Microsoft

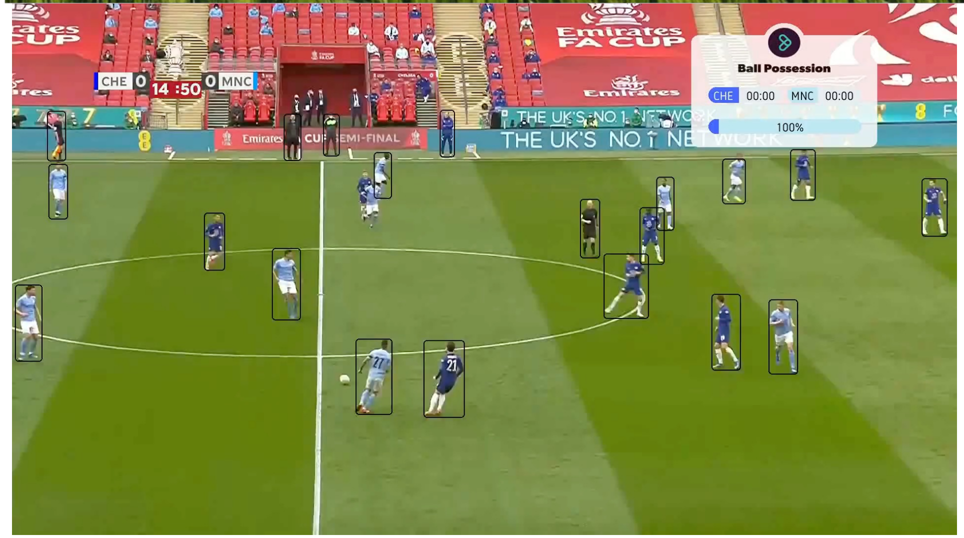
The Tensor Data Platform

Towards an AI-centric Database System

Apurva Gandhi, Yuki Asada, Victor Fu, Advitya Gemawat, Lihao Zhang, Rathijit Sen, Carlo Curino, Jesús Camacho-Rodríguez, **Matteo Interlandi**

AI is growing…and having an impact on applications…and DBMS

Enter your favorite chart showing how AI is taking over the world



Unlocking the value of unstructured data at scale using BigQuery ML and object tables

Anatomy of next gen data-driven applications

1. Support for multimodal data (image, video, relational, audio, etc.)
   - Not many relational system with proper image/video/etc. support
   - Many specialized system are moving towards supporting “scalar” queries

2. Tight integration and interoperability with ML
   - Most systems either (partially) re-implement ML features in SQL …
   - … Or call external ML runtimes

3. Native support for hardware acceleration
   - Most systems are built on single vendor tech (CUDA)
   - Supporting other stacks (AMD, Apple, etc.) requires nontrivial engineering effort

Claim: Building a data engine with all three is hard!

Tensor Runtimes PyTorch

1. Support for multimodal data
   - Thanks to the Tensor abstraction

2. Native support for hardware acceleration
   - Large open-source communities with HW vendors involvement

3. Tight integration and interoperability with ML
   - ML capabilities embedded into the system and language (e.g., autodiff)

? **Question: Can we build a database on top of tensor runtimes?**

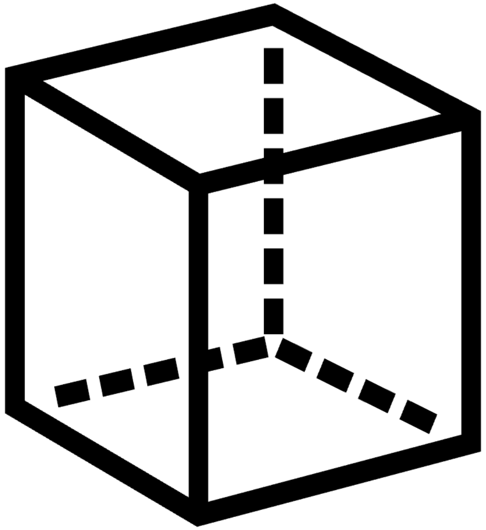
AI-centric Database: Outline

1. Support for multimodal data

2. Native support for hardware acceleration

3. Tight integration and interoperability with ML

Tensor data representation



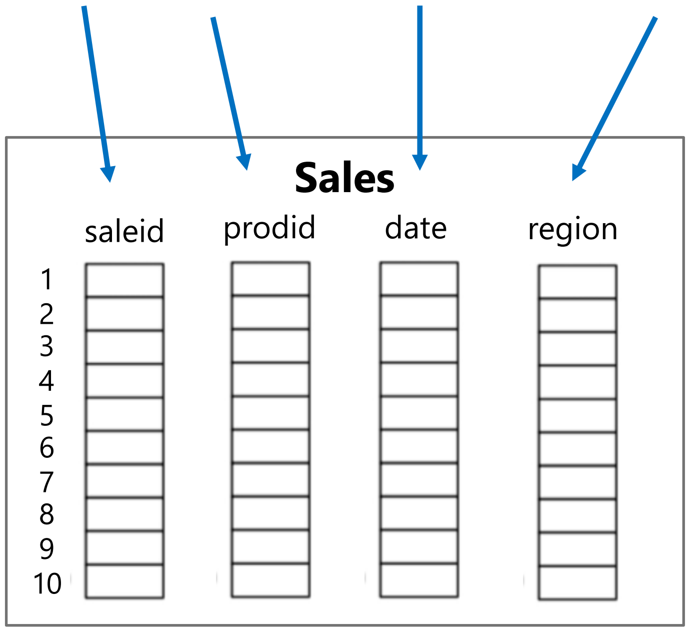
Def Tensor:

A multidimensional matrix that is a cornerstone data structure in AI

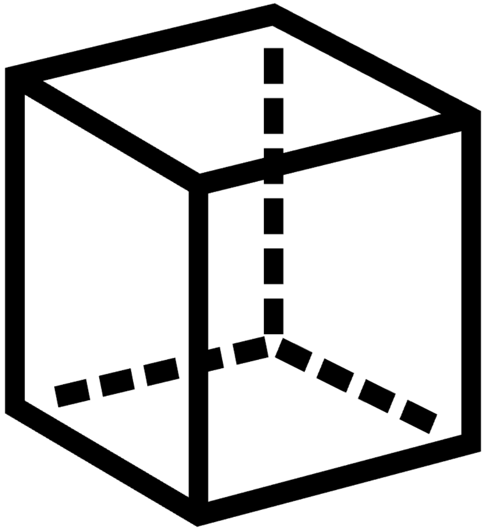
Numeric as 1-d tensor

Dates as 1-d numeric tensor

Strings as UTF-8 2-d tensor (N x max_len)

