Hardware Opportunities in the Machine Learning Lifecycle

Joseph E. Gonzalez
Co-director of the RISE Lab
jegonzal@cs.berkeley.edu

Get the latest slides
and links to literature

https://tinyurl.com/isscc-lifecycle

About Me

- Co-director of the RISE Lab
- Co-founder of Turi Inc.
- Member of the Apache Spark PMC
- Research
  - Artificial Intelligence
  - Data Science
  - Distributed Data Systems
  - Graph Processing Systems
- I don't study processor **architecture**
  - But I probably should …

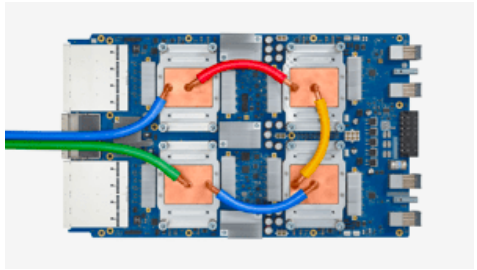
Outline

- History and the **Co-evolution** of Hardware and AI
  - The **Feedback Cycle** driving the 3rd wave of AI
- Machine Learning is **not a single workload**
  - Stages of the **Machine Learning Lifecycle**
- **Security** and machine learning

Along the way, I will talk about some of the research in my group addressing interesting aspects of the lifecycle.

Hardware and the History of AI

- **1950 to 1974:** *Birth of AI*
  - 1951 Marvin Minsky builds first **neural network hardware** (SNARC)
- **1974 to 1980:** *First AI Winter*
  - Limited **processing power** and **data**
- **1980 to 1987:** *Second Wave of AI*
  - XCON (AI for **hardware configuration**) for DEC → boom in AI hardware companies
- **1987 to 1993:** *Second AI Winter*
  - **Brittle** AI and the collapse of the **AI Hardware Market**
- **1993 to 2011:** *AI → Machine Learning*
  - *Confluence of ideas + **Compute** + **Big Data** → AI starts to really work*
- **2011 to 2019:** *Third Wave (Deep Learning)*
  - ***Compute** + **data** + **abstractions** → feedback cycle*



Compute

Feedback Cycle

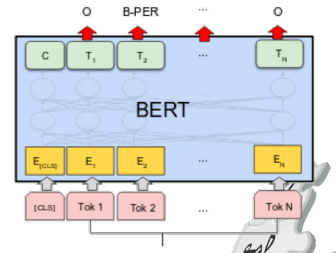
Abstractions

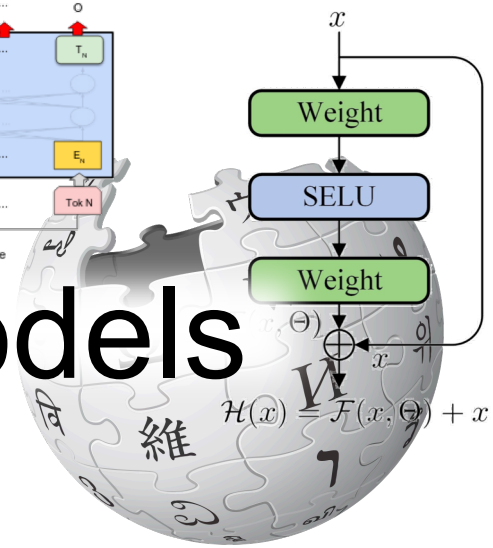
PyTorch

mxnet

TensorFlow

Data & Models





WIKIPEDIA
The Free Encyclopedia

Abstractions are Enabling Innovation

- Much of machine learning before 2010
  - Research focused on machine learning **algorithms**
  - Programs written using **high-level imperative languages**
    - Matlab/R/C++/Java
  - **Big abstractions:** linear algebra, map-reduce, graph systems

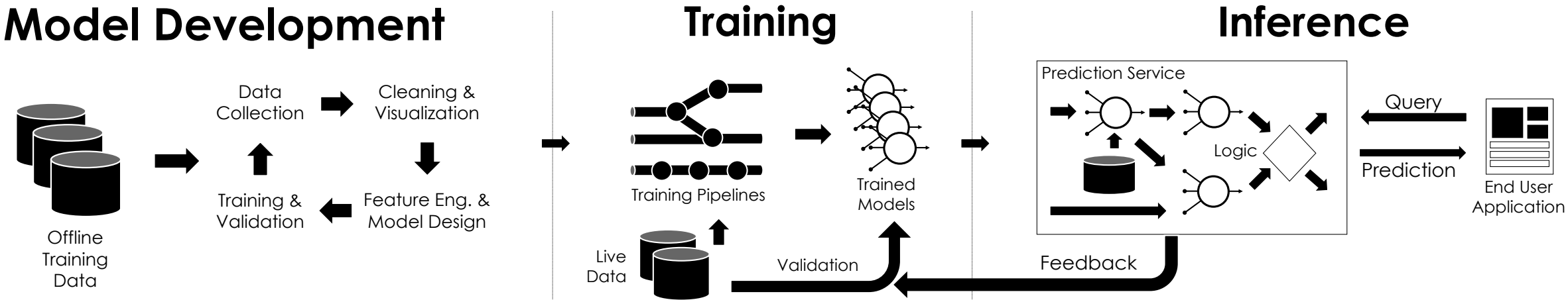
- Today:
  - Research focused on **model design**
  - **Models** written in high-level **DSLs**
    - TensorFlow/Pytorch
  - Big abstractions: **tensor operations**, **loss minimization,** linear algebra, …

- Models written in **TensorFlow** can now run on **hardware** that didn't exist when the models were created.

How do we make hardware for Machine learning?

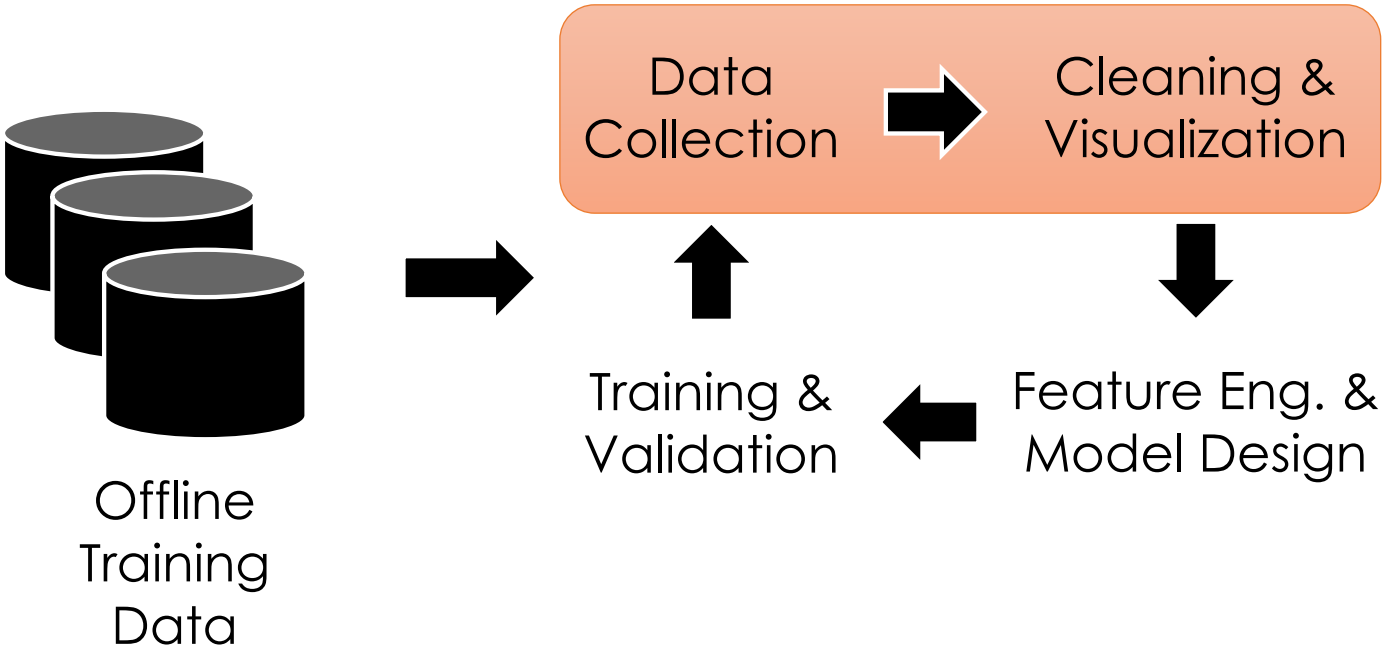
Machine learning is not a single application.

Machine learning is
multiple applications** with **different requirements.



Machine Learning Lifecycle

Model Development



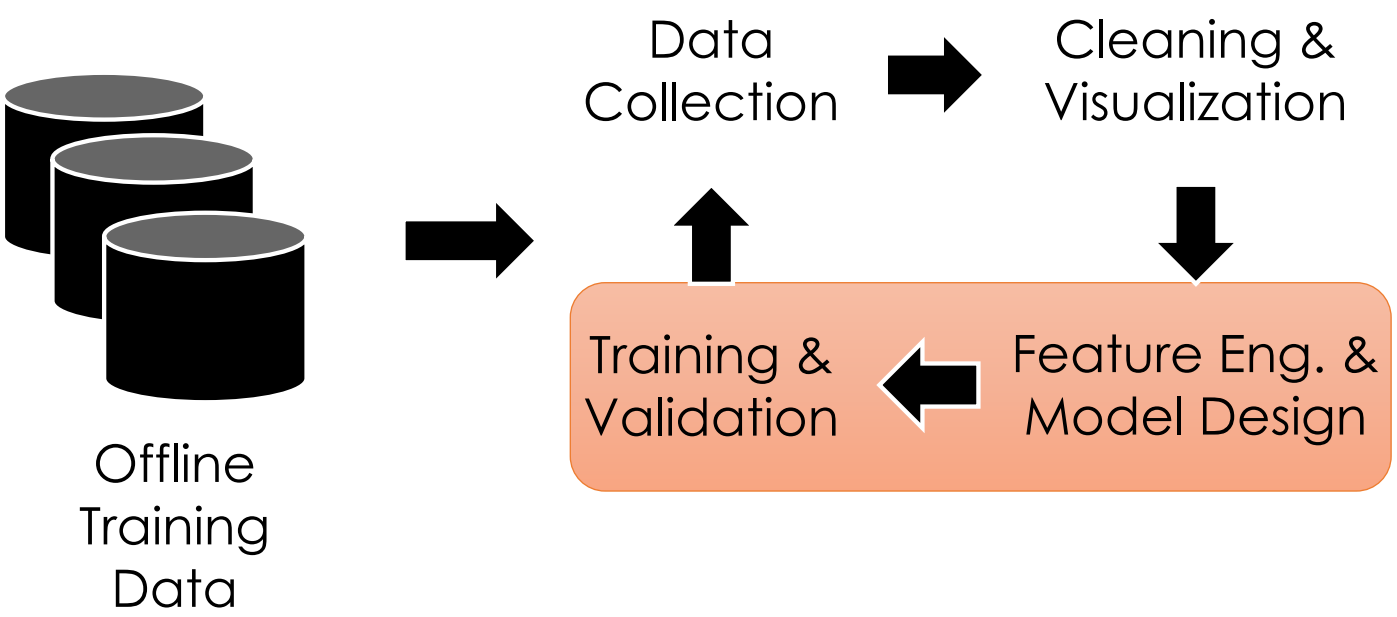
Identifying potential sources of data

Joining data from multiple sources

Addressing **missing values** and **outliers**

Plotting trends to identify **anomalies**

Model Development



Building informative **features functions**

Designing new **model architectures**

Tuning training algos.

Validating prediction accuracy

Model Development *Frameworks*

Offline Training Data

Data Collection → Cleaning & Visualization → Feature Eng. & Model Design → Training & Validation → Data Collection

scikit learn

TensorFlow

R

PYTORCH

Keras

jupyter

mxnet

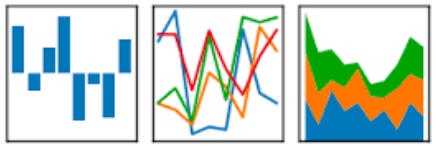
dmlc XGBoost

matplotlib

Caffe2

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



NumPy

APACHE Spark™

DASK

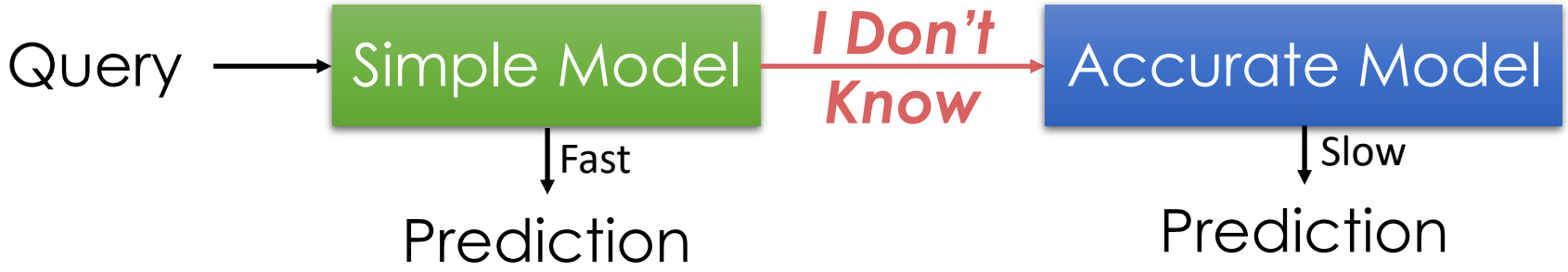
HIVE

Model Development → Hardware

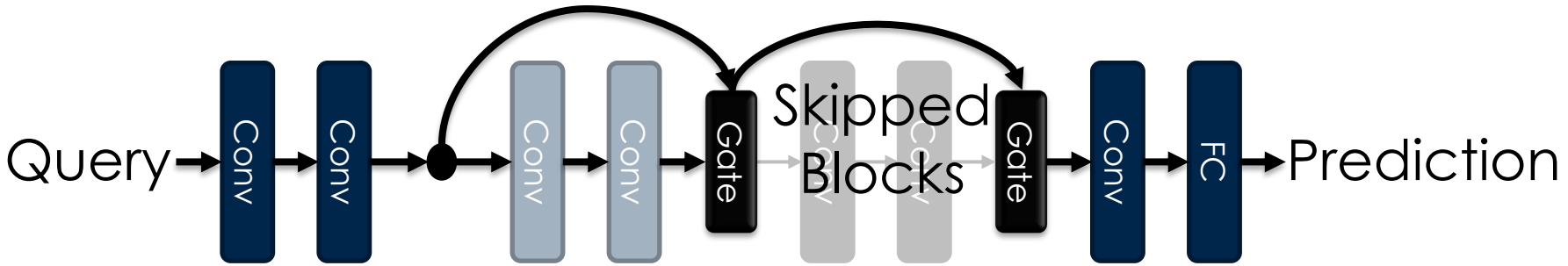
- Need to test **multiple designs and hyperparameters** quickly
  - May be better to run many parallel experiments than one experiment faster
- **Debug heavy** → sources of error → data, hyperparams., & model
  - **System should not be a source of error**
  - **Avoid** cutting corners (e.g., quantization, async) for increased performance
    - Unless you can make a case for stable convergence …
- **Data preparation** is often a bottleneck
  - Opportunity for **data tooling**
  - Accelerate data transformation and augmentation
- Emerging Trends
  - **Attention Models** and **Graph Neural Networks**: reduced locality, sparsity
  - **Dynamic Networks**: gating, cascades, mixtures, …
  - Increased emphasis on **DNN features** and **Fine-Tuning**
    - Reuse of common architectures and weights

Dynamic Networks for **fast** and **accurate** inference

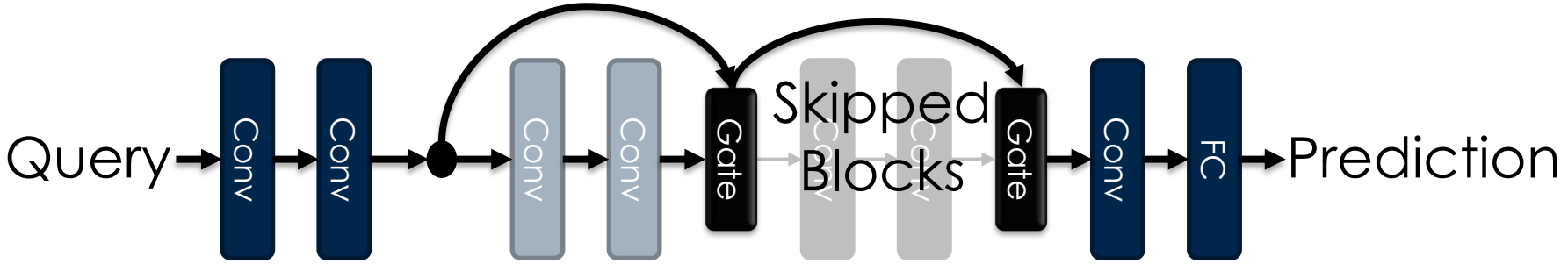
IDK Cascades: Using the fastest model possible [UAI'18]



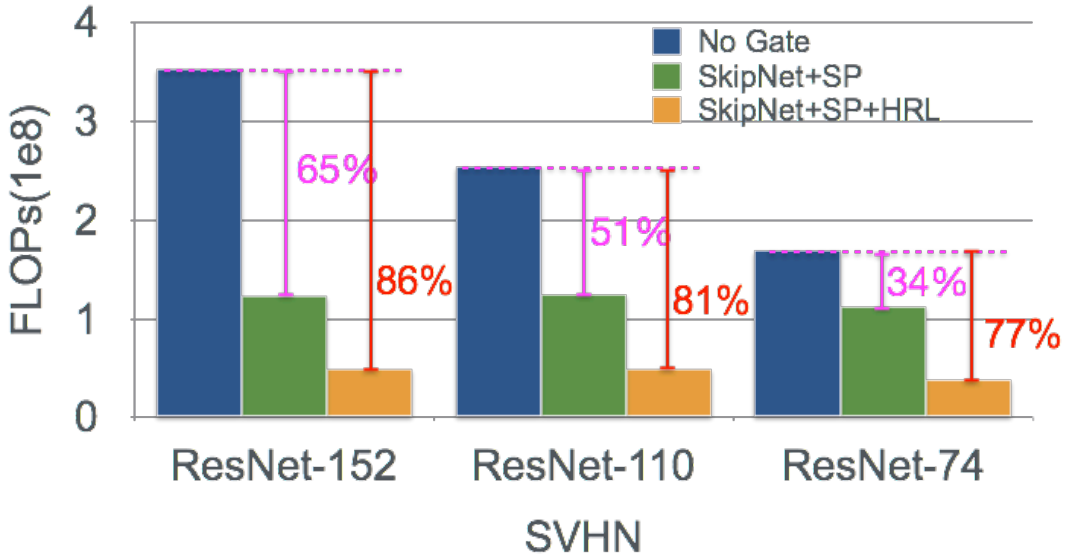
SkipNet: dynamic execution within a model [ECCV'18]



SkipNet: dynamic execution within a model [ECCV'18]



Large Reductions in FLOPS



Skip more layers on clear images

Easy Images
Skip **Many** Layers

Hard Images
Skip **Few** Layers

Task Aware Feature Embeddings [CVPR'19]

Feature Network

FF Net

FF Net

FF Net

Baby

Params

Params

Params

X

X

X

Emb. Networ k

FC Layer

FC Layer

FC Layer

Task Aware Meta-Learner

More **accurate** and **efficient** than existing dynamic pruning networks

Task Aware Feature Embeddings [CVPR'19]

Feature Network

FF Net

FF Net

FF Net

Yes

Params

Params

Params

X

X

X

FC Layer

FC Layer

FC Layer

Task Aware Meta-Learner

Task Description

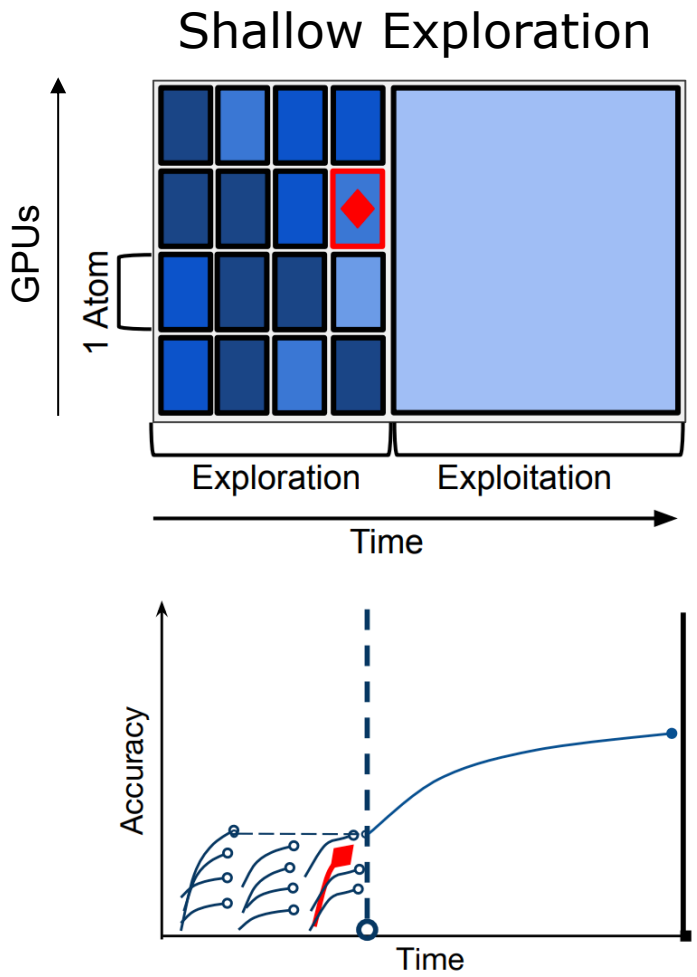
"Smiling Baby"

Emb. Network

4 - 15% improvement on **attribute-object tasks**

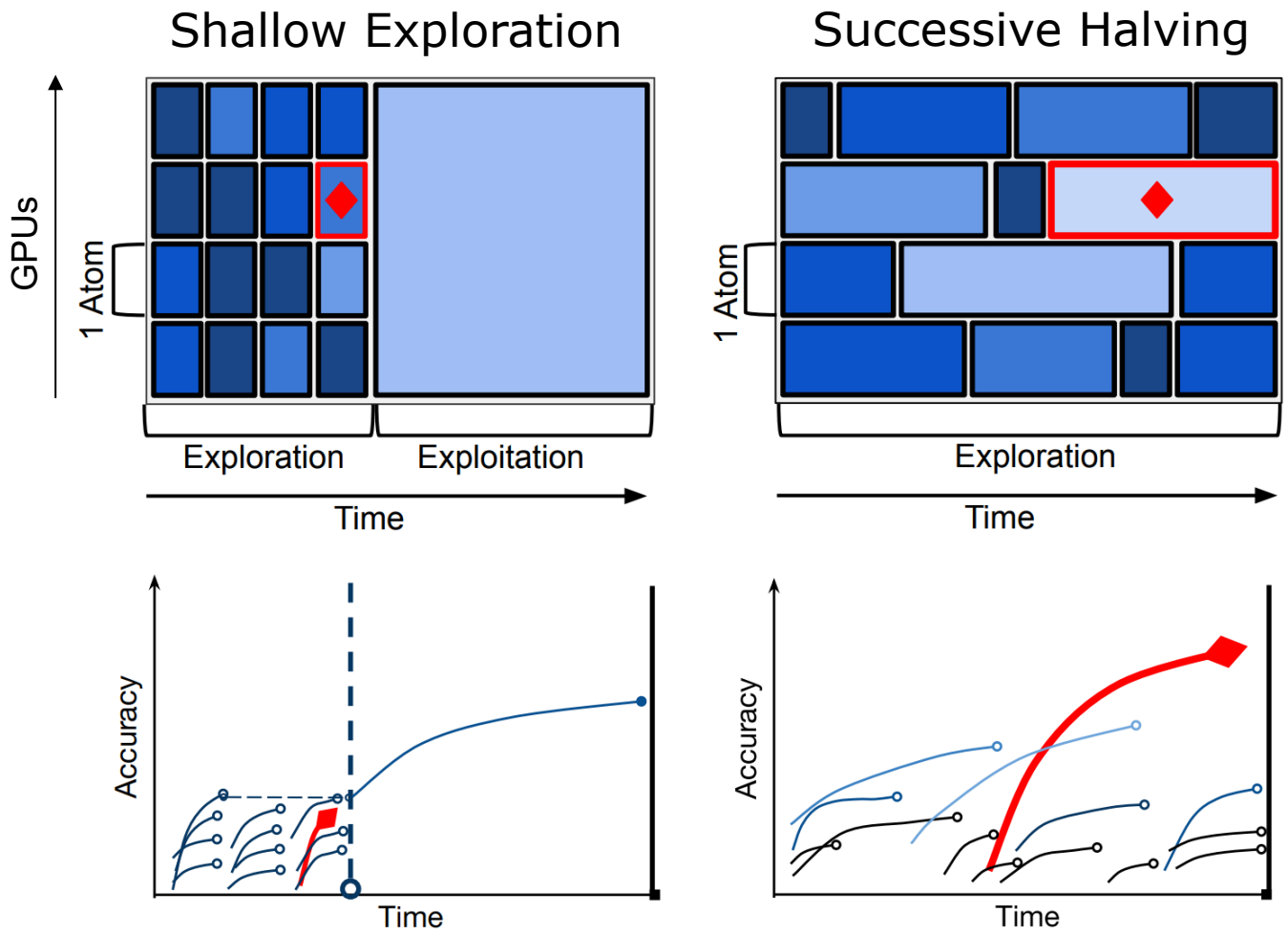
HyperSched [SOCC'19]

Dynamically allocating **parallel resources** to **parallel experiments**.



