Machine Learning for Software Engineering with Scarce Data

Audition DR2 CNRS
CRIStAL Lille (UMR 9189)

Romain Robbes, 30/03/2022

18 years in Software Engineering Research

2004 — 2008: PhD, Lugano, Switzerland
2010 — 2015: Tenure-track, Santiago, Chile
(2014: Visiting, Hamburg, Germany)
2016 — 2017: Tenured, Santiago, Chile
2017 — now : Tenured, Bolzano, Italy

Collaborations: Research and Service

Publications and service in quality venues

12 Full papers in top SE & PL venues:
ICSE, FSE, ASE, ECOOP, OOPSLA
1 award
+10 short papers

13 Articles in top journals:
EMSE, TSE, JSS, CSUR
+11 other journals

17 Full papers in specialist venues:
ICSME, MSR, ICPC, SANER/WCRE
4 awards

3 Track co-chair:
ICSE * 2, OOPSLA

2 Editorial boards:
EMSE, JSS

3 PC co-chair:
MSR * 2, WCRE

Software systems can be very complex …

… and still need to continually change

Software engineering is a complex topic


Human Studies
Mining Software Repositories (MSR)
Machine Learning for Software Engineering (ML4SE)

Major Research contributions

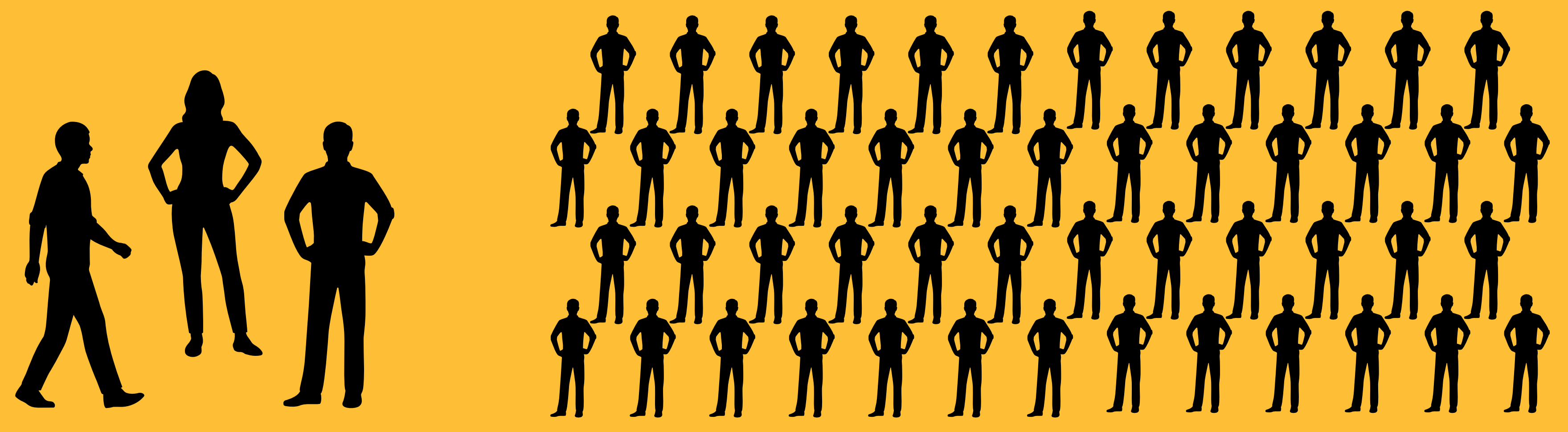
Mining Software Repositories

MSR brings multiple perspectives on Software Systems

Human Studies

Bring back the human perspective:

- **Do we ask the right questions?**
- **Do we have the right answers?**



ML4SE

Machine Learning for Software Engineering can integrate these multiple perspectives

SE artefacts mix structure and language

Hi Alice,

Can you help me figure out this bug?

```
// updating credentials
User u = DB.getUser(id)
u.setPassword(username)
DB.update(id, u)
```

Hope you can help, it's urgent.
Thanks in advance!

