RISE to the Challenges of AI Systems

Joseph E. Gonzalez
Assistant Professor, UC Berkeley
jegonzal@cs.berkeley.edu
Large-Scale parallel and distributed systems
Big Data → Training → Big Model

GraphLab

GraphX
How to do Research in AI Systems

- **Manage Complexity**
  - seek parsimony in system design
  - great systems research is often about what features are taken away
  - Do a few things well and be composable

- **Identify Tradeoffs**
  - With each design decision what do you gain and lose?
  - What trade-offs are fundamental?

- **Evaluate your System**
  - **Positive:** How fast and scalable is it and *why?*
  - **Negative:** When does it fail and what are it’s *limitations*?
What is the best algorithm and level of parallelism for an ML task?

**Trade-off:** Parallelism, Coordination, & Convergence

**Research challenge:** Can we model this trade-off explicitly?

We can estimate $I$ from data on many systems

We can estimate $L$ from data for our problem

*$follow-up work to Shivaram’s Ernest System in NSDI’16*
Hemingway*  
Modeling Throughput and Convergence for ML Workloads

- What is the best algorithm and level of parallelism for an ML task?  
  - **Trade-off:** Parallelism, Coordination, & Convergence
- **Research challenge:** Can we model this trade-off explicitly?

\[
L(i, p) \quad \text{Loss as a function of iterations } i \text{ and cores } p
\]
\[
I(p) \quad \text{Iterations per second as a function of cores } p
\]

\[
\text{loss}(t, p) = L(t \times I(p), p)
\]

- How long does it take to get to a given loss?  
- Given a time budget and number of cores which algorithm will give the best result?

*follow-up work to Shivaram’s Ernest System in NSDI’16
System Performance as a function of **Parallelism**

Convergence as a function of **Parallelism** and **Iterations**

**Training Loss**

**Convergence** as a function of **Time** and **Parallelism**

**Hemingway: Modeling Distributed Optimization Algorithms.**

Xinghao Pan, Shivaram Venkataraman, Zizheng Tai, Joseph Gonzalez.

NIPS’16 ML-Sys Workshop.
Take away ...

try to decouple

System Improvements

Algorithm Improvements

use data collection + sparse modeling to understand your system
Learning

Big Data → Training → Big Model → ?
Big Data → Training → Big Model → Conference Papers
Learning

Big Data

Training

Big Model

Conference Papers

Dashboards and Reports
Big Data → Training → Big Model → Conference Papers and Dashboards → Drive Actions
Learning

Big Data → Training → Big Model

Drive Actions

Hi, I'm Cortana.
Big Data → Training → Big Model → Inference
Big Data → Training → Big Model → Inference → Application

Learning

Inference

- Query
- Decision
Big Data Training

Learning

Big Model

Inference

Query

Decision

Application

Often **overlooked**

Timescale: ~10 milliseconds

**Billions of Queries a Day** → **Costly**
why is **Inference** challenging?

Need to render **low latency** (< 10ms) predictions for **complex** models under **heavy load** with system failures.

**Models**

**Queries**

**Features**

```sql
SELECT * FROM users JOIN items, click_logs, pages WHERE ...
```
Inference is moving beyond the cloud

Augmented Reality
Home Security
Home Automation
Mobile Assistants
Self Driving Cars
Personal Robotics
Inference is moving beyond the cloud

Opportunities
- Reduce latency and improve privacy
- Address network partitions

Research Challenges
- Minimize power consumption
- Limited hardware & long life-cycles
- Develop new hybrid models to leverage the cloud and edge devices
Robust Inference is critical

Self “Parking” Cars

Self “Driving” Cars

Chat AIs
Big Data Training Application

Learning

Time scale: hours to weeks
Often re-run training
Sensitive to feedback loops

Inference

Application

Feedback
Why is **Closing the Loop** challenging?

