NATIONAL RESEARCH
UNIVERSITY

Doctoral School of Computer Science

GENERATIVE MODELS FOR API COMPLETION

Chapter 3

Raychev V. Learning from Large Codebases, 2016

Fedin Gennadii

Moscow, 2018

CONTENT

CONTENT

THE PROBLEM

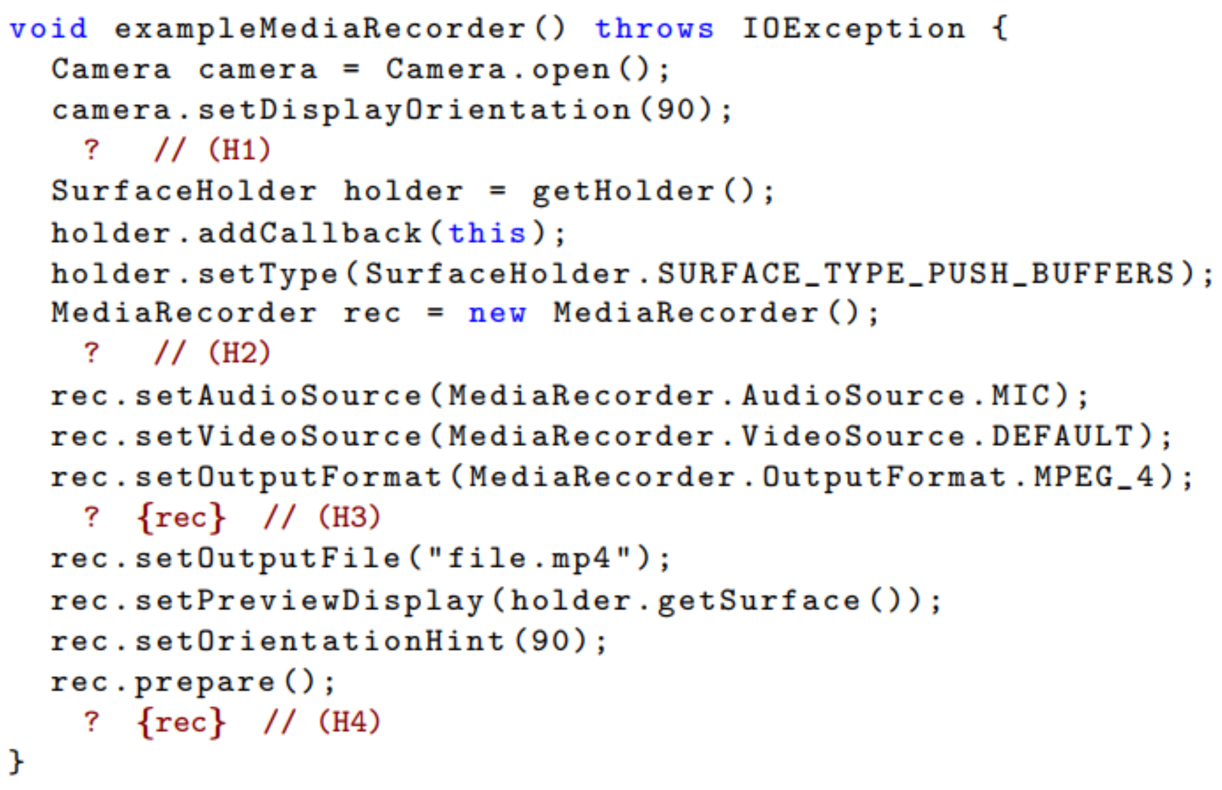
API completion

- Sequences of unknown length
- Ranked list of solutions
- Learning from a corpus where the actual completion positions (holes) are not available at learning time

