how we select parallelism degrees for training.

Definition 7: Parallelism degree of an arbitrary upstream operator ω_i: The parallelism of the upstream operator $P(\omega_i)$ is estimated based on a scaling factor[3] (sf) and the input rate to the upstream operator $In_{ER}(\omega_i)$.

$$P(\omega_i) = sf \cdot In_{ER}(\omega_i) \quad (6)$$

Definition 8: Parallelism degree of an arbitrary downstream operator ω_j *after* ω_i: The parallelism of the downstream operator $P(\omega_j)$ is determined based on a scaling factor and the output rate $Out_{ER}(\omega_i)$ of the previous upstream operator ω_i, as given by the following equation:

$$\begin{aligned} P(\omega_j) &= sf \cdot In_{ER}(\omega_j) \\ &= sf \cdot Out_{ER}(\omega_i) \\ &= sf \cdot In_{ER}(\omega_i) \cdot \text{sel}(\omega_i) \text{ (By Equation (2))} \end{aligned} \quad (7)$$

Following constraints hold true for Equation (6) and (7), for each up- and downstream operator $\omega_j, \omega_i \leq max_P$ and maximum parallelism max_P cannot exceed the number of cores n_{core} in a physical resource it runs on i.e., $\text{max}_P \leq \text{n}_{core}$. Also, note that the selection of parallelism degree for the training data is based on *estimated* selectivities and output rates, and *does not* take actual observed values into account, so the estimations are not perfect. This is to maintain a better trade-off between exploration and exploitation in the training data, such that the model also learns about inefficient PQP when the estimations are off.

V. Experimental Evaluation

In this section, we present the evaluation results of ZeroTune with an aim to assess its efficiency and generalization capabilities on unseen workloads and resources. In the following, we specify the evaluation questions.

- **Exp. 1: Accuracy on seen-unseen workloads.** How accurate are ZeroTune's cost predictions for seen and unseen parallel query structures and benchmarks?
- **Exp. 2: Generalization performance on fine-grained parallelism.** How does the ZeroTune perform for the different parallelism degree ranges?
- **Exp. 3: Generalization for unseen parameters.** How does the ZeroTune predict the cost for unseen parameters such as tuple width?
- **Exp. 4: Data-efficient training.** Is OptiSample training procedure data- and resource-efficient?
- **Exp. 5: Optimizer for parallelism.** Is the ZeroTune, in combination with the optimizer, able to determine the set of parallelism degrees that minimize cost?
- **Exp. 6: Feature ablation study.** Finally, if the transferable features selected by ZeroTune contributes to generalization?

[3]determined by empirically analysing when and how the given streaming operators are backpressured.

Pratyush Agnihotri, Boris Koldehofe, Paul Stiegele, Roman Heinrich, Carsten Binnig, Manisha Luthra. ZeroTune: Learned Zero-Shot Cost Model for Parallelism Tuning in Stream Processing. To Appear in Proceedings of the IEEE 40th International Conference on Data Engineering (ICDE), 14 pages

Cluster Type	Clusters	Nodes	CPU	RAM (GB)	Disk (GB)	Intel Processor	Speed (Ghz)	S/U
Ho	m510	270	8	64	256	Xeon D	2.0	S
Ho	c6420	72	32	384	1024	Skylake	2.6	U
He	rs620	38	8-10	128-384	900	Xeon	2.2	S
He	c8220x	4	20	256	4096	Ivy Bridge	2.2	U
He	c8220	96	20	256	2048	Ivy Bridge	2.2	U
He	dss7500	2	12	128	120	Haswell	2.4	U
He	c6320	84	28	256	1024	Haswell	2.0	U
He	rs6525	32	64	256	1600	AMD EPYC	2.8	U

TABLE II: Cluster of resources utilized on CloudLab tested to perform evaluations. Here, "Ho" and "He" are homogeneous and heterogeneous; "S" and "U" are seen and unseen, respectively. We used a combination of these resources for data generation, training (S), and inference (S and U) on the model.

