Query Processing / Query Optimization / In-database Inference

A Comparative Study of in-Database Inference Approaches

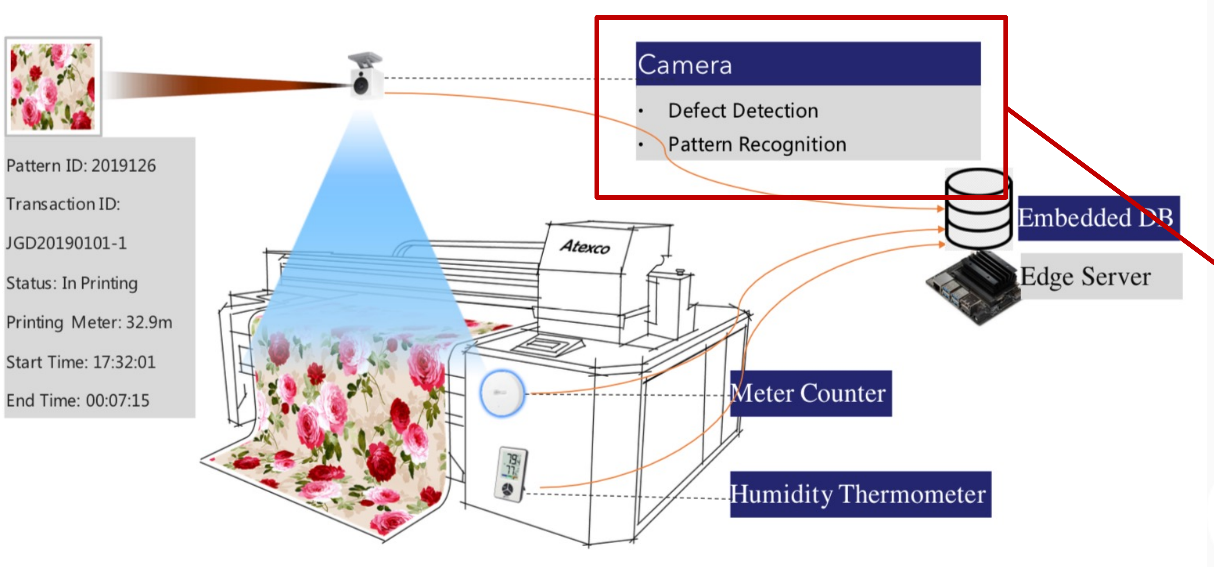
- Qiuru Lin ‡1, Sai Wu ‡2, Junbo Zhao ‡3, Jian Dai #4 , Feifei Li #5, Gang Chen ‡6
- ‡Zhejiang University, China #Alibaba Group, China

May 12, 2022

1 Research Background

- *DL + DB = ?*

Deep Learning(DL) ? DataBase (DB)

