PRETZEL:Opening the Black Box of Machine Learning Prediction Serving Systems

Presented by Qinyuan Sun

Slides are modified from first author Yunseong Lee’s slides

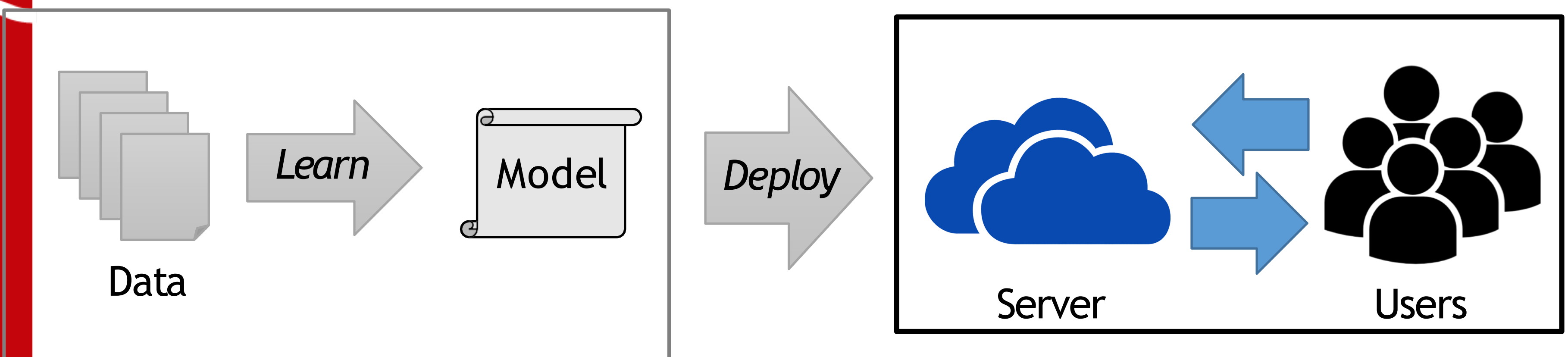
Outline

- **Prediction Serving Systems**
- Limitations of Black Box Approaches
- PRETZEL: White-box Prediction Serving System
- Evaluation
- Conclusion

