PETUUM

SAILING LAB

Laboratory for Statistical Artificial InteLligence & INtegreative Genomics

A Civil Engineering Perspective on Artificial Intelligence From Petuum

Eric Xing

Petuum Inc.

Carnegie Mellon University

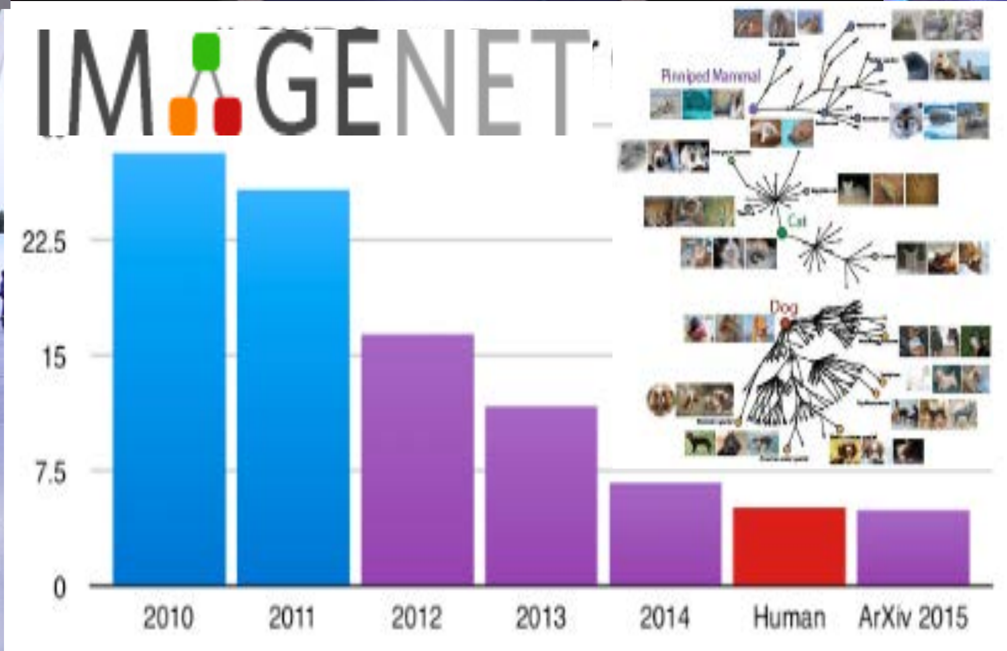
What is AI?


Boston Dynamics


AlphaGo
Lee Sedol

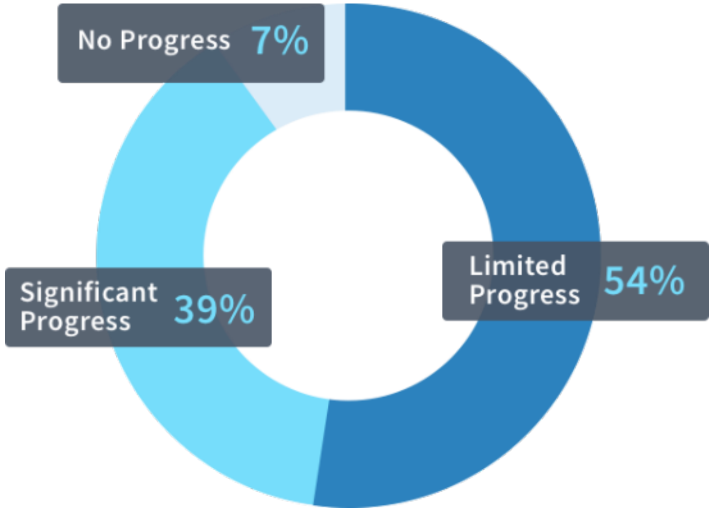

THINK
$300,000
$1,000,000
$200,000


GM
Continental
19

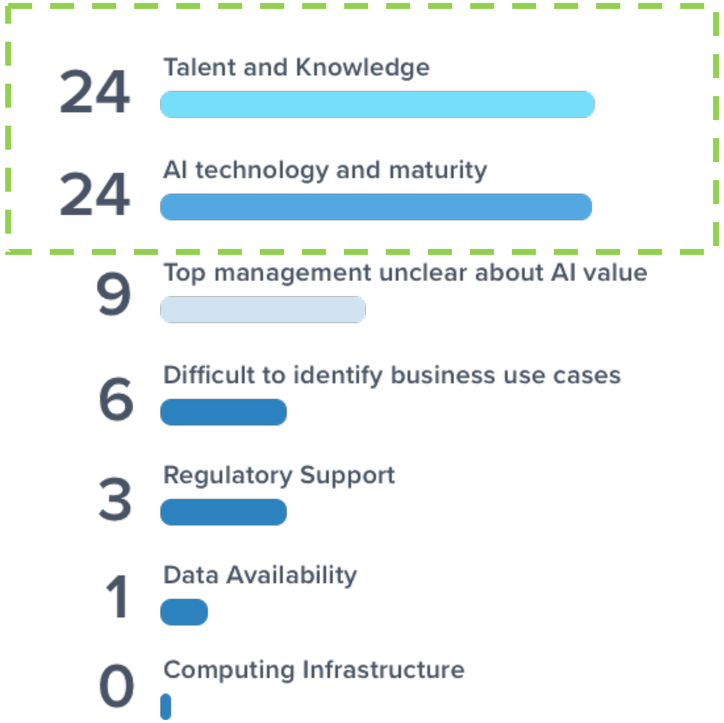

IMAGENET
22.5
15
7.5
0
2010
2011
2012
2013
2014
Human
ArXiv 2015
Dog
Cat

AI Solution Today – major hurdles, and few choices…

How much progress has your company made in the last year implementing AI applications?



What are the top barriers to AI Applications in your company?



Source: McKinsey CEO survey

Building is infeasible…

- ❌ **Talent** – Scare resources in data scientists, ML engineers. Sys engineers
- ❌ **Support** – Little or no enterprise support from open source software
- ❌ Major Infrastructure requirements
- ❌ Long development timelines and delivery risks

Lack of viable buy/rent options…

- ❌ Limited to cloud deployment
- ❌ Limited customization
- ❌ Limited service
- ❌ Limited scalability

Limited capabilities

… or not available at all!

A real ready-to-use AI solution is extremely complex

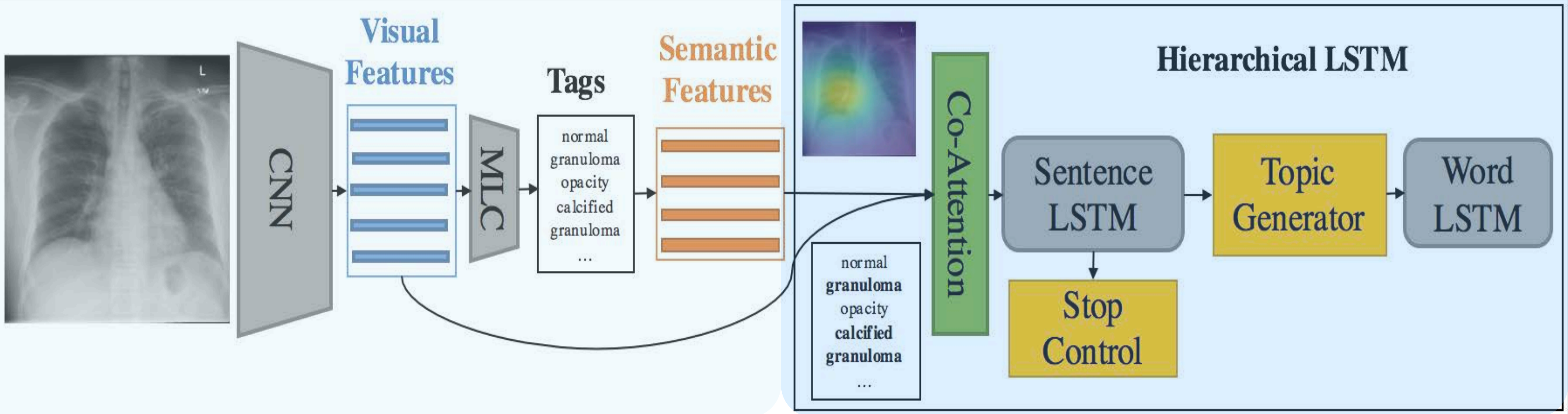
Use Case: Automatic Medical (or other) Report Generation

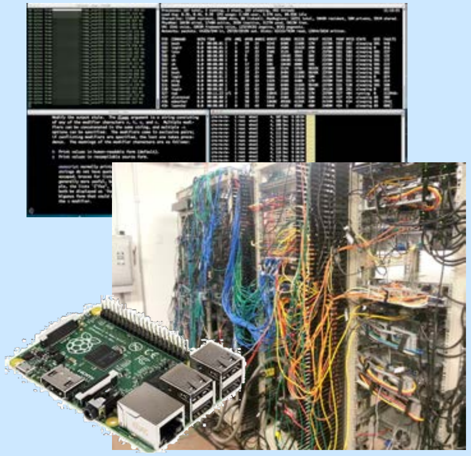
Findings:
There are no focal areas of consolidation.
No suspicious pulmonary opacities.
Heart size within normal limits.
No pleural effusions.
There is no evidence of pneumothorax.
Degenerative changes of the thoracic spine.
Impression:
No acute cardiopulmonary abnormality.

- Abnormal regions in medical images are difficult to identify.
- How to localize the image regions and tags that are relevant to a sentence?
- How to distribute topics across sentences
- How to make report readable to humans?

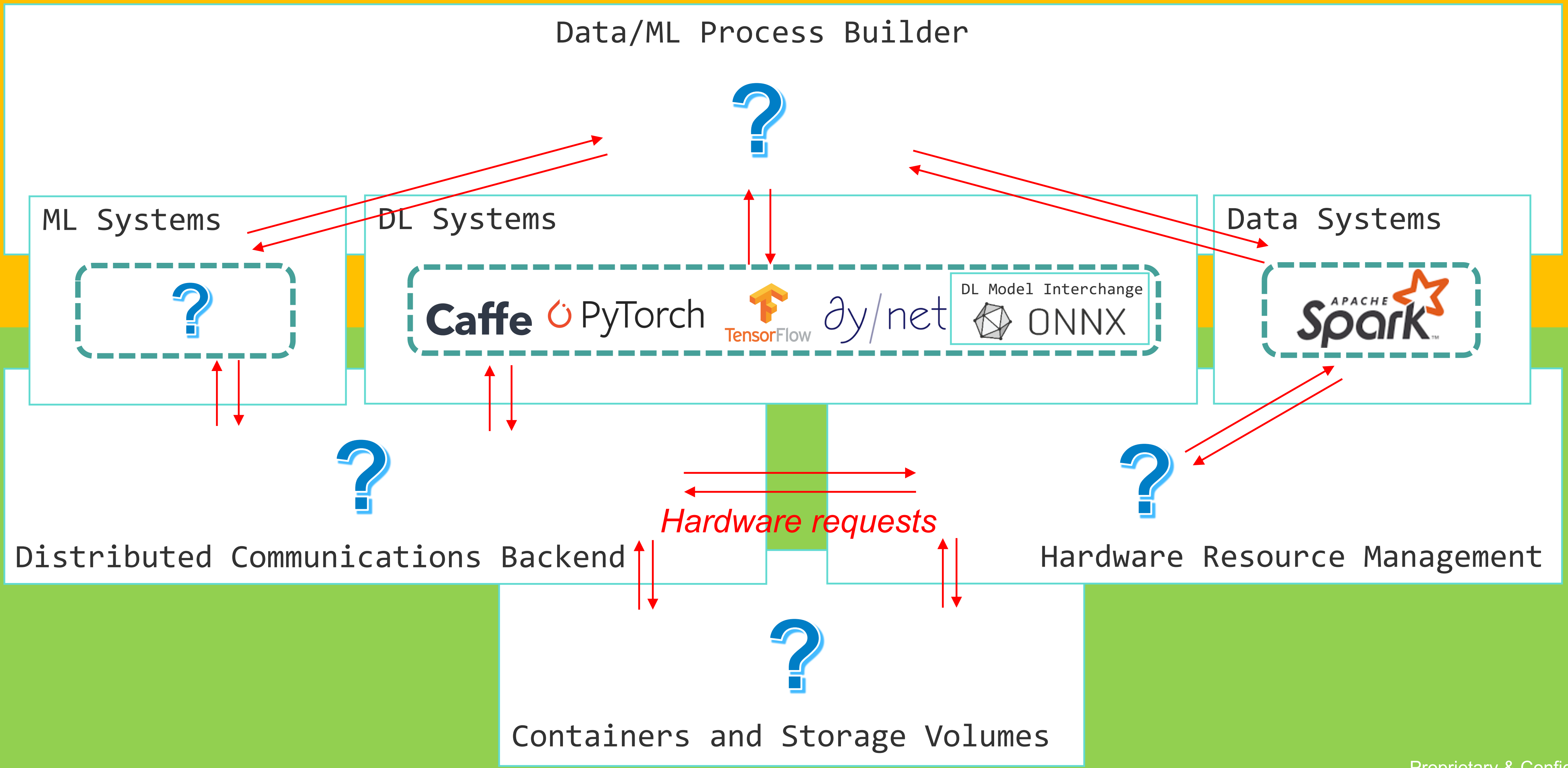
AutoML ?!

Raw Data Enrichment | Model/Algorithm | System/Infra





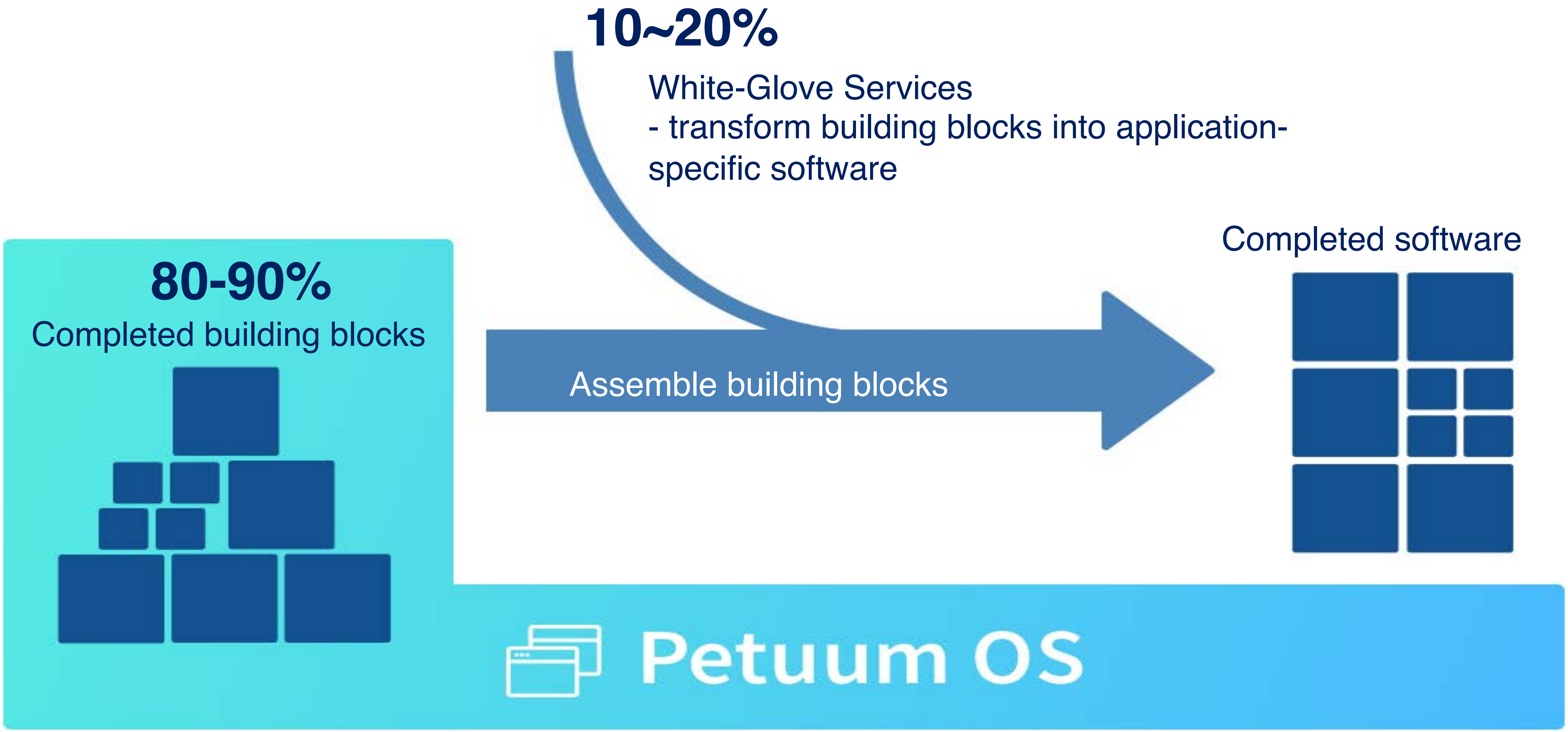
Inter-operability between diverse systems?



An AI solution

Data wrangling
Feature engineering
Model compiling
Algorithm designing
Distributed training
Debugging
Resource provisioning
Hardware management
Fault recovery
…etc

Petuum Vision: AI as “Civil Engineering”



Industry Agnostic

Industrial & utilities

Healthcare & insurance

Finance

Retail

Business

...

- Doctor-less X-Ray
- Insurance Auto-Report
- Virtual EA
- Smart Expense Reports
- Smart Catalog
- Robot Store Staff
- ...