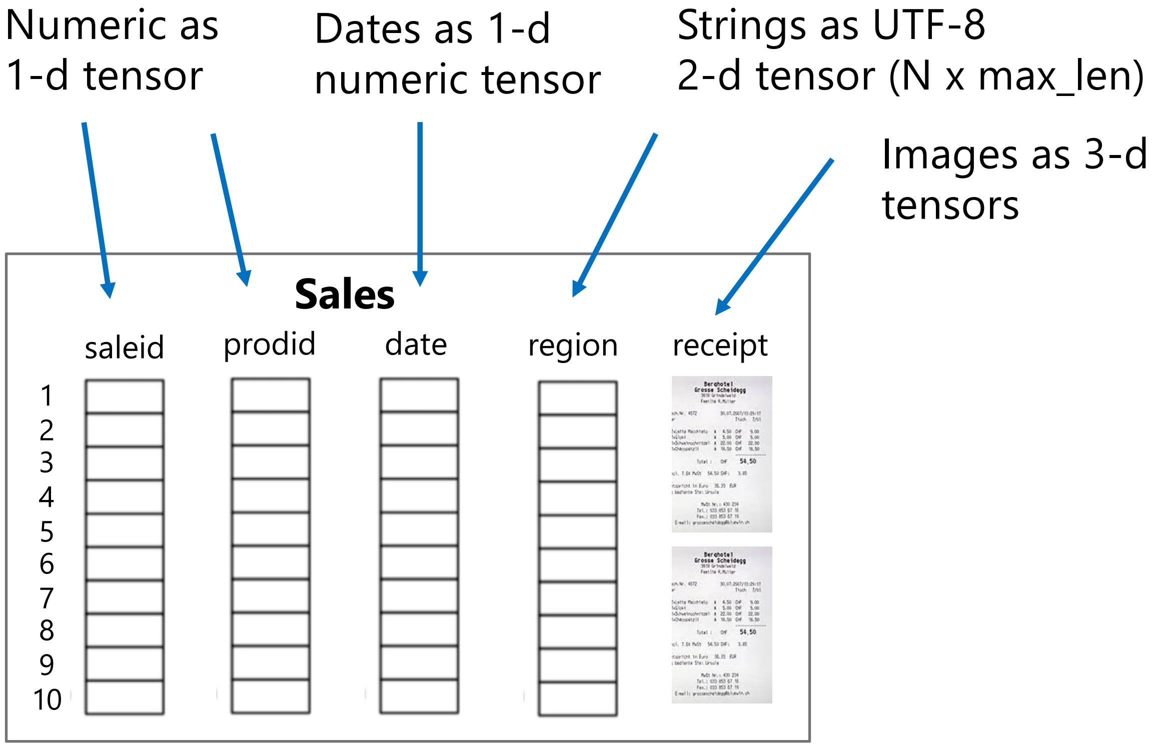


Tensor data representation

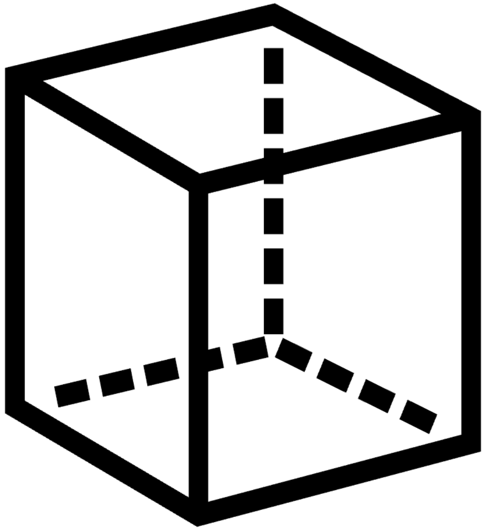


Def Tensor:

A multidimensional matrix that is a cornerstone data structure in AI

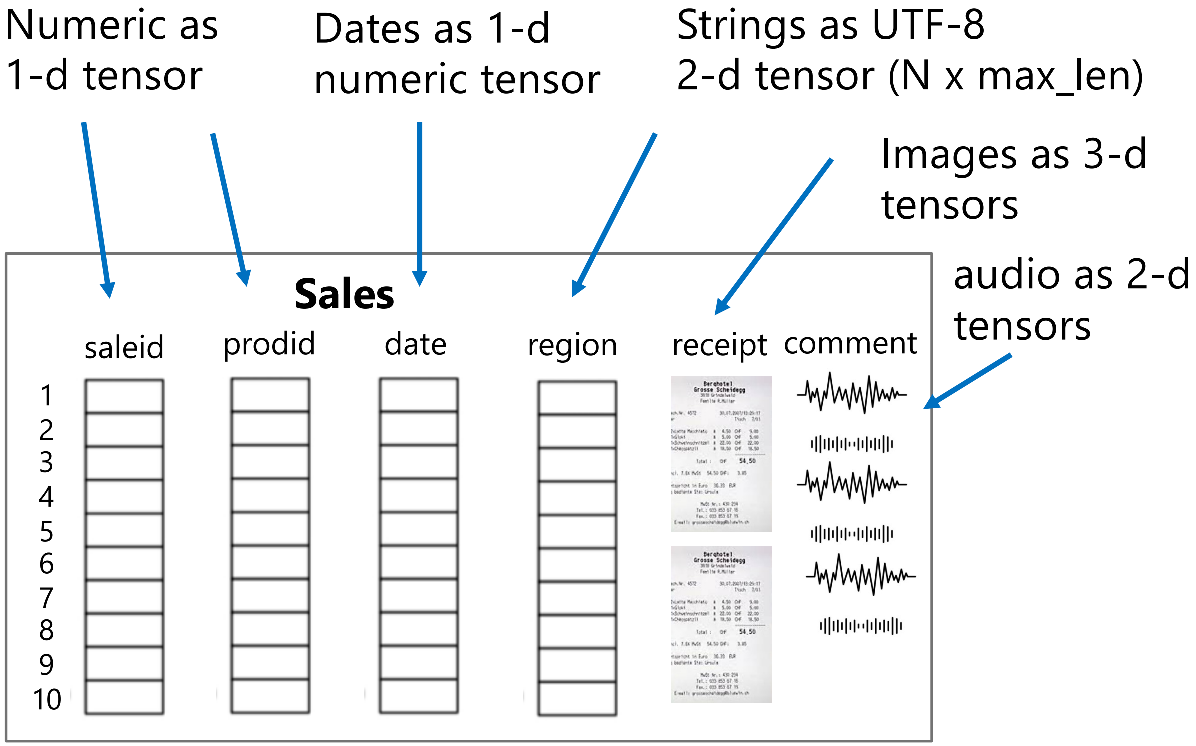


Tensor data representation



Def Tensor:

A multidimensional matrix that is a cornerstone data structure in AI



We leverage torch, torchaudio, torchvideo, etc, for loading data into tensor format

We have our own custom tensor class: EncodedTensor = tensor + metadata

PlainEncoding,

DictionaryEncoding (data tensor + 2-d dictionary metadata tensor)

ProbabilisticEncoding (data tensor + a domain dictionary)

…

Microsoft

SQL on Images Demo

demo-image-search.ipynb M

notebooks > sql > image_search > demo-image-search.ipynb > Quering images > Top-k similarity search

Quering images

The goal of this notebook is to show how image data can be loaded on TQP and how we can use TQP capabilities to query images

1. Setup

2. Filter images based on Natural Language Query

AI-centric Database: Outline

1. Support for multimodal data

2. Native support for hardware acceleration

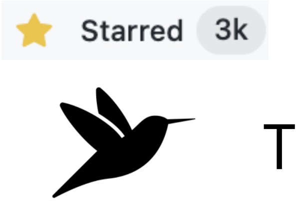
3. Tight integration and interoperability with ML

The Tensor Data Platform (TDP)

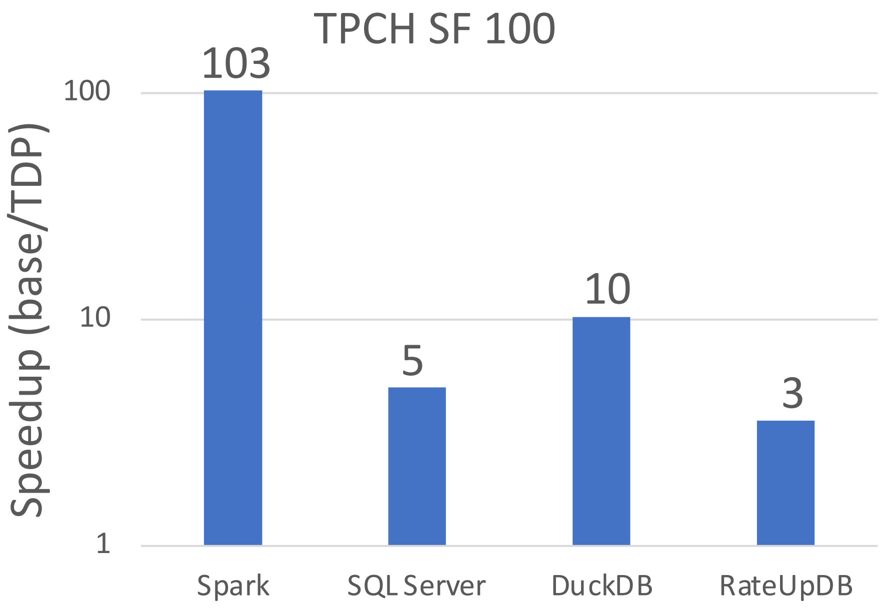
In process and 100% Python!

