Reliable and Interpretable Artificial Intelligence

Martin Vechev
ETH Zurich

Fall 2017

Machine Learning (ML) + Synthesis

So far we looked at "program synthesis in the small", meaning we did not leverage knowledge of prior, similar programs, when synthesizing a new program. In this lecture, we are going to study both directions:

Part I: Leverage ML models to enable program synthesis. This will involve learning from a new kind of data sets (e.g., now programs themselves are examples). Motivated by this we will move to **Part II**

Part II: Leverage synthesis to build more interpretable ML models. An alternative to attention mechanism in neural networks. Applies the iterative synthesis (oracle-based) loop concepts from previous two lectures to learn ML models. The techniques here apply beyond learning from code but natural language and other data.

Probabilistic Learning from Code

Task → **Probabilistic Tool** → **Solution**

ML model

15 million repositories

Billions of lines of code

High quality, tested, maintained programs

Why now?

Advances in Programming Languages
[Automated Reasoning, Synthesis, Constraint Solving]

Confluence of streams

Advances in Machine Learning
[Deep Learning, Graphical Models, Language Models]

machine learning-based programming tools
new rules, new ideas, new opportunities

Data
[> 15 million public repositories]

Machine Learning for Programming

Applications	Code synthesis; Deobfuscation; Program repair; Feedback generation; Translation
Intermediate Representation	Sequences (sentences); Trees; Translation Table; Graphical Models (CRFs); Feature Vectors
Semantic Analysis	typestate analysis; scope analysis; control-flow analysis; alias analysis
Train Model (ML)	Neural Networks; N-gram language model; SVM; Structured SVM
Query Model (ML)	argmax P(y | x), y ∈ Ω; Greedy MAP inference

More information: http://plml.ethz.ch

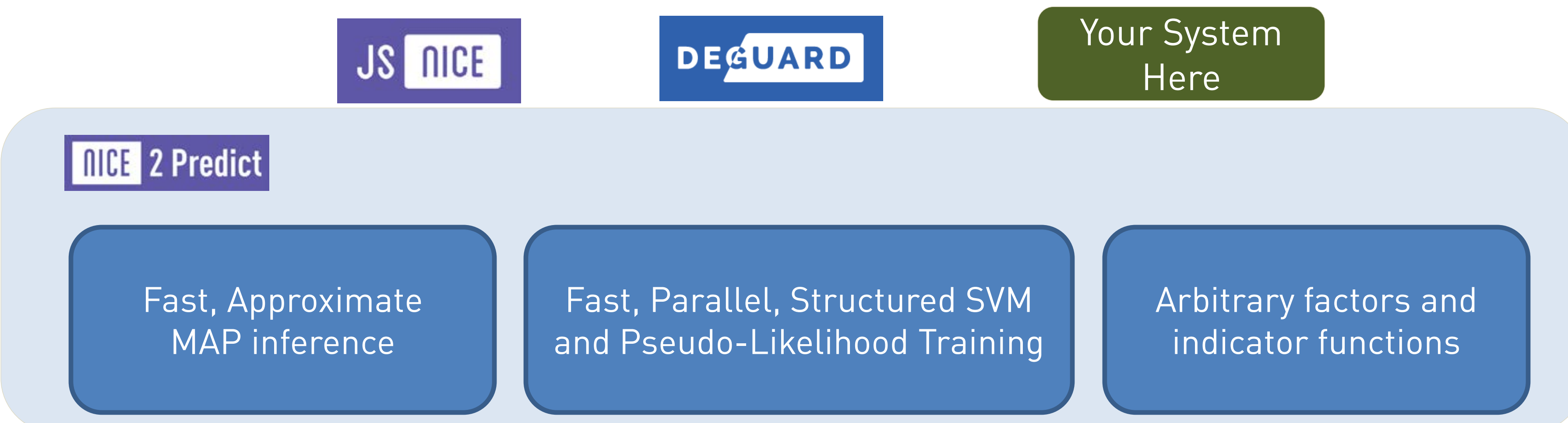
ML Systems we have built, such as jsnice.org and apk-deguard.com have > 300,000 users...

Nice2Predict.org:

scalable structured prediction framework

fully, open sourced, Apache license

used by various groups worldwide



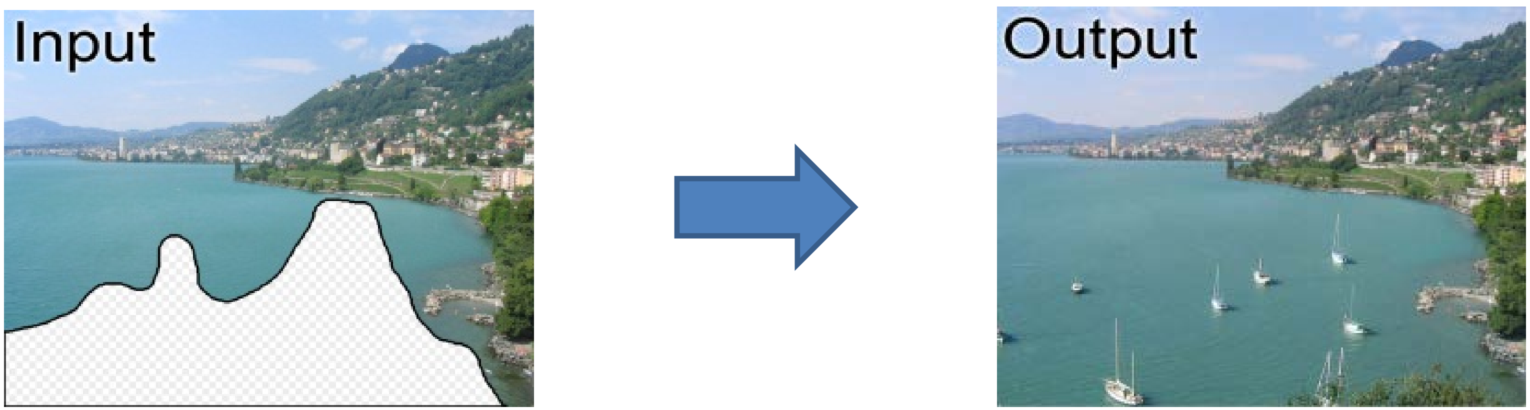
Machine Learning for Programming

Applications	Code synthesis; Deobfuscation; Program repair; Feedback generation; Translation
Intermediate Representation	Sequences (sentences); Trees; Translation Table; Graphical Models (CRFs); Feature Vectors
Semantic Analysis	typestate analysis; scope analysis; control-flow analysis; alias analysis
Train Model (ML)	Neural Networks; N-gram language model; SVM; Structured SVM
Query Model (ML)	$\operatorname{argmax}_{y \in \Omega} P(y \mid x)$; Greedy MAP inference

More information: http://plml.ethz.ch

ML Systems we have built, such as jsnice.org and apk-deguard.com have > 300,000 users...

Intuition: Scene Completion



“Scene Completion Using Millions of Photographs”, J. Hays, A. Efros, ACM SIGGRAPH 2007

Statistical Code Synthesis

[Code Completion with Statistical Language Models, ACM PLDI 2014]

```
Camera camera = Camera.open();
camera.setDisplayOrientation(90);
```

?

```
MediaRecorder rec = new MediaRecorder();
```

?

```
rec.setAudioSource(MediaRecorder.AudioSource.MIC);
rec.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
rec.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
```

?

```
rec.setOutputFile("file.mp4");
...
```

Statistical Code Synthesis

[Code Completion with Statistical Language Models, ACM PLDI 2014]

```
Camera camera = Camera.open();
camera.setDisplayOrientation(90);

camera.unlock();

MediaRecorder rec = new MediaRecorder();

rec.setCamera(camera);

rec.setAudioSource(MediaRecorder.AudioSource.MIC);
rec.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
rec.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);

rec.setAudioEncoder(1);
rec.setVideoEncoder(3);

rec.setOutputFile("file.mp4");
...
```