- Implicit and Delayed Feedback
- Self Reinforcing Feedback Loops
- World Changes at varying rates
Learning
Adaptive
(~1 seconds)

Inference
Responsive
(~10ms)
Learning
Adaptive (~1 seconds)

Inference
Responsive (~10ms)

Secure
Intelligence in Sensitive Contexts

Augmented Reality | Home Monitoring | Voice Technologies | Medical Imaging

Protect the data, the model, and the query
Protect the **data**, the **model**, and the **query**

**High-Value Data is Sensitive**
- Medical Info.
- Home video
- Finance

**Models capture value in data**
- Core Asset
- Sensitive

**Queries** can be as sensitive as the data
Opaque: Analytics on Secure Enclaves

Exploit hardware support to enable computing on encrypted data

- **Today:** prototype system running in Apache Spark
  - support SQL queries in untrusted cloud
  - ~50% reduction in perf.
- **Future:** enable prediction serving on enc. queries

Wenting et al. (NSDI’17)
Clipper
A Low-Latency Online Prediction Serving System
NSDI’17
Daniel Crankshaw
Xin Wang
Giulio Zhou
Michael J. Franklin
Joseph E. Gonzalez
Ion Stoica
Big Data Training Application Learning Inference Feedback Decision Query Application
Big Data

Learning

Inference

- **Training**
- **Slow Changing Parameters**
- **Fast Changing Parameters**
- **Query**
- **Decision**
- **Application**

Feedback
Hybrid Offline + Online Learning

Update “feature” functions offline using batch solvers:
- Leverage high-throughput systems (Tensor Flow)
- Exploit slow change in population statistics

\[ f(x; \theta)^T \]

Update the user weights online:
- Simple to train + more robust model
- Address rapidly changing user statistics
Common modeling structure

\[ f(x; \theta)^T w_u \]

Matrix Factorization

Deep Learning

Ensemble Methods

Users

Items

Ratings

Input
Clipper Online Learning for Recommendations (Simulated News Rec.)

Partial Updates: 0.4 ms
Retraining: 7.1 seconds

>4 orders-of-magnitude faster adaptation
Learning

Big Data → Slow Changing Parameters → Fast Changing Parameters → Application

Feedback: Slow → Fast

Inference
Caffe Big Data Application Learning Inference

Slow Changing Parameters

Caffe TensorFlow

Fast Changing Parameters

Clipper

Big Data

Feedback

Application
Clipper Serves Predictions across ML Frameworks

- Fraud Detection
- Content Rec.
- Personal Asst.
- Robotic Control
- Machine Translation

Clipper

Frameworks:
- theano
- Dato Create
- Caffe
- TensorFlow
- scikit-learn
- Keystone ML
- VW
- dmlc
- mxnet
- KALDI
Key Insight:
The challenges of prediction serving can be addressed between end-user applications and machine learning frameworks

As a result, Clipper is able to:

- **hide complexity** by
  - providing a common interface to applications
- **bound latency** and **maximize throughput**
  - through caching, adaptive batching, model replication
- **enable robust online learning** and **personalization**
  - through model selection and ensemble algorithms

without modifying machine learning frameworks or front-end applications
Clipper Architecture

Applications

Predict

RPC/REST Interface

Observe

Clipper

Frameworks and Libraries:
- Theano
- Dato
- Caffe
- TensorFlow
- scikit-learn
- dmlc mxnet
- Volkswagen
- Kaldi
Clipper Architecture

Applications

Predict \(\uparrow\)  
RPC/REST Interface  
Observe \(\downarrow\)

Clipper

RPC \(\uparrow\)

Model Wrapper (MW)

KeystoneML

RPC \(\uparrow\)

MW

Caffe

RPC \(\uparrow\)

MW

RPC \(\uparrow\)

MW

RPC \(\uparrow\)

MW
Clipper Architecture

**Applications**

**Predict**

**RPC/REST Interface**

**Observe**

**Clipper**

- Improve accuracy through bandit methods, ensembles, online learning, and personalization.

- Provide a common interface to models while bounding latency and maximizing throughput.

**Model Selection Layer**

**Model Abstraction Layer**

RPC/REST Interface

Model Wrapper (MW)

RPC

MW

RPC

MW

RPC

MW

RPC

MW

RPC

MW
Model Selection Layer

Caching

Adaptive Batching

Model Abstraction Layer

RPC

Model Wrapper (MW)

KeystoneML

Caffe

scikit-learn
Provide a **common interface** to models while **bounding latency** and **maximizing throughput**.

- Models run in separate processes as Docker containers
  - Resource isolation
Provide a **common interface** to models while **bounding latency** and **maximizing throughput**.