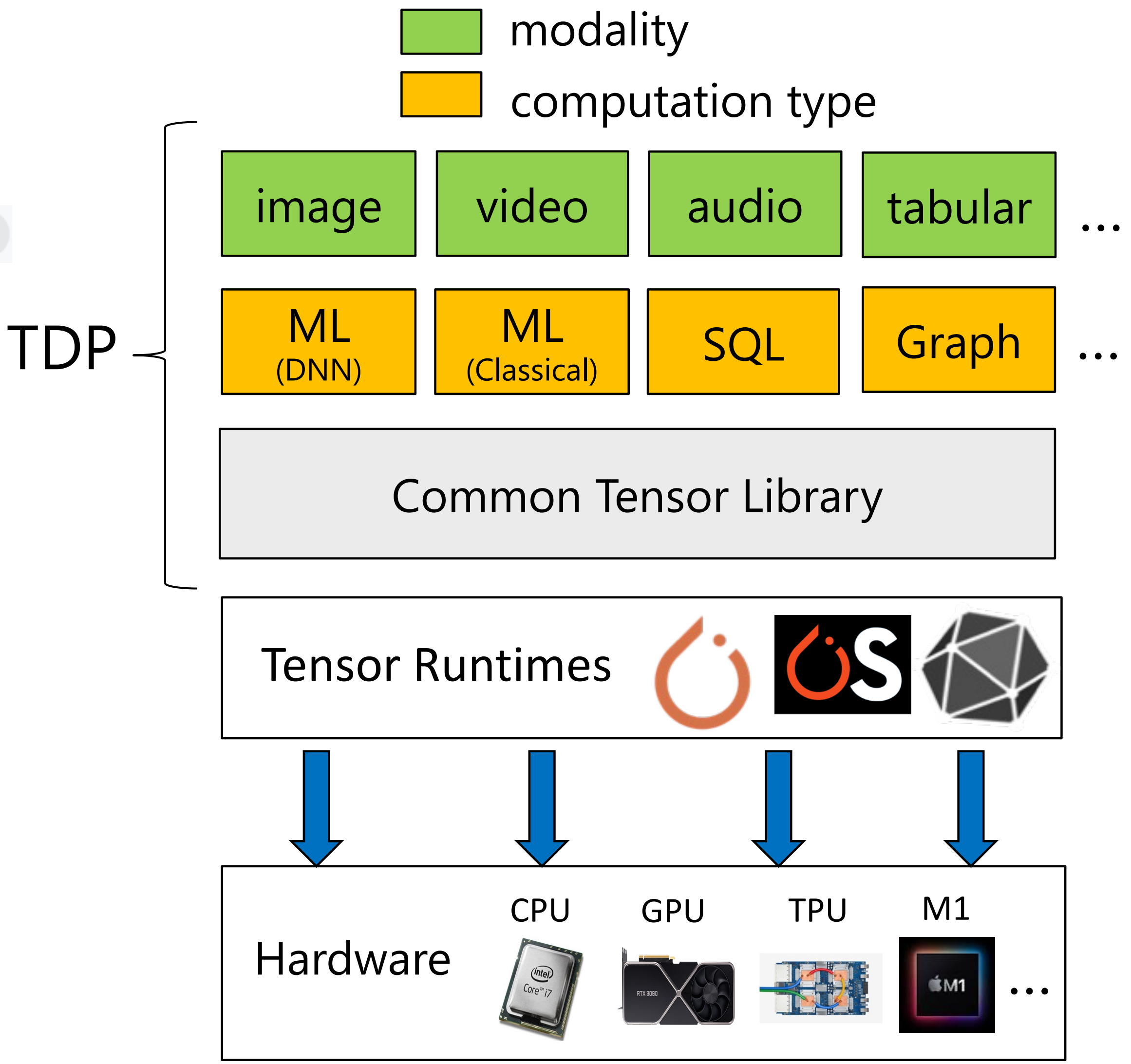
Classical ML Inference: Hummingbird

SQL: TQP



Performance highlights





AI-centric Database: Outline

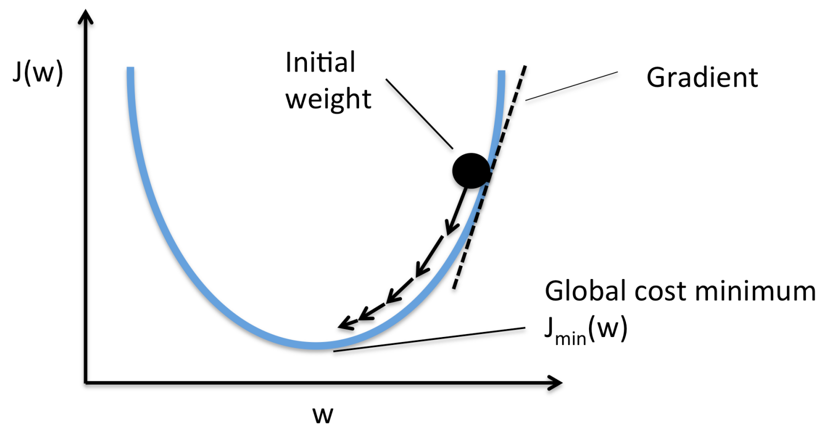
1. Support for multimodal data

2. Native support for hardware acceleration

3. Tight integration and interoperability with ML

SQL as a declarative language for Differentiable Programming

Gradients are the staple mechanism by which we *learn* in machine learning.



Tensor runtimes have a remarkable tool to compute gradients **Automatic Differentiation**

TDP extends SQL by taking advantage of automatic differentiation in PyTorch

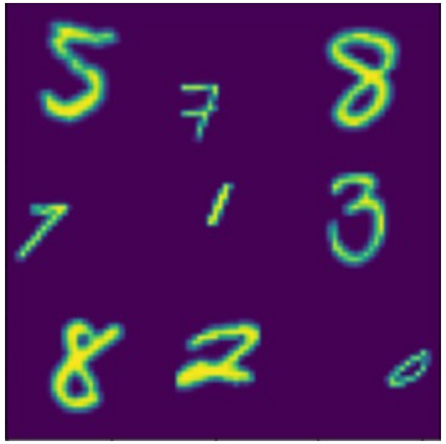
Particularly, we add the following to SQL:

1. **Trainable User Defined Functions (UDFs) and Table Valued Functions (TVFs)**
2. **Differentiable Relational Operators (e.g., Differentiable Group By, Aggregation, Filters, etc.)**

Trainable SQL Queries

We can execute SQL queries that combines trainable operations with relational operators.

MNISTGrid Dataset



Digit	Size	Count
0	Small	1
	Large	0
1	Small	1
	Large	0
2	Small	0
	Large	1
3	Small	0
	Large	1
4	Small	0
	Large	0
5	Small	0
	Large	1
6	Small	0
	Large	0
7	Small	2
	Large	0
8	Small	0
	Large	2
9	Small	0
	Large	0

MNISTGrid Task

Compute the grouped (Digit, Size) counts from the image.

