Managed & Model-less Inference Serving

Francisco Romero*, Qian Li*, Neeraja J.Yadwadkar, and Christos Kozyrakis
A simplified view of the ML life cycle: Training

Datasets → Training Algorithms → Model

- Data Cleaning
- Empirical Data Analysis
- Model Design

Caffe2 · Tensorflow · MXNet · PyTorch
A simplified view of the ML life cycle:

Datasets → Training Algorithms → Model → Prediction → Query → End-user applications
A simplified view of the ML life cycle

Datasets → Training Algorithms → Model → Prediction

Lots of work from ML as well as Systems communities

End-user applications

Training

Only recently started receiving attention

Inference
Growing importance of inference serving

Running Inference in Production drives the majority of cost for ML - about 90%!
Growing importance of inference serving

TF Model Serving

TensorRT Model Serving

MauveDB: Supporting Model-based User Views in Database Systems

LASER: A Scalable Response Prediction Platform For Online Advertising

NoScope: Optimizing Neural Network Queries over Video at Scale

Clipper: A Low-Latency Online Prediction Serving System

Abstract

Machine learning is being deployed in a growing number...
Same model, multiple *model-variants*!

- Compiler optimizations
- TVM, TensorRT

Trained Neural Network → TensorRT Optimizer → TensorRT Network for faster inference
Same model, multiple *model-variants*!

- **Compiler optimizations**
  - TVM, TensorRT

- **Different precisions**
  - INT8, FP16, FP32

- **Hyperparameter optimizations**
  - Batch size

We can compile the same model to 10s (100s) of versions
Heterogeneous hardware architectures

- CPUs
- GPUs
- FPGAs
- ASICs

Different accuracy, cost, performance, and energy trade-offs for inference
Model Registration

User

Register a model

Model Repository

Today’s Inference serving

Invoke a Model

User(s)

Inference

Select a model

- ResNet
- DenseNet
- SqueezeNet
- Inception

Framework

- PyTorch
- Caffe2
- mxnet

Hardware architecture

- CPUs
- GPUs
- FPGAs
- ASICs

Optimizer

- SensorRT
- TVM

Models x Frameworks x Hardware x Optimizers → a very large search space
Diverse application requirements

- Example: Face Recognition

- Object Detection

- Face Recognition

Accuracy vs. Latency

- Social Media

- Latency

- Navigation for visually impaired person

- Cost
Such overprovisioning leads to high costs
Today's Inference serving

Model-less Inference serving

Prediction task:

“Object Detection”,

Training set

App Reqs:

Target accuracy
Target latency

Invoke a Model

User(s)

Inference

Today's Inference Serving

Select a model

- ResNet
- DenseNet
- SqueezeNet
- Inception

Framework

- TVM

Hardware architecture

- CPUs
- GPUs
- FPGAs
- ASICs

Optimizer

- TensorRT
- TVM

App Reqs

Target accuracy
Target latency
Prediction task: “Object Detection”
Training set
App Reqs:
Target accuracy
Target latency

Automatically and efficiently select a model, and hardware
Share the hardware as well as models across users
An INFerence-as-a-Service system that abstracts resource management and model selection from users

INFaaS achieves ease-of-use and resource efficiency by:

• A model-less API
• Sharing of models and resources
• Autoscaling

2x higher throughput

2.8x fewer SLO violations

38% cost savings
Outline

• **INFaaS overview and workflow**
• Model selection and scaling policies
• Evaluation
INFaas overview

0 [One-time]: Users register models

1 The user submits a query using the model-less API

Classification in 200ms

“Cat”

Register Model

“OK”

0

1

INFaas Front-end

INFaas API

Clients

INFaaS

Master

Dispatcher & Load Balancer

Autoscaler

Model Registrar

Worker

Dispatcher

GPU Executor

CPU Executor

Monitoring Daemon & Autoscaler

Metadata Store

Model Repository

Model Profiler and Optimizer
The *model-less* abstraction

**Use-case**

**Model architecture**

**Model-variant**

- classification-imagenet
- vgg16
- resnet50
- vgg16-pytorch-cpu
- resnet50-tf-cpu
- resnet50-caffe2-gpu
The model-less abstraction allows INFaaS to map query requirements to underlying models and resources.
INFaas overview

1. [One-time]: Users register models
2. The user submits a query using the model-less API
3. The master selects a model-variant, then selects a worker to process the query
4. The query proceeds to run on the variant’s target hardware platform
5. Upon completion, the result is returned to the user
INFaaS is easy to use!