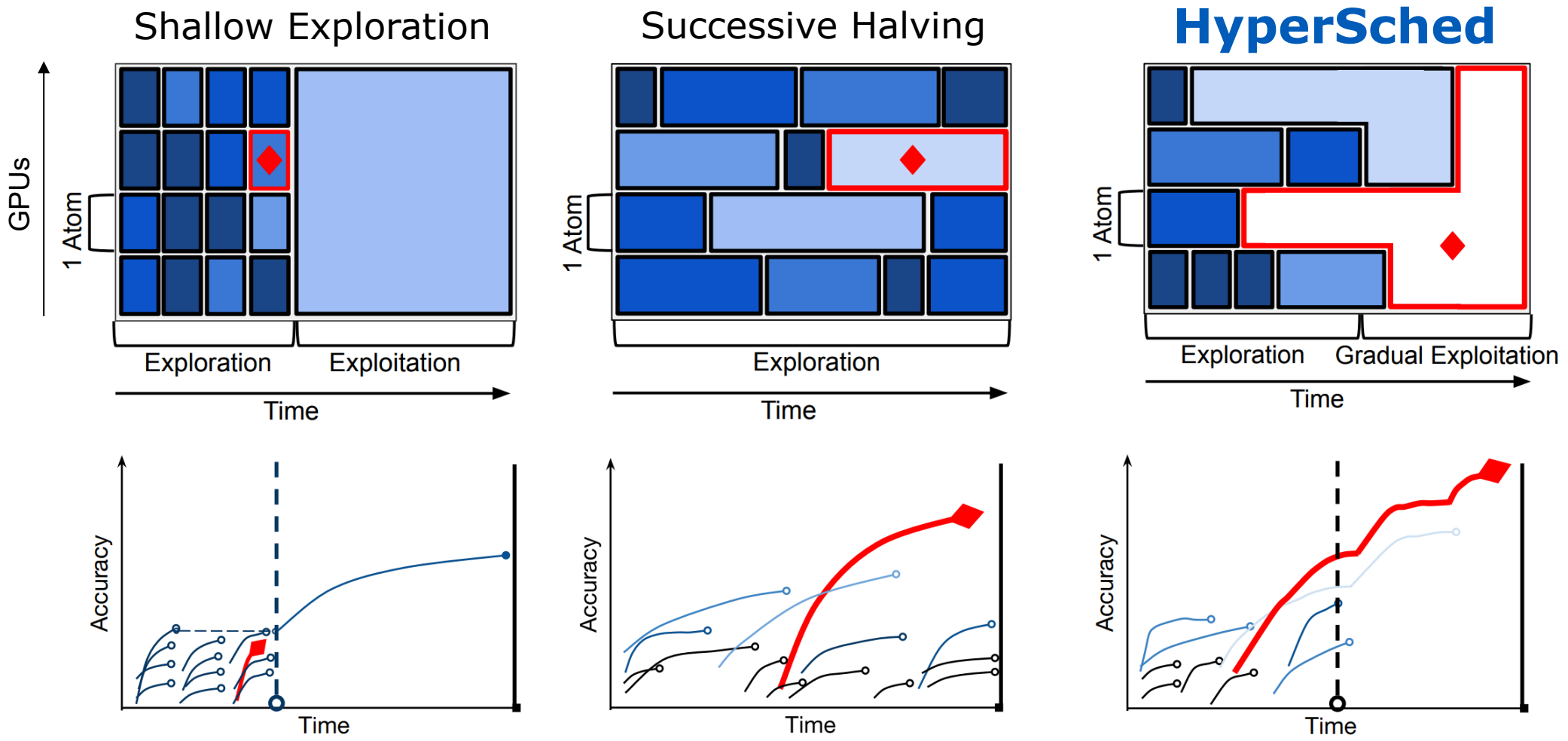
HyperSched [SOCC'19]

Dynamically allocating **parallel resources** to **parallel experiments**.

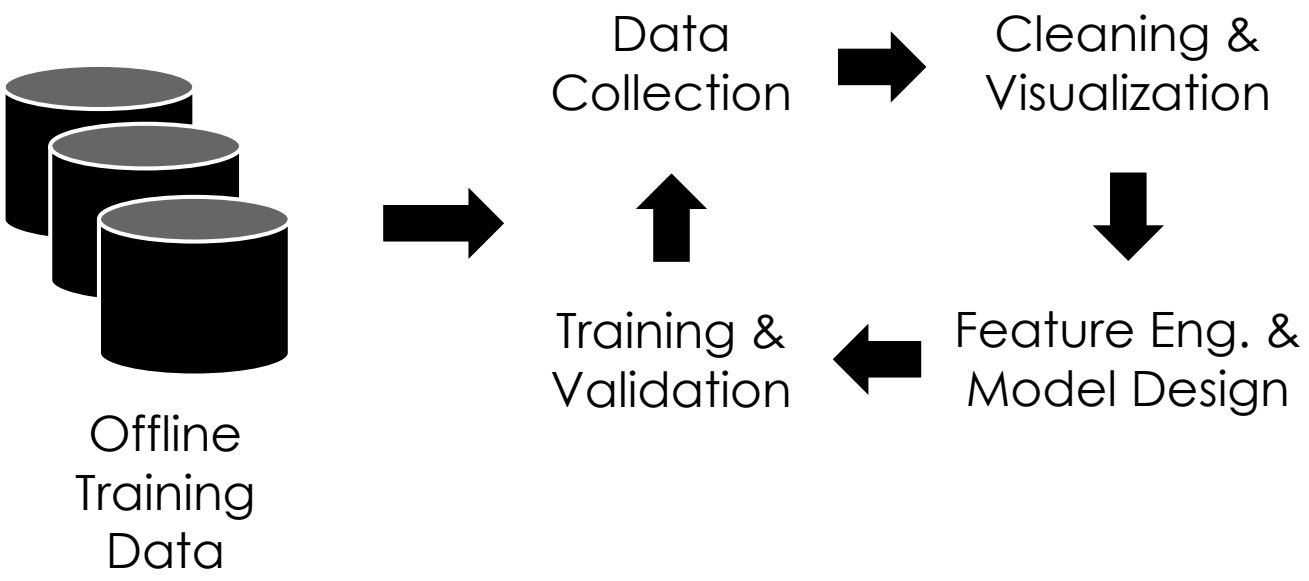


HyperSched [SOCC'19]

Dynamically allocating **parallel resources** to **parallel experiments**.



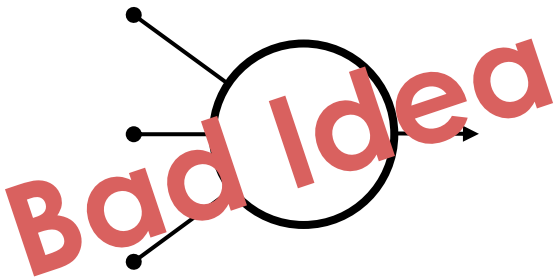
What is the output of Model Development



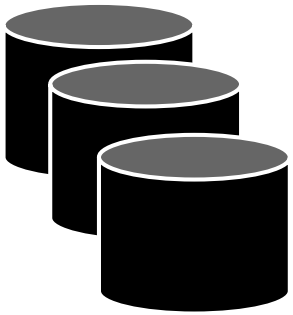
Reports & Dashboards

(insights …)

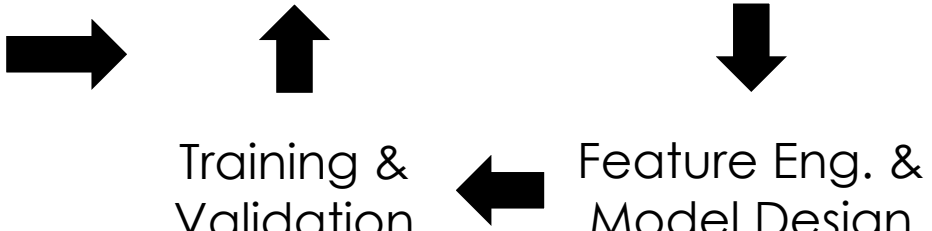
Trained Model



What is the output of Model Development



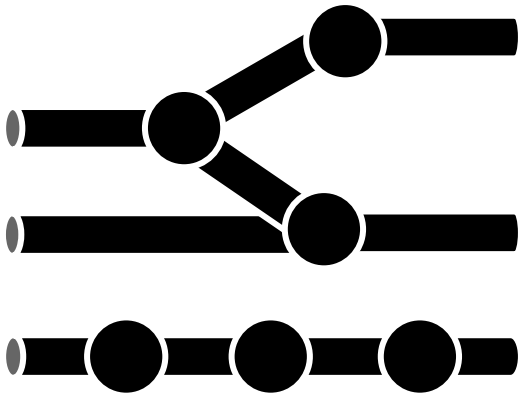
Offline Training Data



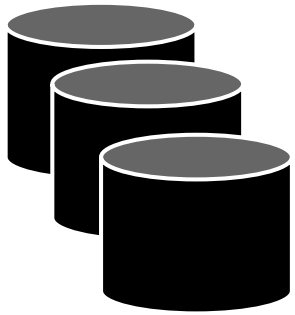
Reports & Dashboards

(insights …)

Training Pipelines



Model Development



Offline Training Data

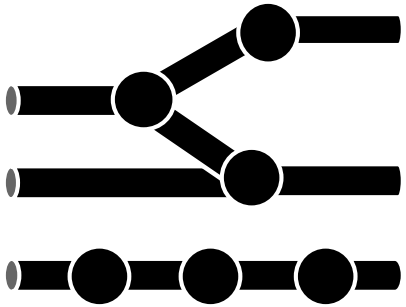
Data Collection

Cleaning & Visualization

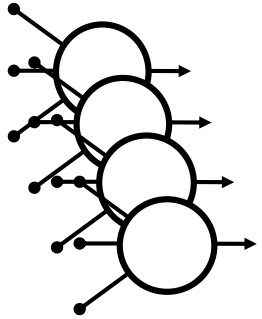
Feature Eng. & Model Design

Training & Validation

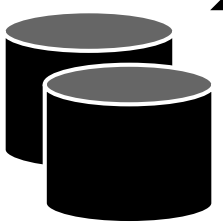
Training



Training Pipelines



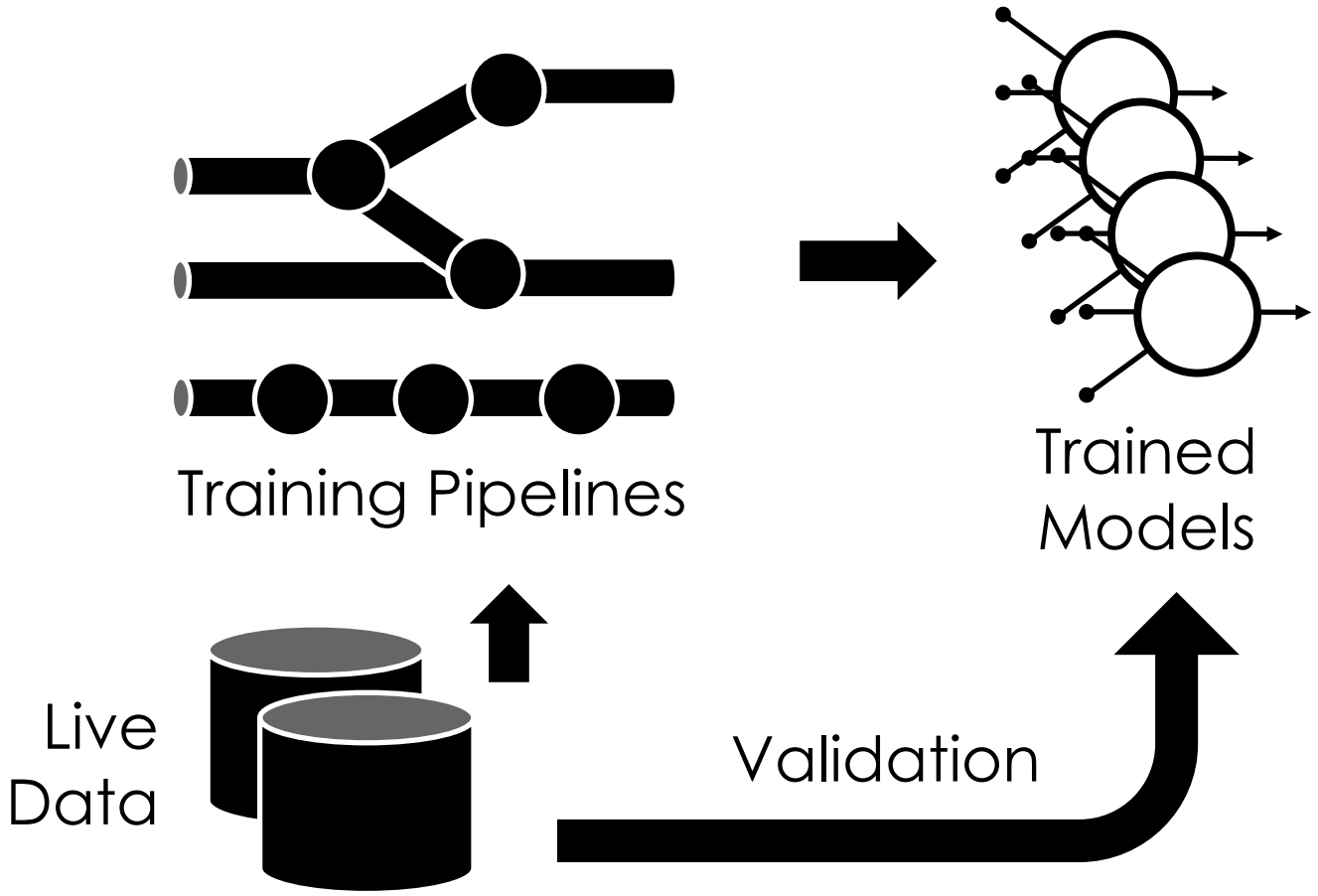
Trained Models



Live Data

Validation

Training



Training models **at scale** on **live data**

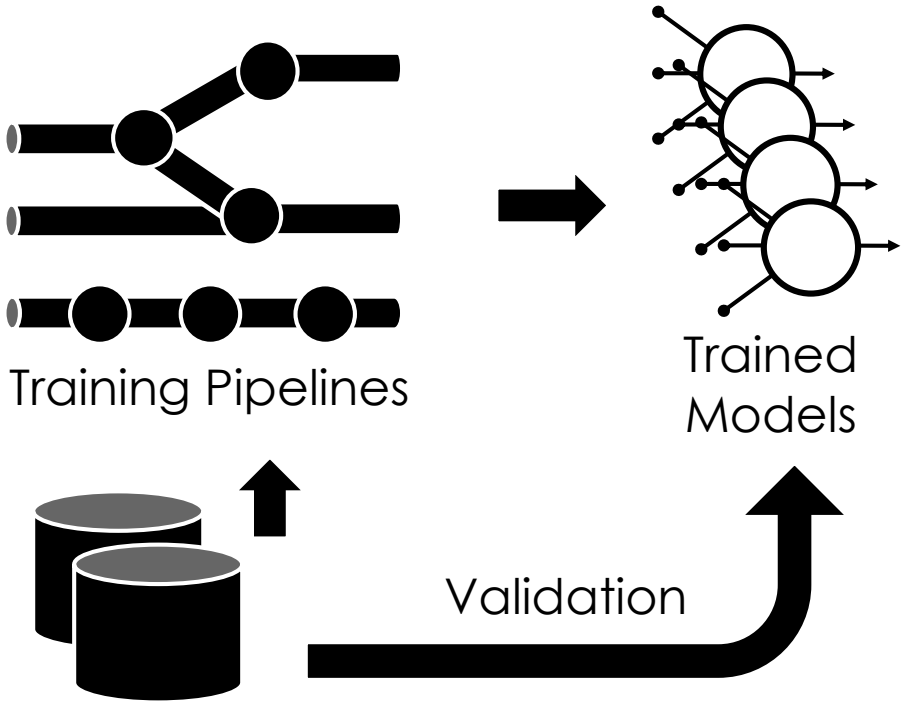
Retraining on new data

Automatically **validate** prediction accuracy

Manage model **versioning**

Requires **minimal expertise** in machine learning

Training *Systems*



Workflow Management:

Apache Airflow

Luigi

Azkaban
Open-source Workflow Manager

OOZIE

Scalable Training:

PYTORCH

mxnet

TensorFlow

HOROVOD

APACHE Spark™

dmlc XGBoost

Model Training → Hardware

- Fewer models to train → need **distributed training** of **individual models**
  - Often train with **more data**
- **Larger models** and **mini-batch sizes**
  - Need larger on-device memory
  - Counter trends → reversable networks, **optimal checkpointing**, …
- Models and hyperparameters are vetted → focus on **system optimizations**
  - Can **tolerate** some **system error** (quantization and async.)
  - Need adequate stability to meet deadlines
- **Data preparation** is still potentially an issue (as with model dev.)
- Need to deal with **composition** of multiple models

Do Bigger Models Train Faster?

(Preliminary unpublished work.)

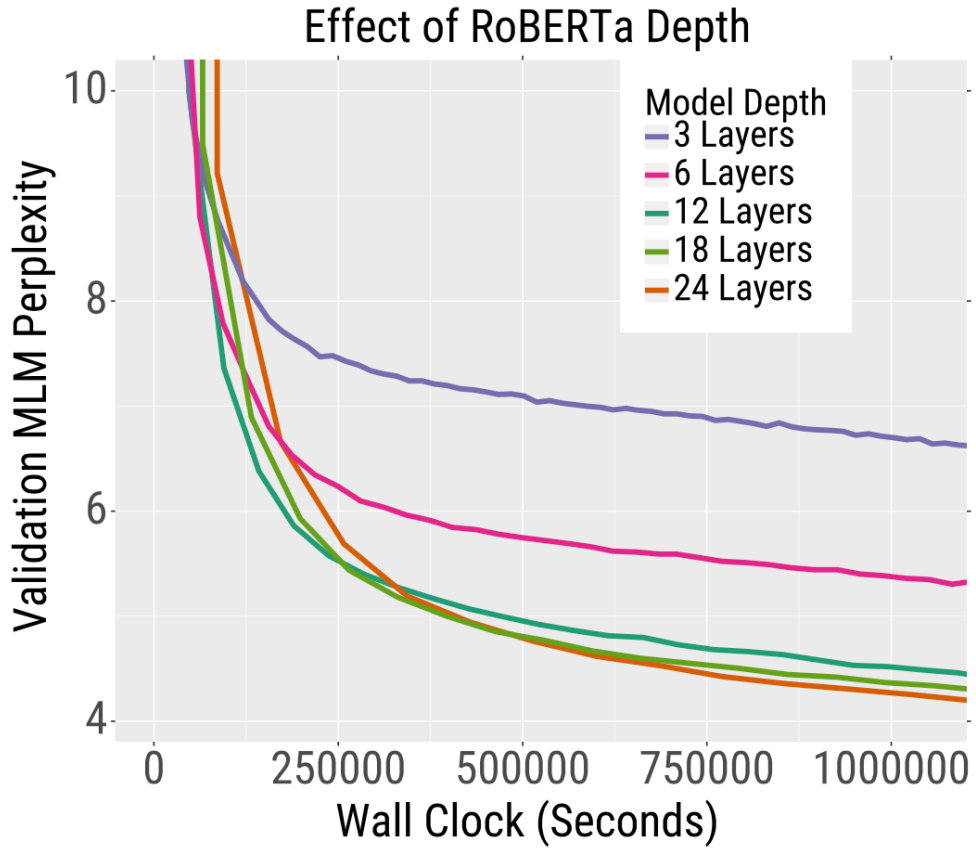
- Studying **pre-training** of large **Transformer Models** for NLP task (e.g., BERT)

Do Bigger Models Train Faster?

(Preliminary unpublished work.)

- Studying **pre-training** of large **Transformer Models** for NLP task (e.g., BERT)

Deeper Models **Reduce Error Faster**

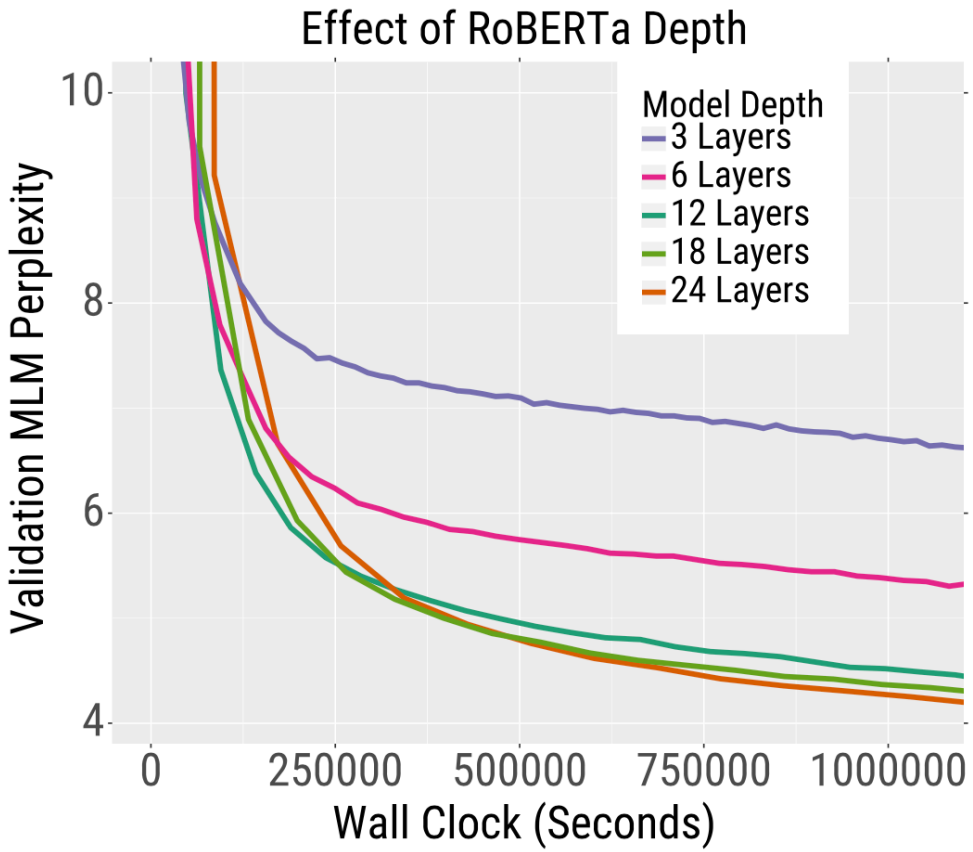


Do Bigger Models Train Faster?

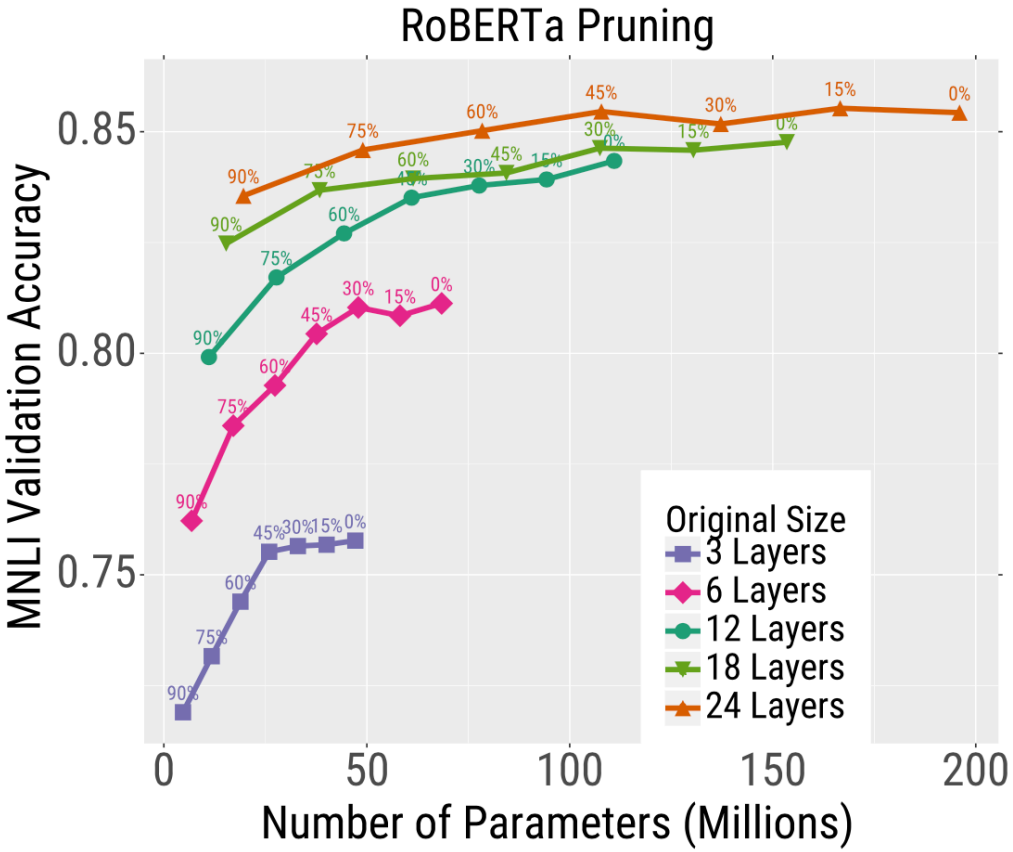
(Preliminary unpublished work.)

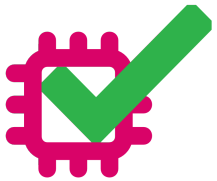
- Studying **pre-training** of large **Transformer Models** for NLP task (e.g., BERT)

Deeper Models **Reduce Error Faster**



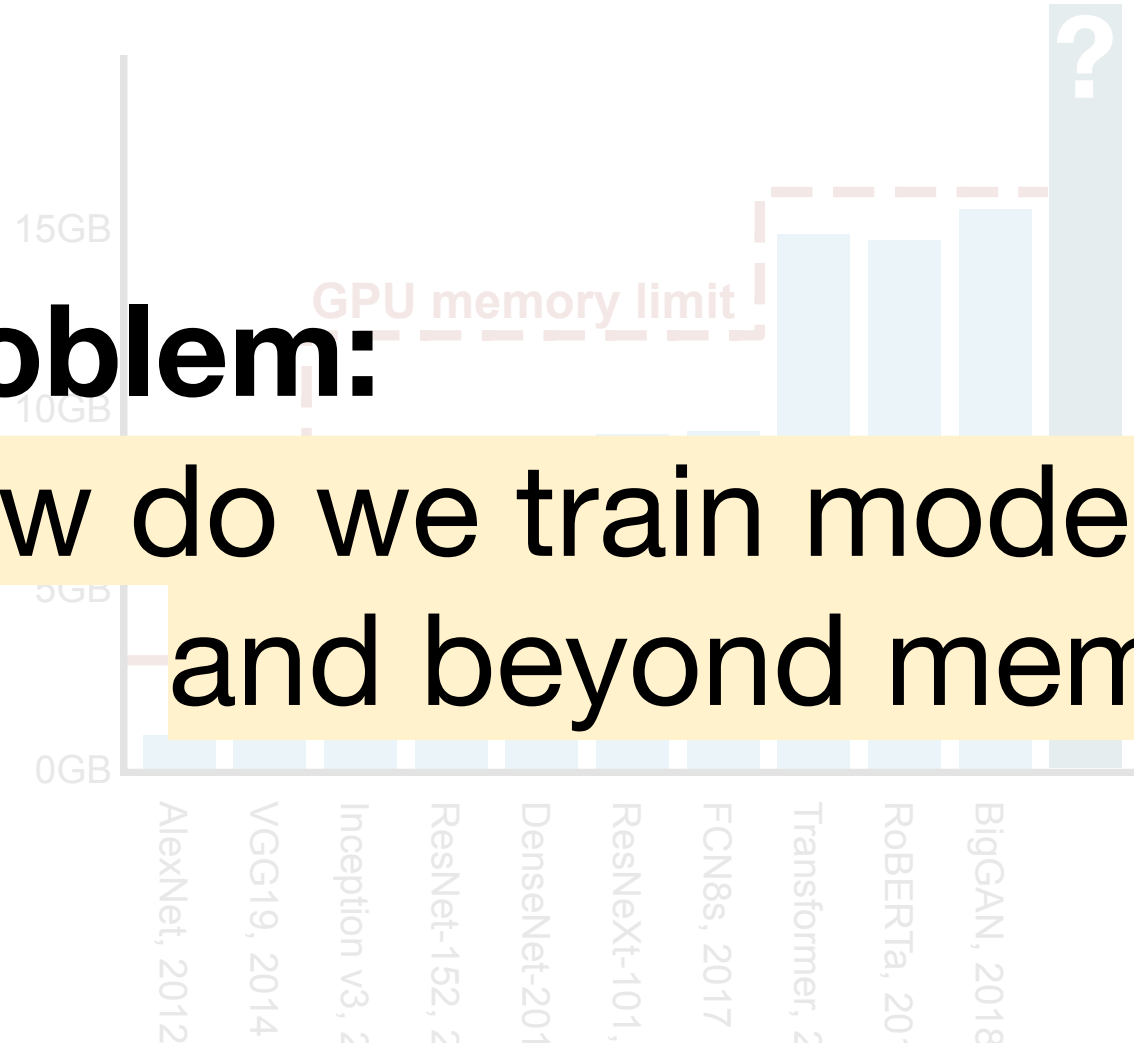
More Resilient to **Lossy Compression**





CheckMate [MLSys 2020]

Scaling deep learning training beyond the GPU memory wall.