THE PROBLEM

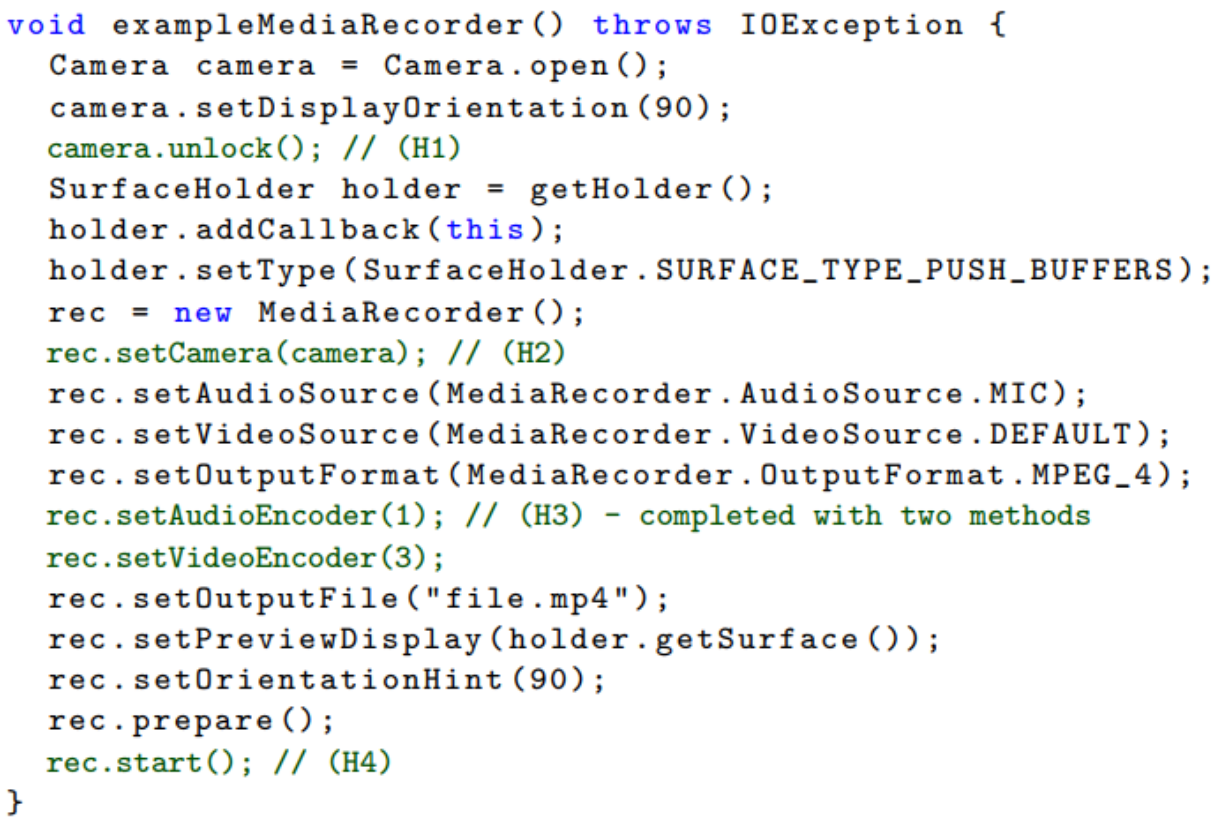
Solution

Input (partial program)



```
void exampleMediaRecorder() throws IOException {
  Camera camera = Camera.open();
  camera.setDisplayOrientation(90);
   ?  // (H1)
  SurfaceHolder holder = getHolder();
  holder.addCallback(this);
  holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
  MediaRecorder rec = new MediaRecorder();
   ?  // (H2)
  rec.setAudioSource(MediaRecorder.AudioSource.MIC);
  rec.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
  rec.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
   ? {rec}  // (H3)
  rec.setOutputFile("file.mp4");
  rec.setPreviewDisplay(holder.getSurface());
  rec.setOrientationHint(90);
  rec.prepare();
   ? {rec}  // (H4)
}
```

Output (completion)



```
void exampleMediaRecorder() throws IOException {
  Camera camera = Camera.open();
  camera.setDisplayOrientation(90);
  camera.unlock(); // (H1)
  SurfaceHolder holder = getHolder();
  holder.addCallback(this);
  holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
  rec = new MediaRecorder();
  rec.setCamera(camera); // (H2)
  rec.setAudioSource(MediaRecorder.AudioSource.MIC);
  rec.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
  rec.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
  rec.setAudioEncoder(1); // (H3) - completed with two methods
  rec.setVideoEncoder(3);
  rec.setOutputFile("file.mp4");
  rec.setPreviewDisplay(holder.getSurface());
  rec.setOrientationHint(90);
  rec.prepare();
  rec.start(); // (H4)
}
```

- Completion across multiple types
- Completion of parameters

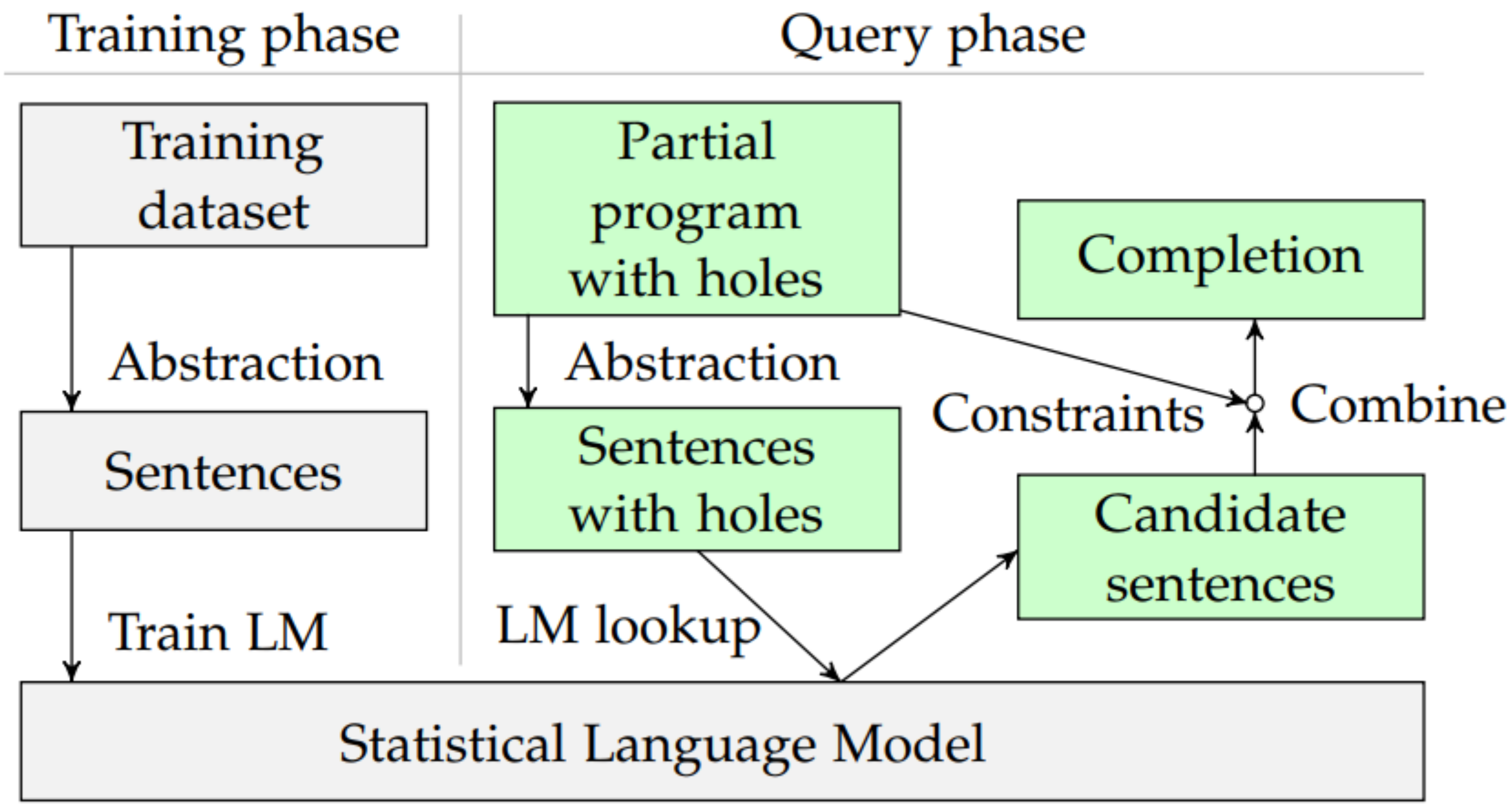
- Holes as sequences
- New fused completions

CONTENT

1. The Problem
2. *The architecture*
    - 2.1 Abstraction
    - 2.2 Language Models
        - 2.2.1 N-gram
        - 2.2.2 RNN
    - 2.3 Training
    - 2.4 Synthesis
3. Implementation
4. Results
5. Conclusion