Hardware Infrastructure

Mobile | Laptop/PC | Datacenter | Cloud | IoT/Edge

Key issues in enabling such a transformation

- First Principles
- The “Civil” Engineering
- Explain the process and outcome
- Analysis and safety under real operation
- Standards, mass production, cost amortization

Key issues in enabling such a transformation

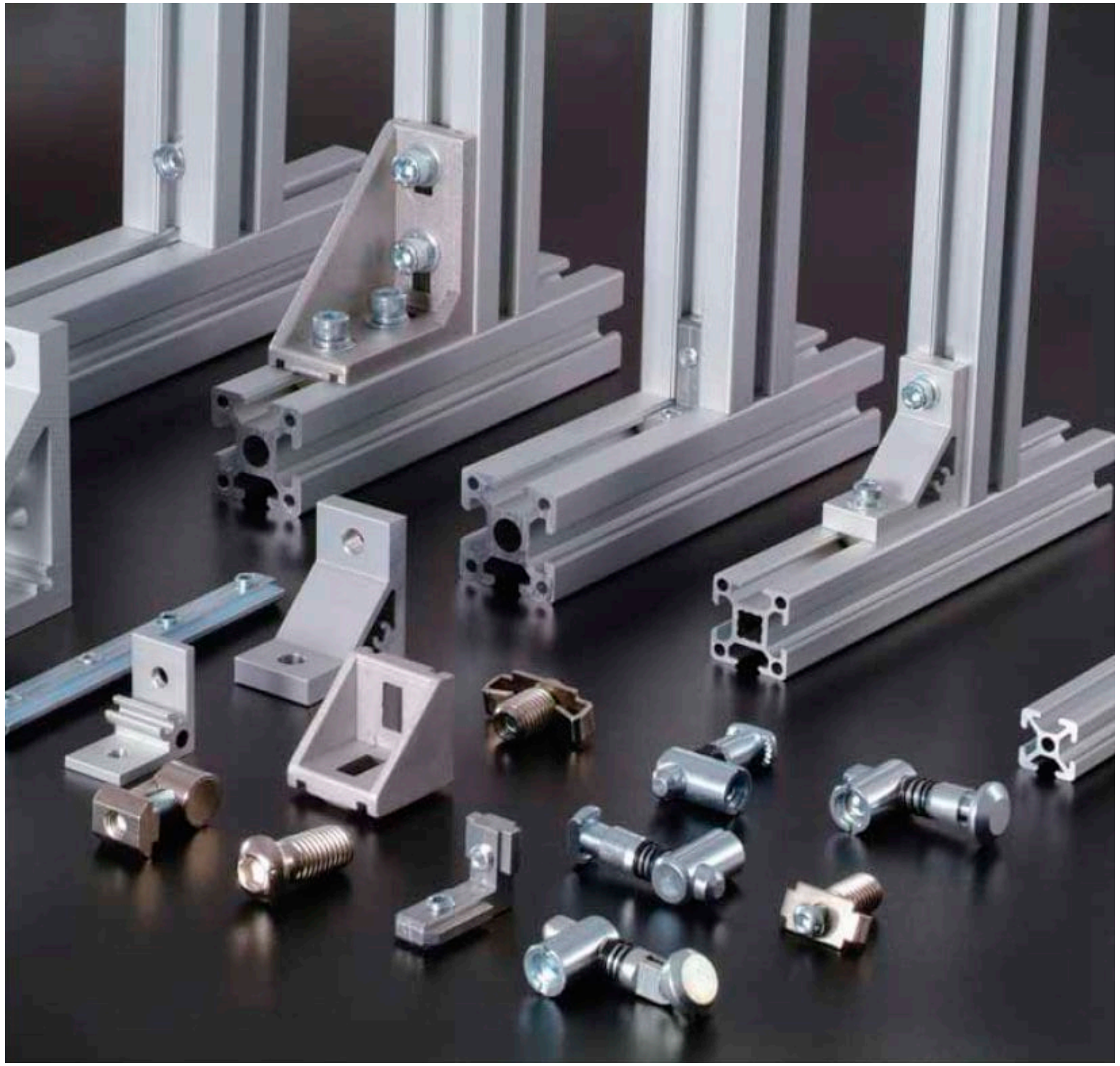
- First Principles
- **The "Civil" Engineering**
- Explain the process and outcome
- Analysis and safety under real operation
- Standards, mass production, cost amortization

Build versus Craft

- Modules, Building-blocks
- Nuts and Bolts
- Interoperability
- Process
- Soundness

AI as an *engineering production process*

- Like **Civil Engineering**, via an assembly-line-like building process, the new generation of AI programs should be:
  - **Modular**
  - **Standardized**
  - **Reusable**
  - **Inter-Operable**



An ML program

$$\underset{\vec{\theta}}{\arg\max} \equiv \mathcal{L}(\{\mathbf{x}_i, \mathbf{y}i\}_{i=1}^{N} \;;\; \vec{\theta}) + \Omega(\vec{\theta})$$

Model Data Parameter

Solved by an iterative convergent algorithm

```
for (t = 1 to T) {
  doThings()
```

$$\vec{\theta}^{t+1} = g(\vec{\theta}^{t}, \Delta_f \vec{\theta}(\mathcal{D}))$$

```
  doOtherThings()
}
```

This computation needs to be parallelized!

Proximal gradient (a.k.a., ISTA)

$$\min_{\mathbf{w}} \ell(\mathbf{w}) + r(\mathbf{w})$$

- ℓ : loss, for now smooth (continuously differentiable)
- r : regularizer, non-differentiable (e.g. 1-norm)

Projected gradient

- *r* represents some constraint

$$r(\mathbf{w}) = \iota_C(\mathbf{w}) = \begin{cases} 0, & \mathbf{w} \in C \\ \infty, & \text{otherwise} \end{cases}$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla \ell(\mathbf{w})$$

$$\mathbf{w} \leftarrow \arg\min_{\mathbf{z}} \frac{1}{2\eta} \|\mathbf{w} - \mathbf{z}\|^2 + \iota_C(\mathbf{z})$$

$$= \arg\min_{\mathbf{z} \in C} \frac{1}{2} \|\mathbf{w} - \mathbf{z}\|^2$$

Proximal gradient

- *r* represents some simple function
  - e.g., 1-norm, constraint C, etc.

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla \ell(\mathbf{w}) \quad \text{gradient}$$

$$\mathbf{w} \leftarrow \underbrace{\arg\min_{\mathbf{z}} \frac{1}{2\eta} \|\mathbf{w} - \mathbf{z}\|^2 + r(\mathbf{z})}_{\text{proximal map}}$$

Accelerated PG (a.k.a. FISTA)

- PG convergence rate $O(1/(\eta t))$
- Can be boosted to $O(1/(\eta t^2))$
  - Same Lipschitz gradient assumption on f; similar per-step complexity!
  - (Beck & Teboulle'09; Nesterov'13; Tseng'08), lots of follow-up work

Proximal Gradient

$$\mathbf{v}^t \leftarrow \mathbf{w}^t - \eta \nabla \ell(\mathbf{w}^t)$$
$$\mathbf{u}^t \leftarrow \mathsf{P}_r^\eta(\mathbf{v}^t)$$
$$\mathbf{w}^{t+1} \leftarrow \mathbf{u}^t + \underbrace{0}_{no} \cdot \underbrace{(\mathbf{u}^t - \mathbf{u}^{t-1})}_{momentum}$$

Accelerated Proximal Gradient

$$\mathbf{v}^t \leftarrow \mathbf{w}^t - \eta \nabla \ell(\mathbf{w}^t)$$
$$\mathbf{u}^t \leftarrow \mathsf{P}_r^\eta(\mathbf{v}^t)$$
$$\mathbf{w}^{t+1} \leftarrow \mathbf{u}^t + \underbrace{\frac{t-1}{t+2}}_{\approx 1} \underbrace{(\mathbf{u}^t - \mathbf{u}^{t-1})}_{momentum}$$

$$\mathsf{P}_r^\eta(\mathbf{w}) := \arg\min_{\mathbf{z}} \tfrac{1}{2\eta} \|\mathbf{w} - \mathbf{z}\|_2^2 + r(\mathbf{z})$$

Smoothing proximal gradient

- Use Moreau envelope as smooth approximation
  - Rich and long history in convex analysis (Moreau'65; Attouch'84)
- Inspired by the proximal point algorithm (Martinet'70; Rockafellar'76)
  - Proximal point alg = PG, when $f \equiv 0$
- Rediscovered in (Nesterov'05), lead to SPG (Chen et al.'12)

$$\min_{\mathbf{w}} f(\mathbf{w}) + g(\mathbf{w})$$ ⇦ original

approx. ⇨ $$\approx \min_{\mathbf{w}} \mathsf{M}_f^\eta(\mathbf{w}) + g(\mathbf{w})$$

- With $\eta = O(1/t)$, SPG converges at $O(1/(\eta t^2)) = O(1/t)$
- Improves subgradient $O(1/\sqrt{t})$
- Requires both efficient P_f^η and P_g^η

Smoothing Proximal Gradient

$$\mathbf{v}^t \leftarrow \overbrace{\mathbf{w}^t - \eta \nabla \mathsf{M}_f^\eta(\mathbf{w}^t)}^{=\mathsf{P}_f^\eta(\mathbf{w}^t)}$$

$$\mathbf{u}^t \leftarrow \mathsf{P}_g^\eta(\mathbf{v}^t)$$

$$\mathbf{w}^{t+1} \leftarrow \mathbf{u}^t + \frac{t-1}{t+2} \underbrace{(\mathbf{u}^t - \mathbf{u}^{t-1})}_{momentum}$$

Data-Parallel for large-scale problems

- Model (e.g. SVM, Lasso …):

$$\min_{\mathbf{a}\in\mathbb{R}^d} \mathcal{L}(\mathbf{a}, D), \quad \text{where} \ \ \mathcal{L}(\mathbf{a}, D) = f(\mathbf{a}, D) + g(\mathbf{a})$$

data D, model a

- Algorithm:
  - Update

$$\mathbf{a}(t) := \text{prox}_g\left(\mathbf{a}^p(t) - \eta(t)\sum_{(p',t')\in Recv^p(t)} \Delta(\mathbf{a}^{p'}(t'), D_{p'})\right)$$

sub-update

proximal step wrt g

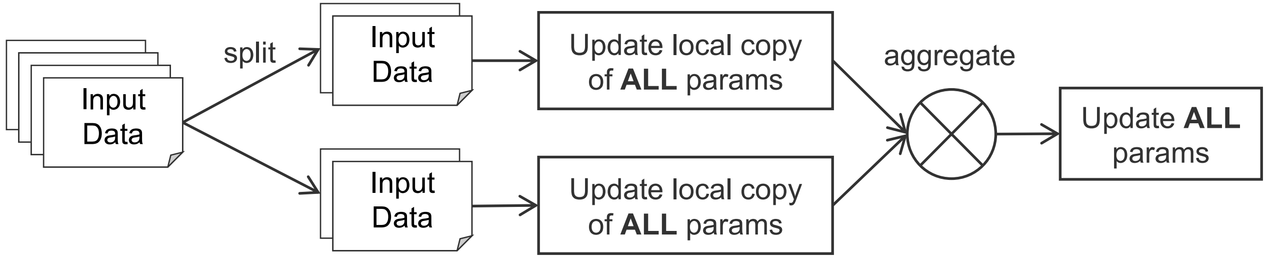
stale sub-updates Δ() received by worker p at iteration t

  - sub-update

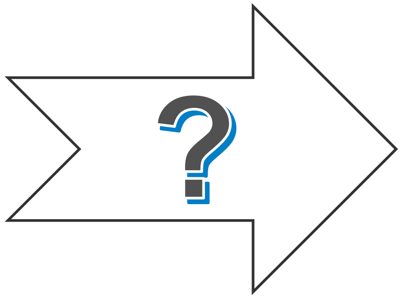
$$\Delta(\mathbf{a}^p(t), D_p) := \nabla f(\mathbf{a}^p(t), D_p)$$

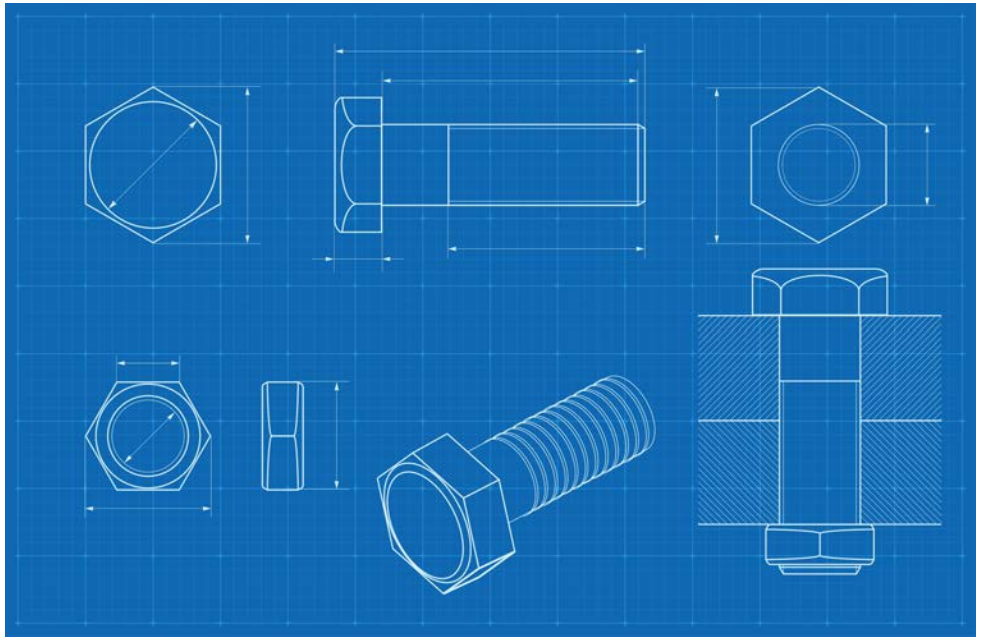
gradient step wrt f

- Data parallel:
  - Data D too large to fit in a single worker, divide among P workers



❑ How to modularize and standardize these steps/elements?





Nuts and Bolts: complete, reusable, robust

- Data wrangling
- Machine learning
- System harmonization

Examples

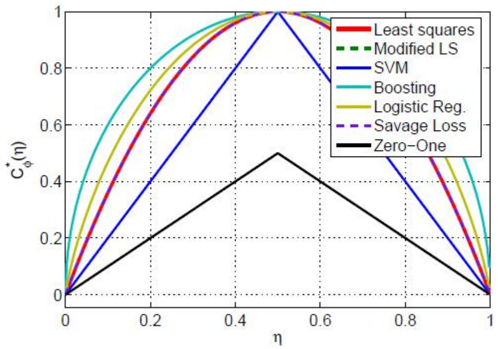
Data Transforms and Feature Engineering

- PCA, Sliding Window, n-th order Derivatives, Discretization, SIFT, Wavelets, Neural Network Embedding, …

Machine Learning

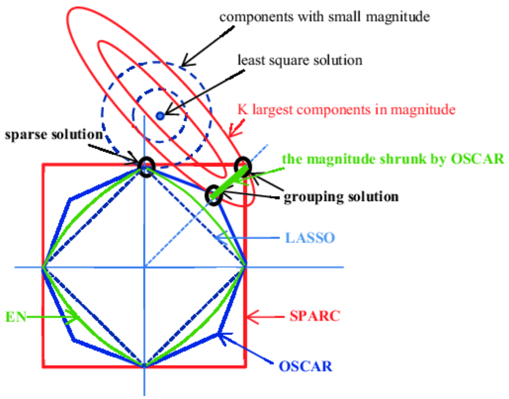
$$\arg\max_{\vec{\theta}} \equiv \mathcal{L}(\{\mathbf{x}_i, \mathbf{y}i\}_{i=1}^{N} \; ; \; \vec{\theta}) + \Omega(\vec{\theta})$$

Loss Functions



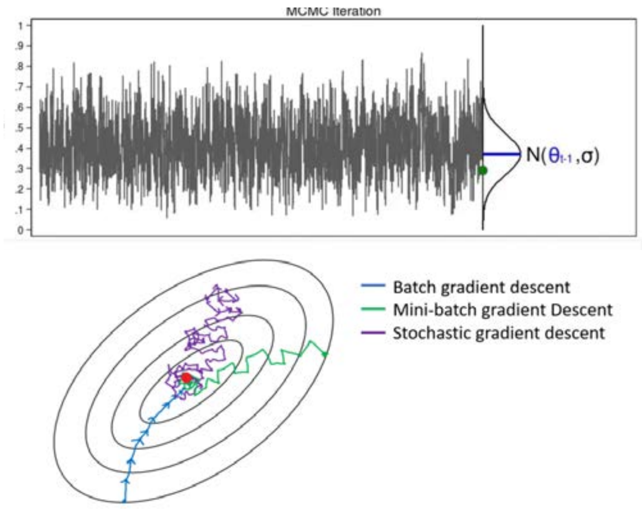
$$\mathcal{L}(\{\mathbf{x}_i, \mathbf{y}i\}_{i=1}^{N} \; ; \; \vec{\theta})$$

Regularizers/Priors



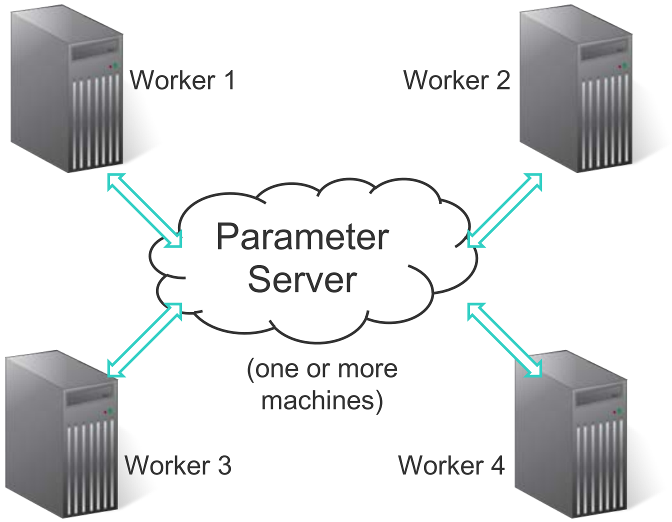
$$\Omega(\vec{\theta})$$

Training Algorithms



Examples cont.

