Department of
Computer Science & Engineering
計算機科學與工程學系

ICPE 2018
Berlin
April 12, 2018

AI Techniques in Software Engineering Paradigm

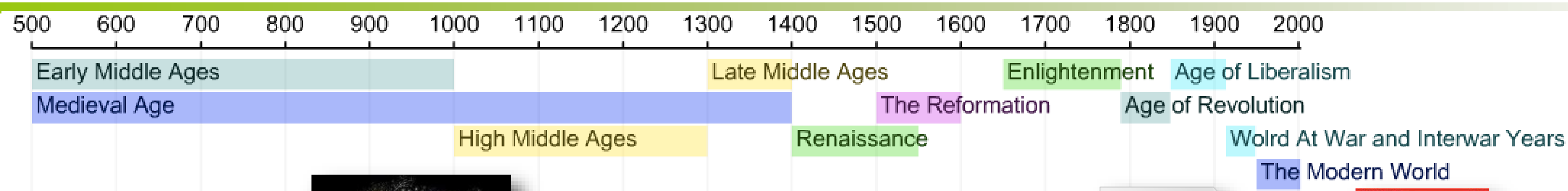
Rung-Tsong Michael LYU

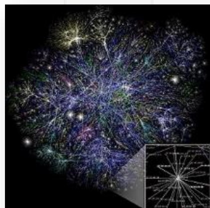
Computer Science & Engineering Department
The Chinese University of Hong Kong

香港中文大学
Chinese University of Hong Kong

- Introduction
- AI Techniques
- Development Phase
- Operational Phase
- Analysis Phase
- Conclusion

A Brief History of the IT World

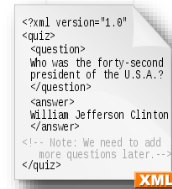




Birth of Internet

IBM Desktop PC

Apple Macintosh



Birth of XML

Time Magazine Person of the Year

1750 1945 1969 1975 1981 1983 1984 1989 1996 2004 2006 2007

Industrial Revolution

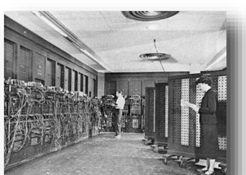
Information Age

Internet Age

WWW Age

Attention Age

ENIAC



The MITS Altair Apple II

Time Magazine Person of the Year

Birth of WWW

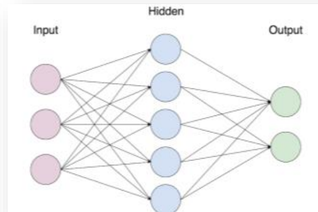
Birth of Web 2.0

Birth of iPhone

A Brief History of AI Development

Birth of "AI"

Representation Language of Logic

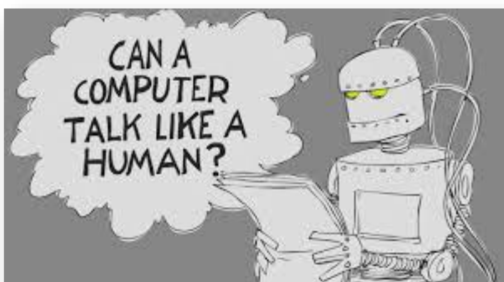


Basic Structure

Deep Blue vs Kasparov

Go Game

1950	1956	1965	1974	Mid 80's	1990-2000	2010	2011	2016	Now
Turing test	Artificial Intelligence	Logic Programming	Expert Systems	Neural Network	Breakthrough	Big Data	IBM Watson wins on Jeopardy	Alpha Go beats Lee Sedol	The Age of AI



What is "Intelligence"?

Symbolics Machine

IBM's Watson

What is Artificial Intelligence (AI)

Human Intelligence

learning

reasoning

feeling

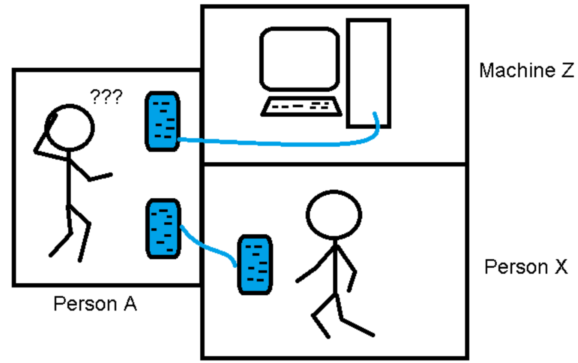
perceiving

understanding

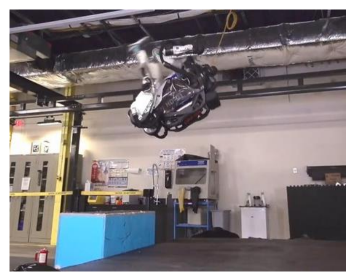
…

Artificial Intelligence

***Artificial Intelligence* is the science and engineering of making intelligent machines**



Turing Test (1950)



Boston Dynamics: Atlas

What Impact Has AlphaGo Achieved?

- **Search space is huge:** $\approx 10^{360}$

Reborn of Artificial Intelligence

Face recognition

Speech recognition

Natural language processing

Intelligent systems (e.g., self-driving)

AI will be everywhere

Software Engineering with AI

- **Software Engineering with Artificial Intelligence:**

  Employing Machine Learning (ML) techniques to assist in labor-intensive and error-prone tasks.

Software Engineering with Intelligence

Development

Source codes | Logging statements

Operation

User behavior data | User reviews

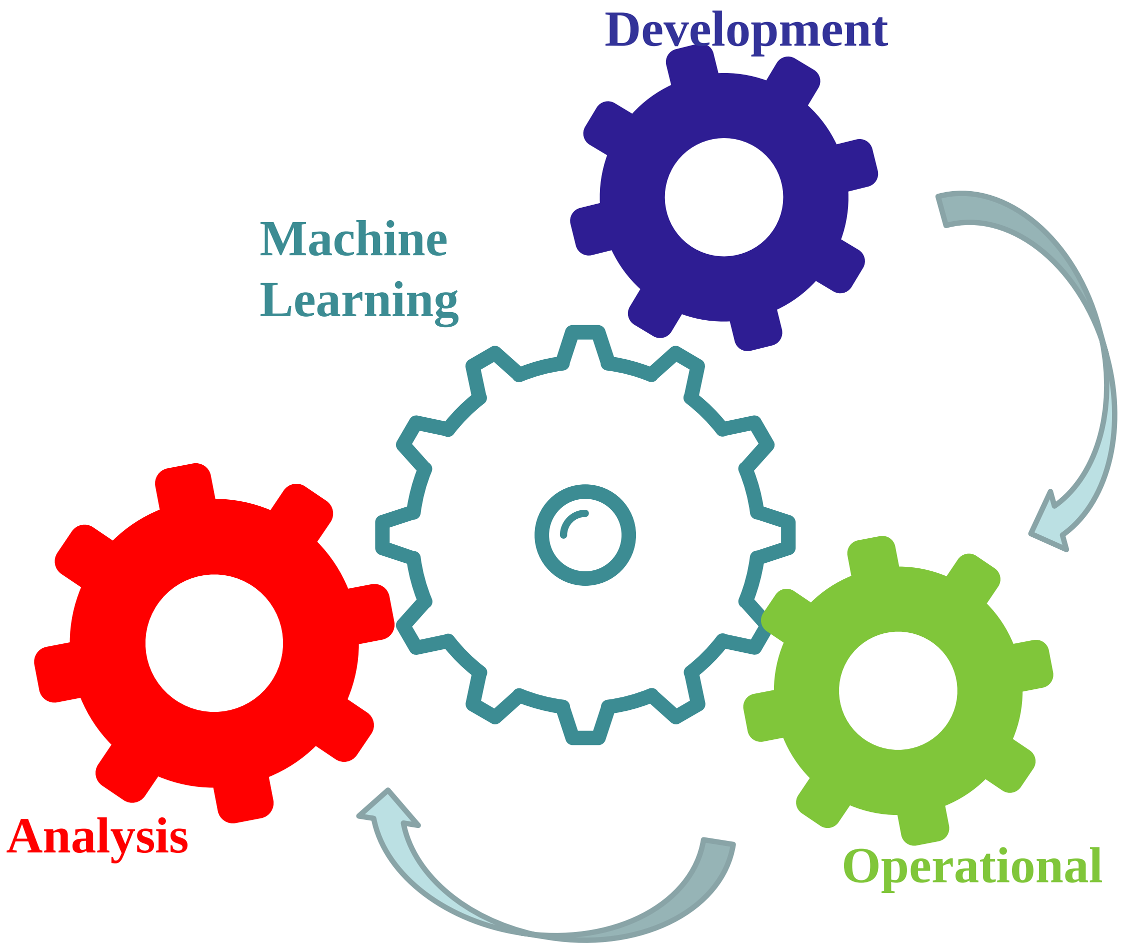
Analysis

QoS values | System logs

Apply ML Techniques

- Introduction
- **AI Techniques**
- Development Phase
- Operational Phase
- Analysis Phase
- Conclusion

Artificial Intelligence for Software Engineering



Artificial Intelligence for Software Engineering

Tasks		Techniques
Development	Code completion Learning to log	RNN with Attention Classification
Operational	App issues prioritizing Emerging issues detection	Classification, Topic modeling,
Analysis	Service reliability prediction Log Management	Matrix factorization, Classification, Parallel computing platform

Machine Learning

Machine Learning Framework

- **General framework:**

$$\arg\min_{\vec{\theta}} \Gamma(\{x_i, y_i\}_{i=1}^{N}; \vec{\theta}) + \Psi(\vec{\theta})$$

Unchanged Data

Frequently updated Parameter

- **Iterative Update:**

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

Very Big: cannot be handled with single PC

Gradient Descent:
Computed in Distributed Environment

Matrix Factorization

R

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
u_1	5	2		3		4		
u_2	4	3			5			
u_3	4		2				2	4
u_4								
u_5	5	1	2		4	3		
u_6	4	3		2	4		3	5

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
u_1	5	2	2.5	3	4.8	4	2.2	4.8
u_2	4	3	2.4	2.9	5	4.1	2.6	4.7
u_3	4	1.7	2	3.2	3.9	3.0	2	4
u_4	4.8	2.1	2.7	2.6	4.7	3.8	2.4	4.9
u_5	5	1	2	3.4	4	3	1.5	4.6
u_6	4	3	2.9	2	4	3.4	3	5

$$\boldsymbol{R \approx U^T V}$$

$$U = \begin{bmatrix} 1.55 & 1.22 & 0.37 & 0.81 & 0.62 & -0.01 \\ 0.36 & 0.91 & 1.21 & 0.39 & 1.10 & 0.25 \\ 0.59 & 0.20 & 0.14 & 0.83 & 0.27 & 1.51 \\ 0.39 & 1.33 & -0.43 & 0.70 & -0.90 & 0.68 \\ 1.05 & 0.11 & 0.17 & 1.18 & 1.81 & 0.40 \end{bmatrix} \quad V = \begin{bmatrix} 1.00 & -0.05 & -0.24 & 0.26 & 1.28 & 0.54 & -0.31 & 0.52 \\ 0.19 & -0.86 & -0.72 & 0.05 & 0.68 & 0.02 & -0.61 & 0.70 \\ 0.49 & 0.09 & -0.05 & -0.62 & 0.12 & 0.08 & 0.02 & 1.60 \\ -0.40 & 0.70 & 0.27 & -0.27 & 0.99 & 0.44 & 0.39 & 0.74 \\ 1.49 & -1.00 & 0.06 & 0.05 & 0.23 & 0.01 & -0.36 & 0.80 \end{bmatrix}$$

Low-Rank Matrix Factorization

- Objective function

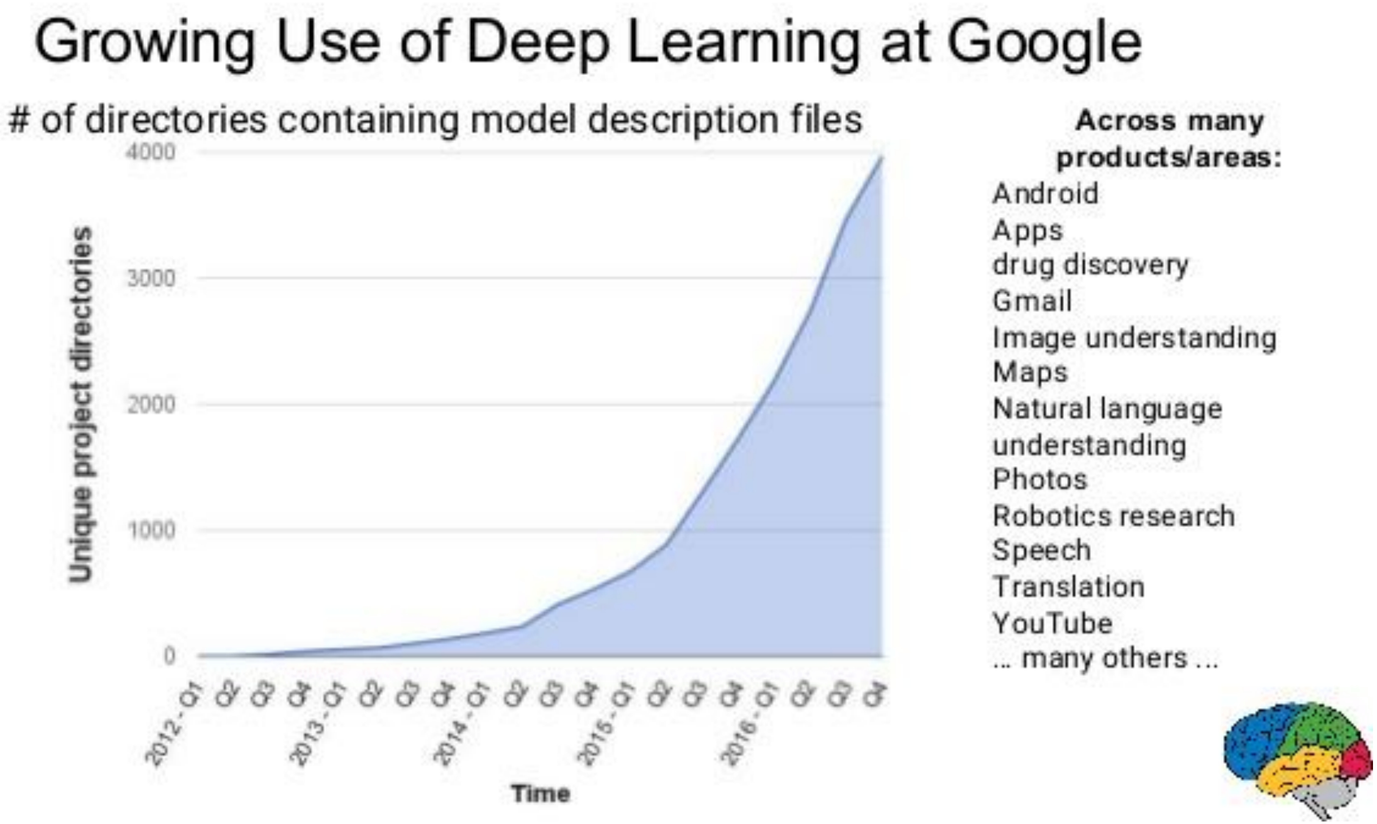
$$\min_{U,V} \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} \boxed{I_{ij}(R_{ij} - U_i^T V_j)^2} + \frac{\lambda_1}{2} ||U||_F^2 + \frac{\lambda_2}{2} ||V||_F^2$$

Main Objective **Regularization terms**

I_{ij} is the indicator function that is equal to 1 if user u_i rated item v_j and equal to 0 otherwise