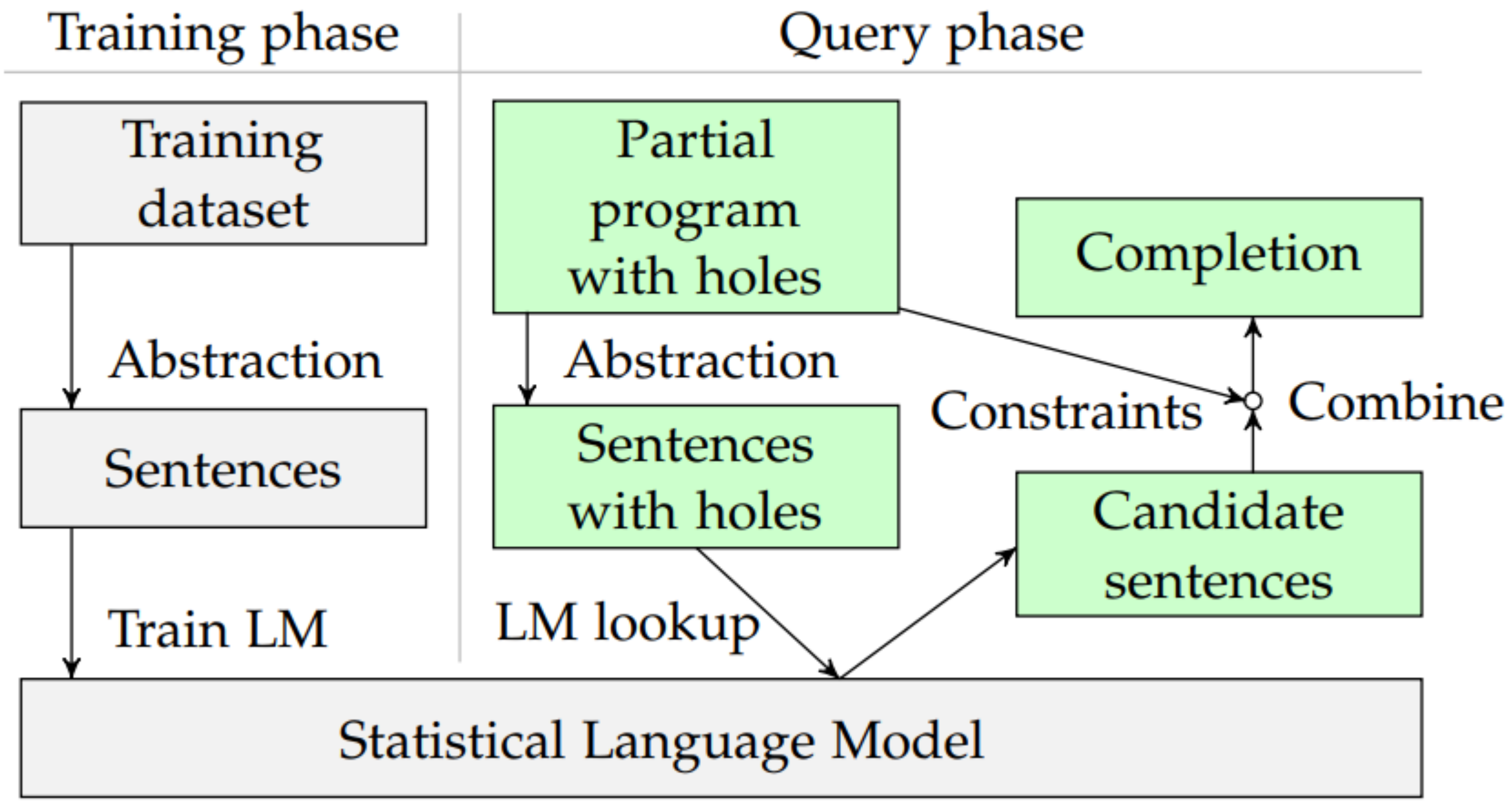
THE ARCHITECTURE

Slang software



CONTENT

1. The Problem
2. The architecture
    - *2.1 Abstraction*
    - 2.2 Language Models
        - 2.2.1 N-gram
        - 2.2.2 RNN
    - 2.3 Training
    - 2.4 Synthesis
3. Implementation
4. Results
5. Conclusion



ABSTRACTION

Semantic intermediate representation

State notations

o – an object,

$objects^*$ - an unbounded set of dynamically allocated objects,

$VarIds$ - a set of local variable identifiers,

$FieldId$ - a set of field identifiers,

L^* - a set of allocated objects,

$v^* \in Val = objects^* \cup \{null\}$,

$p^* \in Env = VarIds \to Val$,

$\pi^* \in Heap = objects^* \times FieldId \to Val$,

$< L^*, p^*, \pi^* >$ - concrete state,

ABSTRACTION

Semantic intermediate representation

History notations

$m(t_1, \dots, t_k)$ – a method signature,
p – “position” of object in the invocation (0 for $this$, $\overline{1, k}$ for position 1 to k, ret for new object),
$event = < m(t_1, \dots, t_k), p >$,
A – API with methods $m_1, \dots, m_n$,

Σ_A - all events over the API A,
$\mathcal{H}$ - set of all sequences of events from Σ_A
$his^* : L^* \to \mathcal{H}$, changes on object allocations and method invocations,
$< L^*, p^*, \pi^*, his^* > \to$
$< L^{*\prime}, p^{*\prime}, \pi^{*\prime}, his^{*\prime} >$,