Trainable Query

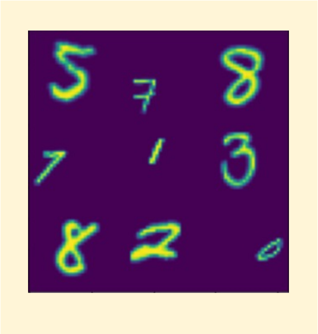
```sql
SELECT Digit, Size, COUNT(*)
FROM parseMNISTGrid(MNISTGrid)
GROUP BY Digit, Size
```

Anatomy of a Trainable Query

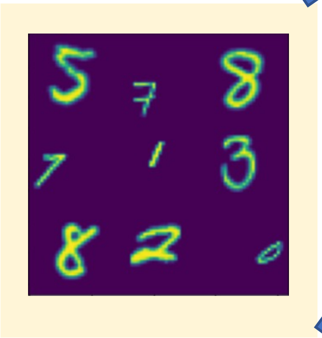
```
SELECT Digit, Size, COUNT(*)
FROM parseMNISTGrid(MNISTGrid)
GROUP BY Digit, Size
```

Anatomy of a Trainable Query

```
SELECT Digit, Size, COUNT(*)
FROM parseMNISTGrid(MNISTGrid)
GROUP BY Digit, Size
```



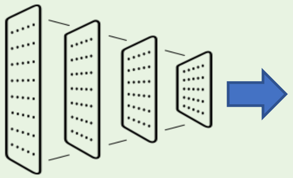
Anatomy of a Trainable Query



Digit Parser

Digit

0	1	2	3	4	5	6	7	8	9
0.02	0	0.05	0	0	0.90	0	0.03	0	0
0	0.2	0	0	0.05	0	0	0.75	0	0
0	0.04	0	0.05	0	0	0	0	0.91	0
0	0.3	0.1	0	0	0	0	0.6	0	0
0	0.8	0	0	0	0	0	0.1	0	0.1
0	0	0	0.82	0	0	0.09	0	0	0.09
0	0.15	0.15	0	0	0	0	0	0.7	0
0.08	0	0.76	0	0	0.08	0	0	0.08	0
0.55	0	0	0	0	0	0	0	0.35	0.10



Size Parser

Size

small	large
0.2	0.8
0.83	0.17
0.3	0.7
0.9	0.1
0.95	0.05
0.3	0.7
0.21	0.79
0.34	0.66
0.85	0.15

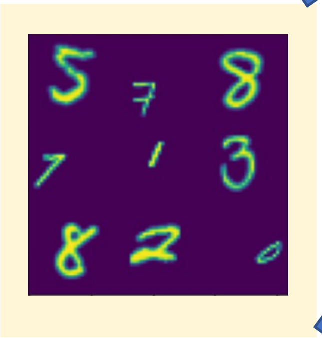
Trainable UDF

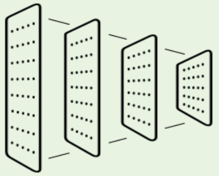
```
SELECT Digit, Size, COUNT(*)
FROM parseMNISTGrid(MNISTGrid)
GROUP BY Digit, Size
```

```
digit_parser = CNN(out_classes=10).to(device)
size_parser = CNN(out_classes=2).to(device)

@tdp_udf("Digit float, Size float")
def parseMNISTGrid(x: torch.Tensor) -> torch.Tensor:
    # Break up grid into a batch of 9 images
    grid = rearrange(x[0], "(h1 h2) (w1 w2) -> (h1 w1) 1 h2 w2", h1=3, w1=3)

    # Parse digits from images
    parsed_digits = digit_parser(grid)
    digit_domain = np.arange(10)
    encoded_digits = ProbabilisticEncoding.encode(parsed_digits, digit_domain)

    # Parse size from images
    parsed_sizes = size_parser(grid)
    size_domain = np.arange(2)
    encoded_sizes = ProbabilisticEncoding.from_encoded_data(parsed_sizes, size_domain)

    return encoded_digits, encoded_sizes
```

Anatomy of a Trainable Query

```
SELECT Digit, Size, COUNT(*)
FROM parseMNISTGrid(MNISTGrid)
GROUP BY Digit, Size
```

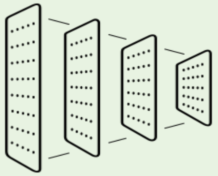




Digit Parser

Digit

0	1	2	3	4	5	6	7	8	9
0.02	0	0.05	0	0	0.90	0	0.03	0	0
0	0.2	0	0	0.05	0	0	0.75	0	0
0	0.04	0	0.05	0	0	0	0	0.91	0
0	0.3	0.1	0	0	0	0	0.6	0	0
0	0.8	0	0	0	0	0	0.1	0	0.1
0	0	0	0.82	0	0	0.09	0	0	0.09
0	0.15	0.15	0	0	0	0	0	0.7	0
0.08	0	0.76	0	0	0.08	0	0	0.08	0
0.55	0	0	0	0	0	0	0	0.35	0.10



Size Parser

Size

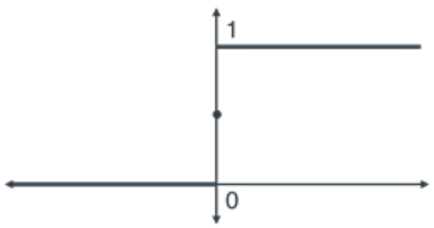
small	large
0.2	0.8
0.83	0.17
0.3	0.7
0.9	0.1
0.95	0.05
0.3	0.7
0.21	0.79
0.34	0.66
0.85	0.15

Trainable UDF

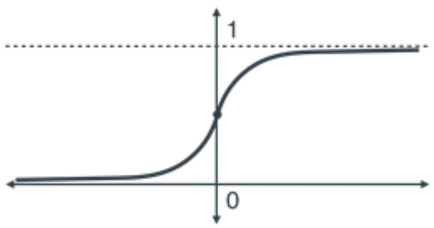
soft_groupby

soft_count

Step function (discrete)



Sigmoid function (continuous)

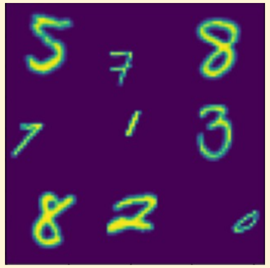


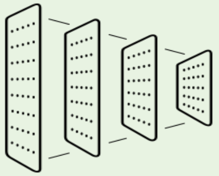
Differentiable Relational Operators

Digit	Size	Count
0	Small	1
	Large	0
1	Small	1
	Large	0
2	Small	0
	Large	1
3	Small	0
	Large	1
4	Small	0
	Large	0
5	Small	0
	Large	1
6	Small	0
	Large	0
7	Small	2
	Large	0
8	Small	0
	Large	2
9	Small	0
	Large	0

Anatomy of a Trainable Query

```
SELECT Digit, Size, COUNT(*)
FROM parseMNISTGrid(MNISTGrid)
GROUP BY Digit, Size
```

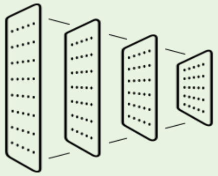




Digit Parser

Digit

0	1	2	3	4	5	6	7	8	9
0.02	0	0.05	0	0	0.90	0	0.03	0	0
0	0.2	0	0	0.05	0	0	0.75	0	0
0	0.04	0	0.05	0	0	0	0	0.91	0
0	0.3	0.1	0	0	0	0	0.6	0	0
0	0.8	0	0	0	0	0	0.1	0	0.1
0	0	0	0.82	0	0	0.09	0	0	0.09
0	0.15	0.15	0	0	0	0	0	0.7	0
0.08	0	0.76	0	0	0.08	0	0	0.08	0
0.55	0	0	0	0	0	0	0	0.35	0.10