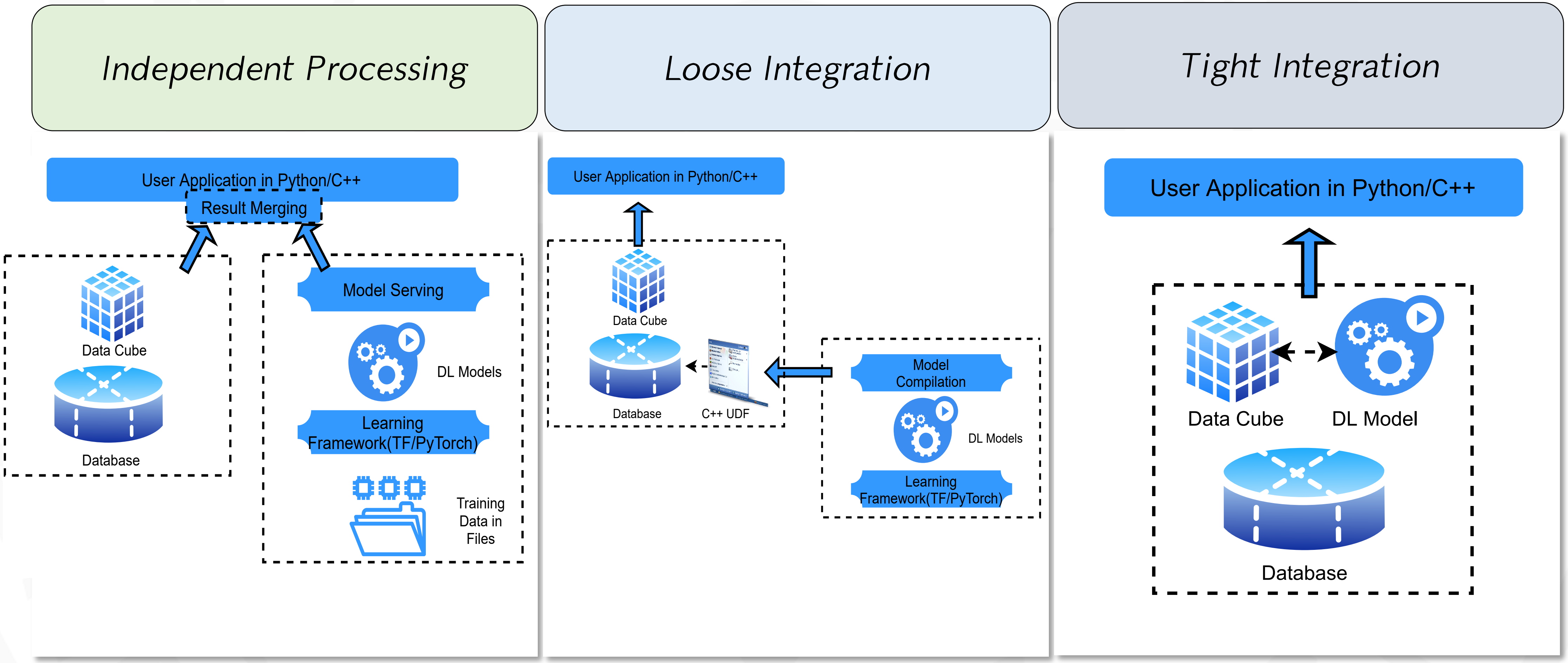



```
SELECT patternID, transID
FROM FABRIC F, Video V
WHERE F.humidity > 80 and F.temperature > 30
    and F.printdate > '2021-01-01'
    and F.printdate < '2021-1-31'
    and F.transID = V.transID
    and V.date > '2021-01-01'
    and V.date < '2021-1-31'
    and nUDF_detect(V.keyframe) = FALSE;
```

the query of DB

the query of DL

2 Analysis: 3 in–database inference strategies



2 *Analysis: 3 in–database inference strategies*

- *Challenge:*

Independent Processing	*Loose Integration*	*Tight Integration*
• *Easy to implement*	• *Reusability*	• *Lower I/O cost*
• *Scalability*	• *Lack of optimization*	• *Scalability*
• *High I/O cost*	• *High I/O cost*	• *Reusability*
• *Lack of reusability*	• *Complexity*	• *Hard to implement*
	• *Scalability*	

3 DL2SQL*: Key idea

- *Tight coupling of DL and DB;*
- *Converting different neural operators into pure SQL queries;*
- *The implemented neural operators can be easily assembled to realize various neural networks;*
- *The collaborative query can be optimized with the database optimizer.*

Lower I/O cost

Reusability

Scalability

3 *Implement Detail (take CNN as an example):*

- ***FeatureMap Table:*** *Input data and feature maps of each layer of neural network stored in specific structure;*
- ***Kernel Table:*** *Storage of convolution kernel parameters;*
- ***Kernel Mapping Table:*** *Storage of row mapping relation between the output of the previous layer and the input of the current layer.*

3 Implement Detail (take CNN as an example):

- FeatureMap Table:

2	1	3	4	5
1	3	4	5	7
3	4	5	7	2
4	5	7	2	1
5	7	2	1	4

5*5*1 Feature Map

3	1	1
1	5	1
0	1	2

0	3	2
2	1	0
1	1	3

Kernels

Matrix ID	Order ID	Value
1	1	2
1	2	1
...	...	...
1	9	5
2	1	3
2	2	4
...	...	...
2	9	2

Feature Map Table

Kernel ID	Order ID	Value
1	1	3
1	2	1
...	...	...
1	9	2
2	1	0
2	2	3
...	...	...
2	9	3

Kernel Table

Algorithm 1: Generation of Feature Map Table

Input: Input F, Kernel K
Output: SQLs for Creating the Feature Map Table

FeatureMap = $\emptyset$, $k = K.height$, $s = K.striding$, $Order\_Header = 0$
for $i = 1$ to $F.channel$ **do**
  $MatrixID = 1$
  for $y = 1$ to $F.height$ **do**
    for $x = 1$ to $F.width$ **do**
      $OrderID \leftarrow Order\_Header$
      for $Coordinate\_y = y$ to $y + k$ **do**
        for $Coordinate\_x = x$ to $x + k$ **do**
          $Values \leftarrow F.$Value($Coordinate\_x$, $Coordinate\_y$)
          Insert $\{MatrixID, OrderID,$ Values$\}$ into FeatureMap
          $OrderID$ ++
        end
      end
      $x \leftarrow x + s$, $MatrixID$ ++
    end
    $y \leftarrow y + s$
  end
  $Order\_Header \leftarrow Order\_Header + k^2$
end
return FeatureMap

3 Implement Detail (take CNN as an example):

- *FeatureMap Table:*

2	1	3	4	5
1	3	4	5	7
3	4	5	7	2
4	5	7	2	1
5	7	2	1	4

5*5*1 Feature Map

Matrix ID	Order ID	Value
1	1	2
1	2	1
...	...	...
1	9	5
2	1	3
2	2	4
...	...	...
2	9	2

Feature Map Table

3	1	1
1	5	1
0	1	2

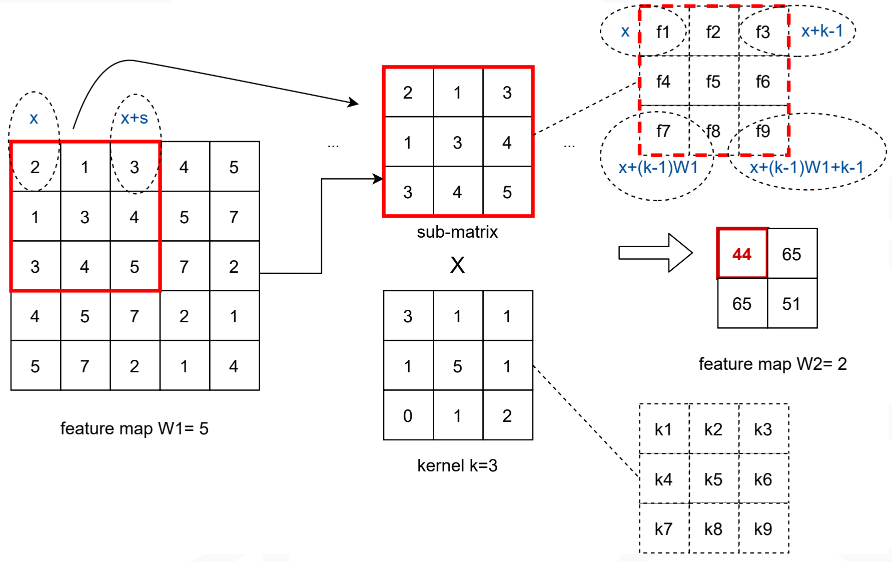
0	3	2
2	1	0
1	1	3

Kernels

Kernel ID	Order ID	Value
1	1	3
1	2	1
...	...	...
1	9	2
2	1	0
2	2	3
...	...	...
2	9	3

