Willump: A Statistically-Aware End-to-end Optimizer for ML Inference

Peter Kraft, Daniel Kang, Deepak Narayanan,
Shoumik Palkar, Peter Bailis, Matei Zaharia

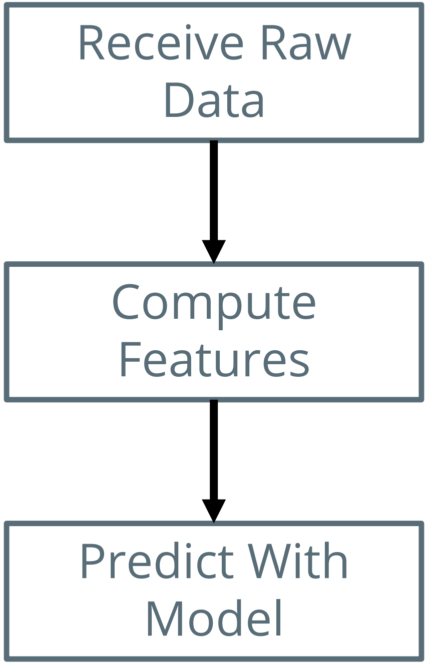
STANFORD DAWN

Problem: ML Inference

- Often performance-critical.
- Recent focus on tools for ML prediction serving.

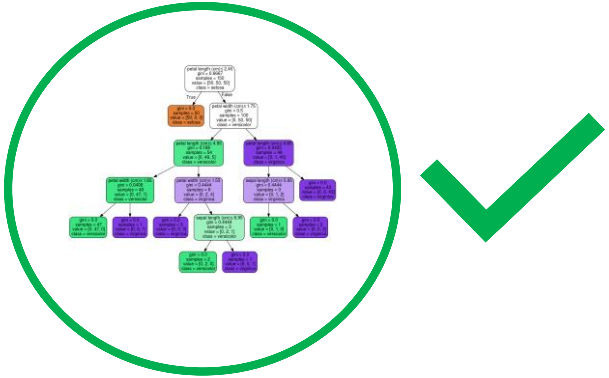
A Common Bottleneck: Feature Computation

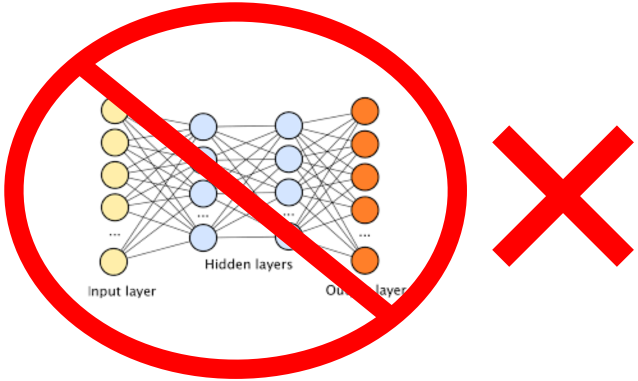
- Many applications bottlenecked by feature computation.
- Pipeline of transformations computes numerical *features* from data for model.



A Common Bottleneck: Feature Computation

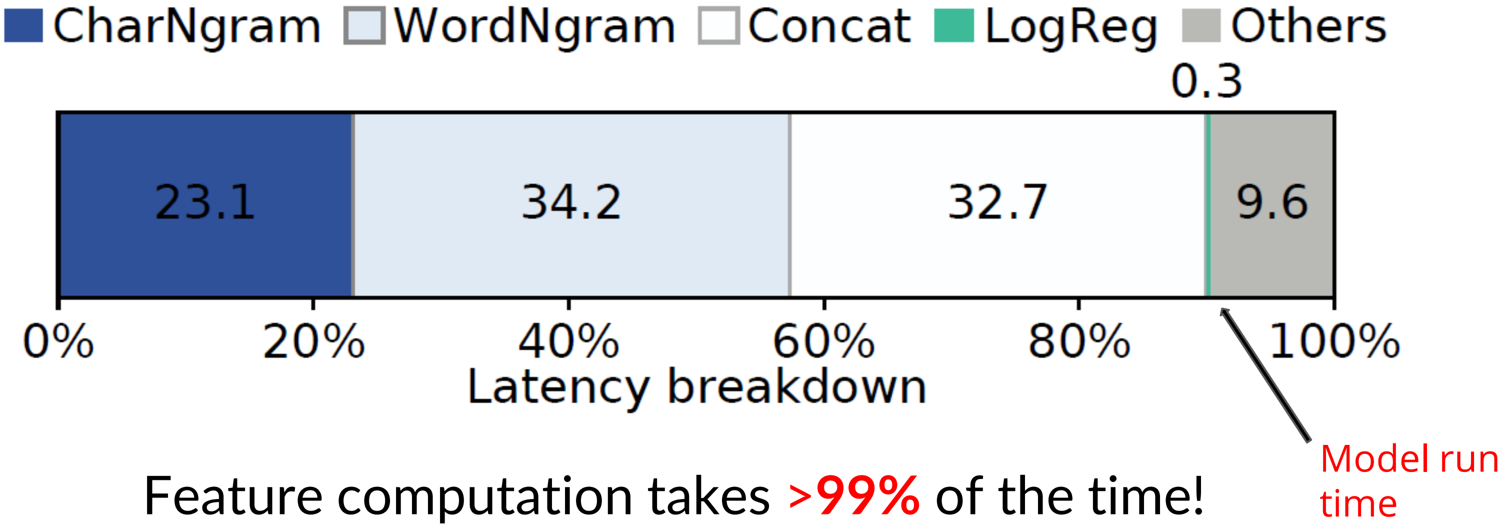
- Feature computation is bottleneck when models are inexpensive—boosted trees, not DNNs.
- Common on tabular/structured data!





A Common Bottleneck: Feature Computation

Production Microsoft sentiment analysis pipeline



Feature computation takes **>99%** of the time!

Source: Pretzel (OSDI '18)

Current State-of-the-art

- Apply traditional serving optimizations, e.g. caching (Clipper), compiler optimizations (Pretzel).
- Neglect unique **statistical properties** of ML apps.

Statistical Properties of ML

Amenability to approximation

Statistical Properties of ML

Amenability to approximation

Easy input:
Definitely not a dog.

Hard input:
Maybe a dog?

Statistical Properties of ML

Amenability to approximation

Easy input:
Definitely not
a dog.

Hard input:
Maybe a
dog?

Existing Systems: Use Expensive Model for Both

Statistical Properties of ML

Amenability to approximation

Easy input: Definitely not a dog.

Hard input: Maybe a dog?

Statistically-Aware Systems: Use cheap model on bucket, expensive model on cat.

Statistical Properties of ML

- Model is often part of a bigger app (e.g. top-K query)

Statistical Properties of ML

- Model is often part of a bigger app (e.g. top-K query)

Artist	Score	Rank
Beatles	9.7	1
Bruce Springsteen	9.5	2
…	…	…
Justin Bieber	5.6	999
Nickelback	4.1	1000

Problem: Return top 10 artists.

Statistical Properties of ML

- Model is often part of a bigger app (e.g. top-K query)

Existing Systems

Artist	Score	Rank
Beatles	9.7	1
Bruce Springsteen	9.5	2
…	…	…
Justin Bieber	5.6	999
Nickelback	4.1	1000

Use expensive model for everything!