Evaluation Setup and Implementation: As DSP system we used Apache Flink v.1.16 and collected the training and testing benchmark by implementing a PQP query generator on top of it for various parameters summarized in Table III, focusing on the training range using the seen cluster nodes in Table II. We used the CloudLab research testbed [28] to perform all our experiments as it provided the necessary distributed infrastructure for configuring and deploying an Apache Flink [26] cluster, enabling us to conduct experiments and assess the performance of ZeroTune. We used OptiSample as a training strategy for ZeroTune and compare it with random wherever explicitly specified (cf. Section IV). For cluster management and task placement, we used Apache Flink's task manager with Kubernetes for orchestration. Moreover, we used Apache Flink's fairness policy and job distribution capabilities to ensure efficient resource allocation and optimized performance.

Training and Testing Parameters: Table III presents the train and test ranges for evaluating ZeroTune model for the above questions, featuring a dataset of 24,000 synthetic queries (linear, 2-, and 3-way joins) with 8,000 queries for each query type. This dataset is split into train (80%), test (10%), and validation (10%) sets. We use the Q-error metric $q(c, c')$ *where* $q \geq 1$, a well-established metric [29] to measure the relative deviation of the true cost c (latency and throughput) and its prediction c', to assess ZeroTune'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 ZeroTune 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 ZeroTune 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)).

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	-	$1 \leq XS < 8, 8 \leq S < 16, 16 \leq M < 32, 32 \leq L < 64, 64 \leq XL < 128$	

TABLE III: Training and testing range for data generation and inference, respectively. The seen range is used to evaluate model performance in general with a classical training-test split, while the unseen range is used for testing the model's generalizability. Here, extrapolation parameters are underlined while the remaining unseen are interpolation parameters.

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

In the first experiment, we evaluate the ZeroTune 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 focused 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 ZeroTune 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 ZeroTune on seen-unseen workloads.

*1) **Seen Workload:*** Table IV: ① 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 ZeroTune trained using the proposed data-efficient OptiSample training strategy. Moreover, ZeroTune consistently outperformed all model architecture baselines incorporating flat vector representation, which do not learn from the structural information of the query graph that ZeroTune 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,

Pratyush Agnihotri, Boris Koldehofe, Paul Stiegele, Roman Heinrich, Carsten Binnig, Manisha Luthra. ZeroTune: Learned Zero-Shot Cost Model for Parallelism Tuning in Stream Processing. To Appear in Proceedings of the IEEE 40th International Conference on Data Engineering (ICDE), 14 pages

Trained Model ZeroTune-OptiSample	Query Structure	Q-error (Latency) Median	Q-error (Latency) 95th	Q-error (Tpt) Median	Q-error (Tpt) 95th
① Seen workload	Linear	**1.21**	2.51	**1.24**	2.31
	2-way-join	**1.37**	3.84	**1.82**	8.05
	3-way-join	**1.38**	3.35	**1.89**	7.20
	Overall	**1.30**	3.35	**1.57**	6.82
② Unseen workload	2-filter-chained	**1.22**	2.15	**1.40**	3.50
	3-filter-chained	**1.24**	2.55	**1.57**	3.96
	4-filter-chained	**1.24**	2.90	**1.64**	5.31
	4-way-join	**1.34**	2.92	**1.92**	6.55
	5-way-join	**1.56**	3.6	**2.93**	16.82
	6-way-join	**1.95**	6.8	**6.19**	36.29
③ Unseen benchmark	Spike Detection	**1.29**	2.40	**2.73**	5.99
	Smart-grid (local)	**1.33**	1.50	**2.30**	4.44
	Smart-grid (global)	**1.44**	1.60	**1.52**	2.40

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 ZeroTune model trained data efficiently provides very accurate costs both within the range and across unseen workloads and public benchmarks.

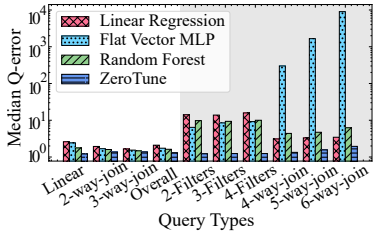


Fig. 5: ZeroTune model offers robust and generalized cost prediction for seen (white) and unseen (shaded) parallel query plans in contrast to other model architectures.

generating 200 test queries per query type for assessment. The results in Table IV: ② show that ZeroTune'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., filter chains, we see an increasing trend in q-errors as the complexity of the unseen parallel query structures increases (e.g., for 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 ZeroTune 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, ZeroTune 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 ZeroTune 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.