Machine Learning Prediction Serving

1. Models are learned from data
2. Models are deployed and served together

Performance goal:
1) Low latency
2) High throughput
3) Minimal resource usage

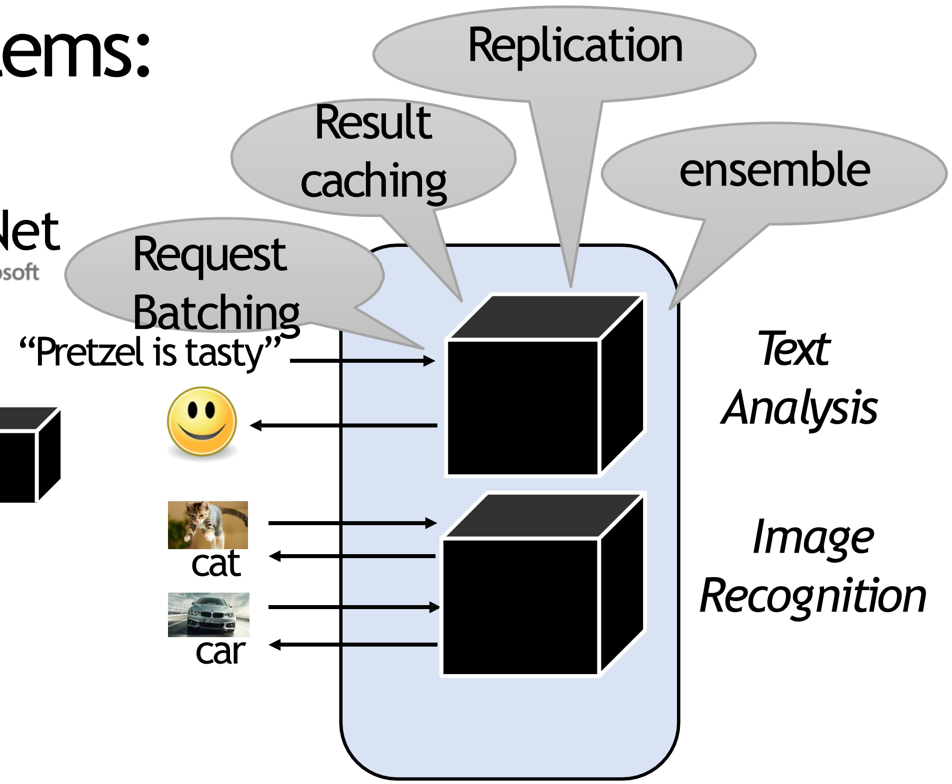


Training

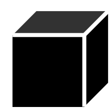
Prediction serving

ML Prediction Serving Systems: State-of-the-art

Clipper riselab · TF Serving Google · ML.Net Microsoft

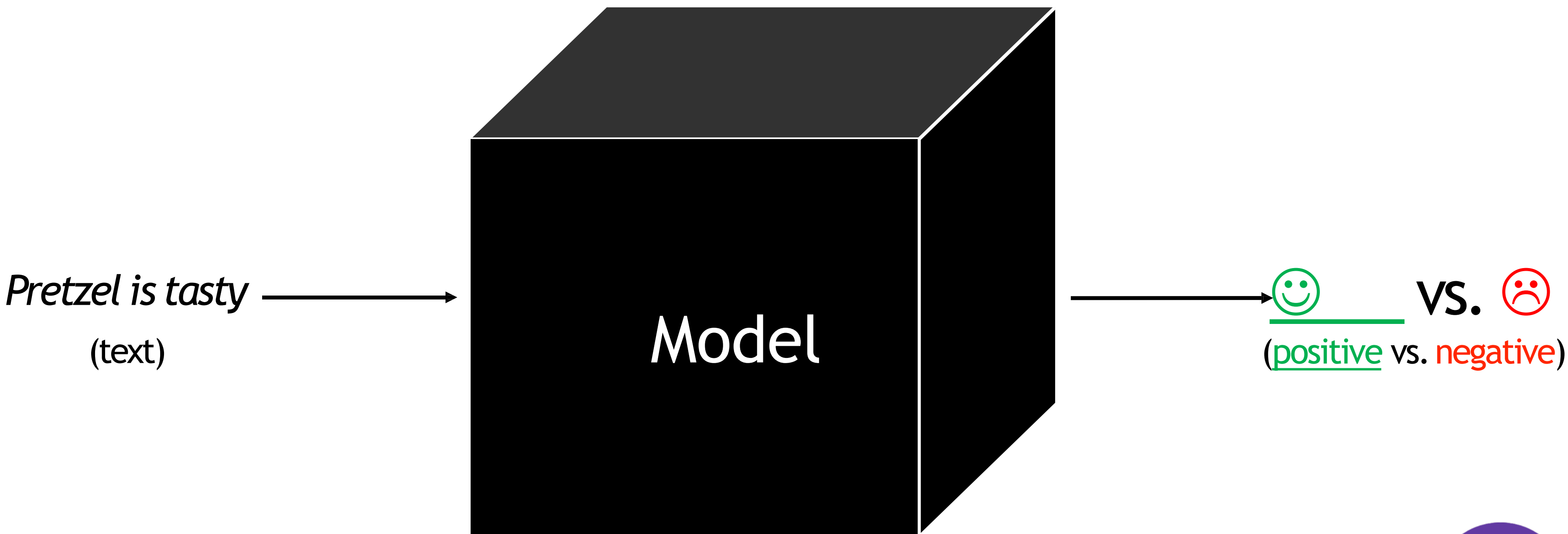