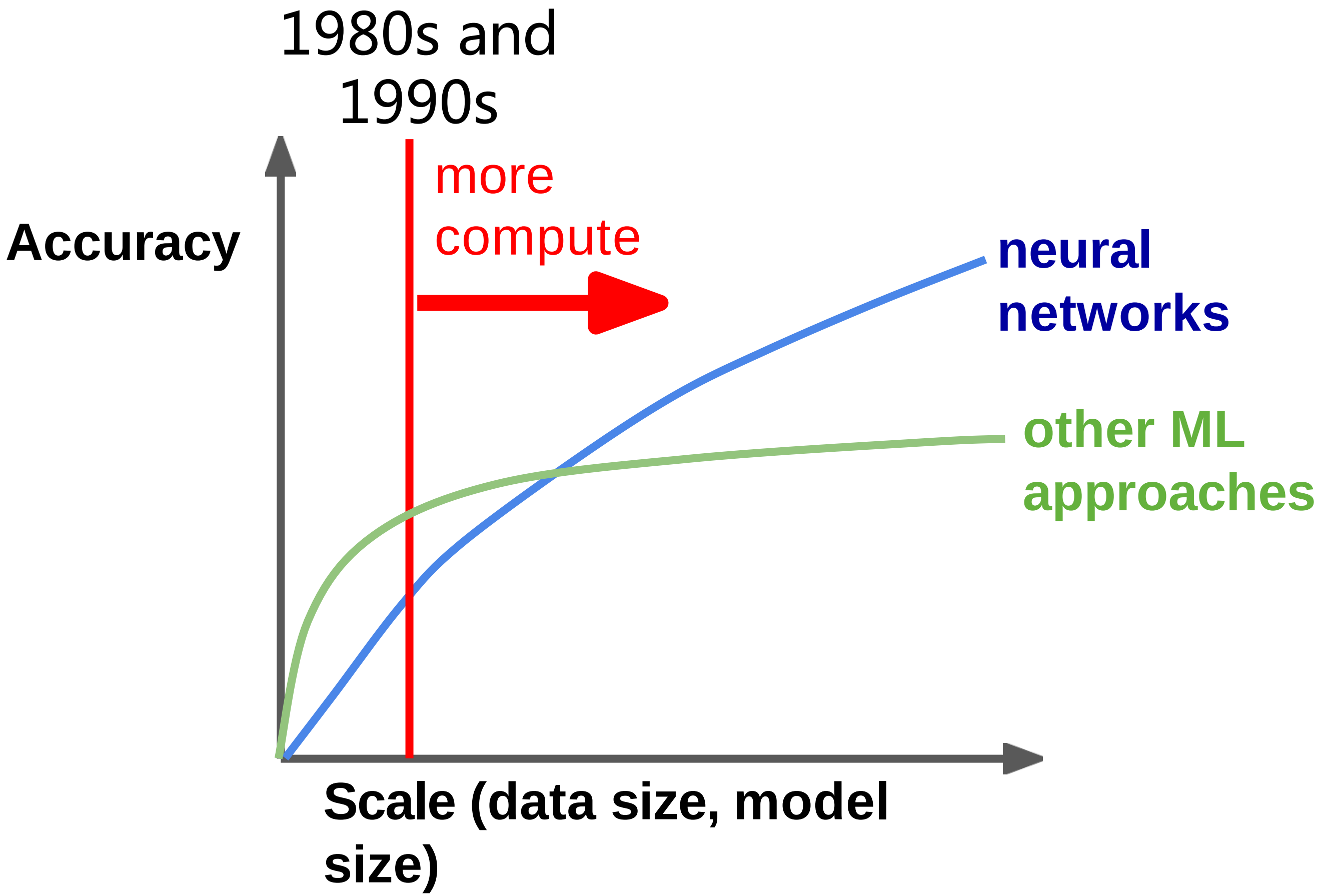
U_i, V_j: **low dimension column vectors to represent user/item preferences.**

The Growing of Deep Learning

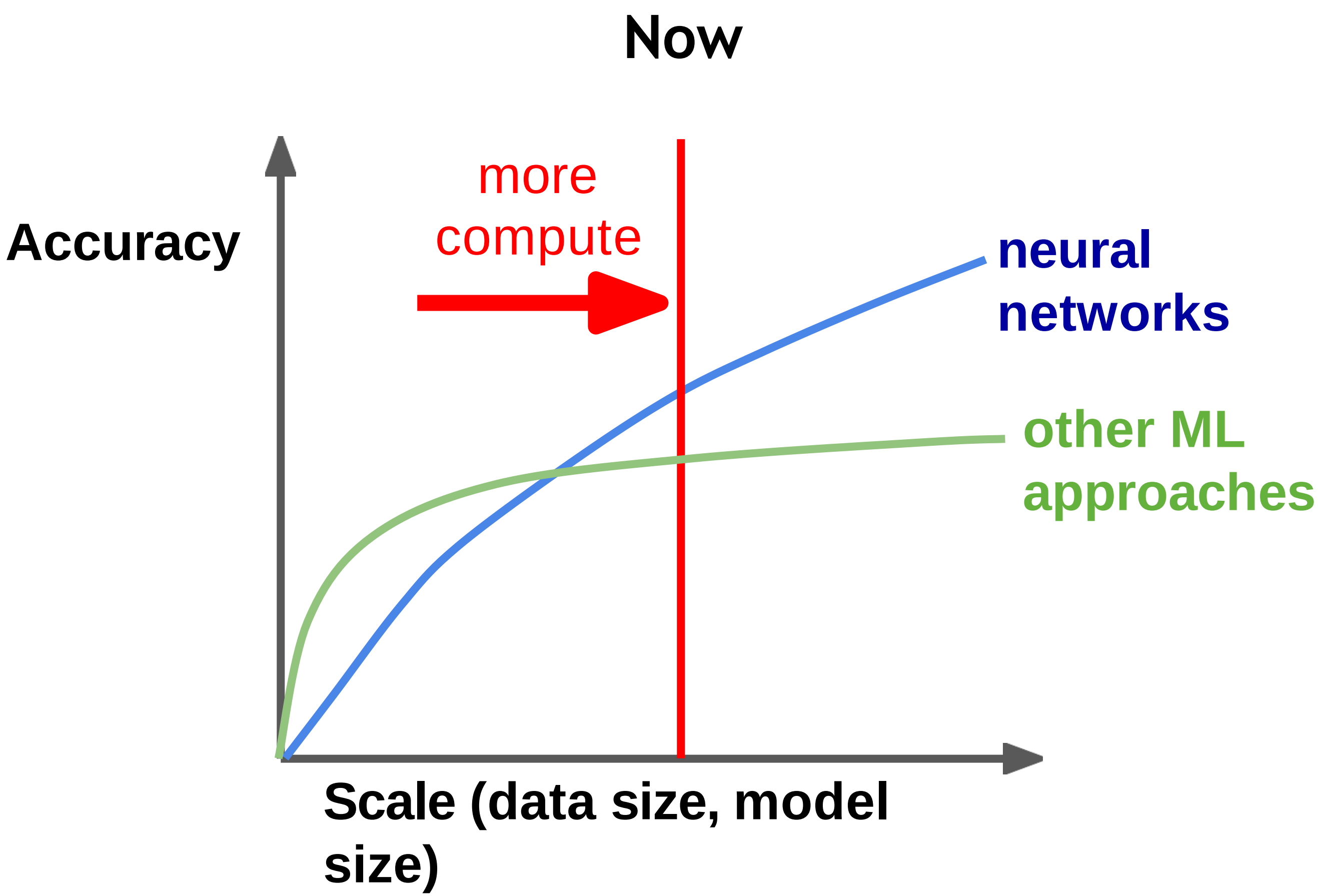


Deep learning trends at Google. Source: SIGMOD/Jeff Dean

Deep Learning Is Neural Networks



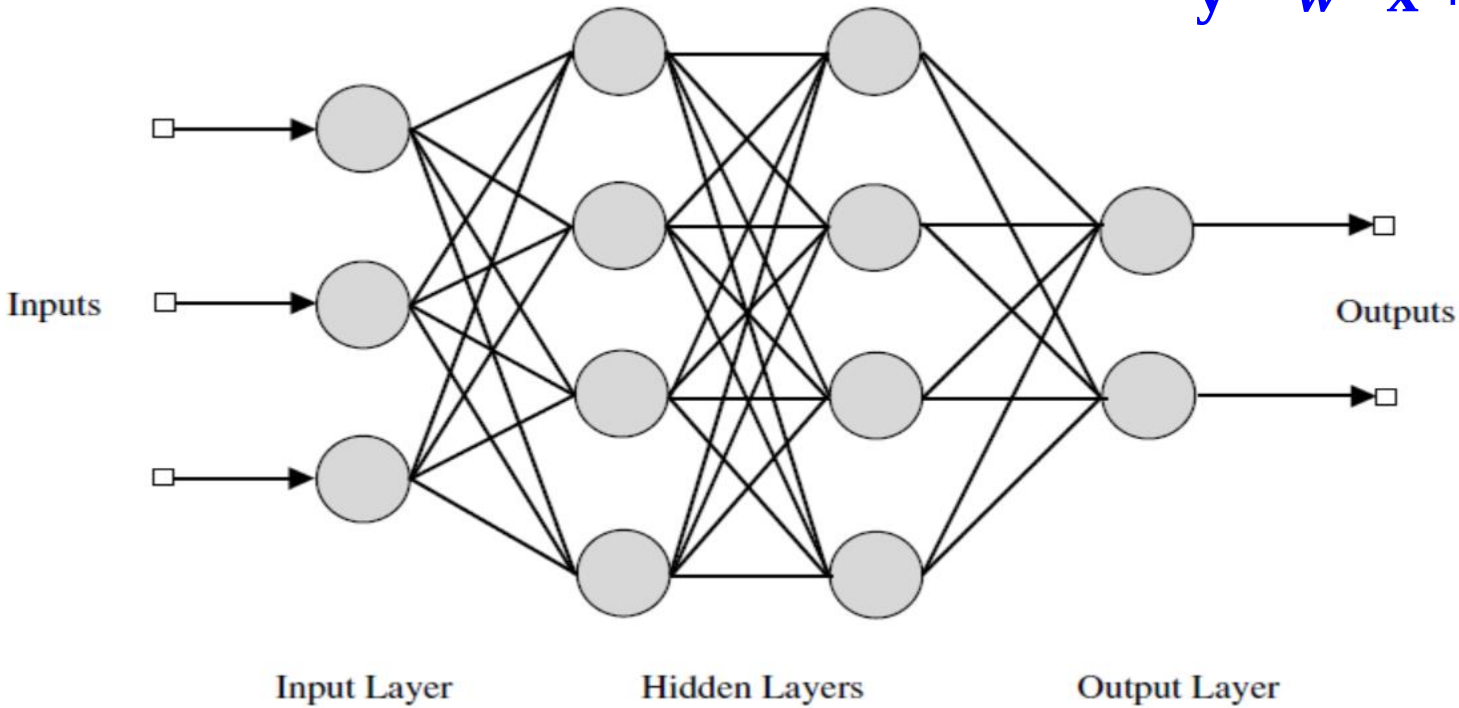
Deep Learning Is Neural Networks



Deep Learning

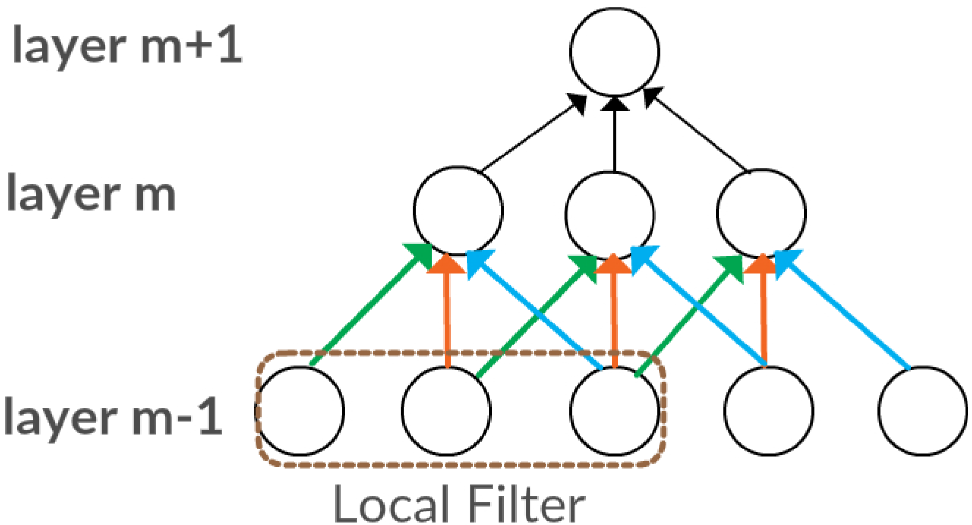
Feedforward Neural Networks (FFN)

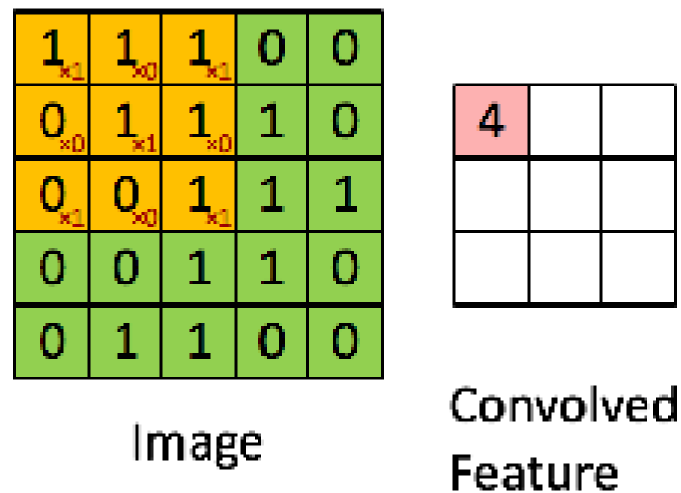
$$\mathbf{y} = \boldsymbol{w} \cdot \mathbf{x} + \boldsymbol{b}$$



Deep Learning

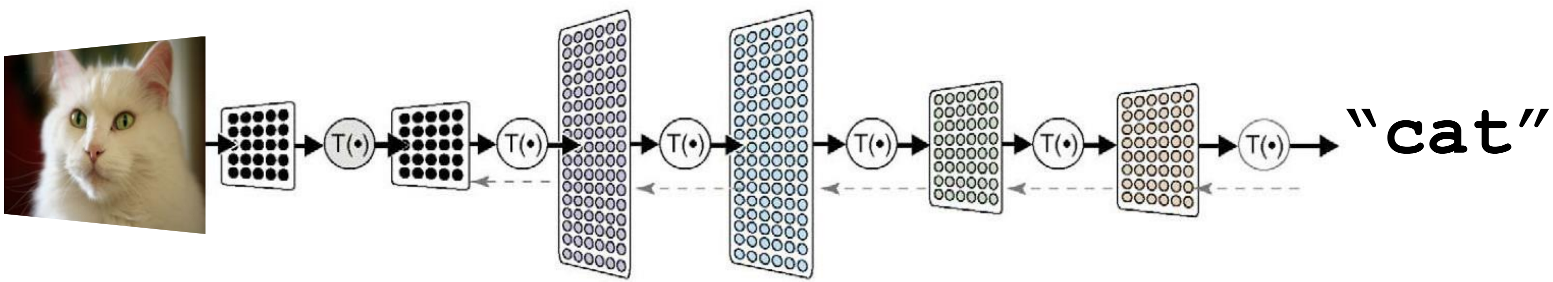
Convolutional Neural Networks (CNN)





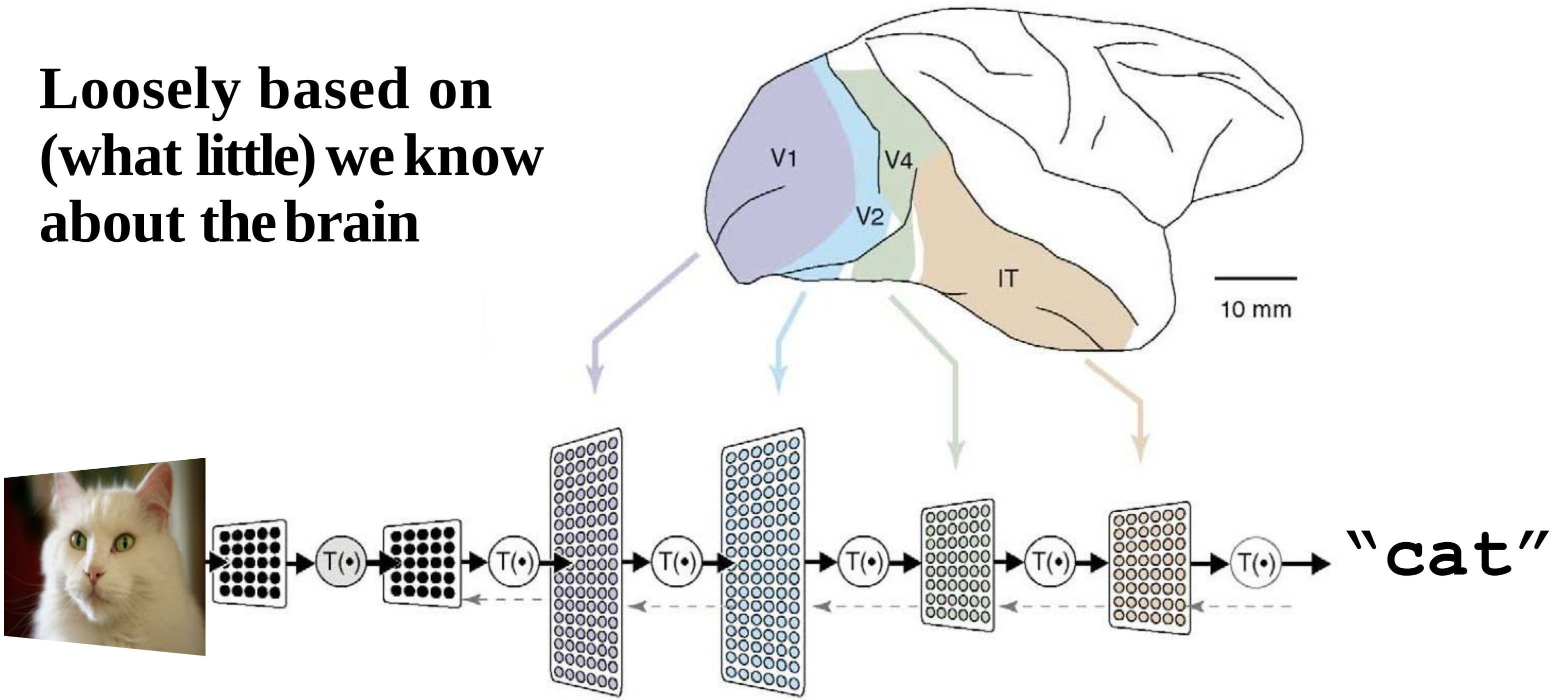
What is Deep Learning?

