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

Peter Kraft, Daniel Kang, Deepak Narayanan, Shoumik Palkar, Peter Bailis, Matei Zaharia
Problem: ML Inference

- ML inference serving is important for many modern applications.
- Examples: spam detection, content recommendation.
- ML inference often performance-critical.
- Willump makes ML inference faster!
Example Pipeline: Music Recommendation

ML Inference Pipeline

Prediction: Will the user like the song?
Example Pipeline: Music Recommendation

User
User Features Lookup

Song
Song Features Lookup

Genre
Genre Features Lookup

Raw Inputs

Database Feature Lookups
Example Pipeline: Music Recommendation

Raw Inputs

User
   User Features Lookup

Song
   Song Features Lookup

Genre
   Genre Features Lookup

Feature Concatenation

Database Feature Lookups
Example Pipeline: Music Recommendation

- **User Features Lookup**
- **Song Features Lookup**
- **Genre Features Lookup**

**Feature Concatenation**

**Boosted Trees Model**

**Raw Inputs**

**Database Feature Lookups**

**Predict From Features**
Motivation: Bottlenecks in ML Inference Serving!

Production Microsoft sentiment analysis pipeline

Featurization takes **99.7%** of the time!

Source: Pretzel (OSDI ‘18)
Current State-of-the-art: Overview

- Existing systems approach ML workloads as an extension of traditional workloads (e.g. web serving, database queries).
Current State-of-the-art: Black-Box Systems

- Deploy models in containers with RPC frontends.
- Apply generic serving optimizations (e.g. caching, adaptive batch size tuning).
Current State-of-the-art: Compiler-based Systems

- Use and optimize a custom DSL (Pretzel) or ML framework (TensorRT, TVM).
- Apply traditional compiler optimizations (e.g. operator fusion).
Willump: Overview

- Optimizer for ML Inference.
- Leverages unique properties of ML inference.
- Optimizations improve performance by 3.6-5.7x over compilation.
- End-to-end improvement up to 23x.
Properties of ML Inference: Approximation

- ML models can be approximated without accuracy loss on many data inputs.
- Classify some data inputs with cheap pipeline, others with more expensive one ("cascading").
Properties of ML Inference: Approximation

- ML models can be approximated without accuracy loss on many data inputs.
- Classify some data inputs with cheap pipeline, others with more expensive one (“cascading”).
Properties of ML Inference: Query Modalities

- Applications give queries higher-level modalities.
- Differing throughput/latency requirements.
- Specific query semantics (e.g. top-K rankings).
Properties of ML Inference: Query Modalities

- Applications give queries higher-level modalities.
- Differing throughput/latency requirements.
- Specific query semantics (e.g. top-K rankings).

Optimization: Query-Aware Inference
Outline

- **Willump Workflow**
- Optimization 1: End-to-end Cascades
- Optimization 2: Query-Aware Inference
- Evaluation
def pipeline(x1, x2):
    input = lib.transform(x1, x2)
    preds = model.predict(input)
    return preds
def pipeline(x1, x2):
    input = lib.transform(x1, x2)
    preds = model.predict(input)
    return preds
def pipeline(x1, x2):
    input = lib.transform(x1, x2)
    preds = model.predict(input)
    return preds

Willump Workflow: Optimization Stage

User Pipeline

Willump Optimization

Optimizations:
1. End-to-end Cascades
2. Query-Aware Inference
def pipeline(x1, x2):
    input = lib.transform(x1, x2)
    preds = model.predict(input)
    return preds

Willump Workflow: Compilation Stage

User Pipeline

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

Willump Optimization

Infer Transformation Graph

Optimizations:
1. End-to-end Cascades
2. Query-Aware Inference

Compile optimized graph through Weld

01010101010111
01101110101101
01101010111111
01010101101101
01010101010111
01101110101101
01101010111111
01010101101101
def pipeline(x1, x2):
    input = lib.transform(x1, x2)
    preds = model.predict(input)
    return preds

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

Willump Optimization

Infer Transformation Graph

Optimizations:
1. End-to-end Cascades
2. Query-Aware Inference

Compile optimized graph through Weld

01010101010111
01101110101101
01101010111111
01010101101101

Willump Workflow: Ending Point
Outline

- Willump Workflow
- Optimization 1: End-to-end Cascades
- Optimization 2: Query-Aware Inference
- Evaluation
Background: Model Cascades

- Approximate “easy” data inputs with cheap model.
- *Cascade* to expensive model for “hard” data inputs.
- Used for image classification and object detection.
- Application-specific, doesn’t generalize.