Size Parser

Size

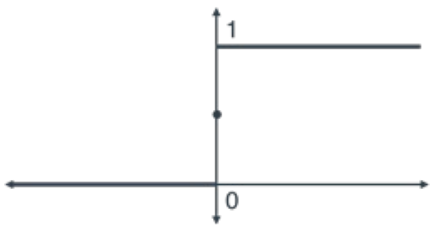
small	large
0.2	0.8
0.83	0.17
0.3	0.7
0.9	0.1
0.95	0.05
0.3	0.7
0.21	0.79
0.34	0.66
0.85	0.15

soft_groupby

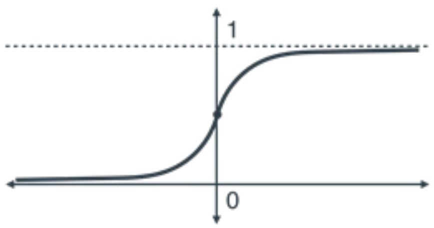
soft_count

Digit	Size	Count
0	Small	1
	Large	0
1	Small	1
	Large	0
2	Small	0
	Large	1
3	Small	0
	Large	1
4	Small	0
	Large	0
5	Small	0
	Large	1
6	Small	0
	Large	0
7	Small	2
	Large	0
8	Small	0
	Large	2
9	Small	0
	Large	0

Step function
(discrete)



Sigmoid function
(continuous)



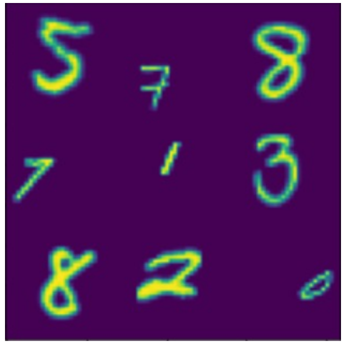
Trainable UDF

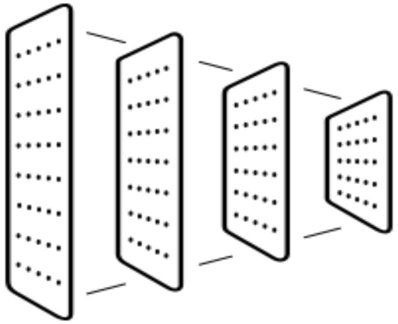
Differentiable Relational Operators

The query combines neural and relational operators and is end-to-end differentiable

The alternative: pure Deep Learning

The standard way to tackle this problem would be to pose it as a multiple regression problem with a single monolithic neural network





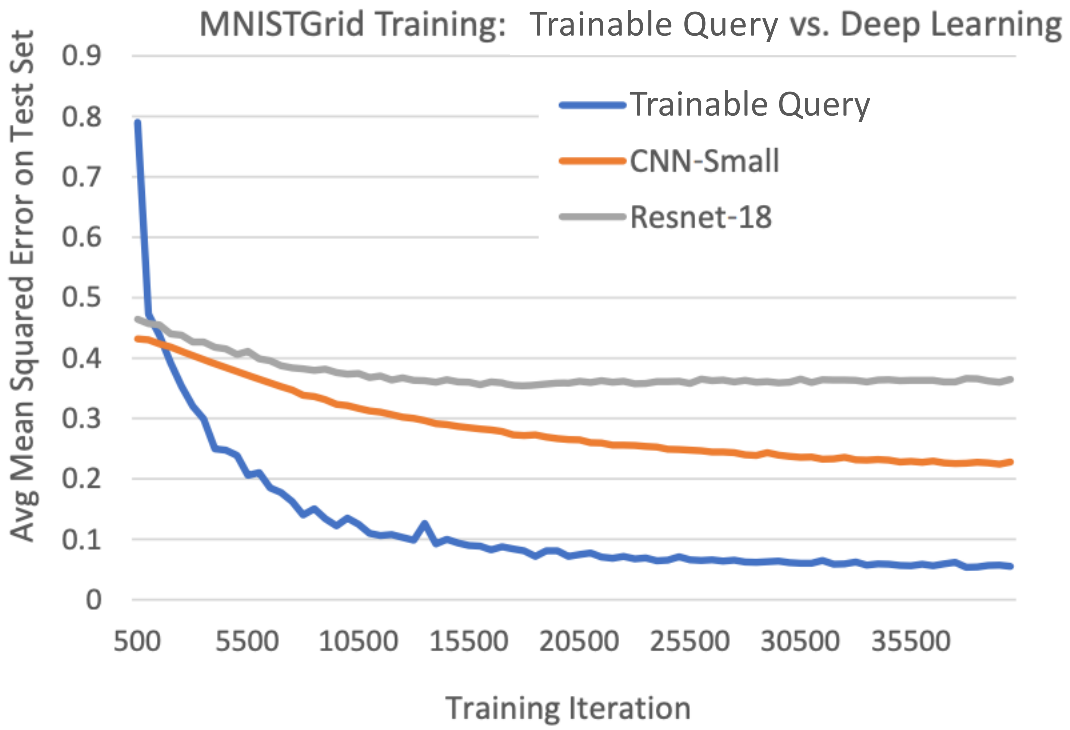
Digit	Size	Count
0	Small	1
	Large	0
1	Small	1
	Large	0
2	Small	0
	Large	1
3	Small	0
	Large	1
4	Small	0
	Large	0
5	Small	0
	Large	1
6	Small	0
	Large	0
7	Small	2
	Large	0
8	Small	0
	Large	2
9	Small	0
	Large	0

Disadvantages:

1. Entanglement of tasks (cannot separate digit classification from size classification or aggregation)
2. Cannot generalize to other tasks
3. Needs to learn from scratch what it means to group and count

Trainable Query vs pure Deep Learning

- Datasets:
  - MNISTGrid Train/Test: 5000/1000 Grids
- Training Hyperparameters (Fixed):
  - Learning Rate = 0.0001
  - Training Iterations = 40,000 iterations
- Architecture (Varied):
  - TDP Trainable Query (860K Parameters)
  - Pure Deep Learning CNN-Small (850K Parameters)
  - Resnet-18 (11.1M Parameters)
- 5 runs per architecture



Our approach trains significantly faster than a purely deep learning model

Our SQL can declaratively express Neurosymbolic [1] systems that are end-to-end trainable

[1] Neurosymbolic AI CACM oct 2022

Summary

The space of AI-powered databases is heating up

AI-centric Database could be a leap forward. Free-ride on:

1. $B of HW/SW investments for AI
2. Multimodal support
3. Seamless integration with latest and biggest ML models
4. Novel querying paradigms such as trainable queries

Exciting future directions

1. TensorFrame API
2. Expressing some ML tasks in a more natural way
    - Learning from Label Proportions

Microsoft

Microsoft Azure Data
Gray Systems Lab

Thank you!

https://aka.ms/gsl

ML-first user experience

```sql
SELECT images
FROM Attachments
WHERE image_text_similarity("dog", images) > 0.80
```

ML within SQL: UDF-based programming model

- We use UDF to access the tensor API
- Still end-to-end on HW accelerators

```python
model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

@tdp_udf("float")
def image_text_similarity(query: str, images: torch.Tensor) -> torch.Tensor:
    inputs = processor(text=[query], images=images, return_tensors="pt", padding=True)
    inputs.to(device)
    outputs = model(**inputs)
    scores = outputs.logits_per_image.flatten() / 30
    return scores
```

Microsoft

ML-first user experience

```sql
SELECT images
FROM Attachments
WHERE image_text_similarity("dog", images) > 0.80
```