- A powerful class of machine learning model
- Modern reincarnation of artificial neural networks
- Collection of simple, trainable mathematical functions
- Compatible with many variants of machine learning

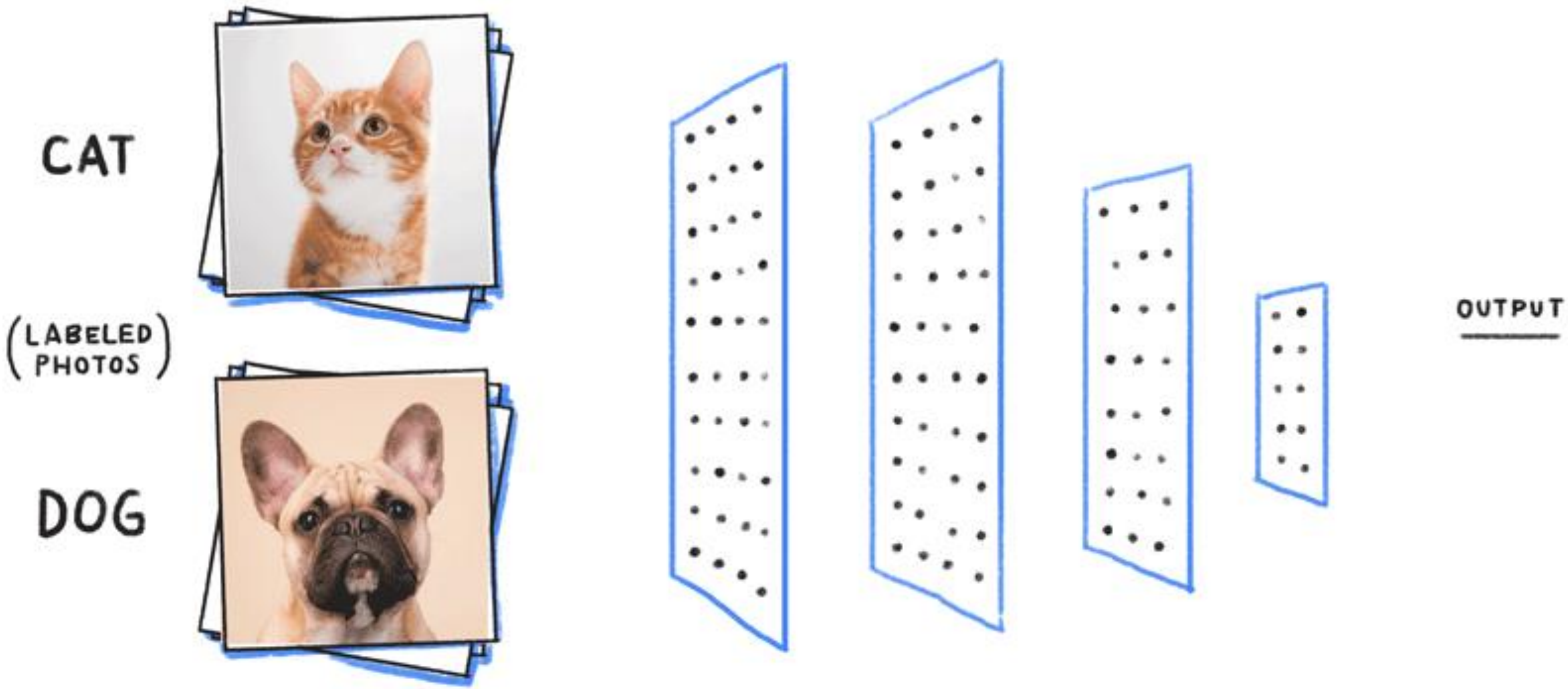


What is Deep Learning?

- **Loosely based on (what little) we know about the brain**

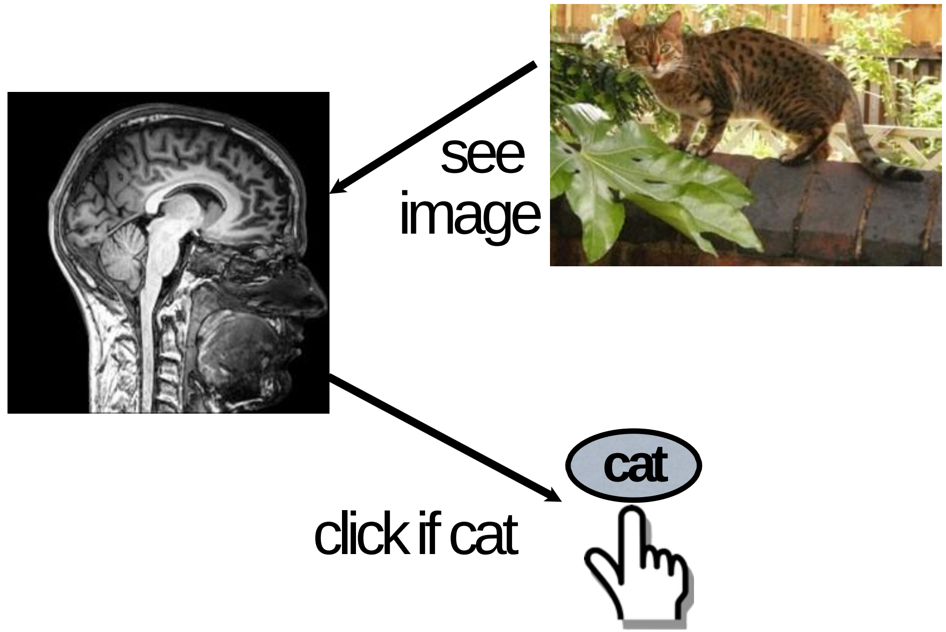


How Do Neural Networks Work?



How Do Neural Networks Work?

0.1 sec:
neurons fire
only 10 times!



Anything humans can do in 0.1 sec, the right big 10-layer network can do too

Computers Can Now See

Combining Vision with Robotics

[*“Deep Learning for Robots: Learning from Large-Scale Interaction”*](), Google Research Blog, March, 2016

“Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection”, Sergey Levine, Peter Pastor, Alex Krizhevsky, & Deirdre Quillen, Arxiv, arxiv.org/abs/1603.02199

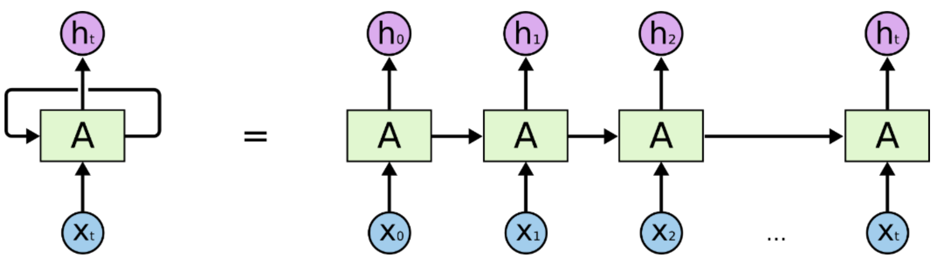
What Can Neural Networks Compute?

Human perception is very fast (0.1 second)

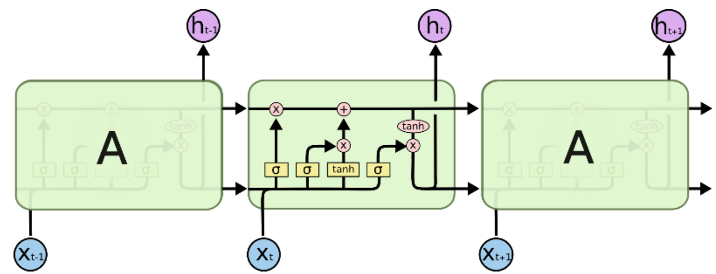
- Recognize objects (“see”) 
- Recognize speech (“hear”) 
- Recognize emotion 
- Instantly see how to solve some problems
- And many more!

These can all be computed by Neural Networks.

Recurrent Neural Networks (RNN)



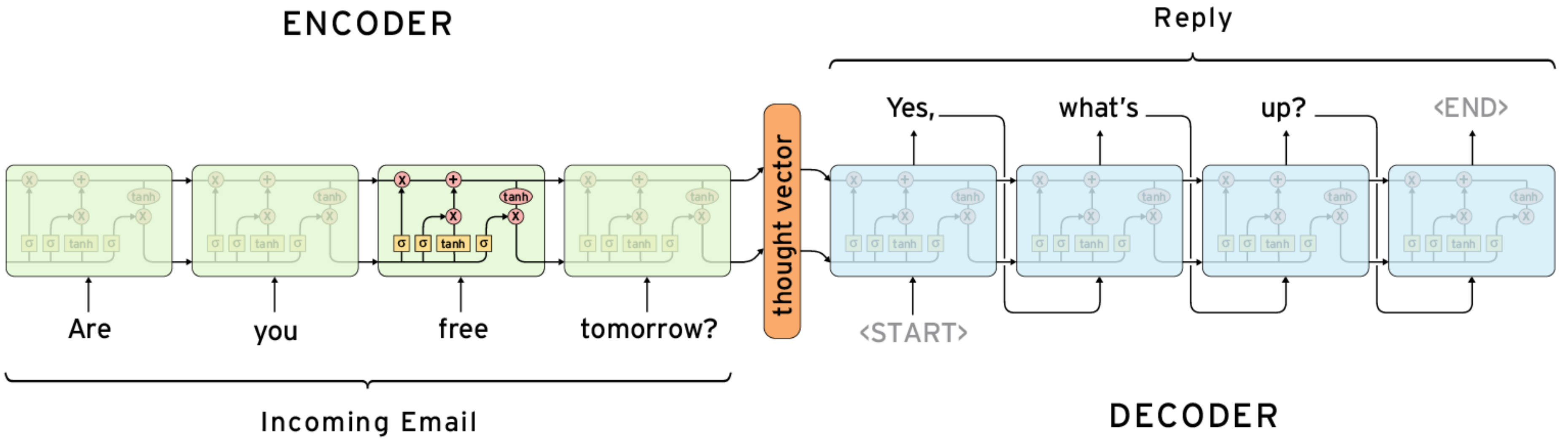
A standard RNN



An LSTM network

Deep Learning

Sequence-to-Sequence Models



Deep Learning Platforms

TensorFlow

PYTORCH

dmlc
mxnet

Microsoft
CNTK

K

Caffe2

Caffe

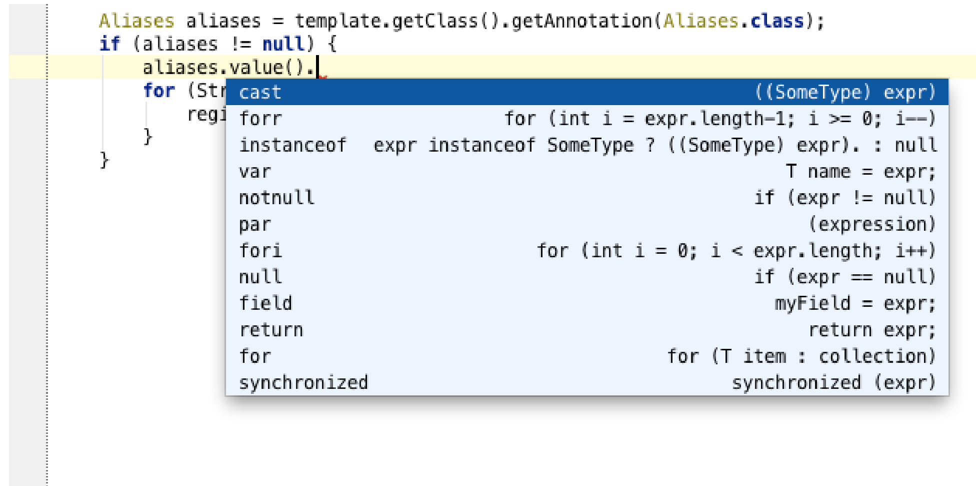
Chainer

theano

- Introduction
- AI Techniques
- **Development Phase**
  - Code completion
  - Learning to log
- Operational Phase
- Analysis Phase
- Conclusion

Development: Code Completion

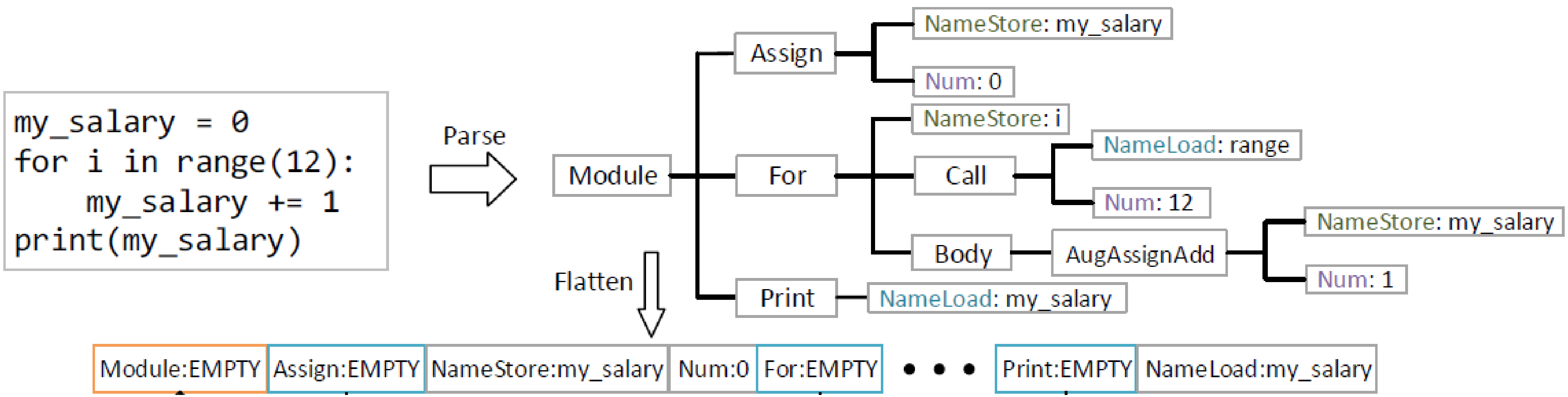
- Code completion



- Intelligent code completion is essential for software engineers
- Programming languages: static vs dynamic
- Out-of-Vocabulary (OoV) problem: many words are sparse, e.g. user-defined identifiers

Development: Code Completion

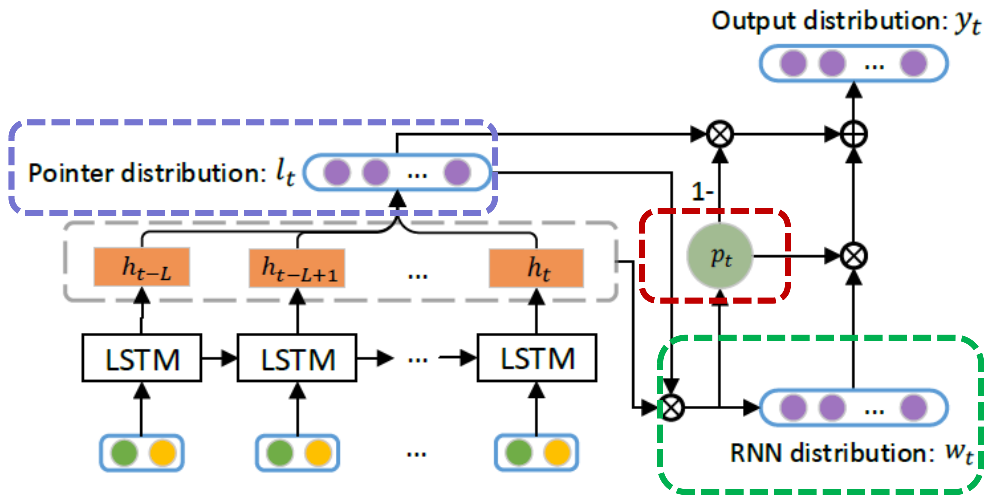
- Abstract syntax tree (AST)
- Locally repeated terms