At a per-GPU level, recent state-of-the-art models have hit a **memory capacity wall**.

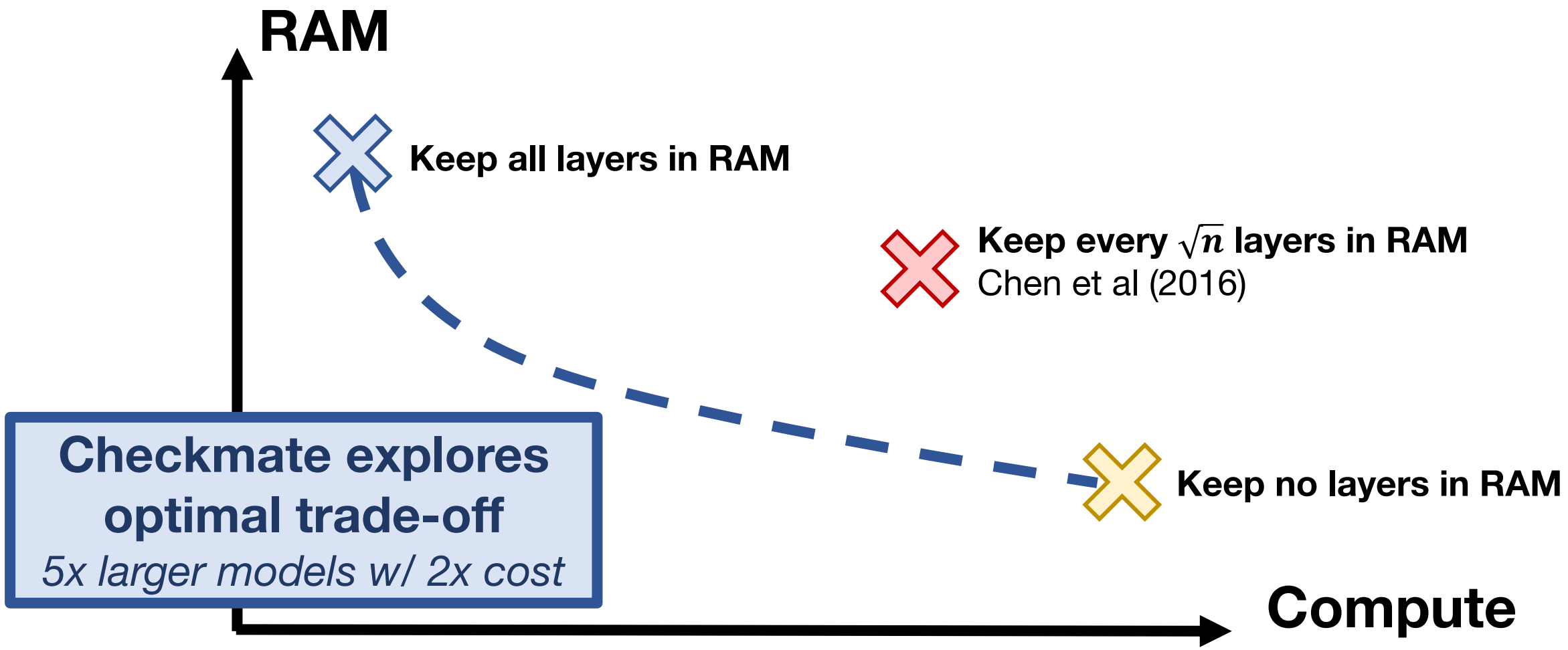
memory is slowing progress in new deep learning models!

Problem:

How do we train models both efficiently and beyond memory limits?