Prediction Serving System

- <u>Assumption: models are black box</u> 
  - Re-use the same code in training phase
  - Encapsulate all operations into a function call (e.g., `predict()`)
  - Apply *external* optimizations

5

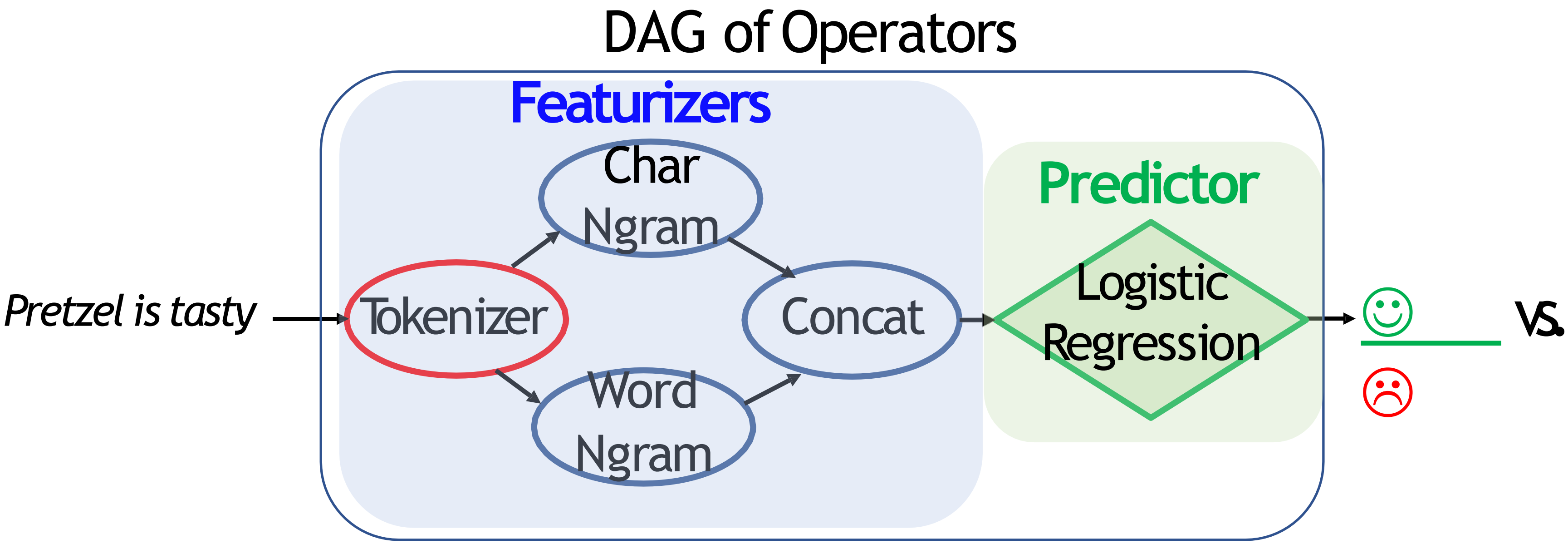
How does Models Look Like inside Boxes?



<Example: Sentiment Analysis>

ML.NET
Microsoft

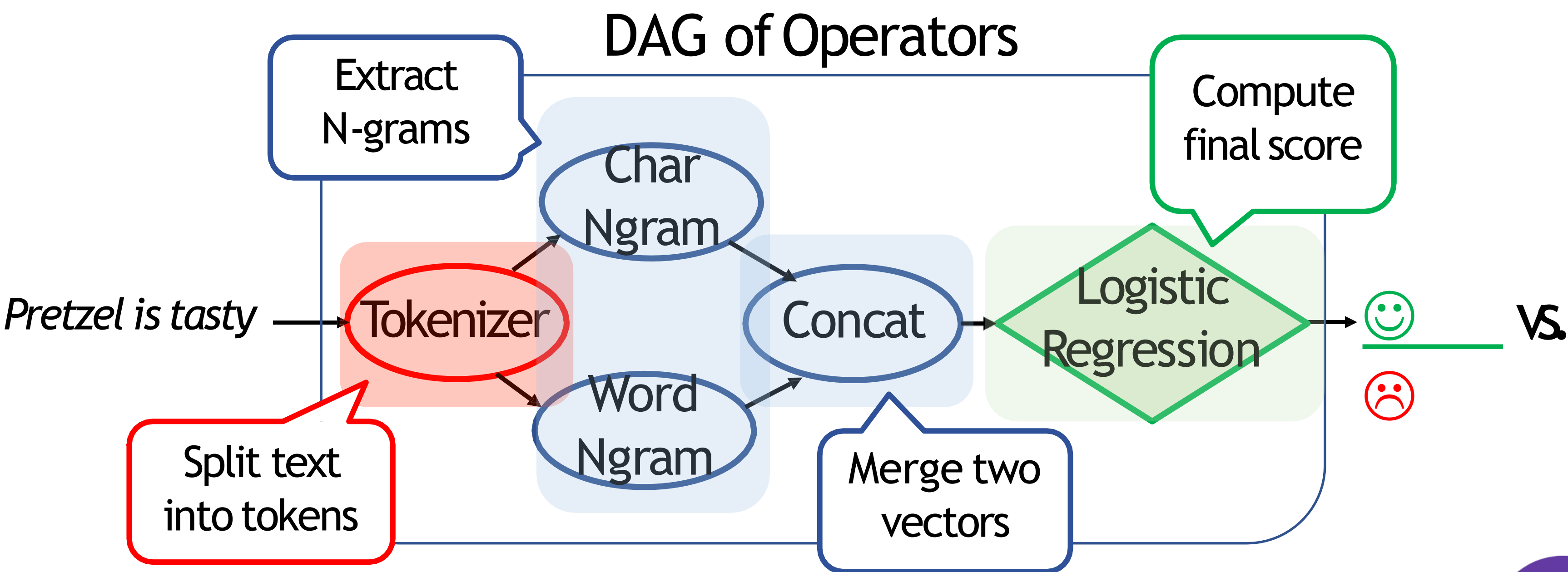
How do Models Look inside Boxes?