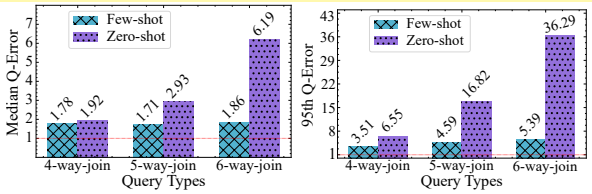


Fig. 6: Few-shot learning using only 500 training queries of unseen joins (all 4-, 5- and 6-way) improves the throughput prediction in ZeroTune by almost 6× (for 6-way join). Red line shows the perfect estimate.

Table IV: ③ shows the accuracy of ZeroTune model on these benchmarks in predicting latency and throughput for unseen parallel query structures. Overall, while estimations for both metrics are highly accurate, the results indicate that latency estimates are better than throughput, which is a general trend in all models. This is because throughput is directly affected by the incoming data distribution, while latency is influenced by indirect factors such as overall system utilization or varying durations for filling count windows.

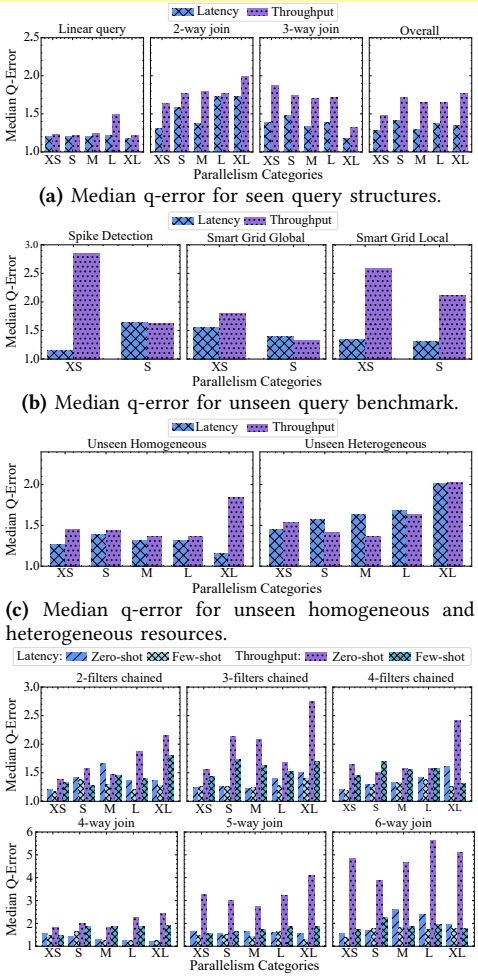
B. ***Exp. 2: Fine-grained Parallelism Analysis***

We further examined the accuracy of the ZeroTune for different parallelism degrees, which is important for the optimizer (see Exp. 6) to select an optimal degree. We categorized parallelism into five classes: *XS, S, M, L,* and *XL*, which state how much the average parallelism degree is for a query per operator. For example, $64 \leq XL < 128$ cores are used by an operator on average in a query (refer to Table III). In the following, we again show how ZeroTune generalizes for seen and unseen query types and public benchmarks for each of these parallelism categories.

Seen Workloads In Figure 7a, we illustrate the accuracy of the model for seen query types within our test range (cf. Table IV ①). ZeroTune model consistently performed well, providing highly accurate cost predictions for different parallelism degrees of seen query types, though the complexity of PQP slightly affects the prediction accuracy. PQP with increasing parallelism and complexity towards *XL* tend to have slightly lower accuracy compared to simple parallel query plans due to the increased computational overhead and data dependencies. However, it is important to note that the decrease in accuracy is relatively small, indicating that ZeroTune exhibits reasonable performance in predicting the cost for complex queries with high parallelism degrees.