Kernel Table

Q1:
```
CREATE TEMP TABLE Layer_Output(
SELECT MatrixID as TupleID,
       SUM(A.Value * B.Value) as Value
FROM FeatureMap A INNER JOIN Kernel B
ON A.OrderID = B.OrderID
GROUP BY KernelID, MatrixID);
```



3 *Implement Detail (take CNN as an example):*

- *Kernel Mapping Table:*

Q1:
```
CREATE TEMP TABLE Layer_Output(
SELECT MatrixID as TupleID,
        SUM(A.Value * B.Value) as Value
FROM FeatureMap A INNER JOIN Kernel B
ON A.OrderID = B.OrderID
GROUP BY KernelID, MatrixID);
```

TupleID	Value
1	2
2	1
3	3
4	4
5	5
6	1

Layer Output from Q1

JOIN

MatrixID	OrderID	TupleID
1	1	1
1	2	2
1	3	3
2	1	3
2	2	4
2	3	5

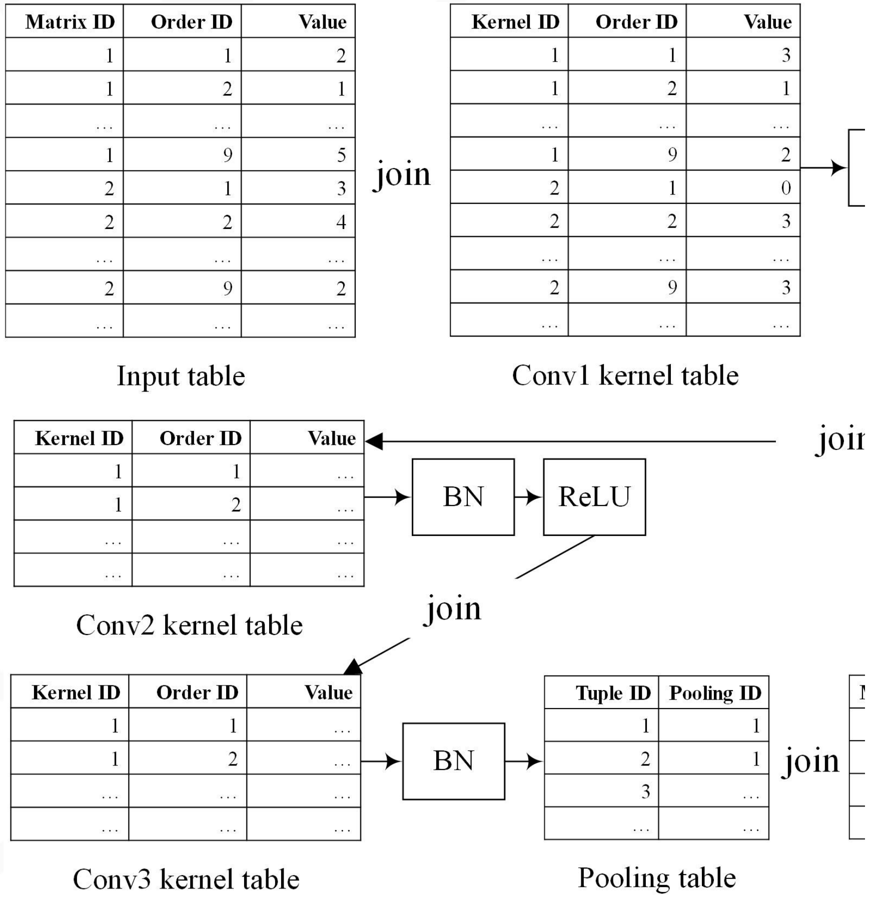
Kernel Mapping Table

=

MatrixID	OrderID	Value
1	1	2
1	2	1
1	3	3
2	1	3
2	2	4
2	3	5

Feature Map Table

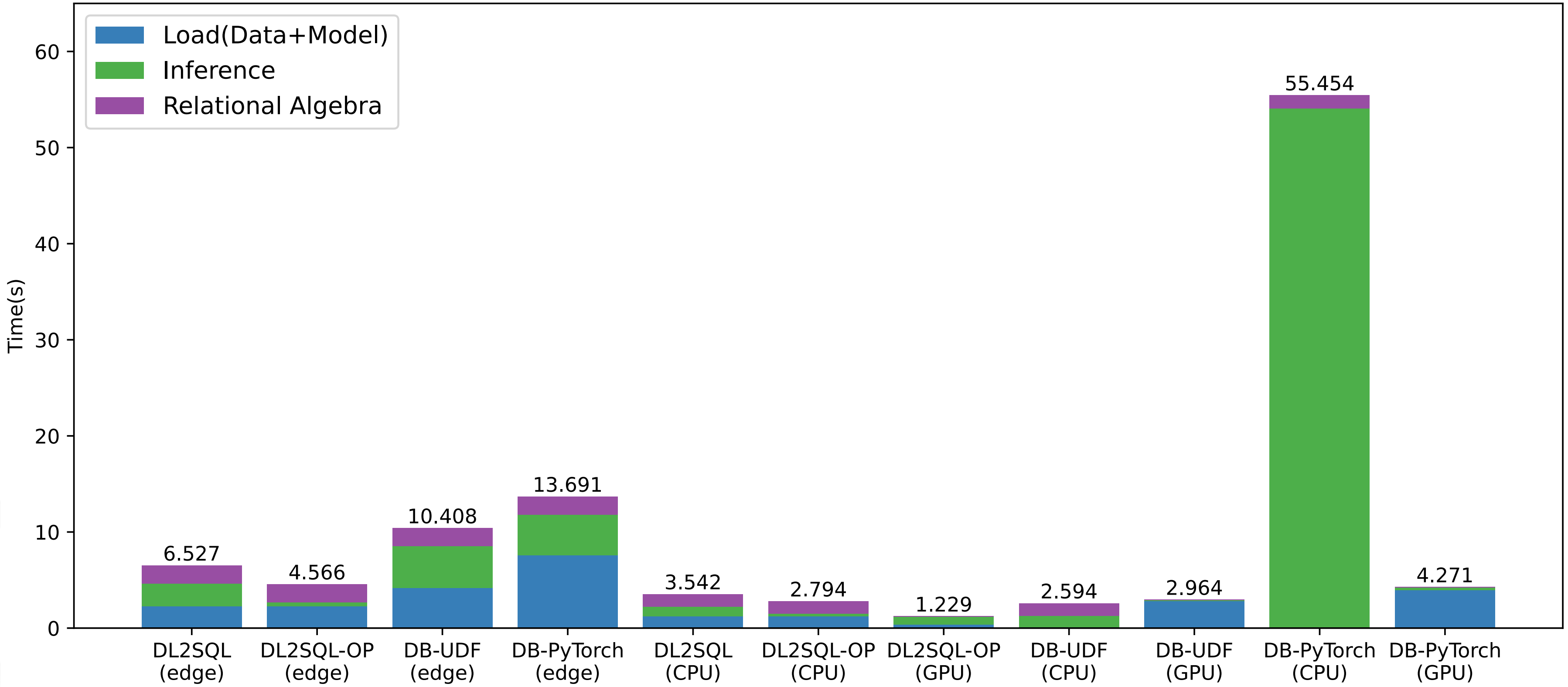
3 Implement Detail (take CNN as an example):