A Python program and its corresponding abstract syntax tree

Development: Code Completion

- Pointer mixture network
  - Global RNN component
  - Local pointer component
  - Controller



- Contributions:
  - Pointer mixture network for better predicting OoV words
  - Effectiveness of attention mechanism
  - Significant improvements in code completion task

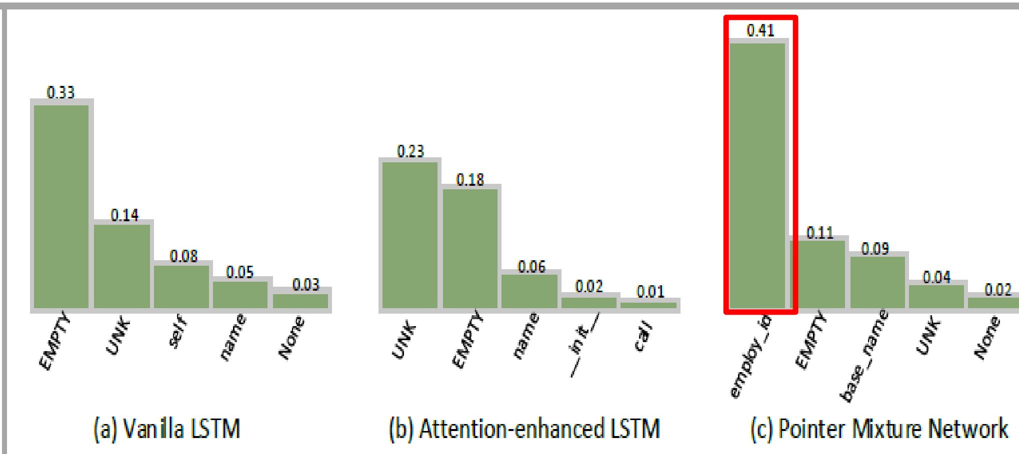
Development: Code Completion

- Case study

```
class Operator(Employee):
  def __init__(self, name, employee_id):
    super(Operator, self).__init__(name, Rank.OPERATOR)
    self.employee_id = employee_id

  def _dispatch_call(self, call, employees):
    for employee in employees:
      employee.take_call(call)

  def record_path(self, base_name):
    return os.path.join(base_name, str(self.___?___))
```



- Pointer Mixture Network successfully point to employee_id, which is an OoV word

Development: Code Completion

- Dataset
  - JavaScript (JS) and Python (PY)

Table 1: Dataset Statistics

	JS	PY
Training Queries	$10.7 * 10^7$	$6.2 * 10^7$
Test Queries	$5.3 * 10^7$	$3.0 * 10^7$
Type Vocabulary	95	329
Value Vocabulary	$2.6 * 10^6$	$3.4 * 10^6$

- Accuracies on next value prediction with different vocabulary sizes

Table 2: Accuracies on next value prediction with different vocabulary sizes. The out-of-vocabulary (OoV) rate denotes the percentage of AST nodes whose value is beyond the global vocabulary.

	JS			PY		
Vocabulary Size (OoV Rate)	1k (20%)	10k (11%)	50k (7%)	1k (24%)	10k (16%)	50k (11%)
Vanilla LSTM	69.9%	75.8%	78.6%	63.6%	66.3%	67.3%
Attention-enhanced LSTM (ours)	71.7%	78.1%	80.6%	64.9%	68.4%	69.8%
Pointer Mixture Network (ours)	**73.2%**	**78.9%**	**81.0%**	**66.4%**	**68.9%**	**70.1%**

Development: Code Completion

- Comparisons against the state-of-the-arts
  – Note that Pointer Mixture Network can be only used for predicting VALUE node (TYPE node has small size of vocabulary)

Table 3: Comparisons against the state-of-the-arts. The upper part is the results from our experiments while the lower part is the results from prior work. TYPE means next node type prediction and VALUE means next node value prediction.

	JS		**PY**	
	TYPE	VALUE	TYPE	VALUE
Vanilla LSTM	87.1%	78.6%	79.3%	67.3%
Attention-enhanced LSTM (ours)	**88.6%**	80.6%	**80.6%**	69.8%
Pointer Mixture Network (ours)	-	81.0%	-	**70.1%**
LSTM (Liu et al. 2016)	84.8%	76.6%	-	-
Probabilistic Model (Raychev et al. 2016)	83.9%	**82.9%**	76.3%	69.2%

- Observations: our models outperform the state-of-the-art in almost all cases

Development: Learning to Log

- Challenges of logging
  - Logging too little
    - Miss valuable runtime information
    - Increase the difficulty for problem diagnosis

User:
"Apache httpd cannot start.
No log message printed."

  - Logging too much
    - Additional cost of code dev. & maintenance
    - Runtime overhead
    - Producing a lot of trivial logs
    - Storage overhead

Development: Learning to Log

- What is logging?

```
Log (level, "logging message %s", variable);
```

  - A common programming practice to record runtime system information
  - Logging functions: e.g., printf, cout, writeline, etc.

- Logs are crucial for system management
  - Various tasks of log analysis
    - Anomaly detection, failure diagnosis, etc.
  - The only data available for diagnosing production failures

Logging is important!

Development: Learning to Log

- Focused snippets: potential error sites
  - Exception snippets: try-catch blocks
  - Return-value-check snippets: function-return errors

Example 1

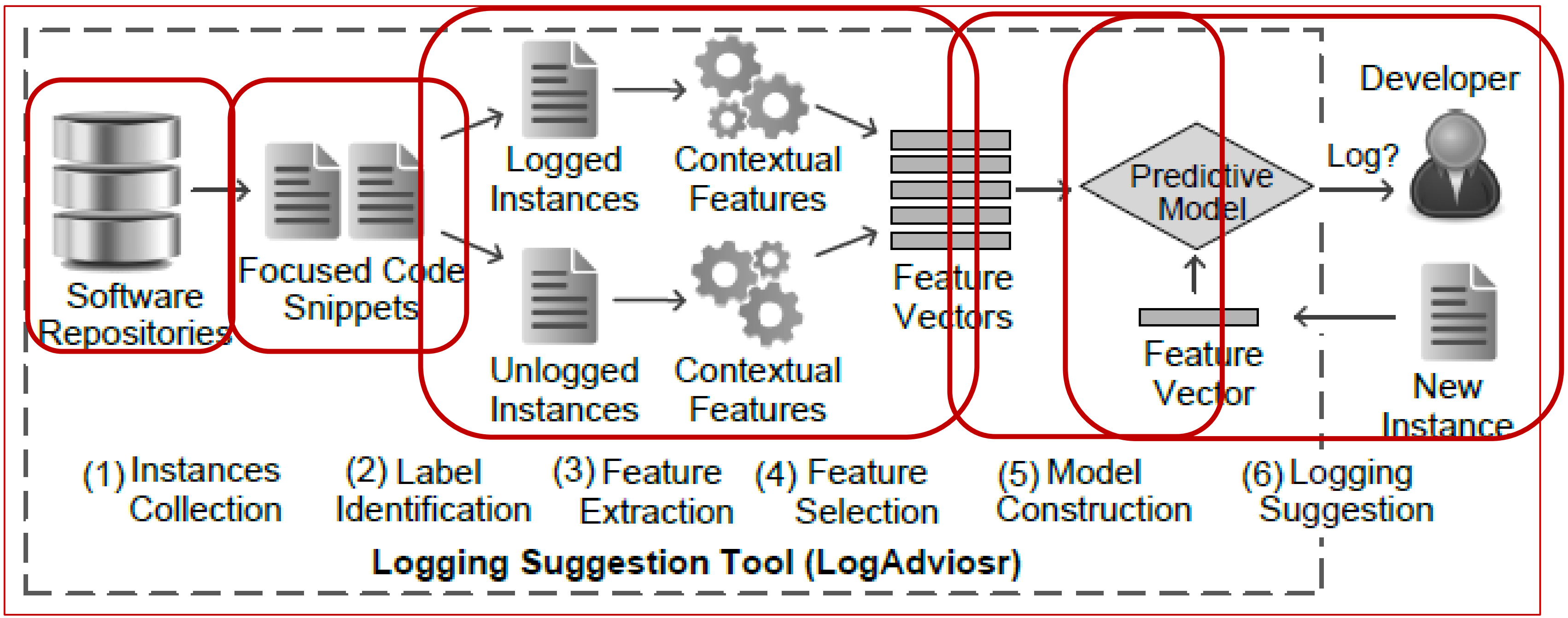
```
try {
    method(…);
}
catch (IOException) {
    log(…);
    …
}
```

Example 2

```
var res = method(…);
if (res == null) {
    log(…);
    …
}
```

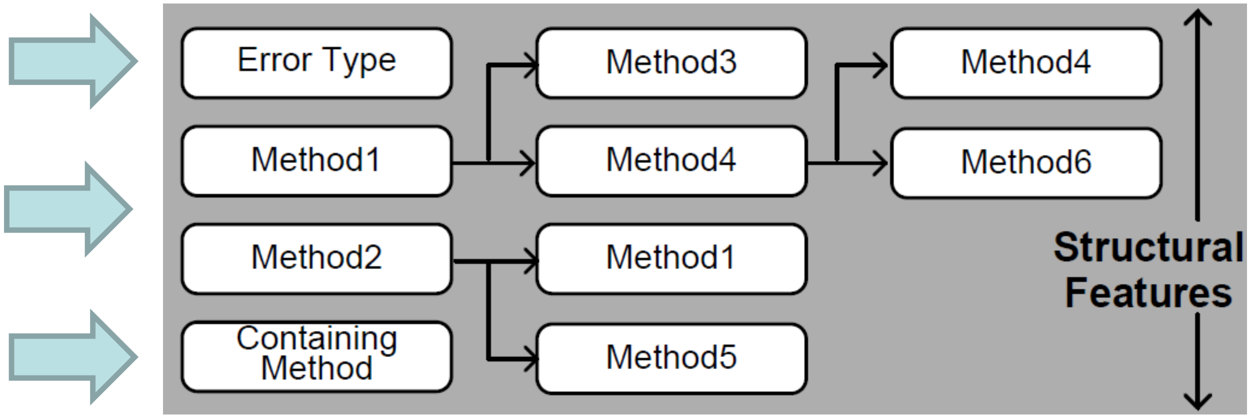
Development: Learning to Log

- Framework of learning to log
  – Similar to other machine learning applications (e.g., defect prediction)



Development: Learning to Log

- Structural features: structural info of code




```
private int LoadRulesFromAssembly (string assembly, ...){
    //Code in Setting
    try {
            AssemblyName aname = AssemblyName.
            GetAssemblyName(Path.GetFullPath (assembly));
            Assembly a = Assembly.Load (aname);
    }
    catch (FileNotFoundException) {
            Console.Error.WriteLine ("Could not load rules
            From assembly '{0}'.", assembly); return 0; }
            ... }
}
```

Exception Type:
0.39 (System.IO.FileNotFoundException)

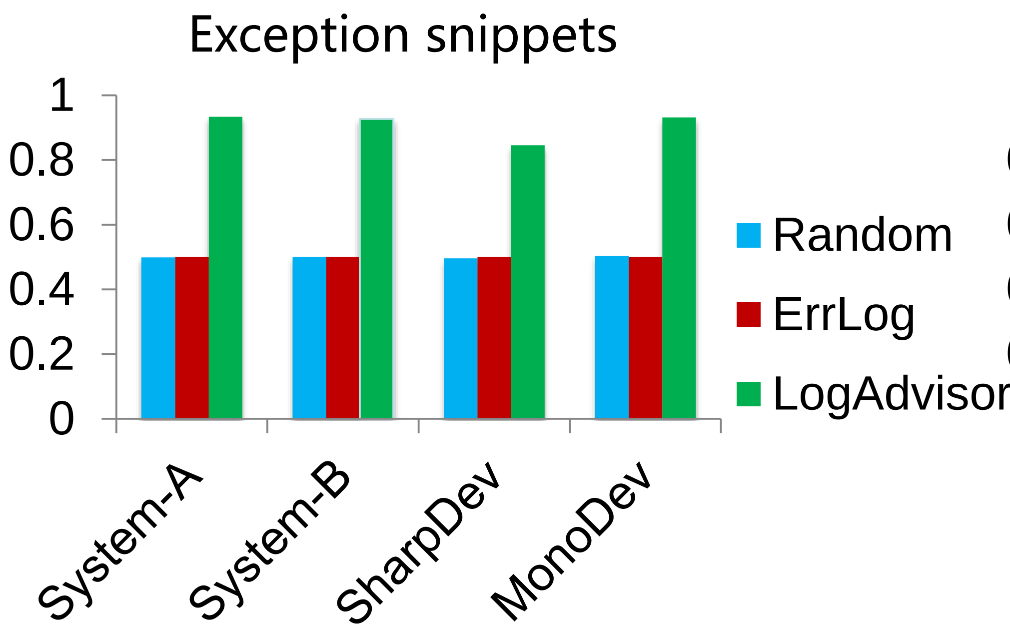
Containing method:
Gendarme.Settings.LoadRulesFromAssembly

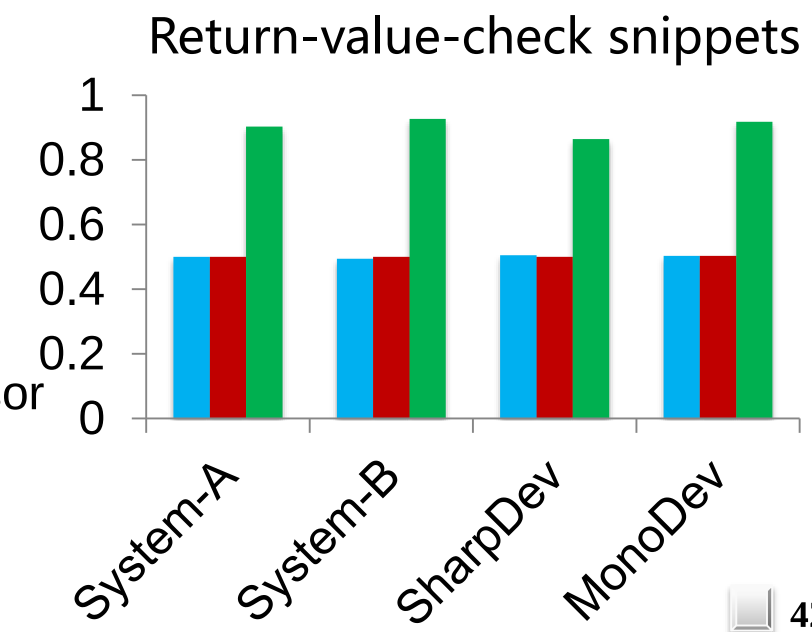
Invoked methods:
System.IO.Path.GetFullPath,
System.Reflection.AssemblyName.GetAssemblyName,
System.Reflection.Assembly.Load

/* A code example taken from MonoDevelop (v.4.3.3), at file: * main\external\mono-tools\gendarme\console\Settings.cs,
* line: 116. Some lines are omitted for ease of presentation. */

Development: Learning to Log

- Within-project evaluation
  – **Random**: randomly logging (as a new developer)
  – **ErrLog** [Yuan et al., OSDI'12]: conservatively logging all focused snippets
  – **LogAdvisor**: 0.846 ~ 0.934 accuracy achieved





香港中文大学
Chinese University of Hong Kong

- Introduction
- AI Techniques
- Development Phase
- **Operational Phase**
  - App issues prioritizing
  - Emerging issues detection
- Analysis Phase
- Conclusion

Operation: App Issues Prioritizing

- **User reviews are valuable source for pinpointing emerging issues for app development.**
- **Capturing user-concerned issues and tracking their trends**

June 1, 2015

Getting stupid Not sure what you've done but now I get a folder called commondir and it wants to clean it and says its safe to delete, its 12gb and that 12 gb is over half my internal storage son that must be my apps and its saying safe to delete?

May 30, 2015

Love it Great app. But only one problem. I had Some problem with my screen and did a wrong pattern and it has taken my photos. Plz let me know how do I delete those "intruder selfie's".