<Example: Sentiment Analysis>

ML.NET
Microsoft

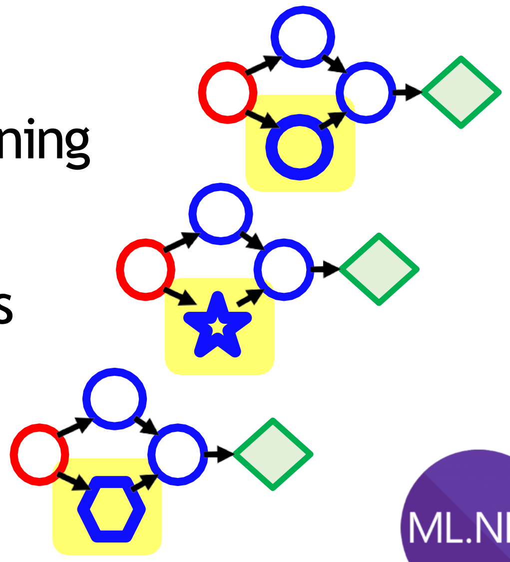
How do Models Look inside Boxes?



<Example: Sentiment Analysis>

Many Models Have Similar Structures

- Many part of a model can be re-used in other models
- Customer personalization, Templates, Transfer Learning
- Identical set of operators with different parameters



Outline

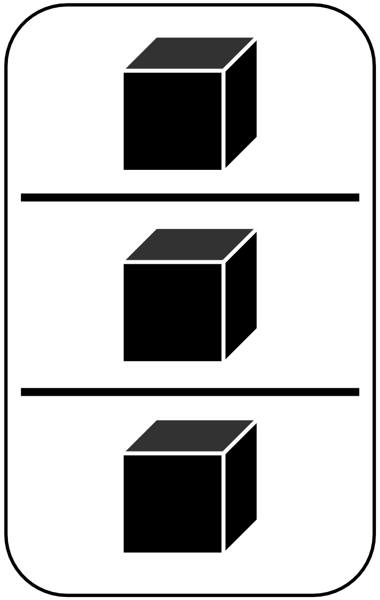
- Prediction Serving Systems
- **Limitations of Black Box Approaches**
- PRETZEL: White-box Prediction Serving System
- Evaluation
- Conclusion

Limitation 1: Resource Waste

- Resources are isolated across Blackboxes

1. Unable to share memory space
   ➔ Waste memory to maintain duplicate objects (despite similarities between models)

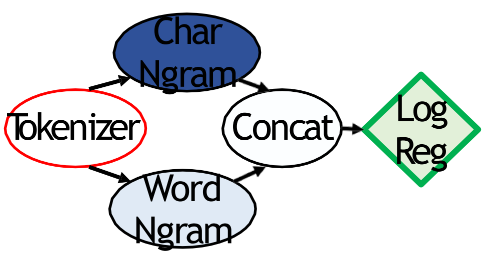
2. No coordination for CPU resources between boxes
   ➔ Serving many models can use too many threads

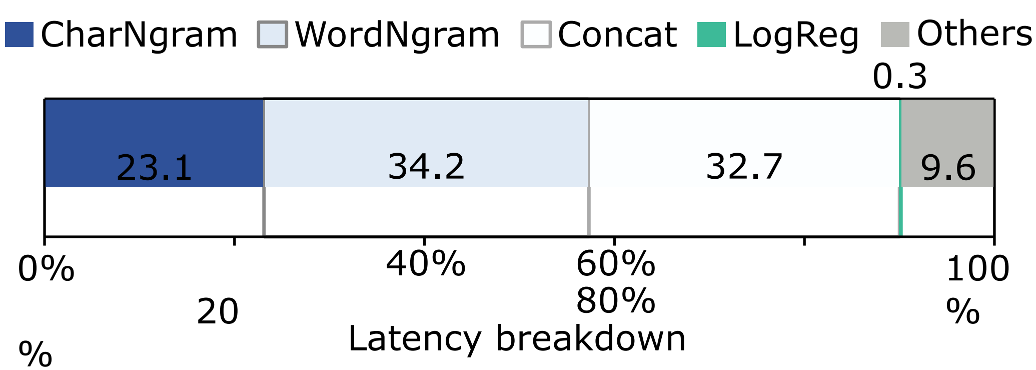


machine

Limitation 2: Inconsideration for Ops' Characteristics

1. Operators have different performance characteristics
   - Concat materializes a vector
   - LogReg takes only 0.3% (contrary to the training phase)
2. There can be a better plan if such characteristics are considered
   - Re-use the existing vectors
   - Apply in-place update in LogReg