Infers sequences not in training data

Handles multiple objects

Infers multiple statements

Key Insight

Regularities in code are similar to regularities in natural language

Key Insight

Regularities in code are similar to regularities in natural language

We want to learn that

```
MediaRecorder rec = new MediaRecorder();
```

is before

```
rec.setCamera(camera);
```

Key Insight

Regularities in code are similar to regularities in natural language

We want to learn that

```
MediaRecorder rec = new MediaRecorder();
```

is before

```
rec.setCamera(camera);
```

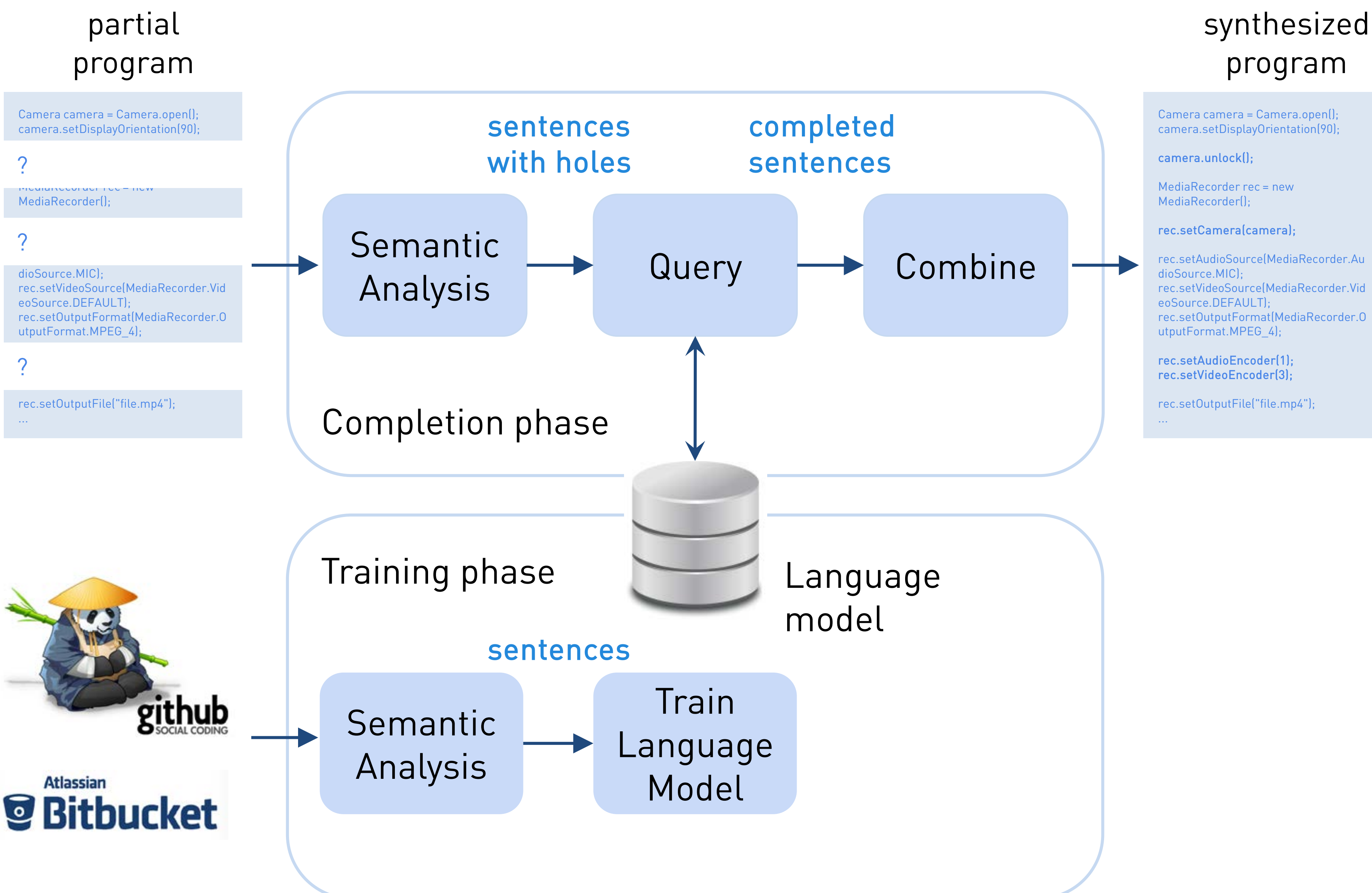
like in natural languages

Hello

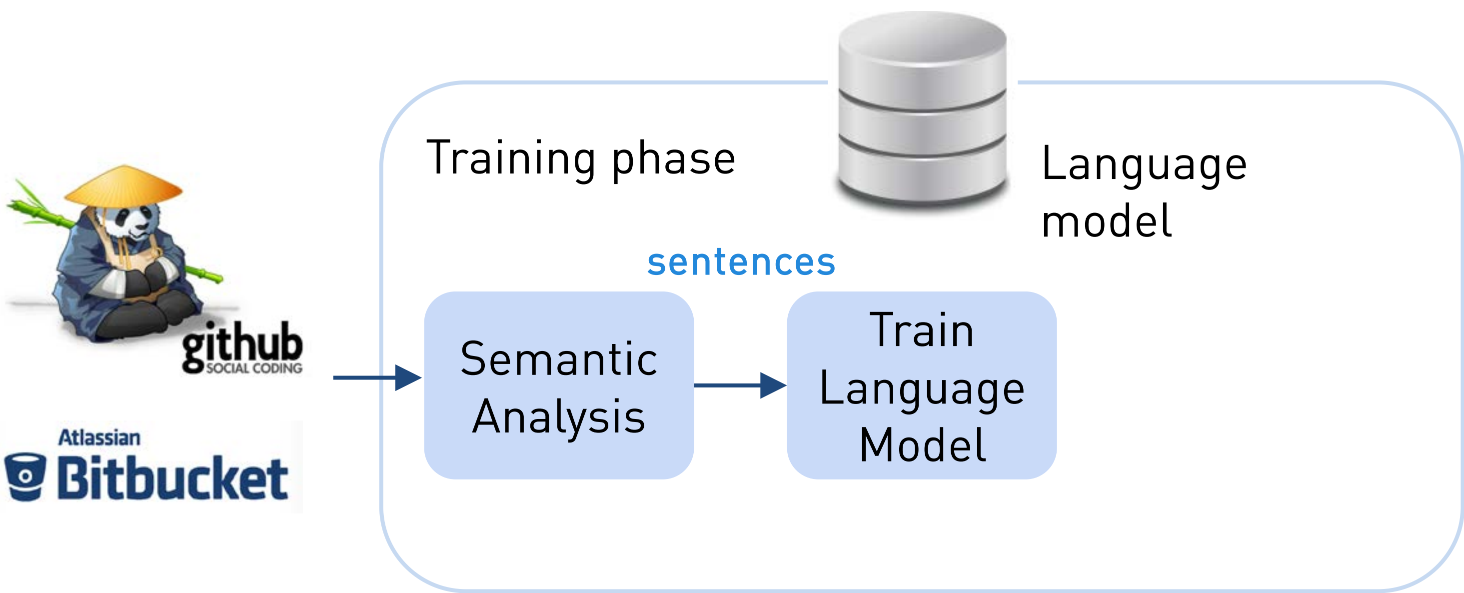
is before

World !

The SLANG System



The SLANG System



Step1: From Programs to Sentences

```
she = new X();
me = new Y();

me.sleep();
if (random()) {
  me.eat();
}
she.enter();
me.talk(she);
```

A note on:

learning from code vs.
learning from natural language (NLP):

Unlike natural language, code has formal semantics that one can benefit from if they have a way to automatically extract these…

This means that with code the actual data can be:

<u>Choice 1</u> (*Syntactic*): a sequence of tokens (words) that make up the program. This follows the NLP mindset.