2,800,000 apps **2,200,000 apps** **669,000 apps**

ANDROID APP ON Google play

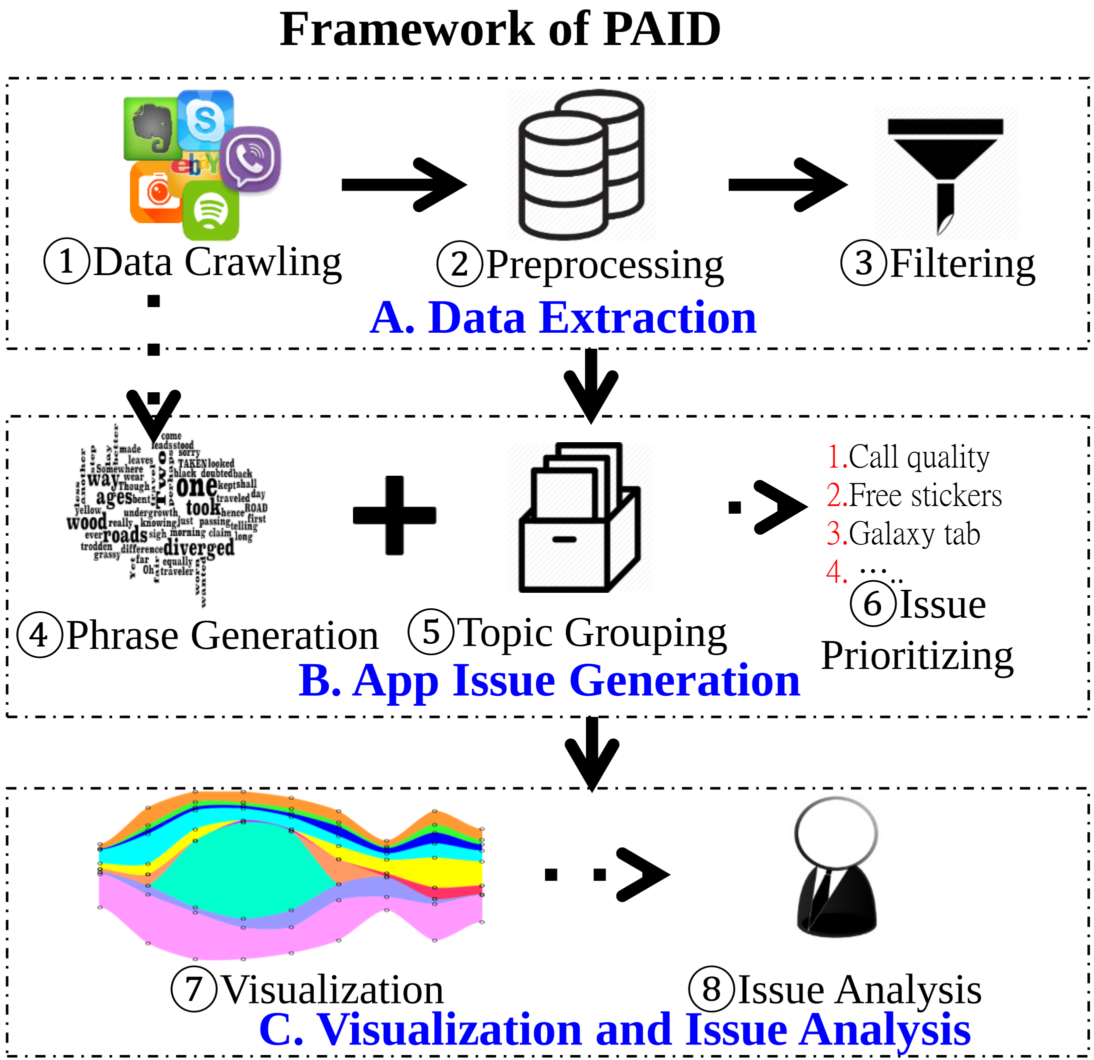
Changelog

What's New

5.9.8

1. Optimized Photo Manager - Easier than ever to manage photos and save space
2. Added anti-intruder feature in AppLock to protect your privacy - Someone who tries to break into your phone will have their picture snapped
3. Various other improvements

Operation: App Issues Prioritizing



Operation: App Issues Prioritizing

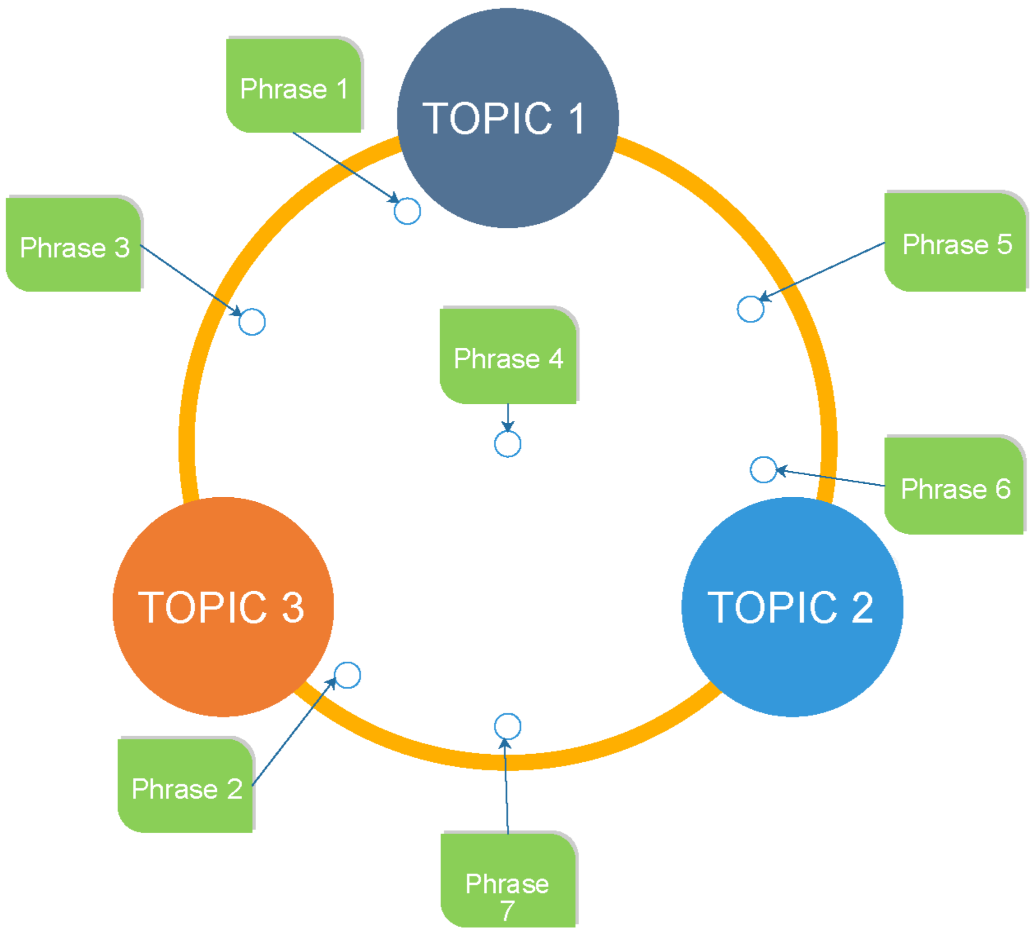
Google Play

Crawler

App #	37
Review #	2,089,737
Time Period	10 months

ID	Title	Review	Date	Stars	Version
1	Crash	Like it cause it doesn’t crash on androids	2014-11-09T08:55:47	5	15.0.0.15.13
2	Rubbish	When I try to connect with Mobile Network Package, this don't work and giving "Connecting.. Problem".	2014-11-12T18:32:25	1	15.0.0.15.10

Operation: App Issues Prioritizing



Based on topic modeling, each topic is labeled with one phrase.

Topic Labeling Process:

Rank phrases for each topic by:

- Semantic aspect:
  KL-Divergence

$$Sem(\beta_i, l) = Sim(\beta_i, l) - \frac{\mu}{k-1} \sum_{j \neq i} Sim(\beta_j, l)$$

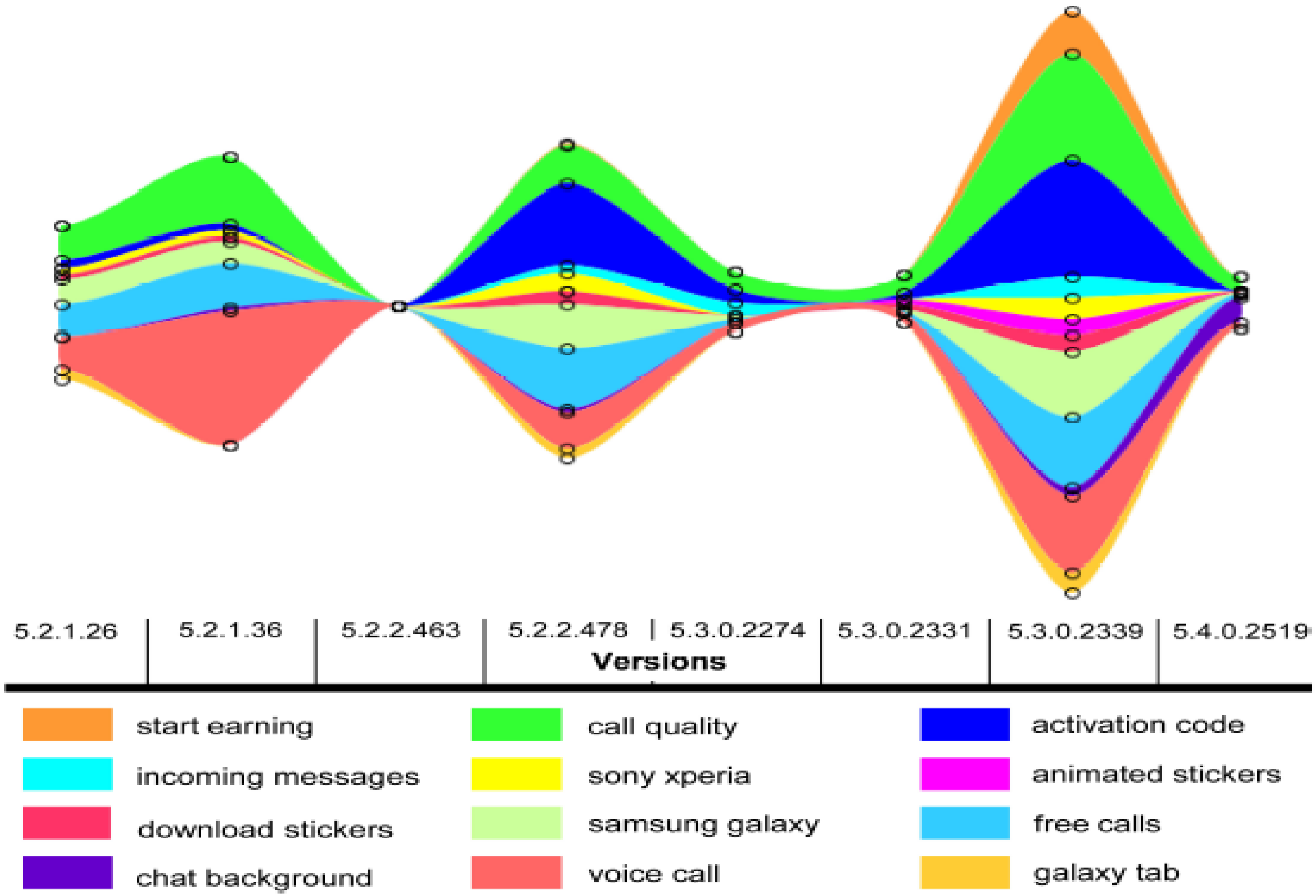
- Sentiment aspect:

$$Sen(l) = e^{\frac{-r}{ln(h)}}$$

- Total score:

$$S(\beta_i, l) = Sem(\beta_i, l) * Sen(l)$$

Operation: App Issues Prioritizing



The Themeriver of Viber.

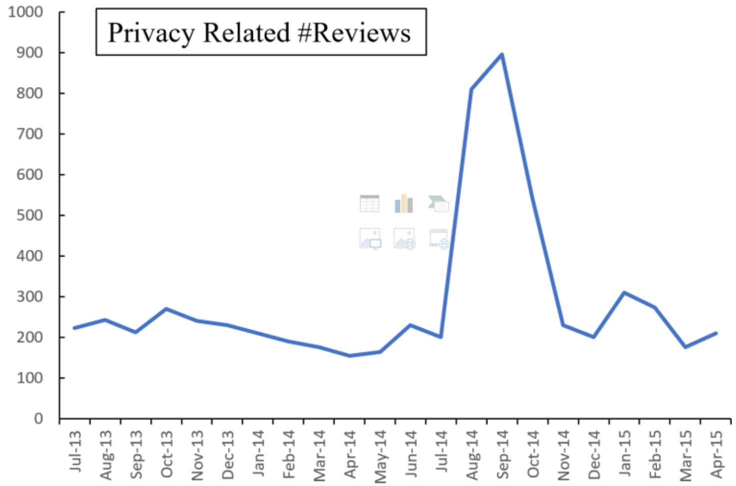
Operation: App Issues Prioritizing

Rank top reviews for each topic:

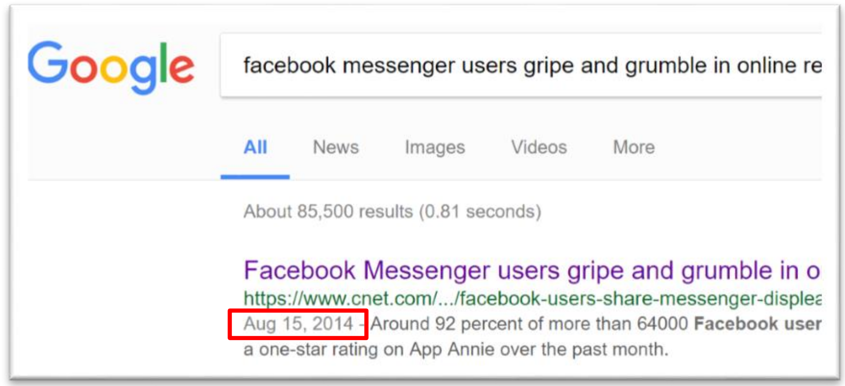
The Top Three Reviews Related to "Activation Code"

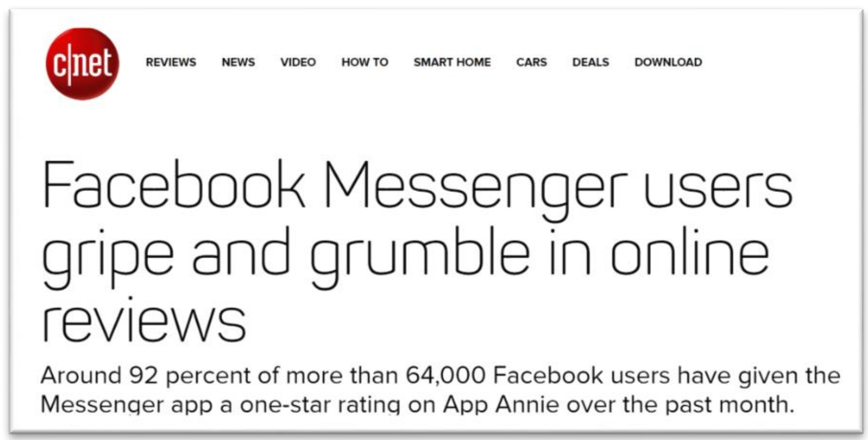
	User Review	Importance Score
1	Upload viber! I went. Enter a phone number. I enter. Asks for sure your phone? It will be sent an activation code. Ok. Messages are not present. He writes to activate viber here, install it to your phone first. But I have it pumped? What to do? Help!	0.836
2	I hard reset my tab 3. Installed viber for activation code when i write my phone number and press okay a white popup written only. ERROR no description given and an okay button on it please help me vibers my only way to contact my son abroad.	0.834
3	I don't know what's wrong with Viber. Just downloaded it nd it keeps on saying activation code sent to your device. For almost a month, no any activation code and it's really pissing me off. Pls fix.	0.828
...	...	...