Neural Blocks	Variants	SQL Support
Pooling	Average Pooling Max Pooling	Supported Supported
Activation	ReLU Sigmoid	Supported Supported
Normalization	Batch Normalization Instance Normalization	Supported Supported
Full Connection	N.A.	Supported
Convolution	N.A.	Supported
Deconvolution	N.A.	Supported
Residual Block	N.A.	Supported
Indentity Block	N.A.	Supported
Dense Block	N.A.	Supported
Attention Block	Basic Attention Self Attention	Supported Unsupported
RNN	LSTM GRU	Unsupported Unsupported
Graph Convolution	N.A.	Supported by Graph DB

4 *EVALUATION: Performance in different scenarios*

- *Average Performance of Multiple Queries on Different Devices with Selectivity=0.01%*



4 EVALUATION: Performance in different scenarios

- Performance Comparison with Different Selectivity on Edge Server

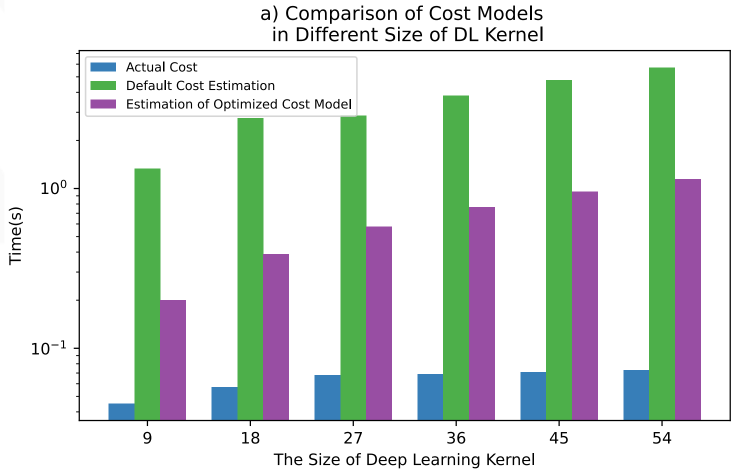
	DL2SQL-OP			DB-UDF			DB-PyTorch		
Selectivity(%)	*Inference(s)*	*Loading(s)*	*All(s)*	*Inference(s)*	*Loading(s)*	*All(s)*	*Inference(s)*	*Loading(s)*	*All(s)*
0.01	**0.441**	**2.256**	**2.697**	4.558	4.617	9.175	4.199	7.589	11.788
0.1	**0.263**	**1.129**	**2.783**	4.63	4.631	9.261	4.2	7.589	11.789
0.2	**0.618**	**2.175**	**2.793**	4.54	4.531	9.071	4.199	7.591	11.79
0.4	**0.857**	**2.259**	**3.116**	4.516	4.41	8.926	4.21	7.592	11.802
0.6	**1.308**	**2.261**	**3.569**	4.341	4.277	8.618	4.23	7.594	11.824
0.8	**2.254**	**2.231**	**4.485**	4.437	4.23	8.667	4.24	7.597	11.837
1	4.651	**2.174**	**6.825**	4.568	4.292	8.86	**4.24**	7.599	11.839

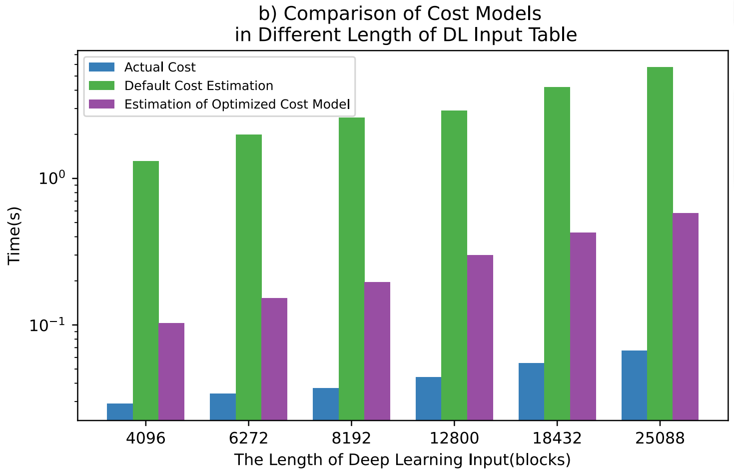
- Performance Comparison with Different Model Depths on Edge Server