- Distributed ML via SSP Parameter Server
- Interoperable – no change/massaging of ML algo implementation!
  - Simply call different subroutine/interface and easily switch to distributed operation

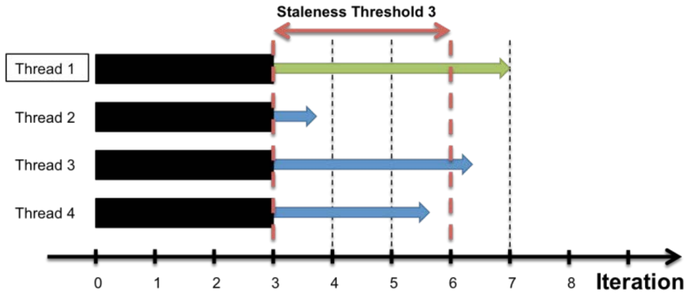


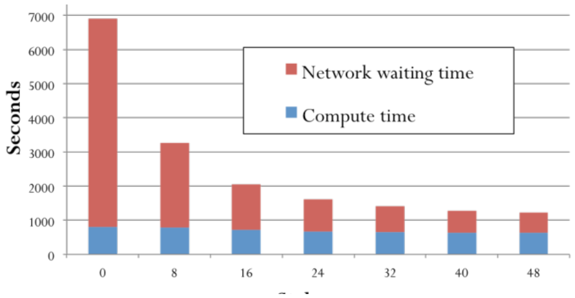
Single Machine Parallel

```
UpdateVar(i) {
  old = y[i]
  delta = f(old)
  y[i] += delta
}
```

Distributed with Bösen PS

```
UpdateVar(i) {
  old = PS.read(y,i)
  delta = f(old)
  PS.inc(y,i,delta)
}
```





An inventory of ML/Sys nuts and bolts

Data Machine	Ingestion	Cleaning & Improvement	Feature Engineering	Embedding	Integration	Interpret (DM X-Ray)	Dev	Resource
Machine Learning	Build	Augment and Tune	Train	Score	Experiment	Interpret (Model X-ray)	Dev	Resource
Operating System	Deploy	Storage	Compatibility	User Accounts and Security	Project Management	Interpret (Process X-ray)	Dev	Resource

Not just DL or a bag of classifiers

Full library of advanced methods

- Supervised ML
- Unsupervised ML
- Bayesian Methods
- Regularization Methods
- Latent Space Embedding and Extraction
- Content (image/text/video) Generation
- Reinforcement Learning
- Imitational Learning
- Active Learning
- Transfer Learning
- Meta-learning (learning to learn)
- Deep Learning
- Deep Learning + human logic + interpretability
- …

Make all data useful for AI

- Dark & unstructured data
- 10X more data access (10% -> 100%)

Enable analytics & prediction

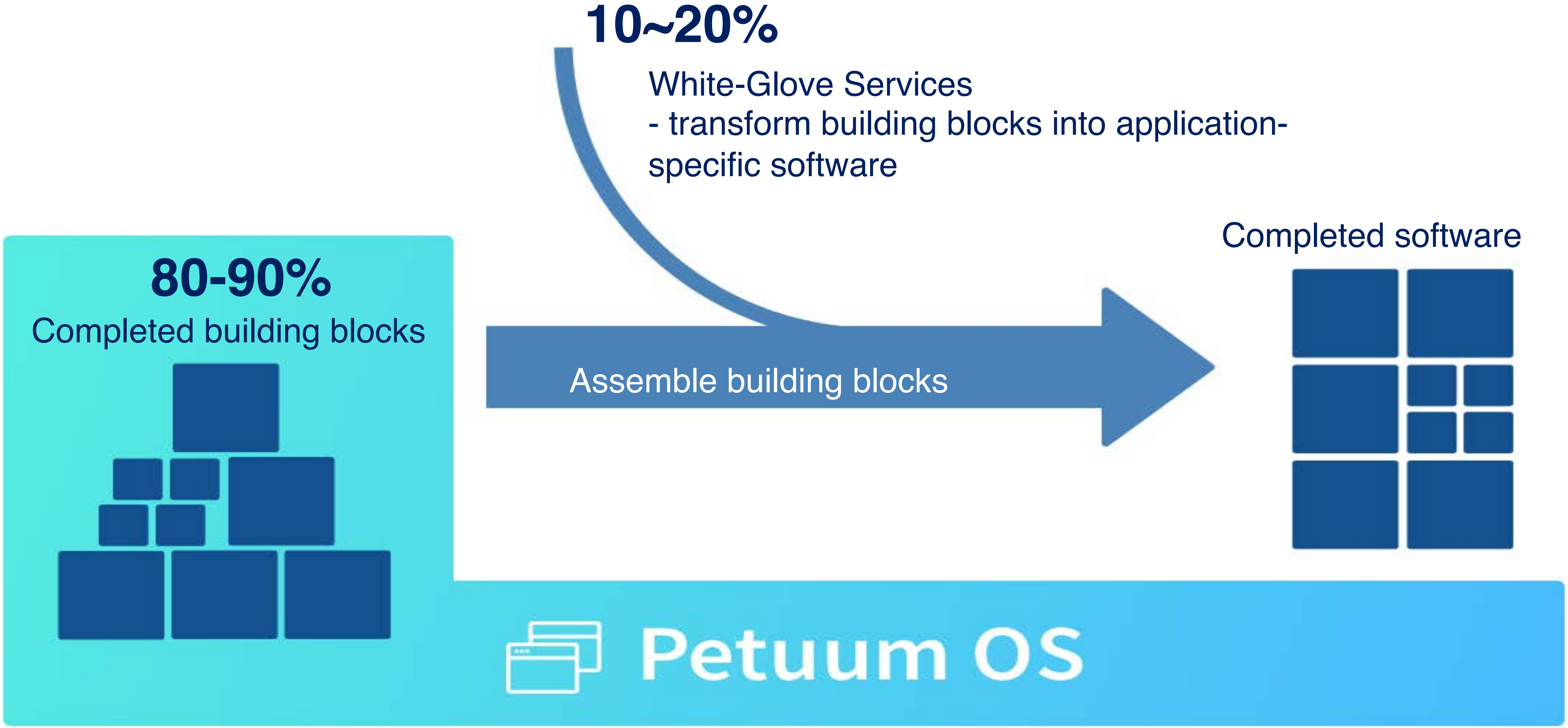
- Decisions in uncertainty
- More and better decisions

Multi-AI collaboration for complex tasks

Automatic AI creation and maintenance

Employ AI in mechanical robots

AI as “Civil Engineering”



Texar: a toolkit for Text Generation

Texar: A Modularized, Versatile, and Extensible Toolkit for Text Generation
Zhiting Hu, Haoran Shi, Zichao Yang, Bowen Tan,Tiancheng Zhao, Junxian He, Wentao Wang, Xingjiang Yu, Lianhui Qin, DiWang, Xuezhe Ma, Hector Liu, Xiaodan Liang, Wanrong Zhu, Devendra SinghSachan, Eric P. Xing

Text Generation Tasks

- Generates *natural language* from input *data or machine representations*
- Spans a broad set of natural language processing (NLP) tasks:

Task	Input *X*	Output *Y* (Text)
Chatbot / Dialog System	Utterance	Response
Machine Translation	English	Chinese
Summarization	Document	Short paragraph
Description Generation	Structured data	Description
Captioning	Image/video	Description
Speech Recognition	Speech	Transcript

Courtesy: Neubig, 2017

Various (Deep Learning) Techniques

Models / Algorithms

- Neural language models
- Encoder-decoders
- Seq/self-Attentions
- Memory networks
- Adversarial methods
- Reinforcement learning
- Structured supervision
- …

Other Techniques

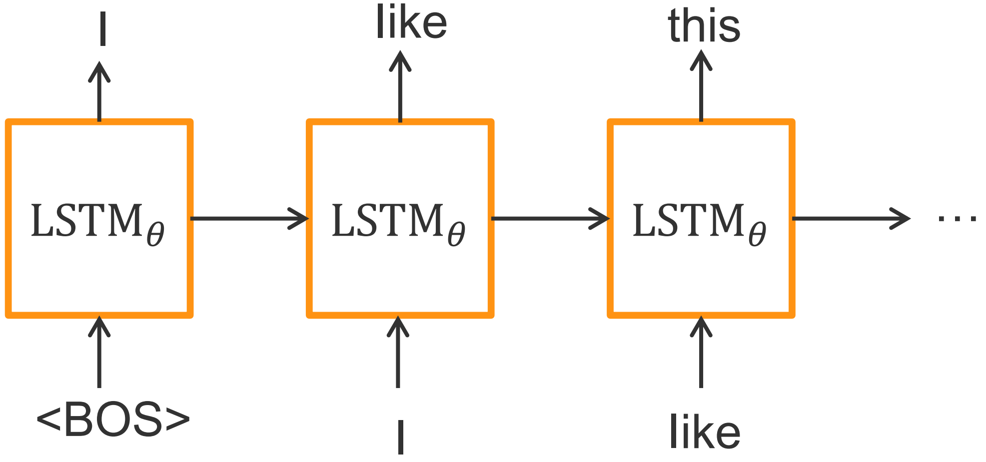
- Optimization
- Data pre-processing
- Result post-processing
- Evaluation
- …

Example: Language Model

- Calculates the probability of a sentence:
  - Sentence:

$$\boldsymbol{y} = (y_1, y_2 \dots, y_T)$$

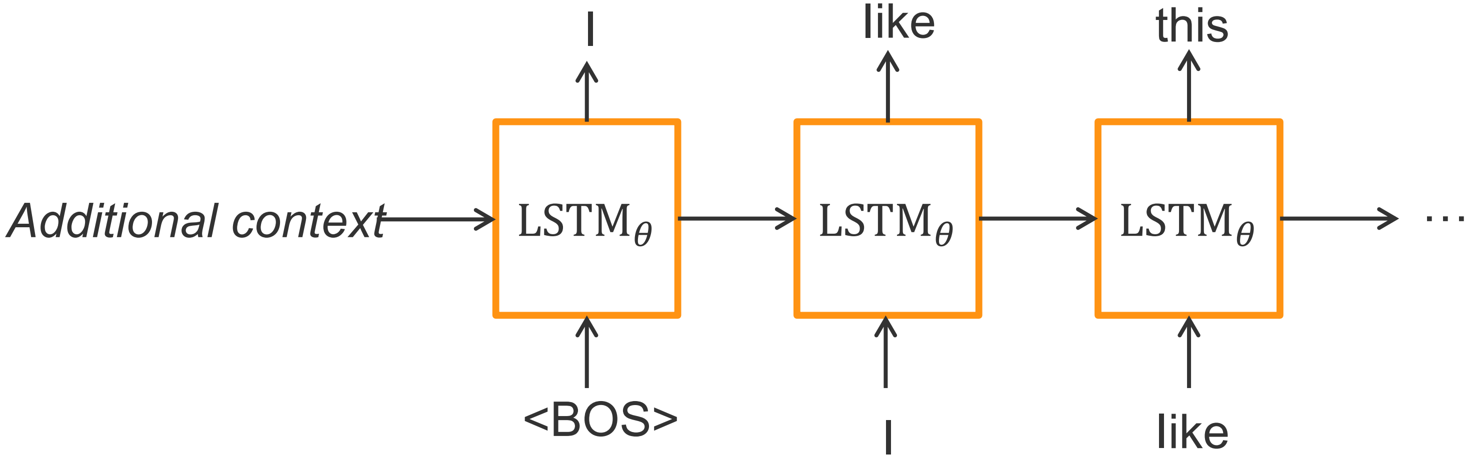
$$p_\theta(\boldsymbol{y}) = \prod_{t=1}^{T} p_\theta(y_t \mid \boldsymbol{y}_{1:t-1})$$



Example: *Conditional* Language Model

- Conditions on additional task-dependent context $\boldsymbol{x}$
  - Machine translation: (representation of) source sentence
  - Medical image report generation: (representation of) medical image

$$p_\theta(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{t=1}^{T} p_\theta(y_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{x})$$

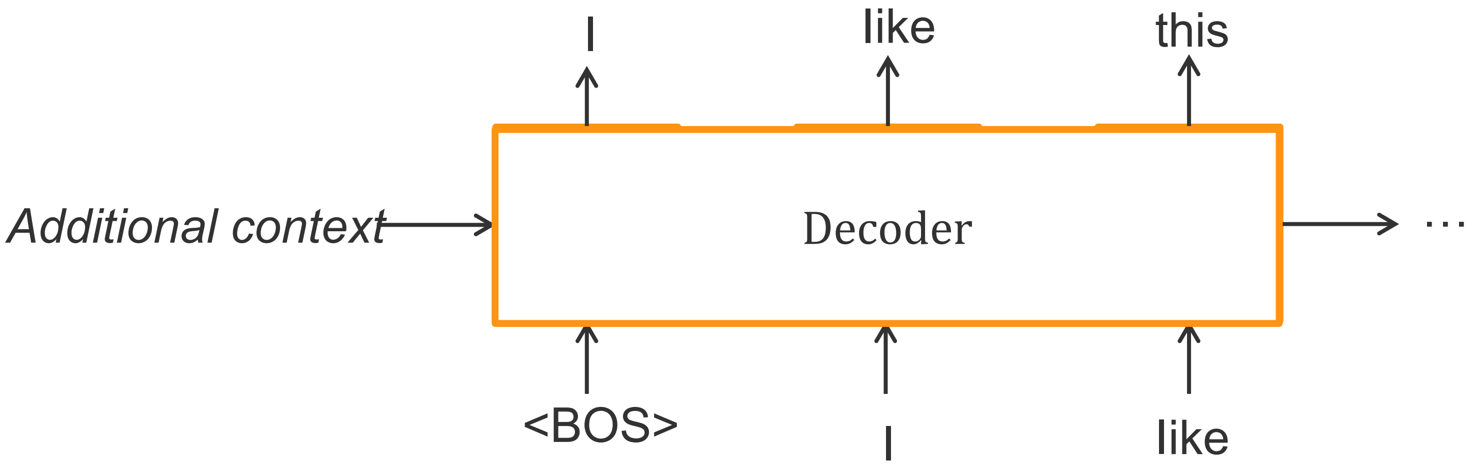


Example: *Conditional* Language Model

- Conditions on additional task-dependent context $\boldsymbol{x}$
  - Machine translation: (representation of) source sentence
  - Medical image report generation: (representation of) medical image

$$p_\theta(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{t=1}^{T} p_\theta(y_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{x})$$

- Language model as a decoder

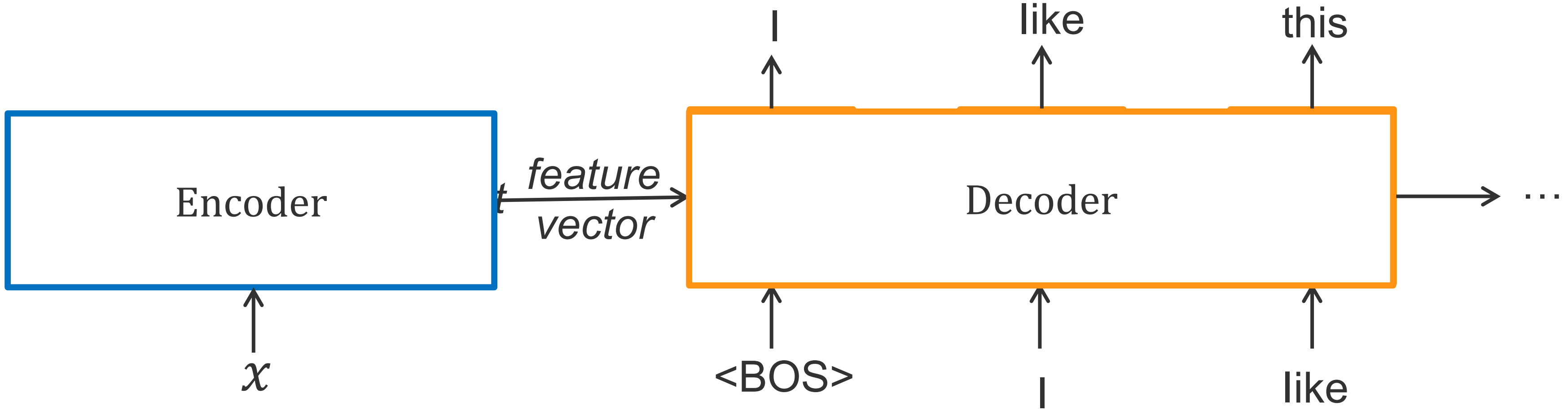


Example: *Conditional* Language Model

- Conditions on additional task-dependent context $\boldsymbol{x}$
  - Machine translation: (representation of) source sentence
  - Medical image report generation: (representation of) medical image

$$p_\theta(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{t=1}^{T} p_\theta(y_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{x})$$

- Language model as a decoder
- Encodes context with an encoder



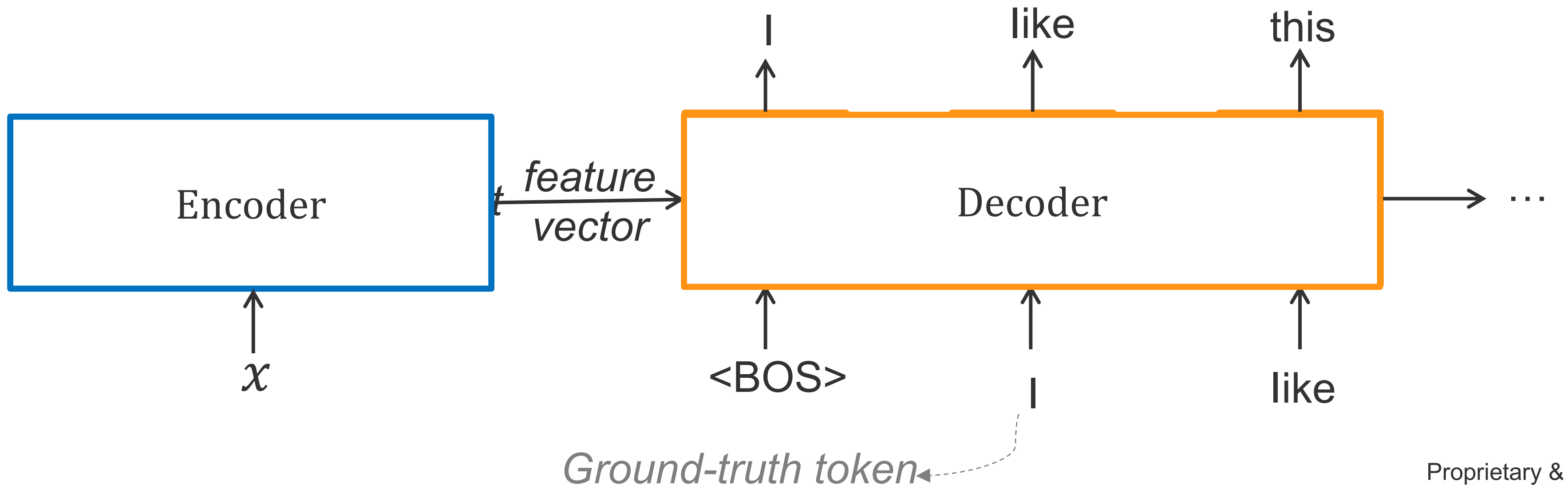
Training: Maximum Likelihood Estimation (MLE)

- Given data example $(\boldsymbol{x}, \boldsymbol{y}^*)$
- Maximizes log-likelihood of the data

$$\max_\theta \mathcal{L}_{\text{MLE}} = \log p_\theta(\boldsymbol{y}^* \mid \boldsymbol{x})$$

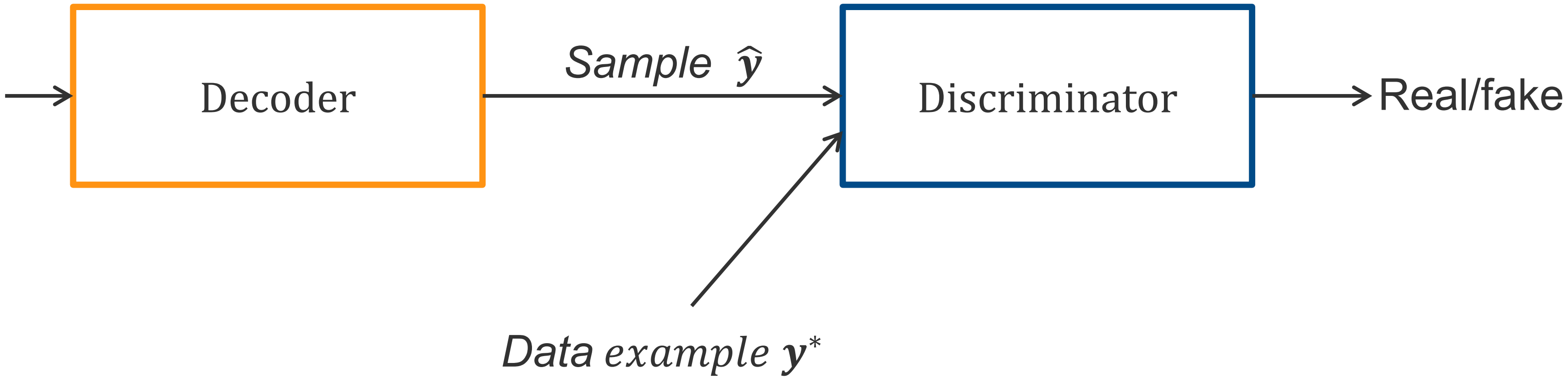
Teacher-forcing decoding:

- For every step t, feeds in the ground-truth token y_t^* to decode next step



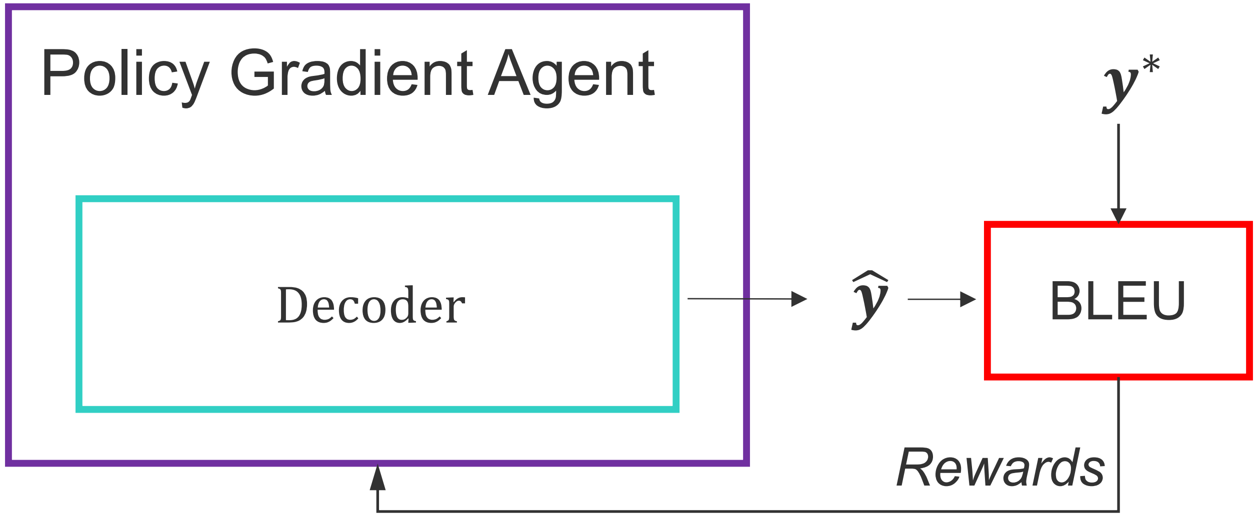
Training: Adversarial Learning

- A discriminator is trained to distinguish between real data examples and fake generated samples
- Decoder is trained to confuse the discriminator
- *Sample* $\hat{\boldsymbol{y}}$ is discrete: not differentiable
  - disables gradient backpropagation from the Discriminator to the Decoder
- Uses a differentiable approximation of $\hat{\boldsymbol{y}}$: *Gumbel-softmax decoding*



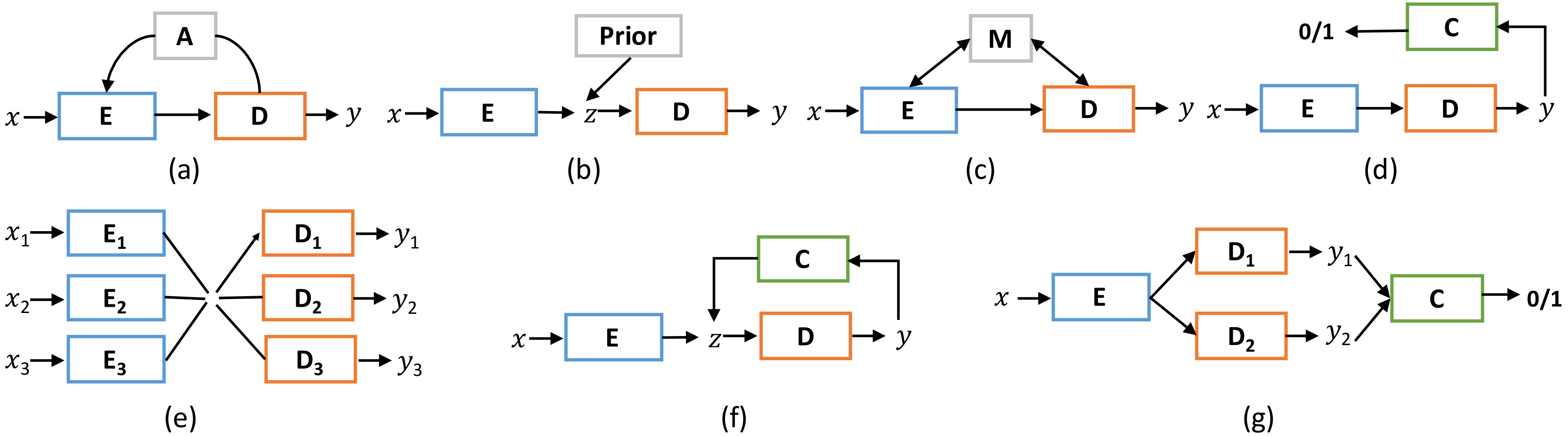
Training: Reinforcement Learning

- Optimizes test metric (e.g., BLEU) directly
- Decoder generates sample $\hat{y}$ which is used to evaluate reward
  - *Greedy decoding / sample decoding / beam search decoding*



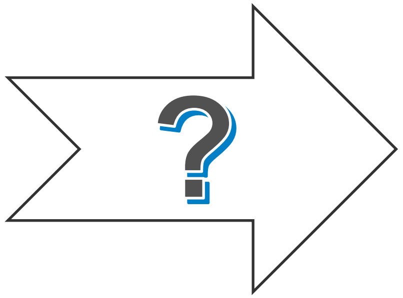
Various (Deep Learning) Techniques (cont'd)

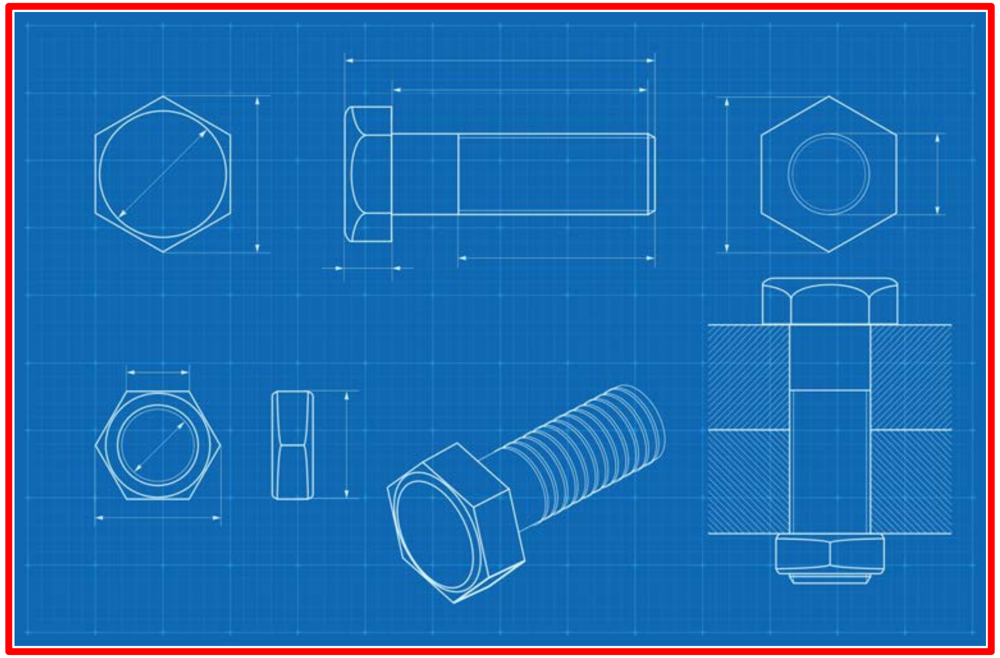
- Techniques are often combined together in various ways to tackle different problems
  - An example of various model architectures



E refers to encoder, D to decoder, C to Classifier, A to attention, Prior to prior distribution, and M to memory

How to modularize and standardize?





Pipeline Decomposition

- Decomposes ML models/algorithms into highly-reusable model **architecture**, **loss**, **learning process,** and **data** modules, among others

$$\max_\theta \mathcal{L}(f_\theta, D)$$

learning process

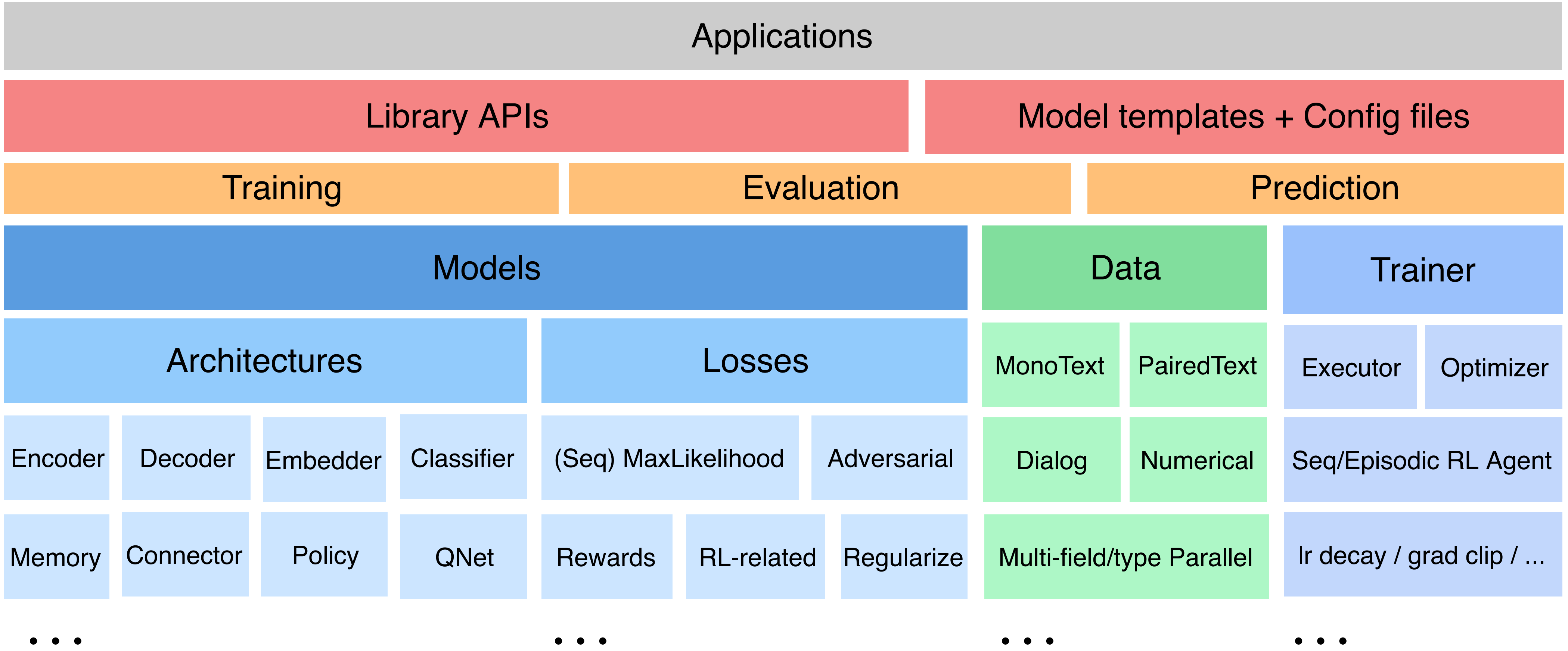
loss

model architecture/
Inference process

data

Texar Stack

***Texar** stack*



Module Catalog

Model architecture

- **Encoder**
  - UnidirectionalRNNEncoder
  - BidirectionalRNNEncoder
  - HierarchicalRNNEncoder
  - ConvEncoder
  - TransformerEncoder
  - ...
- **Decoder**
  - BasicRNNDecoder
  - AttentionRNNDecoder
    - BahdananAttn
    - LuongAttn
    - MonotonicAttn
  - TransformerDecoder
  - Greedy/Sample/BeamSearch/ GumbelSoftmax/... Decoding
  - ...
- **Embedder**
  - WordEmbedder (one-hot / soft)
  - PositionEmbedder
    - Parametrized
    - Sinusoids
  - ...
- **Classifier/ Discriminator**
  - RNNClassifier
  - ConvClassifier
  - HierarchicalClassifier
  - ...
- **Connector**
  - MLPTransformer
  - Stochastic
  - ReparameterizedStochastic
  - Concat
  - Forward
  - ...
- . . .

Model loss

- **Loss**
  - MLE Loss
    - (Sequence) Cross-entropy
    - ...
  - Adversarial Loss
    - Binary Adversarial Loss
    - ...
  - Rewards
  - ...

Trainer

- **RL Agent**
  - Seq RL Agent
    - Seq Policy Gradient Agent
    - ...
  - Episodic RL Agent
    - Policy Gradient Agent
    - DQN Agent
    - Actor-critic Agent
    - ...
- **Optimization**
  - Optimizer
    - Adam/SGD/...
  - Learning Rate Decay
    - Piecewise/Exp/...
  - ...

Data

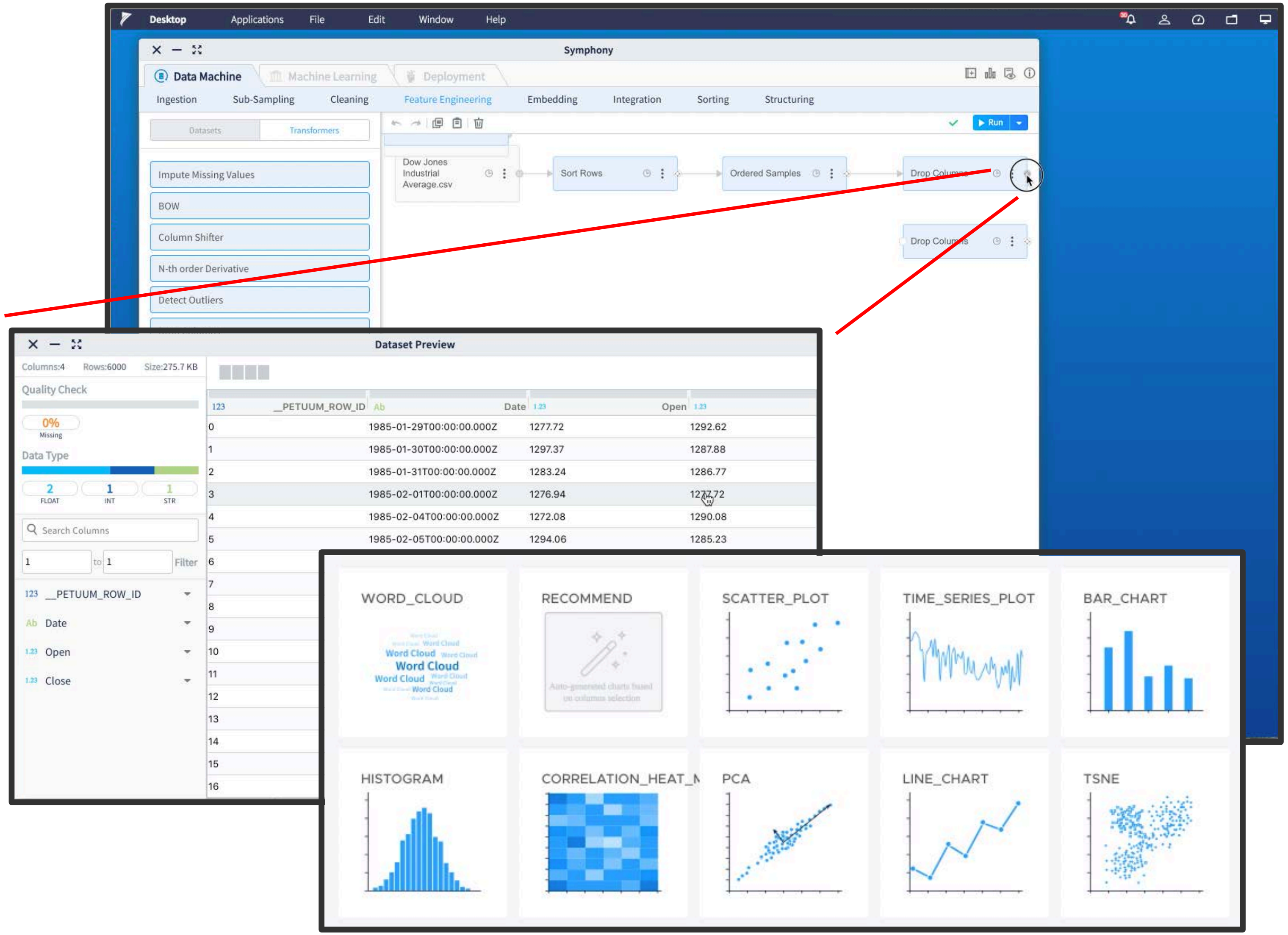
- **Data**
  - MonoText
  - PairedText
  - MultiAligned
  - Dialog
  - Numerical
  - ...
- . . .

Example: Build a sequence-to-sequence model

```
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()

# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
                                batch['source_length'])

# Decode
decoder = AttentionRNNDecoder(memory=enc_outputs,
                                        hparams=decoder_hparams)
outputs, length, _ = decoder(inputs=embedder(batch['target_text_ids']),
                                    seq_length=batch['target_length']-1)

# Loss
loss = sequence_sparse_softmax_cross_entropy(
    labels=batch['target_text_ids'][:,1:], logits=outputs.logits, seq_length=length)
```

Program with Texar Python Library APIs

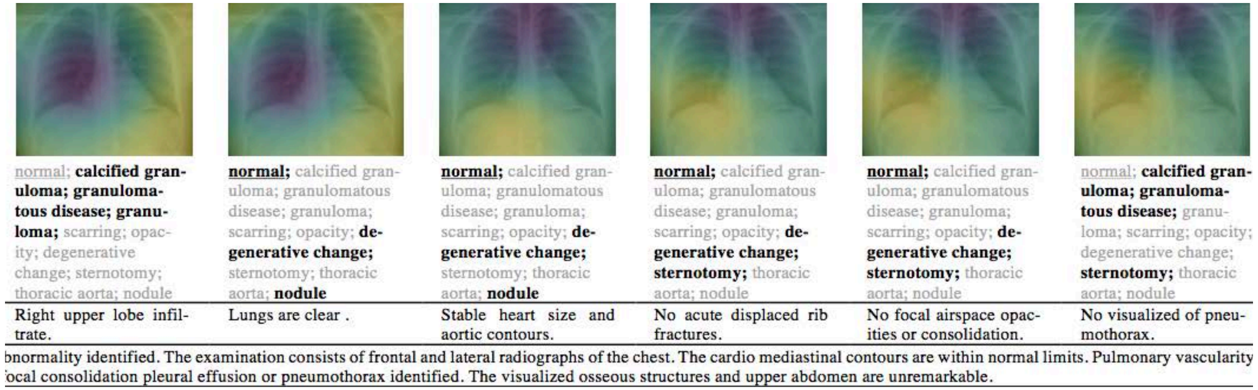
The Petuum Platform: an AI workbench for everyone

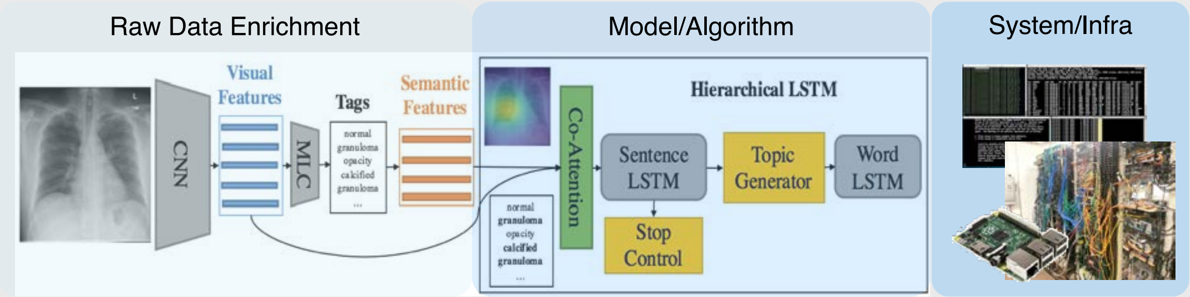


Full Stack Under the Hood

- Process builder UI
- Reusable data & ML blocks
- Workflow engine
- Advanced parallelized processing
- Containerized serving
- Any hardware