ABSTRACTION

Abstract Semantics

Heap

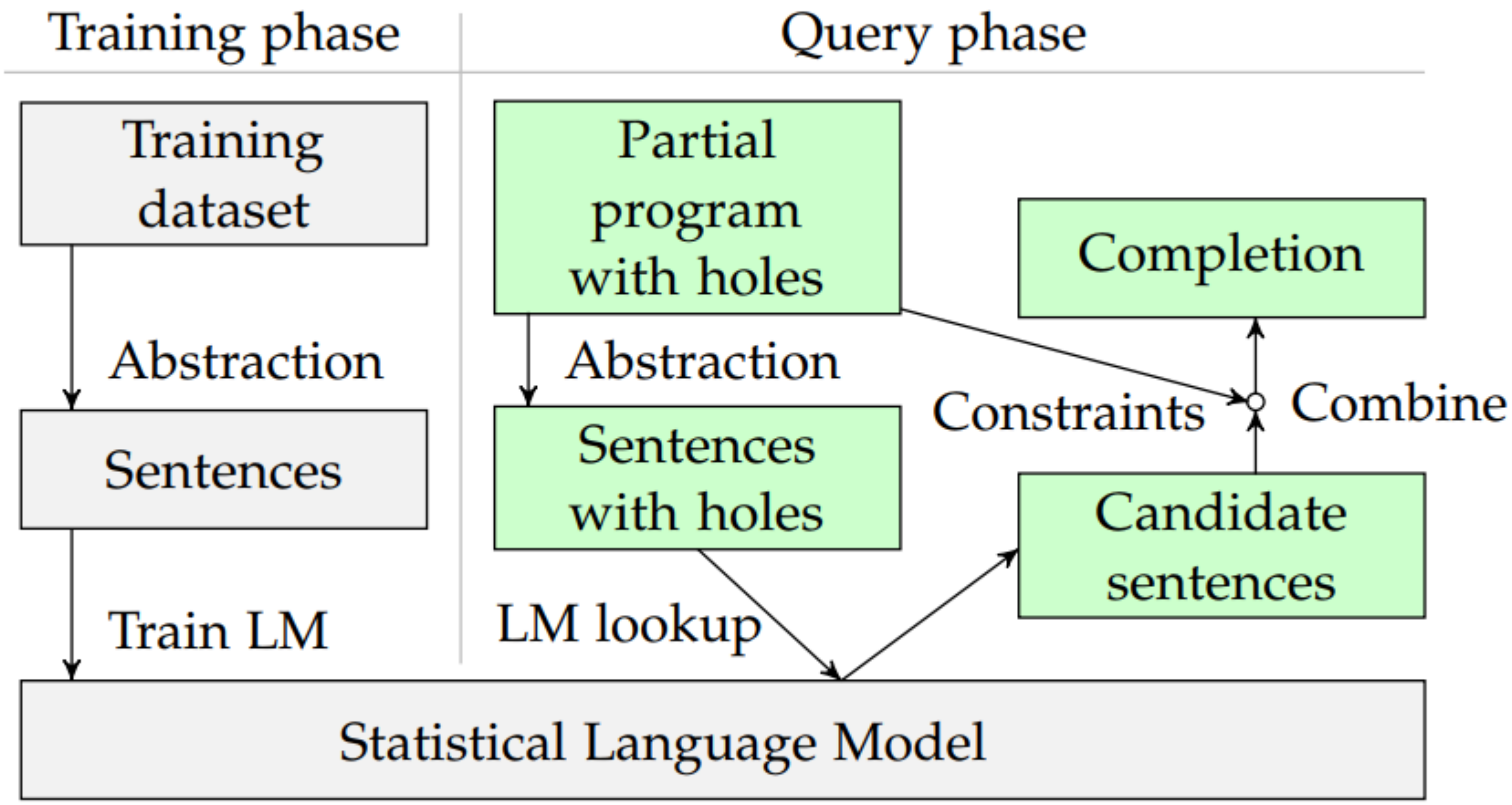
$\pi^* \in Heap = objects^* \times FieldId \to Val,$

$objects^* \to objects$ - bounded set of abstract objects

History

$h \subset \mathcal{H}$ - set of concrete histories of bounded length

CONTENT



Training phase
Query phase
Training dataset
Abstraction
Sentences
Train LM
Partial program with holes
Abstraction
Sentences with holes
LM lookup
Completion
Constraints
Combine
Candidate sentences
Statistical Language Model

LANGUAGE MODELS

General information

Notation

D – dictionary,
$w \in D$ – word,
$s = w_1 \cdot w_2 \cdot \ldots \cdot w_n$- sentence, an ordered sequence of words,
L – language, all sentences used in some particular domain,
$h_i = w_1 \cdot w_2 \cdot \ldots \cdot w_i$ - history,
$\Pr(s)$ – probability of sentence s.

Goal

To build a probabilistic distribution over all possible sentences in a language.

For instance as:

$$\Pr(s) = \prod_{i=1}^{m} \Pr(w_i | h_{i-1})$$

LANGUAGE MODELS

N-gram

$$\Pr(s) = \prod_{i=1}^{m} \Pr(w_i | w_{i-n+1} \cdot \ldots \cdot w_{i-1})$$

Trigram language model (the probability of a word depends on a pair of previous words)

$$\Pr(s) = \prod_{i=1}^{m} \Pr(w_i | w_{i-2} \cdot w_{i-1})$$

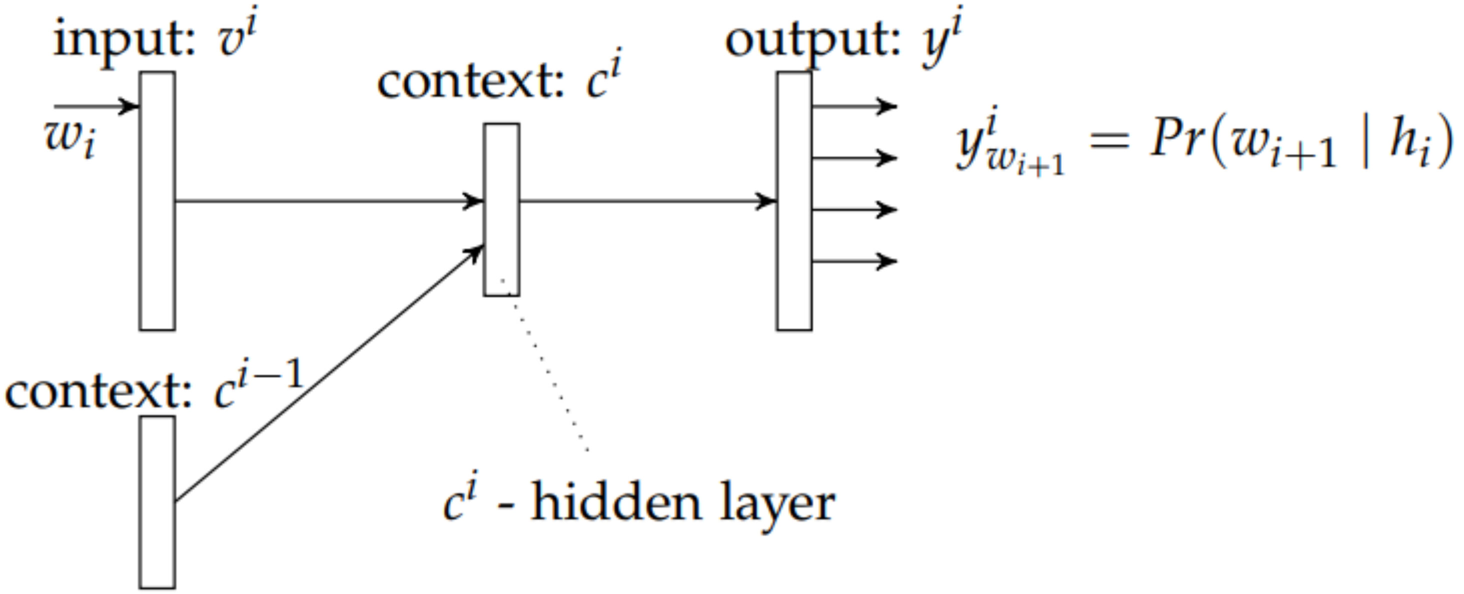
The probabilities are estimated by counting the number of occurrences of trigrams and bi-grams in the training data

