## A METRIC BASED ON PLAN-COST

Definition 4.7, §4 gave the definition for P-Cost to evaluate the goodness of cardinality estimates. Here, we show that we can extend this to a pseudometric to compute the distance between any two cardinality vectors for a given query:

$$\text{P-Error}(Y_1, Y_2) = |\text{P-Cost}(Y_1, \mathbf{Y}) - \text{P-Cost}(Y_2, \mathbf{Y})| \tag{14}$$

Notice, we need to pre-compute the constant vector, **Y** for the given query. This is a pseudo-metric because it satisfies its three properties:

(1) $d(x,x) = 0$; Clearly, P-Error$(Y_1, Y_1) = 0$. Also, notice that there can be other points such that P-Error$(Y_1, Y_2) = 0$.
(2) $d(x,y) = d(y,x)$ (Symmetry); follows due to the absolute value sign in the definition of P-Error.
(3) $d(x,z) \leq d(x,y) + d(y,z)$ (Triangle Inequality); We will present the proof for this below.

For notational convenience, we will use $P_e$ to refer to Plan-Error and $P_c$ to refer to Plan-Cost. $Y_t$ refers to **Y**. Then the proof for the triangle inequality follows:

$$\begin{aligned} P_e(Y_1, Y_3) &= |P_c(Y_1, Y_t) - P_c(Y_3, Y_t)| \\ &= |P_c(Y_1, Y_t) - P_c(Y_2, Y_t) + \\ &\quad P_c(Y_2, Y_t) - P_c(Y_3, Y_t)| \\ &\leq |P_c(Y_1, Y_t) - P_c(Y_2, Y_t)| + \\ &\quad |P_c(Y_2, Y_t) - P_c(Y_3, Y_t)| \\ &= P_e(Y_1, Y2) + P_e(Y_2, Y_3) \end{aligned} \tag{15}$$

The first line is the definition of P-Error. In the second line, we are adding and subtracting same value $P_c(Y_2, Y_t)$. The third line follows from the definition of absolute value, which gives us exactly the statement of the triangle inequality that we were trying to prove.

## B FLOW-LOSS DETAILS

### B.1 Computing Flows

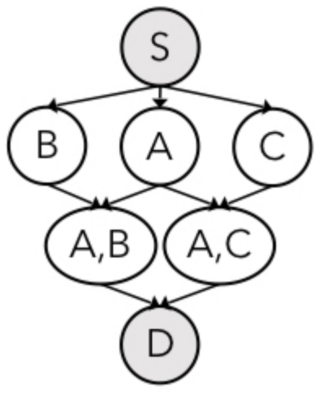


**Figure 21: Plan graph for query shown in Figure 1.**

We will use the example of the query shown in Figure 4 in §1 to show the explicit calculations used in the equations for computing the flows. First, we show the plan graph for the query in Figure 21.

Our goal is to derive the solution for the optimization variable, $F$, in Equation 9 in §5, which assigns a flow to every edge, $e$ in the plan graph. Throughout this derivation, we will assume access to the cost function, $C$ (Definition 4.5, §4), which assigns $C(e, Y)$, a cost to any edge, $e$ in the plan graph given a cardinality vector $Y$. We rewrite the optimization program for $F$ below:

$$\text{F-Opt(Y)} = \arg\min_F \sum_{e \in E} C(Y)_e F_e^2 \tag{16}$$

$$\text{s.t} \sum_{e \in Out(S)} F_e = \sum_{e \in In(D)} F_e = 1 \tag{17}$$

$$\sum_{e \in Out(V)} F_e = \sum_{e \in In(V)} F_e \tag{18}$$

For simplicity, we will use $F$ to refer to the vector solution of the optimization problem above. We can express the set of constraints specified above as a system of linear equations. Recall, that in an electric circuit, every node has an associated voltage. Equation 16 describes an electric circuit; For our example (Figure 21), we can write out the linear equations that need to be satisfied by the constraints in terms of the 'voltages' of each node, and resistances (costs) for each edge, and using Ohm's law as:

$$\begin{aligned} \frac{(v_S - v_A)}{C((S,A),Y)} + \frac{(v_S - v_B)}{C((S,B),Y)} + \frac{(v_S - v_C)}{C((S,C),Y)} &= 1 \\ \frac{(v_B - v_{AB})}{C((B,AB),Y)} + \frac{(v_B - v_S)}{C((S,B),Y)} &= 0 \\ \frac{(v_C - v_{AC})}{C((C,AC),Y)} + \frac{(v_C - v_S)}{C((S,C),Y)} &= 0 \\ \frac{(v_A - v_{AB})}{C((A,AB),Y)} + \frac{(v_A - v_{AC})}{C((A,AC),Y)} + \frac{(v_A - v_S)}{C((S,A),Y)} &= 0 \\ \frac{(v_{AB} - v_D)}{C((AB,D),Y)} + \frac{(v_{AB} - v_A)}{C((A,AB),Y)} + \frac{(v_{AB} - v_C)}{C((C,AB),Y)} &= 0 \\ \frac{(v_{AC} - v_D)}{C((AC,D),Y)} + \frac{(v_{AC} - v_A)}{C((A,AC),Y)} + \frac{(v_{AC} - v_C)}{C((C,AC),Y)} &= 0 \\ \frac{(v_D - v_{AB})}{C((AB,D),Y)} + \frac{(v_{AC} - v_A)}{C((A,AC),Y)} + \frac{(v_{AC} - v_C)}{C((C,AC),Y)} &= -1 \end{aligned} \tag{19}$$

where terms of the form $\frac{v_A}{C((S,A),Y)}$ use Ohm's law to compute the amount of maximum incoming current to node $A$ from $S$, and so on for the other terms. Notice that the first and the last equations satisfy the constraint in Equation 17 — 1 unit of flow is outgoing from $S$ and 1 unit of flow is incoming to $D$. All the intermediate equations represent the conservation constraint (Equation 18) — this will be obvious if you expand out each equations, and separate the terms being added (incoming current) versus the terms being subtracted (outgoing current) in each equation.

Next, we will compactly represent the above linear constraints in terms of matrix operations. The system of linear equations above is over-determined, thus in practice, we remove the first equation and solve the remaining ones. For simplicity, we will express the matrix operations without removing any constraint.

Let $v \in R^N$ be the vector for the voltages $v_S ... v_D$ for all the nodes in the plan graph. Similarly, let $i \in R^N$ be the vector of $1, 0, ..., -1$ (the right hand side of Equation 19). Let us define the edge-vertex incidence matrix, $X \in R^{n,m}$, with rows indexed by vertices and columns indexed by edges:

$$X_{n,e} = \begin{cases} 1 & \text{if } e \in Out(n) \\ -1 & \text{if } e \in In(n) \\ 0 & \text{otherwise.} \end{cases} \tag{20}$$

Next, for simplifying the notation, we will define $\frac{1}{C(e,Y)} = g_e$. And let, $G(Y) \in R^{m,m}$, be a diagonal matrix for the $m$ edges, with $G_{e,e} = g_e = \frac{1}{C(e,Y)}$:

$$g(Y) = \begin{bmatrix} \frac{1}{C(e_1,Y)} \\ \vdots \\ \frac{1}{C(e_m,Y)} \end{bmatrix}; \quad G(Y) = \begin{bmatrix} g(e_1)) & \dots & 0 \\ \vdots & \ddots & \\ 0 & & g(e_m) \end{bmatrix} \tag{21}$$

Thus, $G$ is defined simply using the cost of each edge in the plan graph w.r.t. a given cardinality vector $Y$. Note that in $g$ and $G$ we use the inverse of the costs, because in Equation 19, the cost terms are in the denominator, thus this simplifies the formulas.

We define $B(Y) \in R^{N,N}$ as a linear function of the costs w.r.t. cardinality vector $Y$:

$$B(Y) = X \cdot G(Y) \cdot X^T \tag{22}$$

we can verify that each entry of $B$ is given by the following piece-wise linear function:

$$B_{u,w} = \begin{cases} \sum\limits_{e \in In(u) \cup Out(u)} \frac{1}{C(e,Y)} & \text{if } u = w \\ -\frac{1}{C(e,Y)} & \text{if } e = (u,w) \text{ is an edge} \\ 0 & \text{otherwise.} \end{cases}$$

Now, we are ready to compactly represent all the constraints from Equation 19:

$$\begin{aligned} B(Y)v &= i \\ \implies v &= B(Y)^{-1} i \end{aligned} \tag{23}$$

We do not know $v$, but $B(Y)$ is a deterministic function given a cost model, $C$ and cardinality vector $Y$, while $i$ is a fixed vector. Thus, using the constraints, we have found a way to compute the values for $V$. Note that $B(Y)$ does not have to be invertible, since we can use pseudo-inverses as well.