AI WITH NO TEARS





- Assemble and experiment with reusable building blocks

PetuumMed – Smart Physician Assistant

NLP –powered Clinical Decision Aids

Critical Information Extraction
Instantly extracts key medical history, co-morbidity, and lab test information from EHRs or papers

Diagnosis & Treatment Recommendation
Recommends diagnoses and medications with strong interpretability based on EHRs and clinical notes

ICD Code Filing
Assigns ICD-10 codes automatically based on EHRs

Mortality Risk Prediction
Predicts a daily mortality rate for ICU patients based on clinical data

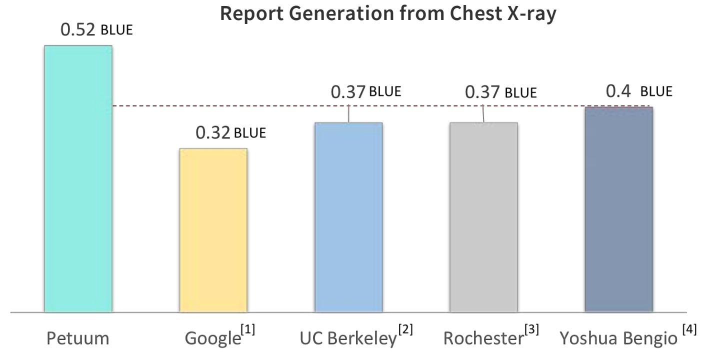
AI-interpreter of Medical Imaging

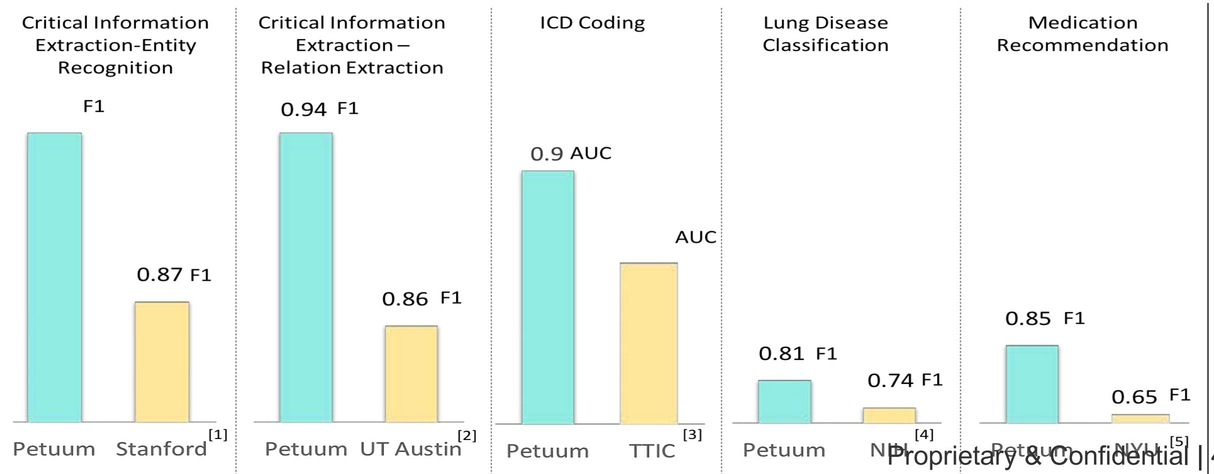
Lung Disease Detection from X-ray
Detect 14 lung lesions with high accuracy from chest x-ray images

Lung Nodule Detection from CT
Detects location, size and shape of lung nodules in CT scans

Report Generation from Chest X-ray
Automatically localizes abnormalities in the image and generates corresponding textual descriptions in a a report

PetuumMed performance leads other AI experts worldwide over all functions: →



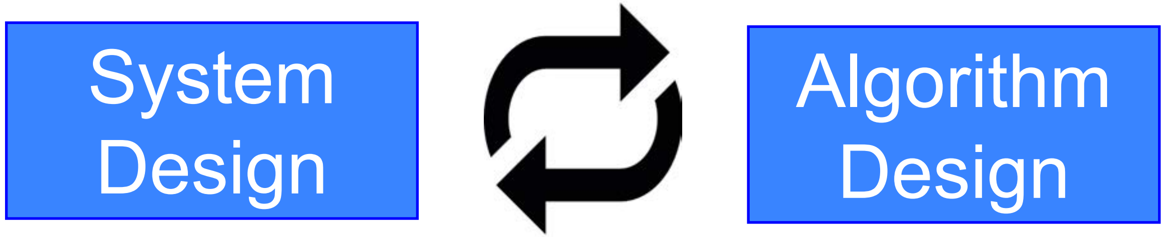


Soundness: correctness guarantees and theory

- Upon ..
  - Data Transformation
  - Model/Parameter estimation
  - Query/Inference
  - Stochastic Sampling
  - Distribution/Parallelization
  - Augmentation/Refinement
  - Porting
  - Operational stress such as faulty infra
  - Security breach
- Characterize the error and risk, or allow simulational study

Now You Can Divide and Conquer!

The Science of SysML



System/Algorithm Co-design

- System design should be tailored to the unique mathematical properties of ML algorithms
- Algorithms can be re-designed to better exploit the system architectures

PETUUM

Distributed Machine Learning

1. How to Distribute? (Ho et al NIPS 2013, Jin, et al. EuroSys, 2015, Wei et al. SoCC 2015)
2. How to Bridge Computation and Communication? (Ho et al NIPS 2013, Dai et al, AAAI 2014)
3. What to Communicate? (Xie et al. UAI 2015, Xie et al, SoCC 2018)
4. How to Communicate? (Zhang et al, ATC 2017, Xie et al, SoCC 2018)

Matrix-parameterized models (MPMs)

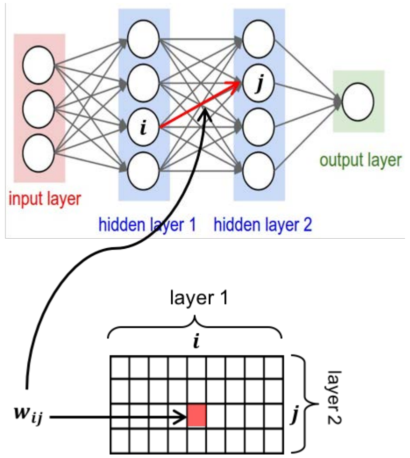
Matrix parameter W

$$\min_{W} \frac{1}{N}\sum_{i=1}^{N} f_i(Wa_i;b_i) + h(W)$$

Loss function

Regularizer

Distance Metric Learning, Topic Models, Sparse Coding, Group Lasso, Neural Network, etc.



Full updates

- Let matrix parameters be W. **Need to send parallel worker updates ΔW to other machines…**
  - Primal stochastic gradient descent (SGD)

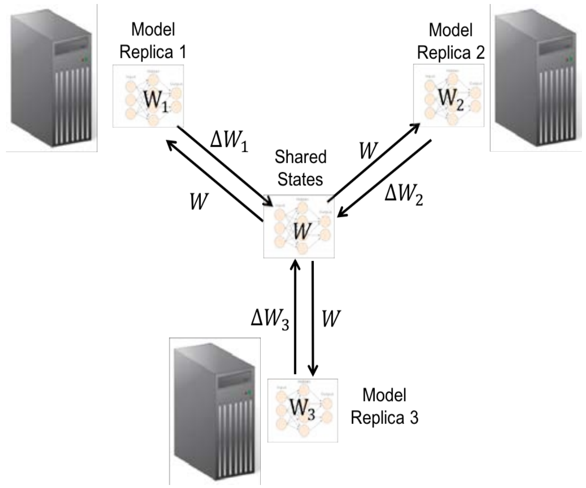
$$\min_W \frac{1}{N}\sum_{i=1}^{N} f_i(Wa_i;b_i) + h(W)$$

$$\boxed{\Delta W = \frac{\partial f(Wa_i, b_i)}{\partial W}}$$

  - Stochastic dual coordinate ascent (SDCA)

$$\min_Z \frac{1}{N}\sum_{i=1}^{N} f_i^*(-z_i) + h^*(\frac{1}{N} ZA^{\mathrm{T}})$$

$$\boxed{\Delta W = (\Delta z_i)a_i}$$



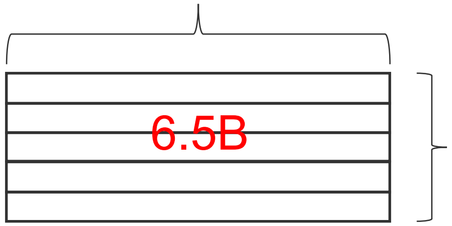
Big MPMs

Billions of params = 10-100 GBs, costly network synchronization

What do we actually need to communicate?

Multiclass Logistic Regression on Wikipedia

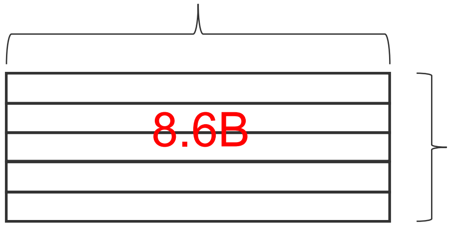
Feature dim. = 20K



#classes=325K

Distance Metric Learning on ImageNet

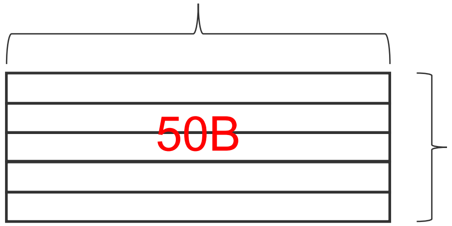
Feature dim. = 172K



Latent dim. = 50K

Topic Model on WWW

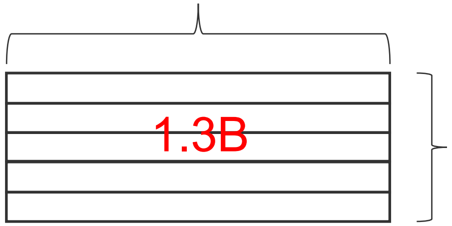
Feature dim. = 1M



Dic. Size= 1M

Neural Network of Google Brain

#neurons in layer 0 = 40K



#neurons in layer 1 = 33K

Pre-update: the sufficient vectors [Xie et al., UAI 2015]

- **Full parameter matrix update** ΔW can be computed as **outer product of two vectors** uv^T -- the sufficient vectors (SV)
  - Primal stochastic gradient descent (SGD)

$$\min_W \frac{1}{N}\sum_{i=1}^{N} f_i(Wa_i;b_i)+h(W)$$

$$\Delta W = uv^{\mathrm{T}} \quad u=\frac{\partial f(Wa_i,b_i)}{\partial(Wa_i)} \quad v=a_i$$

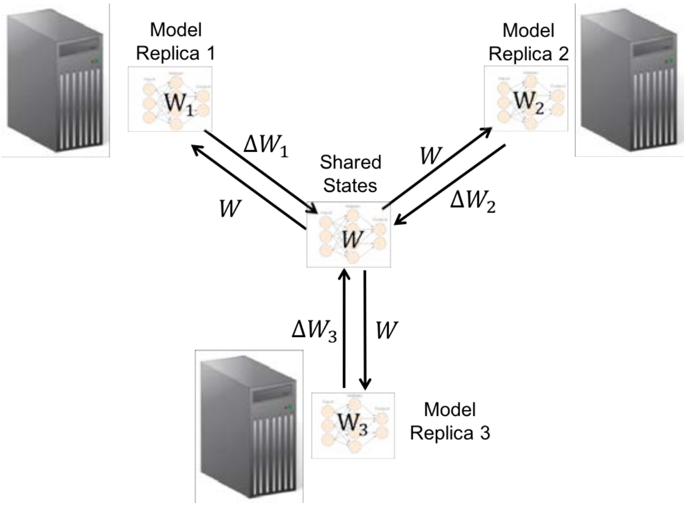
  - Stochastic dual coordinate ascent (SDCA)

$$\min_Z \frac{1}{N}\sum_{i=1}^{N} f_i^*(-z_i)+h^*(\frac{1}{N}ZA^{\mathrm{T}})$$

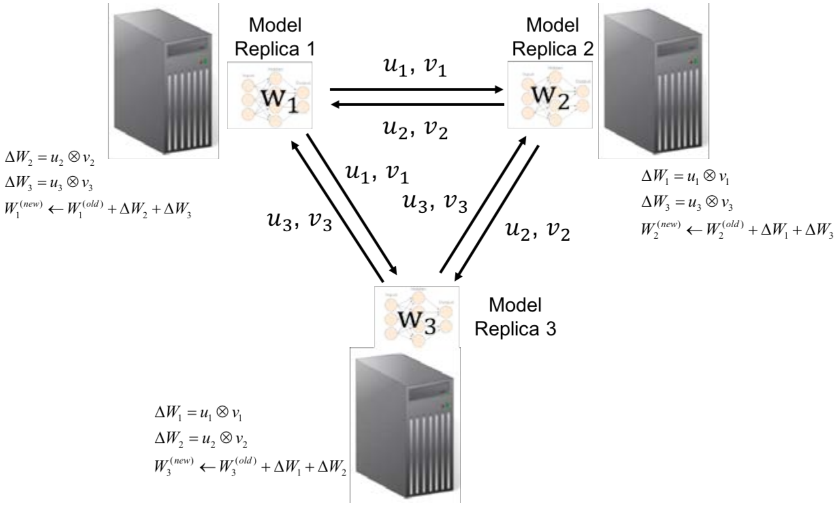
$$\Delta W = uv^{\mathrm{T}} \quad u=\Delta z_i \quad v=a_i$$

Sufficient Vector Broadcaster vs. Parameter Server

parameter server



Transfer SVs instead of ΔW



- A Cost Comparison

	Size of one message	Number of messages	Network Traffic
P2P SV-Transfer	$O(J+K)$	$O(P^2)$	$O((J+K)P^2)$
Parameter Server	$O(JK)$	$O(P)$	$O(JKP)$

Convergence guarantee

Theorem 1. *Let Assumption 1 hold, and let* $\{\mathbf{W}_p^c\}$, $p = 1, \ldots, P$, $\{\mathbf{W}^c\}$ *be the local sequences and the auxiliary sequence, respectively.*

Under full broadcasting (i.e., $Q = P - 1$*) and set the learning rate* $\eta := \eta_c = O(\sqrt{\frac{1}{L\sigma^2 Psc}})$*, we have*

- $\liminf_{c \to \infty} \mathbb{E}\|\nabla F(\mathbf{W}^c)\| = 0$*, hence there exists a subsequence of* $\nabla F(\mathbf{W}^c)$ *that almost surely vanishes;*
- $\lim_{c \to \infty} \max_p \|\mathbf{W}^c - \mathbf{W}_p^c\| = 0$*, i.e., the maximal disagreement between all local sequences and the auxiliary sequence converges to 0 (almost surely);*
- *There exists a common subsequence of* $\{\mathbf{W}_p^c\}$ *and* $\{\mathbf{W}^c\}$ *that converges almost surely to a stationary point of* F*, with the rate* $\min_{c \le C} \mathbb{E}\| \sum_{p=1}^P \nabla F_p(\mathbf{W}_p^c)\|_2^2 \le O\left(\sqrt{\frac{L\sigma^2 Ps}{C}}\right)$

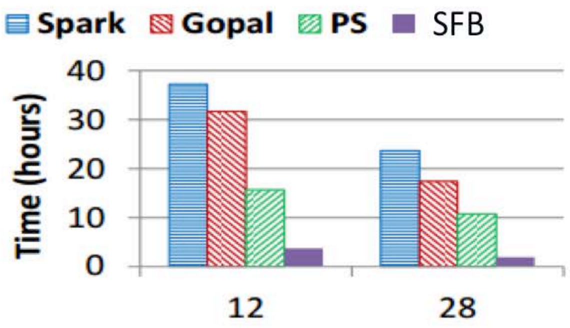
Under partial broadcasting (i.e., $Q < P - 1$*) and set a constant learning rate* $\eta = \frac{1}{CLG(P-Q)}$*, where* C *is the total number of iterations. Then we have*

$$\min_{c \le C} \mathbb{E}\left[\| \sum_{p=1}^P \nabla F_p(\mathbf{W}_p^c)\|_2^2\right] \le O\left(LG(P-Q) + \frac{P(sG + \sigma^2)}{CG(P-Q)}\right).$$

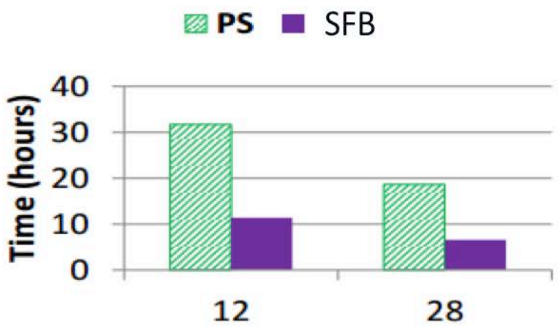
Hence, the algorithm converges to a $O(LG(P-Q))$ *neighbourhood if* $C \to \infty$.

Under partial broadcasting, the algorithm converges to a $O(LG(P - Q))$ neighborhood if $C \longrightarrow \infty$.