<u>Choice 2</u> (*Semantic*): a sequence of words where each word captures something about the semantics of the program

Lets see how Choice 2 works…

From Programs to Sentences

```
she = new X();
me = new Y();

me.sleep();
if (random()) {
  me.eat();
}
she.enter();
me.talk(she);
```

We get 3 sequences:

for abstract object **me**:

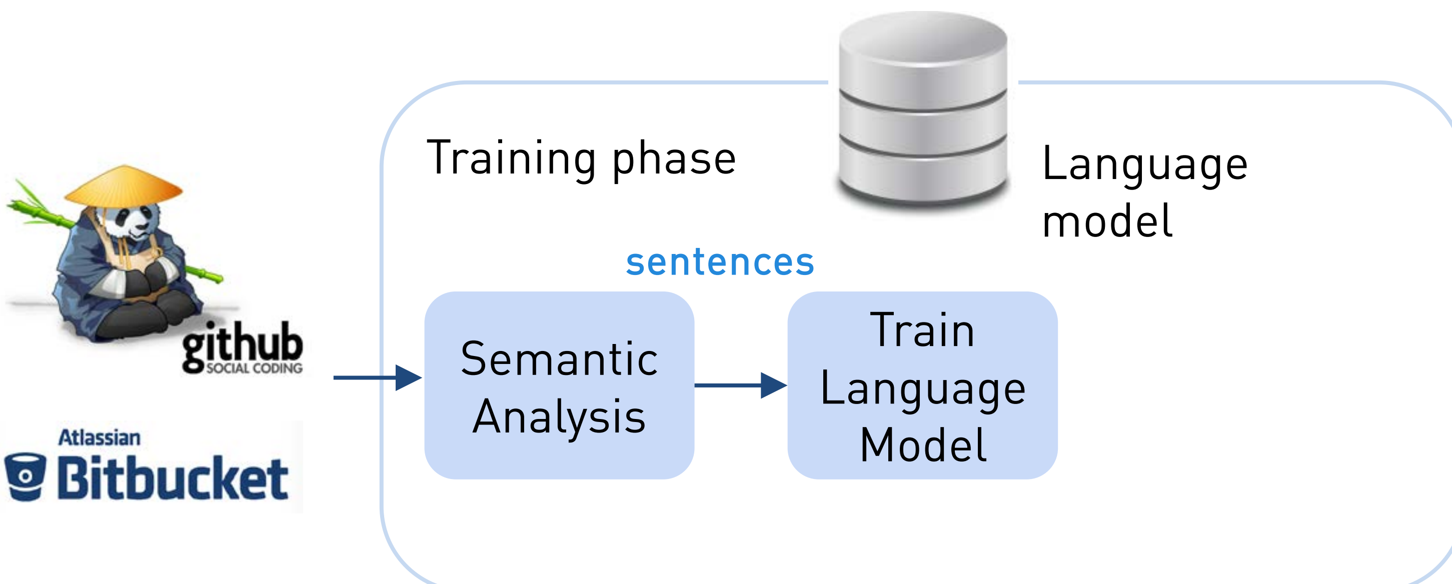
Y_{init} sleep talk
Y_{init} sleep eat talk

for abstract object **she**:

X_{init} enter $talk_{param1}$

abstract (semantic) object:
think of it as the variable denoting all concrete objects assigned to that variable. So many objects assigned to the **she** variable will all be mapped to the same **she** abstract object. In our example we only have 1 concrete object but if we had a loop around **she**=new X(), we would have more concrete objects but still only 1 abstract object.

Step 2: Learn Regularities in Sentences



Learn Regularities

Learn regularities in obtained sentences

Regularities in sentences ⇔ regularities in code usage

If we see many sequences like:

X_{init} enter $talk_{param1}$

then we should learn that $talk_{param1}$ is often after enter

Statistical Language Models

Given a sentence $s = w_1\ w_2\ w_3 \ldots\ w_n$

estimate $\mathbf{P}(\ w_1\ w_2\ w_3 \ldots\ w_n\ )$

Decomposed to conditional probabilities

$$\mathbf{P}(\ w_1\ w_2\ w_3 \ldots\ w_n\ ) = \prod_{i=1..n} \mathbf{P}(\ w_i \mid w_1\ \ldots\ w_{i-1}\ )$$

Statistical Language Models

Given a sentence $s = w_1\, w_2\, w_3 \ldots w_n$

estimate $\mathbf{P}(\, w_1\, w_2\, w_3 \ldots w_n\,)$

Decomposed to conditional probabilities (via chain rule):

$$\mathbf{P}(\, w_1\, w_2\, w_3 \ldots w_n\,) = \prod_{i=1..n} \mathbf{P}(\, w_i \mid w_1 \ldots w_{i-1}\,)$$

P(The quick brown fox jumped) =

P(The) **P**(quick | The) **P**(brown | The quick) **P**(fox | The quick brown) **P**(jumped | The quick brown fox)

Choice I: N-gram language model

Conditional probability based only on previous n-1 words

$$\mathbf{P}(\, w_i \mid w_1 \;\ldots\; w_{i-1}\,) \approx \mathbf{P}(\, w_i \mid w_{i-n+1} \;\ldots\; w_{i-1}\,)$$

N-gram language model

Conditional probability based only on previous n-1 words

$$\mathbf{P}(\, w_i \mid w_1 \; \ldots \; w_{i-1} \,) \approx \mathbf{P}(\, w_i \mid \underbrace{w_{i-n+1} \; \ldots \; w_{i-1}}_{\text{n-1 words}} \,)$$

N-gram language model

Conditional probability based only on previous n-1 words

$$P(w_i \mid w_1 \ldots w_{i-1}) \approx P(w_i \mid \underbrace{w_{i-n+1} \ldots w_{i-1}}_{\text{n-1 words}})$$

Training is achieved by counting n-grams, called MLE = maximum likelihood estimation. E.g., with 3-gram language model, we get:

$$P(\text{jumped} \mid \text{The quick brown fox}) \approx P(\text{jumped} \mid \text{brown fox}) \approx \frac{\#(\text{brown fox jumped})}{\#(\text{brown fox})}$$

#(n-gram) - number of occurrences of n-gram in training data

Time complexity for each word encountered in training is constant, so training is usually fast.