Witten-Bell backoff smoothing

Unseen events as ones not having happened yet, the probability can be modeled by the probability of seeing it for the first time

LANGUAGE MODELS

RNN (recurrent neural networks)



Notations

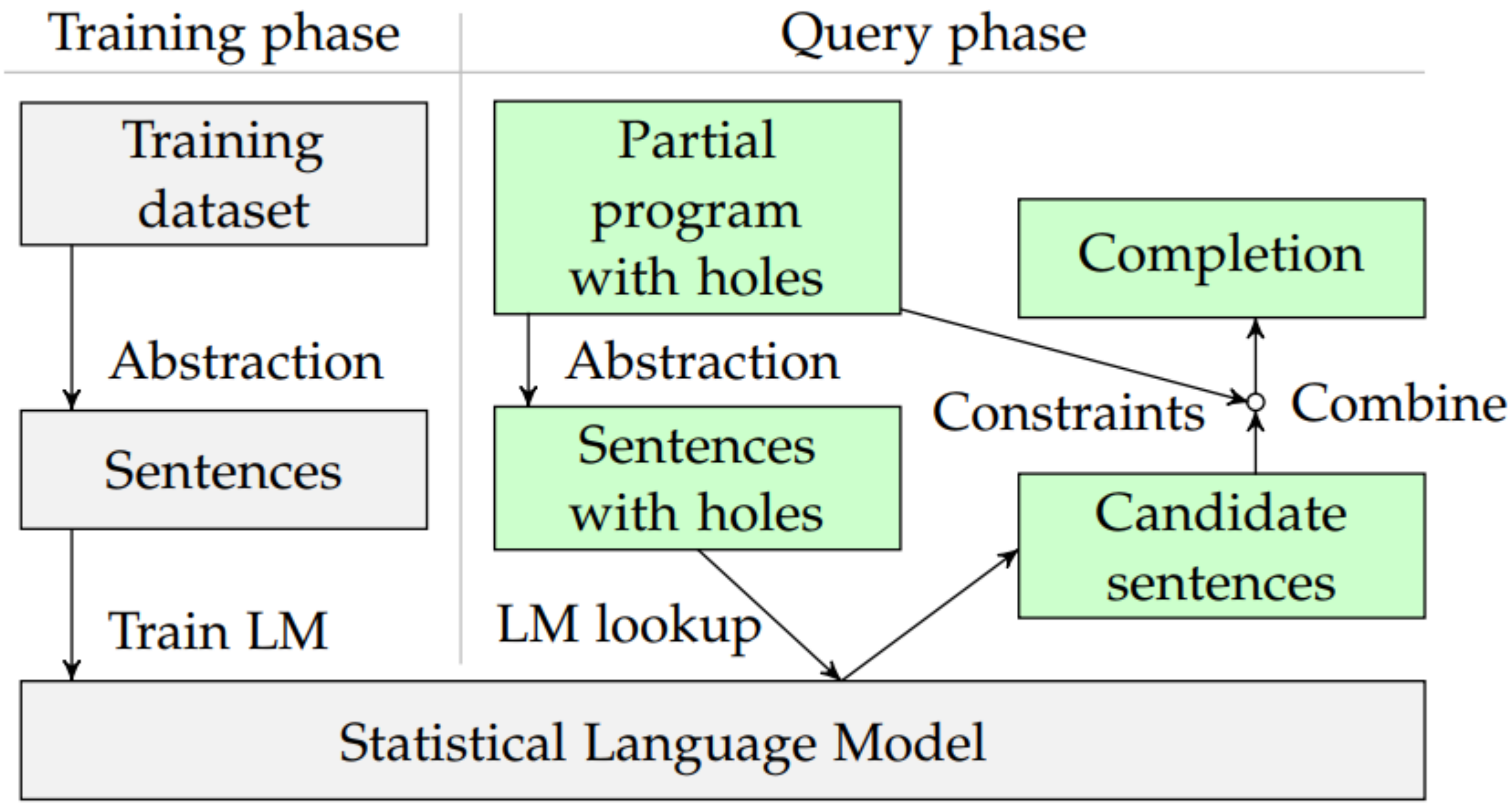
$v^i \in R^{|D|}$,
$y^i \in R^{|D|}$,
p - the size of the hidden layer,
$c^i \in R^p$,

Prediction

$c^i = f(v^i, c^{i-1})$,
$y^i = g(c^i)$,
$\Pr(w_{i+1}|w_1 \cdot \ldots \cdot w_i) \approx y^i_{w_{i+1}}$.