- Models run in separate processes as Docker containers
  - Resource isolation
  - Scaling under heavy load

**Problem:** frameworks optimized for **batch processing** not **latency**
Adaptive Batching to Improve Throughput

Why batching helps:
- A single page load may generate many queries

Optimal batch depends on:
- hardware configuration
- model and framework
- system load

Clipper Solution:
- be as slow as allowed...
- Application specifies latency objective
- Clipper uses TCP-like tuning algorithm to increase latency up to the objective
Tensor Flow Conv. Net (GPU)

Latency (ms)

Throughput (Queries Per Second)

Batch Sizes (Queries)

Optimal Batch Size

Latency Deadline

P99 Latency

Avg. Latency

Throughput

Throughput
Within their own lightweight container (Section 4.4), Clipper deploys models connected to Clipper via a lightweight RPC system. This modular architecture enables caching in Clipper, which serves an important role in reducing overhead and simplifying cross-language integration.

The model abstraction layer (Figure 1) is composed of a prediction cache, an adaptive query-batching component, and a set of algorithmic innovations associated with each. The batching component (Section 4.3) sits below the prediction cache and batch predictions to improve throughput. In addition, caching in Clipper serves an important role in increasing latency by requiring all queries in the batch to be evaluated simultaneously.

The batching component transforms the stream of prediction queries received by Clipper into batches that are dynamically resized for each model container. The batching component improves throughput, but it does so at the expense of increased feedback processing latency.

Clipper needs to join the original predictions with any feedback it receives. Since feedback is likely to return infrequently, Clipper needs to join the original predictions with any feedback it receives. Since feedback is likely to return infrequently, Clipper needs to join the original predictions with any feedback it receives.

The batching component transforms the stream of prediction queries received by Clipper into batches that are dynamically resized for each model container. The batching component improves throughput, but it does so at the expense of increased feedback processing latency.

The batching component transforms the stream of prediction queries received by Clipper into batches that are dynamically resized for each model container. The batching component improves throughput, but it does so at the expense of increased feedback processing latency.
The model abstraction layer (Section 4.4) is composed of a prediction interface across machine learning frameworks. The model containers connected to Clipper via a lightweight RPC system provide uniform interface to Clipper and simplify the deployment of new models. The lightweight RPC system is constructed for a given model it is dispatched via the RPC container to maximize throughput. Once a mini-batch is constructed for a model, it is dynamically resized for each model container to maximize throughput. For many applications (e.g., content recommendation), predictions concerning popular items are requested frequently. By maintaining a prediction cache, Clipper can serve these frequent queries without evaluating the model. This substantially reduces latency and system load by eliminating the additional cost of model evaluation.


The model selection layer joins this feedback with the corresponding pre-materialization mechanism for frequent queries as a partial pre-materialization mechanism for hot items. Any feedback the application collects about the quality of recent predictions and feedback. This layer is responsible for selecting models based on the confidence estimate and replies to the end-user application.


The batching component (Section 4.3) sits below the prediction cache and translates point queries into mini-batches that are dynamically resized for each model container. The batching component increases latency by requiring all queries in the batch to complete before returning a single prediction. However, because adaptive model selection occurs as a partial pre-materialization mechanism for hot items, infrequent queries will not be evicted and the cache serves predictions made by machine learning frameworks while simultaneously amortizing RPC and system overhead. Batching queries that more closely match the workload assumption.


The batching component reduces latency by requiring all queries in the batch to complete before returning a single prediction. However, because adaptive model selection occurs as a partial pre-materialization mechanism for hot items, infrequent queries will not be evicted and the cache serves predictions made by machine learning frameworks while simultaneously amortizing RPC and system overhead. Batching queries that more closely match the workload assumption.


ture and simplifies cross-language integration. The batching component transforms the stream of prediction queries received by Clipper into batches of predictions. The batching component increases latency by requiring all queries in the batch to complete before returning a single prediction. However, because adaptive model selection occurs as a partial pre-materialization mechanism for hot items, infrequent queries will not be evicted and the cache serves predictions made by machine learning frameworks while simultaneously amortizing RPC and system overhead. Batching queries that more closely match the workload assumption.


ture and simplifies cross-language integration. The batching component transforms the stream of prediction queries received by Clipper into batches of predictions. The batching component increases latency by requiring all queries in the batch to complete before returning a single prediction. However, because adaptive model selection occurs as a partial pre-materialization mechanism for hot items, infrequent queries will not be evicted and the cache serves predictions made by machine learning frameworks while simultaneously amortizing RPC and system overhead. Batching queries that more closely match the workload assumption.