Unseen Benchmark. We further extend our evaluations to examine the accuracy and generalization of ZeroTune model for unseen benchmark queries. We observed that the OptiSample strategy deliberately selected low parallelism degree categories XS and S due to the simplicity and arbitrarily low incoming event rates of the benchmarks as shown in in Figure 7b. From the results, it is evident that the model can accurately predict the cost for different parallelism categories in the context of benchmark queries as well. However, in comparison to latency, the cost prediction errors for throughput are slightly higher as the data distribution for benchmarking queries differs from that of the synthetic queries that are in line with the overall results (cf. Table IV

Pratyush Agnihotri, Boris Koldehofe, Paul Stiegele, Roman Heinrich, Carsten Binnig, Manisha Luthra. ZeroTune: Learned Zero-Shot Cost Model for Parallelism Tuning in Stream Processing. To Appear in Proceedings of the IEEE 40th International Conference on Data Engineering (ICDE), 14 pages



(a) Median q-error for seen query structures.

(b) Median q-error for unseen query benchmark.

(c) Median q-error for unseen homogeneous and heterogeneous resources.

(d) Median q-error for unseen query structures.

Fig. 7: Q-errors of cost prediction of PQP for varying parallelism: (a) seen plans, (b) unseen benchmark plans, (c) plans on unseen homogeneous and heterogeneous hardware and (d) comparison of zero-shot and few-shot for unseen plans for ZeroTune model based on OptiSample training strategy.

③). Still, fine-tuning the model can effectively improve ZeroTune's performance in these cases (shown later).

Unseen Resources. In addition, we evaluated the ZeroTune model's generalization ability on *unseen* configurations of heterogeneous and homogeneous resources and the impact of parallelism categories (cf. type "U" in Table II). Figure 7c, reveals that the ZeroTune model consistently delivers accurate cost predictions also for *unseen* hardware resources and their parallelism categories. In line with the other plots, here, too, we see an increasing trend in q-errors with the increasing parallelism degrees due to the complex trends that may occur due to this increase, like load imbalance, parallelism granularity, or resource contention. Nevertheless, the relatively low q-errors demonstrate the model's capability to learn from diverse hardware configurations and effectively correlate them with parallelism degrees due to the resource-related transferable features learned by the model.

Unseen Workload. Next, in Figure 7d, we assess ZeroTune's accuracy for unseen and complex parallel query plans 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 ②. 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*

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

Pratyush Agnihotri, Boris Koldehofe, Paul Stiegele, Roman Heinrich, Carsten Binnig, Manisha Luthra. ZeroTune: Learned Zero-Shot Cost Model for Parallelism Tuning in Stream Processing. To Appear in Proceedings of the IEEE 40th International Conference on Data Engineering (ICDE), 14 pages

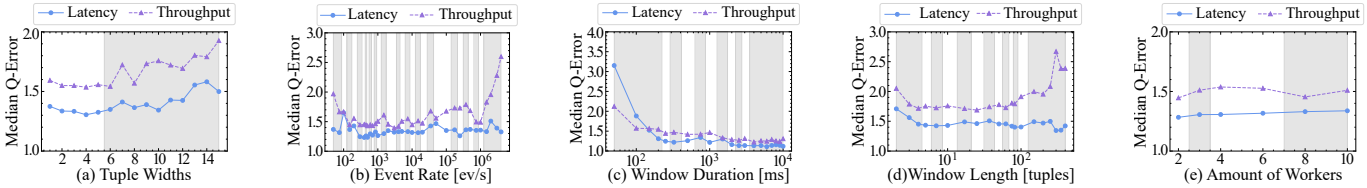


Fig. 8: Median q-errors for cost prediction of PQP for unseen parameters - (a) tuple widths (per data type of src.), (b) event rate, (c) window duration, (d) window length, and (e) amount of workers. The white area represents the training range, while the grey area shows the unseen range of these parameters. ZeroTune generalizes very accurately across all these unseen parameters also where extrapolation ranges are higher.

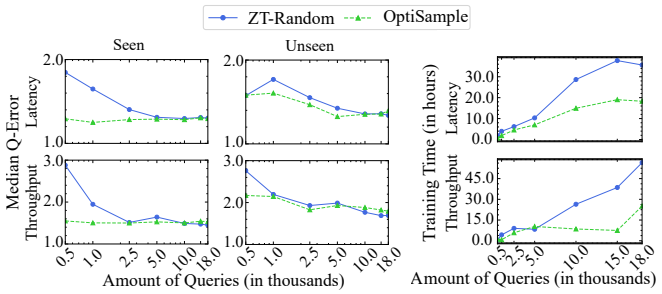
predict costs over a range of window durations (cf. Table III). The model initially shows slightly high median q-errors for smaller unseen windows but overall, it exhibits comparable good performance as the duration increases. As small windows lead to rapid data processing and high data turnover, it 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.