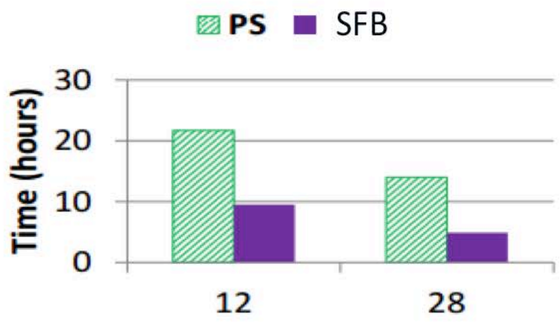
Convergence Speedup



Multiclass Logistic Regression (MLR)



Distance Metric Learning (DML)



Sparse Coding (SC)

- 3 Benchmark ML Programs
  - Big parameter matrices with 6.5-8.6b entries (30+GB), running on 12- & 28-machine clusters
- 28-machine SFB finished in 2-7 hours
  - Up to 5.6x faster than 28-machine PS, 12.3x faster than 28-machine Spark
- PS cannot support SF communication, which requires decentralized storage

Xie et al., UAI 2015

Hybrid Updates: PS + SFB

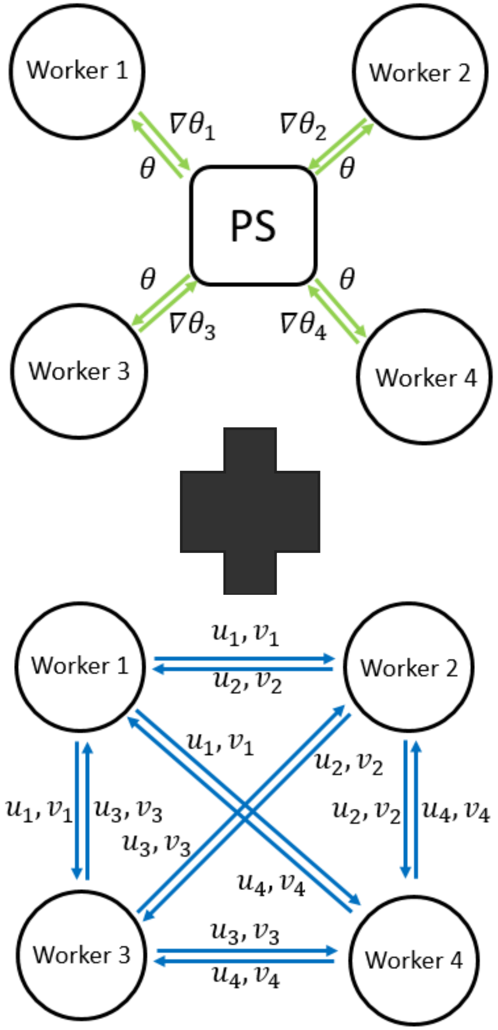
- Hybrid communications:

Parameter Server +

Sufficient Factor Broadcasting

  - Parameter Server: Master-Slave topology
  - Sufficient factor broadcasting: P2P topology

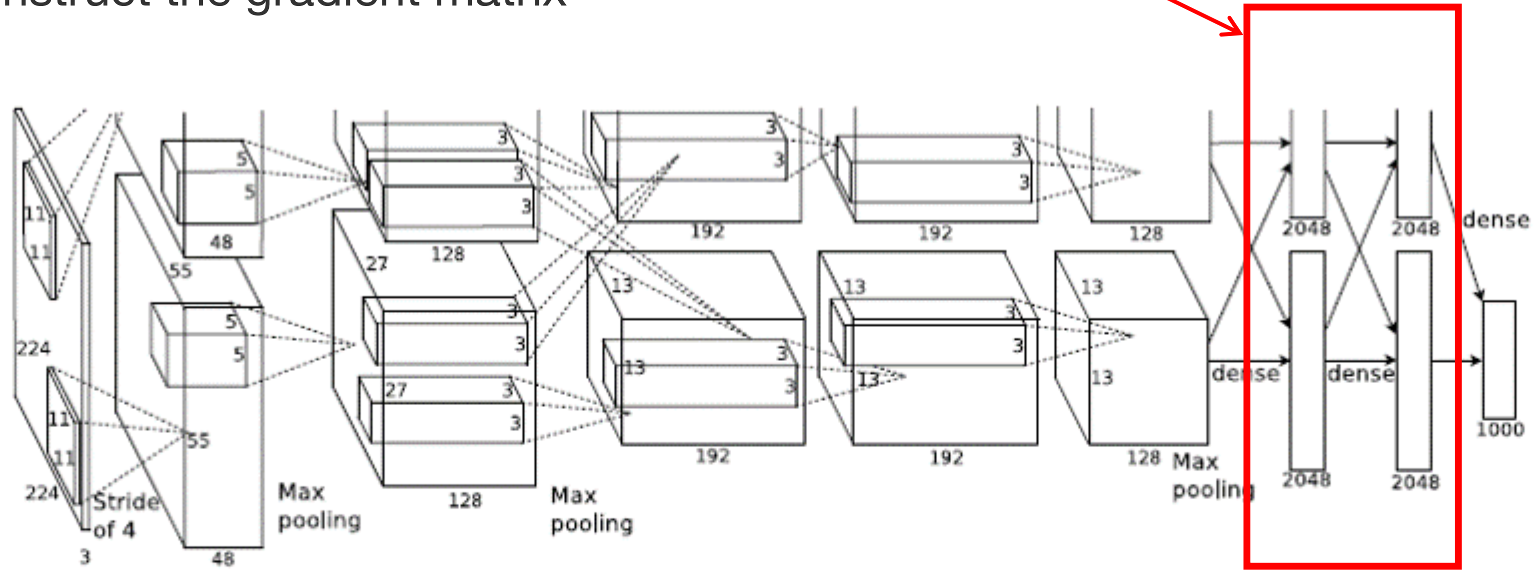
- For problems with a mix of large and small matrices,
  - Send small matrices via PS
  - Send large matrices via SFB



Zhang et al., 2015, Zhang et al. 2017

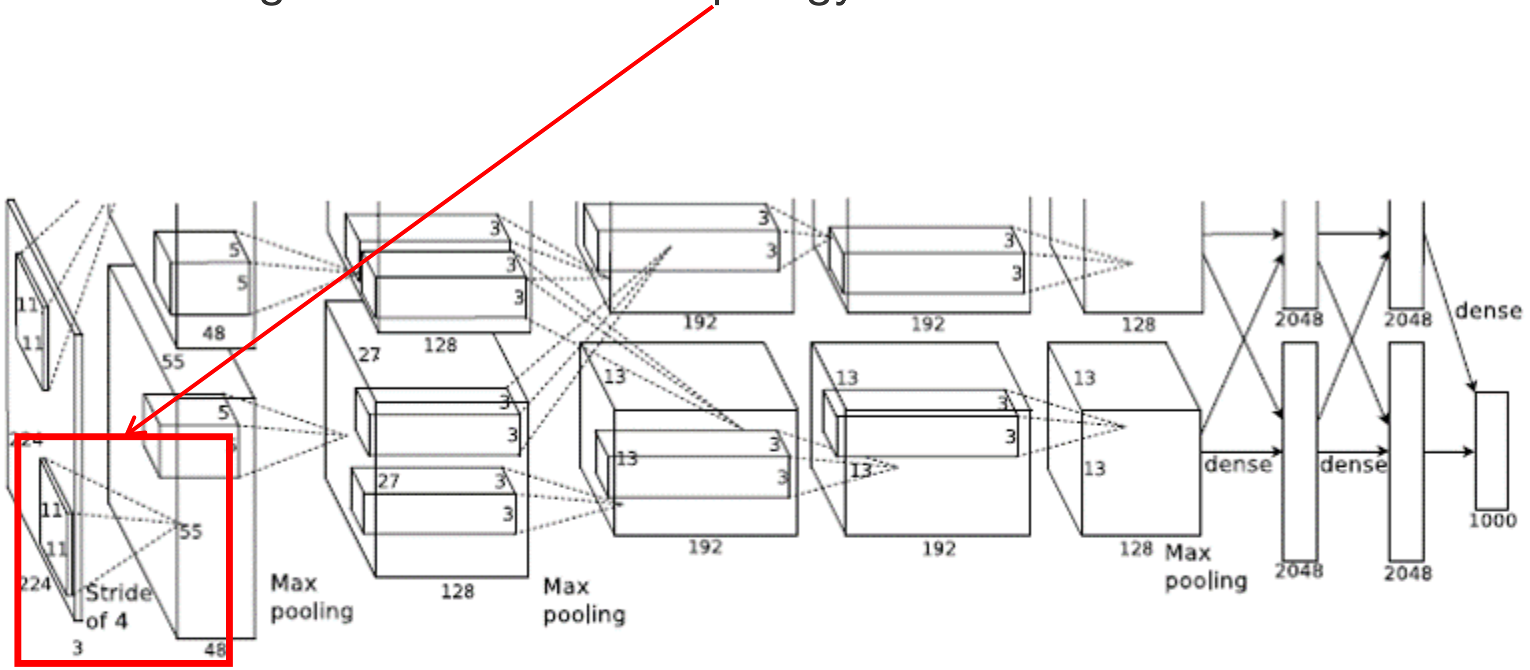
Hybrid example: CNN [Zhang et al., 2015]

- Example: AlexNet CNN model
  - Final layers = 4096 * 4096 matrix (17M parameters)
  - Use SFB to communicate
    - 1. Decouple into two 4096 vectors: u, v
    - 2. Transmit two vectors
    - 3. Reconstruct the gradient matrix



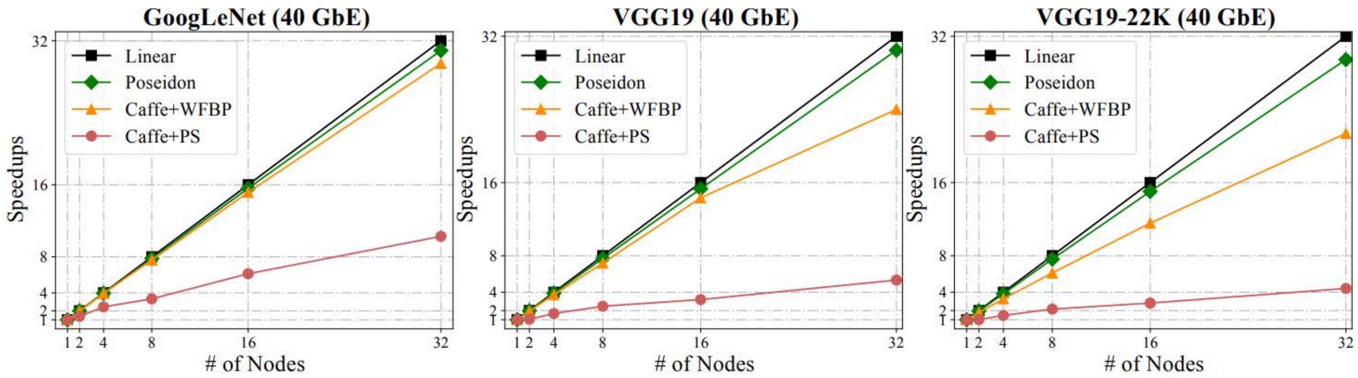
Hybrid example: CNN [Zhang et al., 2015]

- Example: AlexNet CNN model
  - Convolutional layers = e.g. 11 * 11 matrix (121 parameters)
  - Use Full-matrix updates to communicate
    - 1. Send/receive using Master-Slave PS topology

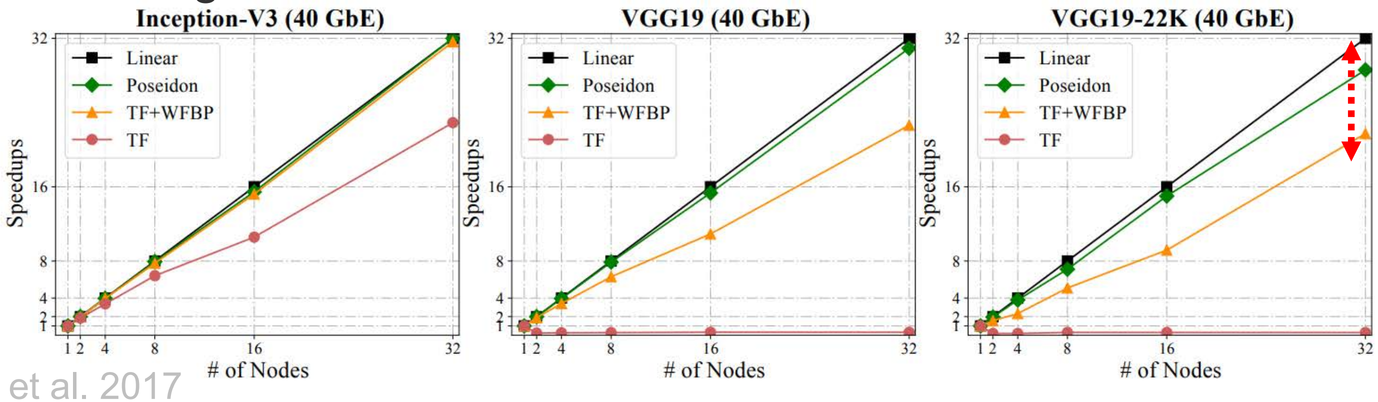


Hybrid Communication

- Results: achieve linear scalability across different models/data with 40GbE bandwidth
  - Using Caffe as an engine:



  - Using TensorFlow as engine

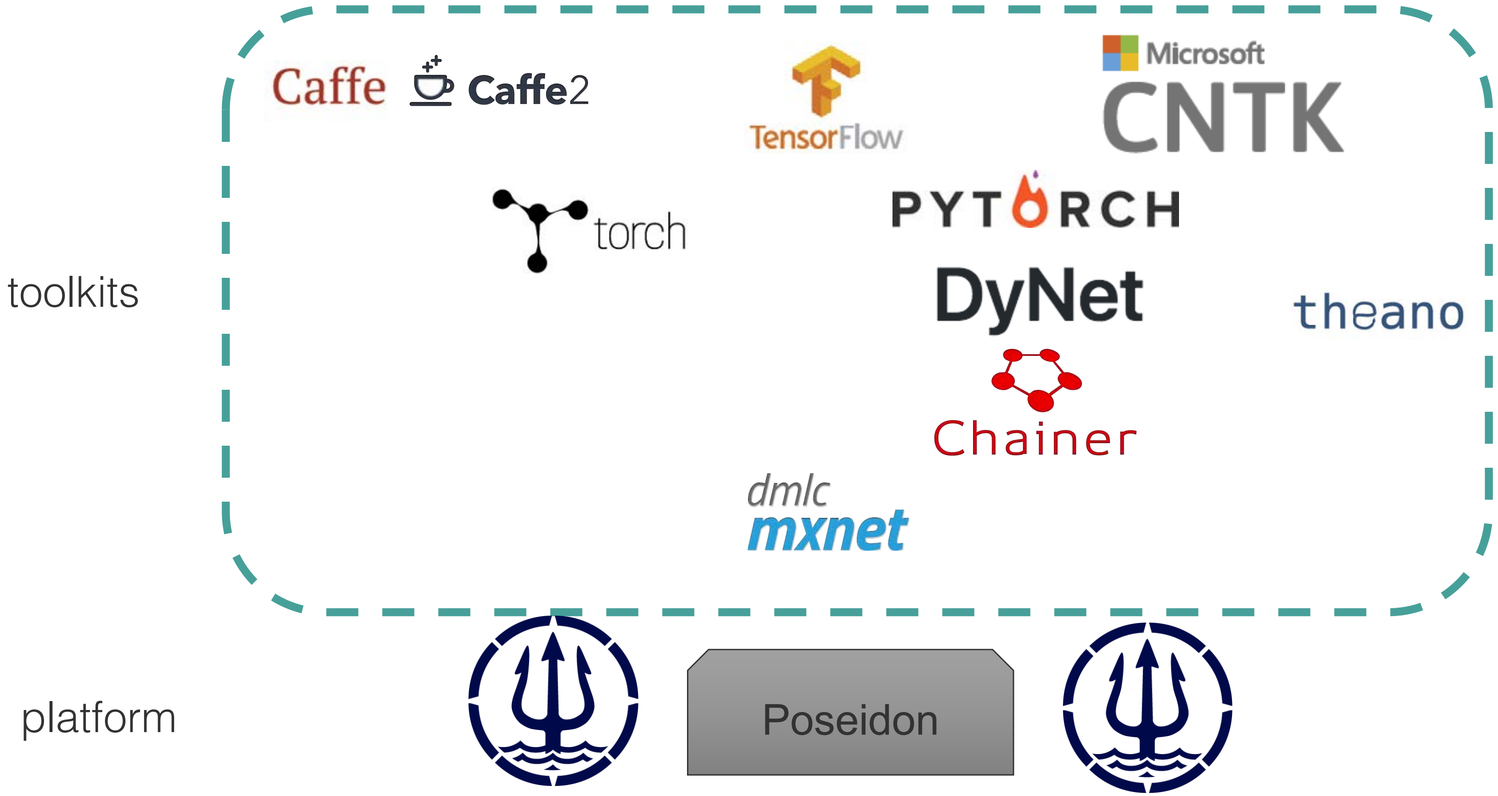


Improve over WFBP

Zhang et al., 2015, Zhang et al. 2017

The Petuum Poseidon Engine [Zhang et al., ATC 2017]

- Poseidon: An efficient communication architecture
  - A distributed platform to amplify existing DL toolkits



Programming Interface and Software Frameworks

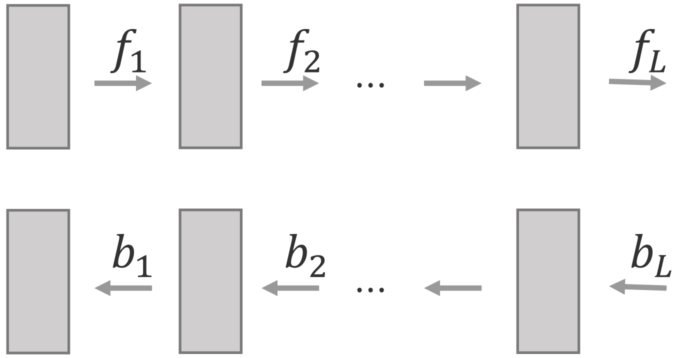
From TensorFlow

to DyNet (Neubig et al, NIPS 2016)

to Cavs (Zhang et al, SoCC 2018)

Deep Learning as Dataflow Graphs

- Gradient Descent via Backpropagation
  - Gradients can be computed by auto differentiation
  - Automatically derive the gradient flow graph from the forward dataflow graph



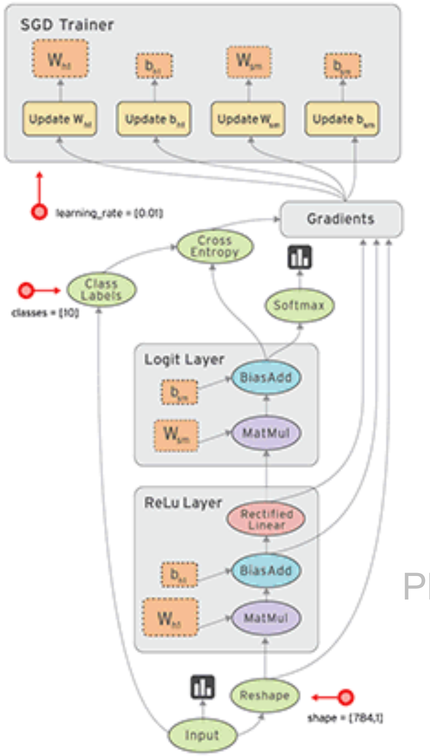
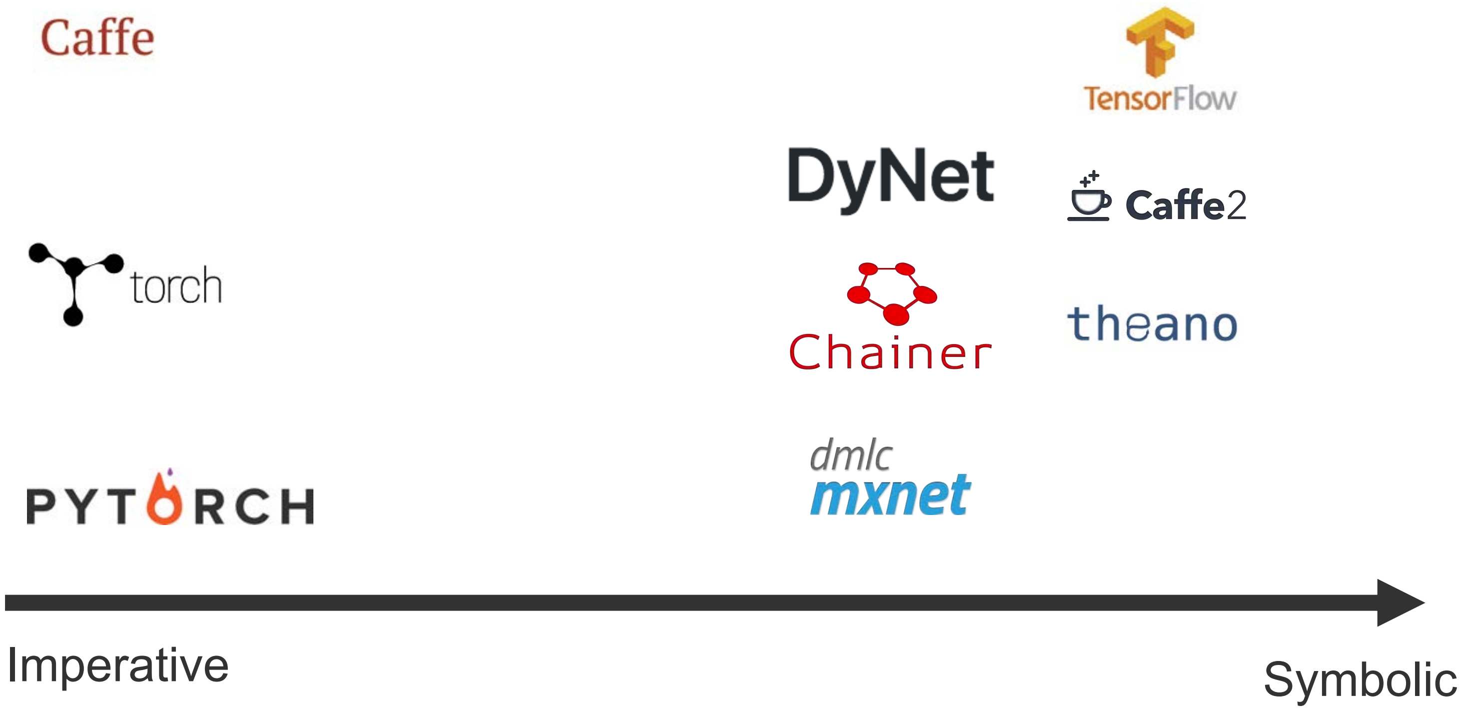