Operation: Emerging Issues Detection



Number of Privacy Related Reviews in Facebook during July 2013 to April 2015



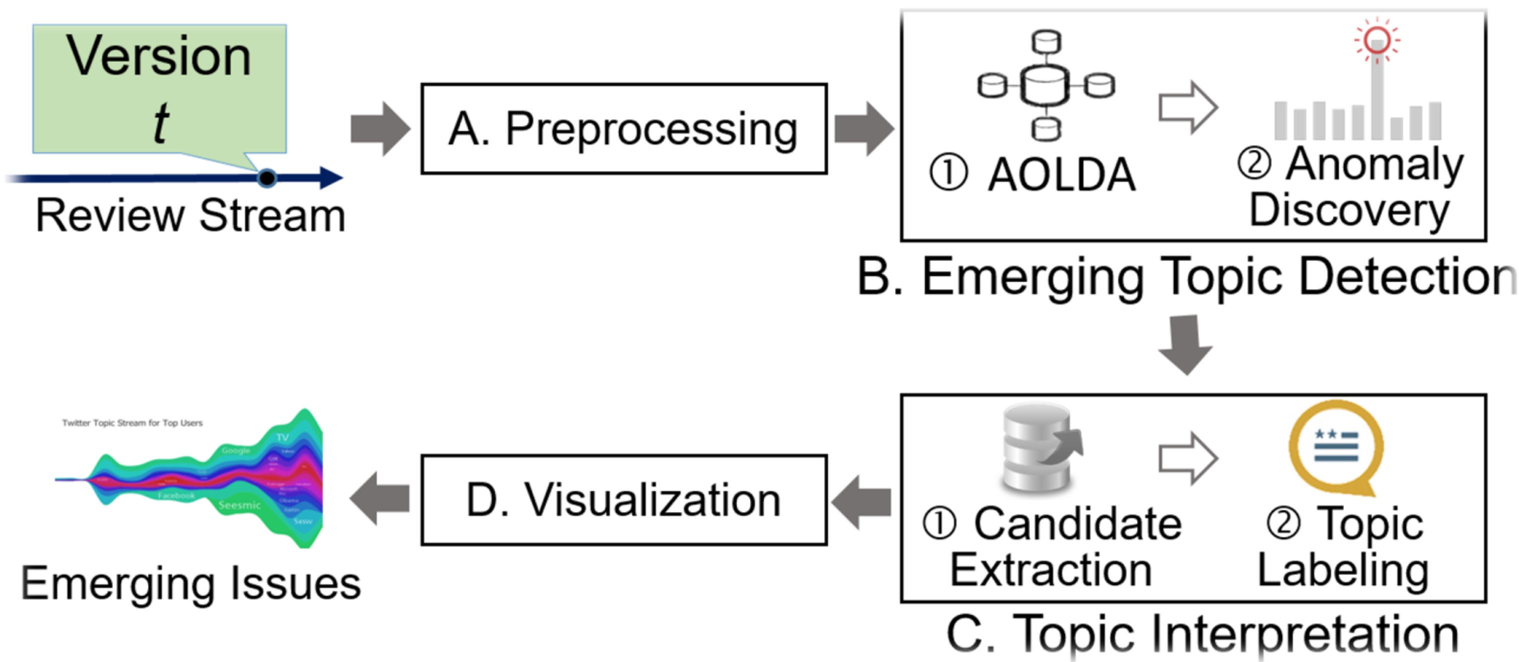


IDEA

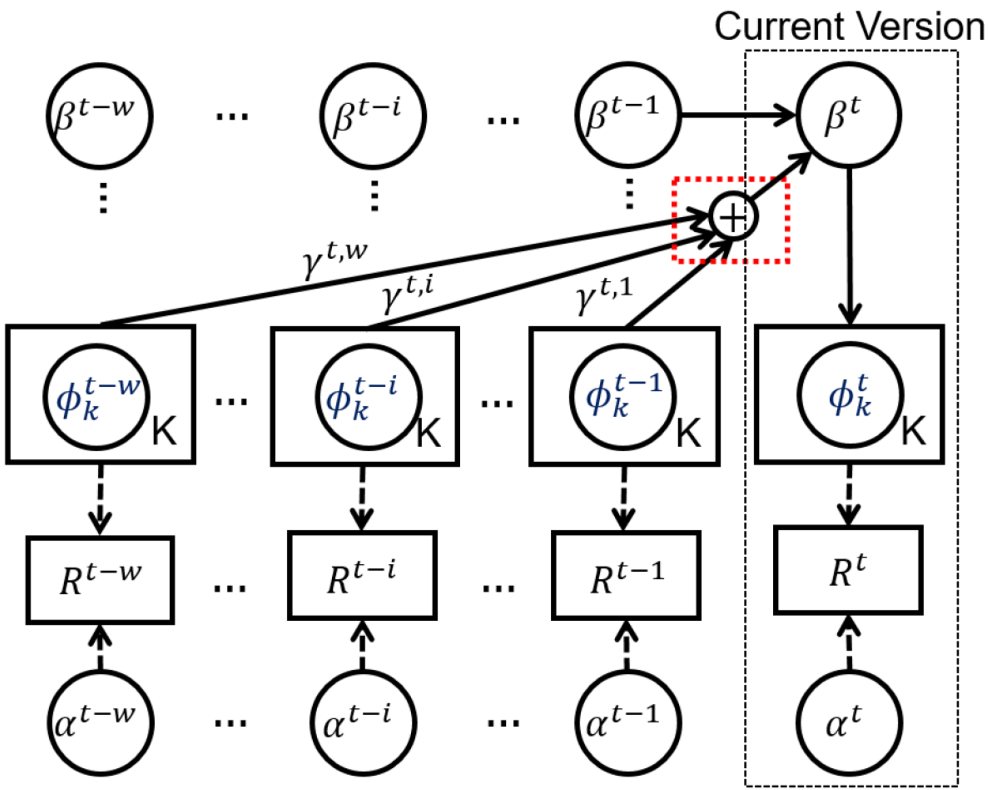
IDentifying Emerging Issues from App Reviews

- Automatic tool for app review analysis
- Discovering emerging issues dynamically
- Comprehensive issue interpretation
- Visualizing issue progression over versions

Overall Framework

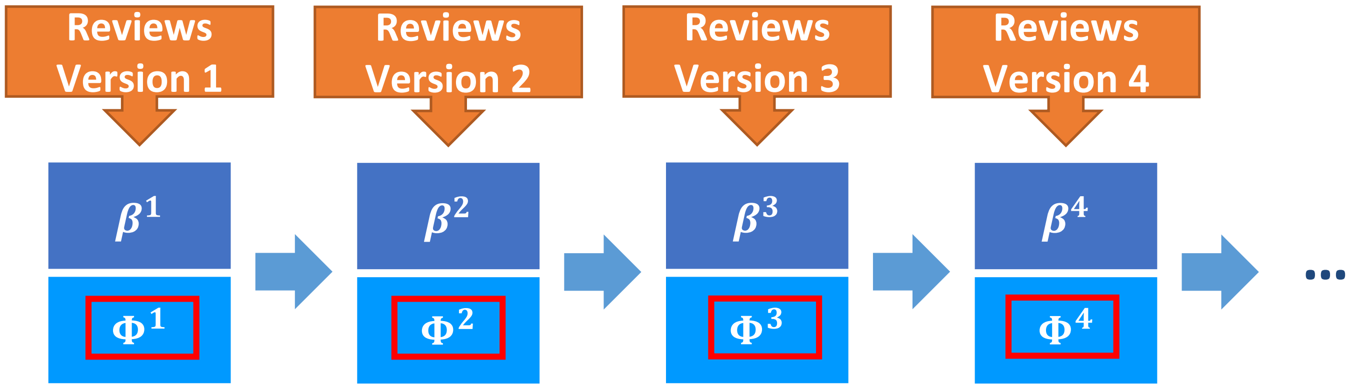


Online Topic Modeling



Overview of AOLDA (Adaptively Online Latent Dirichlet Allocation). The red rectangle with dashed dots highlights the adaptive integration of the topics of the w previous versions.

Emerging Issue Detection



Anomaly Detection - Jensen-Shannon Divergence

$$D_{JS}(\phi_k^t||\phi_k^{t-1}) = \frac{1}{2}D_{KL}(\phi_k^t||M) + \frac{1}{2}D_{KL}(\phi_k^{t-1}||M)$$

$$D_{KL}(P||Q) = \sum_i P(i)\log\frac{P(i)}{Q(i)}.$$

Experimental Result

App Name (#avg. reviews)	Method	Phrase			Sentence		
		Precision_E	Recall_L	F_{hybrid}	Precision_E	Recall_L	F_{hybrid}
NOAA Radar (523)	OLDA	0.468	0.528	0.473	**0.482**	0.622	0.534
	IDEA-R	**0.606**	0.461	0.520	0.478	0.570	0.503
	IDEA-S	0.250	**0.530**	0.340	0.417	0.547	0.473
	IDEA^+	0.571	0.497	**0.531**	0.476	**0.639**	**0.546**
Youtube (1,143)	OLDA	0.441	0.462	0.451	0.578	0.664	0.597
	IDEA-R	0.506	0.429	0.456	0.550	0.659	0.586
	IDEA-S	0.548	0.466	0.502	0.456	0.656	0.522
	IDEA^+	**0.592**	**0.472**	**0.523**	**0.628**	**0.666**	**0.636**
Viber (2,141)	OLDA	0.157	0.305	0.166	0.313	0.550	0.375
	IDEA-R	0.542	0.326	0.407	**0.625**	0.571	0.597
	IDEA-S	0.500	**0.342**	0.406	0.500	0.518	0.509
	IDEA^+	**0.625**	0.340	**0.440**	**0.625**	**0.651**	**0.638**
Clean Master (6,332)	OLDA	0.300	0.269	0.160	0.200	0.421	0.129
	IDEA-R	0.500	0.216	0.301	**0.750**	0.377	0.502
	IDEA-S	0.067	0.289	0.366	0.500	0.398	0.443
	IDEA^+	**0.667**	**0.318**	**0.431**	0.667	**0.434**	**0.526**
Ebay (3,943)	OLDA	0.167	0.238	0.196	0.500	0.488	0.494
	IDEA-R	**0.229**	0.243	0.220	**0.646**	0.496	0.561
	IDEA-S	0.125	**0.285**	0.132	0.354	0.476	0.406
	IDEA^+	**0.229**	0.251	**0.227**	**0.646**	**0.527**	**0.580**
SwiftKey (1,313)	OLDA	0.100	0.567	0.148	0.367	0.617	0.458
	IDEA-R	0.333	0.611	0.376	0.417	**0.733**	0.515
	IDEA-S	0.333	0.622	0.372	0.500	0.711	**0.587**
	IDEA^+	**0.517**	**0.653**	**0.523**	**0.583**	0.700	**0.587**

https://remine-lab.github.io/

香港中文大学
Chinese University of Hong Kong

Introduction

AI Techniques

Development Phase

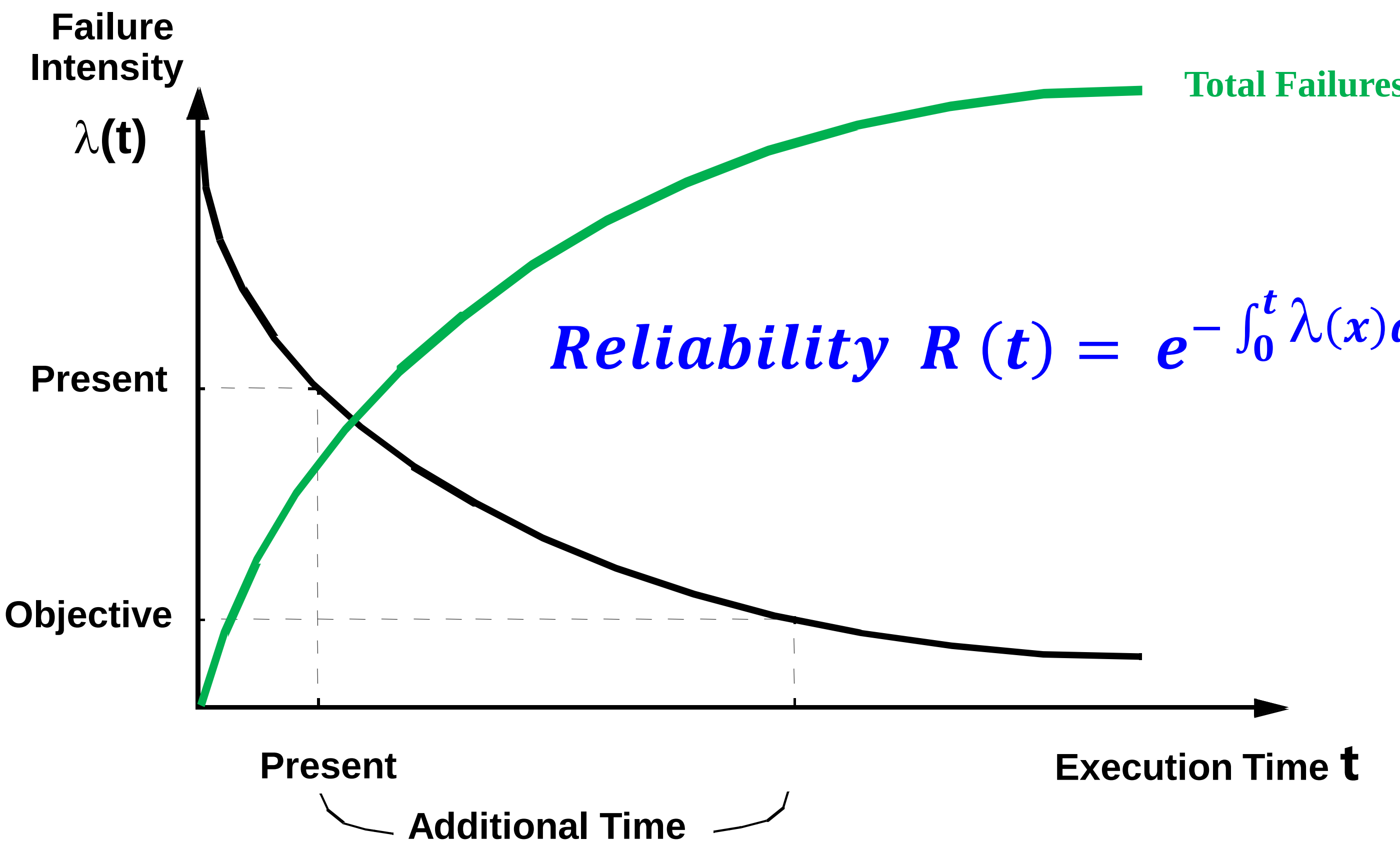
Operational Phase

Analysis Phase

Service reliability prediction
Log Management

Conclusion

Software Reliability Prediction: Small Data Modeling



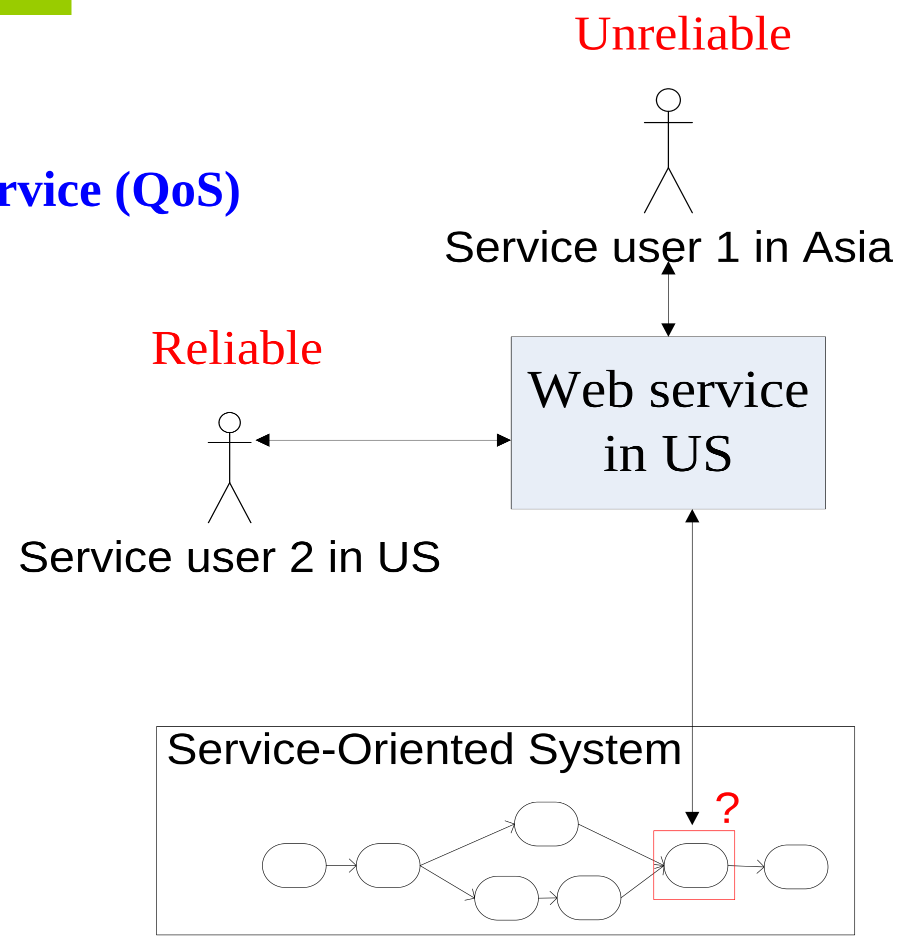
$$Reliability\ R(t) = e^{-\int_0^t \lambda(x)dx}$$

Analysis: Service Reliability Prediction

➢ Approach 1: Neighborhood-based

Reliability is extended to Quality-of-Service (QoS)

- Key idea: Using past usage experiences of similar users.
- Issue: How to calculate user similarity?