ML within SQL: UDF-based programming model

- We use UDF to access the tensor API
- Still end-to-end on HW accelerators

```python
model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

@tdp_udf("float")
def image_text_similarity(query: str, images: torch.Tensor) -> torch.Tensor:
    inputs = processor(text=[query], images=images, return_tensors="pt", padding=True)
    inputs.to(device)
    outputs = model(**inputs)
    scores = outputs.logits_per_image.flatten() / 30
    return scores
```

SQL within ML: Embedding queries into PyTorch programs

- Use the right tool for the right task
- Thanks to trainable SQL queries

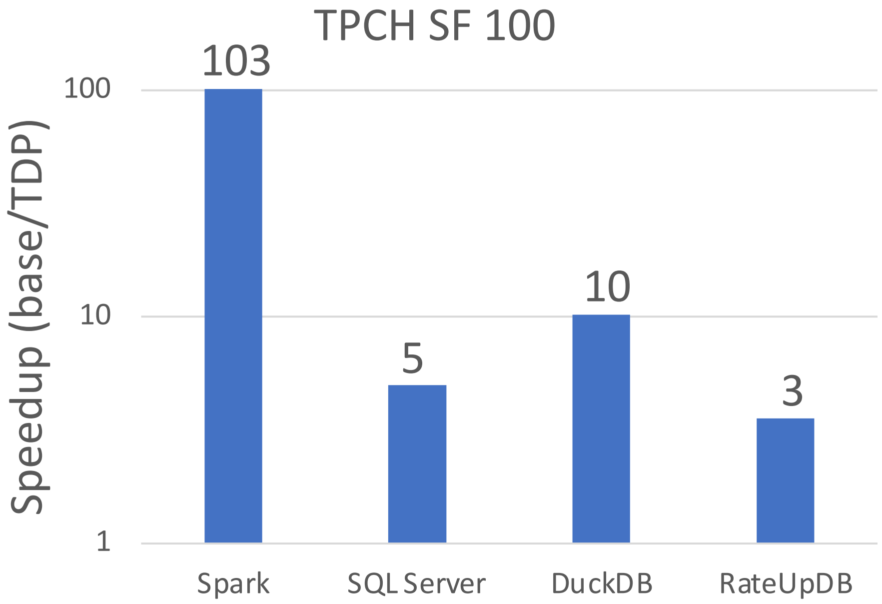
```python
def train(compiled_query, num_iterations, optimizer, mnist_grids, target_counts):
    for i in range(num_iterations):
        optimizer.zero_grad()

    # Register MNISTGrid and perform inference with the query
    tqp.sql.register_tensor(mnist_grids[i], "MNIST_Grid")
    predicted_counts = compiled_query.run()

    # Compute loss. Here we use MSE between the counts.
    loss = ((predicted_counts - target_counts[i])**2).mean()

    # Backpropagate and perform optimization step
    loss.backward()
    optimizer.step()
```