Statistical Properties of ML

- Model is often part of a bigger app (e.g. top-K query)

Statistically-aware Systems

Artist	Score	Rank
Beatles	9.7	1
Bruce Springsteen	9.5	2
…	…	…
Justin Bieber	5.6	999
Nickelback	4.1	1000

High-value:
Rank precisely,
return.

Low-value:
Approximate,
discard.

Prior Work: Statistically-Aware Optimizations

- Statistically-aware optimizations exist in literature.
- Always application-specific and custom-built.
- Never automatic!

Source:
Cheng et al.
(DLRS’ 16),
Kang et al.
(VLDB ‘17)

ML Inference Dilemna

- ML inference systems:
    - Easy to use.
    - Slow.
- Statistically-aware systems:
    - Fast
    - Require a lot of work to implement.

Can an ML inference system be fast and easy to use?

Willump: Overview

- Statistically-aware optimizer for ML Inference.
- Targets feature computation!
- ***Automatic*** model-agnostic statistically-aware opts.
- 10x throughput+latency improvements.

Outline

- **System Overview**
- Optimization 1: End-to-end Cascades
- Optimization 2: Top-K Query Approximation
- Evaluation

Willump: Goals

- Automatically maximize performance of ML inference applications whose performance bottleneck is feature computation

System Overview

Input Pipeline

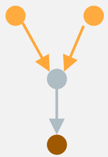
```
def pipeline(x1, x2):
  input = lib.transform(x1, x2)
  preds = model.predict(input)
  return preds
```

System Overview

Input Pipeline

```
def pipeline(x1, x2):
  input = lib.transform(x1, x2)
  preds = model.predict(input)
  return preds
```

Willump Optimization



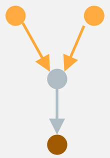
Infer Transformation Graph

System Overview

Input Pipeline

```
def pipeline(x1, x2):
  input = lib.transform(x1, x2)
  preds = model.predict(input)
  return preds
```

Willump Optimization



Infer Transformation Graph

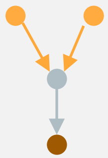
Statistically-Aware Optimizations:
1. End-To-End Cascades
2. Top-K Query Approximation

System Overview

Input Pipeline

```
def pipeline(x1, x2):
  input = lib.transform(x1, x2)
  preds = model.predict(input)
  return preds
```

Willump Optimization



Infer Transformation Graph

Statistically-Aware Optimizations:

1. End-To-End Cascades
2. Top-K Query Approximation

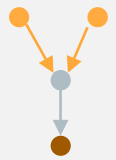
Compiler Optimizations (Weld—Palkar et al. VLDB ‘18)

System Overview

Input Pipeline

```
def pipeline(x1, x2):
  input = lib.transform(x1, x2)
  preds = model.predict(input)
  return preds
```

Willump Optimization



Infer Transformation Graph

Statistically-Aware Optimizations:
1. End-To-End Cascades
2. Top-K Query Approximation

Compiler Optimizations (Weld—Palkar et al. VLDB ‘18)

Optimized Pipeline

```
def willump_pipeline(x1, x2):
  preds = compiled_code(x1, x2)
  return preds
```

Outline

- System Overview
- **Optimization 1: End-to-end Cascades**
- Optimization 2: Top-K Query Approximation
- Evaluation

Background: Model Cascades

- Classify "easy" inputs with cheap model.
- *Cascade* to expensive model for "hard" inputs.

Easy input:
Definitely not
a dog.

Hard input:
Maybe a
dog?

Background: Model Cascades

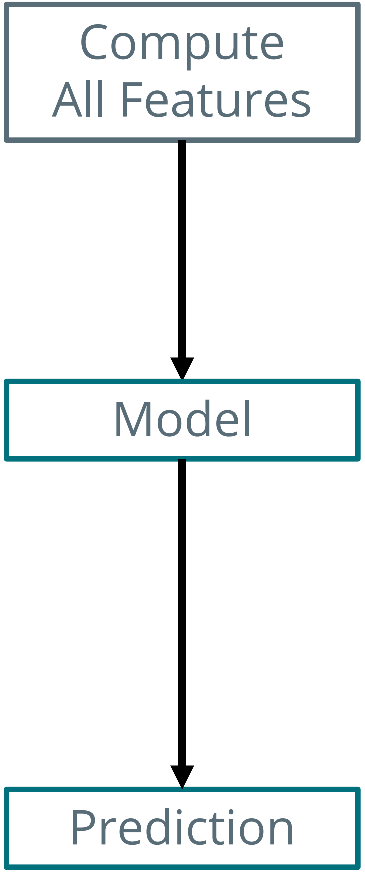
- Used for image classification, object detection.
- Existing systems application-specific and custom-built.

Source:
Viola-Jones
(CVPR’ 01),
Kang et al.
(VLDB ‘17)