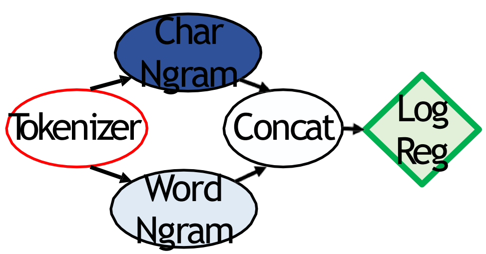


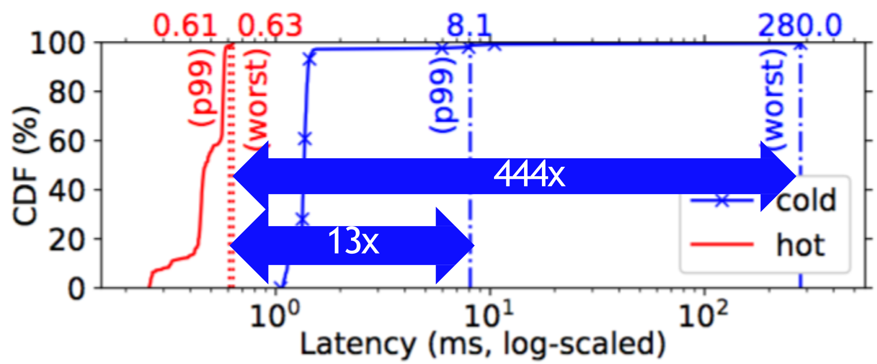
ML.NET
Microsoft

1
2

Limitation 3: Lazy Initialization

- ML.Net initializes code and memory lazily (efficient in training phase)
- Run 250 Sentiment Analysis models 100 times
  ➔ cold: first execution / hot: average of the rest 99
- Long-tail latency in the cold case
  - Code analysis, Just-in-time (JIT) compilation, memory allocation, etc
  - Difficult to provide strong Service-Level-Agreement (SLA)





ML.NET
Microsoft

Outline

- (Black-box) Prediction Serving Systems
- Limitations of Black Box Approaches
- **PRETZEL: White-box Prediction Serving System**
- Evaluation
- Conclusion

PRETZEL: White-box Prediction Serving

- We analyze models to optimize the internal execution
- We let models co-exist on the same runtime, sharing computation and memory resources
- We optimize models in two directions:
    1. End-to-end optimizations
    2. Multi-model optimizations

End-to-End Optimizations

Optimize the execution of **individual** models from start to end

1. [Ahead-of-time Compilation]
   Compile operators' code in advance
   → No JIT overhead
2. [Vector pooling]
   Pre-allocate data structures
   → No memory allocation on the data path

Multi-model Optimizations

Share computation and memory **across models**

1. [Object Store]

   Share Operators parameters/weights

   → Maintain only one copy

2. [Sub-plan Materialization]

   Reuse intermediate results computed by other models

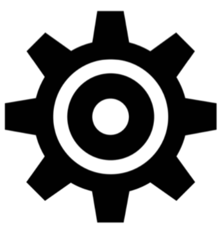
   → Save computation

System Components

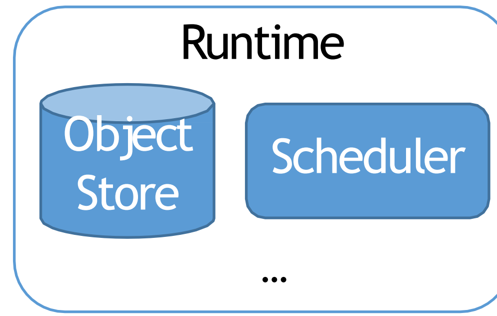
1. Flour: Intermediate Representation

```
var fContext = ...;
var Tokenizer = ...;
return fPrgm.Plan();
```

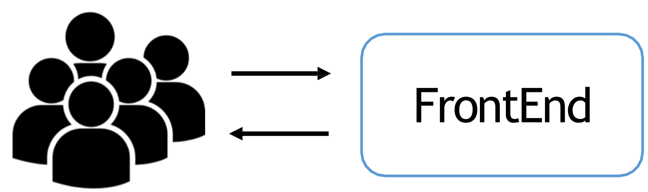
2. Oven: Compiler/Optimizer



3. Runtime: Execute inference queries

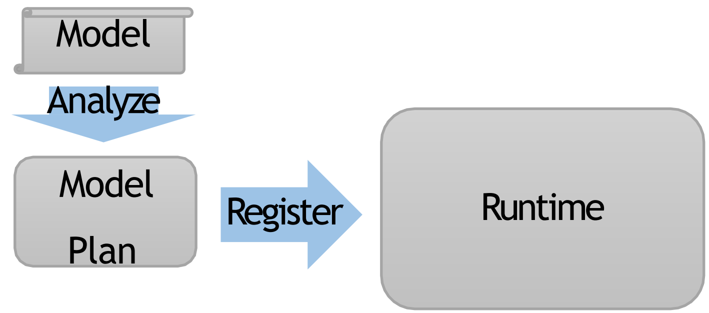


4. FrontEnd: Handle user requests

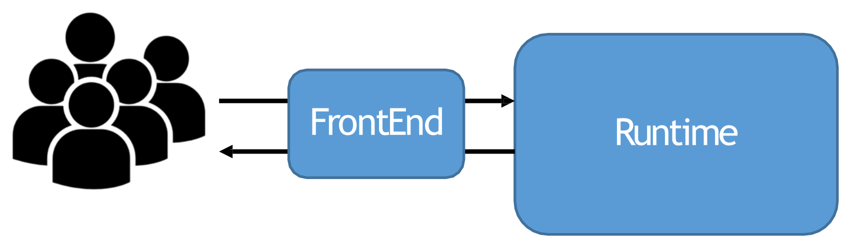


Prediction Serving with PRETZEL

1. Offline
   - Analyze structural information of models
   - Build *ModelPlan* for optimal execution
   - Register *ModelPlan* to Runtime