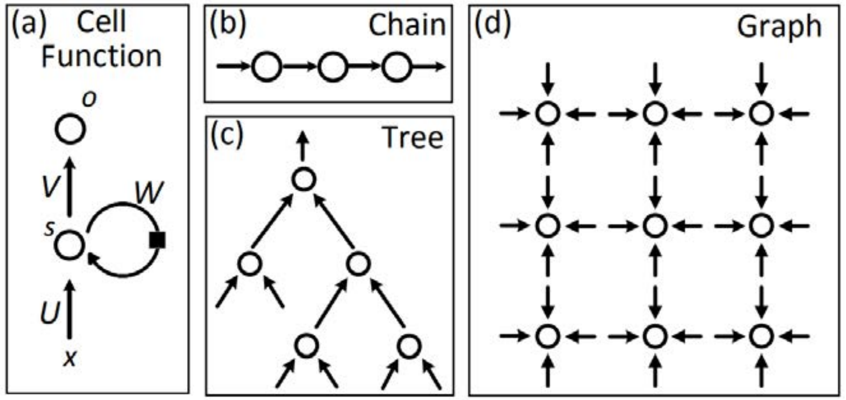
Photo from TensorFlow website

Deep Learning Toolkits



From Static to Dynamic Neural Networks

- Static Declaration vs. Dynamic Declaration
  - Move the graph declaration and construction (and optimization) from outside of the loop to inside the loop
  - Perform single instance training because it is hard to batch

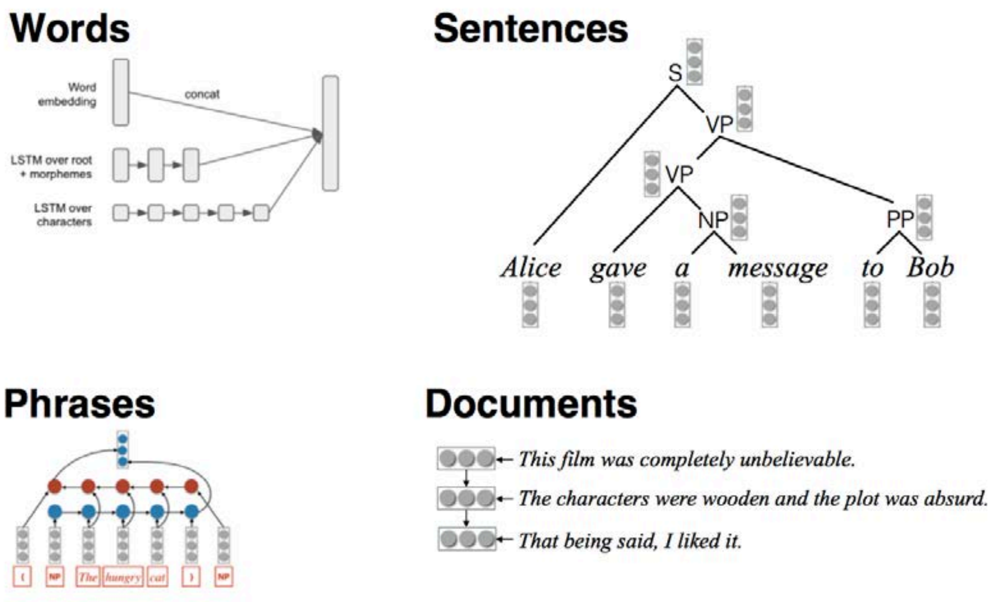


/ (a) static declaration */*
// all samples must share one graph
declare a static data flow graph $\mathcal{D}$.
for $t = 1 \rightarrow T$:
  read the tth data batch $\{x_i^t\}_{i=1}^K$.
  batched computation: $\mathcal{D}(\{x_i^t\}_{i=1}^K)$.

/ (b) dynamic declaration */*
for $t = 1 \rightarrow T$:
  read the tth data batch $\{x_i^t\}_{i=1}^K$.
  for $k = 1 \rightarrow K$:
    declare a data flow graph $\mathcal{D}_i^t$ for x_i^t.
    single-instance computation: $\mathcal{D}_i^t(x_i^t)$.

DyNet (CMU/Petuum) (Neubig et al, NIPS 2016)

- Designed for dynamic deep learning workflow, e.g.



- Key Ingredients
  - Separate parameter declaration and graph construction
  - Declare trainable parameters and construct models first
  - Construct computation graphs
  - Conclusion: Define parameter once, but define graphs dynamically depending on inputs → therefore making the graph construction lighter-weight

Lighter Programming

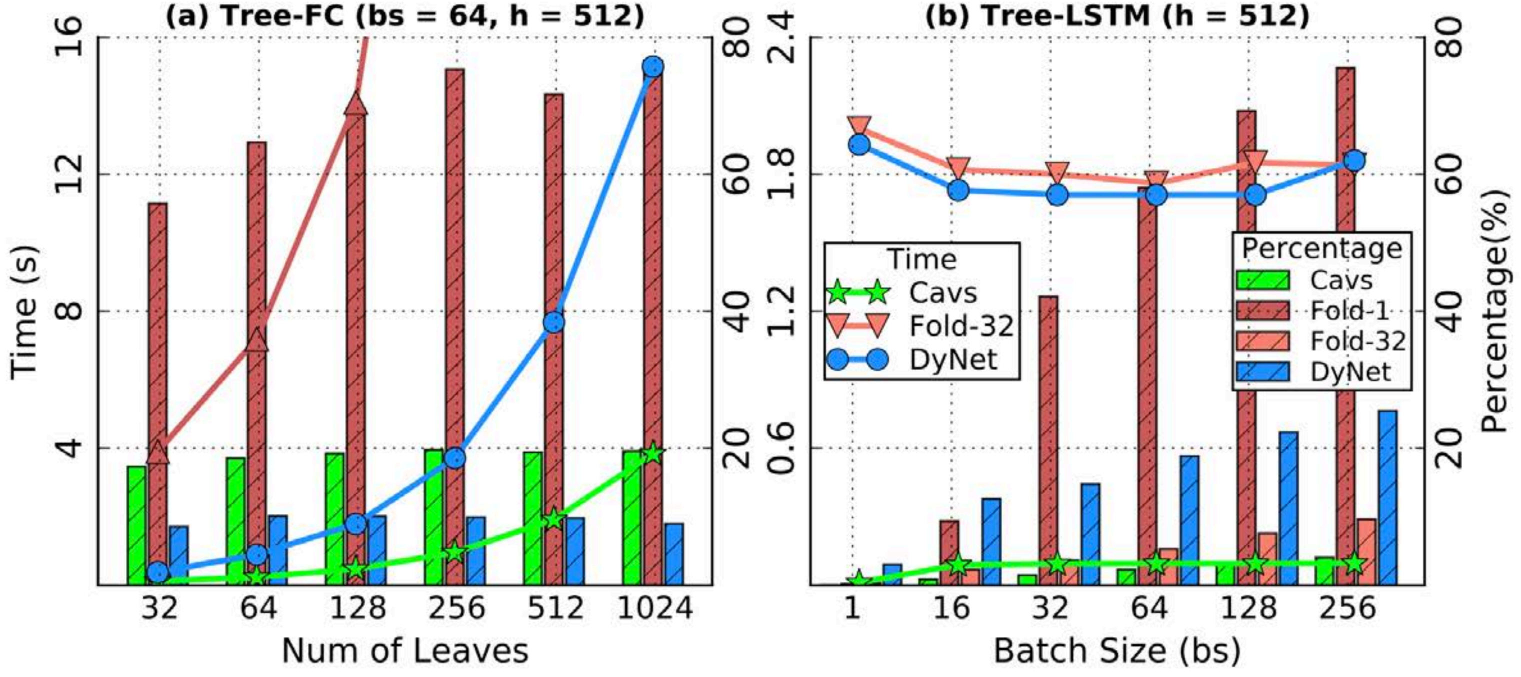
- A visual comparison: implement a TreeRNN

```python
class TreeRNNBuilder(object):
    def __init__(self, model, word_vocab, hdim):
        self.W = model.add_parameters((hdim, 2*hdim))
        self.E = model.add_lookup_parameters((len(word_vocab),hdim))
        self.w2i = word_vocab

    def encode(self, tree):
        if tree.isleaf():
            return self.E[self.w2i.get(tree.label,0)]
        elif len(tree.children) == 1: # unary node, skip
            expr = self.encode(tree.children[0])
            return expr
        else:
            assert(len(tree.children) == 2)
            e1 = self.encode(tree.children[0])
            e2 = self.encode(tree.children[1])
            W = dy.parameter(self.W)
            expr = dy.tanh(W*dy.concatenate([e1,e2]))
            return expr

model = dy.Model()
U_p = model.add_parameters((2,50))
tree_builder = TreeRNNBuilder(model, word_vocabulary, 50)
trainer = dy.AdamTrainer(model)
for epoch in xrange(10):
  for in_tree, out_label in read_examples():
    dy.renew_cg()
    U = dy.parameter(U_p)
    loss = dy.pickneglogsoftmax(U*tree_builder.encode(in_tree), out_label)
    loss.forward()
    loss.backward()
    trainer.update()
```

DyNet TreeLSTM (30 LoC)

TensorFlow TreeLSTM (200 LoC)

Faster



- Graph construction literally takes 80% of time in TensorFlow Fold
- Curve (left axis): absolute time; bar (right): percentage time

Dynamic Declaration: Problems

- Dynamic declaration scarifies efficiency for flexibility
  - Graph construction overhead grows linearly with # of samples
  - Only single-instance computation can be performed – no batching!
  - Hard to incorporate graph-level optimization

/ (b) dynamic declaration */*

for $t = 1 \rightarrow T$:

  read the tth data batch $\{x_i^t\}_{i=1}^K$.

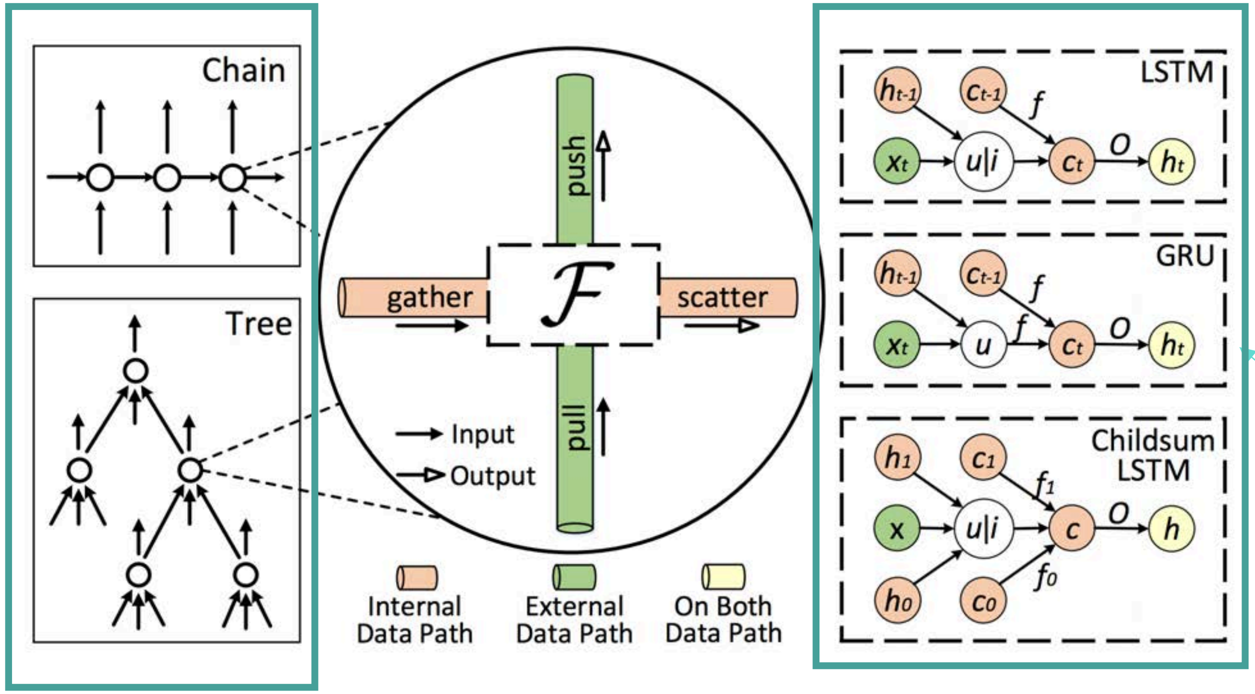
  for $k = 1 \rightarrow K$:

    declare a data flow graph $\mathcal{D}_i^t$ for x_i^t.

    single-instance computation: $\mathcal{D}_i^t(x_i^t)$.

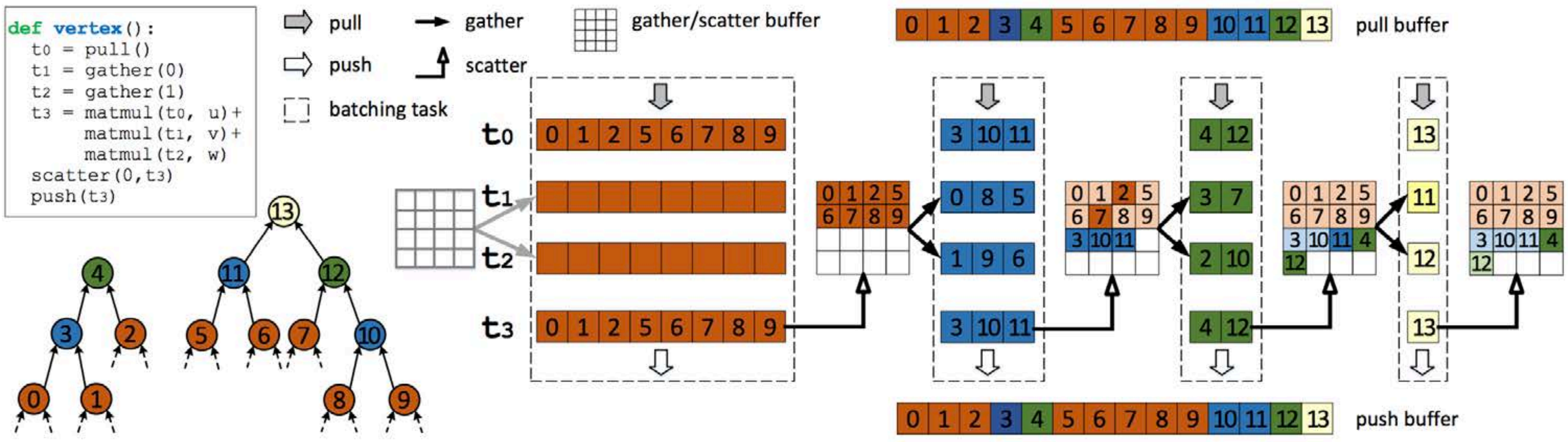
Cavs (Petuum): DL as a Vertex Program (Zhang et al, SoCC 2018)

- Key idea: separate out static ML model declaration from the data-dependent dynamics of input samples
- Vertex-centric representation for DL, decompose a dynamic NN as two modules
  - A vertex function F, which is static;
  - An input graph G, which is data-dependent and dynamic;

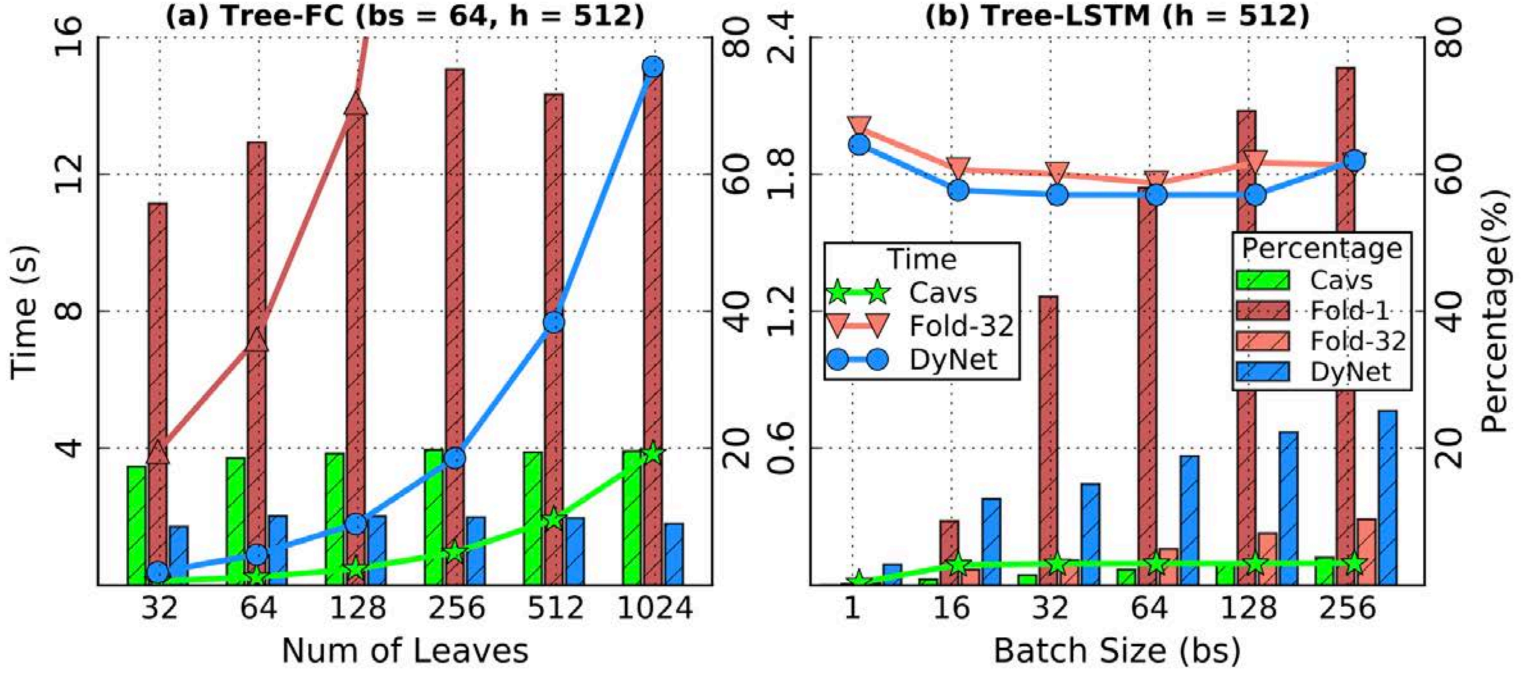


Advanced Memory Management – Dynamic Tensor

- DynamicTensor, to ensure memory continuity: more flexible for dynamically-varying batch size
- With dynamic tensors, Cavs designs a memory management mechanism to guarantee the coalesce of input contents of batched operations on memory