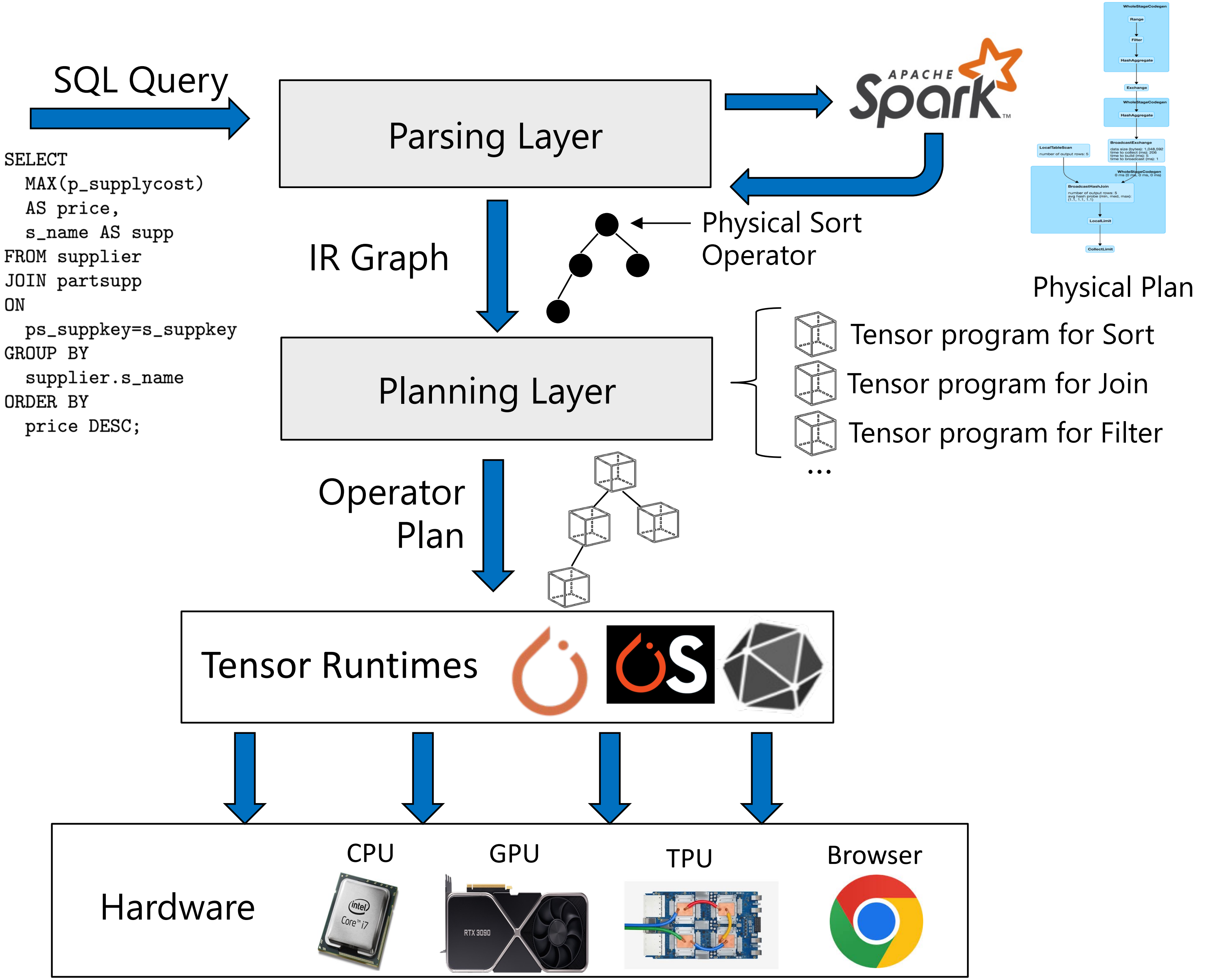
TQP

100% Python

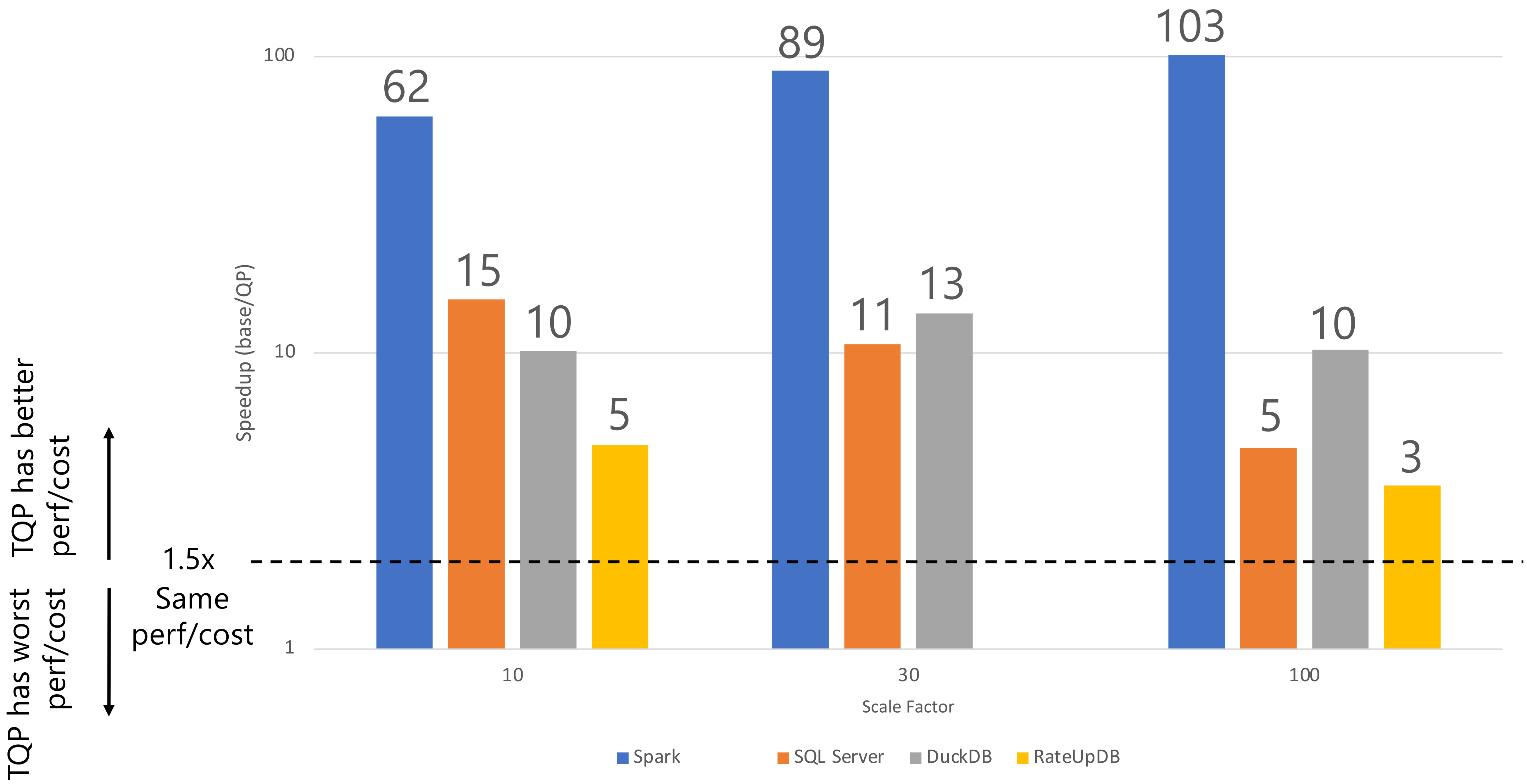
TQP supports the full TPCH benchmark

Performance highlights





TQP Scalability Comparison



TQP: A100 with 80GB. Spark/SQLServer/DuckDB: 32 cores machine with 256GB. RateUp: Nvidia Quadro RTX 8000

Microsoft

Differentiable Grouped Aggregation

Let's see how we might make the "Group By + Aggregation" operation differentiable.

Inventory

Fruit	Vegetable	Price
apple	carrot	4.0
banana	carrot	2.0
apple	carrot	4.0
banana	potato	3.5

Query

```
SELECT Fruit, Vegetable, COUNT(*)
FROM Inventory
GROUP BY Fruit, Vegetable
```

Query Answer

Fruit	Vegetable	Count
apple	carrot	2
apple	potato	0
banana	carrot	1
banana	potato	1

Differentiable Grouped Aggregation

Let's see how we might make the "Group By + Aggregation" operation differentiable.

Inventory

Fruit	Vegetable	Price
apple	carrot	4.0
banana	carrot	2.0
apple	carrot	4.0
banana	potato	3.5

Query

```
SELECT Fruit, Vegetable, COUNT(*)
FROM Inventory
GROUP BY Fruit, Vegetable
```

We can do this in three steps:

1. Relax discrete data to continuous representation.
2. Create masks corresponding to each group.
3. Perform aggregation using the mask and data.

Differentiable Grouped Aggregation

Let's see how we might make the "Group By + Aggregation" operation differentiable.

Inventory

Fruit	Vegetable	Price
apple	carrot	4.0
banana	carrot	2.0
apple	carrot	4.0
banana	potato	3.5

Query

```
SELECT Fruit, Vegetable, COUNT(*)
FROM Inventory
GROUP BY Fruit, Vegetable
```

We can do this in three steps:

1. Relax discrete data to continuous representation. (Assume data is pre-encoded)
2. Create masks corresponding to each group. (Needs to be differentiable)
3. Perform aggregation using the mask and data. (Needs to be differentiable)

Differentiable Grouped Aggregation

Step 1: Relax discrete data to continuous representation.

Inventory

Fruit	Vegetable	Price
apple	carrot	4.0
banana	carrot	2.0
apple	carrot	4.0
banana	potato	3.5

Inventory

Fruit		Vegetable		Price
1.	0.	1.	0.	4.0
0.	1.	1.	0.	2.0
1.	0.	1.	0.	4.0
0.	1.	0.	1.	3.5
apple	banana	carrot	potato	

We can use One Hot Encoding (OHE) for categorical columns.

We assume data is pre-encoded to this format before being fed into our differentiable operator.

Differentiable Grouped Aggregation

Step 2: Create masks corresponding to each group.

Inventory

Fruit		Vegetable		Price
1.	0.	1.	0.	4.0
0.	1.	1.	0.	2.0
1.	0.	1.	0.	4.0
0.	1.	0.	1.	3.5
apple	banana	carrot	potato	

Mask for Group: (apple, carrot)

(apple, carrot)
1.0
0.0
1.0
0.0

With the OHE strategy of categorical data representation, creating a group mask requires only element-wise product (which is differentiable).

Differentiable Grouped Aggregation

Step 3: Perform aggregation using the mask and data.

Inventory

Fruit		Vegetable		Price
1.	0.	1.	0.	4.0
0.	1.	1.	0.	2.0
1.	0.	1.	0.	4.0
0.	1.	0.	1.	3.5
apple	banana	carrot	potato	

Mask for Group: (apple, carrot)

(apple, carrot)
1.0
0.0
1.0
0.0

Aggregation

$\sum \rightarrow$ 2.0

Differentiable Grouped Aggregation (GROUP BY + COUNT)

Step 3: Perform aggregation using the mask and data.

Inventory

Fruit		Vegetable		Price
1.	0.	1.	0.	4.0
0.	1.	1.	0.	2.0
1.	0.	1.	0.	4.0
0.	1.	0.	1.	3.5
apple	banana	carrot	potato	

Mask for Group: (apple, carrot)

(apple, carrot)
1.0
0.0
1.0
0.0

Aggregation

$\Sigma \rightarrow 2.0$

Query Answer

Fruit	Vegetable	Count
apple	carrot	2
apple	potato	0
banana	carrot	1
banana	potato	1

We have only used product and sum, both of which are differentiable.

Differentiable Grouped Aggregation (GROUP BY + SUM)

Step 3: Perform aggregation using the mask and data.