Recall, that the flows in Equation 16 correspond to the current in the context of electrical circuits. Using Ohm's law, and the voltages calculated above, we can calculate the optimal flow (current) of an edge as:

$$F_{(u,w)} = (v_u - v_w) \cdot g_{u,w} \tag{24}$$

where $g_{u,w} = \frac{1}{C((u,w),Y)}$ is the inverse of the cost of edge $(u,w)$, and $v_u, v_w$ are the voltages' associated with nodes $u$ and $w$. The linear equations for the flow, $F$, on each edge can be represented as a matrix multiplication:

$$\begin{aligned} F(Y) &= G(Y) X^T v \\ &= G(Y) X^T B(Y)^{-1} i \end{aligned} \tag{25}$$

where the second Equation uses Equation 25. In §5, Equation 12, we gave the general form of the solution for $F(Y)$, which is satisfied by the precise definition in Equation 25.

## B.2 Flow-Loss

Note that we found the flows, $F$, using the estimated cardinalities, $Y$, and the costs induced by them on each of the edges. To compute the true costs, which is used to calculate Flow-Loss, we will need to use the true cardinalities, $\mathbf{Y}$. For convenience, we will define the following diagonal matrix with the true cost of each edge:

$$C^{true} = \begin{bmatrix} C(e_1, \mathbf{Y}) & \dots & 0 \\ \vdots & \ddots & \\ 0 & & C(e_m, \mathbf{Y}) \end{bmatrix} \tag{26}$$

We will rewrite Flow-Loss, as also shows in Equation 13 in §5:

$$\sum_e C^{true}_{e,e} F_e^2 \tag{27}$$

where we sum over all edges in the plan graph. In terms of matrix operators, we can write it as:

$$F^T C^{true} F \tag{28}$$

## B.3 Flow-Loss gradient

We present the dependency structure in the computation for *Flow−Loss* below. Notice that the cardinalities, $Y$ only impact Flow-Loss through the costs, represented by $g$.

$$Y \xrightarrow{C(\cdot)} \text{g} \xrightarrow{Opt(\cdot), C^{true}} \text{Flow-Loss.}$$

For convenience, we use $g$ to represent costs — recall, $g$ is just the inverse of the costs. The definitions of $g$ and $C^*$ are given in Equations 21, 26. For simplicity, we refer to $\hat{Y}$ as simply $Y$. We will use the vector notations $\vec{g}$ or $\vec{Y}$ to refer to vectors, and when gradients are taken w.r.t. vectors or scalar quantities. Recall, gradient of an $n$ dimensional vector w.r.t. an $m$ dimensional vector is the $nxm$ dimensional Jacobian matrix; while gradient of a scalar w.r.t. an $n$ dimensional vector is still an $n$ dimensional vector. For simplicity, we will also avoid explicitly writing out the dependencies as function arguments like we have done so far — so instead of $G(Y)$ (Equation 21), we will just write $G$.

$$\begin{aligned} \nabla_{\vec{Y}} \text{Flow-Loss} &= (\nabla_{\vec{Y}} \vec{g}) \; (\nabla_{\vec{g}} \text{Flow-Loss}) \\ &= (\nabla_{\vec{Y}} \vec{g}) \; (\nabla_{\vec{g}} \vec{F}) \; (2\, C^{true} \vec{F}) \end{aligned} \tag{29}$$

where $\vec{F}$ are the flows for each edge, which we can compute using Equation 25. The first line follows because given $g$ (inverse estimated costs of each edge), computing the Flow-Loss does not depend on $Y$. Therefore, we can use the chain rule to separate it out into two independent gradients. The second line follows because consider the partial derivative of Flow-Loss (as in Equation 27) w.r.t. a single element of $g$:

$$\frac{\partial \text{Flow-Loss}}{\partial g_j} = 2 \sum_e C^{true}_{e,e} F_e \frac{\partial F_e}{\partial g_j} \tag{30}$$

Here, $2 \sum_e C^{true}_{e,e} F_e$ corresponds to the term $(2C^{true}F)$ in Equation 29.