Model Configuration		DL2SQL-OP			DB-UDF			DB-PyTorch		
Depth	*Parameters*	*Inference(s)*	*Loading(s)*	*All(s)*	*Inference(s)*	*Loading(s)*	*All(s)*	*Inference(s)*	*Loading(s)*	*All(s)*
5	828418	**0.138**	**1.198**	**1.336**	2.282	2.243	4.525	2.478	1.957	4.435
10	3781890	**0.165**	2.341	**2.506**	2.291	**2.274**	4.565	1.982	2.524	4.506
15	6734850	**0.199**	3.237	**3.436**	2.29	**2.263**	4.553	1.987	2.558	4.545
20	9687810	**0.227**	4.555	4.782	2.309	**2.282**	4.591	1.975	2.572	**4.547**
25	12640770	**0.258**	4.508	4.766	2.321	**2.308**	4.629	2.001	2.596	**4.597**
30	15593730	**0.289**	4.306	4.595	2.321	**2.327**	4.648	1.959	2.542	**4.501**
35	18546690	**0.319**	4.593	4.912	2.332	**2.341**	4.673	1.961	2.543	**4.504**
40	20909570	**0.348**	6.194	6.542	2.343	**2.358**	4.701	1.969	2.546	**4.515**

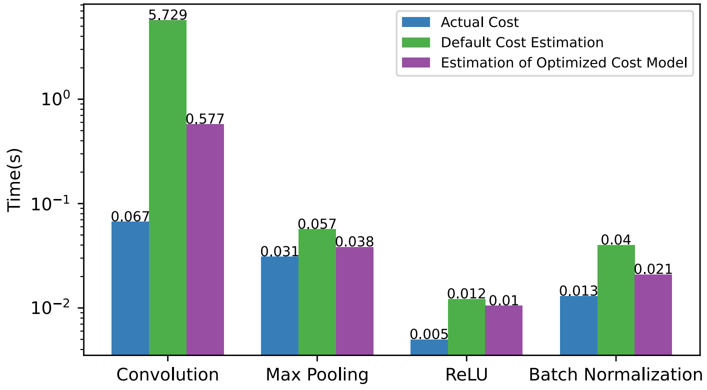
4 EVALUATION: Cost Model

- *Comparison of Different Cost Models in Collaborative Query*

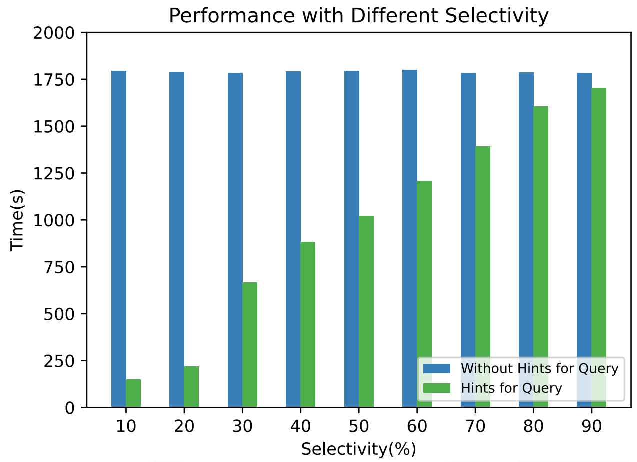




- *Comparison of Different Neural Operators*



- *Performance of Hints for Collaborative Query*



5 Conclusion

- *We investigate a new type of query, the collaborative query, and compare three possible processing strategies.*
- *We propose a new tight integration approach, DL2SQL, which transforms the neural model inference into pure SQL statements by implementing popular neural operators as SQL queries..*
- *We propose our customized cost model and apply hint rules for the database's optimizer to choose a proper query plan.*

Thank you!