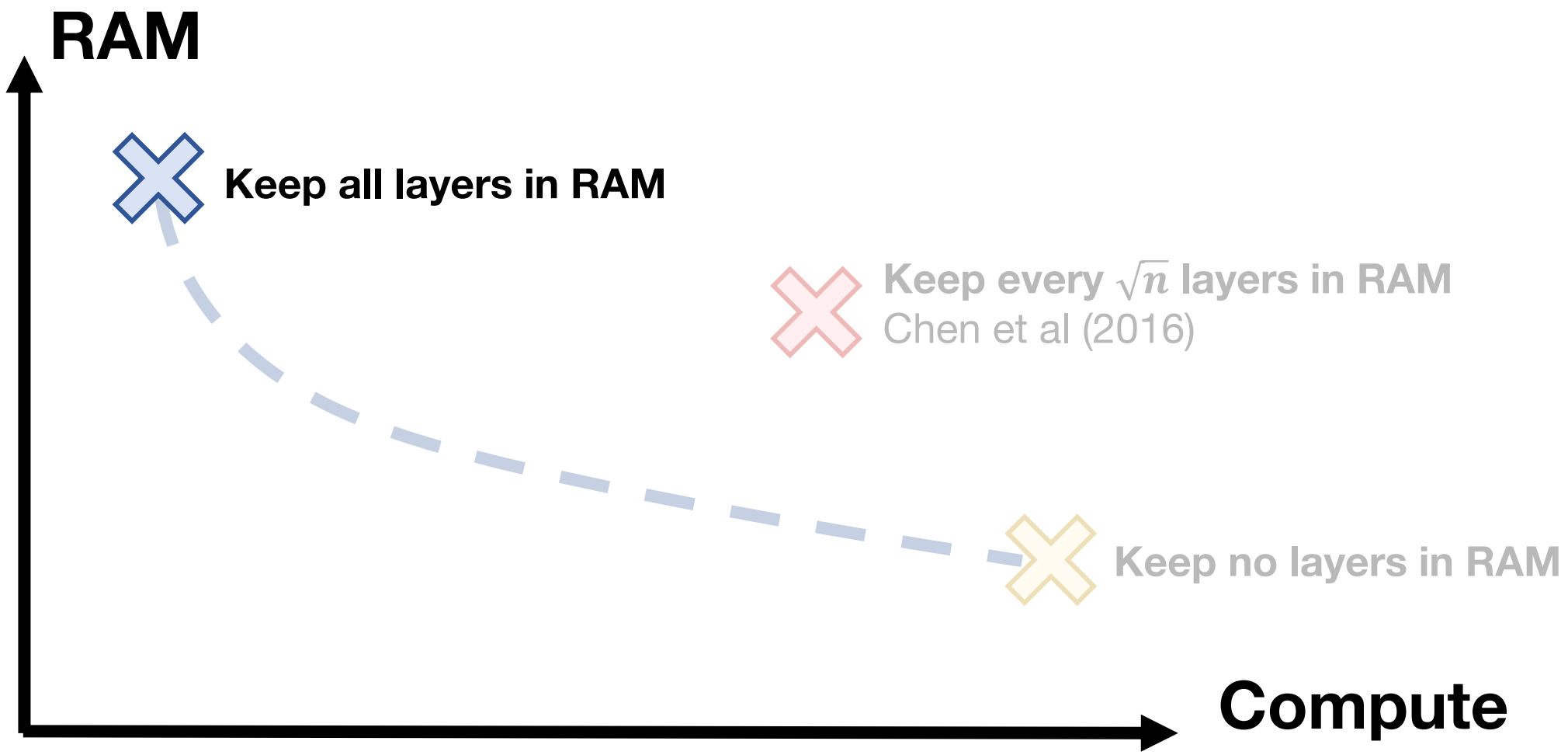
checkmateai.github.io

Efficiently **trade-off RAM** and **Compute**



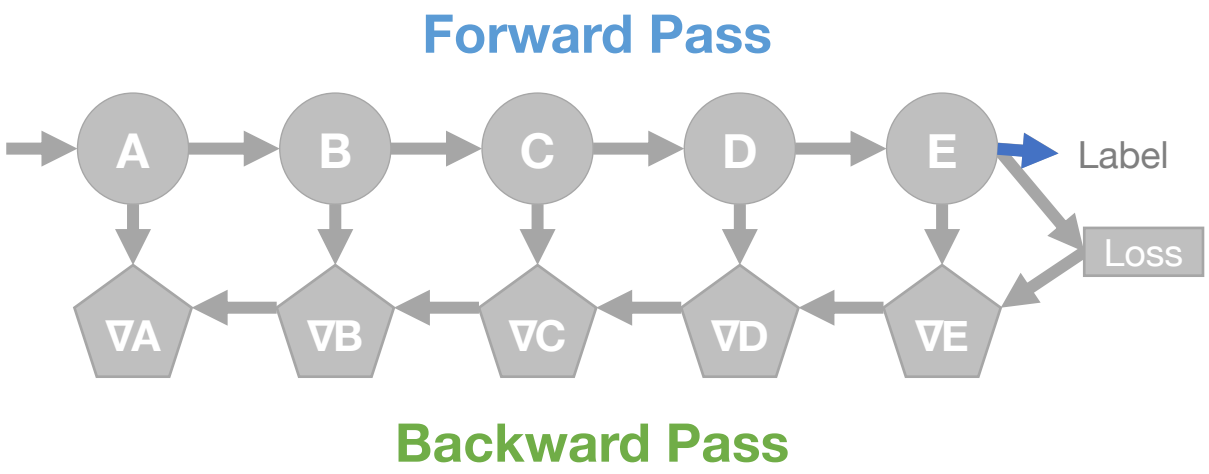
RAM-hungry backpropagation policy

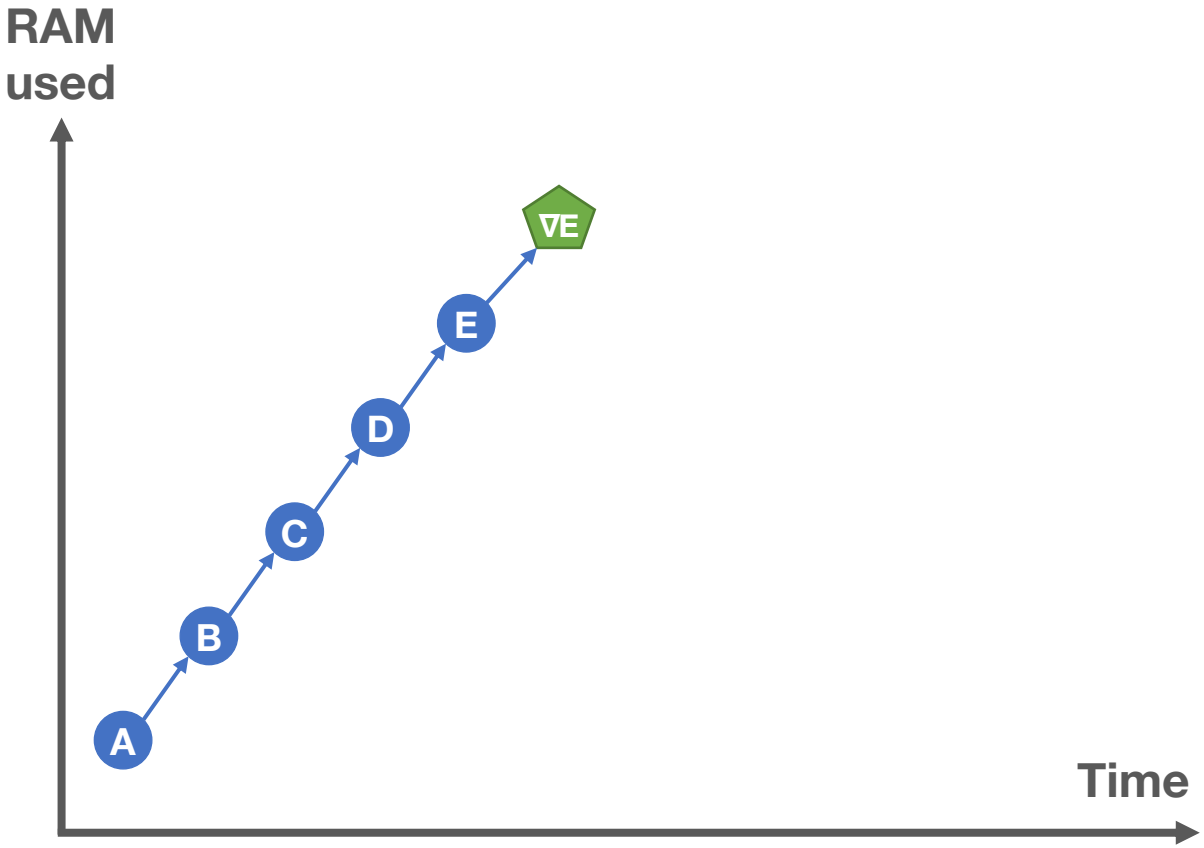
Keep all layers in RAM



RAM-hungry backpropagation policy

Keep all layers in RAM

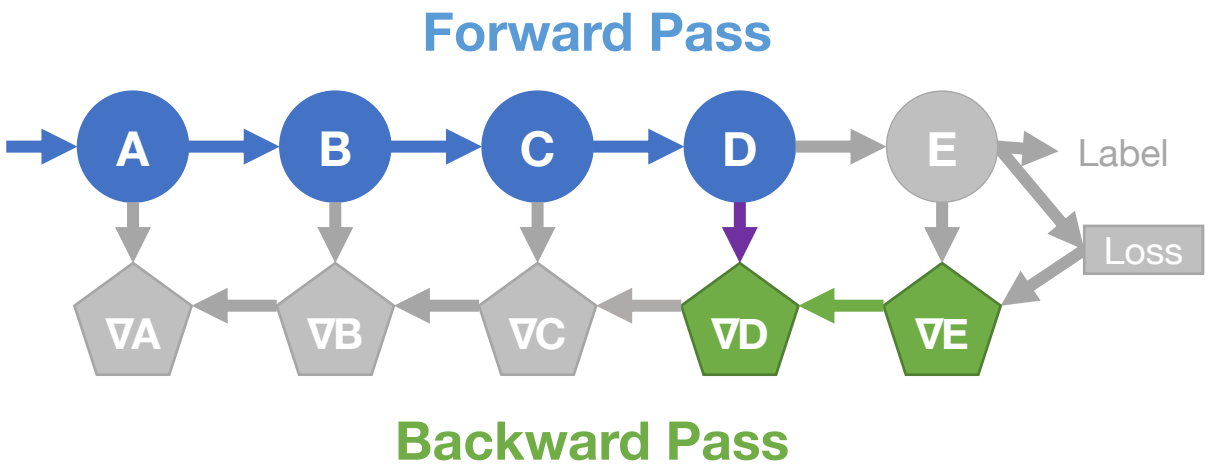


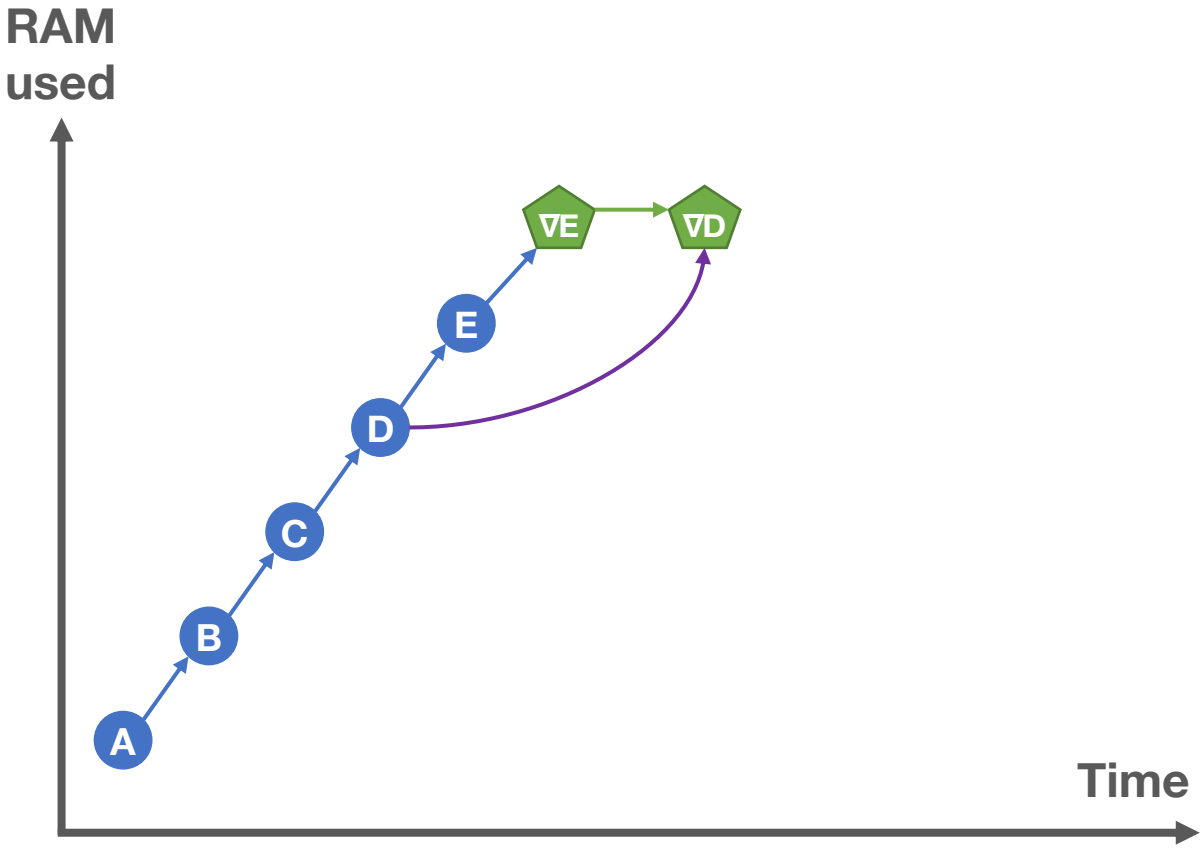


checkmateai.github.io

RAM-hungry backpropagation policy

Keep all layers in RAM

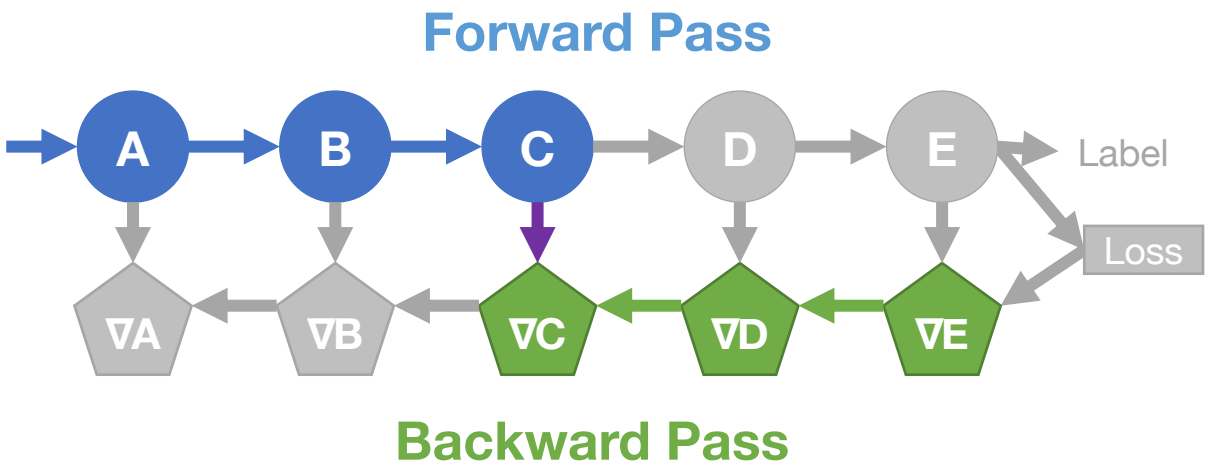


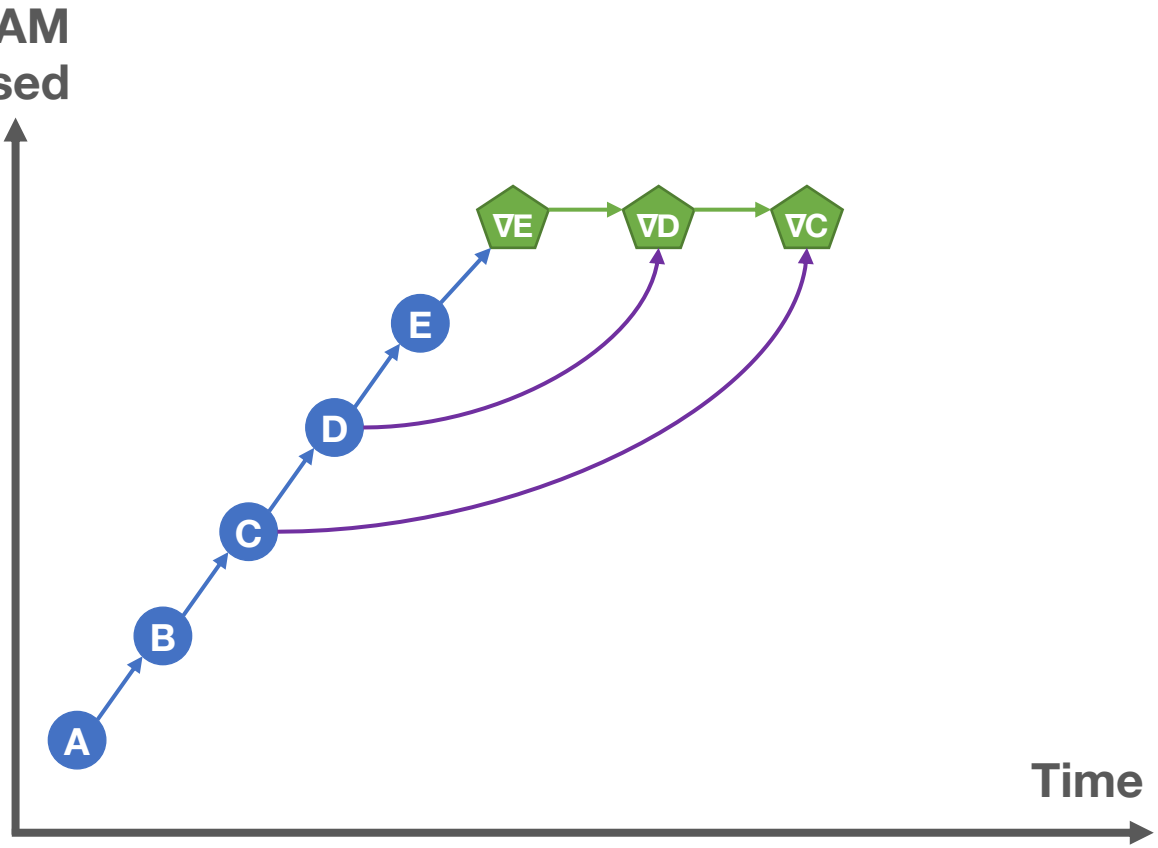


checkmateai.github.io

RAM-hungry backpropagation policy

Keep all layers in RAM

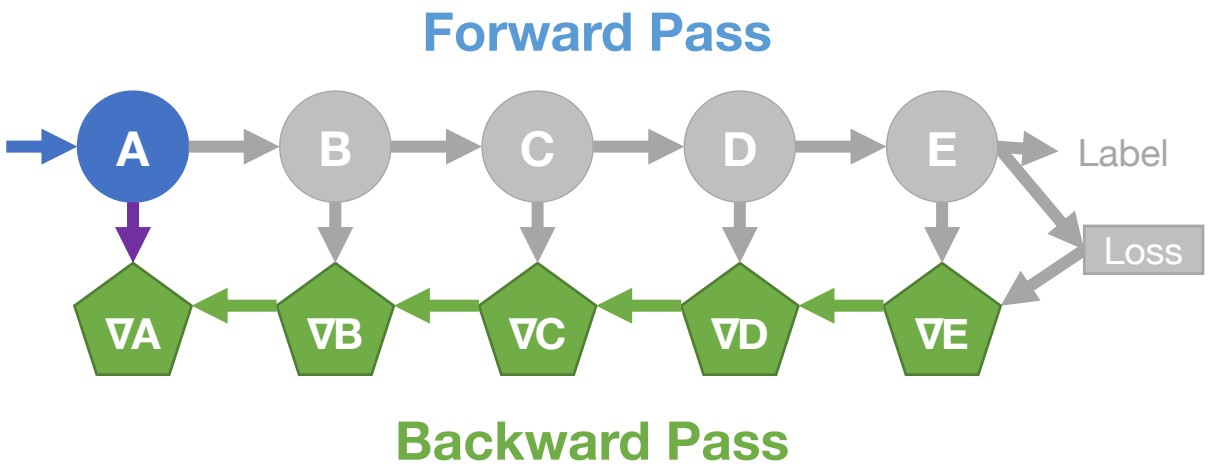


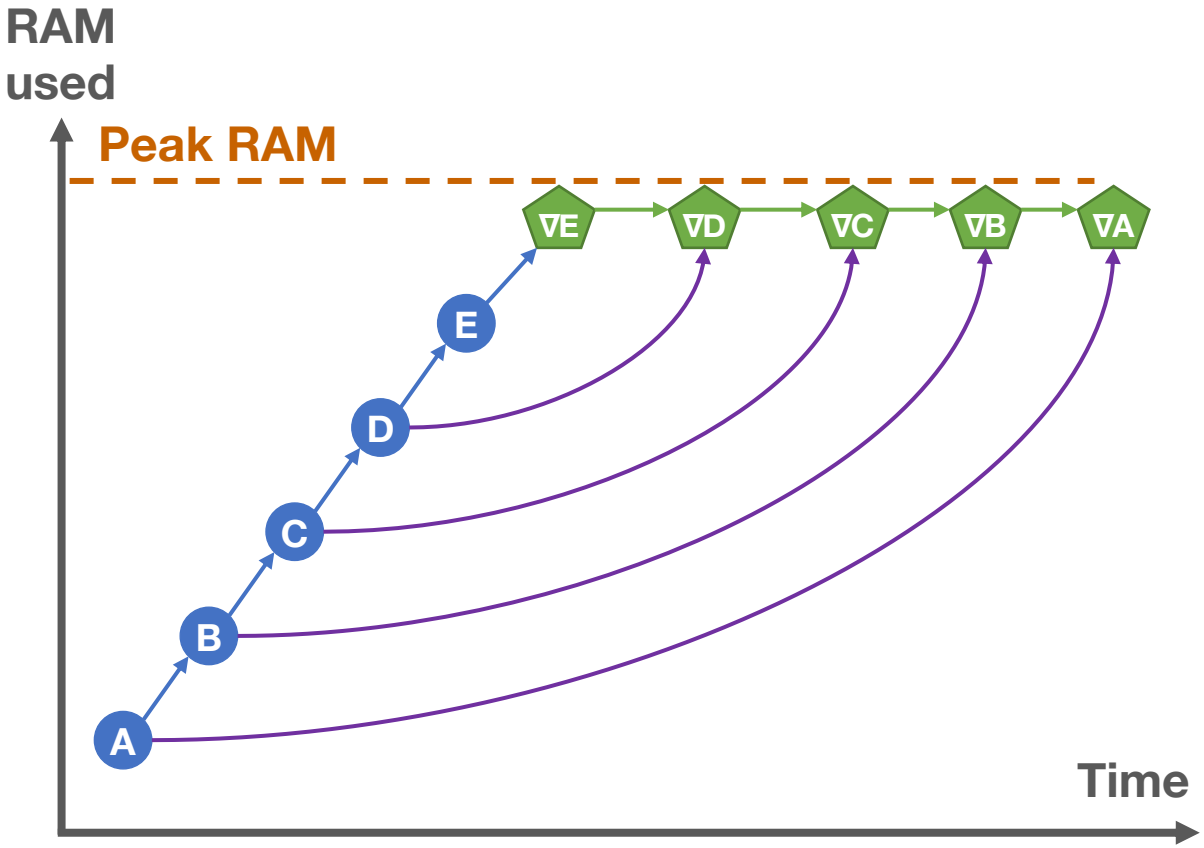


checkmateai.github.io

RAM-hungry backpropagation policy

Keep all layers in RAM

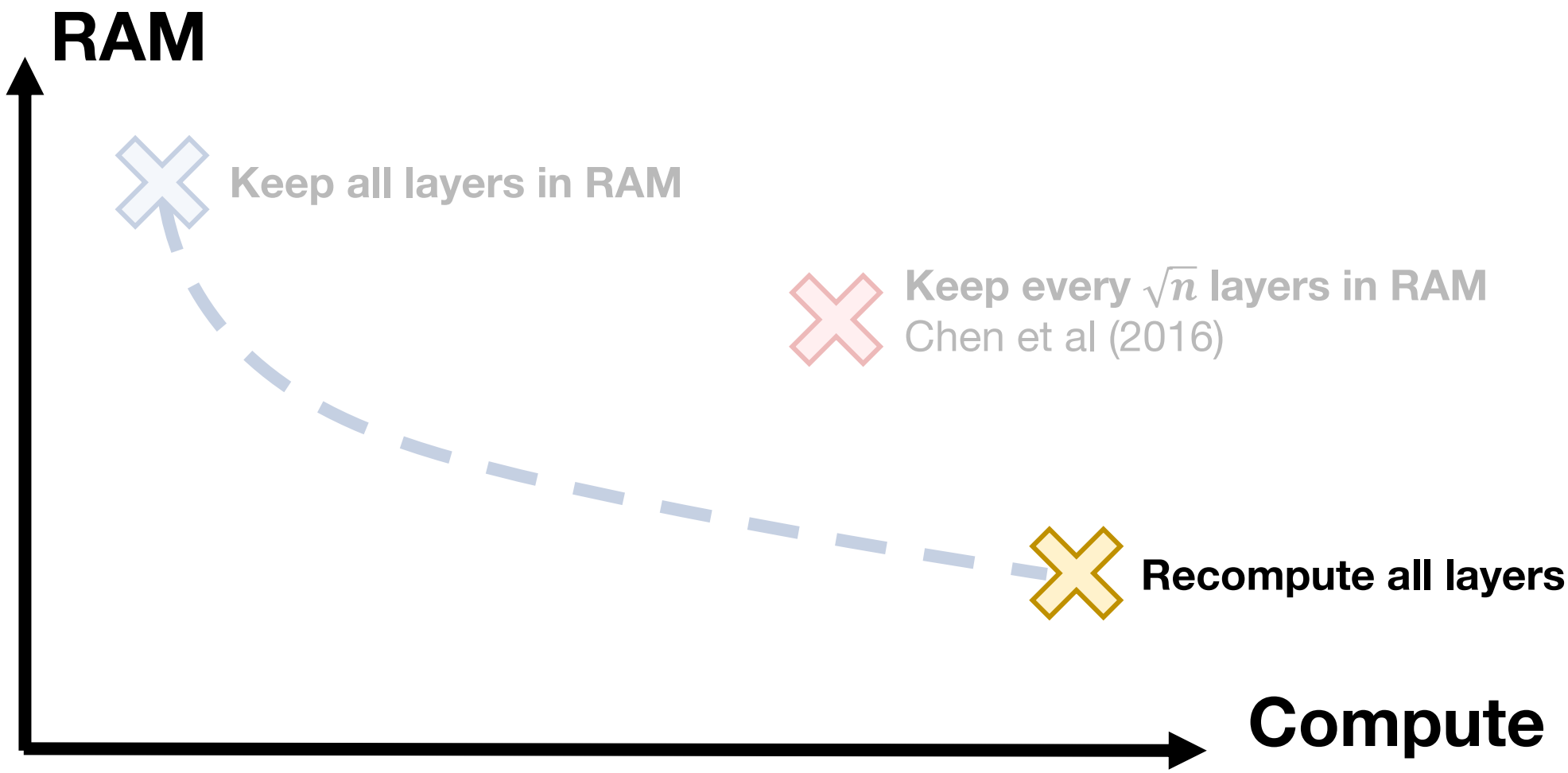




checkmateai.github.io

Compute-hungry backpropagation policy

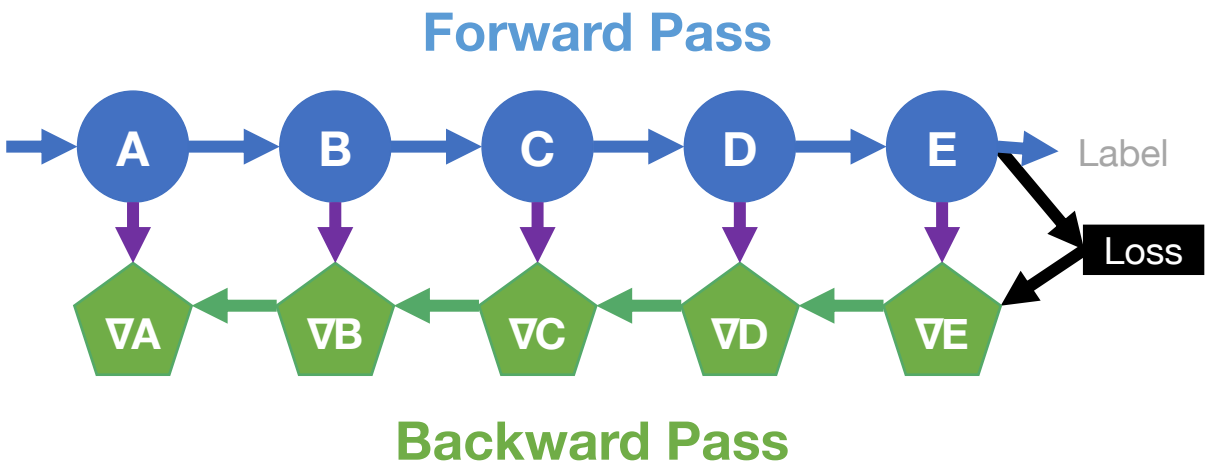
Recompute all layers as needed, storing none

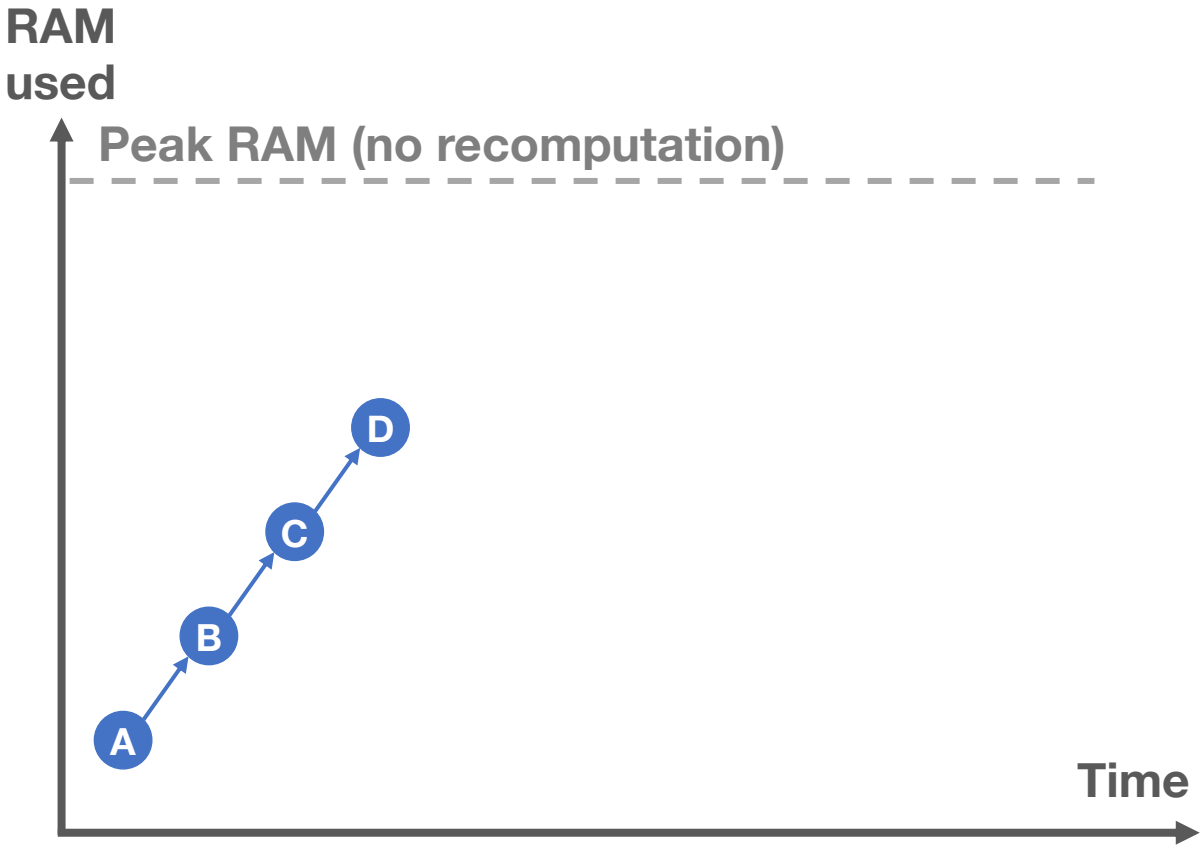


checkmateai.github.io

Compute-hungry backpropagation policy

Recompute all layers





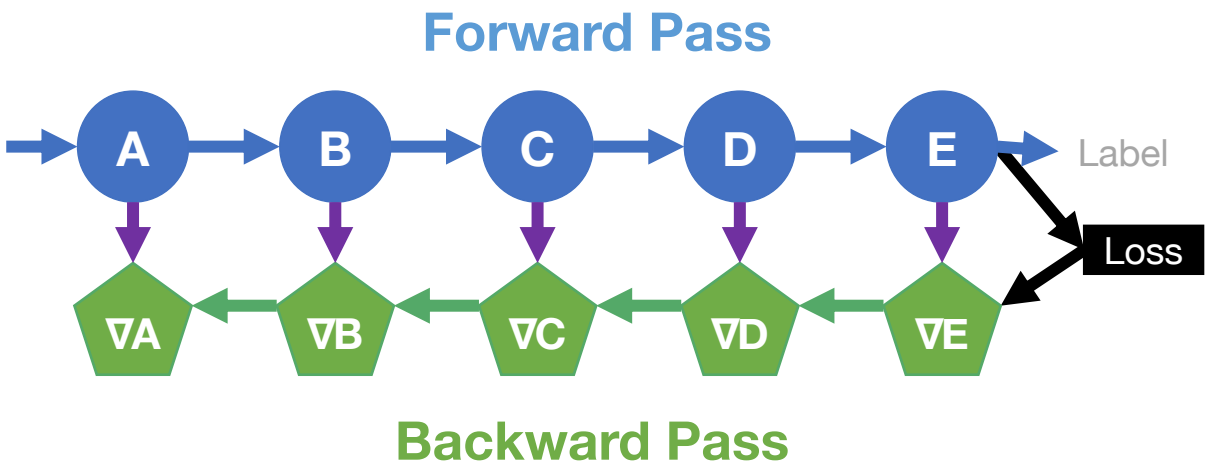
How can we use less memory?

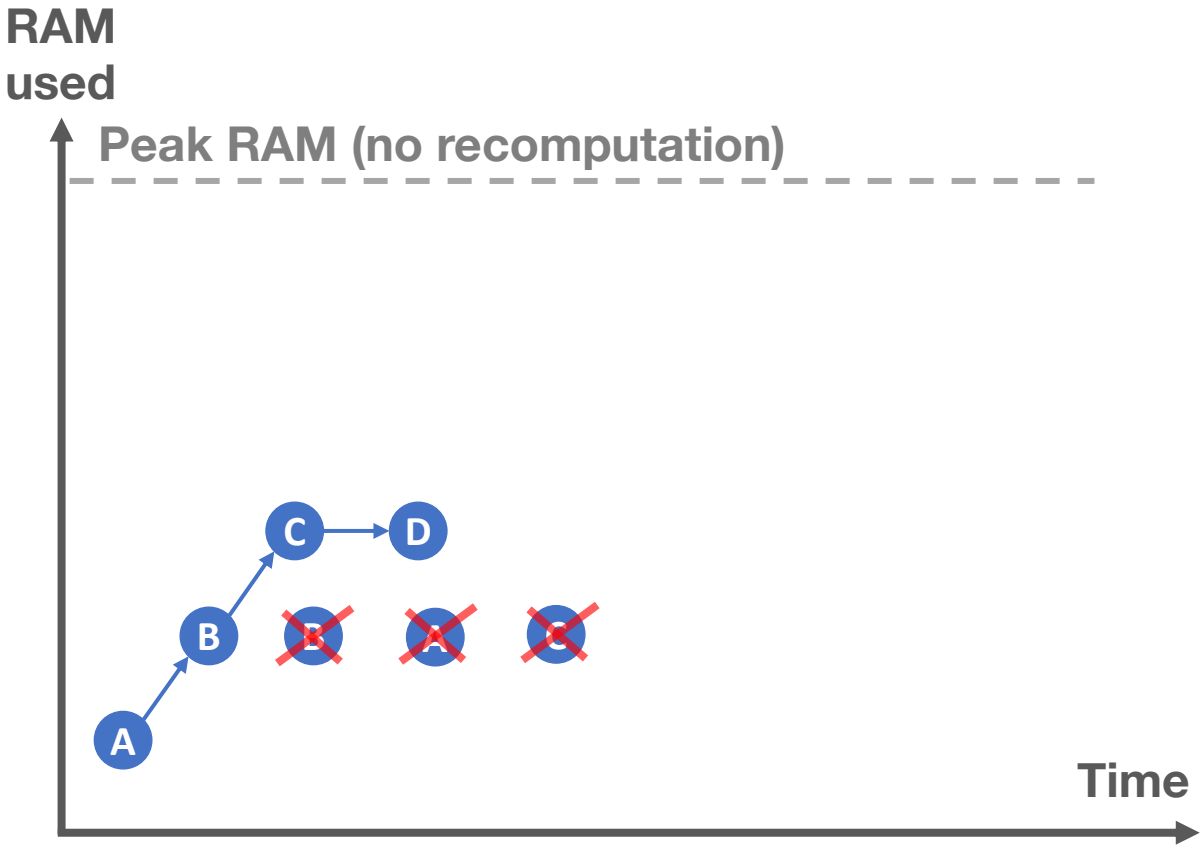
Free early & recompute

checkmateai.github.io

Compute-hungry backpropagation policy

Recompute all layers





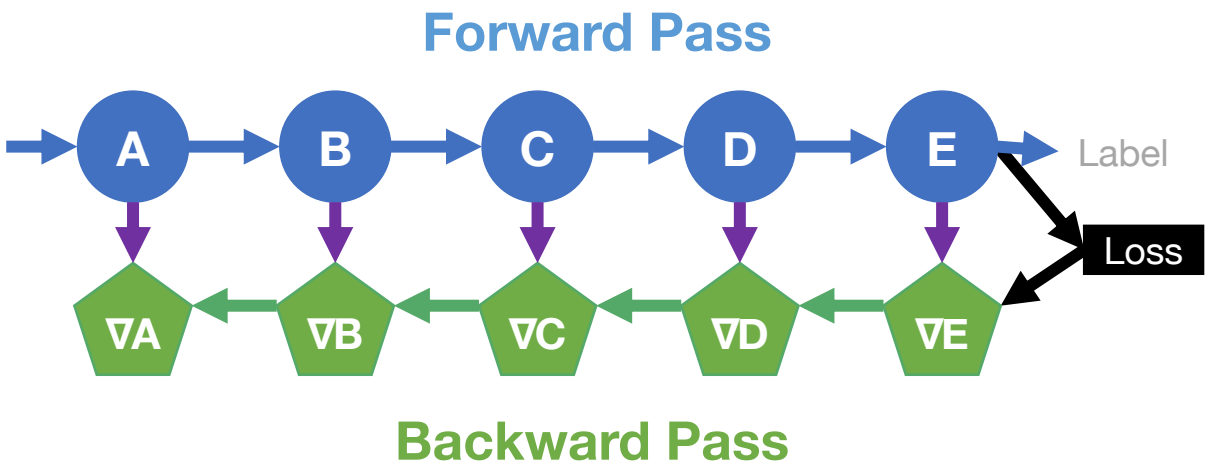
How can we use less memory?

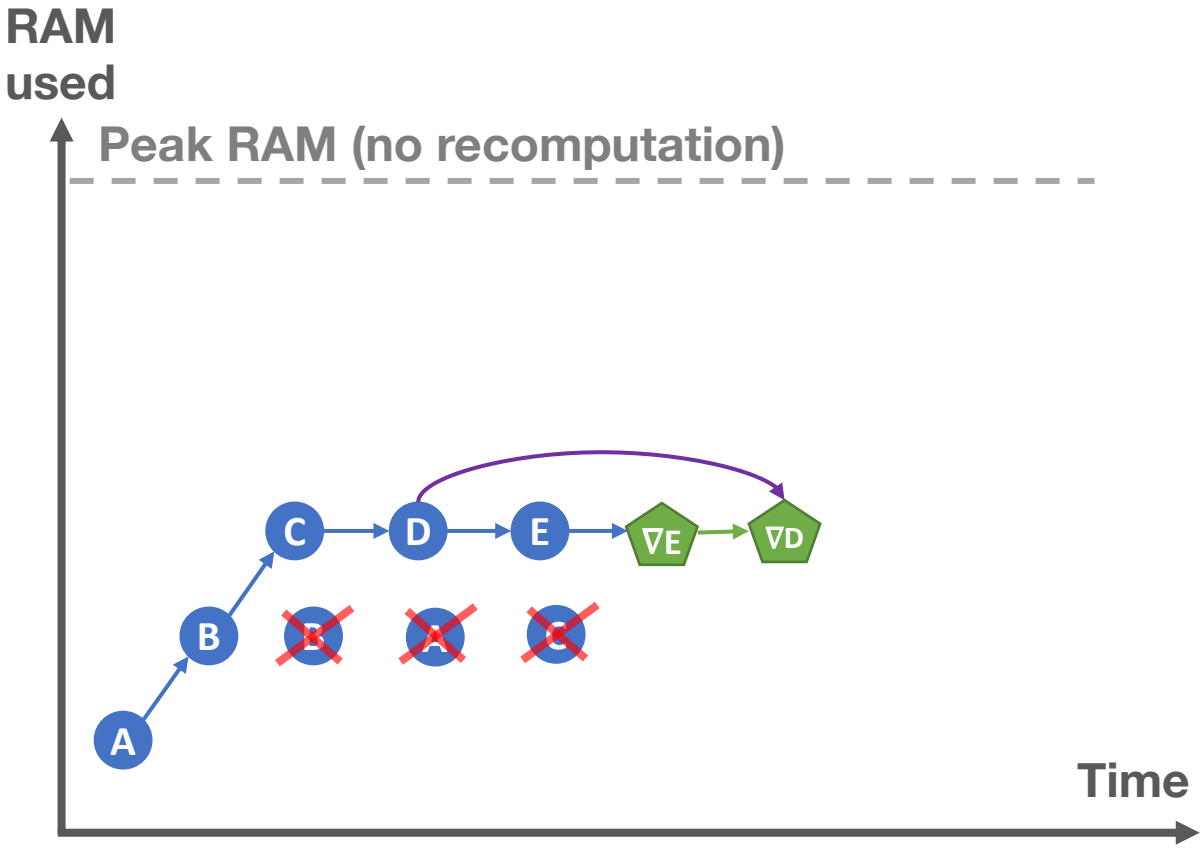
Free early & recompute

checkmateai.github.io

Compute-hungry backpropagation policy

Recompute all layers





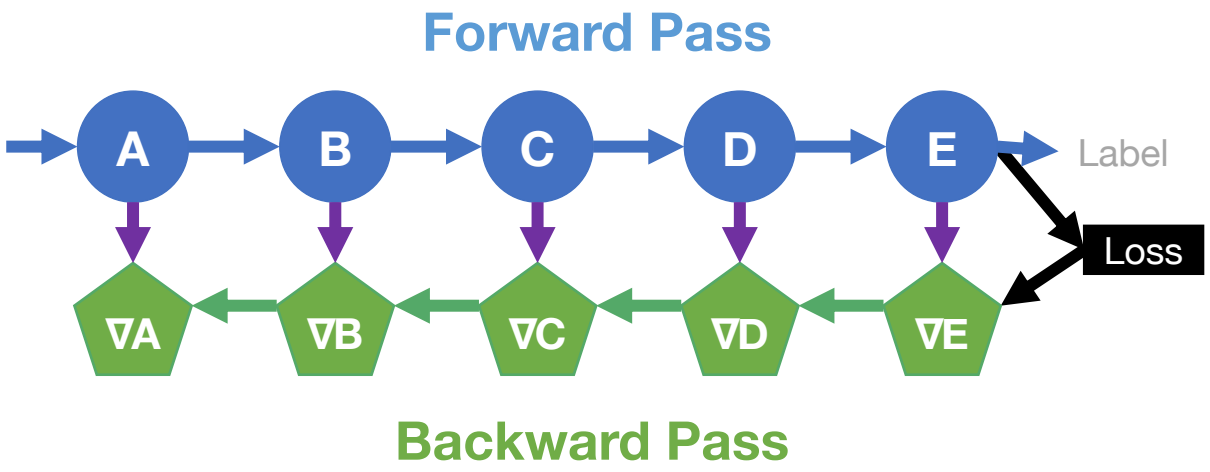
How can we use less memory?

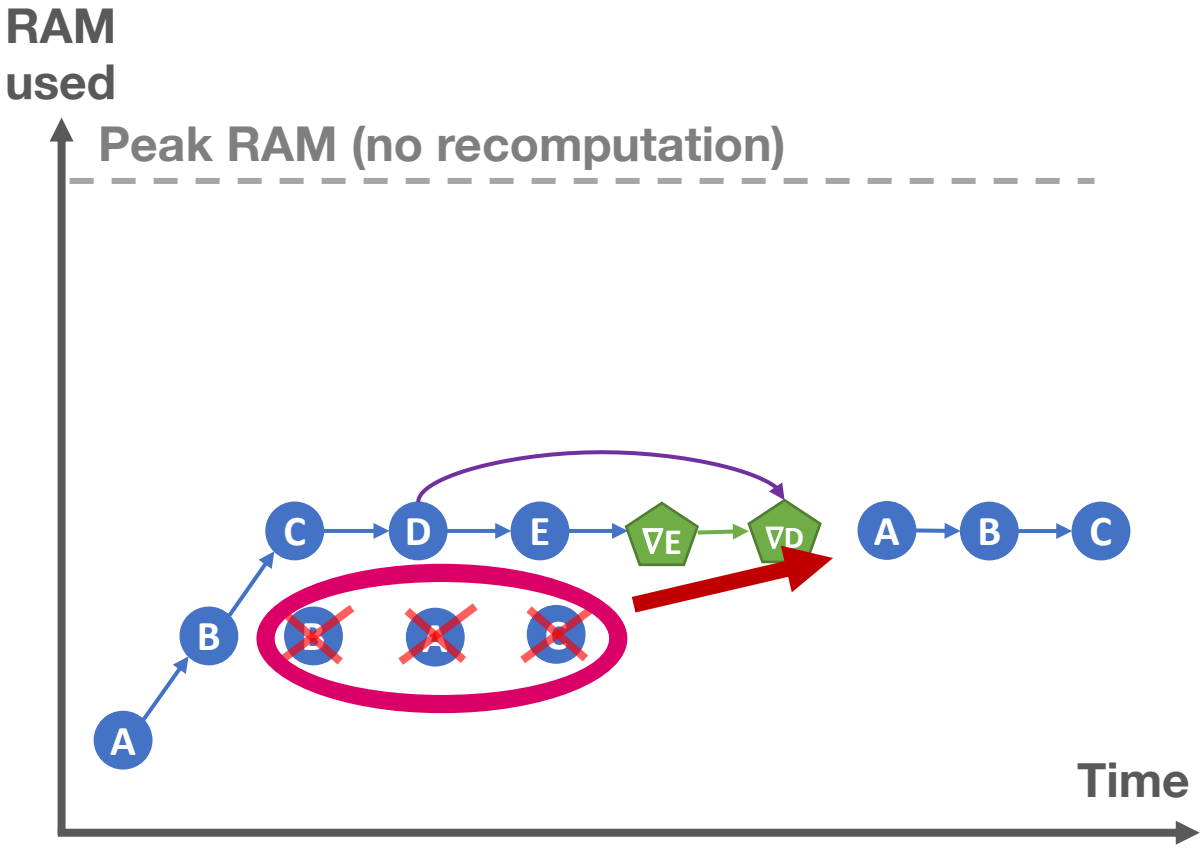
Free early & recompute

checkmateai.github.io

Compute-hungry backpropagation policy

Recompute all layers





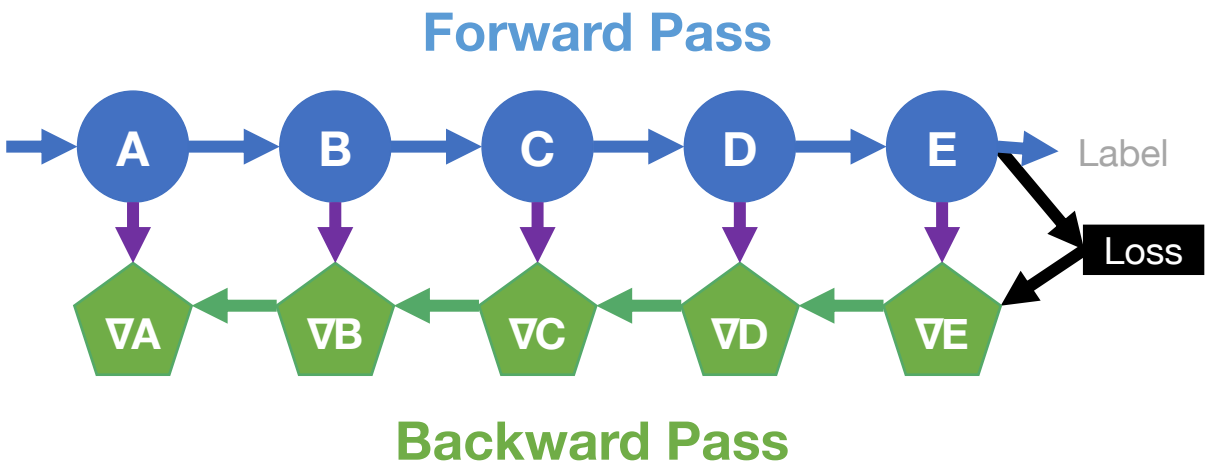
How can we use less memory?