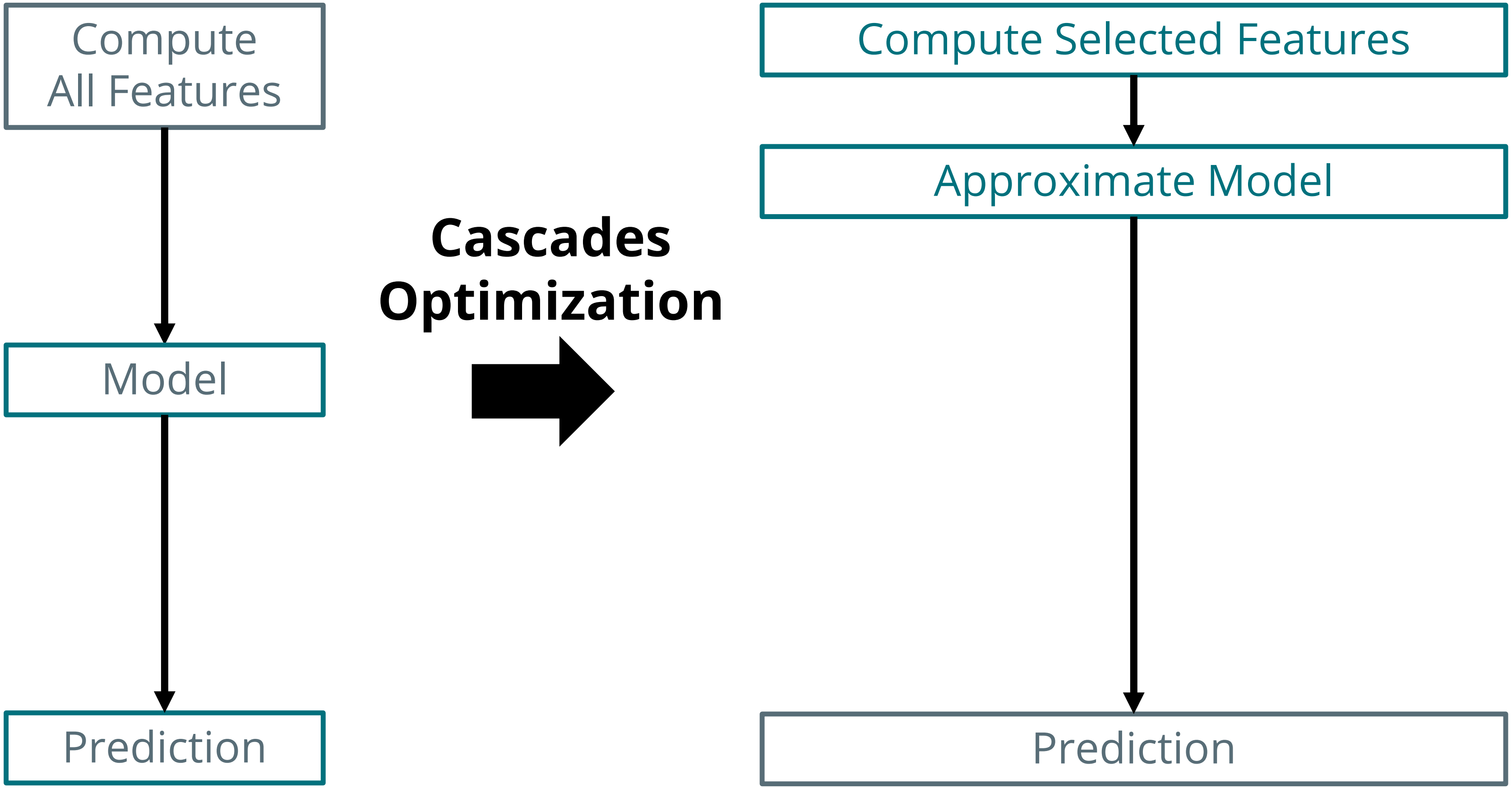
Our Optimization: End-to-end cascades

- Compute only some features for “easy” data inputs; cascade to computing all for “hard” inputs.
- Automatic and model-agnostic, unlike prior work.
    - Estimates for runtime performance & accuracy of a feature set
    - Efficient search process for tuning parameters