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

In Figure 9a, we compare the accuracy of our ZeroTune models trained with random data: ZT-Random and OptiSample strategy against the increasing number of queries (note log scale in x-axis). For both seen and unseen data, the OptiSample-based model converges rapidly, i.e., as low as $5k$ queries, whereas the random model requires over $18k$ queries for similar accuracy, demonstrating the superior performance of OptiSample in terms of data efficiency.



(a) Amount of queries **(b)** Training time

Fig. 9: (a) Median q-errors for models trained with varying numbers of seen (left) and unseen (middle) queries, and (b) training times required to achieve efficiency for ZeroTune using random (ZT-Random) and OptiSample-based data. The OptiSample-based strategy achieves the same accuracy with half the data and time.

We further analyze in Figure 9b (linear scale in x-axis) the training time required with an increasing number of queries. The OptiSample model achieves higher accuracy in approximately half the time, taking around 4.6 hours, compared to the random model, which takes approximately 10.3 hours. This validates our hypothesis that our data-efficient OptiSample strategy can achieve equal or better accuracy with fewer queries and less training time, addressing a significant bottleneck in training zero-shot models.

E. *Exp. 5: Optimizer for Parallelism Tuning*

We evaluate how ZeroTune aids in selecting parallelism degrees in combination with optimizer as detailed in Section III-C3. First, in Figure 10a, we present the mean speed-ups for the different query structures, including *unseen*. The metric indicates the factor of speed-up induced by executing queries using the selected parallelism degrees by ZeroTune model in comparison to a greedy heuristic [20]. Here, we randomly selected 100 query types with different parameters, which were kept deterministic, while we performed inference for different parallelism degrees based on the enumeration strategies. The speed-ups, compared to a greedy heuristic [20], showed promising results, reaching up to $12\times$ for simple linear queries and about $3.04\times$ for unseen and more complex query types $n-$way joins. Additionally, we

Pratyush Agnihotri, Boris Koldehofe, Paul Stiegele, Roman Heinrich, Carsten Binnig, Manisha Luthra. ZeroTune: Learned Zero-Shot Cost Model for Parallelism Tuning in Stream Processing. To Appear in Proceedings of the IEEE 40th International Conference on Data Engineering (ICDE), 14 pages

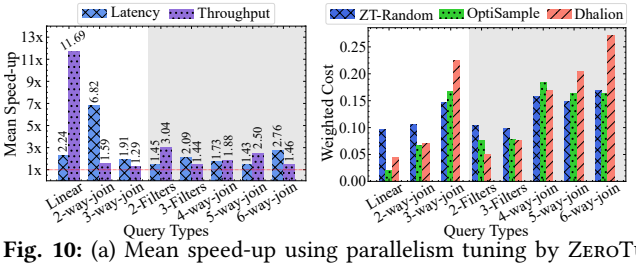


Fig. 10: (a) Mean speed-up using parallelism tuning by ZeroTune compared to a greedy heuristic [20], showing high speed-up in latency and throughput. (b) Comparative performance of query types with parallelism degrees determined by ZeroTune versus Dhalion [19]. The plot shows weighted average runtimes,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

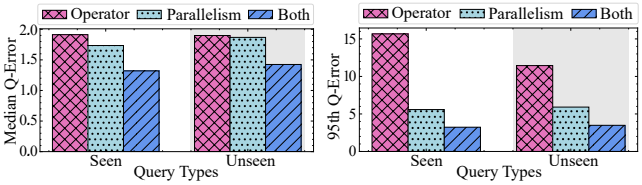


Fig. 11: Feature ablation when using *only* (1) operator-, (2) parallelism- and resource-related features, and (3) *all* features for latency cost prediction.

compared the performance of our model with *Dhalion* [19], the well-known and widely used traditional (non-learned) parallelism tuning algorithm for Heron [33]. 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 10b, 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.