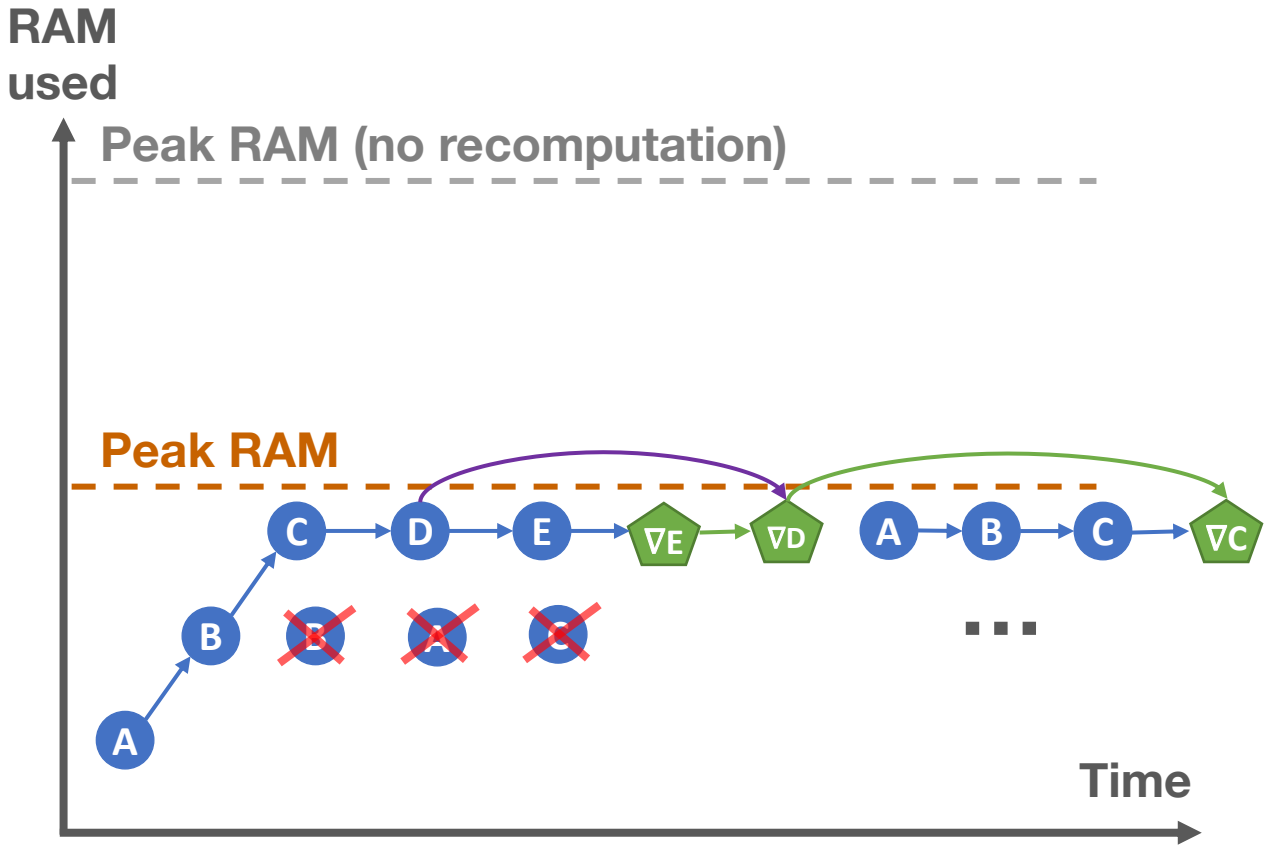
Free early & recompute

checkmateai.github.io

Compute-hungry backpropagation policy

Recompute all layers



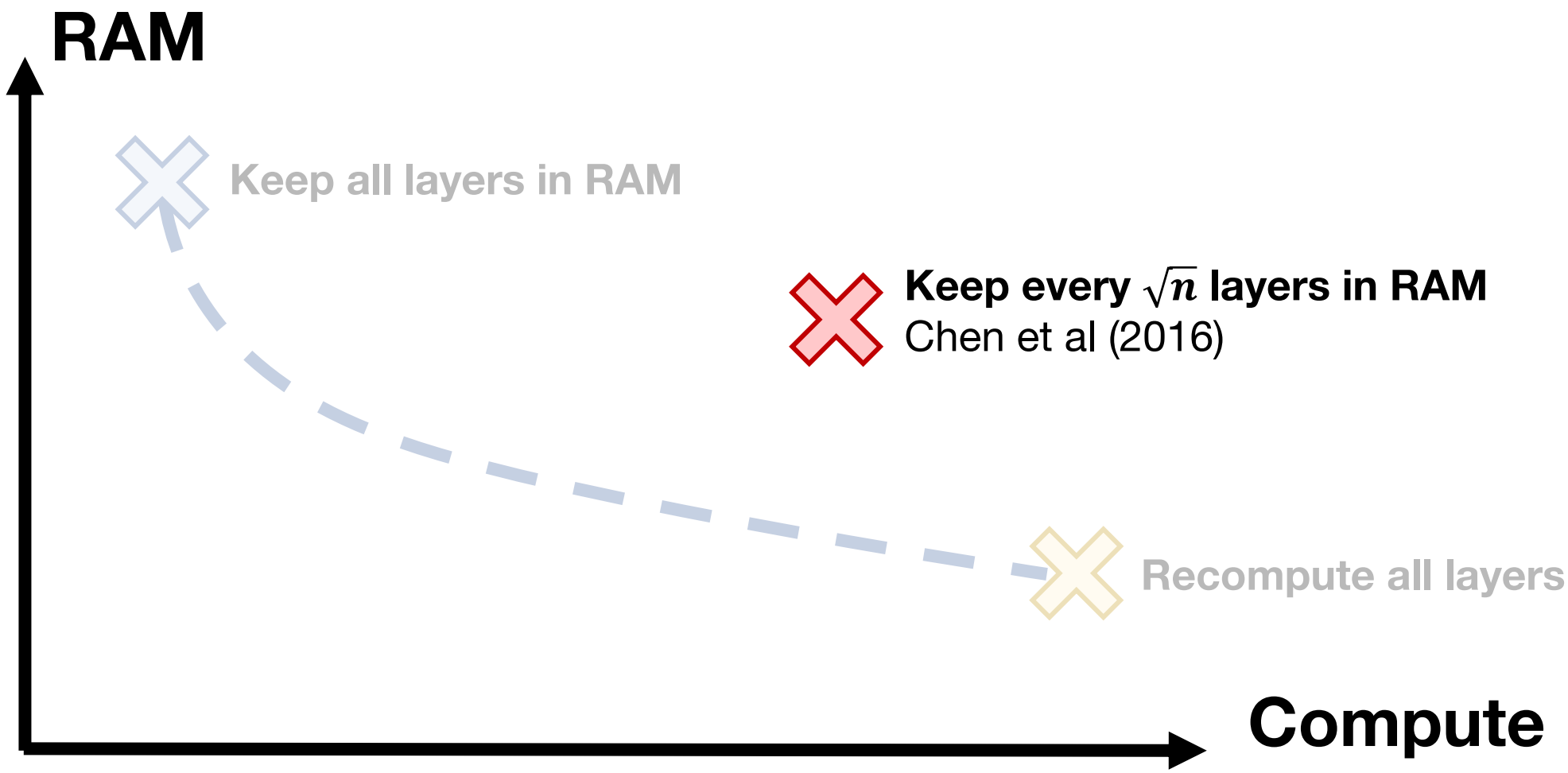


How can we use less memory?

Free early & recompute

checkmateai.github.io

Prior heuristic as an intermediate trade-off point



checkmateai.github.io

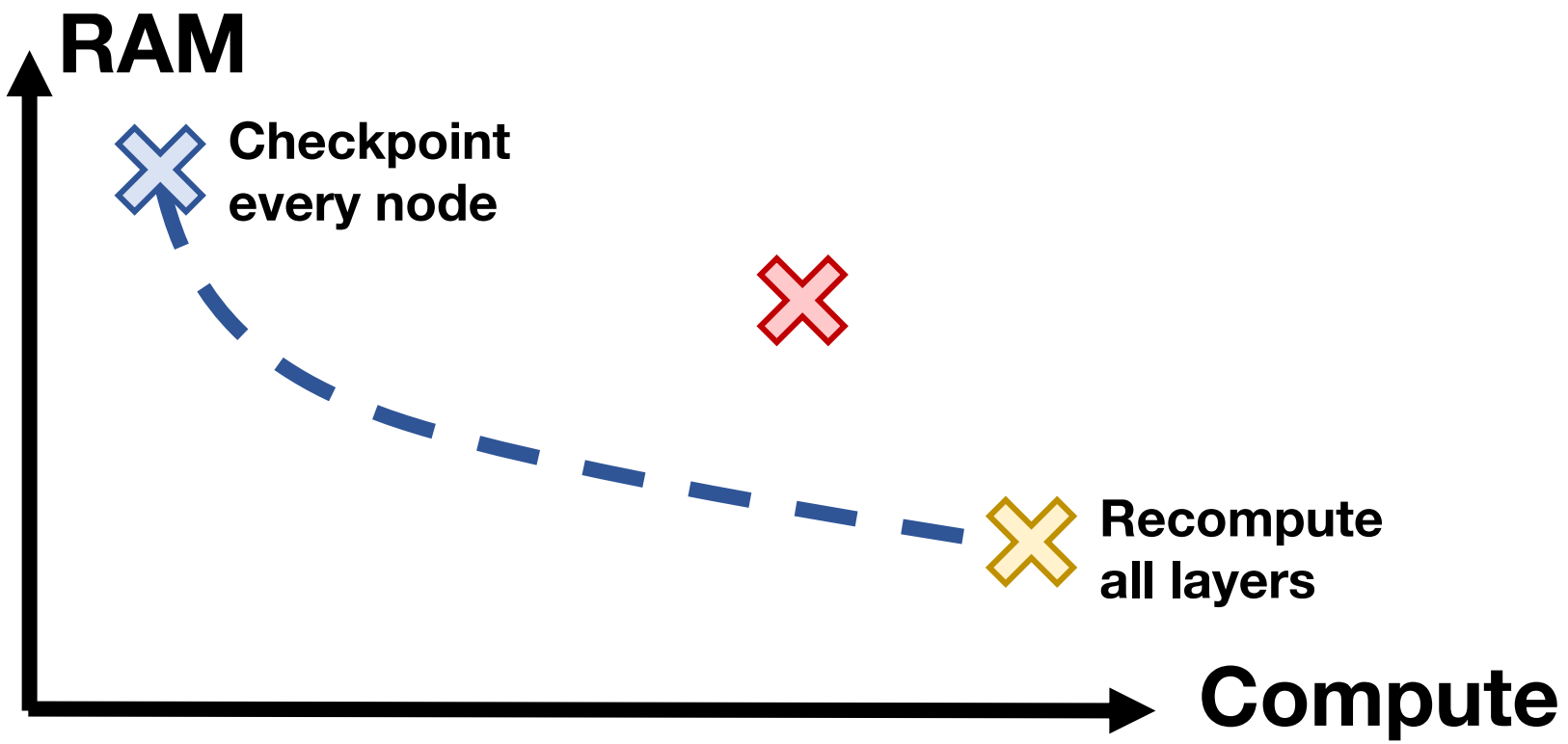
How to trade-off RAM for compute optimally?

Challenges:

1. Variable runtime per layer

2. Variable RAM usage per layer

3. Real DNNs are non-linear

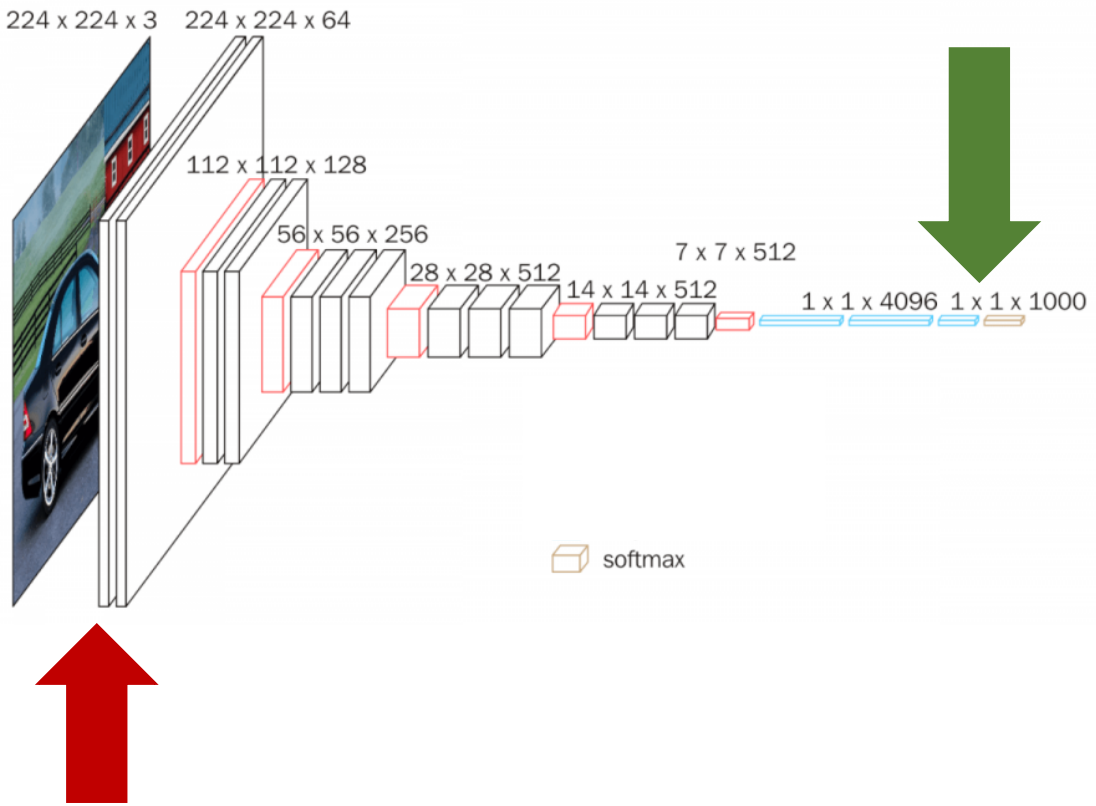


checkmateai.github.io

Why do fixed heuristics perform poorly?

1. Latency is not constant between layers

10^6x compute gap between biggest and smallest layer in VGG19



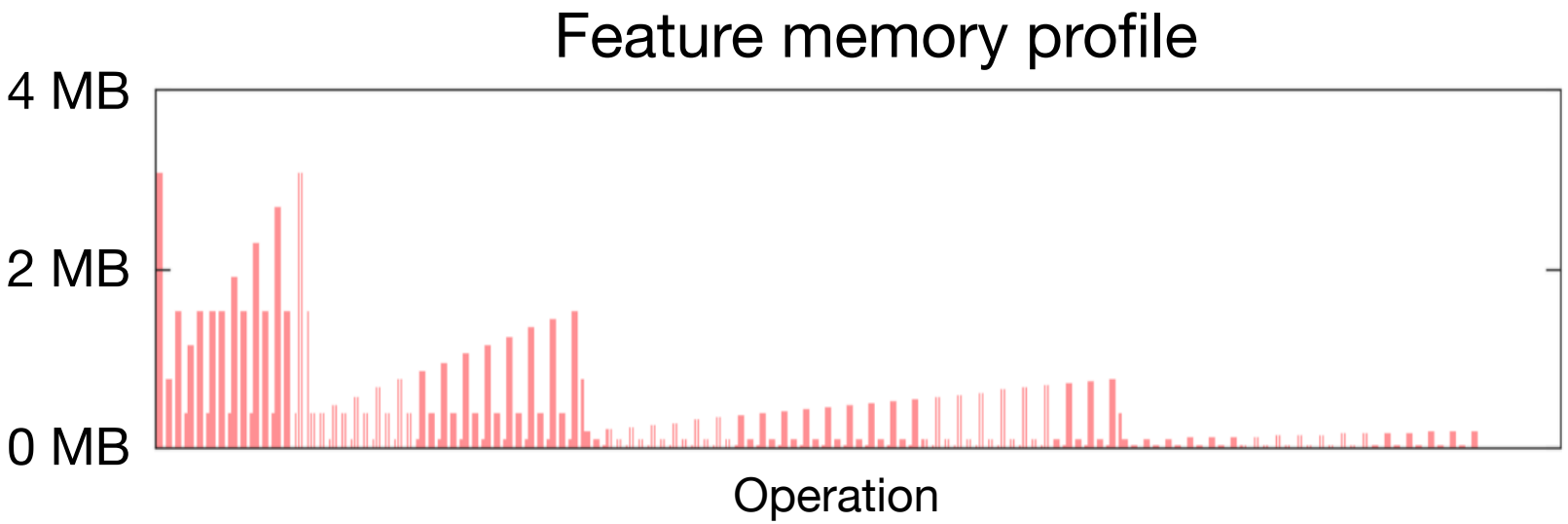
x10^6

checkmateai.github.io

Why do fixed heuristics perform poorly?

2. Tensors are not all the same size

DenseNet-201 has large variability in activation sizes between layers

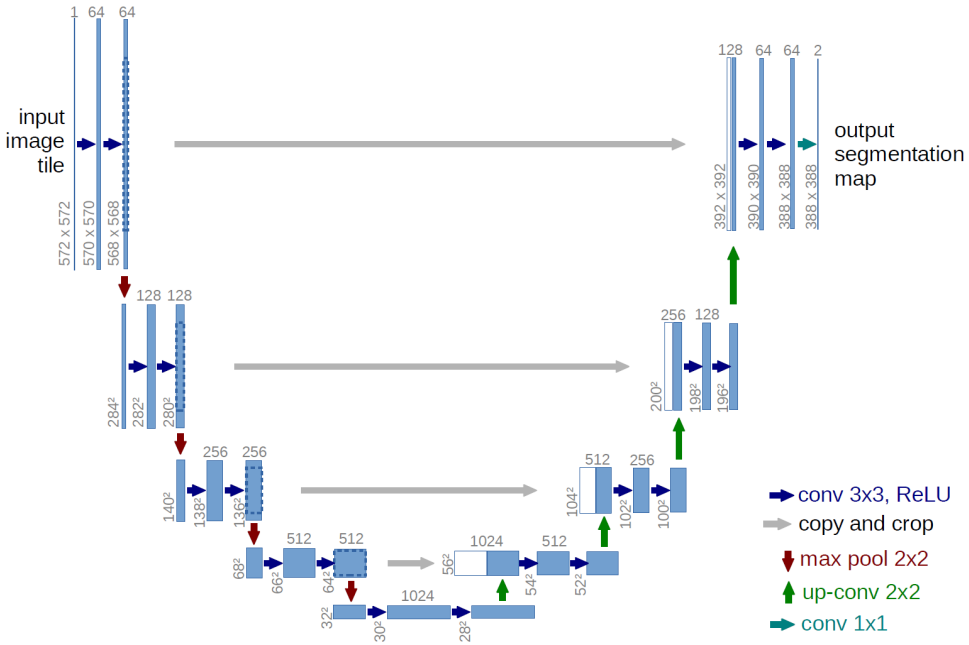


x10³

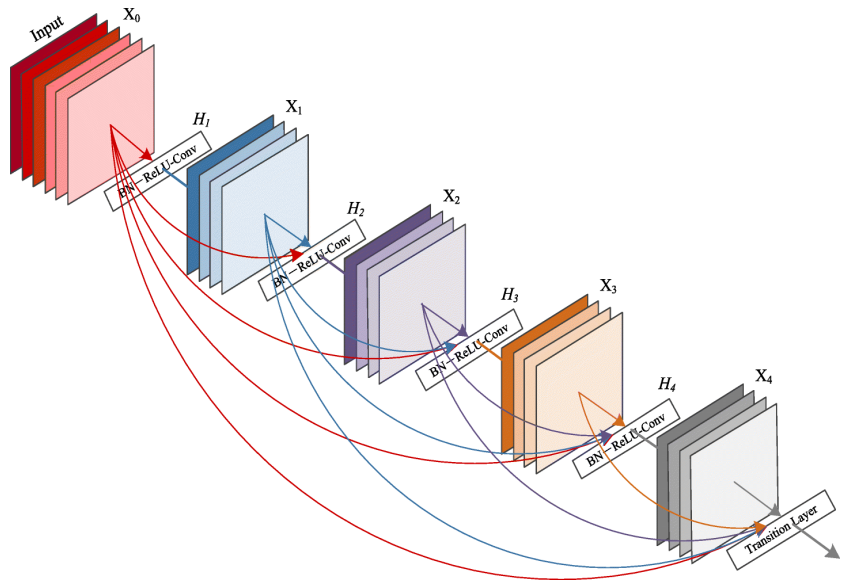
Data from https://github.com/albanie/convnet-burden/blob/master/reports/densenet201.md

Why do fixed heuristics perform poorly?

3. Real DNN architectures are non-linear



U-Net: 11k citations

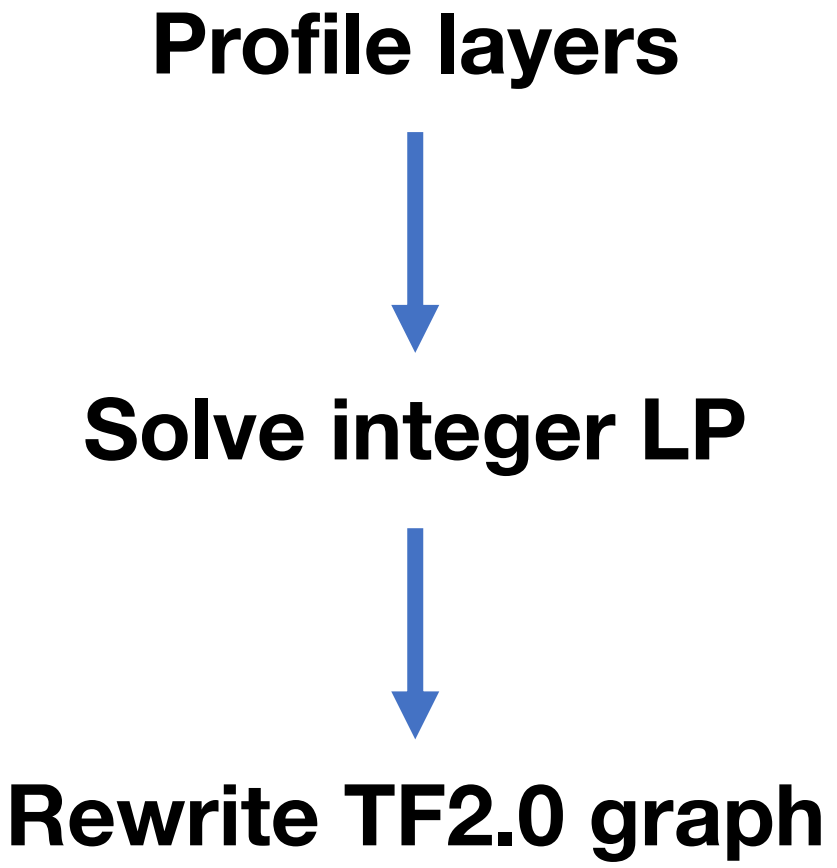


DenseNet: 7k citations

Checkmate

A system for optimal tensor rematerialization

- Statically optimize graph once (10s to 1hr)
- Train optimized graph for weeks
- Checkmate composed of 3 parts:
  - Profiling: hardware/RAM aware schedules
  - Integer LP: enables finding optimal schedule
  - TF2.0 graph pass: support TPU, GPU, CPU



Variance Reduction for Quantized Training

(Preliminary unpublished work.)

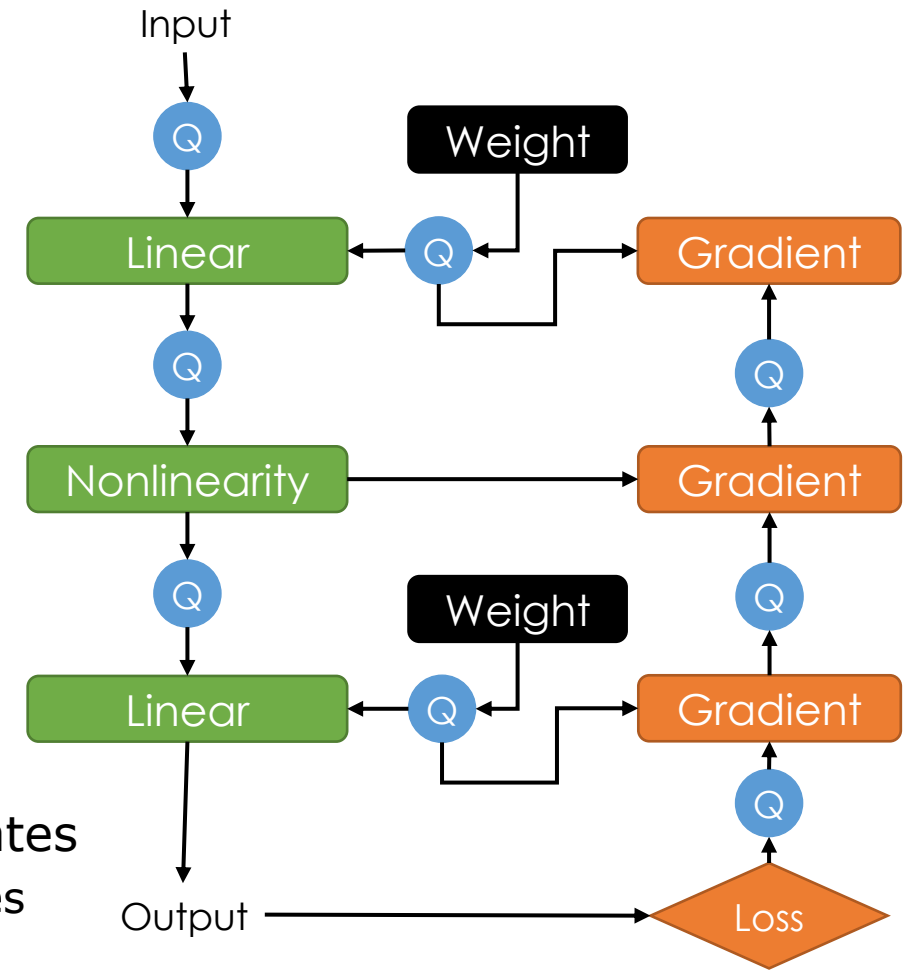
Jianfei Chen (Postdoc)

- Quantized weights, activations, and **gradients**
- Studying uniform **stochastic rounding**
  - **Prove** gradient is **unbiased**
  - hardware support needed…

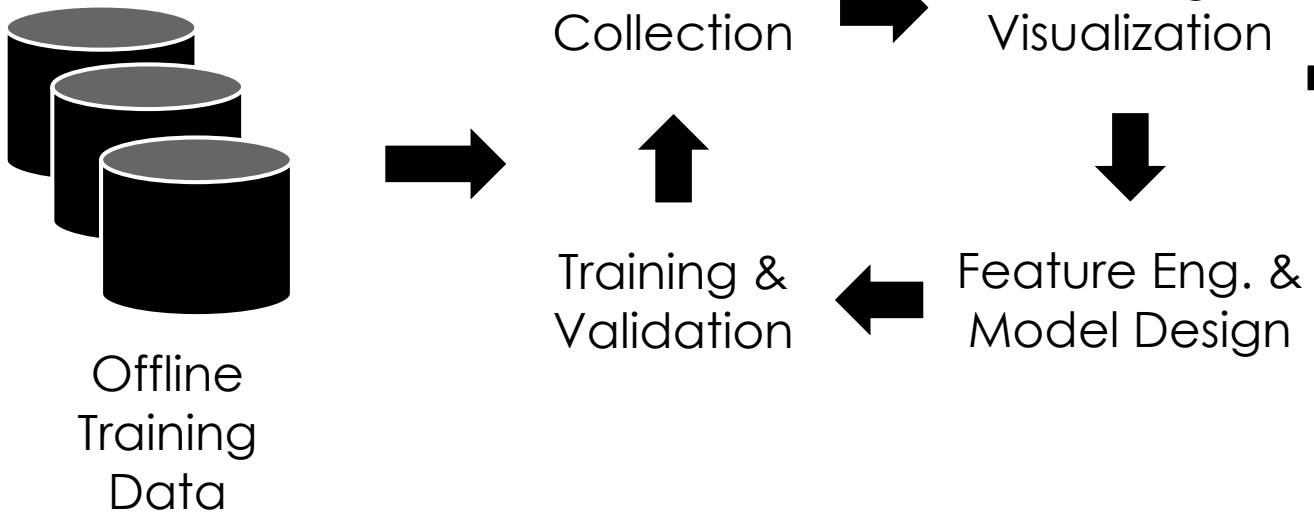
Consequence:

Quantization preserves "correctness" of SGD
→ Does affect convergence rate

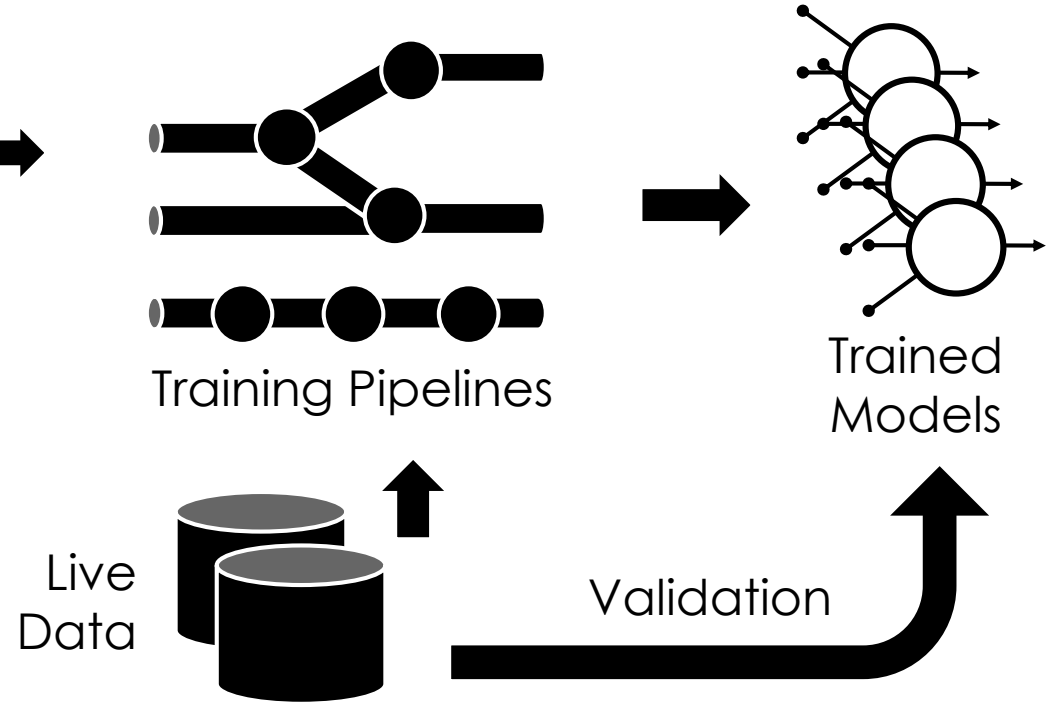
- **Analyze Variance**
  - 1 less bit > 2x increase in stdev of gradient estimates
    - 4x increase in batch size to recover convergence rates
- Studying variance reduction mechanism
  - developing **quantization preconditioners**

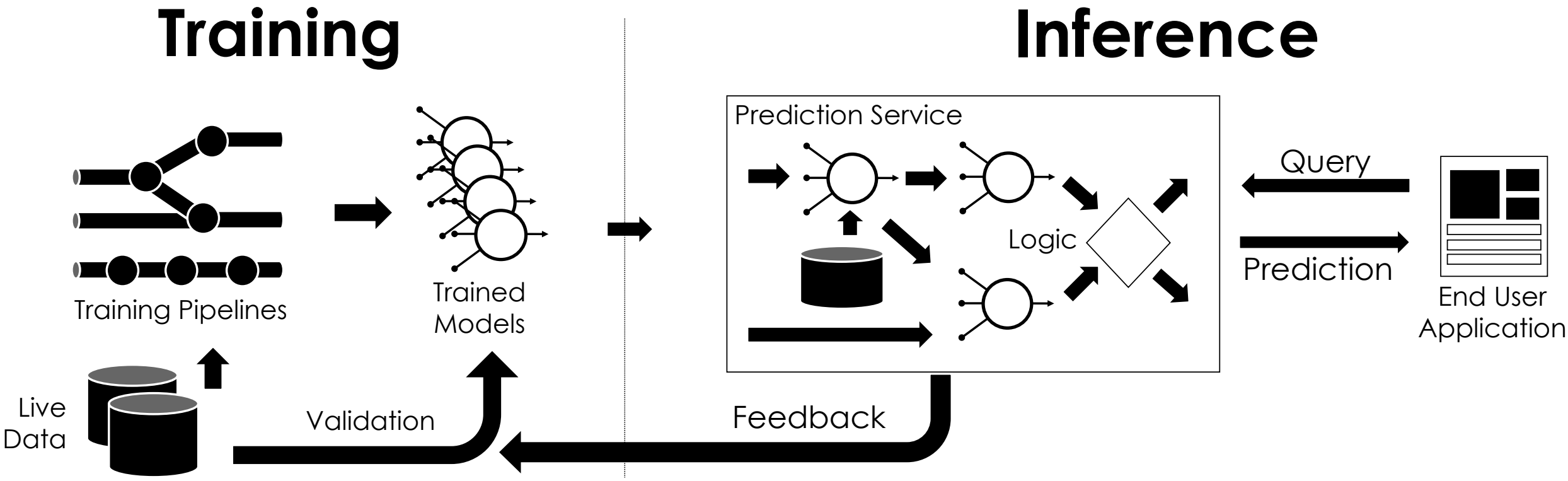


Model Development

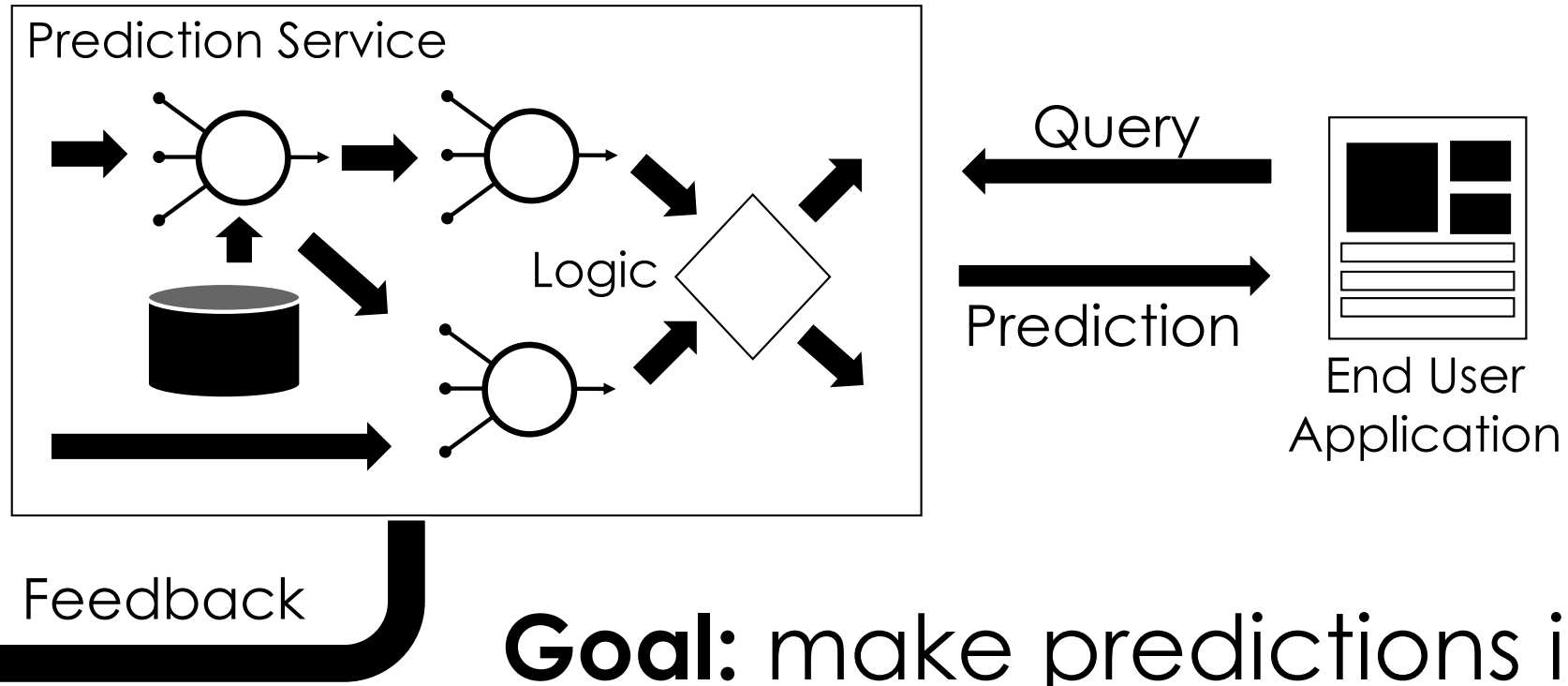


Training




Training
Inference
Training Pipelines
Trained Models
Prediction Service
Logic
Query
Prediction
End User Application
Live Data
Validation
Feedback

Inference



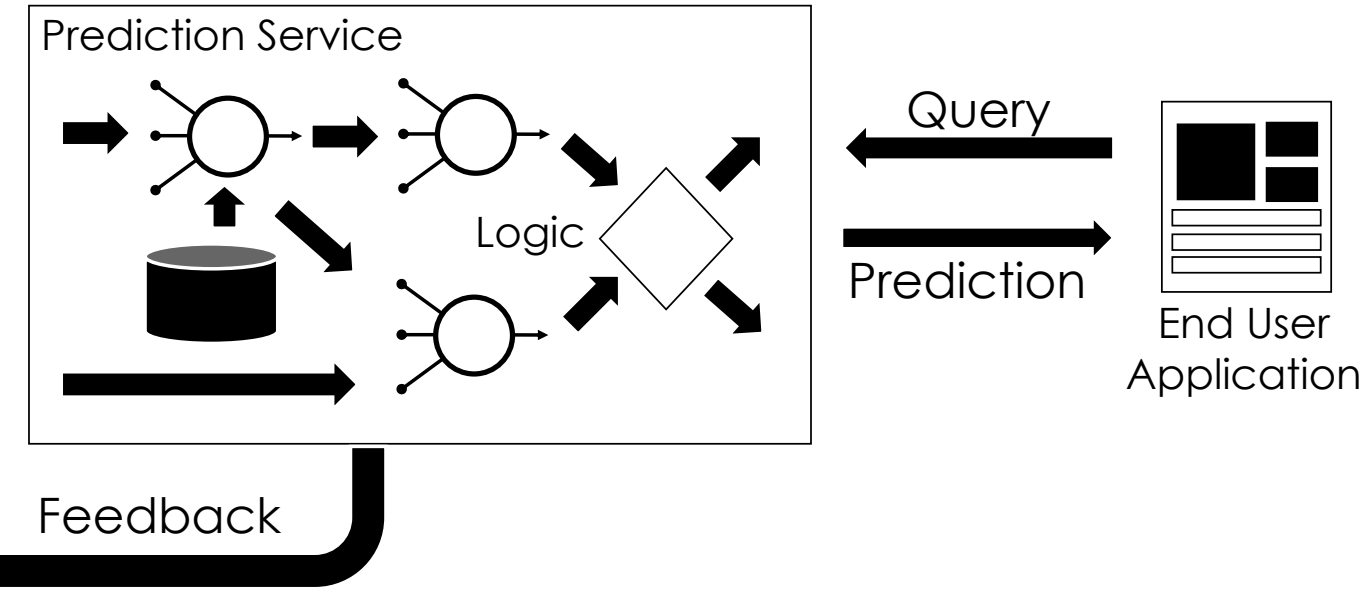
Focus of AI Hardware

Goal: make predictions in ~10ms under heavy load

Complicated by **Deep Neural Networks**

➔ New **ML Algorithms** and **Systems**

Inference *is multiple Applications.*



End User Application

The Cloud

The "Edge"

Mobile Devices

The Cloud

The "Edge"

Mobile Devices

Throughput Oriented

Latency Oriented

Variable Load

Predictable Load

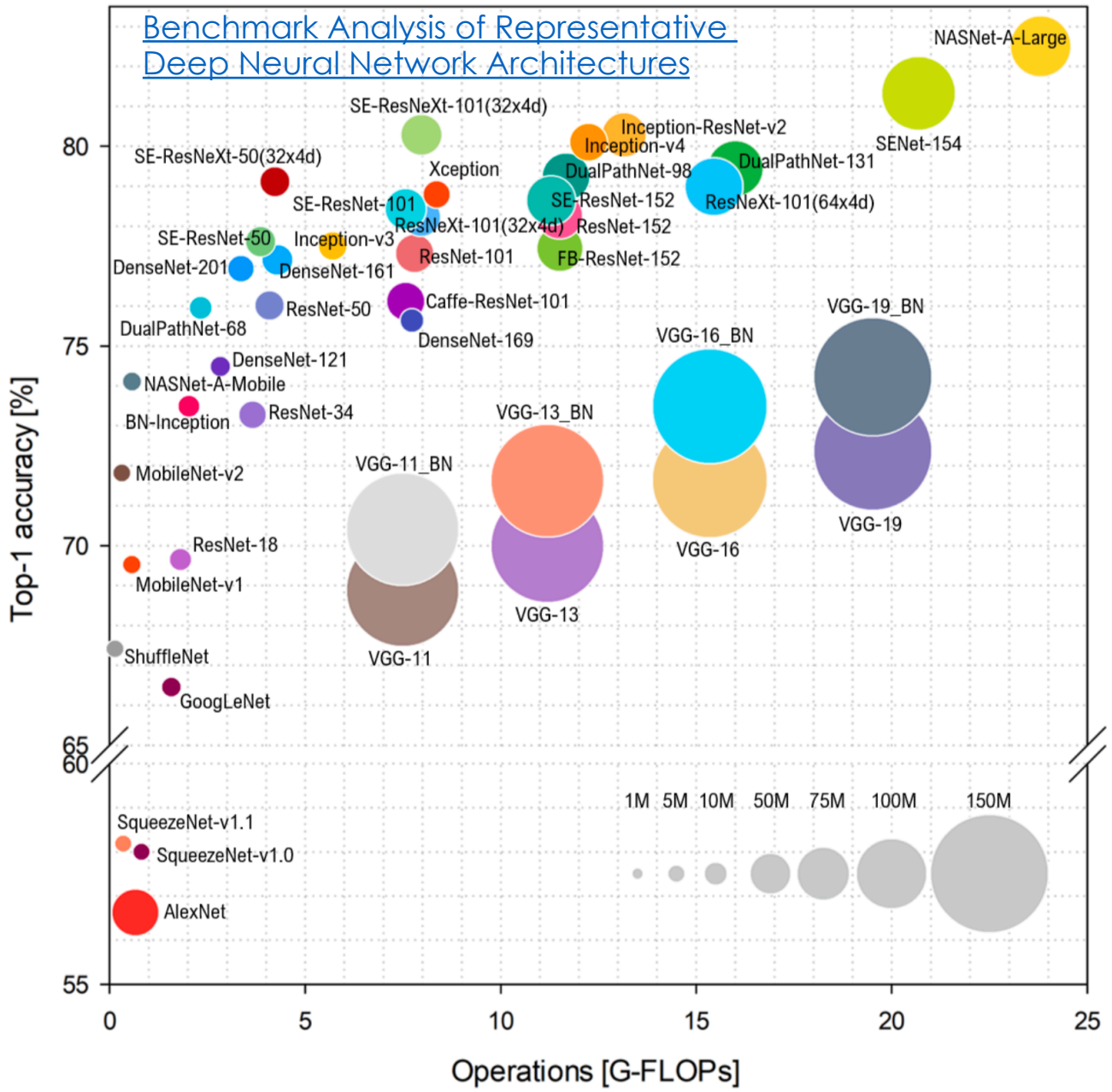
Budget

Power

Energy

Inference in Deep Learning Models

- **Compute intensive**
- **Less memory intensive** than training



Inference in Deep Learning Models

Benchmark Analysis of Representative Deep Neural Network Architectures

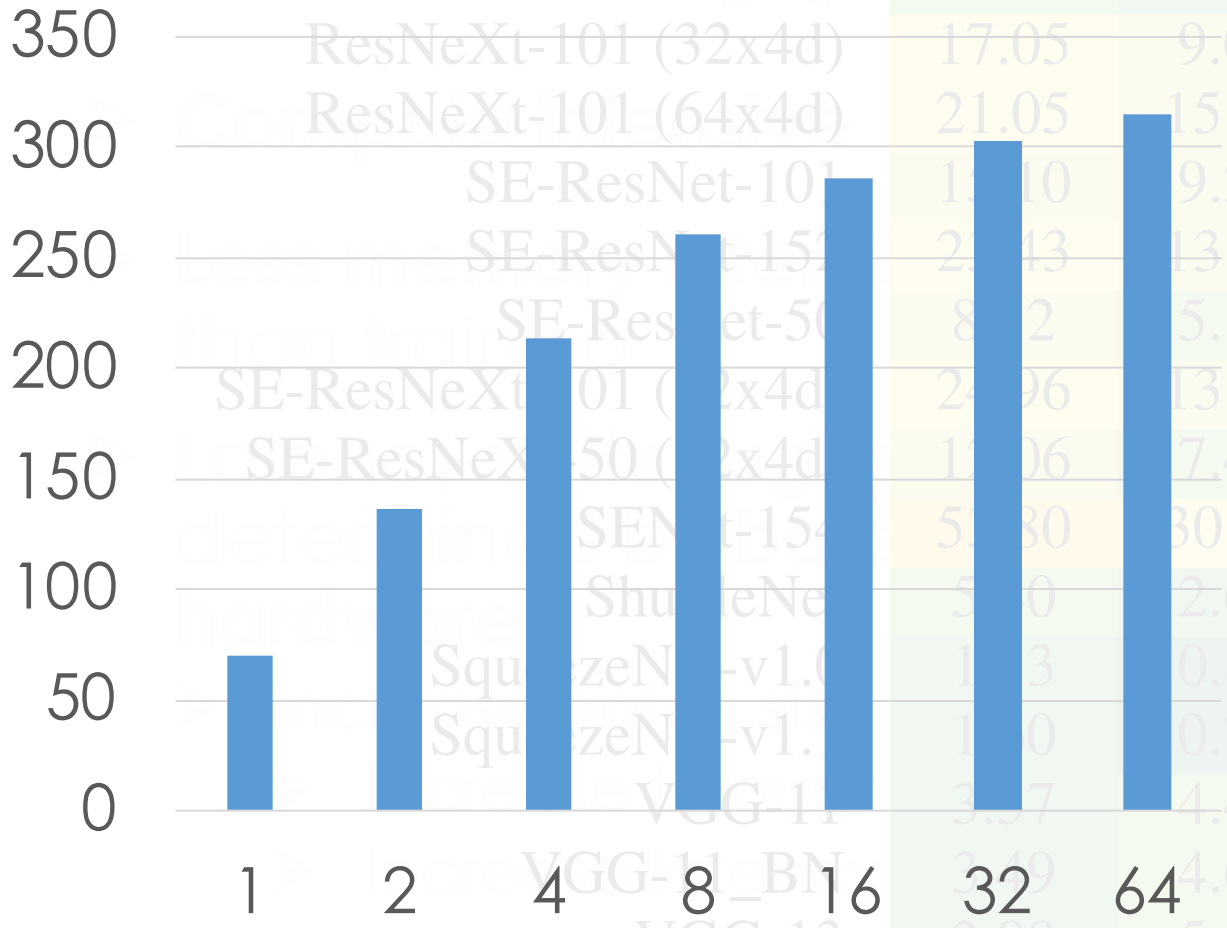
TitanXP

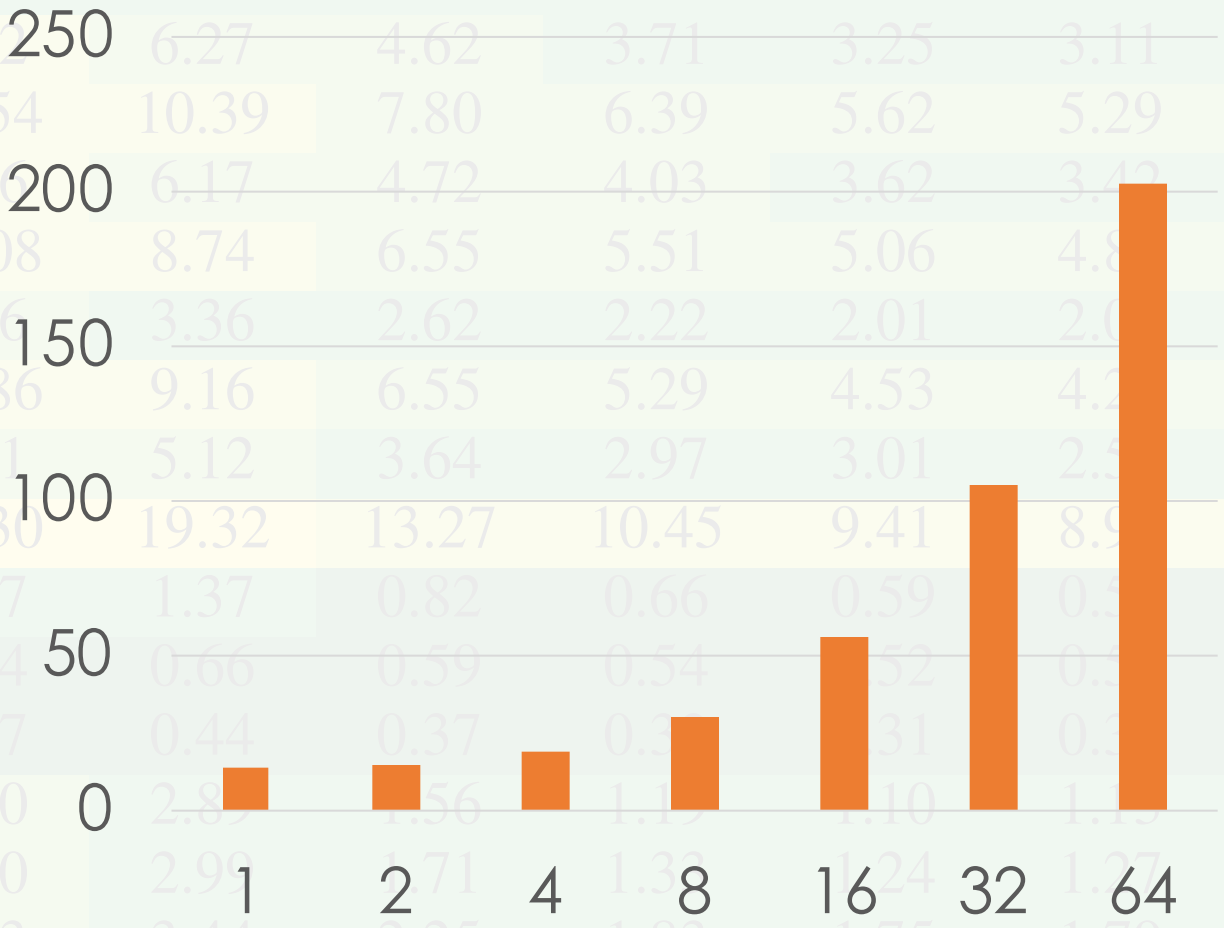
FPS	>1000	>250	>125	>62.5	>30	>15	>5	<=5
ms	<1	<4	<8	<16	<33	<66	<200	>=200

- **Compute intensive**
- **Less memory intensive** than training
- Latency vs throughput tradeoff determined by **batch size** and **hardware**
  - **Increase batch size**
    - Increase throughput ☺
    - Increase latency ☹

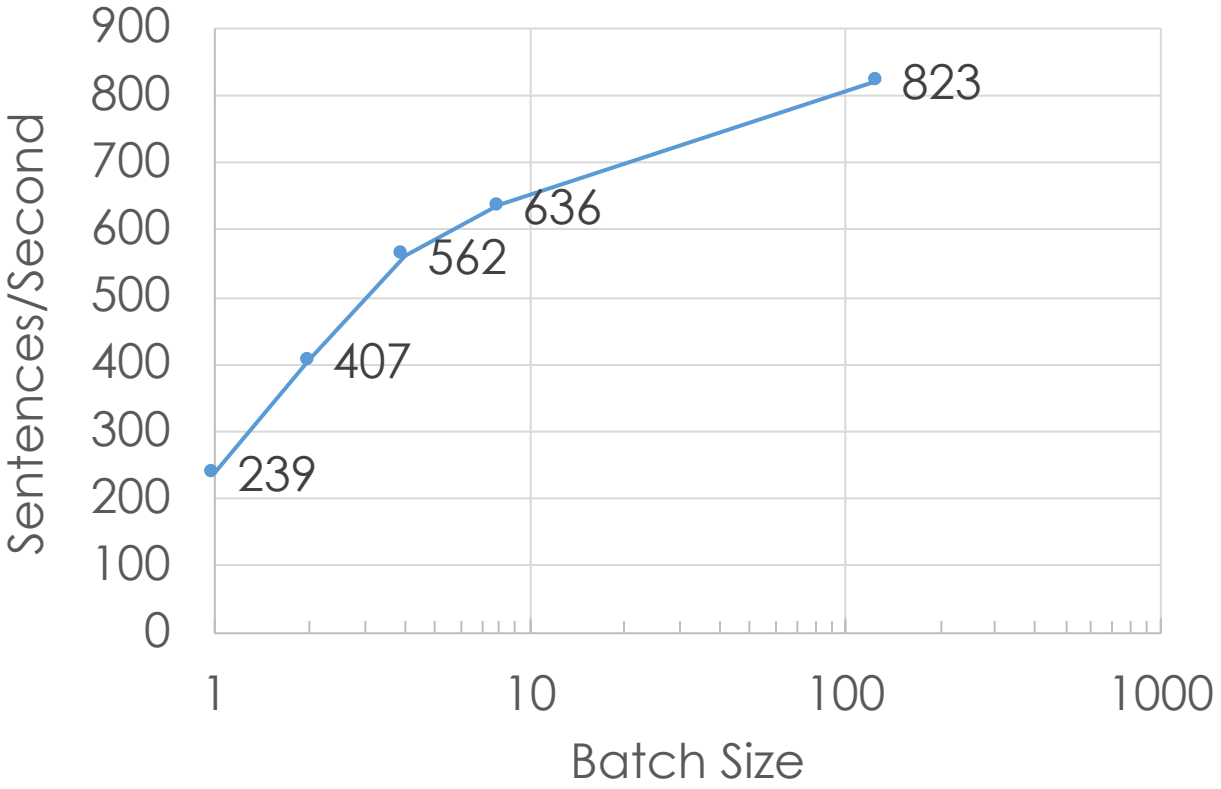
DNN	1	2	4	8	16	32	64
AlexNet	1.28	0.70	0.48	0.27	0.18	0.14	0.15
BN-Inception	5.79	3.00	1.64	1.10	0.87	0.77	0.71
CaffeResNet-101	8.20	4.82	3.32	2.54	2.27	2.16	2.08
DenseNet-121 (k=32)	8.93	4.41	2.64	1.96	1.64	1.44	1.39
DenseNet-169 (k=32)	13.03	6.72	3.97	2.73	2.14	1.87	1.75
DenseNet-201 (k=32)	17.15	9.25	5.36	3.66	2.84	2.41	2.27
DenseNet-161 (k=48)	15.50	9.10	5.89	4.45	3.66	3.43	3.24
DPN-68	10.68	5.36	3.24	2.47	1.80	1.59	1.52
DPN-98	22.31	13.84	8.97	6.77	5.59	4.96	4.72
DPN-131	29.70	18.29	11.96	9.12	7.57	6.72	6.37
FBResNet-152	14.55	7.79	5.15	4.31	3.96	3.76	3.65
GoogLeNet	4.54	2.44	1.65	1.06	0.86	0.76	0.72
Inception-ResNet-v2	25.94	14.36	8.82	6.43	5.19	4.88	4.59
Inception-v3	10.10	5.70	3.65	2.54	2.05	1.89	1.80
Inception-v4	18.96	10.61	6.53	4.85	4.10	3.77	3.61
MobileNet-v1	2.45	0.89	0.68	0.60	0.55	0.53	0.53
MobileNet-v2	3.34	1.63	0.95	0.78	0.72	0.63	0.61
NASNet-A-Large	32.30	23.00	19.75	18.49	18.11	17.73	17.77
NASNet-A-Mobile	22.36	11.44	5.60	2.81	1.61	1.75	1.51
ResNet-101	8.90	5.16	3.32	2.69	2.42	2.29	2.21
ResNet-152	14.31	7.36	4.68	3.83	3.50	3.30	3.17
ResNet-18	1.79	1.01	0.70	0.56	0.51	0.41	0.38
ResNet-34	3.11	1.80	1.20	0.96	0.82	0.71	0.67
ResNet-50	5.10	2.87	1.99	1.65	1.49	1.37	1.34
ResNeXt-101 (32x4d)	17.05	9.02	6.27	4.62	3.71	3.25	3.11
ResNeXt-101 (64x4d)	21.05	15.54	10.39	7.80	6.39	5.62	5.29
SE-ResNet-101	15.10	9.26	6.17	4.72	4.03	3.62	3.42
SE-ResNet-152	23.43	13.08	8.74	6.55	5.51	5.06	4.85
SE-ResNet-50	8.32	5.16	3.36	2.62	2.22	2.01	2.06
SE-ResNeXt-101 (32x4d)	24.96	13.86	9.16	6.55	5.29	4.53	4.29
SE-ResNeXt-50 (32x4d)	12.06	7.41	5.12	3.64	2.97	3.01	2.56
SENet-154	53.80	30.30	19.32	13.27	10.45	9.41	8.91
ShuffleNet	5.40	2.67	1.37	0.82	0.66	0.59	0.56
SqueezeNet-v1.0	1.53	0.84	0.66	0.59	0.54	0.52	0.53
SqueezeNet-v1.1	1.60	0.77	0.44	0.37	0.32	0.31	0.30
VGG-11	3.57	4.40	2.89	1.56	1.19	1.10	1.13
VGG-11_BN	3.49	4.60	2.99	1.71	1.33	1.24	1.27
VGG-13	3.88	5.03	3.44	2.25	1.83	1.75	1.79
VGG-13_BN	4.40	5.37	3.71	2.42	2.05	1.97	2.00
VGG-16	5.17	5.91	4.01	2.84	2.20	2.12	2.15
VGG-16_BN	5.04	5.95	4.27	3.06	2.45	2.36	2.41
VGG-19	5.50	6.26	4.71	3.29	2.59	2.52	2.50
VGG-19_BN	6.17	6.67	4.86	3.56	2.88	2.74	2.76
Xception	6.44	5.35	4.90	4.47	4.41	4.41	4.36