turing whether the entry is in the cache. The cache checks the cache and returns the query result if present. If the entry is not already present, it returns a boolean indicating that the query is not found.


generic prediction function: Predict(m: ModelId, x: X) -> y: Y. The cache exposes a simple non-blocking put function to store the corresponding model prediction put(m, x) = y. The cache fetches the currently stored prediction for the given model and query using the standard CLOCK [16] cache eviction algorithm. With an adequately sized cache, frequent queries will not be evicted and the cache serves predictions made by machine learning frameworks while simultaneously amortizing RPC and system overhead. Batching queries that more closely match the workload assumption.


ture and simplifies cross-language integration. The batching component transforms the stream of prediction queries received by Clipper into batches of predictions. The batching component increases latency by requiring all queries in the batch to complete before returning a single prediction. However, because adaptive model selection occurs as a partial pre-materialization mechanism for hot items, infrequent queries will not be evicted and the cache serves predictions made by machine learning frameworks while simultaneously amortizing RPC and system overhead. Batching queries that more closely match the workload assumption.


turing whether the entry is in the cache. The cache checks the cache and returns the query result if present. If the entry is not already present, it returns a boolean indicating that the query is not found.


generic prediction function: Predict(m: ModelId, x: X) -> y: Y. The cache exposes a simple non-blocking put function to store the corresponding model prediction put(m, x) = y. The cache fetches the currently stored prediction for the given model and query using the standard CLOCK [16] cache eviction algorithm. With an adequately sized cache, frequent queries will not be evicted and the cache serves predictions made by machine learning frameworks while simultaneously amortizing RPC and system overhead. Batching queries that more closely match the workload assumption.


ture and simplifies cross-language integration. The batching component transforms the stream of prediction queries received by Clipper into batches of predictions. The batching component increases latency by requiring all queries in the batch to complete before returning a single prediction. However, because adaptive model selection occurs as a partial pre-materialization mechanism for hot items, infrequent queries will not be evicted and the cache serves predictions made by machine learning frameworks while simultaneously amortizing RPC and system overhead. Batching queries that more closely match the workload assumption.


turing whether the entry is in the cache. The cache checks the cache and returns the query result if present. If the entry is not already present, it returns a boolean indicating that the query is not found.


generic prediction function: Predict(m: ModelId, x: X) -> y: Y. The cache exposes a simple non-blocking put function to store the corresponding model prediction put(m, x) = y. The cache fetches the currently stored prediction for the given model and query using the standard CLOCK [16] cache eviction algorithm. With an adequately sized cache, frequent queries will not be evicted and the cache serves predictions made by machine learning frameworks while simultaneously amortizing RPC and system overhead. Batching queries that more closely match the workload assumption.
Overhead of modularity?

The decoupled Clipper architecture can be as fast as the in-process approach adopted by TensorFlow-Serving.

Better

40000 is Good Enough

The decoupled Clipper architecture can be as fast as the in-process approach adopted by TensorFlow-Serving.
Approximate Caching to Reduce Latency

- Opportunity for caching
  Popular items may be evaluated frequently

- Need for approximation
  High Dimensional and continuous valued queries have low cache hit rate.

Clipper Solution: Approximate Caching

- Apply locality sensitive hash functions

- Bag-of-Words Model
  Images

- Cache Hit
- Cache Miss
- Cache Hit
- Error
Clipper Architecture

Applications

Predict ↑

RPC/REST Interface

Observe ↓

Clipper

Selection Policy

Model Selection Layer

Caching

Model Abstraction Layer

Adaptive Batching

RPC

RPC

RPC

RPC

Model Wrapper (MW)

MW

MW

MW

...
Goal:

Maximize **accuracy** through **bandits**, **ensembles**, **online learning**, and **personalization**

Incorporate feedback in real-time to achieve:

- **robust predictions** by adaptively combining predictions from multiple models and frameworks
- **online learning** and **personalization** by selecting and personalizing **predictions** in response to feedback
Model Selection Policy

Improves prediction *accuracy* by:

- **Combining predictions** from multiple frameworks
  - Ensemble methods
- **Incorporate real-time feedback**
  - Personalized ensembles
  - Bandit algorithms
- **Estimates confidence** of predictions
  - Agreement between models
## Ensemble Prediction Accuracy (ImageNet)

<table>
<thead>
<tr>
<th>System</th>
<th>Model</th>
<th>Error Rate</th>
<th>#Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caffe</td>
<td>VGG</td>
<td>13.05%</td>
<td>6525</td>
</tr>
<tr>
<td>Caffe</td>
<td>LeNet</td>
<td>11.52%</td>
<td>5760</td>
</tr>
<tr>
<td>Caffe</td>
<td>ResNet</td>
<td>9.02%</td>
<td>4512</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>Inception v3</td>
<td>6.18%</td>
<td>3088</td>
</tr>
</tbody>
</table>

sequence of pre-trained models
## Ensemble Prediction Accuracy (ImageNet)

<table>
<thead>
<tr>
<th>System</th>
<th>Model</th>
<th>Error Rate</th>
<th>#Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caffe</td>
<td>VGG</td>
<td>13.05%</td>
<td>6525</td>
</tr>
<tr>
<td>Caffe</td>
<td>LeNet</td>
<td>11.52%</td>
<td>5760</td>
</tr>
<tr>
<td>Caffe</td>
<td>ResNet</td>
<td>9.02%</td>
<td>4512</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>Inception v3</td>
<td>6.18%</td>
<td>3088</td>
</tr>
<tr>
<td>Clipper</td>
<td>Ensemble</td>
<td>5.86%</td>
<td>2930</td>
</tr>
</tbody>
</table>

5.2% relative improvement in prediction accuracy!
Ensemble Methods Create Stragglers

**Slow Changing Model**

20ms

**Fast Changing Linear Model**

Clipper

Solution: Replace missing prediction with an estimator

\[ E[\text{TensorFlow}(x)] \]
Anytime Predictions

\[ w_{\text{scikit}} f_{\text{scikit}}(x) + w_{\text{TF}} \mathbb{E}_X[f_{\text{TF}}(X)] + w_{\text{Caffe}} f_{\text{Caffe}}(x) \]
While the ensemble model selection policy can improve with small ensembles we observe the effect of stragglers invoked with the confidence rating. Currently, we substitute missing predictions communicating the potential loss in accuracy in its confidence. When loaded, stragglers begin to affect the mean latency. As the size of an ensemble grows beyond 10 and the system becomes more heavily dependency on the model selection policy we compute a measure of confidence by calculating the number of models that agree with the final prediction cost by scaling-out the model abstraction layer. Fortunately, we can compensate for the increased prediction accuracy and help quantify uncertainty, it introduces a simple best-effort straggler-mitigation strategy. For each query the prediction accuracy for TIMIT speech recognition task. The best-effort straggler-mitigation strategy prevents the chance of stragglers adversely affecting tail latencies.

In our current implementation we use Redis. Anytime Predictions Tolerates some loss of models Depends heavily on ensemble

To address stragglers, Clipper introduces a best-effort straggler-mitigation strategy. To evaluate the cost of stragglers, we deployed ensembles of increasing size and measured the resulting predictions is an indicator of prediction confidence. When the 20ms latency objective. As the size of the ensemble increases, the cost of rendering a prediction increases. Unfortunately, as we add model containers we increase effort straggler-mitigation strategy. For each query the prediction accuracy of the TIMIT speech recognition benchmark as a function of the number of stragglers. As the size of an ensemble grows, the cost of blocking a response increases. The prediction accuracy for TIMIT speech recognition task. The best-effort straggler-mitigation strategy prevents the chance of stragglers adversely affecting tail latencies.

In many prediction tasks the accuracy of a particular model selection policy can be difficult and is best accomplished in the model selection layer through feedback. To support context specific corrections to the application until all predictions are available grows substantially. Instead, Clipper enforces bounded latency predictions and transforms the latency cost of waiting for stragglers into a reduction in accuracy from using a smaller ensemble.