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 [36, 37] and determine the optimal parallelism degree. For instance, monitoring-based heuristic approaches [12, 19, 38–41] extensively use parameters such as observed rate, processing rate, CPU utilization, etc., to determine optimal parallelism degree. Similarly, other mathematical optimization approaches like [42, 43], ILP [25], game theory [44], queuing-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 [7] that is an orthogonal direction. An alternative prominent technique that has gained popularity is incorporating elasticity into the DSP system by allocating or de-allocating 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

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 $5\times$ 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 $4\times$ for generalization.

Acknowledgment

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.

Pratyush Agnihotri, Boris Koldehofe, Paul Stiegele, Roman Heinrich, Carsten Binnig, Manisha Luthra. ZeroTune: Learned Zero-Shot Cost Model for Parallelism Tuning in Stream Processing. To Appear in Proceedings of the IEEE 40th International Conference on Data Engineering (ICDE), 14 pages

References

[1] F. Li, "Cloud-Native Database Systems at Alibaba: Opportunities and Challenges," *Proceedings of VLDB Endowment*, vol. 12, no. 12, p. 2263–2272, aug 2019.

[2] P. Carbone, S. Ewen, G. Fóra, S. Haridi, S. Richter, and K. Tzoumas, "State management in apache flink®: Consistent stateful distributed stream processing," *Proceedings of VLDB Endowment*, vol. 10, no. 12, p. 1718–1729, 2017.

[3] D. Flair. (2016) Apache Flink Use Cases – Real life case studies of Apache Flink. https://data-flair.training/blogs/apache-flink-use-cases/. [Online; accessed 10-07-2023].

[4] A. Ganapathi, H. Kuno, U. Dayal, J. L. Wiener, A. Fox, M. Jordan, and D. Patterson, "Predicting multiple metrics for queries: Better decisions enabled by machine learning," in *Proceedings of IEEE 25th International Conference on Data Engineering*, 2009, pp. 592–603.

[5] T. Heinze, Z. Jerzak, G. Hackenbroich, and C. Fetzer, "Latency-aware elastic scaling for distributed data stream processing systems," in *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*. Association for Computing Machinery, 2014, p. 13–22.

[6] G. Russo Russo, V. Cardellini, and F. Lo Presti, "Hierarchical auto-scaling policies for data stream processing on heterogeneous resources," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 18, no. 4, 2023.

[7] E. Zapridou, I. Mytilinis, and A. Ailamaki, "Dalton: Learned partitioning for distributed data streams," *Proceeding of VLDB Endowment*, vol. 16, no. 3, p. 491–504, 2022.

[8] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng, "A survey of machine learning for big data processing," *EURASIP Journal on Advances in Signal Processing*, vol. 2016, pp. 1–16, 2016.

[9] M. Gheisari, G. Wang, and M. Z. A. Bhuiyan, "A survey on deep learning in big data," in *Proceedings of the IEEE international conference on computational science and engineering (CSE) and IEEE international conference on embedded and ubiquitous computing (EUC)*, 2017, pp. 173–180.

[10] V. Cardellini, F. Lo Presti, M. Nardelli, and G. Russo Russo, "Decentralized self-adaptation for elastic data stream processing," *Future Generation Computer System*, vol. 87, p. 171–185, oct 2018.

[11] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, "Optimized iot service placement in the fog," *Service Oriented Computing and Applications*, vol. 11, pp. 427–443, 2017.

[12] V. Kalavri, J. Liagouris, M. Hoffmann, D. Dimitrova, M. Forshaw, and T. Roscoe, "Three steps is all you need: fast, accurate, automatic scaling decisions for distributed streaming dataflows," in *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, 2018, pp. 783–798.

[13] B. Hilprecht and C. Binnig, "One model to rule them all: Towards zero-shot learning for databases," in *Processing of the 12th Conference on Innovative Data Systems Research*, 2022.

[14] R. Addanki, S. B. Venkatakrishnan, S. Gupta, H. Mao, and M. Alizadeh, "Learning generalizable device placement algorithms for distributed machine learning," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, 2019, pp. 3983–3993.