Cheers,
Bob

Deep Learning learns vector representations of the meaning of words in context

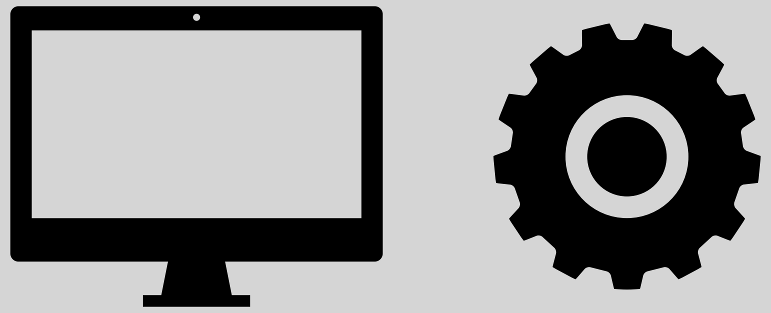
Bug	.8	.2	.7	.1	.9	.5	.7	.3	.1	.3	.4	.9
Defect	.9	.2	.8	.2	.8	.4	.6	.3	.3	.4	.4	.8

Homogeneous vector representations support multiple types of artefacts

Some models support structure such as trees or graphs

Code: machine and human perspectives

Program analysis ignores identifiers and code comments



```
// … … … … … …
i272.f428(i823);
```

Identifiers and comments are critical for program understanding


```
// updating credentials
u.setPassword(username);
```

Merging machine and human perspectives can detect new kinds of bugs (e.g., code-comment incoherence)

Example: Code Completion

Every programmer uses code completion

alphabeticallySorted
exhaustive
identifier
listWhichIsQuiteLong
manyWhoAre
not
remotelyCloseTo
whatYouWant

→

whatYouWant
shortList

2008 **We define code completion as a task and develop optimized algorithms**

2010 **We implement and evaluate a code completion tool, still used in practice**

2017 **Others use Neural Language Models for code completion, but struggle**

2020 **We successfully apply Neural Language Models to code completion**

Source Code Neural Language Models (NLMs)

NLMs are pre-trained on source code by predicting tokens based on context

```
public static void main(String [ ] args) {
     String commandName = args [0] ;
     String uncommonFlag = args [1] ;
```