* technicality: to actually get a probability distribution across all sentences we typically add N-1 start symbols and N-1 end symbols </s> to each sentence.

3-gram language model: example

$$P(w_1 \cdot w_2 \cdot w_3 \cdot \ldots w_n) \sim= P(w_1) * P(w_2 \mid w_1) * P(w_3 \mid w_1 \cdot w_2) * \ldots * P(w_n \mid w_{n-2} \cdot w_{n-1})$$

3-grams	# of occurrences
brown fox jumped	125
brown fox walked	45
brown fox snapped	30

P (jumped | brown fox) ~= 125/200 ~= 0.625

P (brown fox jumped) ~=

P (brown)	*	P (fox | brown)	*	P (jumped | brown fox)	
200 / 600	*	200 / 200	*	125 / 200	~= 0.208

Key Problem: Sparsity of Data

What if this number is 0?

$$P(\text{ jumped } | \text{ brown fox }) \approx \frac{\#(\text{ brown fox jumped })}{\#(\text{ brown fox })}$$

The problem of sparsity gets worse as the size of the n-gram becomes larger.

We need to handle n-grams with 0 or few occurrences in the training data.
Techniques that can do that are: **smoothing**, **discounting**

Solution: Smoothing Techniques

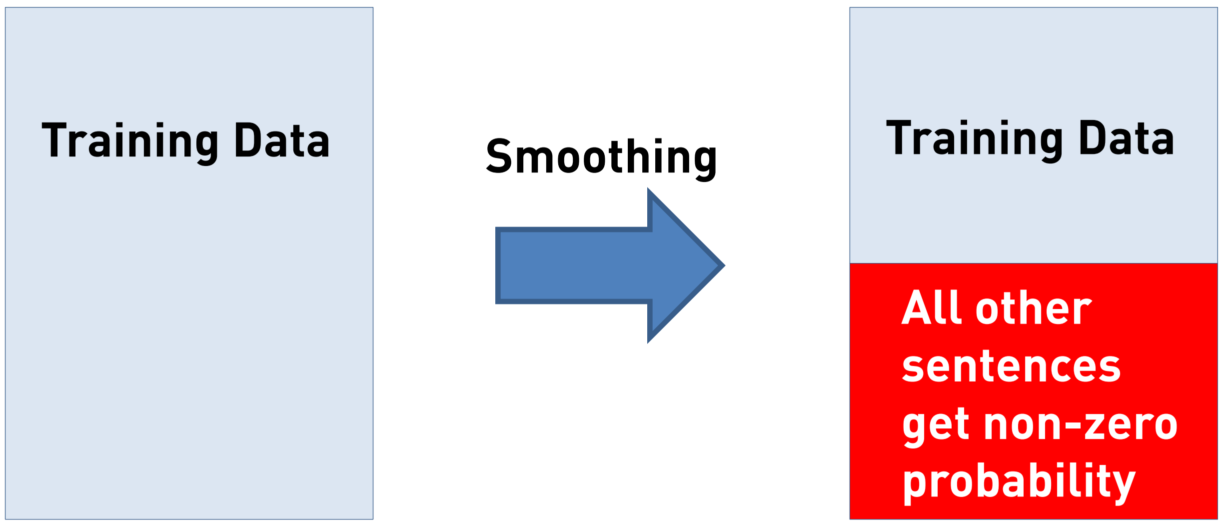
- Smoothing techniques give non-zero probability to n-grams not in the training data.
- Essentially, they try to estimate how likely it is that the n-gram is missing due to the limited size of the training data.
- They work by taking the probability mass of the existing n-grams and redistributing that mass over n-grams that occur zero times.
- Typically, probability of such n-grams is estimated by looking at the probabilities of n-1 grams , n-2 grams, unigrams, etc.

Example smoothing techniques are: Witten-Bell Interpolated (WBI), Witten-Bell Backoff (WBB), Natural Discounting (ND), Stupid-Backoff (SB)`