Inventory

Fruit		Vegetable		Price
1.	0.	1.	0.	4.0
0.	1.	1.	0.	2.0
1.	0.	1.	0.	4.0
0.	1.	0.	1.	3.5
apple	banana	carrot	potato	

Summands for Group:
(apple, carrot)

(apple, carrot)
4.0
0.0
4.0
0.0

Aggregation

Σ → 8.0

Query

```
SELECT Fruit, Vegetable, SUM(Price)
FROM Inventory
GROUP BY Fruit, Vegetable
```

Query Answer

Fruit	Vegetable	SUM(Price)
apple	carrot	8.0
apple	potato	0.0
banana	carrot	2.0
banana	potato	3.5

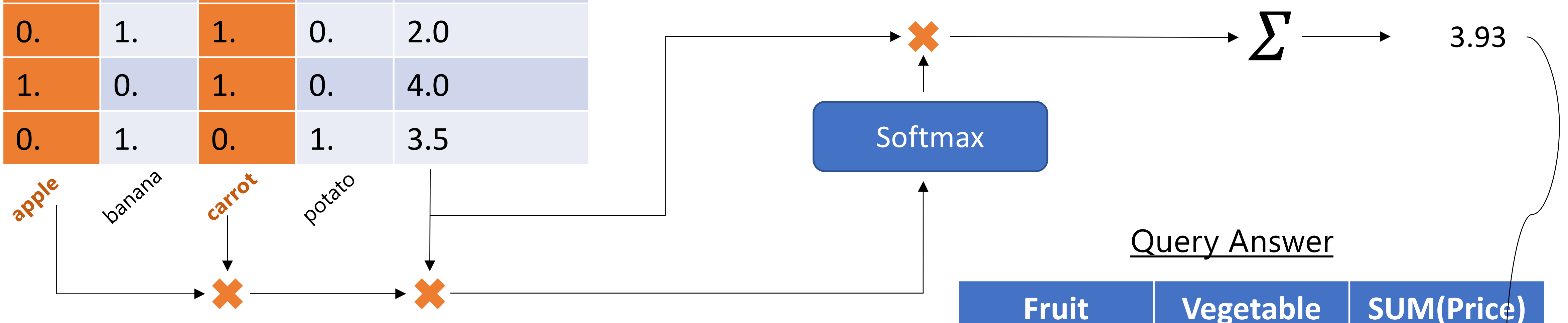
Differentiable Grouped Aggregation (GROUP BY + MAX)

Step 3: Perform aggregation using the mask and data.

Inventory

Fruit		Vegetable		Price
1.	0.	1.	0.	4.0
0.	1.	1.	0.	2.0
1.	0.	1.	0.	4.0
0.	1.	0.	1.	3.5

Aggregation



Query Answer

Fruit	Vegetable	SUM(Price)
apple	carrot	3.93
banana	carrot	2.0
banana	potato	3.5

Query

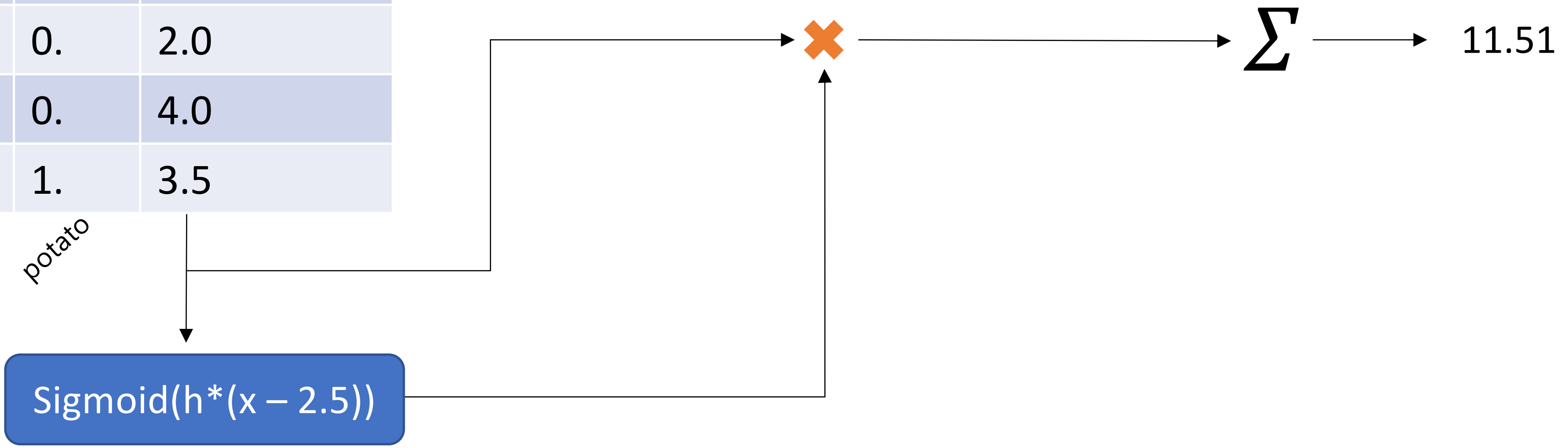
```
SELECT Fruit, Vegetable, MAX(Price)
FROM Inventory
GROUP BY Fruit, Vegetable
```

Differentiable Filtered Aggregation (WHERE + SUM)

Inventory

Fruit		Vegetable		Price
1.	0.	1.	0.	4.0
0.	1.	1.	0.	2.0
1.	0.	1.	0.	4.0
0.	1.	0.	1.	3.5
apple	banana	carrot	potato	

Aggregation



Query

```sql
SELECT SUM(Price)
FROM Inventory
WHERE Price > 2.5
```

Case Study: Multimodal Email Search

MAIDAP has been working with MSAI to explore multimodal search capabilities for outlook.

An example of relevant data analysis:

What is the count of the different types of image attachments in outlook emails?

Regular Images

Receipts

Regards,

Microsoft University Recruiting

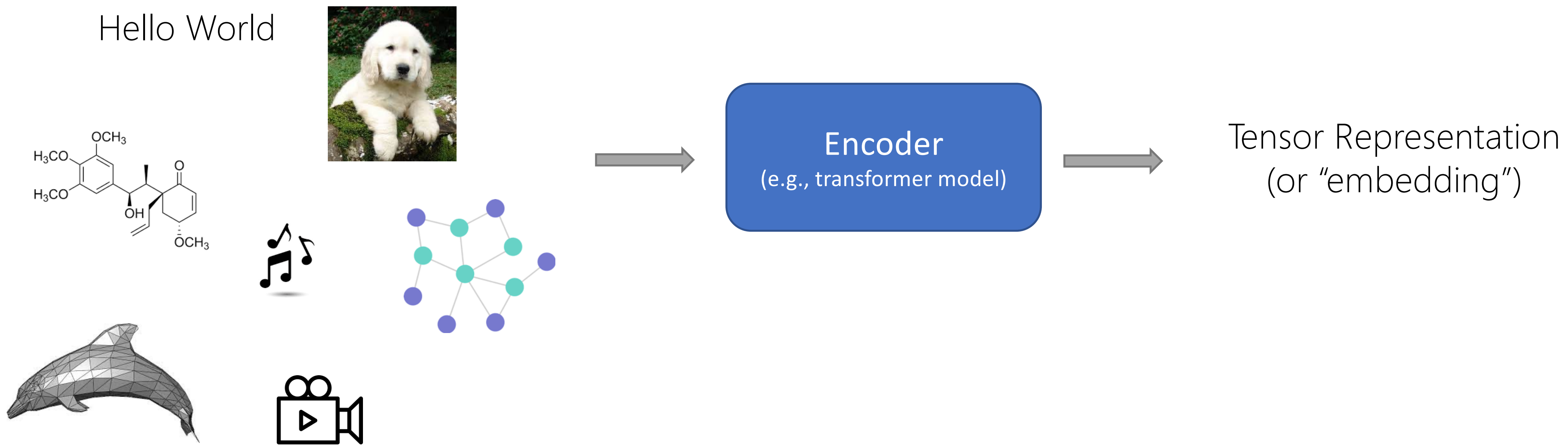
Microsoft

Company Logos

Surakav's multimodal support makes it easy to answer such queries.

Tensors are the de facto data structure for multimodal computation

The tensor data structure has been used to represent numerous rich entities.



Surakav can exploit tensors for multimodal query support.