<table>
<thead>
<tr>
<th></th>
<th>INFaaS</th>
<th>CloudML</th>
<th>SageMaker</th>
<th>Clipper</th>
<th>TFS / TRTIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model-variant selection</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Hardware selection</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Scaling strategy configuration</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>N/A</td>
</tr>
<tr>
<td>Online / Offline setup</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

INFaaS removes the system configuration burden and improves ease-of-use
Outline

• INFaaS overview and workflow
• Model selection and scaling policies
• Evaluation
Selecting a model-variant

• **Goal**: meet diverse user requirements while taking the system’s state (e.g., running or overloaded variant instances) into account

• **Insights**:
  • Leverage model-variant diversity
  • Organize metadata store according to *model-less* abstraction
  • Cache decisions
Selecting a model-variant – Scenario 1

**Scenario**: Requirement cached, and cached variant is running

**Outcome**: Send query to running variant on least-loaded worker
Selecting a model-variant – Scenario 2

Scenarios:
- Requirement cached, and cached variant is not running
- Requirement not cached

Outcome: Search through all variants; send query to running variant that meets requirements on least-loaded worker
Selecting a model-variant – Scenario 3

Scenarios:
- Requirement cached, and cached variant is not running
- Requirement not cached

Outcome: Search through all variants; start variant with lowest load-inference latency on worker with lowest utilization
Autoscaling – Master

- **Goal**: dynamically adjust the number of each type of worker machine (CPU/GPU) based on resource utilization and SLO violations.
**Autoscaling – Worker**

- **Goal:** respond to changes in load to meet SLOs by: 1) replication and 2) upgrading/downgrading

![Diagram of Autoscaler with GPU Worker, Autoscaler, and Replication nodes](image)
**Autoscaling – Worker**

- **Goal:** respond to changes in load to meet SLOs by: 1) replication and 2) upgrading/downgrading

---

### GPU Worker

- **Autoscaler**
  - Load (Reqs/sec)
  - SLO violations

- **Σ-GPU batch 1**
  - Cost
    - Startup latency
    - Peak memory
    - Hardware cost

- **Σ-GPU batch 8**

- **Upgrading**
Outline

• INFaaS overview and workflow
• Model selection and scaling policies
• **Evaluation**
Evaluation

• We deploy INFaaS on AWS EC2
  • GPU worker has 1 NVIDIA V100 GPU
  • master / CPU worker / client are CPU-only machines

• Baselines:
  • TensorRT Inference Server (TRTIS), TensorFlow Serving (TFS), Clipper, SageMaker, and CloudML
  • STATIC (TRTIS and TFS): preload and persist variants
  • GPU-S and CPU-S
  • INDV (Clipper, SageMaker, and CloudML): replication on/across worker
How well does INFaaS scale on a single worker?

ResNet50
- TensorFlow CPU
  - avg latency 500ms
- TensorRT optimized for batch 1 / 4 / 8 / 16 (GPU)
  - avg latency 20ms
How well does INFaaS scale on a single worker?

CPU-S provides lowest cost, but violates SLOs at high load.

GPU-S achieves lowest latency and meets the load, but is expensive.

ResNet50
- TensorFlow CPU
  - avg latency 500ms
- TensorRT optimized for batch 1 / 4 / 8 / 16
  - avg latency 20ms

Baselines
- CPU-S preloads two TF-cpu
- GPU-S preloads one trt-8
How well does INFaaS scale on a single worker?

INDV fails to meet SLOs at higher load

INFaaS proactively upgrades / unloads variants: 38% cost savings over GPU-S

ResNet50
- TensorFlow CPU
  - avg latency 500ms
- TensorRT optimized for batch 1 / 4 / 8 / 16
  - avg latency 20ms

Baselines
- CPU-S preloads two TF-cpu
- GPU-S preloads one trt-8
- INDV: TF-cpu for 500ms and trt-1 for 20ms SLO
How well does INFaaS scale on a single worker?

- INFaaS scales and adapts to changes in load and query patterns, and maintains low cost by better resource allocation.

- ResNet50:
  - TensorFlow CPU
    - avg latency 500ms
  - TensorRT optimized for batch 1 / 4 / 8 / 16
    - avg latency 20ms

- Baselines:
  - CPU-S preloads two TF-cpu
  - GPU-S preloads one trt-8
  - INDV: TF-cpu for 500ms and trt-1 for 20ms SLO

- INFaaS proactively upgrades / unloads variants: 38% cost savings over GPU-S

- INDV fails to meet SLOs at higher load
Summary of other results

- Realistic workload with 270 variants:
  - INFaaS improves throughput by $2\times$ at high load
  - INFaaS reduces SLO violations by at least $2.8\times$
  - INFaaS uses fewer worker machines at low load
- Low decision overheads

INFaaS achieves high performance and resource utilization
Future work

• Support for new hardware
• White box inference serving
• Security and privacy
Conclusion

• INFaaS is a managed and model-less inference serving system that:
  • Exposes a model-less API to query models and specify performance needs
  • Leverages multi-tenant sharing across models and users
  • Manages when model-variants should be scaled or upgraded

• INFaaS improves ease-of-use and cost for both users and providers

https://github.com/stanford-mast/INFaaS

Questions?

{faromero,qianl15,neerajay,kozyraki}@stanford.edu