Smoothing: Intuitively

Distribution of probability mass before smoothing

Distribution of probability mass after smoothing



All other sentences get 0 probability

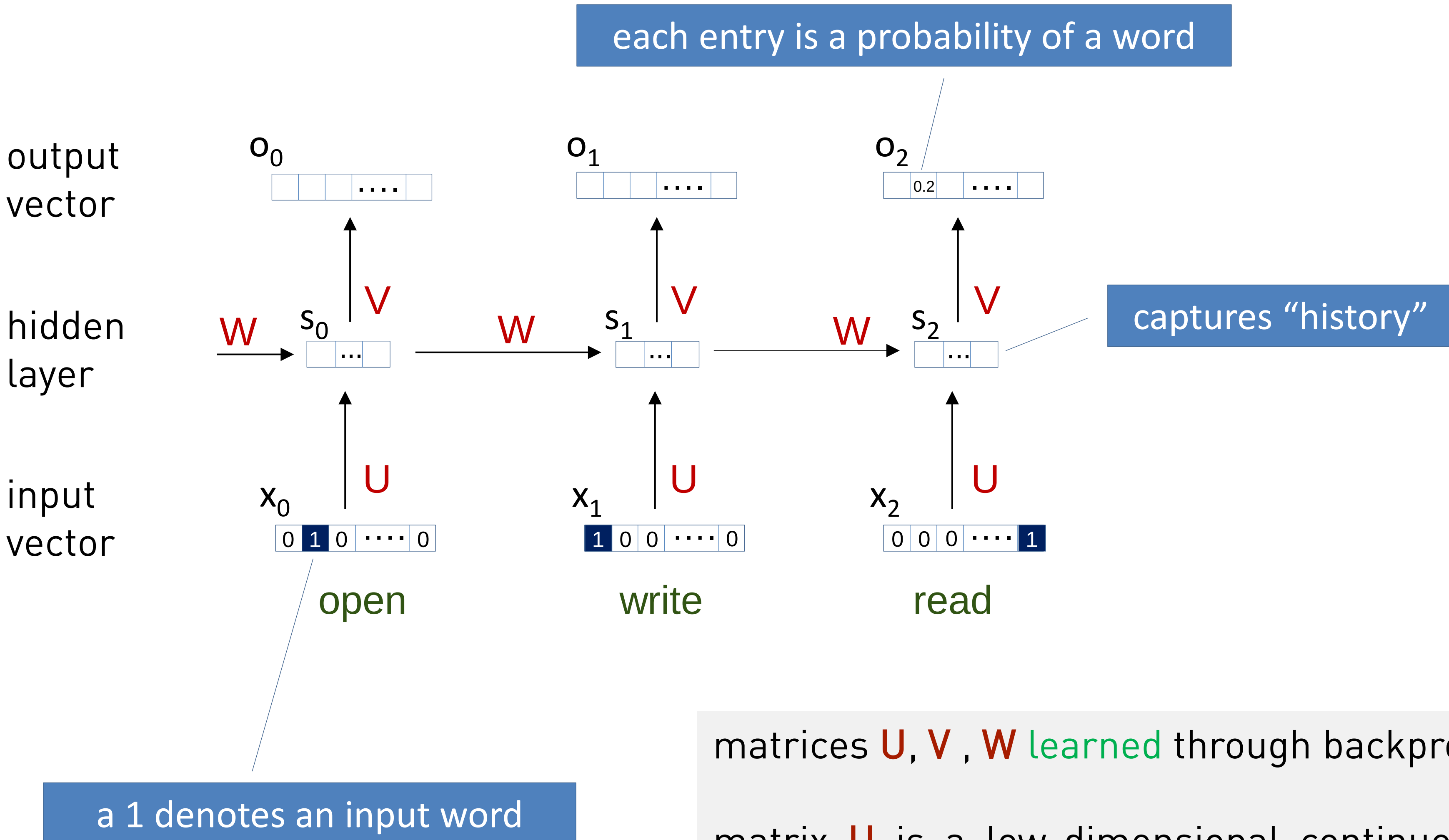
N-gram language model

Conditional probability only on previous n-1 words

$$\mathbf{P}(\,w_i \mid w_1 \;\ldots\; w_{i-1}\,) \approx \mathbf{P}(\,w_i \mid \underbrace{w_{i-n+1} \;\ldots\; w_{i-1}}_{\text{n-1 words}}\,)$$

Existing library implementing language models is SRILM:
http://www.speech.sri.com/projects/srilm/

Choice II: Recurrent Neural Networks (RNN)

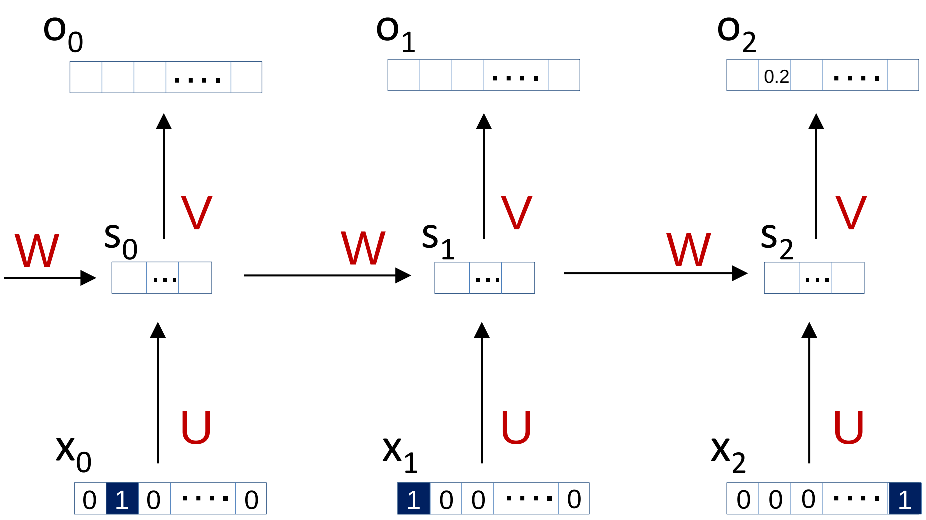


matrices U, V , W learned through backprop

matrix U is a low-dimensional continuous vector representation of input words, also referred to as "word embedding"

RNNs: more details on board

Choice II: Recurrent Neural Networks (RNN)



+ can learn dependencies beyond the prior several words (e.g., LSTM)
+ learns continuous representation of vocabulary (helps avoid sparsity)
- slow and tricky to train
- predictions hard to explain