[15] B. Hilprecht and C. Binnig, "Zero-shot cost models for out-of-the-box learned cost prediction," *Proceeding of VLDB Endowment*, vol. 15, no. 11, p. 2361–2374, 2022.

[16] P. Agnihotri, B. Koldehofe, C. Binnig, and M. Luthra, "Zero-shot cost models for parallel stream processing," in *Proceedings of the Sixth International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, 2023, pp. 1–5.

[17] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS'20, 2020.

[18] E. Strickland, "Andrew ng, ai minimalist: The machine-learning pioneer says small is the new big," *IEEE Spectrum*, vol. 59, no. 4, pp. 22–50, 2022.

[19] A. Floratou, A. Agrawal, B. Graham, S. Rao, and K. Ramasamy, "Dhalion: Self-regulating stream processing in heron," *Proceedings of VLDB Endowment*, vol. 10, no. 12, p. 1825–1836, 2017.

[20] Y. Tang and B. Gedik, "Autopipelining for data stream processing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 12, pp. 2344–2354, 2012.

[21] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," *SIGMOD Rec.*, vol. 34, no. 4, p. 42–47, dec 2005. [Online]. Available: https://doi.org/10.1145/1107499.1107504

[22] J. Karimov, T. Rabl, A. Katsifodimos, R. Samarev, H. Heiskanen, and V. Markl, "Benchmarking distributed stream data processing systems," in *Proceeding of IEEE 34th International Conference on Data Engineering*, 2018, pp. 1507–1518.

[23] M. Luthra, B. Koldehofe, N. Danger, P. Weisenberger, G. Salvaneschi, and I. Stavrakakis, "Tcep: Transitions in operator placement to adapt to dynamic network environments," *Journal of Computer and System Sciences*, vol. 122, pp. 94–125, 2021.

[24] M. V. Bordin, D. Griebler, G. Mencagli, C. F. Geyer, and L. G. L. Fernandes, "Dspbench: A suite of benchmark applications for distributed data stream processing systems," *IEEE Access*, vol. 8, pp. 222 900–222 917, 2020.

[25] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, "Optimal operator replication and placement for distributed stream processing systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 44, pp. 11–22, 2017.

[26] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *The Bulletin of the Technical Committee on Data Engineering*, vol. 38, no. 4, pp. 28–38, 2015.

[27] R. Heinrich, M. Luthra, H. Kornmayer, and C. Binnig, "Zero-shot cost models for distributed stream processing," in *Proceedings of the 16th ACM International Conference on Distributed and Event-Based Systems*, 2022, p. 85–90.

[28] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra, "The design and operation of CloudLab," in *Proceedings of the 2019 USENIX Conference on Annual Technical Conference*, 2019, p. 1–14.

[29] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann, "How good are query optimizers, really?" vol. 9, no. 3, p. 204–215, 2015.

[30] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, "Generalizing from a few examples: A survey on few-shot learning," *ACM computing surveys (csur)*, vol. 53, no. 3, pp. 1–34, 2020.

[31] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in *Proceeding of IEEE International Conference on Data Mining Workshops*. IEEE, 2010, pp. 170–177.

[32] P. Bodik, W. Hong, C. Guestrin, S. Madden, M. Paskin, R. Thibaux, J. Polastre, and R. Szewczyk, "Intel lab data," 6 2004. [Online]. Available: http://db.csail.mit.edu/labdata/labdata.html

[33] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja, "Twitter heron: Stream processing at scale," in *Proceedings of the 2015 ACM SIGMOD international conference on Management of data*, 2015, pp. 239–250.

[34] M. H. Iqbal, T. R. Soomro *et al.*, "Big data analysis: Apache storm perspective," *International journal of computer trends and technology*, vol. 19, no. 1, pp. 9–14, 2015.

[35] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, "Big data analytics on apache spark," *International Journal of Data Science and Analytics*, vol. 1, pp. 145–164, 2016.

[36] C. Pohl, P. Götze, and K. Sattler, "A cost model for data stream processing on modern hardware," in *International Workshop on Accelerating Analytics and Data Management Systems Using Modern Processor and Storage Architectures,(ADMS)*, 2017.