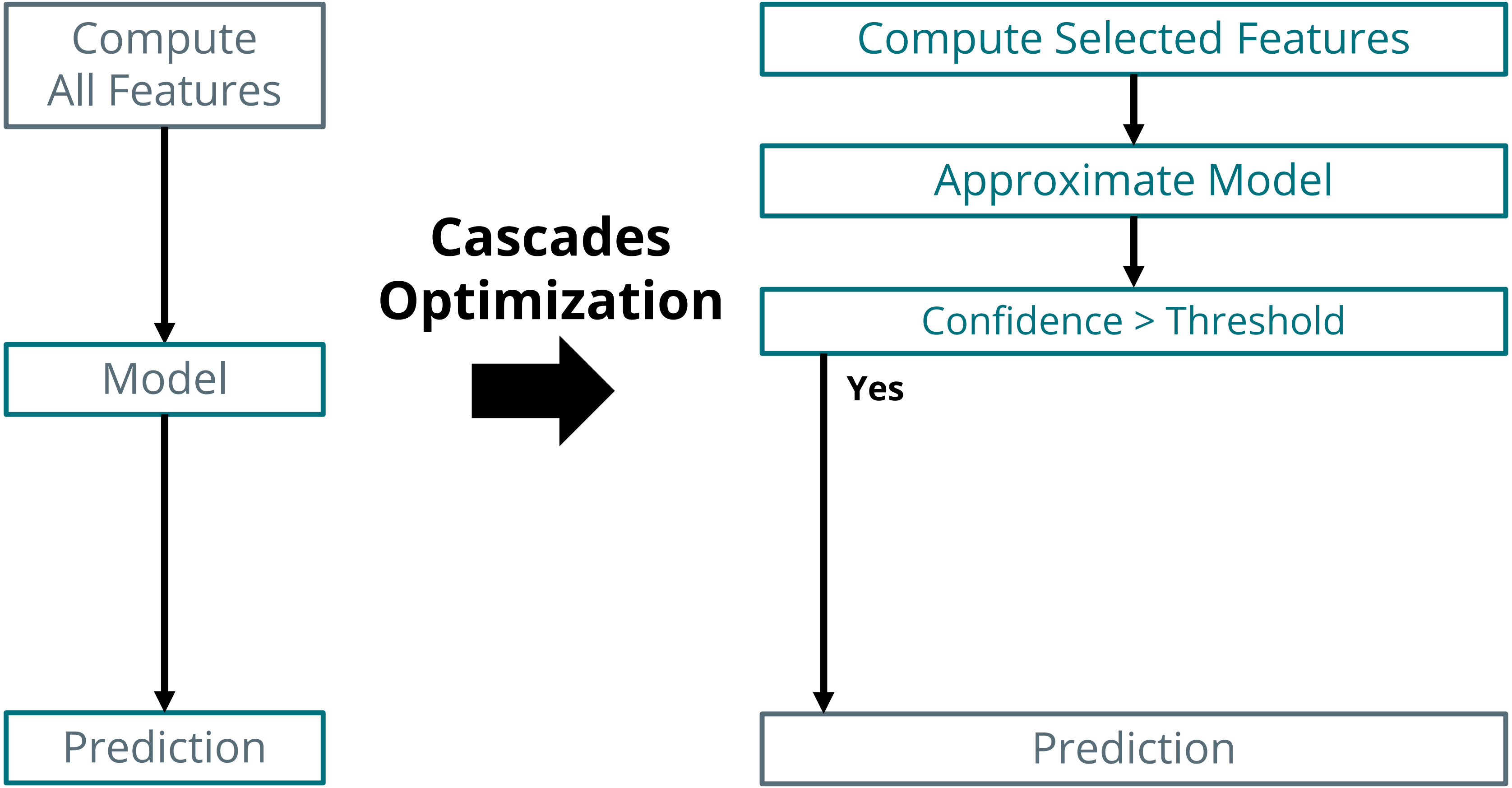
End-to-end Cascades: Original Model



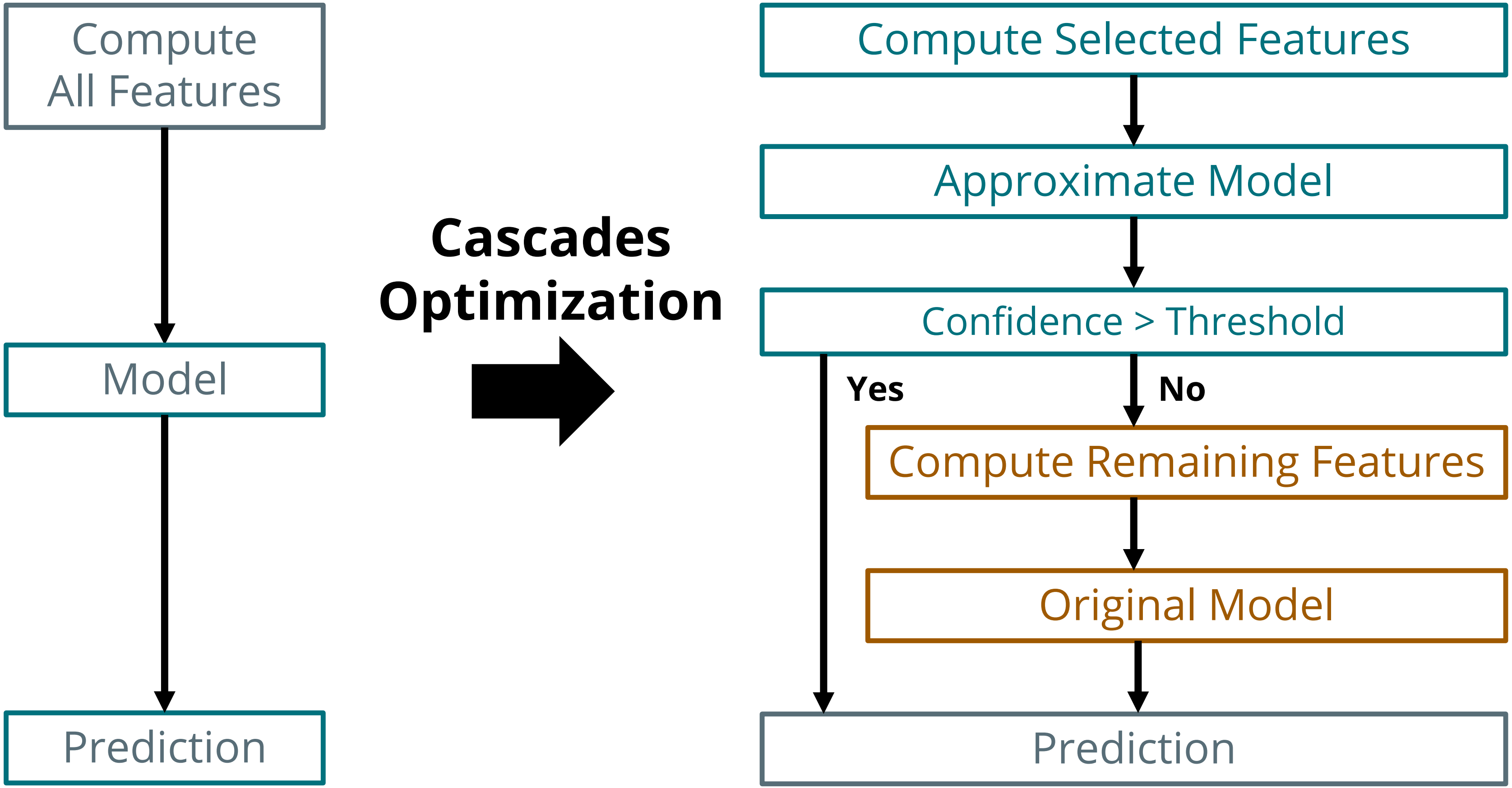
End-to-end Cascades: Approximate Model



End-to-end Cascades: Confidence



End-to-end Cascades: Final Pipeline

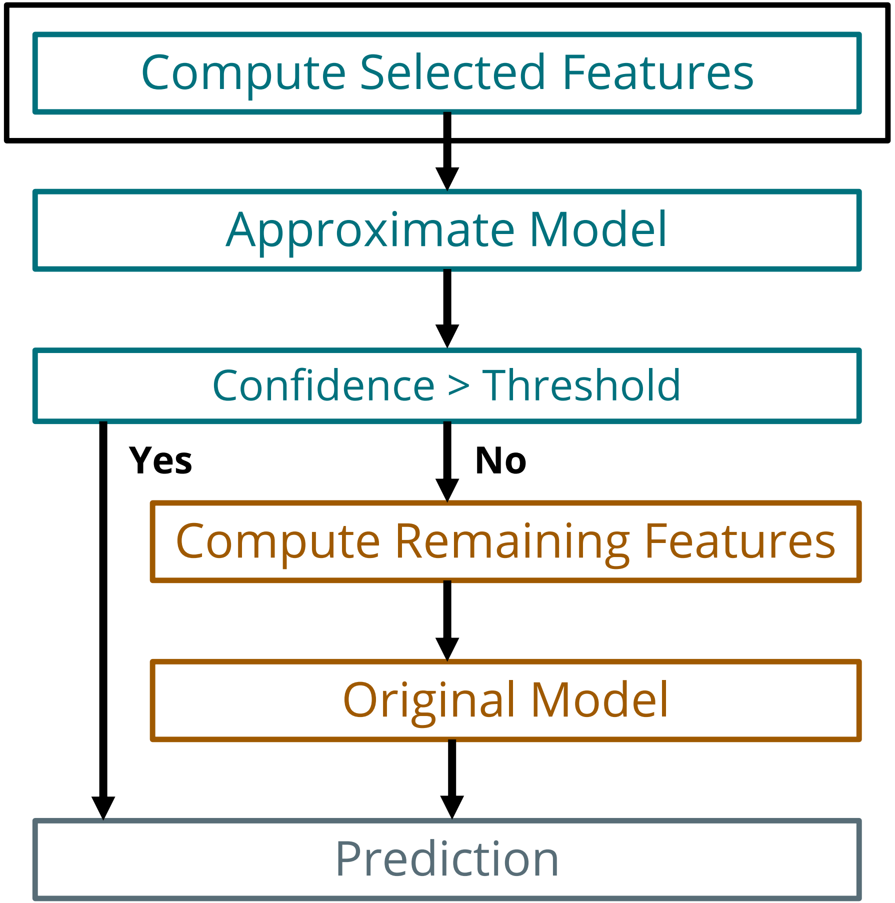


End-to-end Cascades: Constructing Cascades

- Construct cascades during model training.
- Need model training set and an accuracy target.

End-to-end Cascades: Selecting Features

Key question: Select which features?



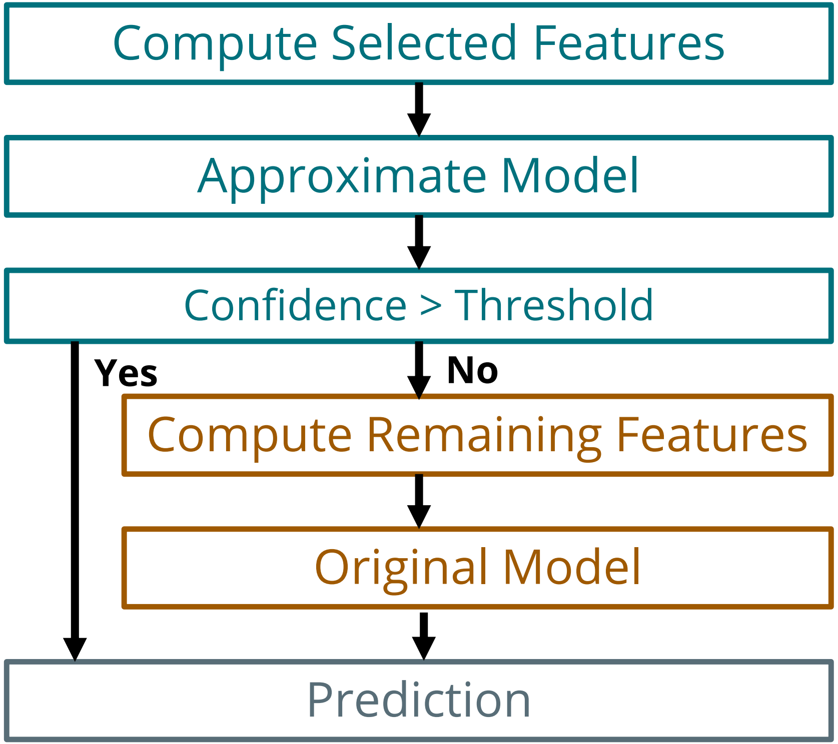
End-to-end Cascades: Selecting Features

- Goal: Select features that minimize expected query time given accuracy target.