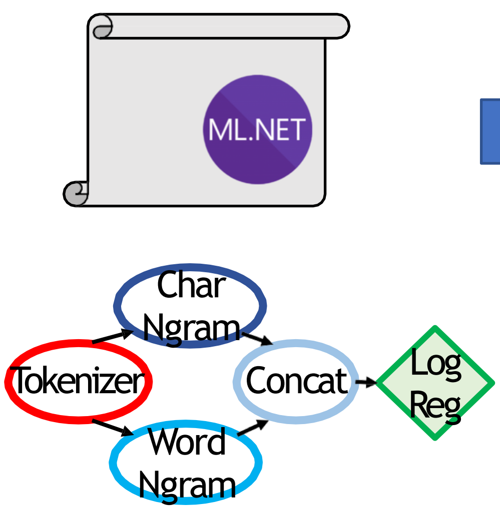
2. Online
   - Handle prediction requests
   - Coordinate CPU & memory resources



System Design: Offline Phase

1. Translate Model into Flour Program

<Model>



<Flour Program>

```
var fContext = new FlourContext(...)
var tTokenizer = fContext.CSV
       .FromText(fields, fieldsType, sep)
       .Tokenize();

var tCNgram = tTokenizer.CharNgram(numCNgrms, ...);
var tWNgram = tTokenizer.WordNgram(numWNgrms, ...);
var fPrgrm = tCNgram
              .Concat(tWNgram)
              .ClassifierBinaryLinear(cParams);

return fPrgrm.Plan();
```

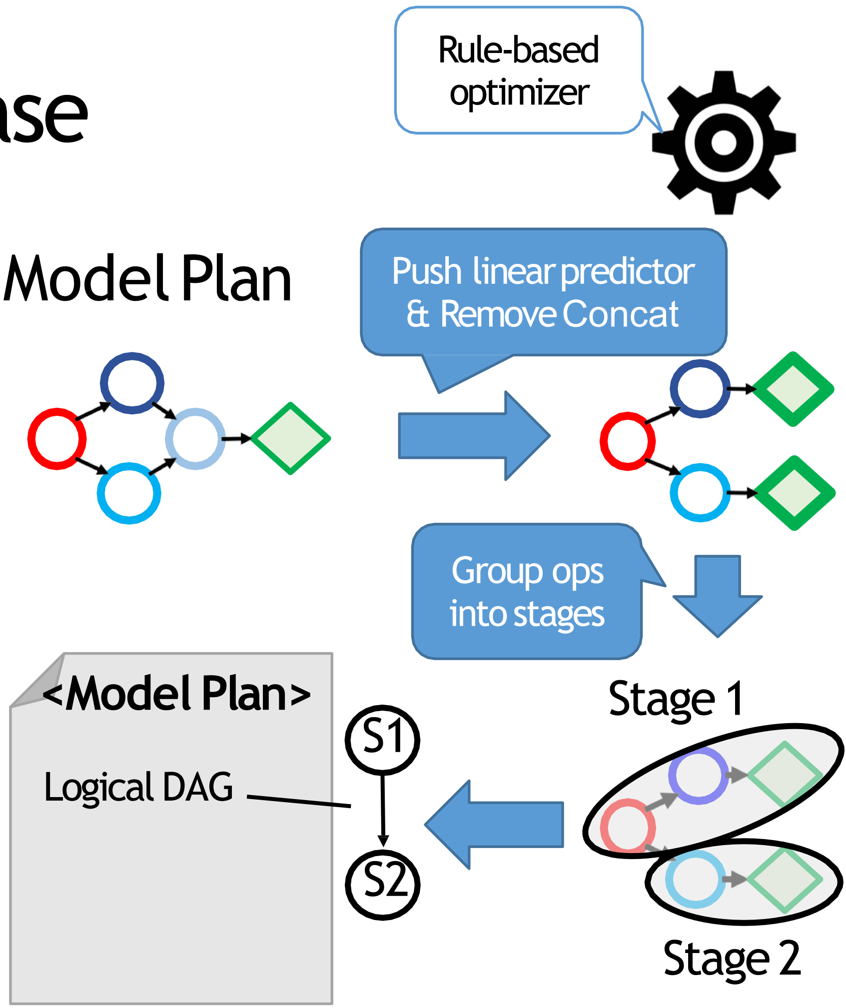
System Design: Offline Phase

2. Oven optimizer/compiler build Model Plan

<Flour Program>

```
var fContext = new FlourContext(...)
var tTokenizer = fContext.CSV
       .FromText(fields, fieldsType, sep)
       .Tokenize();

var   tCNgram   =   tTokenizer.CharNgram(numCNgrms,
...);                      var         tWNgram         =
tTokenizer.WordNgram(numWNgrms,  ...);   var  fPrgrm
= tCNgram
               .Concat(tWNgram)
               .ClassifierBinaryLinear(cParams);
return fPrgrm.Plan();
```



System Design: Offline Phase

2. Oven optimizer/compiler build Model Plan

Rule-based optimizer

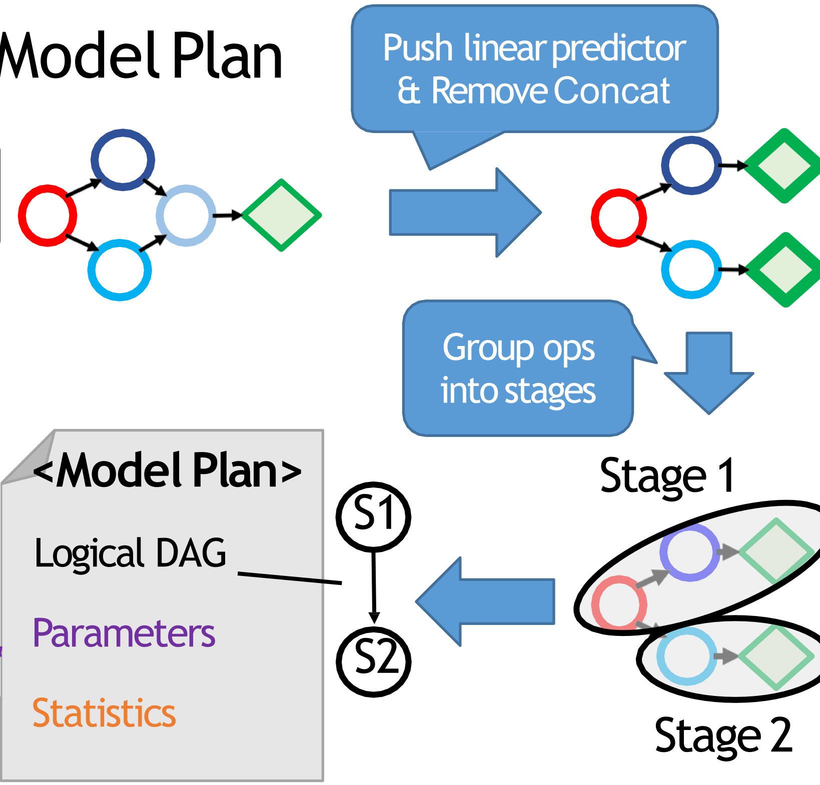
<Flour Program>

```
var fContext = new FlourContext(...)
var tTokenizer = fContext.CSV
        .FromText(fields, fieldsType, sep)
        .Tokenize();

var   tCNgram   =   tTokenizer.CharNgram(numCNgrms,
...);                   var         tWNgram        =
tTokenizer.WordNgram(numWNgrms,  ...);   var  fPrgrm
= tCNgram
              .Concat(tWNgram)
              .ClassifierBinaryLinear(cParams);
return fPrgrm.Plan();
```