[37] O. Papaemmanouil, U. Çetintemel, and J. Jannotti, "Supporting generic cost models for wide-area stream processing," in *Proceedings of IEEE 25th International Conference on Data Engineering*, 2009.

[38] L. Xu, B. Peng, and I. Gupta, "Stela: Enabling stream processing systems to scale-in and scale-out on-demand," in *Proceedings of IEEE International Conference on Cloud Engineering (IC2E)*, 2016, pp. 22–31.

[39] C. Li, H. Zhuang, Q. Wang, and X. Zhou, "Sslb: Self-similarity-based load balancing for large-scale fog computing." *Arabian Journal for Science & Engineering (Springer Science & Business Media BV)*, vol. 43, no. 12, 2018.

[40] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina *et al.*, "The design of the borealis stream processing engine," in *Proceedings of the*

Pratyush Agnihotri, Boris Koldehofe, Paul Stiegele, Roman Heinrich, Carsten Binnig, Manisha Luthra. ZeroTune: Learned Zero-Shot Cost Model for Parallelism Tuning in Stream Processing. To Appear in Proceedings of the IEEE 40th International Conference on Data Engineering (ICDE), 14 pages

Conference on Innovative Data Systems Research, vol. 5, no. 2005, 2005, pp. 277–289.

[41] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, C. Soriente, and P. Valduriez, "Streamcloud: An elastic and scalable data streaming system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 12, pp. 2351–2365, 2012.

[42] S. Kabirzadeh, D. Rahbari, and M. Nickray, "A hyper heuristic algorithm for scheduling of fog networks," in *Proceedings of the 21st Conference of Open Innovations Association (FRUCT)*, 2017, pp. 148–155.

[43] M. Selimi, L. Cerdà Alabern, F. Freitag, L. Veiga, A. Sathiaseelan, and J. Crowcroft, "A lightweight service placement approach for community network micro-clouds," *Journal of Grid Computing*, vol. 17, pp. 169–189, 2019.

[44] J. A. Khan, C. Westphal, and Y. Ghamri-Doudane, "Offloading content with self-organizing mobile fogs," in *Proceedings of the 29th International Teletraffic Congress (ITC)*, 2017, pp. 223–231.

[45] L. Liu, Z. Chang, and X. Guo, "Socially aware dynamic computation offloading scheme for fog computing system with energy harvesting devices," *IEEE Internet of Things Journal*, vol. 5, pp. 1869–1879, 2018.

[46] R. Mayer, B. Koldehofe, and K. Rothermel, "Predictable low-latency event detection with parallel complex event processing," *IEEE Internet of Things Journal*, vol. 2, pp. 274–286, 2015.

[47] J. Shin, D. Arroyo, A. Tantawi, C. Wang, A. Youssef, and R. Nagi, "Cloud-native workflow scheduling using a hybrid priority rule and dynamic task parallelism," in *Proceedings of the 13th Symposium on Cloud Computing*. Association for Computing Machinery, 2022, p. 72–77.

[48] N. Hidalgo, D. Wladdimiro, and E. Rosas, "Self-adaptive processing graph with operator fission for elastic stream processing," *Journal of Systems and Software*, vol. 127, no. C, p. 205–216, 2017.

[49] W. Dai, L. Qiu, A. Wu, and M. Qiu, "Cloud infrastructure resource allocation for big data applications," *IEEE Transactions on Big Data*, vol. 4, no. 3, pp. 313–324, 2018.

[50] R. K. Kombi, N. Lumineau, and P. Lamarre, "A preventive auto-parallelization approach for elastic stream processing," in *Proceedings of IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 1532–1542.

[51] P. Agnihotri, "Autonomous resource management in distributed stream processing systems," in *Proceedings of the 22nd International Middleware Conference: Doctoral Symposium*, 2021, p. 19–22.

[52] P. Agnihotri, B. Koldehofe, C. Binnig, and M. Luthra, "Panda: performance prediction for parallel and dynamic stream processing," in *Proceedings of the 16th ACM International Conference on Distributed and Event-Based Systems*, 2022, p. 180–181.

[53] Z. Fan, R. Sen, P. Koutris, and A. Albarghouthi, "Automated tuning of query degree of parallelism via machine learning," in *Proceedings of the Third International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, 2020.