Next, we will write out the explicit formulas for each of the unknown terms in the above equation. First, $\nabla_{\vec{Y}} \vec{g}$ is the Jacobian matrix of the cost function, with input estimated cardinalities, and outputs costs.

$$\nabla_{\vec{Y}}\vec{g} = \begin{bmatrix} \frac{dg_1}{dY_1} & \frac{dg_2}{dY_1} & \cdots \frac{dg_m}{dY_1} \\ \vdots & \ddots & \\ \frac{dg_1}{dY_n} & & \frac{dg_m}{dY_n} \end{bmatrix} \tag{31}$$

Notice that this is the only place where we need to take the gradient of the cost function, $C$ (Definition 4.5, §4). Thus, we just need to know how to take the gradient of the cost of a single edge w.r.t. the two cardinality estimates that are used to compute that cost, as in Definition 4.5, §4. Thus, we could potentially be using significantly more complex cost models as long as this value can be approximated. Also, most terms in this Jacobian matrix end of trivially being zeros since the cost of a particular edge will only depend on two elements of $Y$. This is one of the ways we can significantly speed up the gradient computations by explicitly coding up the formulas.

Calculating $\nabla_{\vec{g}}\vec{F}$ is more involved. The solution comes out to be:

$$\nabla_{\vec{g}}\vec{F} \in R^{M,N} = \begin{bmatrix} i^T B^{-T}\left(\frac{\partial GX}{\partial g_1} - GXB^{-1}\frac{\partial G}{\partial g_1}\right) \\ \vdots \\ i^T B^{-T}\left(\frac{\partial GX}{\partial g_1} - GXB^{-1}\frac{\partial G}{\partial g_m}\right) \end{bmatrix} \tag{32}$$

Note, each row in the above matrix is a vector in $R^n$. $X$ was defined in Equation 20, $B$ was defined in Equation 22, and $G$ was defined in Equation 21. As we can see there are many results being re-used from the computations of $F$; in terms of implementation, this means that the forward and backward passes of a neural network can reuse intermediate results, which results in significantly more efficient code as well. Also, once again we see that many of the partial derivatives would be 0, which don't need to be computed when implementing these gradients.

## C EVALUATING COST MODEL APPROXIMATION

*Quantitatively Evaluating a cost model:* In Figure 8, we showed a scatter plot, showing intuition why our cost model was a reasonable approximation to the PostgreSQL cost model. But, it is helpful to have a more quantitative way to evaluate how good an approximation is. When defining Plan-Cost, we assumed a cost model, $C$. This could be an arbitrary cost model. To quantitatively assess how good $C$ is, we can compare the quality of its plans to a target cost model. For instance, let $PG$ be the PostgreSQL cost model with hash joins and merge joins disabled. We disable these since $C$ is defined over the set of left deep plans — this is just to evaluate how good an approximation $C$ is, therefore, we should define both the cost models over the same set of plans. Note that our evaluations of Postgres Plan Cost are done without these restrictions on the PostgresSQL cost model.

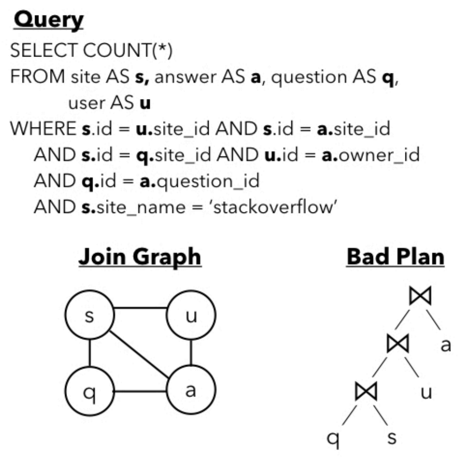
Now, we can assess the quality of $C$'s approximation, given true cardinalities $\mathbf{Y}$, as:

$$\text{CM-Approx}(C, PG) = \frac{\text{Cost}_{PG}(\mathbf{Y}, P^*_C(\mathbf{Y})}{\text{Cost}_{PG}(\mathbf{Y}, P^*_{PG}(\mathbf{Y}))} \tag{33}$$

The denominator represents the cost of the optimal PostgreSQL plan, and the numerator represents the *PostgreSQL cost* of the optimal plan found using $C$. Clearly, we would not expect expect the plans using our simple cost model to exactly match ones found by PostgreSQL. But, CM-Approx provides a useful, quantitative way to compare between different approximations of candidate cost models. For choosing a cost model, we iterate over a few candidate cost models until we can achieve reasonably low error in terms of CM-Approx. For instance, the cost model, $C$, defined in Equation 6, has a mean CM-Approx error of 2.05 on the IMDb workload.