RNNLM-40 Download: http://rnnlm.org/

TensorFlow https://www.tensorflow.org/

ML-based Code Synthesis

```
smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
  list = smsMgr.divideMessage(message);
  ? {smsMgr, list} // (Hole H1)
} else {
  ? {smsMgr, message} // (Hole H2)
}
```

ML-based Code Synthesis

```
smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
  list = smsMgr.divideMessage(message);
  ? {smsMgr, list} // (Hole H1)
} else {
  ? {smsMgr, message} // (Hole H2)
}
```

Abstract object **smsMgr**:

$\text{getDefault}_{\text{result}}$ divideMessage H1

$\text{getDefault}_{\text{result}}$ H2

Abstract object **list**:

$\text{divideMessage}_{\text{result}}$ H1

Abstract object **message**:

length $\text{divideMessage}_{\text{param1}}$

length H2

ML-based Code Synthesis

$\text{getDefault}_{\text{result}}$ divideMessage **H1**	

$\text{getDefault}_{\text{result}}$ **H2**	

$\text{divideMessage}_{\text{result}}$ **H1**	
length **H2**	

ML-based Code Synthesis

	Completion probability
getDefault$_{result}$ divideMessage **sendMultipartTextMessage**	0.0033
getDefault$_{result}$ divideMessage **sendTextMessage**	0.0016

getDefault$_{result}$ **sendTextMessage**	0.0073
getDefault$_{result}$ **sendMultipartTextMessage**	0.0010

divideMessage$_{result}$ **sendMultipartTextMessage$_{param3}$**	0.0821

length **length**	0.0132
length **split**	0.0080
length **sendTextMessage$_{param3}$**	0.0017

ML-based Code Synthesis

	Completion probability
getDefault$_{result}$ divideMessage **sendMultipartTextMessage**	0.0033
getDefault$_{result}$ divideMessage **sendTextMessage**	0.0016
getDefault$_{result}$ **sendTextMessage**	0.0073
getDefault$_{result}$ **sendMultipartTextMessage**	0.0010
divideMessage$_{result}$ **sendMultipartTextMessage$_{param3}$**	0.0821
length **length**	0.0132
length **split**	0.0080
length **sendTextMessage$_{param3}$**	0.0017

ML-based Code Synthesis

	Completion probability
getDefault$_{result}$ divideMessage **sendMultipartTextMessage**	0.0033
getDefault$_{result}$ divideMessage **sendTextMessage**	0.0016
getDefault$_{result}$ **sendTextMessage**	0.0073
getDefault$_{result}$ **sendMultipartTextMess**	
divideMessage$_{result}$ **sendMultipartTextMes**	
length **length**	
length **split**	0.0080
length **sendTextMessage**$_{param3}$	0.0017

Not a feasible solution: completions disagree on selected method

The solution must satisfy program constraints

ML-based Code Synthesis

	Completion probability
getDefault$_{result}$ divideMessage **sendMultipartTextMessage**	0.0033
getDefault$_{result}$ divideMessage **sendTextMessage**	0.0016
getDefault$_{result}$ **sendTextMessage**	0.0073
getDefault$_{result}$ **sendMultipartTextMessage**	0.0010
divideMessage$_{result}$ **sendMultipartTextMessage$_{param3}$**	0.0821
length **length**	0.0132
length **split**	0.0080
length **sendTextMessage$_{param3}$**	0.0017