Source: Viola-Jones (CVPR’ 01)
Our Optimization: End-to-end Cascades

- Extend cascading to feature computation.
- Compute only some features for easy data inputs.
- Cascade to all features for hard data inputs.
End-to-end Cascades: Original Pipeline

Compute Features → Concatenate Features → Model → Prediction
End-to-end Cascades: Partition Features

1. Compute Efficient Features
2. Compute Inefficient Features
3. Concatenate Features
4. Model
5. Prediction
End-to-end Cascades: Choosing Efficient Features

- Analyze dataflow to find features computed together.
- Compute features’ computational cost and importance to model (e.g. weight).
End-to-end Cascades: Choosing Efficient Features

- Greedily select most cost-effective features.
- Stop when efficient features more expensive than remainder.
- Stop when next feature much less cost-effective than average efficient feature.
End-to-end Cascades: Small Model

- Compute Efficient Features
- Compute Inefficient Features
- Concatenate Features
- Model
- Prediction

Cascades Optimization

- Compute Efficient Features
- Small Model
- Prediction
End-to-end Cascades: Confidence

Compute Efficient Features

> Concatenate Features

> Model

> Prediction

Compute Inefficient Features

> Compute Efficient Features

> Confidences > Threshold

Yes

Cascades Optimization

Small Model

Prediction
End-to-end Cascades: Final Pipeline

- Compute Efficient Features
- Compute Inefficient Features

Concatenate Features

Model

Prediction

Cascades Optimization

- Compute Efficient Features
- Compute Inefficient Features

Concatenate Features

Model

Prediction

Confidence > Threshold

Small Model

Yes

Full Model

Prediction

No

Compute Inefficient Features

Confidence > Threshold

Small Model

Yes

Full Model
End-to-end Cascades: Results

- Speedups of up to 4x without statistically significant accuracy loss.
- Full eval at end of talk!
Outline

- Willump Workflow
- Optimization 1: End-to-end Cascades
- Optimization 2: Query-Aware Serving
- Evaluation
Query-Aware Serving

- Applications give queries higher-level modalities.
- Some queries have tight latency constraints.
- Some queries inherit application semantics (e.g. top-K ranking).
Query-Aware Serving: Low-latency Parallelization

- All model serving systems parallelize batch queries by element, for throughput.
- Willump can parallelize computation of individual data inputs for latency.
- More details in paper!
Query-Aware Serving: Top-K Queries

- Top-K problem: Rank K highest-scoring elements of a batch.
- Top-K example: Find 10 songs a user would like most (recommender system).
Query-Aware Serving: Top-K Asymmetry

- High-value elements must be predicted, ranked precisely.
- Low-value elements need only be identified as low value.
Query-Aware Serving: Top-K Filtering

- Use cheap “filter model” to identify and discard low-value elements.
- Predict high-value elements with full model, return ranking.
Query-Aware Serving: Top-K Cascades

- Use cascades approximate model as filter!
- Example (music recommendation):
  1. Compute genre features.
  2. Discard songs from genres the user dislikes.
  3. Rank remaining songs with full model.
Query-Aware Serving: Results

- Speedups of up to 5.7x for top-K queries.
- Full eval at end of talk!
Outline

- Willump Workflow
- Optimization 1: End-to-end Cascades
- Optimization 2: Query-Aware Inference
- 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):
  - Toxic (toxic comment detection)
  - Product (assessing goods in online store)
  - Music (music recommendation)
Benchmark Transformation Graphs

Product

Toxic

Music
End-to-End Cascades Evaluation: Performance

- Toxic: 16x
- Product: 3.5x
- Music: 4.3x

Speedup (Python=1)
End-to-End Cascades Evaluation: Performance

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Python</th>
<th>Compilation</th>
<th>Cascades</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toxic</td>
<td>4.1x</td>
<td>16x</td>
<td></td>
</tr>
<tr>
<td>Product</td>
<td>3.5x</td>
<td>7.3x</td>
<td></td>
</tr>
<tr>
<td>Music (Local)</td>
<td>4.3x</td>
<td>4.3x</td>
<td></td>
</tr>
<tr>
<td>Music (Remote)</td>
<td>1.0x</td>
<td>1.4x</td>
<td></td>
</tr>
</tbody>
</table>
Parallelization Results

![Graph showing speedup vs. number of threads]

- **X-axis**: Number of threads
- **Y-axis**: Speedup
- **Lines**:
  - Light blue: toxic
  - Dark blue: product
Parallelization Results
Top-K Results

- Toxic: 23x
- Product: 15x
- Music (Remote): 3.3x

Speedup (Python=1)

- Python
- Compilation
- Cascades
Integration Possibilities

- Willump integrates with general-purpose model serving systems.
- Optimize with Willump, serve with existing system (e.g. Clipper)
Clipper Integration Results

Latency on Toxic with varying batch size $b$. 

- $b=1$: 3x
- $b=10$: 3.5x
- $b=100$: 6.8x
Summary

- We introduce Willump, a statistically-aware end-to-end optimizer for ML inference.
- We present novel ML inference optimizations.
  1. End-to-End Cascades.
  2. Query-Aware Inference.
- Willump speeds up ML inference by up to 23x.