## D DATASET

### D.1 StackExchange timeouts



**Query**

```
SELECT COUNT(*)
FROM site AS s, answer AS a, question AS q,
     user AS u
WHERE s.id = u.site_id AND s.id = a.site_id
   AND s.id = q.site_id AND u.id = a.owner_id
   AND q.id = a.question_id
   AND s.site_name = 'stackoverflow'
```

**Figure 22: An example of a timed out subquery on the Stack-Exchange database.**

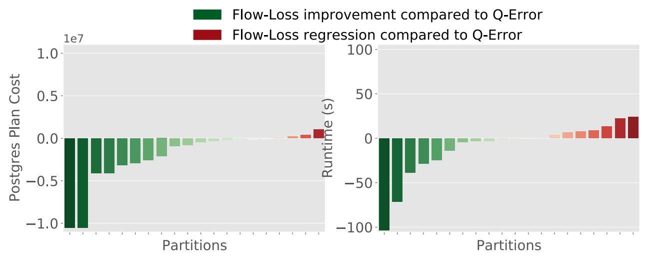
On the stackexchange datasets, three of the six templates have a large proportion of timeouts when generating the ground truth data for the sub-plans. This is due to the unusual join graph, presented with a simplified query in Figure 22. As the accompanying query makes clear, there is a relationship between the joins on user, $u$ and answer $a$. But there is no relationship between question $q$ and user $u$ without $a$; still, the sub-plan $q \bowtie s \bowtie u$ is not a cross-join since all tables have a relationship with site $s$. Essentially, $q \bowtie s \bowtie u$ will give us the cross-join of all questions (potentially, satisfying some filters) along with all users (satisfying some filters) — which will blow up and lead to timeouts. Such join graphs can occur in natural queries, for instance, to process information about users who answer certain types of questions (e.g., what is the location of users who answer stackoverflow questions with Javascript tags).

## E ADDITIONAL RESULTS

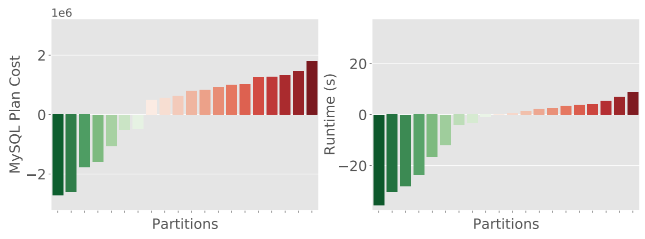
Here, we present additional results that did not make it into the main paper.

**Table 2: Models trained and evaluated on the same IMDb templates. We show $\pm 1 stddev$ for Q-Error and PPC from three repeated experiments, and execute plans from one random run. $Xp$ refers to the $Xth$ percentile.**