End-to-end Cascades: Selecting Features

Two possibilities for a query: Can approximate or not.

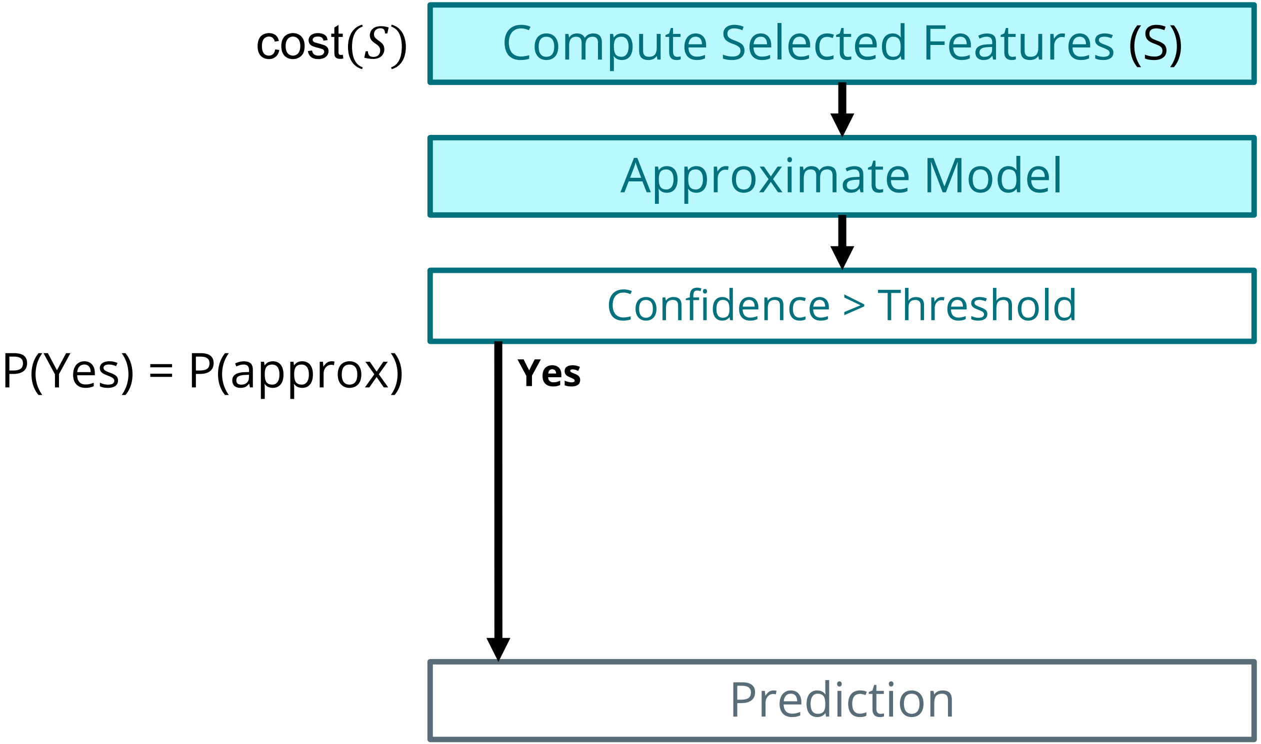
Can approximate query.



Can't approximate query.