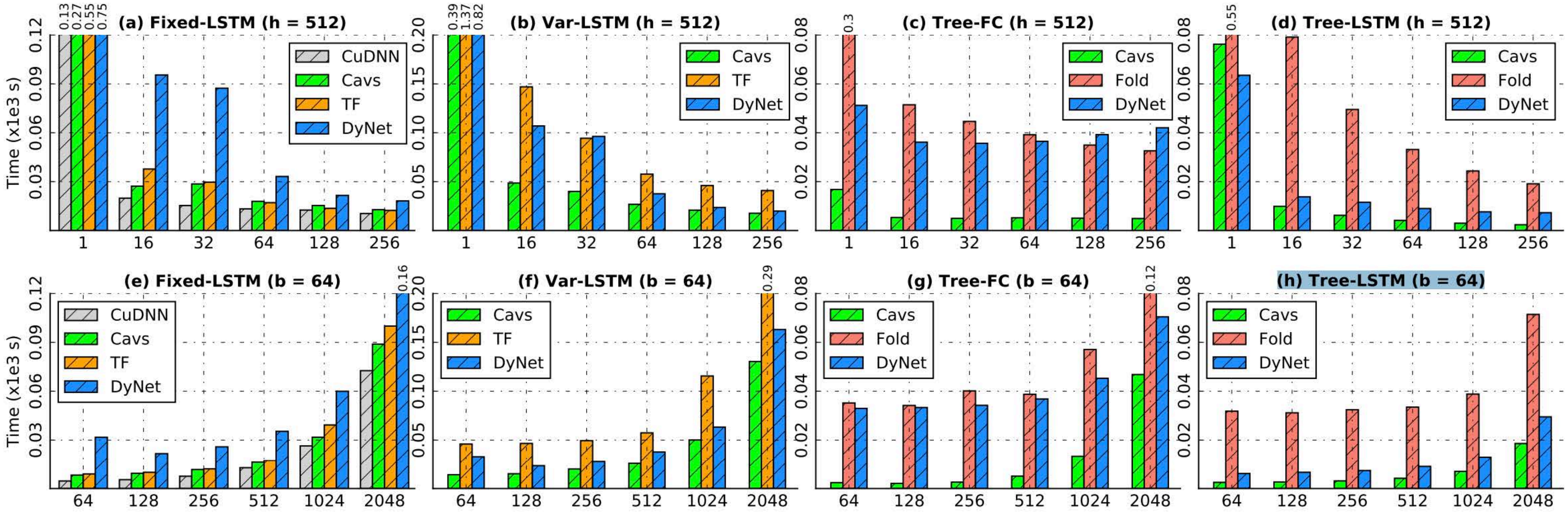
Faster



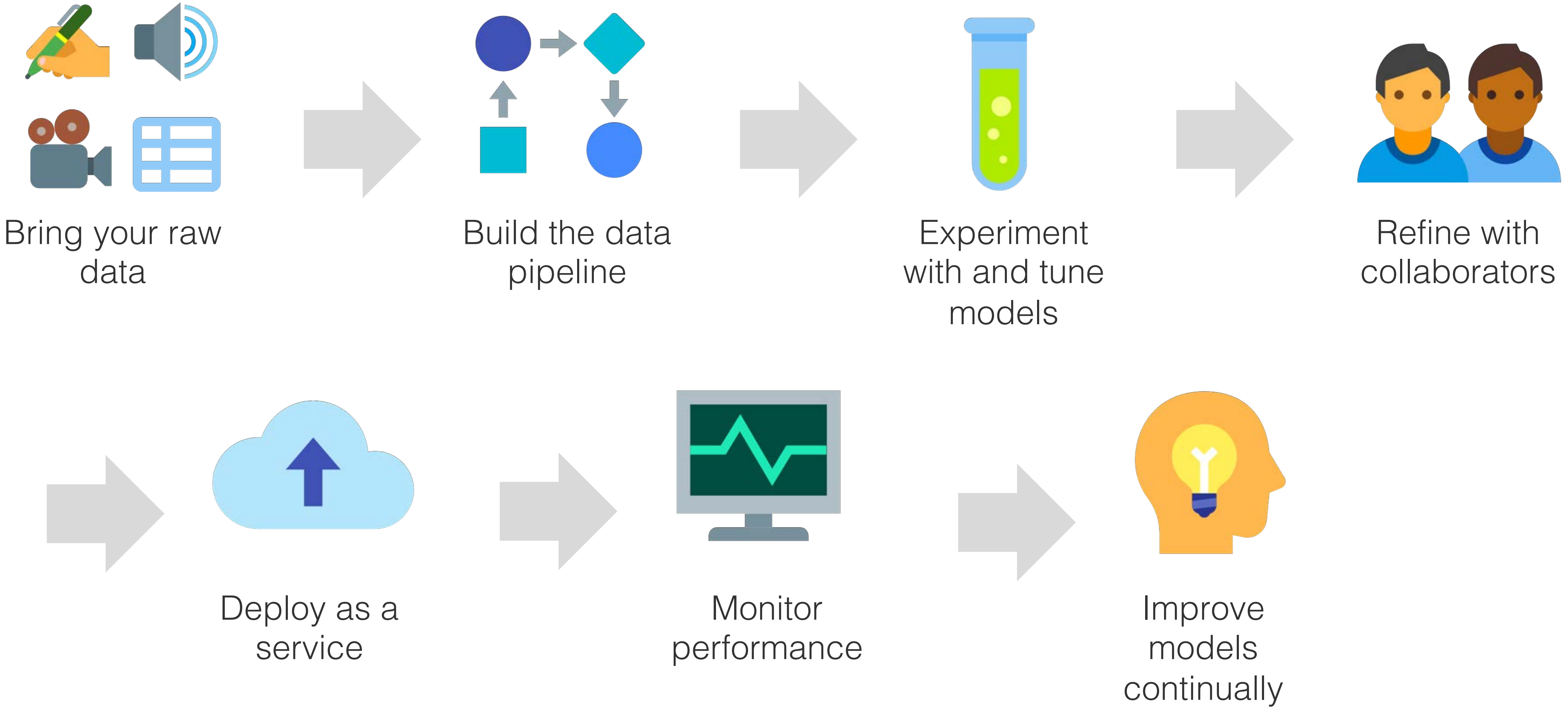
- Graph construction literally takes 80% of time in TensorFlow Fold
- Curve (left axis): absolute time; bar (right): percentage time

Overall Performance

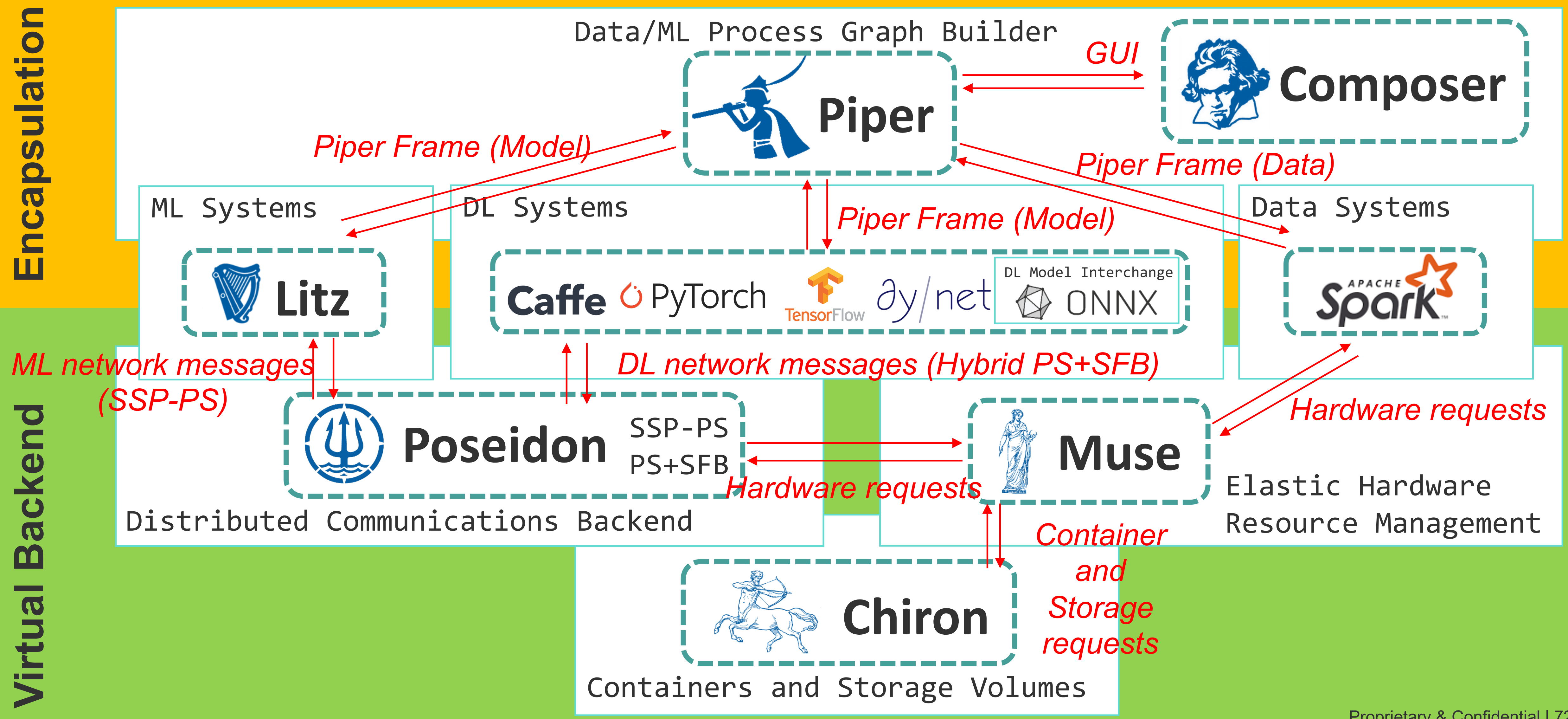
- Overall, Cavs is 1 – 2 orders of magnitude faster than state-of-the-art systems such as DyNet and TensorFlow-Fold.



Summary:
Petuum Facilitates A Full End-to-end AI-Build Process

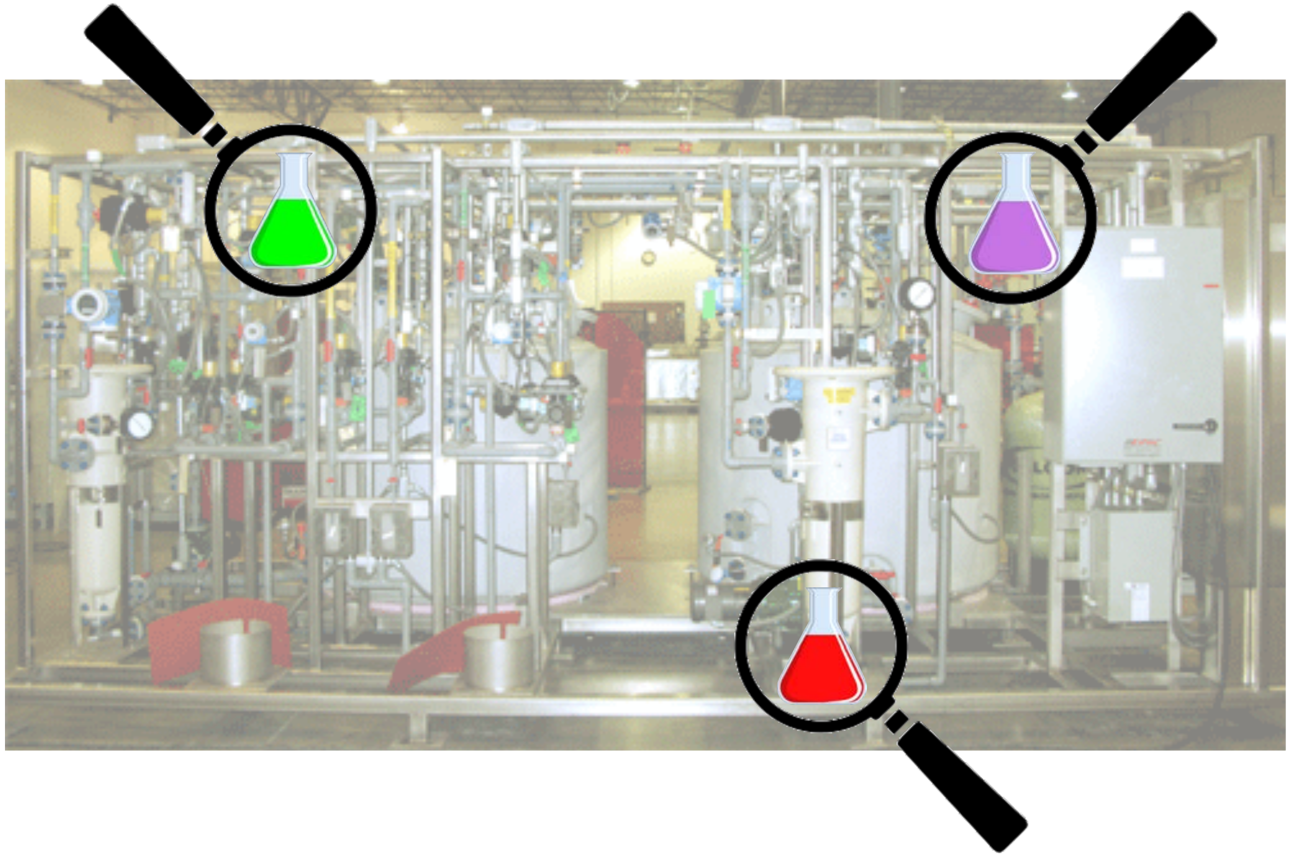


Full Inter-operation



Explainability/Interpretability

- **Data explainability**
  - How the data is pre-processed
- **Model explainability**
  - What you've learned, e.g., feature weights
- **Inference explainability**
  - How each result is inferred
- **Process explainability**
  - Factors beyond or complementary to the mechanisms/mathematics of ML



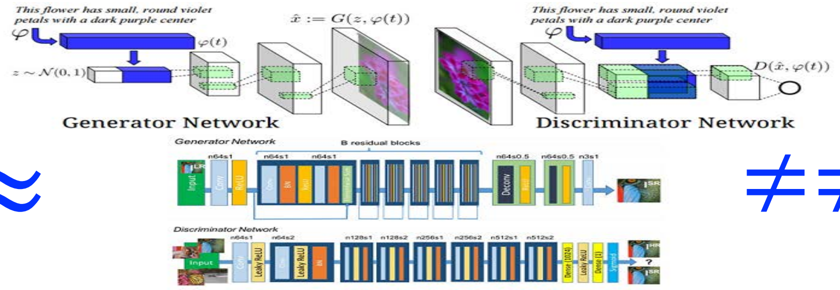
- Post-hoc reason codes may not be sufficient to explain complex AI processes

Discussion:

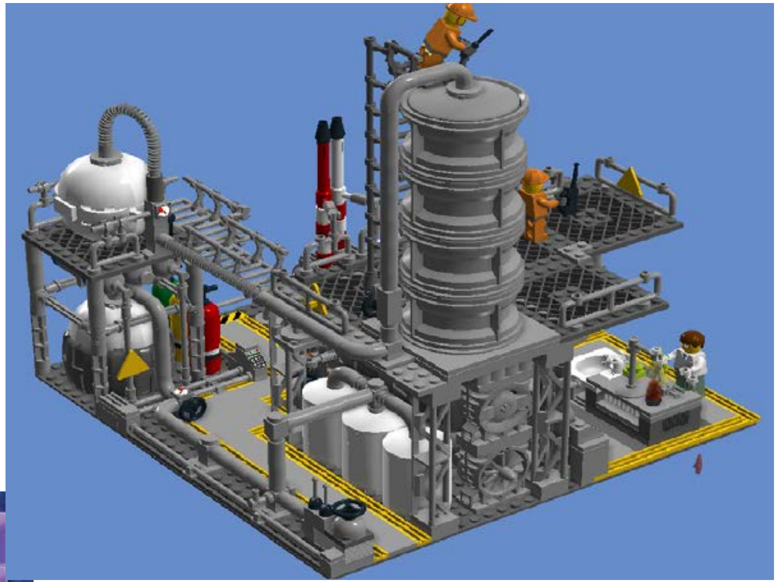
AI as of now: still in medieval age

- Alchemy vs. chemistry vs. chemical engineering

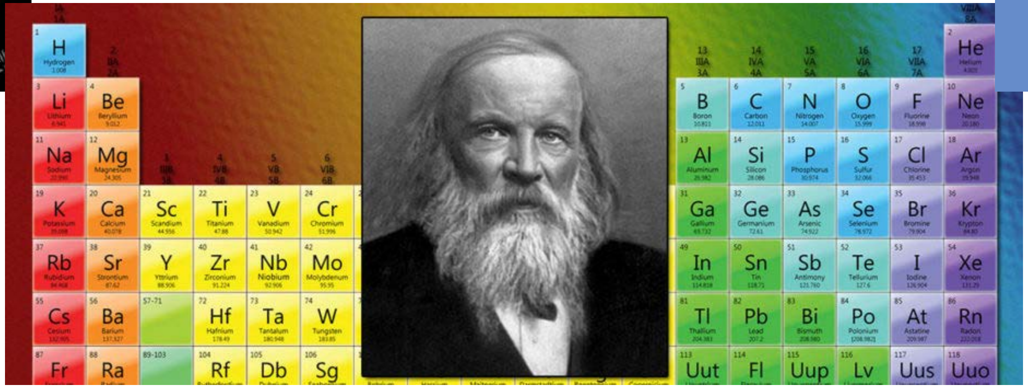
≈



≠≠



⧣



AI-Create Now Is Anything But Efficient & E

AI-Build requires a sound scientific & engineering process
--- not just model/algorithm fiddling

- First Principles
- The “Civil” Engineering
- Explain the process and outcome
- Analysis and safety under real deployment and operation
- Standardize, mass production, cost amortization

PETUUM

Petuum's Mission

Industrialize AI technology
– turning it from black-box artisanship into standardized engineering process

Transform enterprises across industries
– turning them into owners, builders, and informed users of AI

Best AI talent in the field

(We are hiring!)

One foundation for your current and future AI-building needs

WORLD ECONOMIC FORUM

WORLD ECONOMIC FORUM Technology Pioneers

A SoftBank portfolio company and
A 2018 WEF TP Winner

PETUUM

Thank You