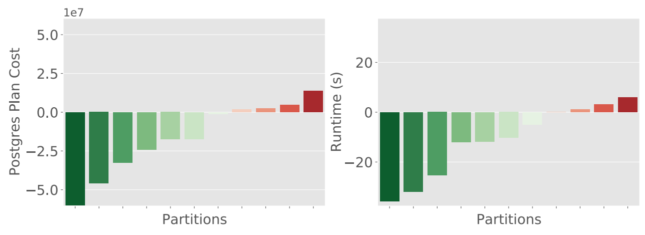
| | Q-Error | | | Postgres Plan Cost (Millions) | | | Runtime | | |
|---|---|---|---|---|---|---|---|---|---|
| | 50p | 90p | 99p | Mean | 90p | 99p | Mean | 90p | 99p |
| **Baselines** | | | | | | | | | |
| True | 1 | 1 | 1 | **4.7** | 1.1 | 131.5 | **13.23** | 28.96 | 60.83 |
| PostgreSQL | 42.87 | 48K | 2.2M | 10.7 | 9.6 | 269.7 | 20.86 | 46.86 | 101.57 |
| **FCNN** | | | | | | | | | |
| Q-Error | **2.0 ± 0.1** | 10.5 ± 1.6 | 0.1K ± 32.0 | 6.4 ± 0.5 | 1.8 ± 0.1 | 142.6 ± 11.7 | **13.83** | 30.82 | 60.10 |
| Prioritized Q-Error | 2.5 ± 0.1 | 17.1 ± 1.9 | 0.4K ± 83.9 | **5.9 ± 0.4** | 1.7 ± 0.0 | 135.8 ± 18.9 | 13.91 | 31.48 | 61.40 |
| Flow-Loss | 4.0 ± 0.9 | 83.4 ± 36.3 | 4.1K ± 1.9K | 6.0 ± 0.7 | 1.8 ± 0.1 | 136.0 ± 23.9 | 13.93 | 30.97 | 60.50 |
| **MSCN** | | | | | | | | | |
| Q-Error | **2.0 ± 0.0** | 9.6 ± 0.2 | 0.1K ± 3.9 | 6.4 ± 0.3 | 1.9 ± 0.1 | 189.1 ± 6.6 | **13.69** | 30.49 | 58.49 |
| Prioritized Q-Error | 2.8 ± 0.4 | 25.2 ± 8.3 | 0.7K ± 0.4K | 6.1 ± 0.4 | 2.4 ± 0.4 | 163.8 ± 10.1 | 13.83 | 31.10 | 60.00 |
| Flow-Loss | 3.0 ± 0.1 | 44.4 ± 5.6 | 2.1K ± 0.4K | **5.5 ± 0.6** | 2.2 ± 0.1 | 161.9 ± 16.2 | 14.08 | 31.61 | 59.88 |



**(a) IMDb workload, PostgreSQL DBMS, FCNN or MSCN model.**

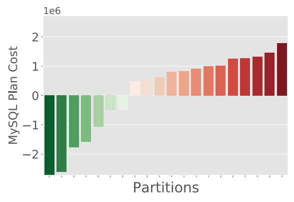


**(b) [IMDb workload, MySQL DBMS, FCNN or MSCN model.]**

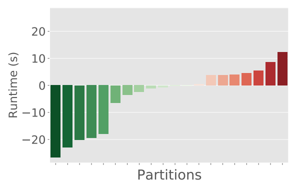


**(c) StackExchange workload, PostgreSQL DBMS, MSCN model.**

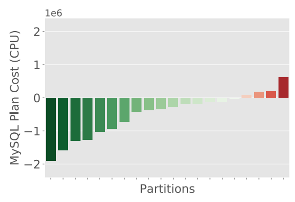
**Figure 25: Each bar shows the PPC or runtime improvement/regression of Flow-Loss over Q-Error on an unseen partition and the same model. *Lower is better.***