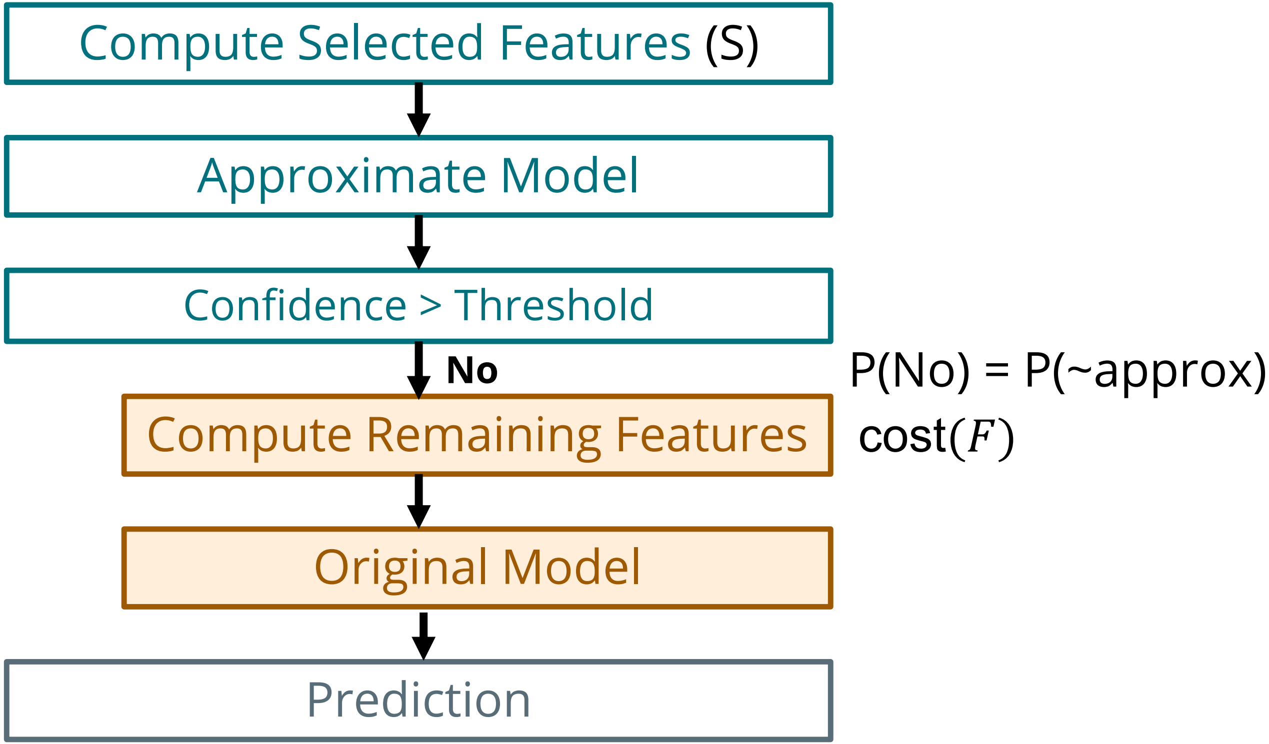
End-to-end Cascades: Selecting Features

$$\min_{S} P(\text{approx})\text{cost}(S) + P(\sim\text{approx})\text{cost}(F)$$



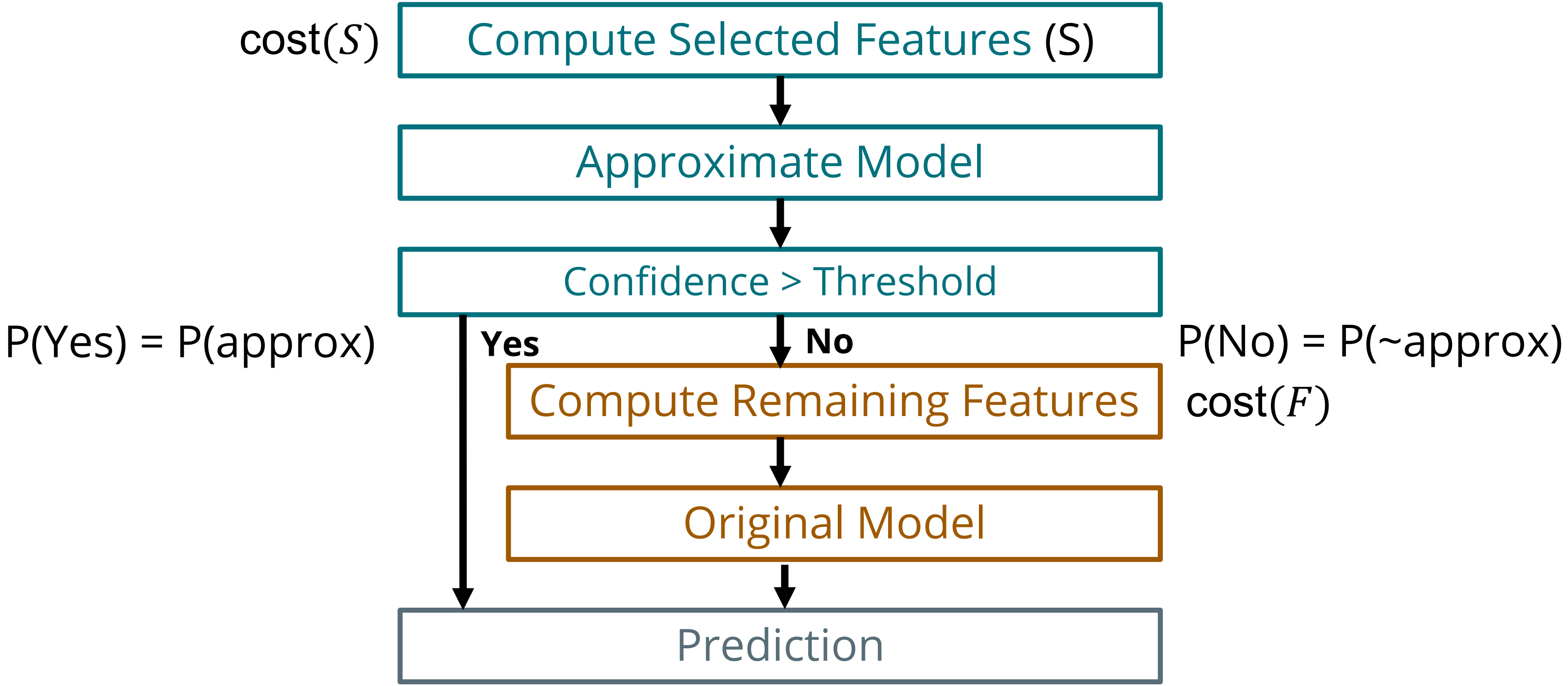
End-to-end Cascades: Selecting Features

$$\min_{S} P(\text{approx})\text{cost}(S) + P(\sim\text{approx})\text{cost}(F)$$



End-to-end Cascades: Selecting Features

$$\min_{S} P(\text{approx})\text{cost}(S) + P(\sim\text{approx})\text{cost}(F)$$



End-to-end Cascades: Selecting Features

- Goal: Select feature set S that minimizes query time:

$$\min_{S} P(\text{approx})\text{cost}(S) + P(\sim\text{approx})\text{cost}(F)$$

End-to-end Cascades: Selecting Features

- Goal: Select feature set S that minimizes query time:

$$\min_{S} P(\text{approx})\text{cost}(S) + P(\sim\text{approx})\text{cost}(F)$$

- Approach:
  - **Choose several potential values of cost(S).**
  - Find best feature set with each cost(S).
  - Train model & find cascade threshold for each set.
  - Pick best overall.

End-to-end Cascades: Selecting Features

- Goal: Select feature set S that minimizes query time:

$$\min_{S} P(\text{approx})\text{cost}(S) + P(\sim\text{approx})\text{cost}(F)$$

- Approach:
    - Choose several potential values of cost(S).
    - **Find best feature set with each cost(S).**
    - Train model & find cascade threshold for each set.
    - Pick best overall.

End-to-end Cascades: Selecting Features

- Goal: Select feature set S that minimizes query time:

$$\min_S P(\text{approx})\text{cost}(S) + P(\sim\text{approx})\text{cost}(F)$$

- Approach:
  - Choose several potential values of cost(S).
  - Find best feature set with each cost(S).
  - **Train model & find cascade threshold for each set.**
  - Pick best overall.

End-to-end Cascades: Selecting Features

- Goal: Select feature set S that minimizes query time:

$$\min_{S} P(\text{approx})\text{cost}(S) + P(\sim\text{approx})\text{cost}(F)$$

- Approach:
  - Choose several potential values of $\text{cost}(S)$.
  - Find best feature set with each cost(S).
  - Train model & find cascade threshold for each set.
  - **Pick best overall.**

End-to-end Cascades: Selecting Features

- Goal: Select feature set S that minimizes query time:

$$\min_{S} P(\text{approx})\text{cost}(S) + P(\sim\text{approx})\text{cost}(F)$$

- Approach:
  - **Choose several potential values of cost(S).**
  - Find best feature set with each cost(S).
  - Train model & find cascade threshold for each set.
  - Pick best overall.

End-to-end Cascades: Selecting Features

- Goal: Select feature set S that minimizes query time:

$$\min_{S} P(\text{approx})\text{cost}(S) + P(\sim\text{approx})\text{cost}(F)$$

- Approach:
    - Choose several potential values of $\text{cost}(S)$.
    - **Find best feature set with each cost(S).**
    - Train model & find cascade threshold for each set.
    - Pick best overall.

End-to-end Cascades: Selecting Features

- Subgoal: Find S minimizing query time if $cost(S) = c_{max}$.

$$\min_{S} P(\text{approx})\text{cost}(S) + P(\sim\text{approx})\text{cost}(F)$$

End-to-end Cascades: Selecting Features

- Subgoal: Find S minimizing query time if $cost(S) = c_{max}$.

$$\min_S P(\text{approx})\text{cost}(S) + P(\sim\text{approx})\text{cost}(F)$$

- Solution:
    - Find S maximizing approximate model accuracy.

End-to-end Cascades: Selecting Features

- Subgoal: Find S minimizing query time if $cost(S) = c_{max}$.

$$\min_{S} P(\text{approx})\text{cost}(S) + P(\sim\text{approx})\text{cost}(F)$$

- Solution:
  - Find S maximizing approximate model accuracy.
  - Problem: Computing accuracy expensive.