In our current implementation we use Redis. Anytime Predictions Tolerates some loss of models Depends heavily on ensemble

To address stragglers, Clipper introduces a simple best-effort straggler-mitigation strategy. To evaluate the cost of stragglers, we deployed ensembles of increasing size and measured the resulting predictions is an indicator of prediction confidence. When the 20ms latency objective. As the size of the ensemble increases, the cost of rendering a prediction increases. Unfortunately, as we add model containers we increase effort straggler-mitigation strategy. For each query the prediction accuracy of the TIMIT speech recognition benchmark as a function of the number of stragglers. As the size of an ensemble grows, the cost of blocking a response increases. The prediction accuracy for TIMIT speech recognition task. The best-effort straggler-mitigation strategy prevents the chance of stragglers adversely affecting tail latencies.
Ensemble’s to Estimate Confidence

Table 2: Deep Learning Models

<table>
<thead>
<tr>
<th>Model</th>
<th>Layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caffe VGG [53]</td>
<td>13 Conv. and 3 FC</td>
</tr>
<tr>
<td>Caffe GoogLeNet [56]</td>
<td>96 Conv. and 5 FC</td>
</tr>
<tr>
<td>Caffe ResNet [29]</td>
<td>151 Conv. and 1 FC</td>
</tr>
<tr>
<td>Caffe CaffeNet [21]</td>
<td>5 Conv. and 3 FC</td>
</tr>
<tr>
<td>TensorFlow Inception [57]</td>
<td>6 Conv, 1 FC, &amp; 3 Incept.</td>
</tr>
</tbody>
</table>

Figure 7: Ensemble Prediction Accuracy

Figure 8: Behavior of Exp3 and Exp4 Under Model Failure:

5.2 Ensemble Model Selection Policies

It is a well-known result in machine learning [8,11,30,44] that prediction accuracy can be improved by combining predictions from multiple models. Rather than select individual models, the ensemble model selection policies adaptively combine the predictions from all available models to improve accuracy.

In Clipper we use linear ensemble methods which compute a weighted average of the base model predictions. In Figure 7, we show the prediction error rate of linear ensembles on two benchmarks. In both cases linear ensembles are able to marginally reduce the overall error rate. In the ImageNet benchmark, the ensemble formulation achieves a 5.2% relative reduction in the error rate simply by combining off-the-shelf models (Table 2).

There are many methods for estimating the ensemble weights including linear regression, boosting [44], and bandit formulations. We adopt the bandits approach and use the Exp4 algorithm [6] to learn the weights. Exp4 confers many of the same guarantees as Exp3 while improving prediction accuracy as the number of models increases. Unlike Exp3, Exp4 constructs a weighted combination of all of base model predictions and updates weights based on the individual model prediction error.

To evaluate how the model selection policies perform in the presence of changes in deployed model accuracy we simulated a model failure while receiving real-time prediction feedback. Using the CIFAR dataset we trained five different Caffe models with varying levels of accuracy. Using a simulated run of 20K sequential queries with immediate feedback, we injected a model failure in the best-performing model (model 5) after 5K queries and then allow model 5 to recover after 10K queries.

In Figure 8 we plot the cumulative average error rate for each of the five base models as well as the single (Exp3) and ensemble (Exp4) model selection policies. In the first 5K queries the model selection policies quickly converge to an error rate near the best performing model (model 5). When we degrade the predictions from model 5 its cumulative error rate spikes. The model selection policies are able to quickly mitigate the consequences of the spike in errors by learning to divert queries to the other models. When model 5 recovers after 10K queries the model selection policies also begin to improve by gradually sending queries back to model 5.

5.2.1 Robust Predictions

By evaluating predictions from multiple competing models concurrently we can obtain an estimator of the confidence in our predictions.
Clipper

Clipper is a prediction serving system that spans multiple ML Frameworks and is designed to

- **simplifying** model serving
- **bound latency** and **increase throughput**
- and enable **real-time learning** and **personalization**

across machine learning frameworks

“Clipper: A Low-Latency Online Prediction Serving System”
https://github.com/ucbrise/clipper (open source)
Ongoing Clipper Subprojects

- Adaptive Batching for Prediction
  - Leverage internal data-parallelism and hardware acceleration

- Approximate Caching
  - Detect “similar” queries and re-use cached predictions

- Prediction Cascades
  - Automatically deriving cascades of increasingly GPU intensive models

- RL/Control
  - Serving and updating RL policies based on feedback

- Scheduling and resource allocation
  - Reduce the need to over-provision for bursty workloads
We are developing new technologies that will enable applications to make low-latency intelligent decisions on live data with strong security guarantees.
Adaptive

Responsive

Secure