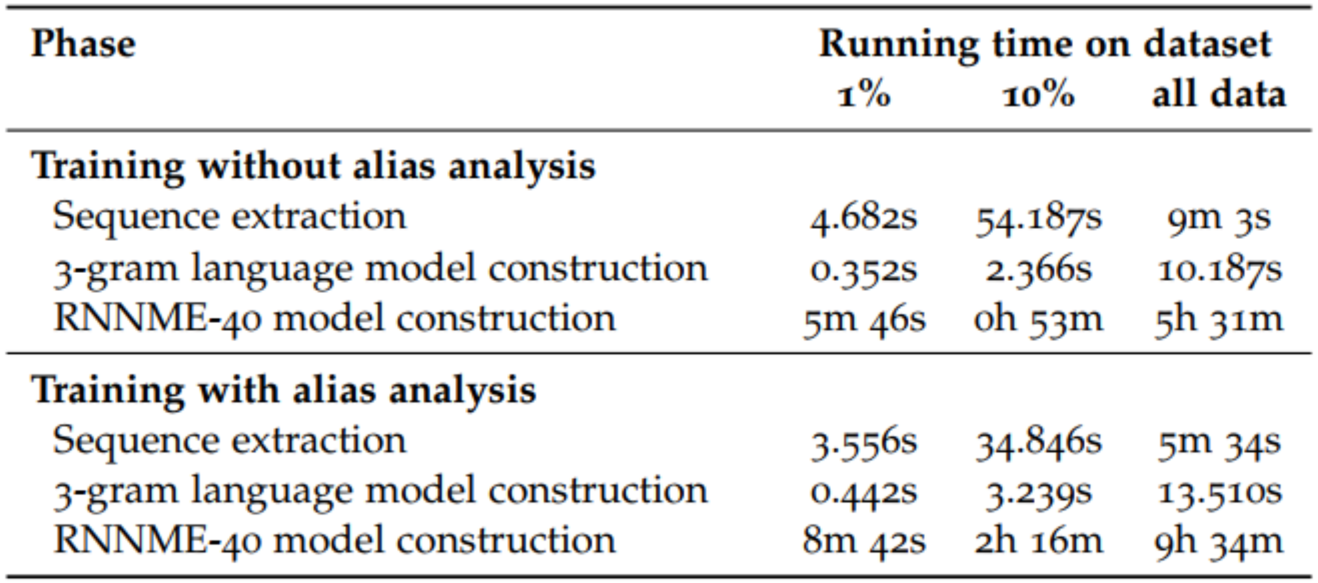
CONTENT

1. The Problem
2. The architecture
   - 2.1 Abstraction
   - 2.2 Language Models
     - 2.2.1 N-gram
     - 2.2.2 RNN
   - *2.3 Training*
   - 2.4 Synthesis
3. Implementation
4. Results
5. Conclusion



TRAINING

Training the language models

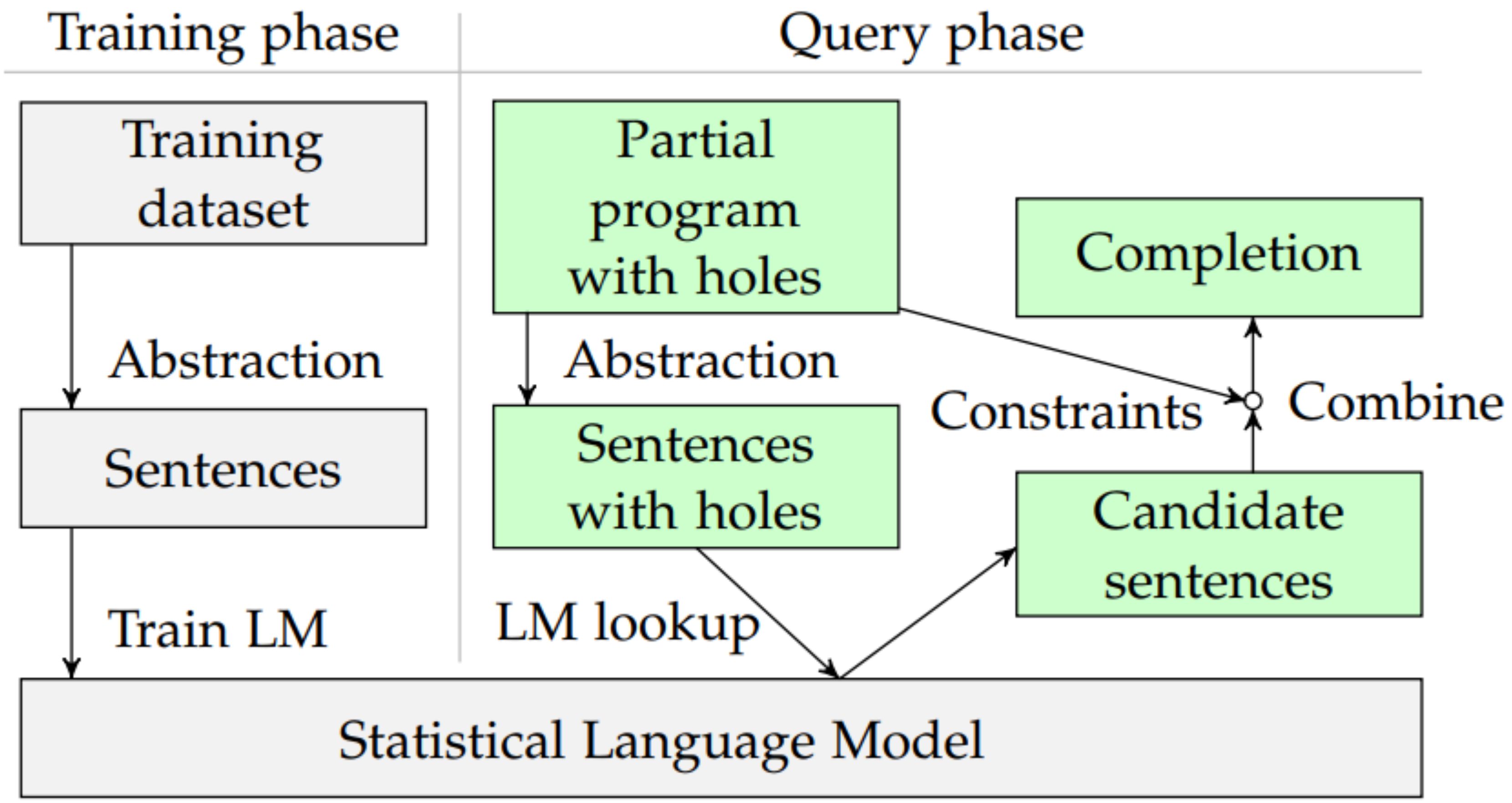


Phase	Running time on dataset 1%	10%	all data
Training without alias analysis			
Sequence extraction	4.682s	54.187s	9m 3s
3-gram language model construction	0.352s	2.366s	10.187s
RNNME-40 model construction	5m 46s	0h 53m	5h 31m
Training with alias analysis			
Sequence extraction	3.556s	34.846s	5m 34s
3-gram language model construction	0.442s	3.239s	13.510s
RNNME-40 model construction	8m 42s	2h 16m	9h 34m