End-to-end Cascades: Selecting Features

- Subgoal: Find *S* minimizing query time if $cost(S) = c_{max}$.

$$\min_{S} P(\text{approx})\text{cost}(S) + P(\sim\text{approx})\text{cost}(F)$$

- Solution:
  - Find *S* maximizing approximate model accuracy.
  - Problem: Computing accuracy expensive.
  - Solution: Estimate accuracy via **permutation importance** -> knapsack problem.

End-to-end Cascades: Selecting Features

- Goal: Select feature set S that minimizes query time:

$$\min_{S} P(\text{approx})\text{cost}(S) + P(\sim\text{approx})\text{cost}(F)$$

- Approach:
  - Choose several potential values of cost(S).
  - Find best feature set with each cost(S).
  - **Train model & find cascade threshold for each set.**
  - Pick best overall.

End-to-end Cascades: Selecting Features

- Subgoal: Train model & find cascade threshold for S.

$$\min_{S} P(\text{approx})\text{cost}(S) + P(\sim\text{approx})\text{cost}(F)$$

- Solution:
  - Compute empirically on held-out data.

End-to-end Cascades: Selecting Features

- Subgoal: Train model & find cascade threshold for S.

$$\min_{S} P(\text{approx})\text{cost}(S) + P(\sim\text{approx})\text{cost}(F)$$

- Solution:
    - Compute empirically on held-out data.
    - Train approximate model from S.

End-to-end Cascades: Selecting Features

- Subgoal: Train model & find cascade threshold for S.

$$\min_{S} P(\text{approx})\text{cost}(S) + P(\sim\text{approx})\text{cost}(F)$$

- Solution:
  - Compute empirically on held-out data.
  - Train approximate model from S.
  - Predict held-out set, determine cascade threshold empirically using accuracy target.

End-to-end Cascades: Selecting Features

- Goal: Select feature set S that minimizes query time:

$$\min_{S} P(\text{approx})\text{cost}(S) + P(\sim\text{approx})\text{cost}(F)$$

- Approach:
  - Choose several potential values of $\text{cost}(S)$.
  - Find best feature set with each cost(S).
  - Train model & find cascade threshold for each set.
  - **Pick best overall.**

End-to-end Cascades: Results

- Speedups of up to 5x without statistically significant accuracy loss.
- Full evaluation at end of talk!

Outline

- System Overview
- Optimization 1: End-to-end Cascades
- **Optimization 2: Top-K Query Approximation**
- Evaluation

Top-K Approximation: Query Overview

- Top-K problem: Rank K highest-scoring items of a dataset.
- Top-K example: Find 10 artists a user would like most (recommender system).

Top-K Approximation: Asymmetry

- High-value items must be predicted, ranked precisely.
- Low-value items need only be identified as low value.

Artist	Score	Rank
Beatles	9.7	1
Bruce Springsteen	9.5	2
…	…	…
Justin Bieber	5.6	999
Nickelback	4.1	1000

High-value:
Rank precisely,
return.

Low-value:
Approximate,
discard.

Top-K Approximation: How it Works

- Use approximate model to identify and discard low-value items.
- Rank high-value items with powerful model.

Top-K Approximation: Prior Work

- Existing systems have similar ideas.
- However, we automatically generate approximate models for any ML application—prior systems don't.
- Similar challenges as in cascades.

Source:
Cheng et al.
(DLRS '16)

Top-K Approximation: Automatic Tuning

- Automatically selects features, tunes parameters to maximize performance given accuracy target.
- Works similarly to cascades.
- See paper for details!

Top-K Approximation: Results

- Speedups of up to 10x for top-K queries.
- Full eval at end of talk!

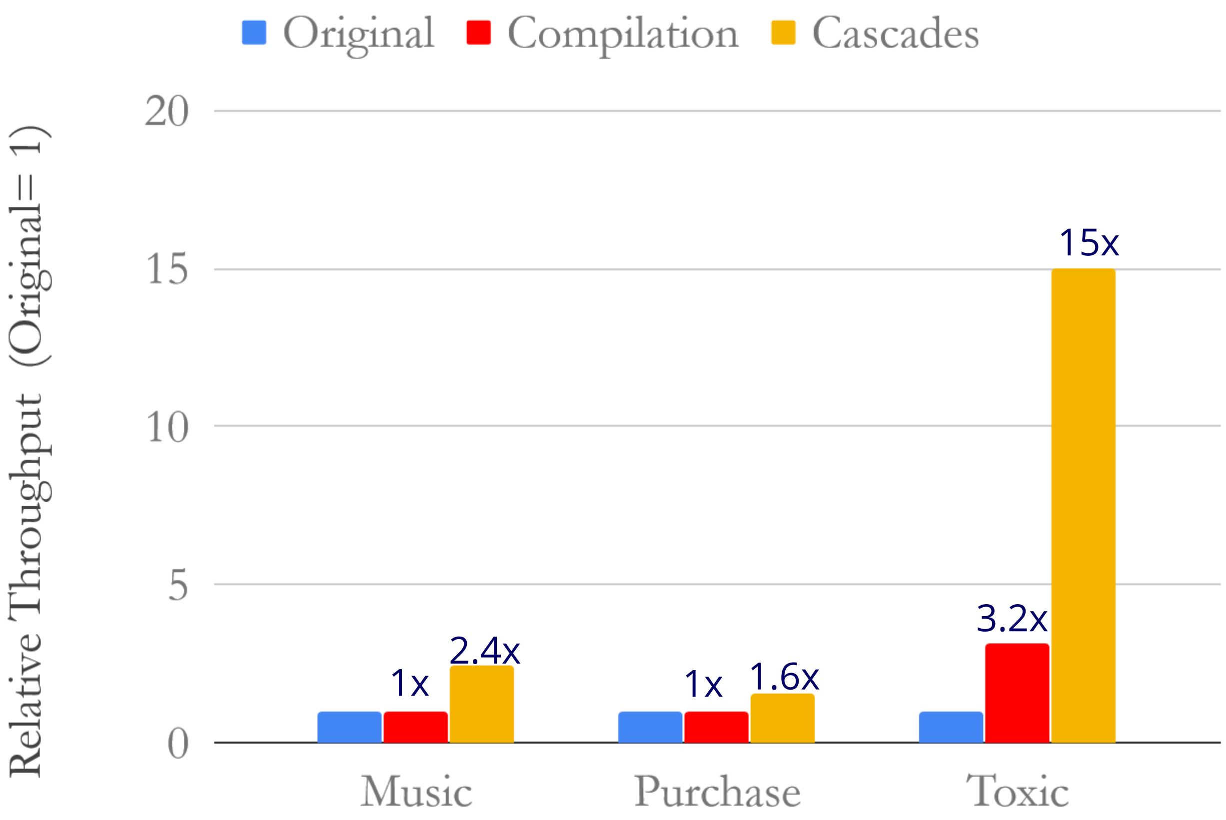
Outline

- System Overview
- Optimization 1: End-to-end Cascades
- Optimization 2: Top-K Query Approximation
- **Evaluation**