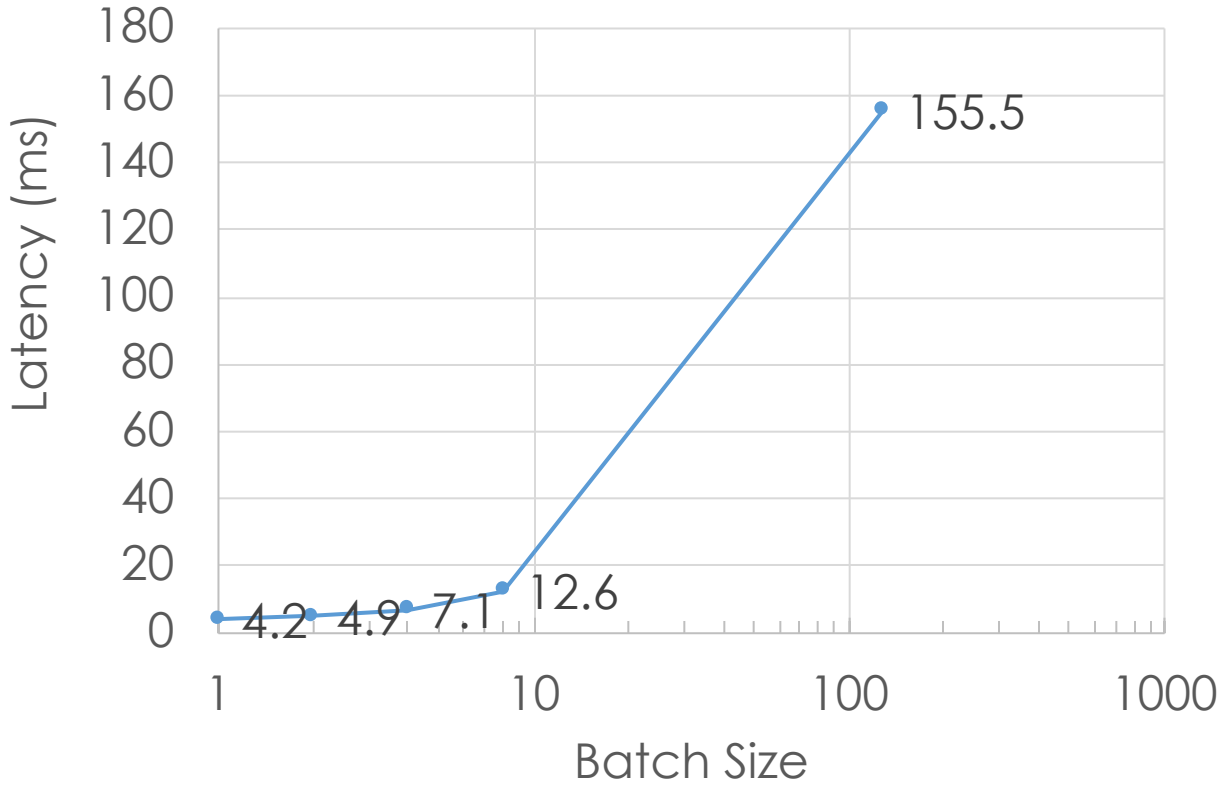
Batch Size.	1	2	4	8	16	32	64
ResNet-152	14.31	7.36	4.68	3.83	3.50	3.30	3.17





BERT-Large on a V100 (~$10K)



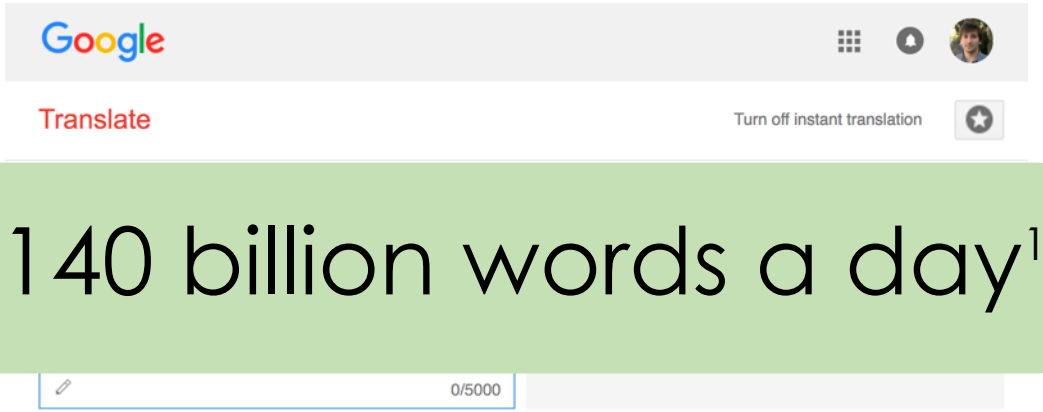


Results included Mixed precision optimizations!

Numbers obtained from: https://developer.nvidia.com/deep-learning-performance-training-inference

Google Translate

Serving



140 billion words a day[1]

82,000 GPUs
running 24/7

Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi
yonghui,schuster,zhifengc,qvl,mnorouzi@google.com

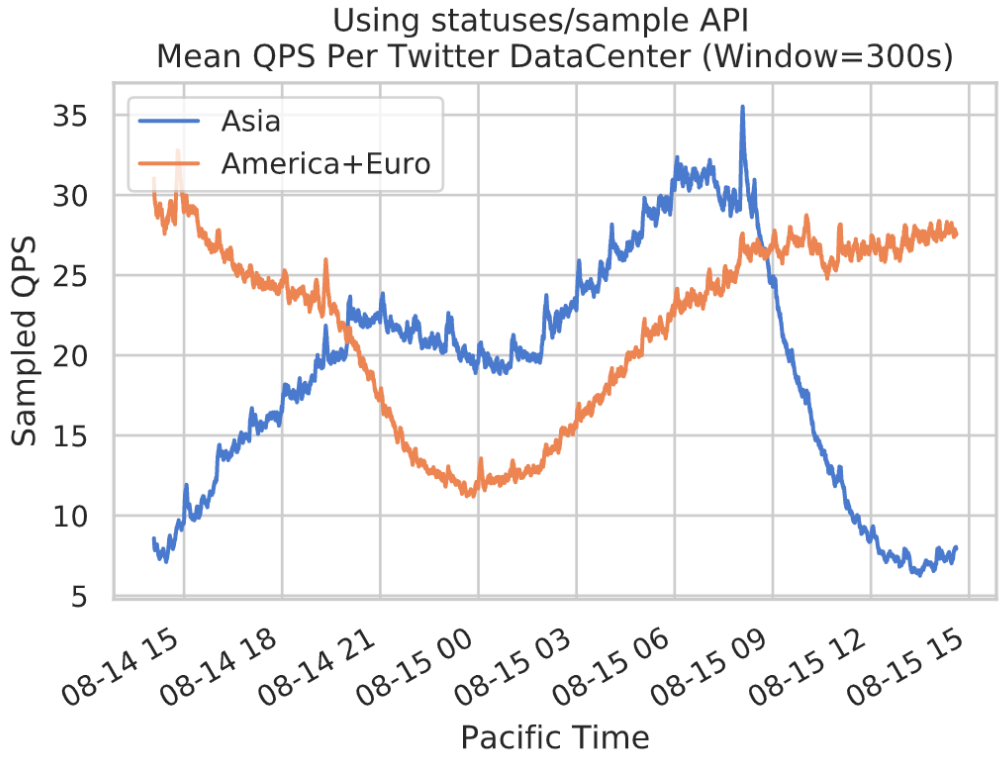
Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, Jeffrey Dean

*"If each of the **world's Android phones** used the new Google voice search for just **three minutes a day**, these engineers realized, the company would **need twice as many data centers.**"*
– Wired

Designed New Hardware!
Tensor Processing Unit (TPU)

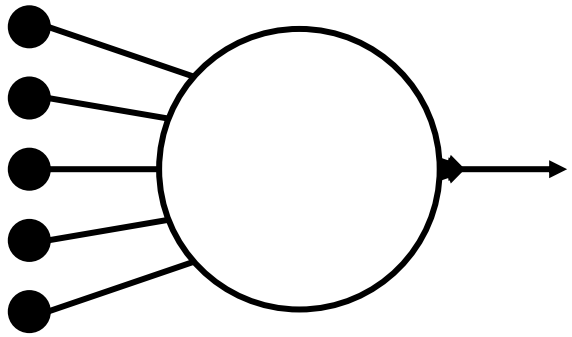
[1] https://www.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html

Other Challenges?

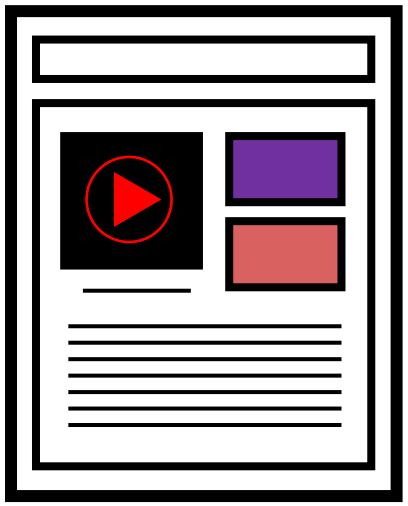


- **Bursty load** →
  - overprovision resources →
    - **expensive**
  - TPU reports 28% utilization of vector units in production
  - Solutions
    - statistical multiplexing → hardware not designed for multitenancy
    - could try to predict arrival process → generally difficult to predict
- **Versioning** and **testing models**
- **Prediction pipelines** → more on this soon

Inference



Big Model

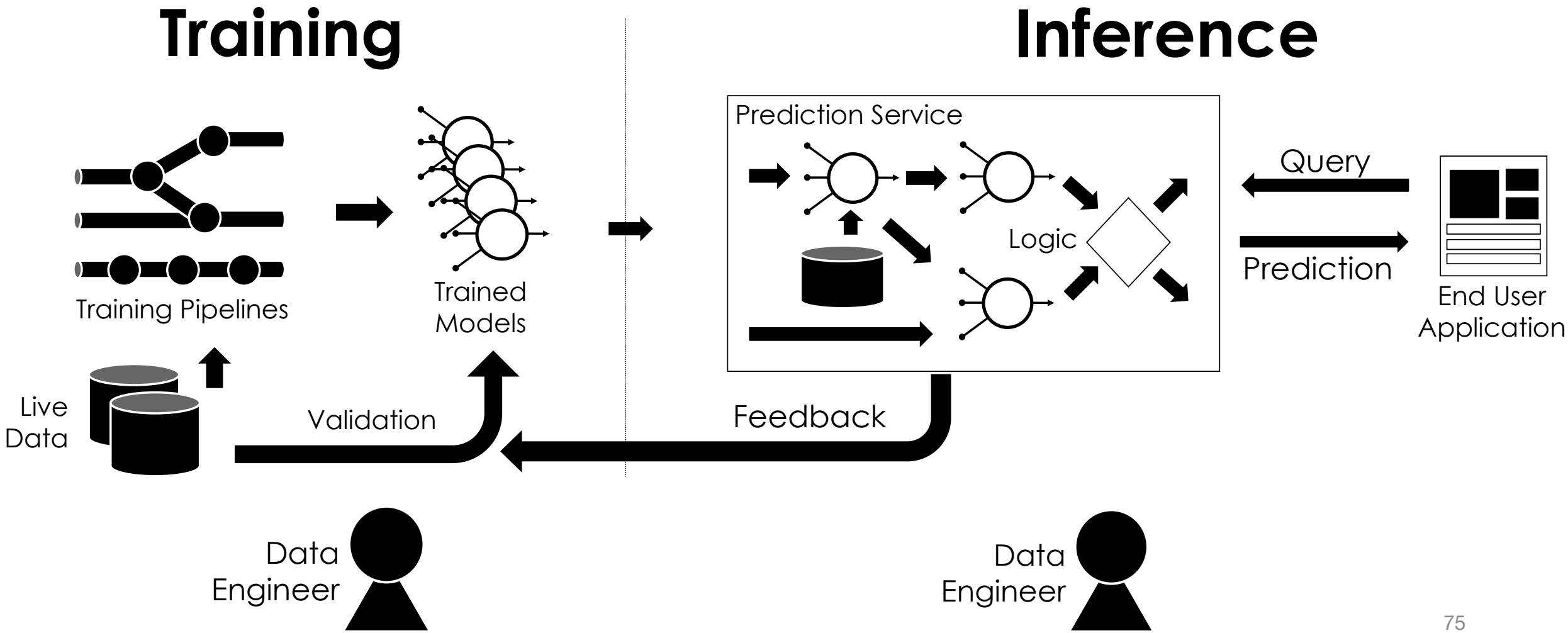


Application

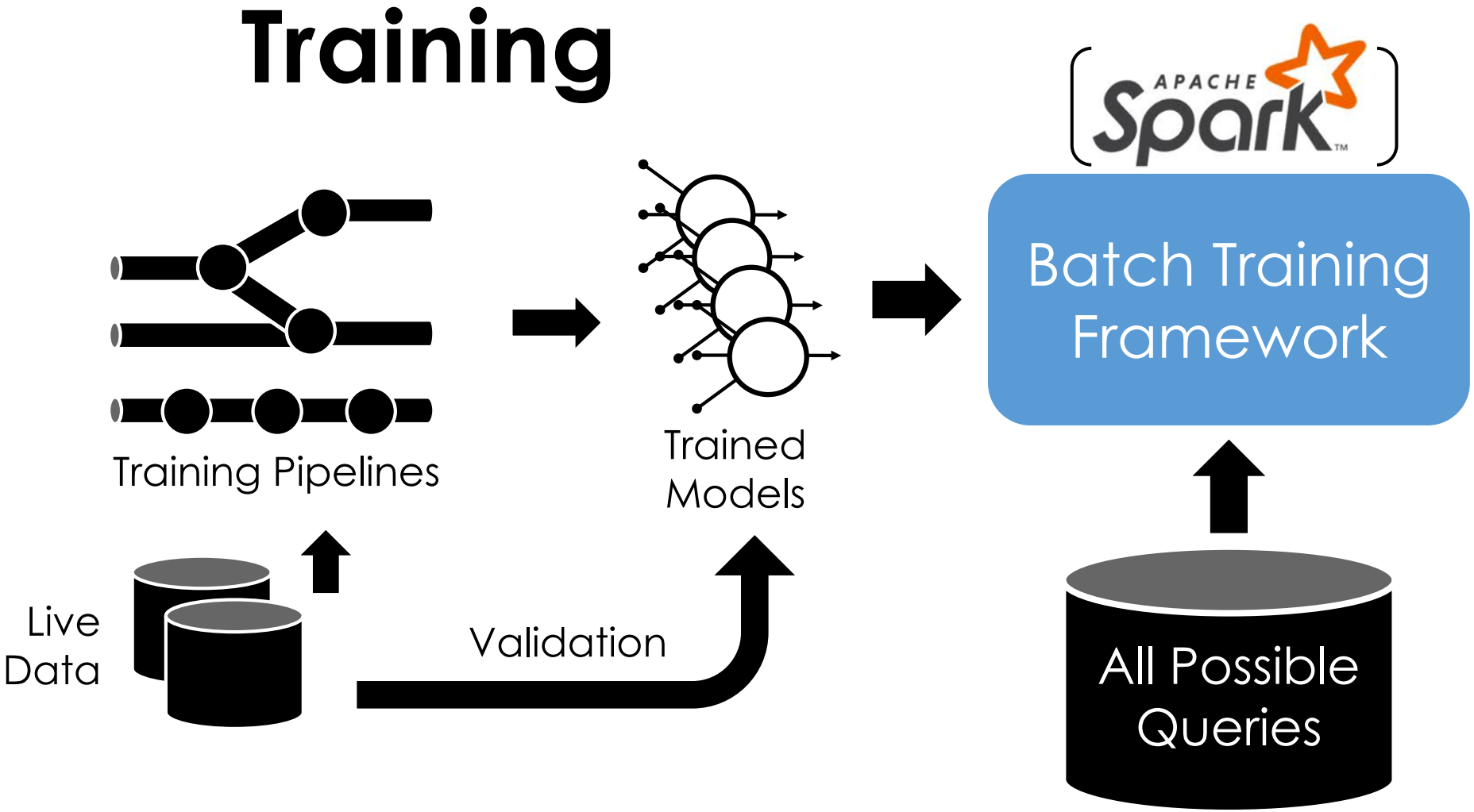
Two Approaches

- ***Offline:*** Pre-Materialize Predictions
- ***Online:*** Compute Predictions on the fly

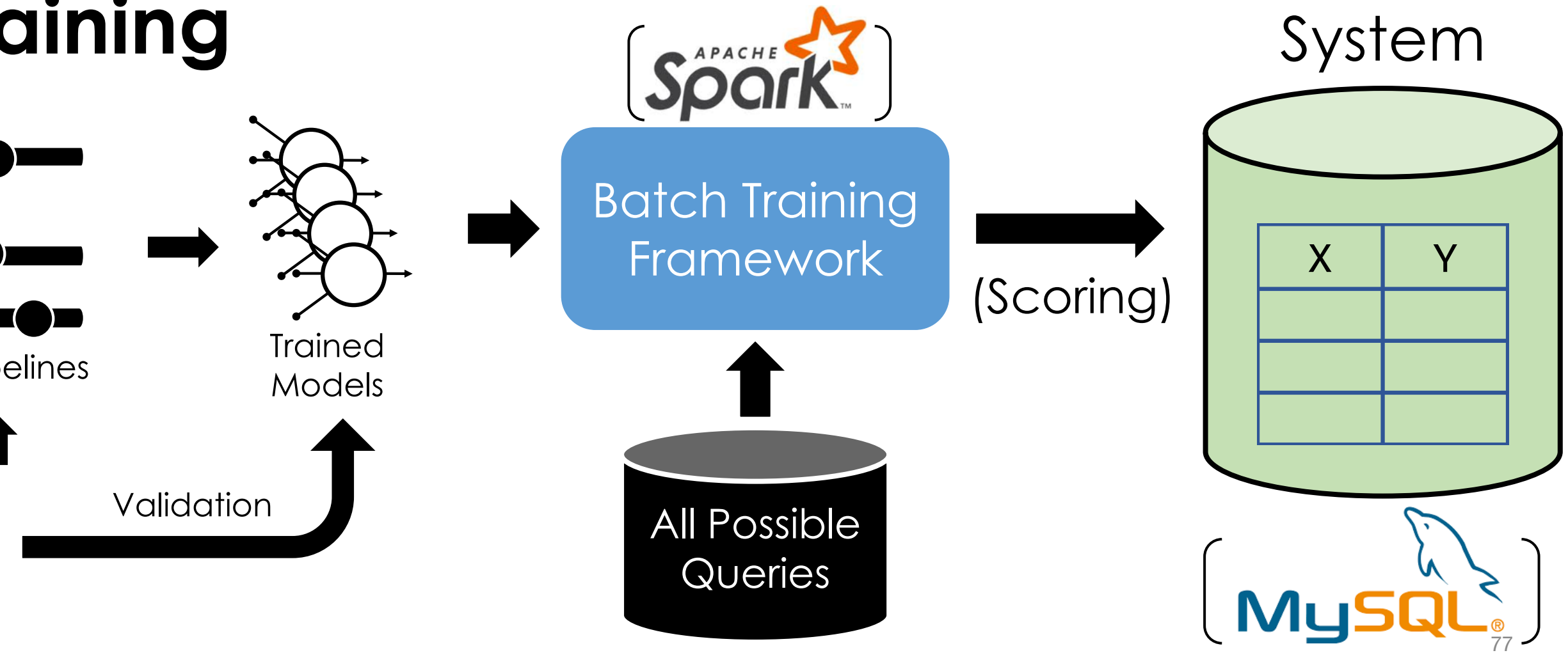
Pre-materialized Predictions



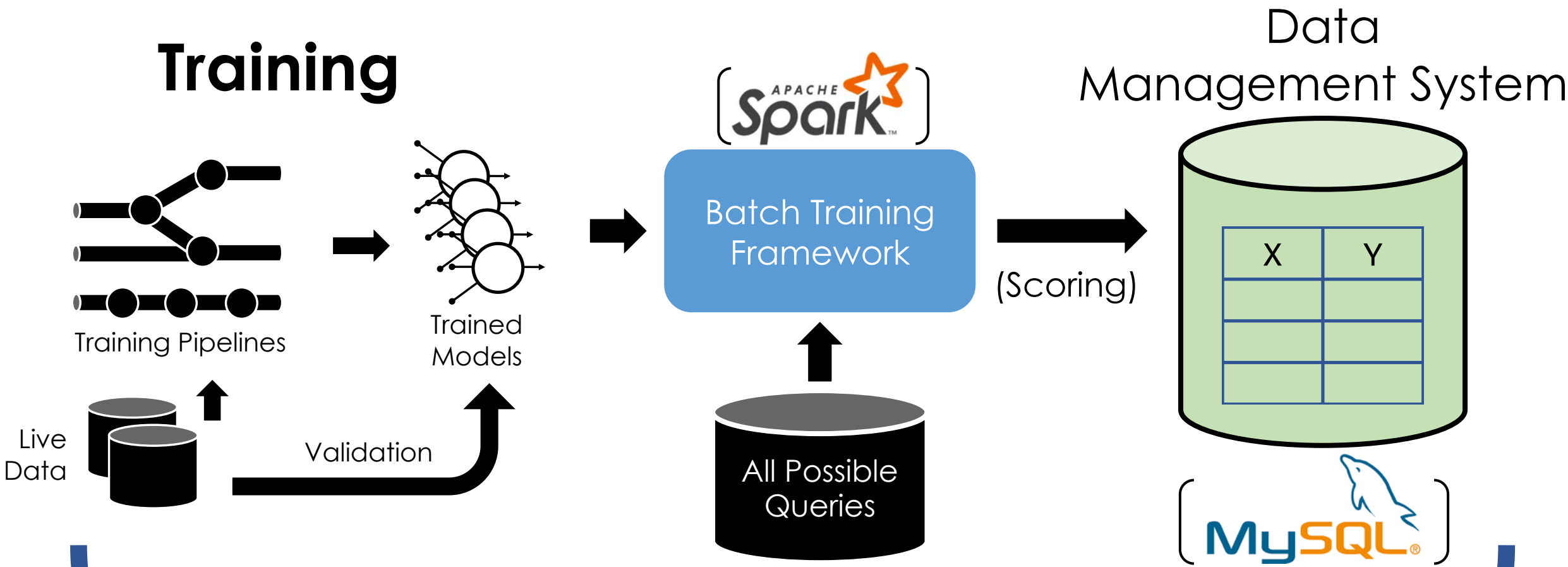
Pre-materialized Predictions



Pre-materialized Predictions

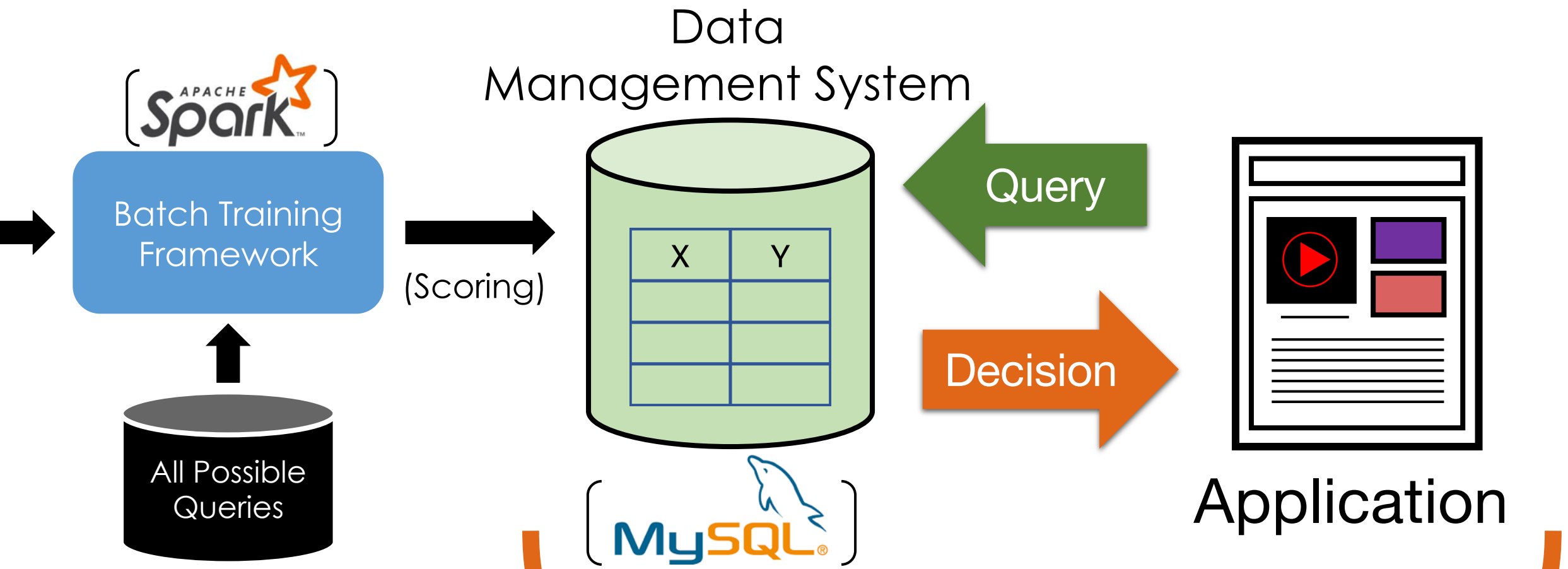


Pre-materialized Predictions



Standard Data Eng. Tools

Serving **Pre-materialized** Predictions



Serving **Pre-materialized** Predictions

Advantages:

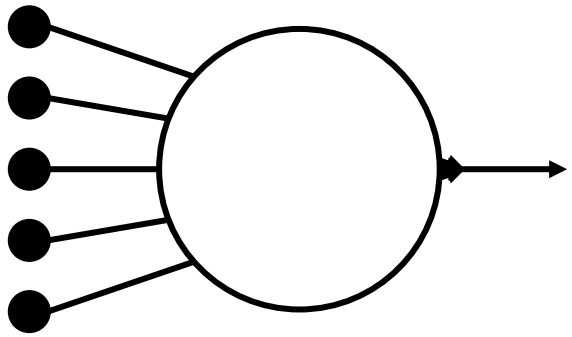
- Leverage existing **data serving** and **model training** infrastructure
- **Batch processing** improves hardware perf.
- **Indexing** support for complex queries
  - Find all ***Pr("cute")*** dresses **where price < $20**
- More **predictable** performance

Serving **Pre-materialized** Predictions

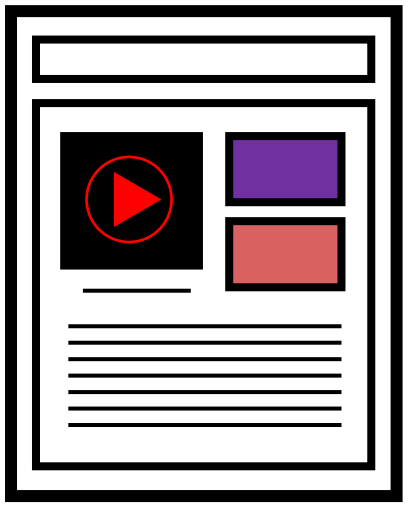
Problems:

- Requires full set of **queries ahead of time**
  - Small and **bounded input domain**
- Requires substantial **computation** and **space**
  - Example: *scoring all content for all customers!*
- Costly update → rescore everything!