3.5GHz Core i7 2700K, 16GB RAM, a solid state drive storage, 64-bit
Ubuntu 12.04, OpenJDK 1.7

CONTENT

1. The Problem
2. The architecture
   - 2.1 Abstraction
   - 2.2 Language Models
     - 2.2.1 N-gram
     - 2.2.2 RNN
   - 2.3 Training
   - *2.4 Synthesis*
3. Implementation
4. Results
5. Conclusion

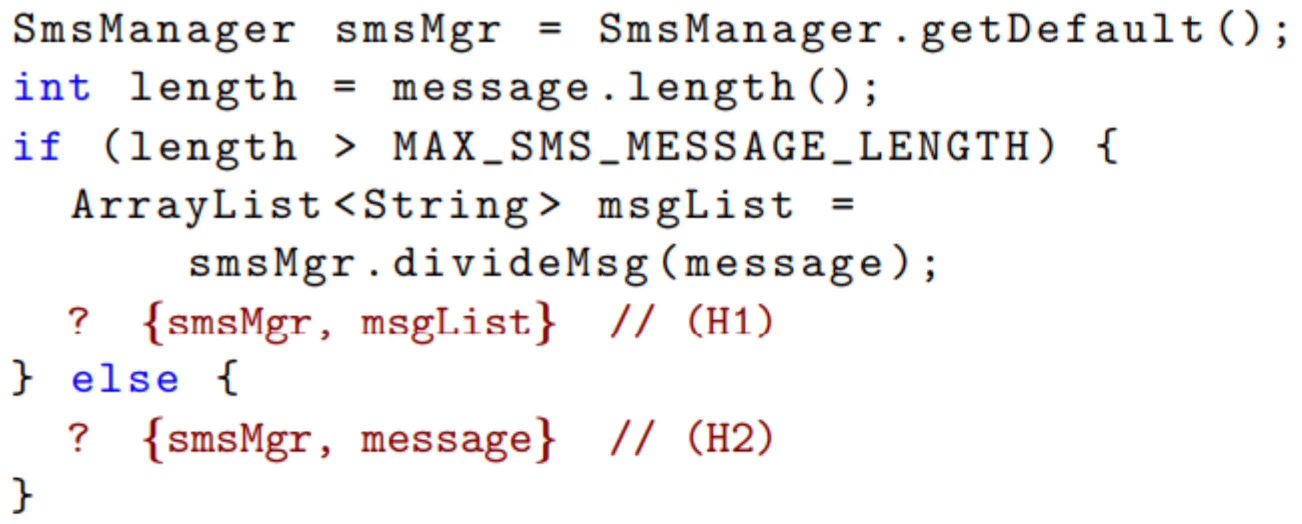


SYNTHESIS

Query phase

Specifying holes

$?\,lvars\colon l\colon u$

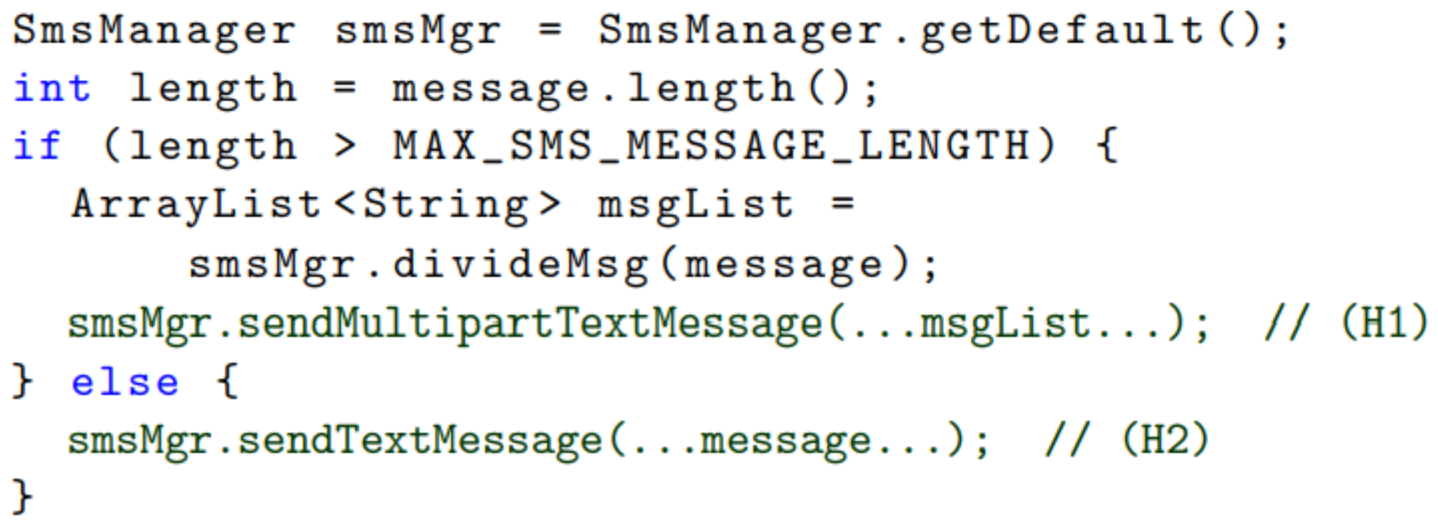


```
SmsManager smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
  ArrayList<String> msgList =
      smsMgr.divideMsg(message);
  ? {smsMgr, msgList}  // (H1)
} else {
  ? {smsMgr, message}  // (H2)
}
```

(a)

Specifying holes

1. Extract abstract histories with holes,
2. Compute candidate completions,
3. Compute an optimum and consistent solution



```
SmsManager smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
  ArrayList<String> msgList =
      smsMgr.divideMsg(message);
  smsMgr.sendMultipartTextMessage(...msgList...);  // (H1)
} else {
  smsMgr.sendTextMessage(...message...);  // (H2)
}
```

(b)

CONTENT

IMPLEMENTATION

Details

- The number of loop iterations L=2
- Sequences with more than K=16 words (invocations) are not considered
- Words that occur less than a certain number of times in the training corpus are replaced with placeholder unknown words
- The probability of a constant value as a parameter of a method is estimated by the number of times each constant was given as a parameter to the method in the training data