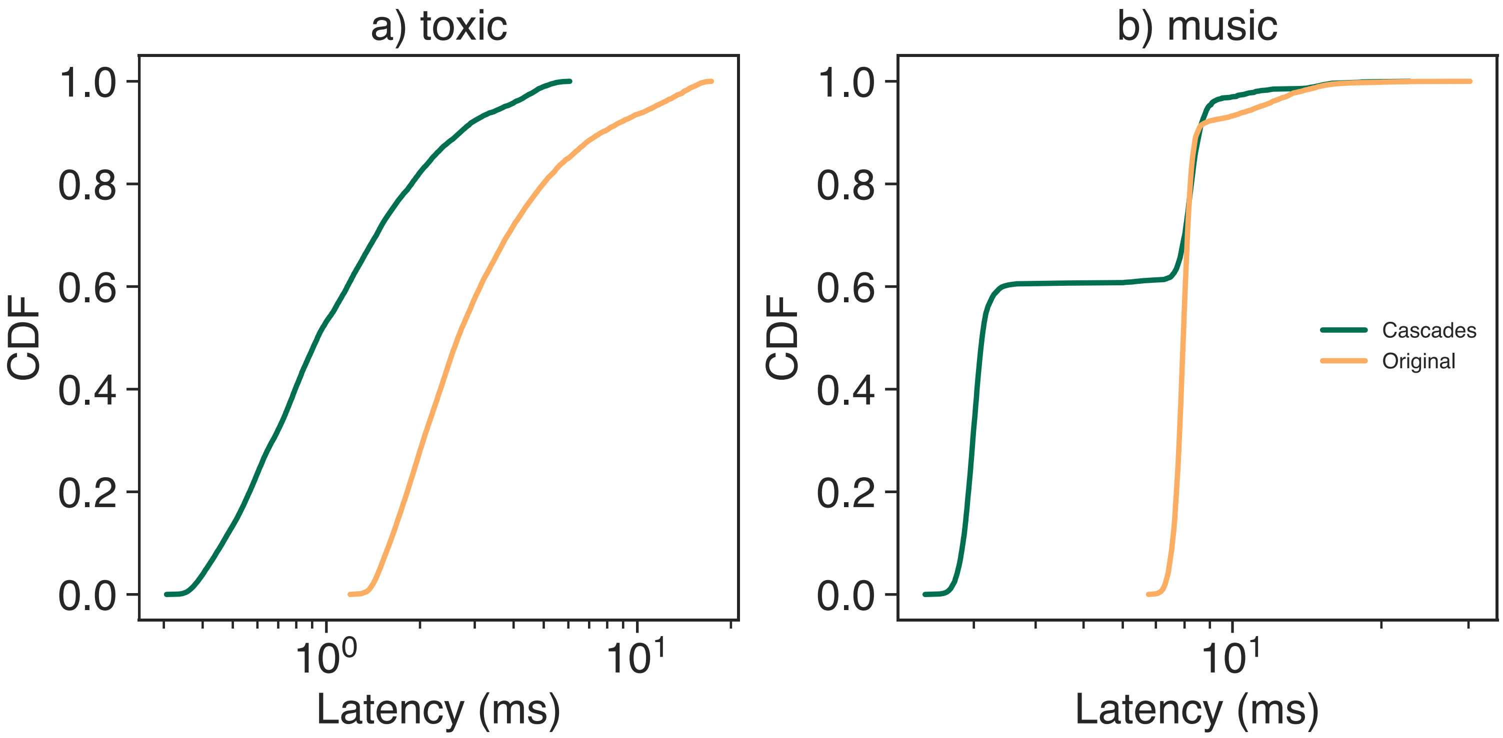
Willump Evaluation: Benchmarks

- Benchmarks curated from top-performing entries to data science competitions (e.g. Kaggle, WSDM, CIKM).
- Three benchmarks in presentation (more in paper):
  - **Music** (music recommendation– queries remotely stored precomputed features)
  - **Purchase** (predict next purchase, tabular AutoML features)
  - **Toxic** (toxic comment detection – computes string features)

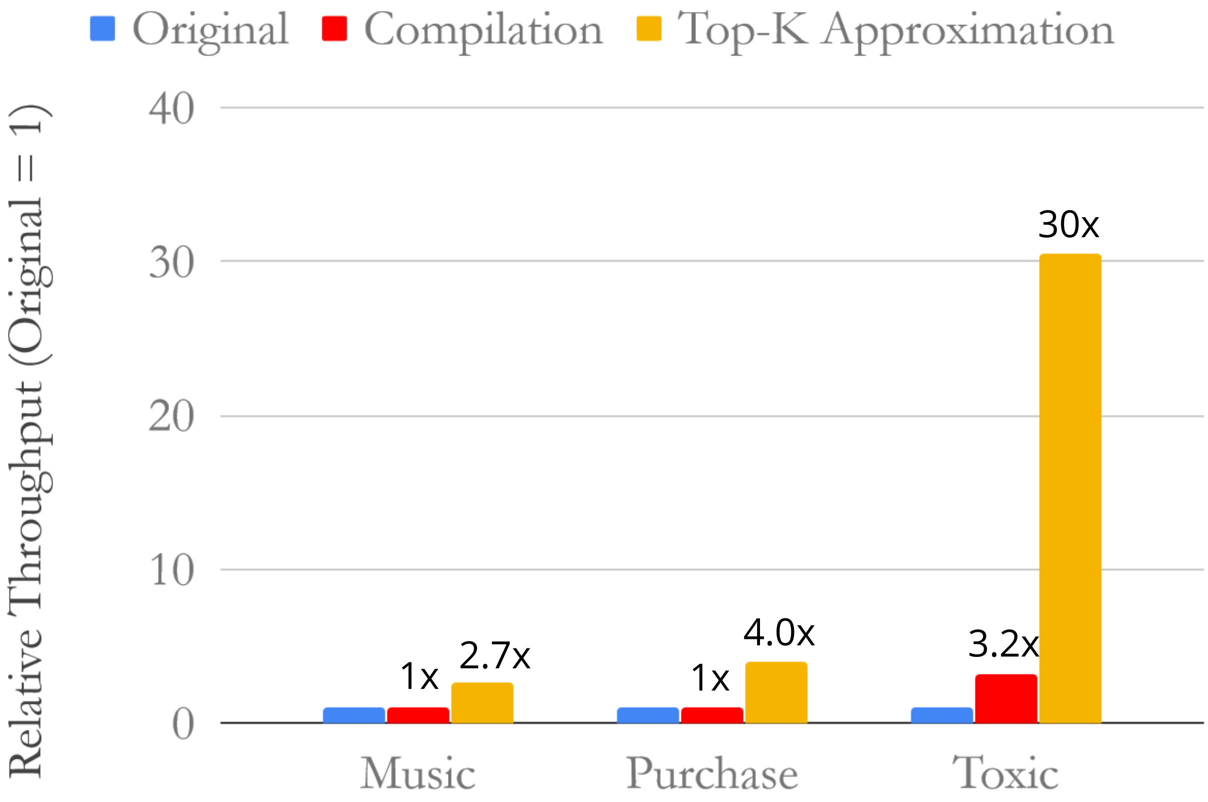
End-to-End Cascades Evaluation: Throughput



End-to-End Cascades Evaluation: Latency



Top-K Query Approximation Evaluation



Summary

- We introduce Willump, a statistically-aware end-to-end optimizer for ML inference.

- Statistical nature of ML enables new optimizations: Willump applies them automatically for 10x speedups.

github.com/stanford-futuredata/Willump