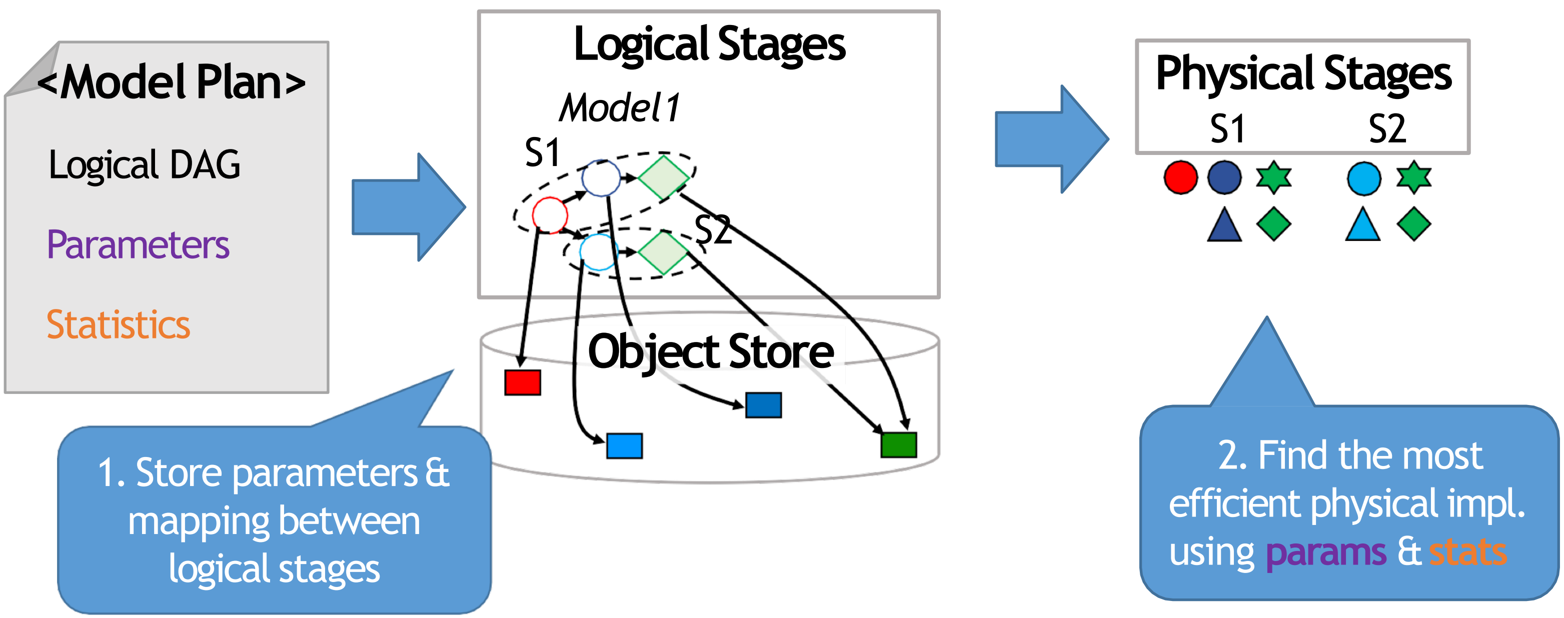
e.g., Dictionary, N-gram Length

e.g., dense vs. sparse, maximum vector size



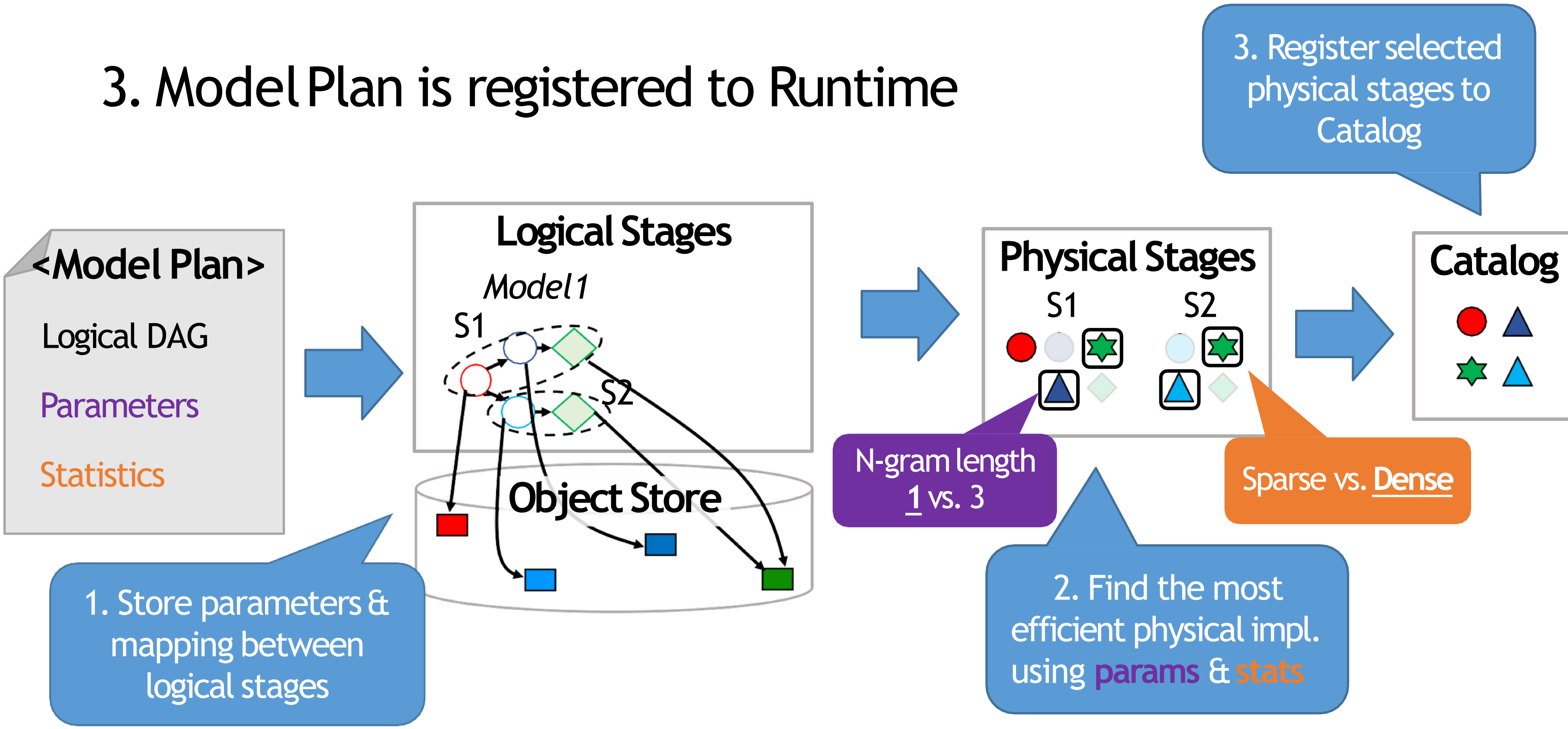
System Design: Offline Phase

3. Model Plan is registered to Runtime

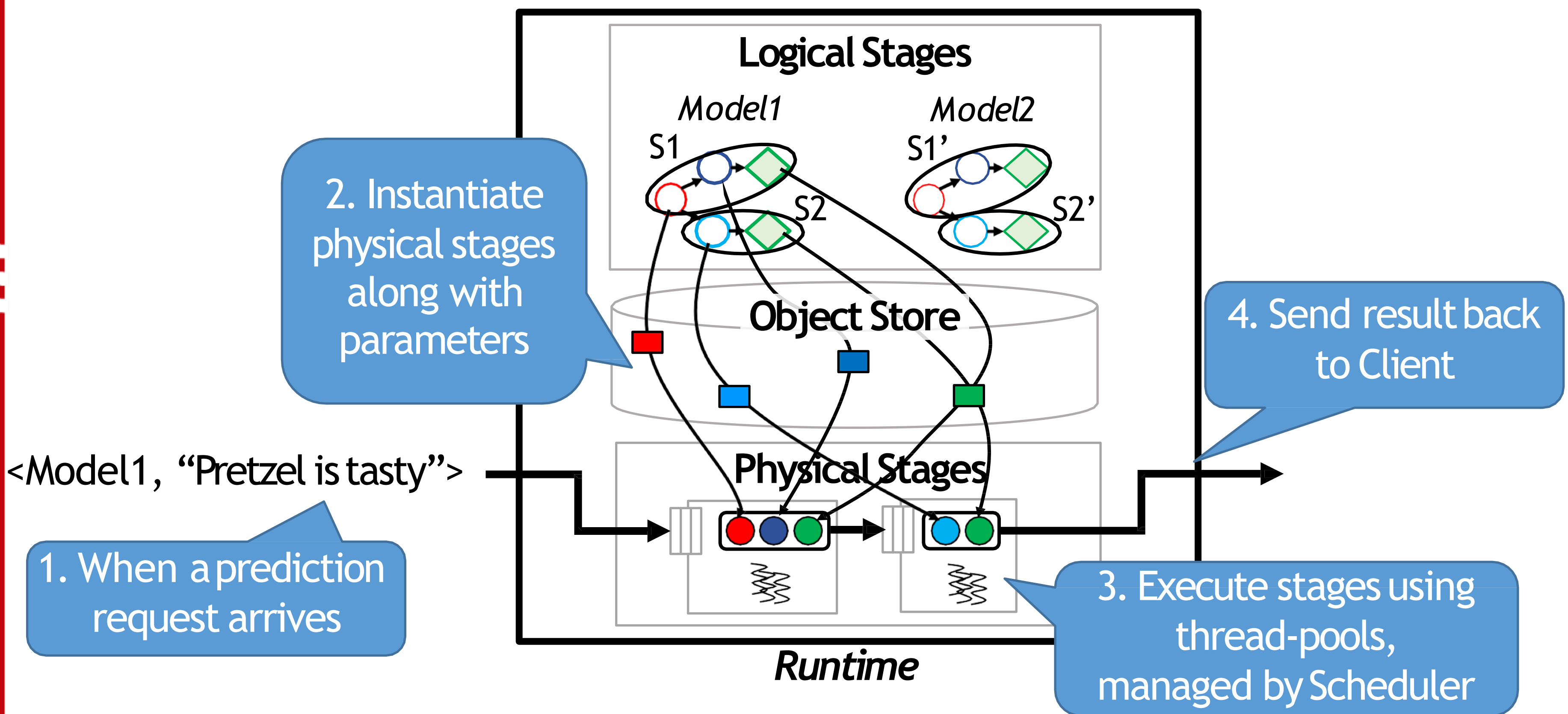


System Design: Offline Phase

3. Model Plan is registered to Runtime



System Design: Online Phase



25

Outline

- (Black-box) Prediction Serving Systems
- Limitations of Black box approaches
- PRETZEL: White-box Prediction Serving System
- **Evaluation**
- Conclusion

Evaluation

- Q. How PRETZEL improves performance over black-box approaches?
  - in terms of *latency*, *memory* and *throughput*