Used "out of the box" for code completion ✓

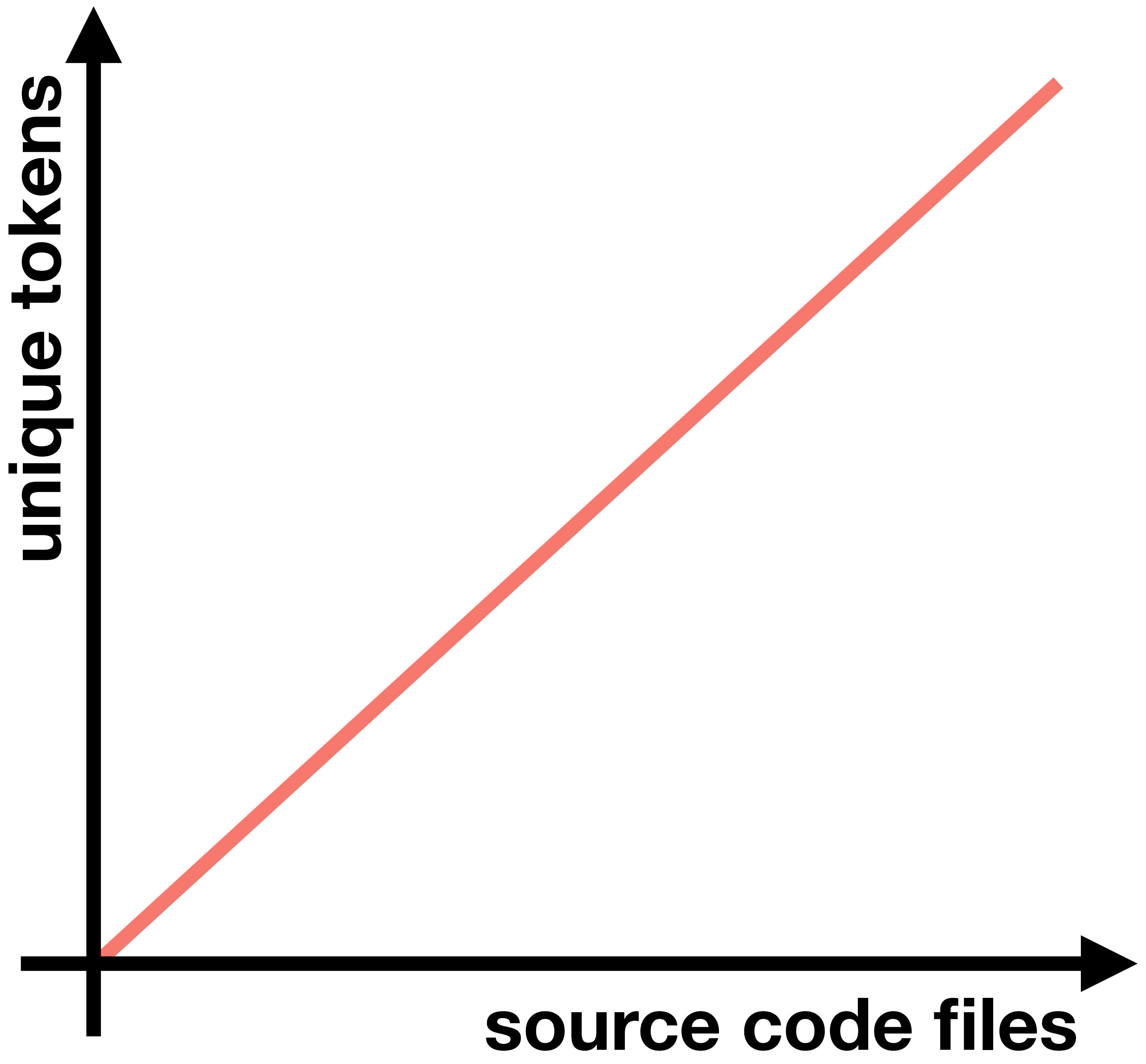
Fine-tunable on many other tasks

Buggy code? $\begin{cases} \text{Yes: 0.02} \\ \text{No: 0.98} \end{cases}$ ✓

NLMs have a closed vocabulary of pre-defined tokens ✗

Developers create new identifiers at will

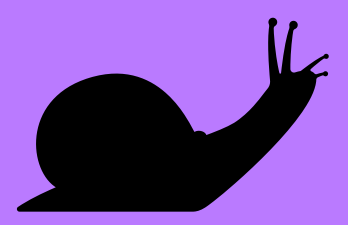
```
foo
Foo
Bar
FooBar
FOO_BAR
a234
a282
a9095
my_taylor_is_rich
MeinTaylorIstReich
mon-tailleur-est-riche
FooBarManagerController
ArbitrarilyLongAndComplexIdentifierIsAllowed
FooBarManagerControllerRuntimeInstantiationException
```



New code contains new tokens, leading to a Vocabulary Explosion

Large vocabulary leads to three issues

Slow training



Does not scale to millions of tokens

Infrequent tokens

?	?	?	?	?	?