Data

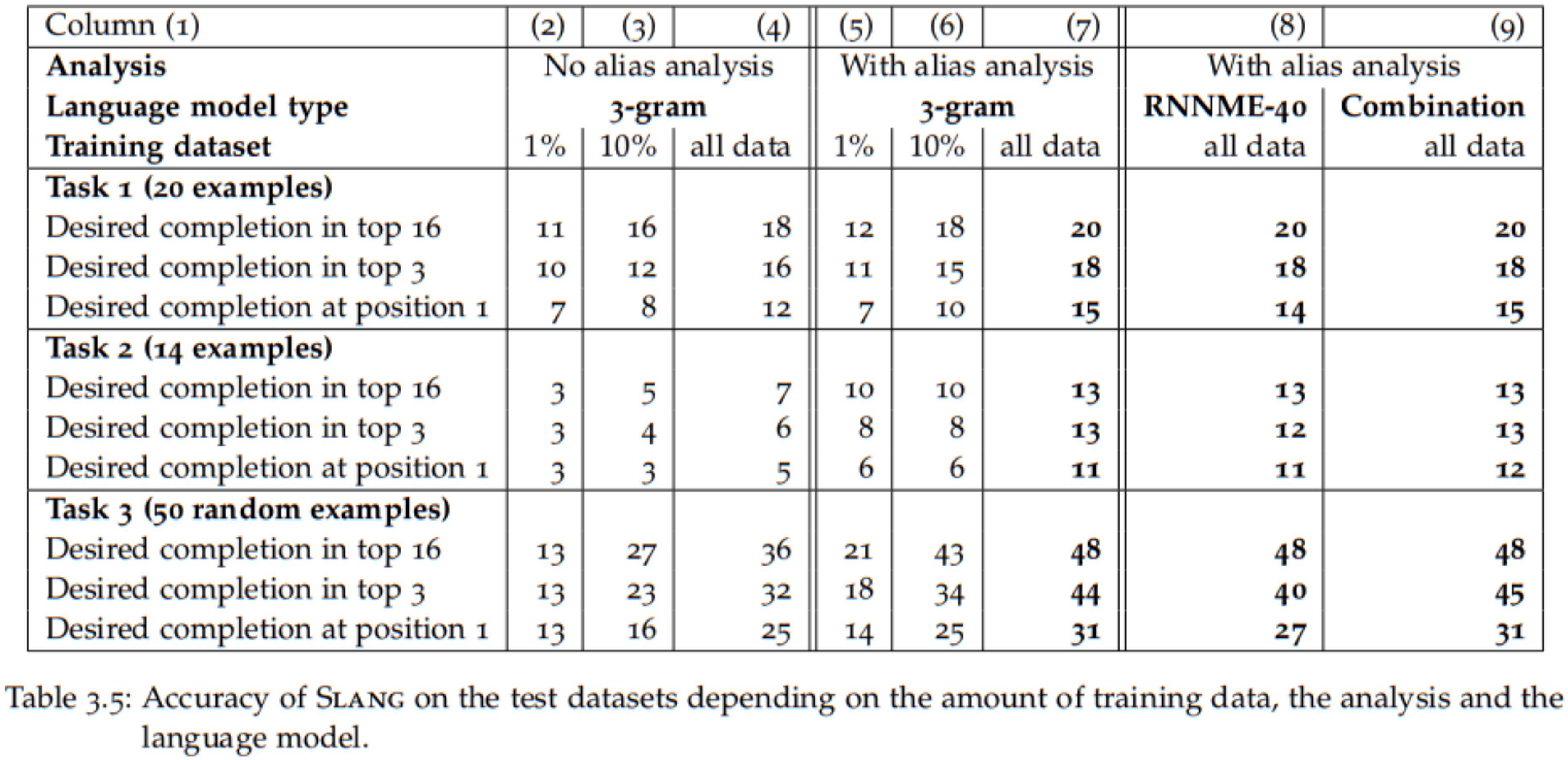
- 3,090,194 Android methods were used as training data
- Sources were compiled using a specially modified version of the partial compiler
- Compiled programs were converted into bytecode
- Bytecode was fed as training data into Slang

CONTENT

1. **The Problem**
2. **The architecture**
    - **2.1 Abstraction**
    - **2.2 Language Models**
        - **2.2.1 N-gram**
        - **2.2.2 RNN**
    - **2.3 Training**
    - **2.4 Synthesis**
3. **Implementation**
4. ***Results***
5. **Conclusion**

RESULTS

Evaluation of the software



Column (1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
Analysis	No alias analysis			With alias analysis			With alias analysis	
Language model type	**3-gram**			**3-gram**			**RNNME-40**	**Combination**
Training dataset	1%	10%	all data	1%	10%	all data	all data	all data
Task 1 (20 examples)								
Desired completion in top 16	11	16	18	12	18	**20**	**20**	**20**
Desired completion in top 3	10	12	16	11	15	**18**	**18**	**18**
Desired completion at position 1	7	8	12	7	10	**15**	**14**	**15**
Task 2 (14 examples)								
Desired completion in top 16	3	5	7	10	10	**13**	**13**	**13**
Desired completion in top 3	3	4	6	8	8	**13**	**12**	**13**
Desired completion at position 1	3	3	5	6	6	**11**	**11**	**12**
Task 3 (50 random examples)								
Desired completion in top 16	13	27	36	21	43	**48**	**48**	**48**
Desired completion in top 3	13	23	32	18	34	**44**	**40**	**45**
Desired completion at position 1	13	16	25	14	25	**31**	**27**	**31**

Table 3.5: Accuracy of SLANG on the test datasets depending on the amount of training data, the analysis and the language model.

Task 1. Single object single-method completion (20 tasks)
Task 2. General completion (14 code snippets)
Task 3. Random completion (50 methods, 23 with multiple holes)

CONTENT

1. The Problem
2. The architecture
    - 2.1 Abstraction
    - 2.2 Language Models
        - 2.2.1 N-gram
        - 2.2.2 RNN
    - 2.3 Training
    - 2.4 Synthesis
3. Implementation
4. Results
5. *Conclusion*

CONCLUSION

GENERATIVE MODELS FOR API COMPLETION

- A new approach for creating probabilistic models of code was presented
- A link between statistical models for code and statistical models for natural languages was established
- A tool for code completion “Slang” was implemented
- An experimental evaluation of this tool was proposed

HIGHER • SCHOOL • OF • ECONOMICS

NATIONAL RESEARCH
UNIVERSITY