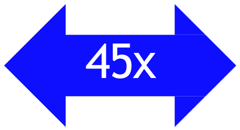
- 500 Models from Microsoft Machine Learning Team
  - 250 Sentiment Analysis (Memory-bound)
  - 250 Attendee Count (Compute-bound)

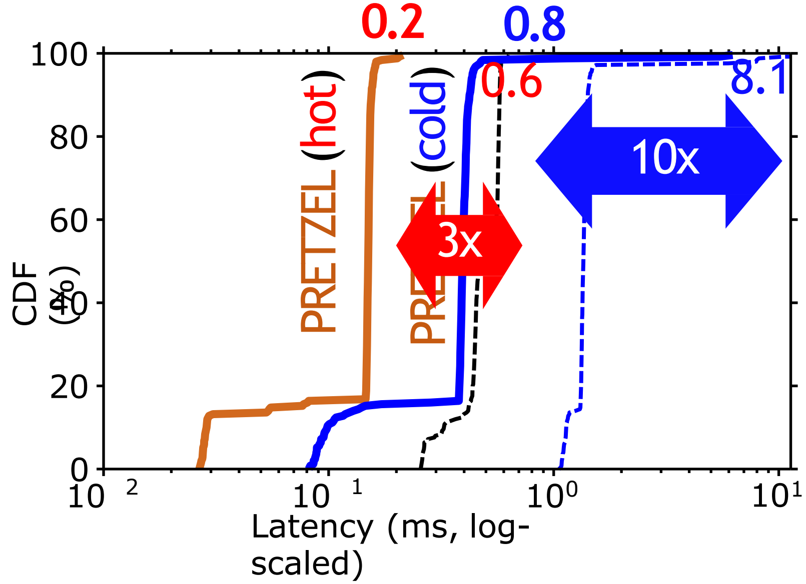
- System configuration
  - 16 Cores CPU, 32GB RAM
  - Windows 10, .Net core 2.0

Evaluation: Latency

- Micro-benchmark (No server-client communication)
  - Score 250 Sentiment Analysis models 100 times for each
  - Compare ML.Net vs. PRETZEL

	ML.Net	PRETZEL
P99 (hot)	0.6	0.2
P99 (cold)	8.1	0.8
Worst (cold)	**280.2**	**6.2**

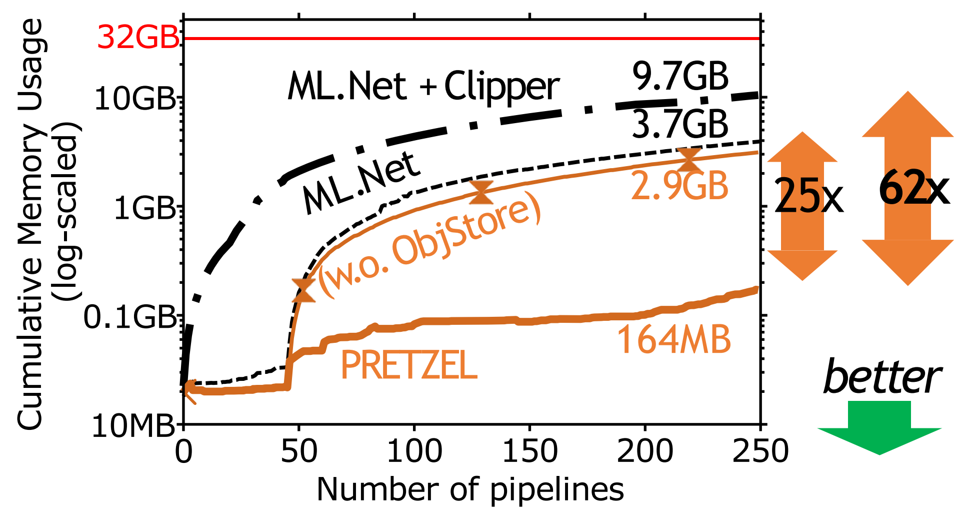




Evaluation: Memory

- Measure Cumulative Memory Usage after loading 250 models
  - Attendee Count models (smaller size than Sentiment Analysis)
  - 4 settings for Comparison

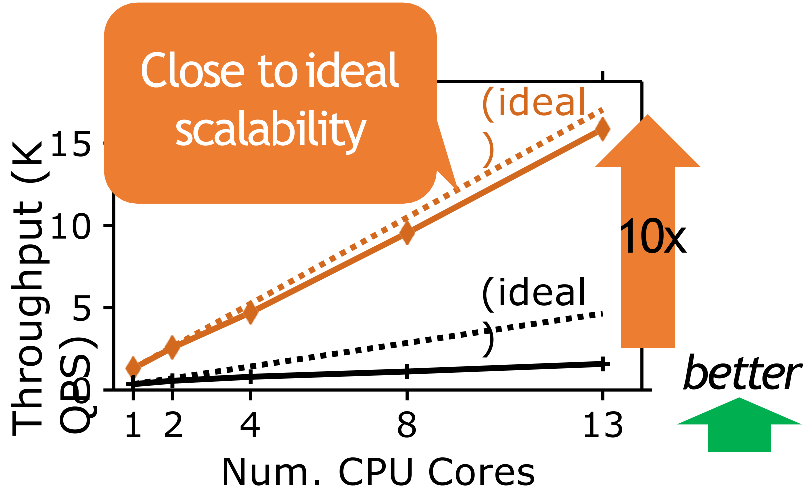
Settings	Shared Objects	Shared Runtime
ML.Net + Clipper		
ML.Net		✓
PRETZEL without ObjectStore		✓
PRETZEL	✓	✓



Evaluation: Throughput

- Micro-benchmark
  - Score 250 Attendee Count models 1000 times for each
  - Request 1000 queries in a batch
  - Compare ML.Net vs. PRETZEL

More results in the paper!



Conclusion

- PRETZEL is the first white-box prediction serving system for ML pipelines

- By using models' structural info, we enable two types of optimizations:
  - End-to-end optimizations generate efficient execution plans for a model
  - Multi-model optimizations let models share computation and memory resources

- Our evaluation shows that PRETZEL can improve performance compared to Black-box systems (e.g., ML.Net)
  - Decrease latency and memory footprint
  - Increase resource utilization and throughput

- A lot of external optimizations used by Cipper are orthogonal to PRETZEL

Thank you!
Questions?