Analysis: Service Reliability Prediction

➢Similarity Computation

- User-item matrix: M×N, each entry is the failure probability of a Web service

	ws_1	ws_2	ws_3	ws_4	ws_5	ws_6
u_1	0.1	0.1		0.2	0?5	0.3
u_2		0.1		0.2	0.5	0.3
u_3	0.4		0.3		0.1	
u_4		0.6		0.4		
u_5	0.5		0.3			0.3

- Pearson Correlation Coefficient (PCC)

$$Sim(a,u) = \frac{\sum_{i \in I_a \cap I_u} (p_{a,i} - \overline{p_a})(p_{u,i} - \overline{p_u})}{\sqrt{\sum_{i \in I_a \cap I_u} (p_{a,i} - \overline{p_a})^2} \sqrt{\sum_{i \in I_a \cap I_u} (p_{u,i} - \overline{p_u})^2}}$$

Analysis: Service Reliability Prediction

➢WSRec: Hybrid Prediction Approach

- Similar users + Similar Web services

$$p_{u,i} = \lambda \times \boxed{(\overline{p_u} + \sum_{a \in S(u)} w_a \times (p_{a,i} - \overline{p_a}))} + \rightarrow \text{UPCC}$$

$$(1-\lambda) \times \boxed{(\overline{p_i} + \sum_{k \in S(i)} w_k \times (p_{u,k} - \overline{p_k}))}, \rightarrow \text{IPCC}$$

Analysis: Service Reliability Prediction

➢Performance Comparison

MAE and RMSE Comparison With Basic Approaches (A smaller MAE or RMSE value means a better performance)

Metric	Density	Methods	Training Users = 100						Training Users = 140					
			Response Time			Failure Rate			Response Time			Failure Rate		
			G10	G20	G30	G10	G20	G30	G10	G20	G30	G10	G20	G30
MAE	10%	UMEAN	1623	1539	1513	5.71%	5.58%	5.53%	1521	1439	1399	5.01%	5.00%	4.97%
		IMEAN	903	901	907	2.40%	2.36%	2.46%	861	872	855	1.62%	1.58%	1.68%
		UPCC	1148	877	810	4.85%	4.20%	3.86%	968	782	684	4.11%	3.47%	3.28%
		IPCC	768	736	736	2.24%	2.16%	2.21%	585	596	605	1.39%	1.33%	1.42%
		WSRec	**758**	**700**	**672**	**2.21%**	**2.08%**	**2.08%**	**560**	**533**	**500**	**1.36%**	**1.26%**	**1.24%**
	20%	UMEAN	1585	1548	1508	5.74%	5.53%	5.51%	1464	1410	1390	5.21%	4.98%	4.95%
		IMEAN	866	859	861	2.36%	2.34%	2.29%	833	837	840	1.56%	1.61%	1.62%
		UPCC	904	722	626	4.40%	3.43%	2.85%	794	626	540	3.93%	2.96%	2.43%
		IPCC	606	610	639	2.01%	1.98%	1.98%	479	509	538	1.17%	1.22%	1.28%
		WSRec	**586**	**551**	**546**	**1.93%**	**1.80%**	**1.70%**	**445**	**428**	**416**	**1.10%**	**1.08%**	**1.07%**
	30%	UMEAN	1603	1543	1508	5.64%	5.58%	5.56%	1494	1430	1387	5.12%	4.98%	4.93%
		IMEAN	856	854	853	2.26%	2.29%	2.30%	823	823	827	1.56%	1.58%	1.58%
		UPCC	915	671	572	4.25%	3.25%	2.58%	803	576	491	3.76%	2.86%	2.06%
		IPCC	563	566	602	1.84%	1.83%	1.86%	439	467	507	1.10%	1.12%	1.17%
		WSRec	**538**	**504**	**499**	**1.78%**	**1.69%**	**1.63%**	**405**	**385**	**378**	**1.05%**	**1.00%**	**0.98%**
RMSE	10%	UMEAN	3339	3250	3192	15.47%	15.04%	14.74%	3190	3109	3069	14.75%	14.42%	13.99%
		IMEAN	1441	1436	1442	5.61%	5.58%	5.85%	1112	1140	1107	3.27%	3.26%	3.38%
		UPCC	2036	1455	1335	10.84%	7.51%	6.55%	1585	1174	1005	8.86%	5.42%	4.96%
		IPCC	1335	1288	1278	5.36%	5.27%	5.53%	850	871	867	2.87%	2.82%	2.96%
		WSRec	**1329**	**1247**	**1197**	**5.31%**	**5.12%**	**5.11%**	**819**	**789**	**734**	**2.80%**	**2.61%**	**2.61%**
	20%	UMEAN	3332	3240	3211	15.49%	15.05%	14.80%	3190	3124	3062	14.72%	14.24%	14.07%
		IMEAN	1269	1252	1257	4.67%	4.62%	4.54%	997	1001	1002	2.53%	2.61%	2.63%
		UPCC	1356	1128	1019	8.07%	5.31%	4.58%	1028	837	730	7.35%	4.20%	3.24%
		IPCC	1020	1016	1056	4.15%	4.13%	4.12%	664	700	731	2.00%	2.09%	2.19%
		WSRec	**997**	**946**	**937**	**4.04%**	**3.83%**	**3.67%**	**620**	**598**	**581**	**1.88%**	**1.84%**	**1.83%**
	30%	UMEAN	3336	3246	3197	15.49%	15.00%	14.68%	3178	3103	3086	14.68%	14.25%	14.07%
		IMEAN	1207	1209	1203	4.21%	4.23%	4.22%	955	954	957	2.28%	2.29%	2.28%
		UPCC	1267	1035	924	7.72%	5.09%	4.15%	988	741	644	6.49%	3.90%	2.66%
		IPCC	950	957	995	3.72%	3.71%	3.75%	611	642	685	1.73%	1.74%	1.81%
		WSRec	**921**	**884**	**869**	**3.64%**	**3.46%**	**3.37%**	**564**	**540**	**528**	**1.64%**	**1.55%**	**1.52%**

Analysis: Service Reliability Prediction

➢Drawbacks of Neighborhood-based Approach

- Computational complexity $O(mn + n^2)$
- Matrix sparsity problem
  - –Not easy to find similar users (or similar items)

	i_1	i_2	i_3	i_4	i_5	i_6
u_1	0.5					0.4
u_2		0.1				
u_3					0.9	
u_4				0.7		
u_5	0.5					

Analysis: Service Reliability Prediction

➢Approach 2: Model-based Approach

- Each row of U^T is a set of feature factors, and each column of V is a set of linear predictors ⇒ Matrix Factorization (MF)

	s_1	s_2	s_3	s_4	s_5	s_6
u_1	0.98	0.23		0.22		
u_2	0.13		0.27		0.25	
u_3		0.37			0.36	
u_4	0.69		0.22	0.22		0.34

The error between the actual Value and the prediction

$$\min_{U,V} \mathcal{L}(R,U,V) = \boxed{\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n} I_{ij}^{R}(R_{ij} - U_i^T V_j)^2} + \boxed{\frac{\lambda_U}{2}\|U\|_F^2 + \frac{\lambda_V}{2}\|V\|_F^2},$$

Regularization terms

$$\underbrace{\begin{bmatrix} 0.32 & 0.15 & 0.31 & 0.33 \\ 0.23 & 0.15 & 0.26 & 0.28 \\ 0.30 & 0.20 & 0.24 & 0.34 \\ 0.47 & 0.23 & 0.59 & 0.21 \end{bmatrix}}_{U^T} \times \underbrace{\begin{bmatrix} 0.73 & 0.35 & 0.31 & 0.26 & 0.32 & 0.42 \\ 0.60 & 0.31 & 0.27 & 0.22 & 0.28 & 0.36 \\ 0.69 & 0.37 & 0.32 & 0.27 & 0.33 & 0.45 \\ 0.95 & 0.46 & 0.42 & 0.35 & 0.41 & 0.54 \end{bmatrix}}_{V}$$

Analysis: Service Reliability Prediction

➢NIMF: Neighborhood–Integrated Matrix Factorization

	i_1	i_2	i_3	i_4	i_5	i_6
u_1	0.5	1.2		0.3		0.4
u_2		0.8		0.6	0.5	
u_3	0.4		0.3		0.9	
u_4		0.6		0.7		
u_5	0.5		0.7			0.3

(a) User-Item Matrix

User's own rating

Rating due to similar users

$$\mathcal{L}(R,S,U,V) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n} I_{ij}^{R}\Big(R_{ij} - \big(\alpha \boxed{U_i^T V_j} + (1-\alpha)\boxed{\sum_{k\in\mathcal{T}(i)} S_{ik} U_k^T V_j}\big)\Big)^2 + \frac{\lambda_U}{2}\|U\|_F^2 + \frac{\lambda_V}{2}\|V\|_F^2,$$

$$S_{ik} = \frac{PCC(i,k)}{\sum_{k\in\mathcal{T}(i)} PCC(i,k)}$$

Analysis: Service Reliability Prediction

➢Performance Comparison

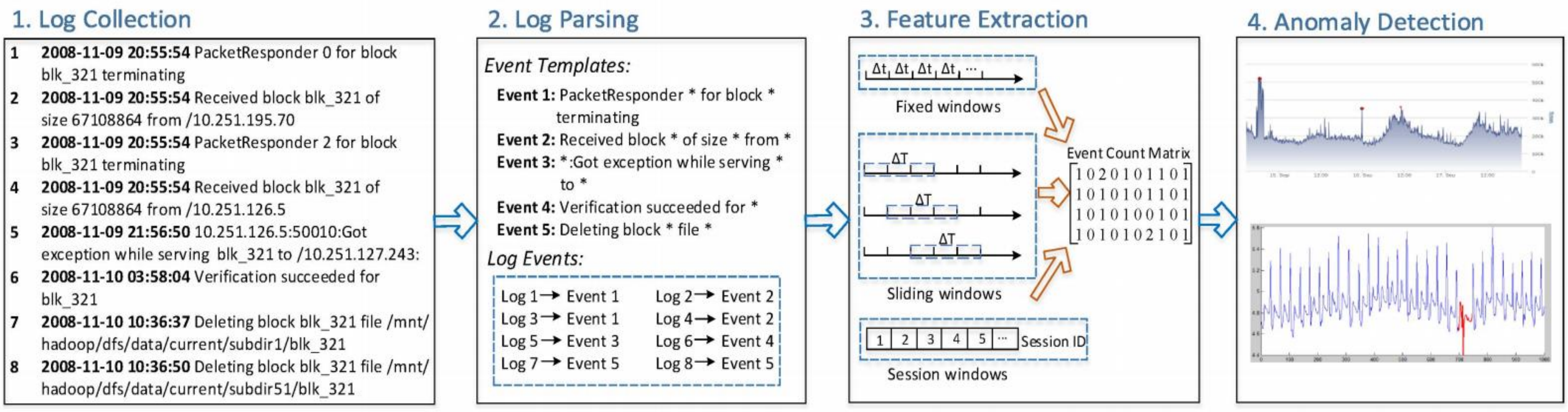
Table 2: Performance Comparison (A Smaller MAE or RMSE Value Means a Better Performance)

QoS	Methods	Matrix Density=5%		Matrix Density=10%		Matrix Density=15%		Matrix Density=20%	
		MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
Response-time (0-20 s)	UMEAN	0.8785	1.8591	0.8783	1.8555	0.8768	1.8548	0.8747	1.8557
	IMEAN	0.7015	1.5813	0.6918	1.5440	0.6867	1.5342	0.6818	1.5311
	UPCC	0.6261	1.4078	0.5517	1.3151	0.5159	1.2680	0.4884	1.2334
	IPCC	0.6897	1.4296	0.5917	1.3268	0.5037	1.2552	0.4459	1.2095
	WSRec	0.6234	1.4078	0.5365	1.3043	0.4965	1.2467	0.4407	1.2012
	NMF	0.6182	1.5746	0.6040	1.5494	0.5990	1.5345	0.5982	1.5331
	PMF	0.5678	1.4735	0.4996	1.2866	0.4720	1.2163	0.4492	1.1828
	NIMF	**0.5514**	**1.4075**	**0.4854**	**1.2745**	**0.4534**	**1.1980**	**0.4357**	**1.1678**
Throughput (0-1000 kbps)	UMEAN	54.0084	110.2821	53.6700	110.2977	53.8792	110.1751	53.7114	110.1708
	IMEAN	27.3558	66.6344	26.8318	64.7674	26.6239	64.3986	26.6364	64.1082
	UPCC	26.1230	61.6108	21.2695	54.3701	18.7455	50.7768	17.5546	48.2621
	IPCC	29.2651	64.2285	27.3993	60.0825	26.4319	57.8593	25.0273	55.4970
	WSRec	25.8755	60.8685	19.9754	54.8761	17.5543	47.8235	16.0762	47.8749
	NMF	25.7529	65.8517	17.8411	53.9896	15.8939	51.7322	15.2516	48.6330
	PMF	19.9034	54.0508	16.1755	46.4439	15.0956	43.7957	14.6694	42.4855
	NIMF	**17.9297**	**51.6573**	**16.0542**	**45.9409**	**14.4363**	**43.1596**	**13.7099**	**41.1689**

Reliability Prediction of Web Services

- Approach 1: Neighborhood-based approach – to consider users
- Approach 2: Model-based approach – to consider data sparsity
- Approach 3: Time-aware approach – to consider temporal factor
- Approach 4: Network coordinate based approach – to consider spatial factor
- Approach 5: Ranking-based approach – to consider ranking

Analysis: Log Management



Log Analysis Framework

Analysis: Log Management

Raw Log Messages

1	**2008-11-11 03:40:58** BLOCK* NameSystem.allocateBlock: /user/root/randtxt4/_temporary/_task_200811101024_0010_m_000011_0/part-00011.blk_904791815409399662
2	**2008-11-11 03:40:59** Receiving block blk_904791815409399662 src: /10.251.43.210:55700 dest: /10.251.43.210:50010
3	**2008-11-11 03:41:01** Receiving block blk_904791815409399662 src: /10.250.18.114:52231 dest: /10.250.18.114:50010
4	**2008-11-11 03:41:48** PacketResponder 0 for block blk_904791815409399662 terminating
5	**2008-11-11 03:41:48** Received block blk_904791815409399662 of size 67108864 from /10.250.18.114
6	**2008-11-11 03:41:48** PacketResponder 1 for block blk_904791815409399662 terminating
7	**2008-11-11 03:41:48** Received block blk_904791815409399662 of size 67108864 from /10.251.43.210
8	**2008-11-11 03:41:48** BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.43.210:50010 is added to blk_904791815409399662 size 67108864
9	**2008-11-11 03:41:48** BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.250.18.114:50010 is added to blk_904791815409399662 size 67108864
10	**2008-11-11 08:30:54** Verification succeeded for blk_904791815409399662

Log Parsing

Log Events

Event1	BLOCK* NameSystem.allocateBlock: *
Event2	Receiving block * src: * dest: *
Event3	PacketResponder * for block * terminating
Event4	Received block * of size * from *
Event5	BLOCK* NameSystem.addStoredBlock: blockMap updated: * is added to * size *
Event6	Verification succeeded for *