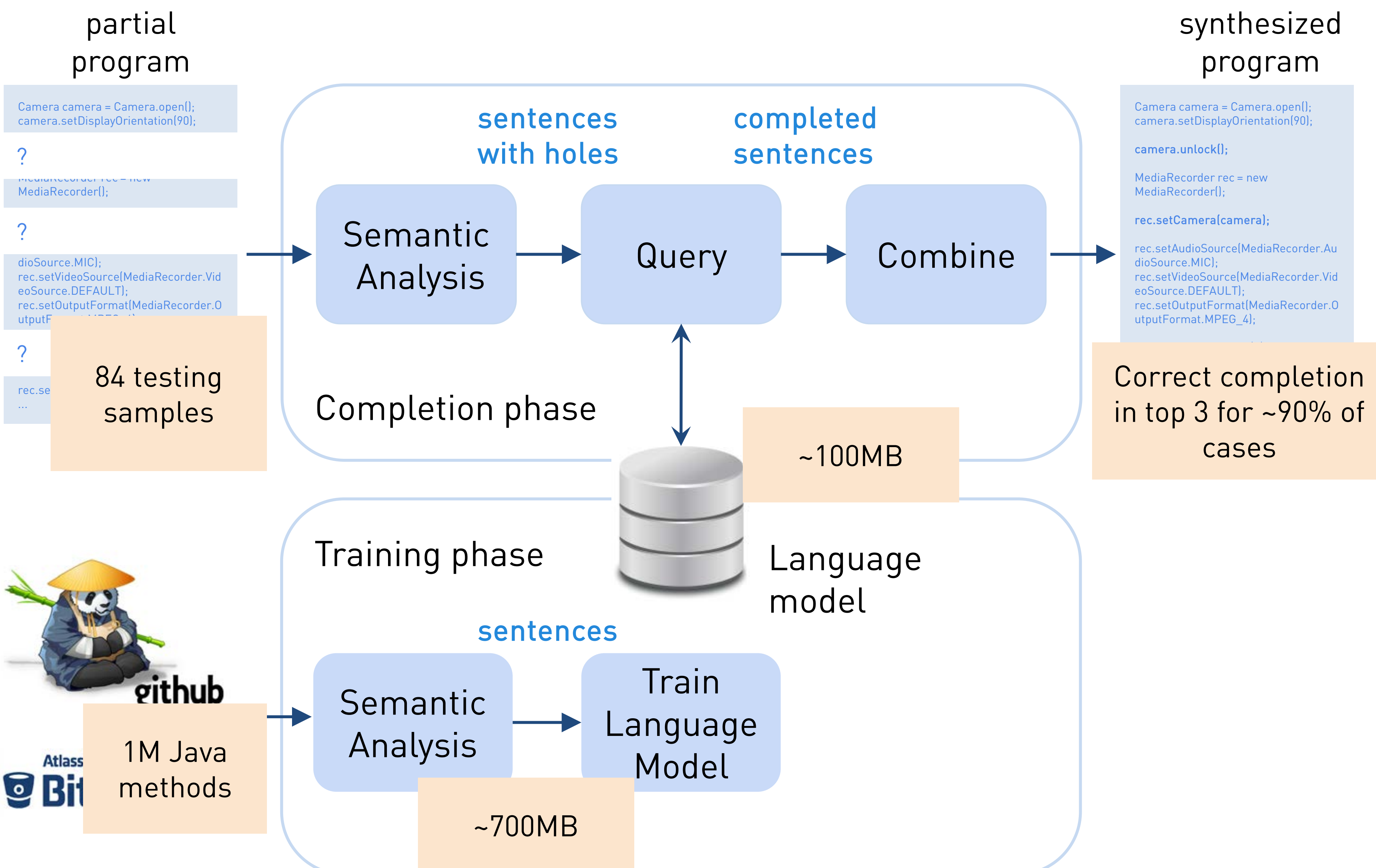
ML-based Code Synthesis

	Completion probability
getDefault$_{result}$ divideMessage **sendMultipartTextMessage**	0.0033
getDefault$_{result}$ divideMessage **sendTextMessage**	0.0016
getDefault$_{result}$ **sendTextMessage**	0.0073
getDefault$_{result}$ **sendMultipartTextMessage**	0.0010
divideMessage$_{result}$ **sendMultipartTextMessage$_{param3}$**	0.0821
length **length**	0.0132
length **split**	0.0080
length **sendTextMessage$_{param3}$**	0.0017

ML-based Code Synthesis

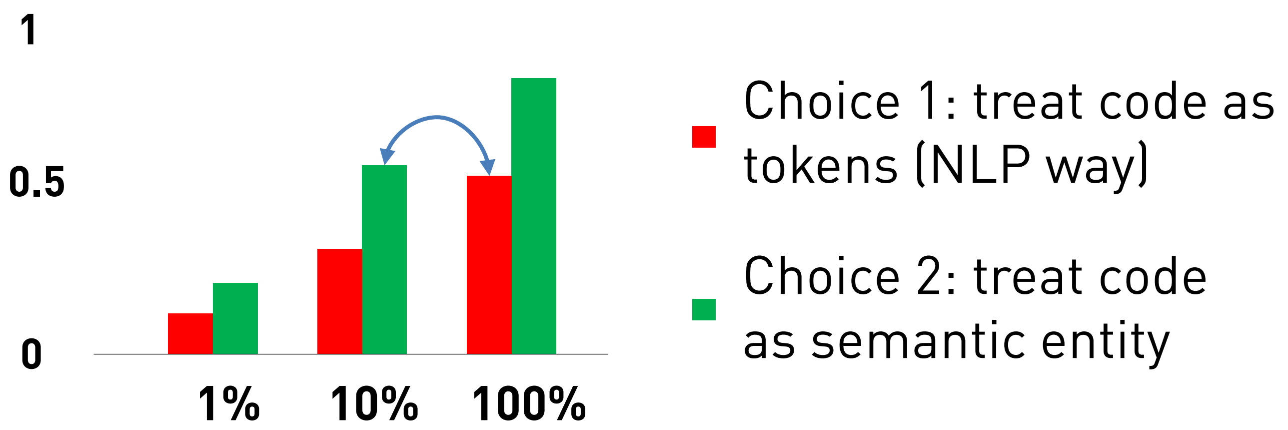
```
smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
  list = smsMgr.divideMessage(message);
  smsMgr.sendMultipartTextMessage(...list...);
} else {
  smsMgr.sendTextMessage(...message...);
}
```

The SLANG System



Do semantics help?

precision vs. % **of data used**



*need to decide what to ``compile'' into a word

semantic analysis benefit = 10x more data

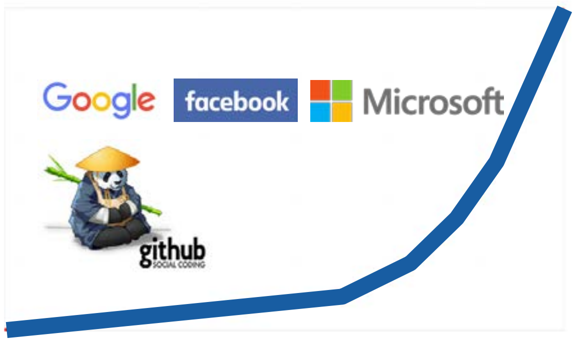
Next: Synthesis for Interpretable ML

Fundamental Question

Data

Learning

Model



Probabilistic Model

Widely Applicable

Efficient Learning

High Precision

Interpretable Model