Inference



Big Model

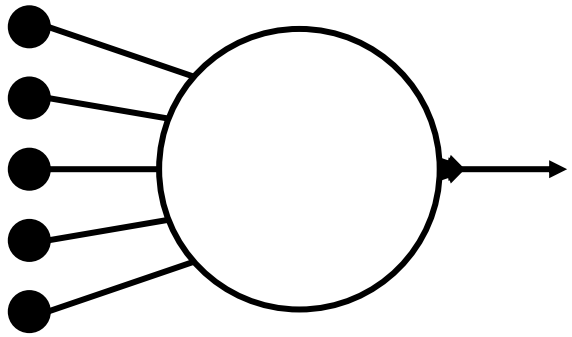


Application

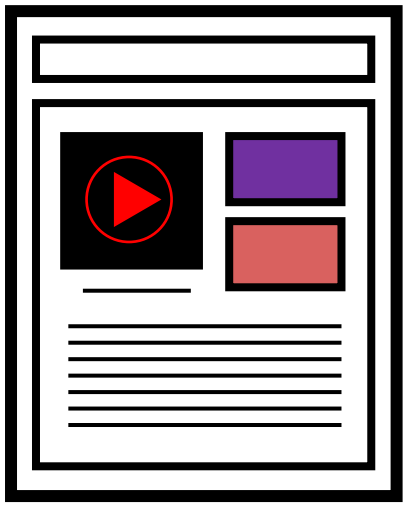
Two Approaches

- ***Offline:*** Pre-Materialize Predictions
- ***Online:*** Compute Predictions on the fly

Inference



Big Model

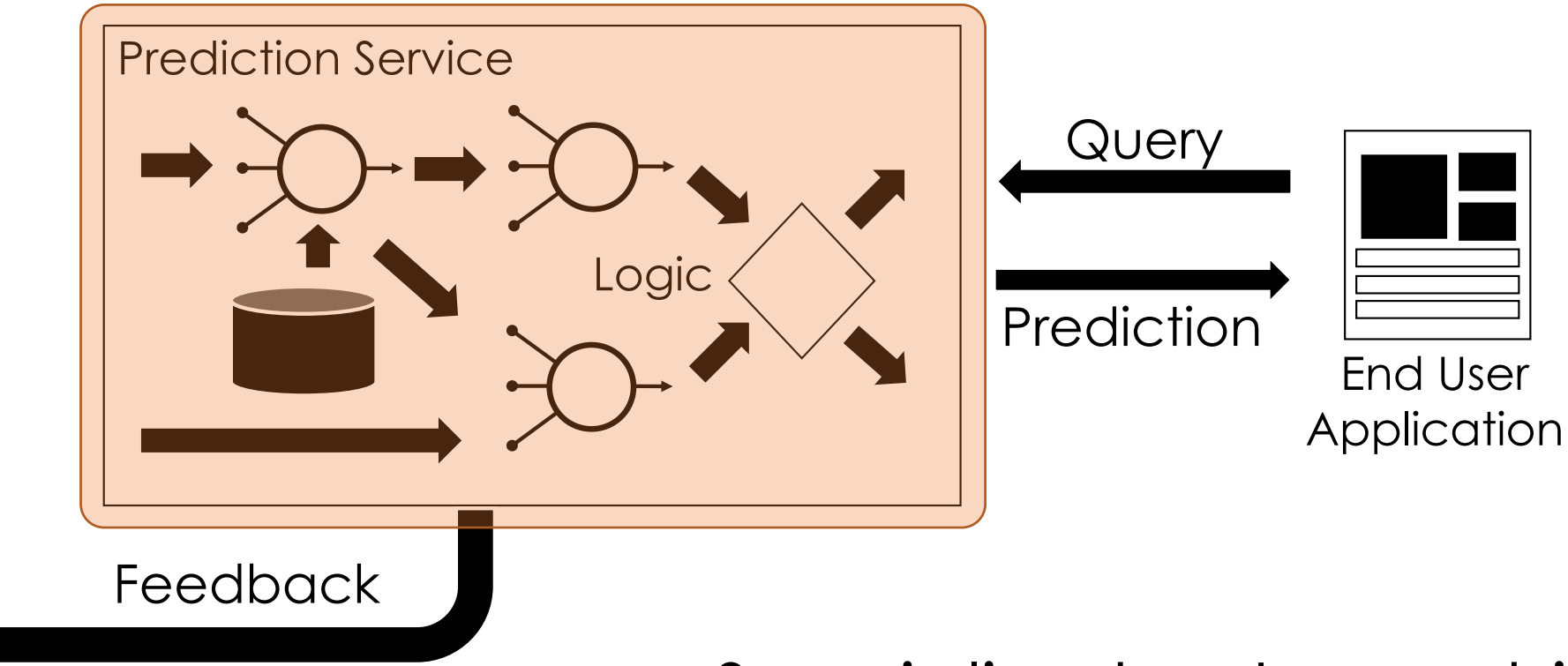


Application

Two Approaches

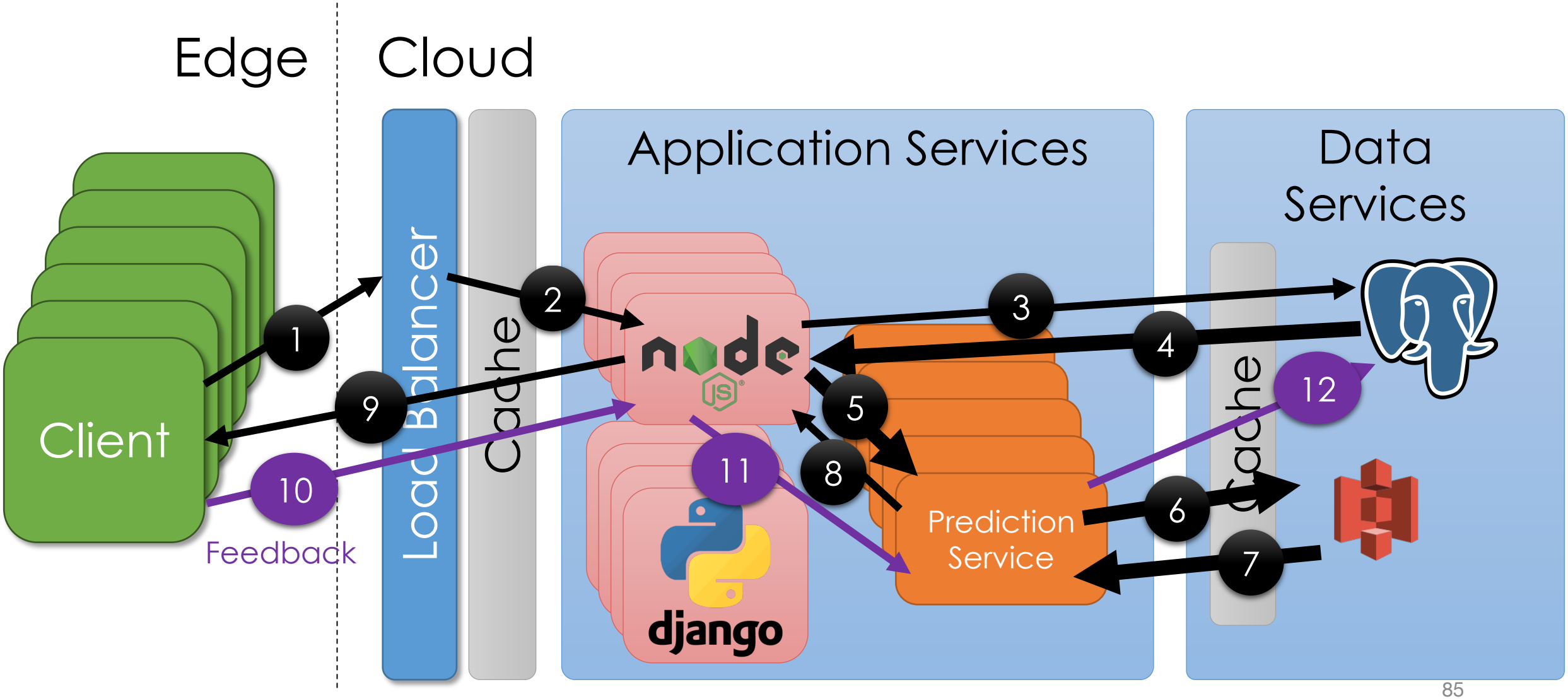
- ***Offline:*** Pre-Materialize Predictions
- ***Online:*** Compute Predictions on the fly

Prediction Services



Specialized systems which render predictions at **query time**.

Architecture of a Prediction Service

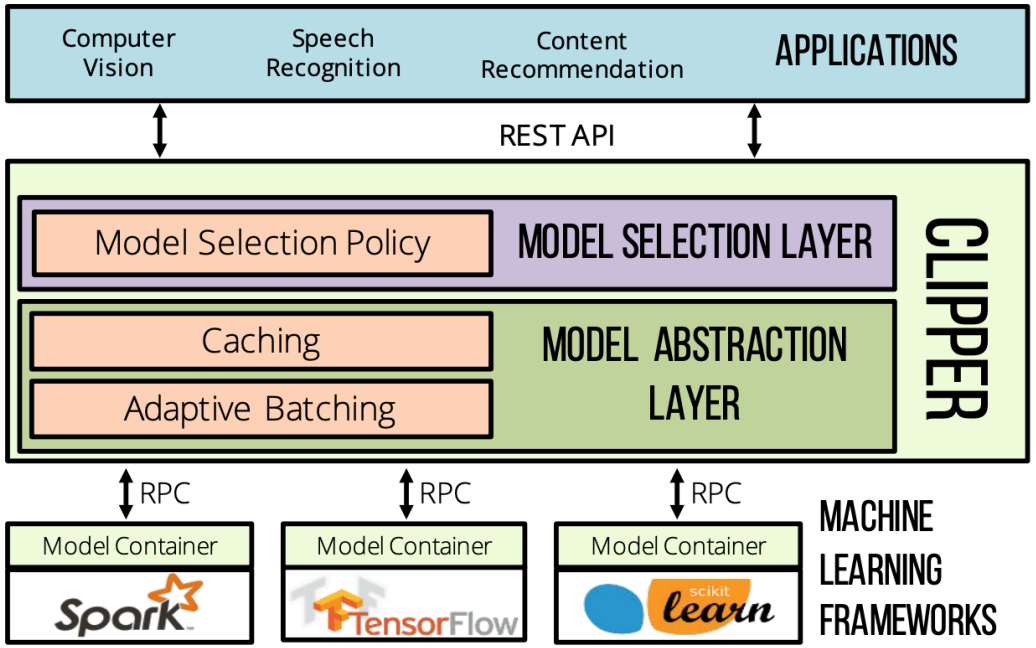


Online: Compute Predictions at Query Time

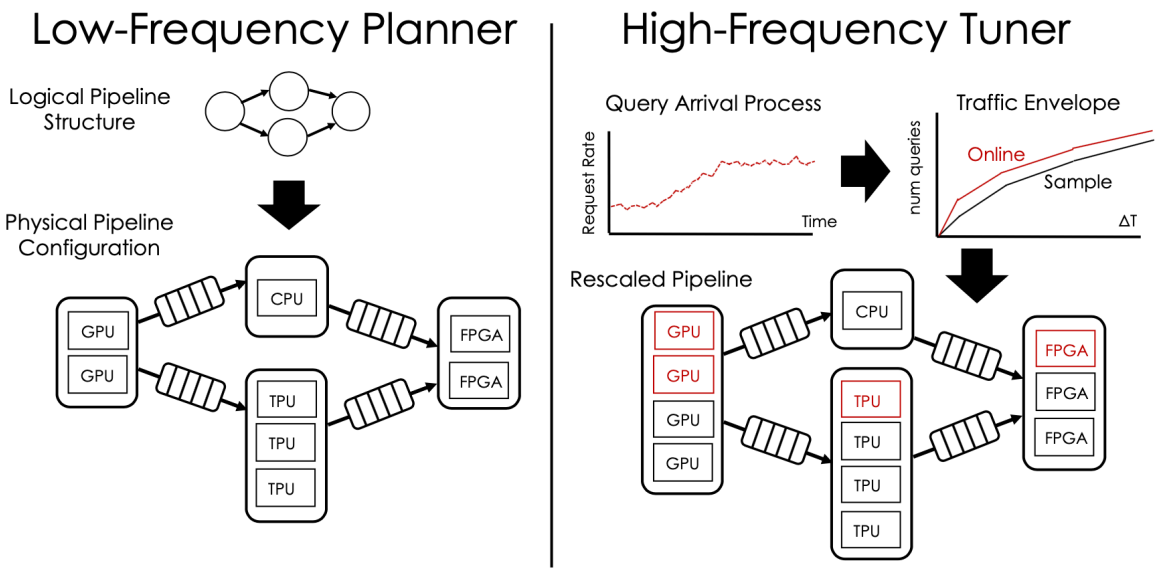
- **Examples**
  - Signals processing: speech recognition & image tagging
  - Ad-targeting based on search terms, available ads, user features
- **Advantages**
  - Compute only necessary queries
  - Enables models to be changed rapidly (e.g., bandit exploration)
  - Queries do not need to be from small ground set
- **Disadvantages**
  - Increases complexity and computation overhead of serving system
  - Requires low and predictable latency from models

Active Area of Research in my Group

Clipper Prediction Serving System
[NSDI'17]



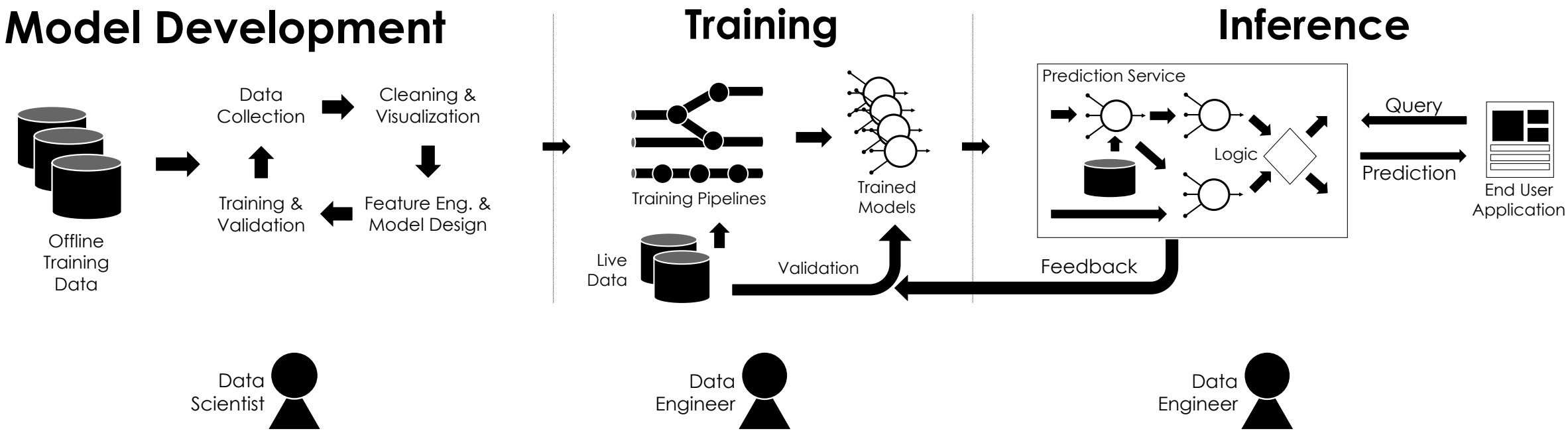
InferLine Pipeline Provisioning System
[Under review]

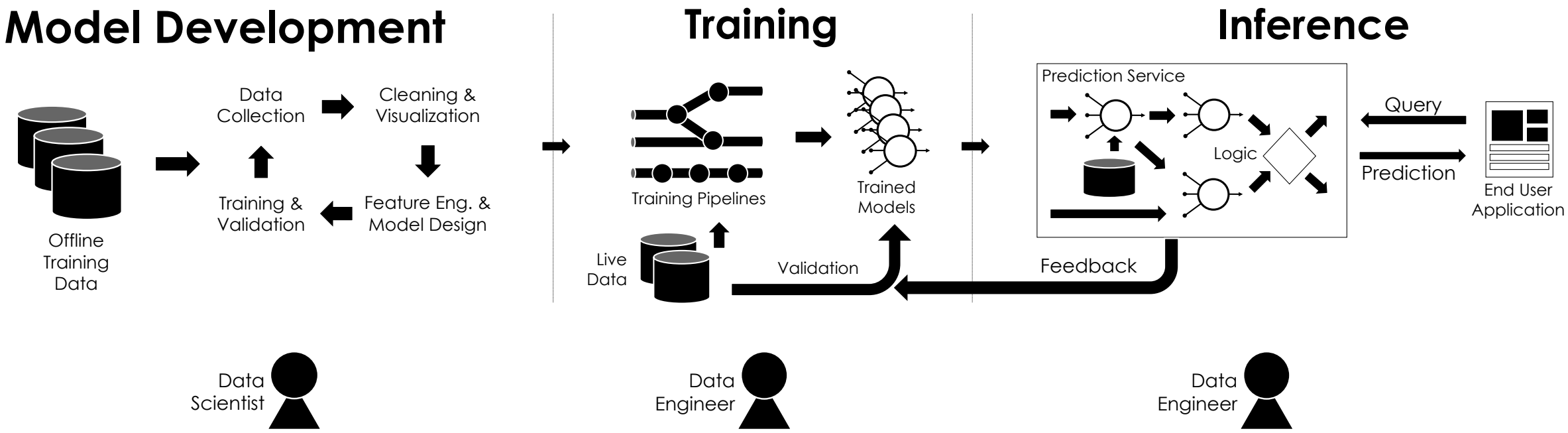


Prediction Serving → Hardware

- Inference requires less memory → **focus on compute**
- Greater emphasis on **latency** instead of throughput
  - Focus on **small batch** inference (batch size = 1)
  - Opportunity to exploit **pipeline parallelism**
  - Need **high availability** → esp. in mission critical settings
- Often runs multiple **concurrent prediction tasks**
  - Cloud → Multitenancy → Performance isolation
  - Edge → supporting multiple data streams
- Tolerate **model compression** and **quantization**
  - As low as 4-bit activations and weights
- **Bursty load**
  - Statistical multiplexing
  - Use inference hardware for background training?

Machine Learning Lifecycle




Model Development
Training
Inference
Data Collection
Cleaning & Visualization
Training & Validation
Feature Eng. & Model Design
Offline Training Data
Training Pipelines
Trained Models
Live Data
Validation
Prediction Service
Logic
Feedback
Query
Prediction
End User Application
Data Scientist
Data Engineer
Data Engineer

Security?

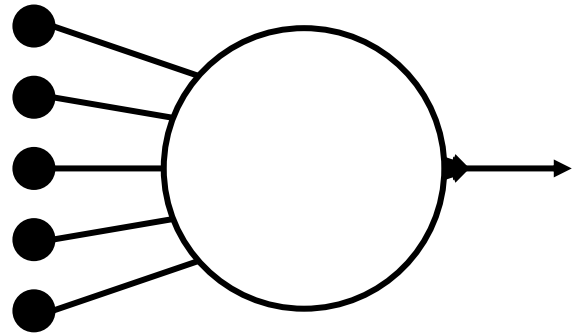
Protect the **data**, the **model**, and the **query**

High-Value **Data** *is* ***Sensitive***

- Medical Info.
- Home video
- Finance

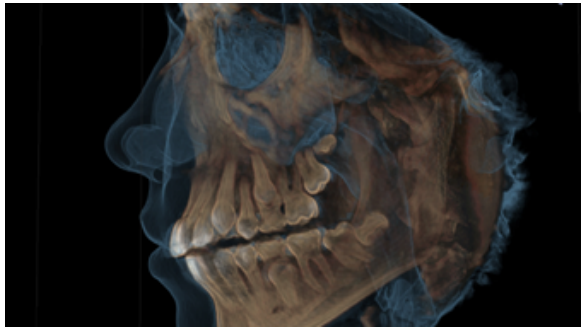
Models capture **value** in data

- Core Asset
- "Contain" the data



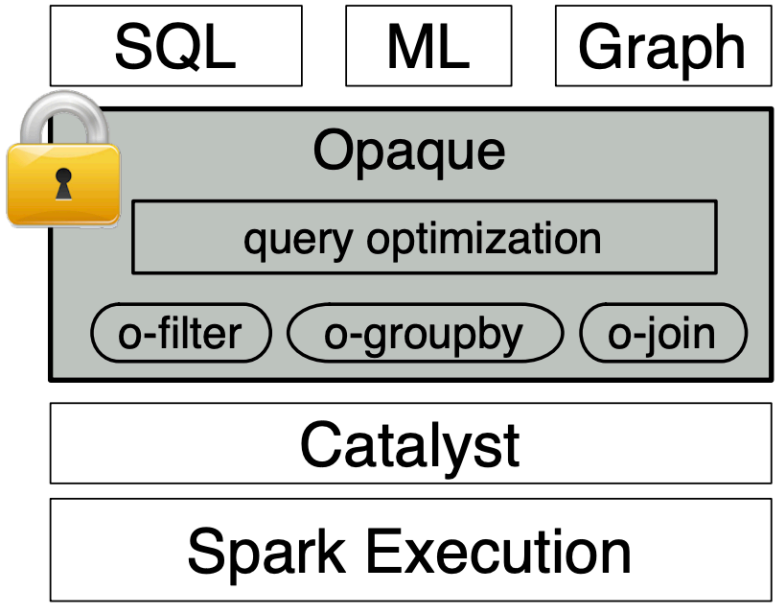
Biggest opportunity for hardware in ML.

Queries can be as sensitive as the data

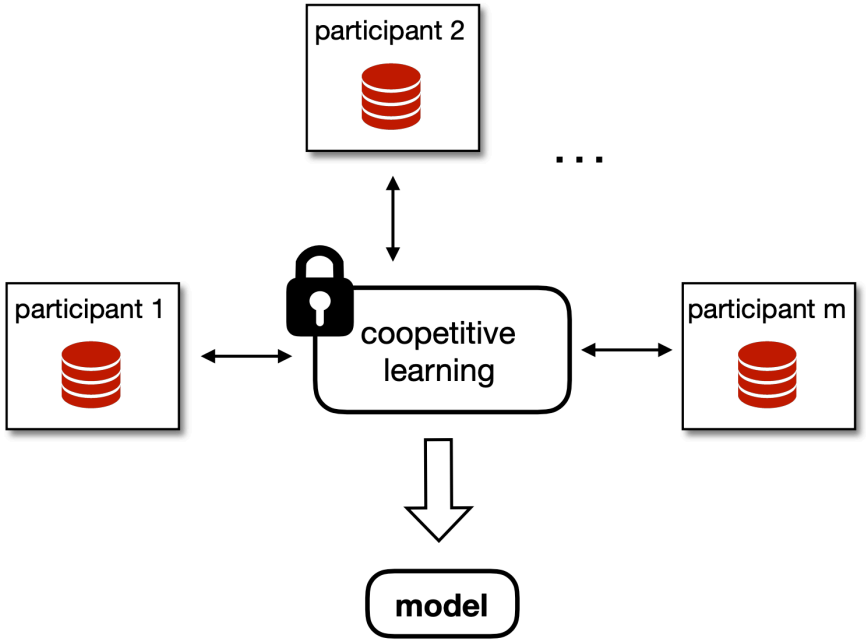


Our recent work in secure ML

Opaque [NSDI'17] | Oblivious Spark over **SGX**



Helen [S&P'19] | **Coopetitive Learning** Using Cryptographic Primitives



Security and Hardware

- Improved Access to Data
  - User willing to share data with models but **not companies (people)**
  - **Differential Privacy** can increase data sharing incentives
- Better **isolation** of co-tenant models on hardware accelerators
- **Coopetitive Learning**: Secure multiparty computation for ML
  - **Example:** *Competing banks collaborate to construct a shared fraud model without sharing data.*
- Models have access **to more sensitive inputs**
  - **Example:** *Alexa could see where you are when asking to turn on a light.*

Conclusion

- **History:** AI and Computer Systems have Co-evolved
- **Feedback Cycle:** Hardware, Abstractions, and Data
- **ML is many Applications:** *Machine Learning Lifecycle*
  - *Model Development:* Exploration
  - *Training:* Scale and Composition
  - *Inference:* Cloud – Edge Spectrum
- **Security:** Opportunity for hardware innovation in AI

Thank you!