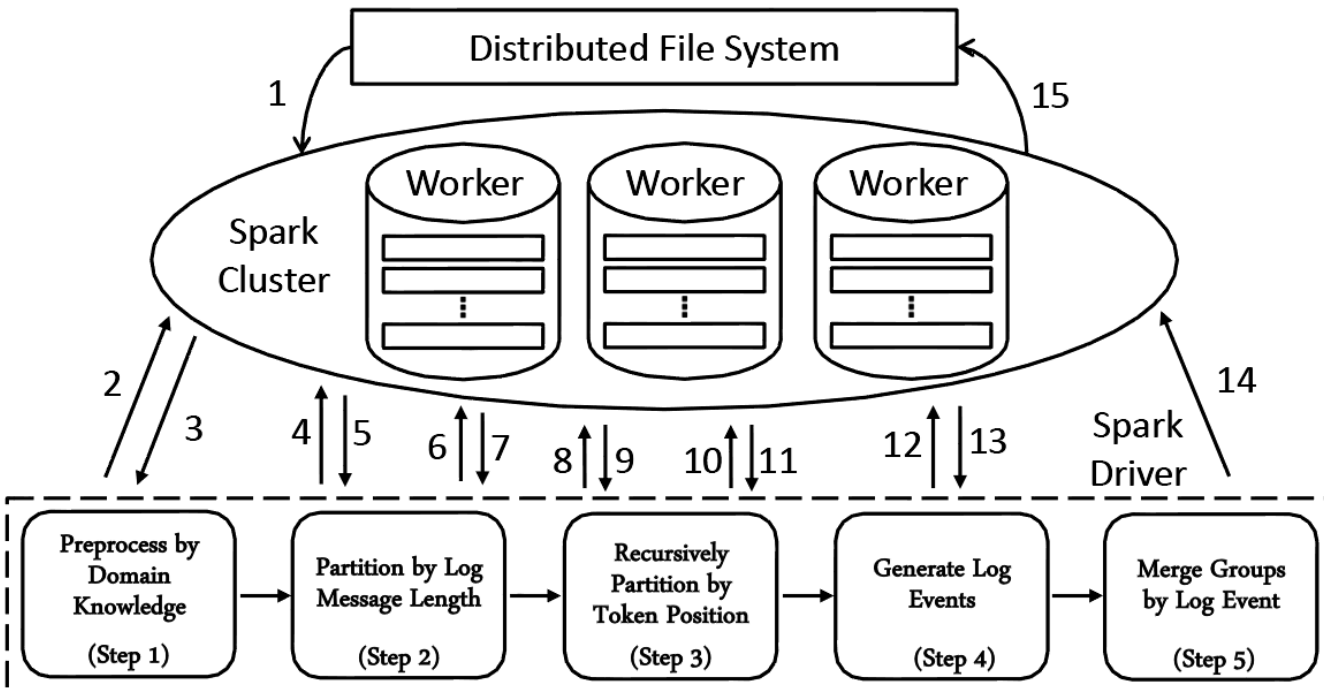
Structured Logs

1	2008-11-11 03:40:58 Event1
2	2008-11-11 03:40:59 Event2
3	2008-11-11 03:41:01 Event2
4	2008-11-11 03:41:48 Event3
5	2008-11-11 03:41:48 Event4
6	2008-11-11 03:41:48 Event3
7	2008-11-11 03:41:48 Event4
8	2008-11-11 03:41:48 Event5
9	2008-11-11 03:41:48 Event5
10	2008-11-11 08:30:54 Event6

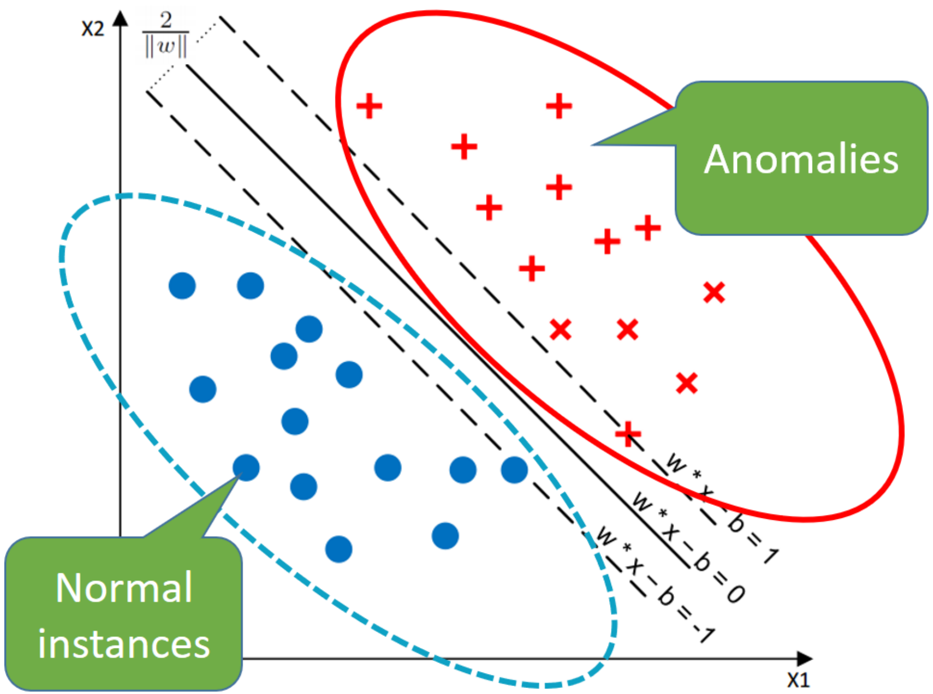
Log Parsing

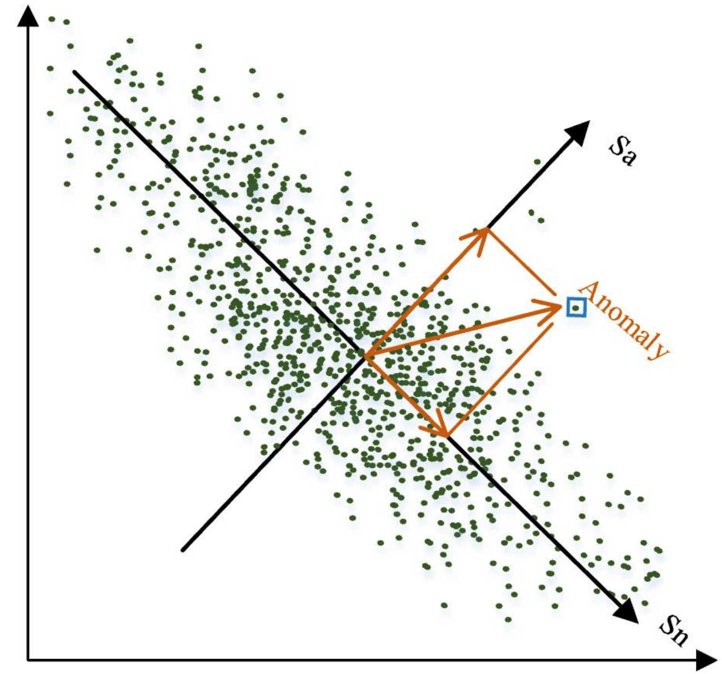
Analysis: Log Management

- We design and implement a **parallel** log parser (namely POP) on top of Spark.
- It can process **200 million** lines of raw log messages within **7** min while keeping high accuracy.



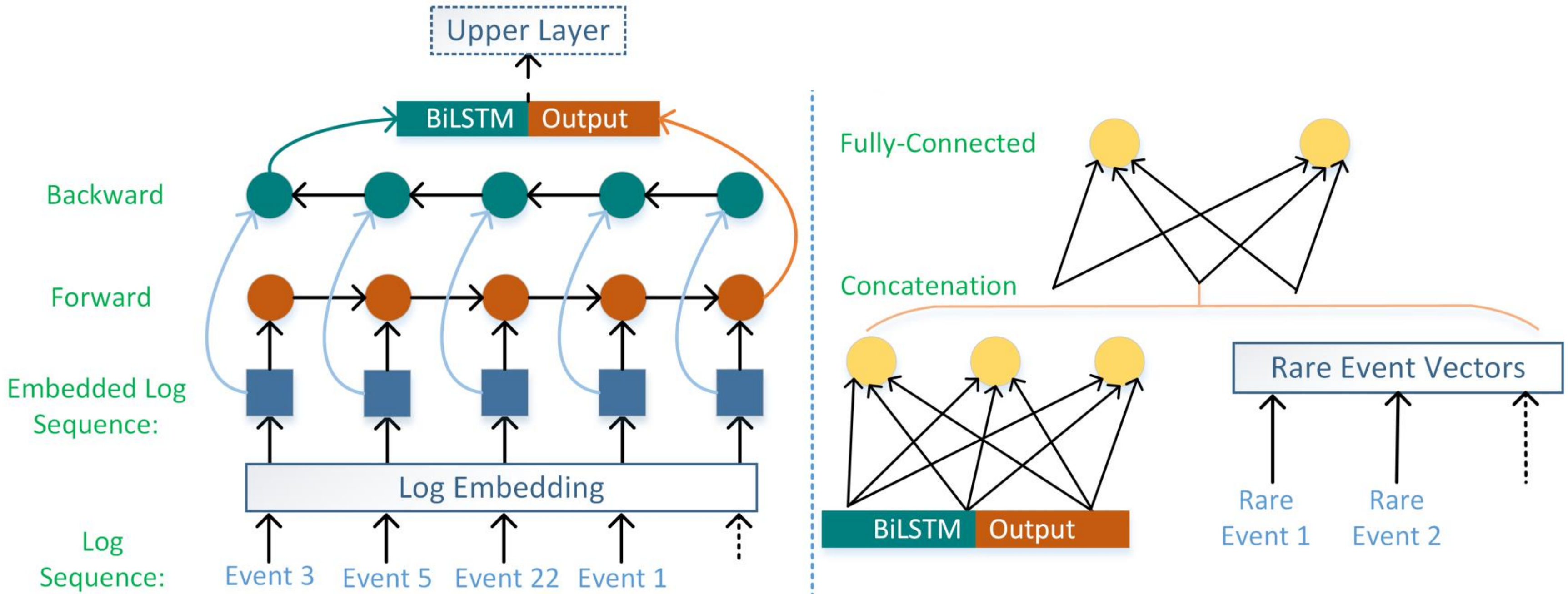
Analysis: Log Management





Existing anomaly detection methods: SVM (left) and PCA (right)

Analysis: Log Management



Our method: Deep Log Embedding based Anomaly Detection (D-Lead)

Analysis: Log Management

LogPAI

(Log Powered by AI)

https://github.com/logpai

loghub

A collection of system log datasets for massive log analysis

log-analysis · logs · console-log · log-parsing · unstructured-logs

★ 16 · 3 forks · Updated 23 days ago

LogAdvisor

Learning to Log: A framework for determining optimal logging points

machine-learning · logging · code-analysis

C# · ★ 1 · 2 forks · Updated on May 1

logparser

logparser: A toolkit for automated log parsing

log-analysis · log · log-parser · log-mining · log-parsing

Python · ★ 25 · 14 forks · MIT · Updated on Jul 27

loglizer

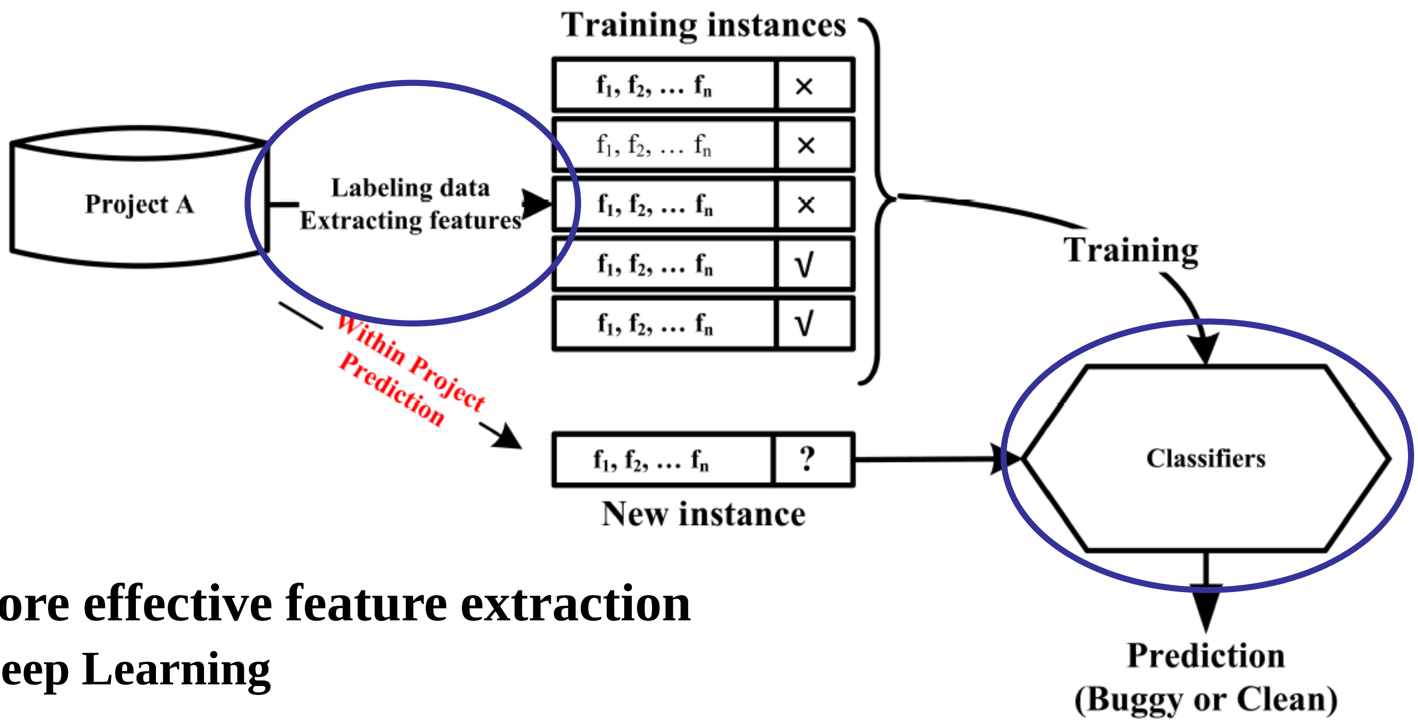
loglizer: A log analysis toolkit for automated anomaly detection

log-analysis · log-management · anomaly-detection · unstructured-logs

Python · ★ 33 · 17 forks · MIT · Updated on Sep 21

Defect Prediction

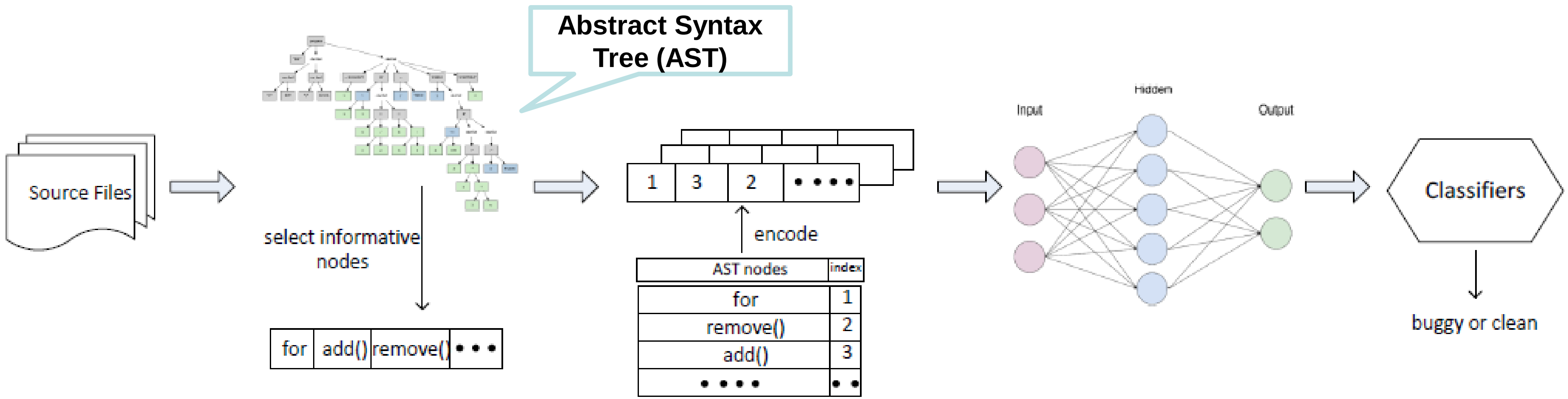
- **Software defect prediction: build classifiers to predict code areas that potentially contain defects, based on code features.**



❑ **More effective feature extraction**
--Deep Learning

Defect Prediction

❑ The overall workflow of proposed DP-CNN



1. Parsing source code and extracting token vectors
2. Encoding token vectors
3. Generating features via CNN
4. Predicting defects

Defect Prediction

❑ Performance on 8 open source projects

Project	Traditional	DBN	DBN+	CNN	DP-CNN
camel	0.329	0.335	0.375	0.505	0.508
jEdit	0.573	0.480	0.549	0.631	0.580
lucene	0.618	0.758	0.761	0.761	0.761
xalan	0.627	0.681	0.681	0.676	0.696
xerces	0.273	0.261	0.276	0.311	0.374
synapse	0.500	0.503	0.486	0.512	0.556
poi	0.748	0.780	0.782	0.778	0.784
eclipse	0.273	0.290	0.349	0.337	0.367
Average	0.493	0.511	0.532	0.564	0.578

- Introduction
- AI Techniques
- Development Phase
- Operational Phase
- Analysis Phase
- **Conclusion**

Conclusion

- Before AI becomes conscientious, its intelligence is still artificial.
- Software is eating the world, and AI is eating the software.

---Nvidia CEO Jensen Huang

- AI may replace many people's job, but it will certainly enhance software engineers to do a better job.
- Our goal is to employ AI to provide more efficient and effective software development, operation, and analysis.
- The current achievement is just a small step ahead in a largely unexplored area in existing software engineering research paradigms.

Thank You!