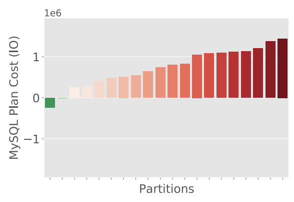
**(a) MySQL Plan Cost.**



**(b) Runtime (using MySQL).**

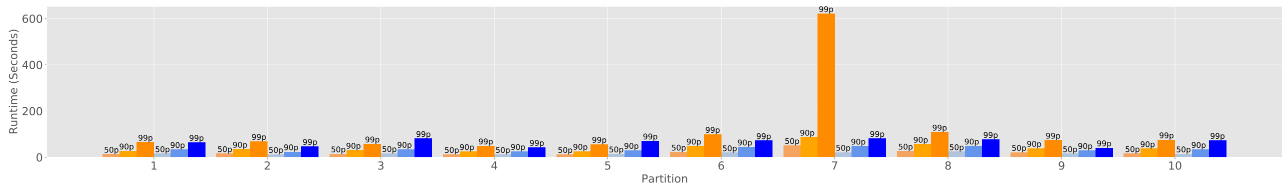


**(c) MySQL Plan Cost (CPU).**

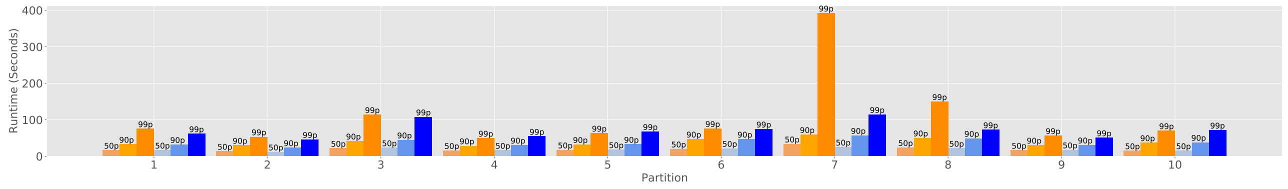


**(d) MySQL Plan Cost (IO).**

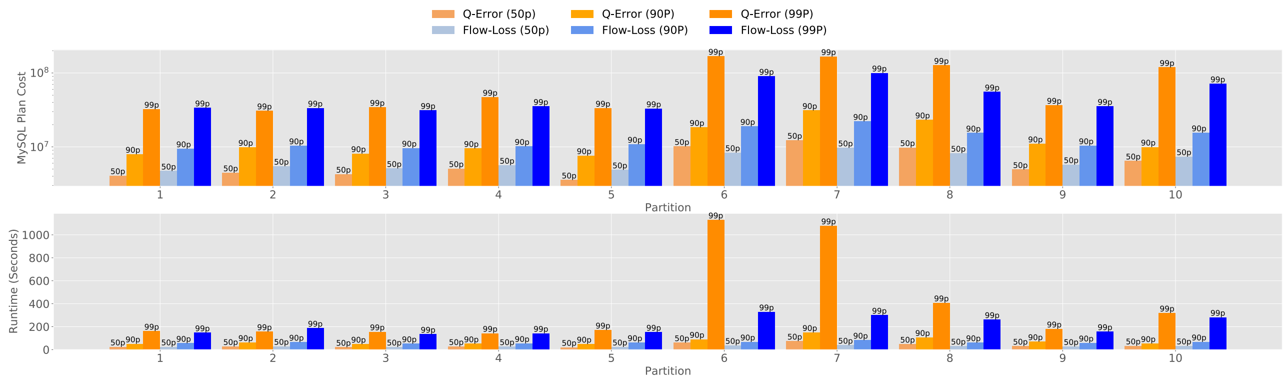
**Figure 26: Each bar shows the mean runtime, or MySQL Plan Cost, improvement (green) or regression (red) of Flow-Loss over Q-Error on a single unseen partition using the same model architecture. *Lower is better for Flow-Loss models.***



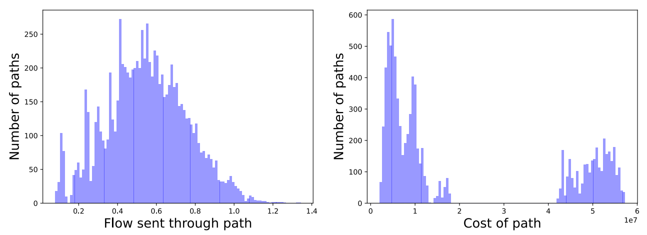
**(a) Query runtime for all unseen templates partitions, for FCNN models trained with Flow-Loss or Q-Error.**



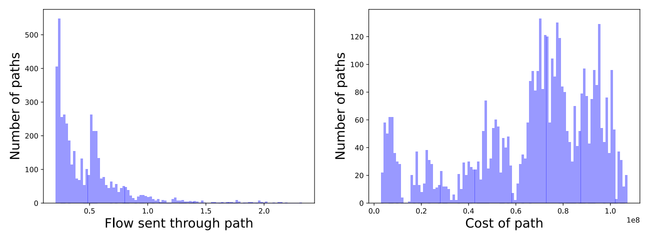
**(b) Query runtime for all unseen templates partitions, for MSCN models trained with Flow-Loss or Q-Error.**



**(c) MySQL Plan Cost and query runtime for all unseen templates partitions, for FCNN models trained with Flow-Loss or Q-Error.**



**(a) Template 1a.**



**(b) Template 9a.**

**Figure 28: Each template has queries with the same set of paths / plans. For every query, we can compute the average of flow sent through that path, and the cost of that path (both of these are computed using true cardinalities here). The histograms show the average flow sent through a path (left), and average cost of a path (right) across all queries on two templates.**