Don't have good representations

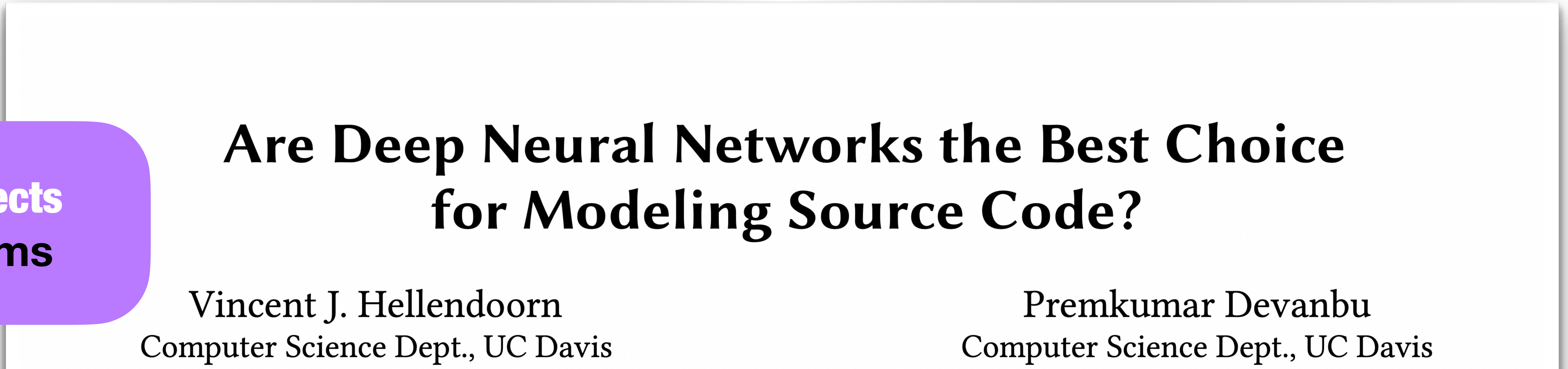
Unseen tokens

<UNK>

New code has unknown tokens

Only ~100 projects

NLM < n-grams



Are Deep Neural Networks the Best Choice for Modeling Source Code?

Vincent J. Hellendoorn
Computer Science Dept., UC Davis

Premkumar Devanbu
Computer Science Dept., UC Davis

We went back to pre-processing

Exhaustive preprocessing on 10,000 Java projects (also C, Python)

	# Unique Tokens	Corpus Size	Frequency (%< 10)	OOV (% @ 75K)
No preprocessing `FooBarManagerController`	11.6M	1.0	83%	79%
Coding Conventions `Foo Bar Manager Controller`	1.3M	1.6	81%	20%
Best standard approach **(Out of a dozen)**	0.5M	1.9	70%	9%

HUGE vocabulary

Many rare tokens

Many OOV tokens

This works, but … this is not enough

We looked further

Exhaustive preprocessing on 10,000 Java projects (also C, Python)	# Unique Tokens	Corpus Size	Frequency (%< 10)	OOV (% @ 75K)
No preprocessing `FooBarManagerController`	11.6M	1.0	 83%	79%
Best standard approach (Out of a dozen)	0.5M	1.9	 70%	9%

Byte-Pair Encoding (BPE) was used in Translation, but not yet in NLMs

BPE learns an **open vocabulary** bottom-up, from common character sequences

BPE solves all the vocabulary issues

Exhaustive preprocessing on 10,000 Java projects (also C, Python)	# Unique Tokens	Corpus Size	Frequency (%< 10)	OOV (% @ 75K)
No preprocessing `FooBarManagerController`	11.6M	1.0	83%	79%
Best standard approach (Out of a dozen)	0.5M	1.9	70%	9%
Byte-Pair Encoding (BPE)	**10K**	**1.4**	**1%**	**0%**

```
Foo Bar ManagerController

http client lib

Se hr Lan ge Que ll code Ken nun g
```

Performs well

Degrades gracefully

1,000 times smaller

Few Rare tokens

No OOV

Outcomes

We built an **open-vocabulary** NLM (with additional code-specific extensions)

It scales to **10,000+** software projects, **100 X** more than previous work

A thorough evaluation shows **large improvements**

MODEL	Java							Java Identifiers			C				Python			
	Static		Dynamic		Maintenance		Bugs	Dynamic			Static		Dynamic		Static		Dynamic	
	Ent	MRR	Ent	MRR	Ent	MRR	% Ent ↓	R@1	R@10	MRR	Ent	MRR	Ent	MRR	Ent	MRR	Ent	MRR
Small Train																		
n-gram	6.25	53.16	5.54	56.21	5.30	58.32	1.81	17.24	34.66	22.26	6.51	55.20	4.14	57.34	5.30	43.63	4.81	47.39
Nested	-	-	3.65	66.66	2.94	71.43	-	37.46	56.85	43.87	-	-	3.61	62.25	-	-	4.05	54.02
Cache	-	-	3.43	69.09	3.32	70.23	-	40.13	59.52	46.57	-	-	2.19	75.09	-	-	3.22	62.27
Nested Cache	-	-	2.57	74.55	2.23	77.04	-	49.93	70.09	56.81	-	-	2.01	76.77	-	-	2.89	65.97
Closed NLM	**4.30**	62.28	3.07	71.01	-	-	1.81	30.96	49.93	37.20	4.51	60.45	3.20	72.66	3.96	**81.73**	3.34	84.02
Heuristic NLM	4.46	53.95	3.34	64.05	-	-	1.04	39.54	58.37	45.28	4.82	52.30	3.67	61.43	4.29	65.42	3.56	71.35
BPE NLM (512)	4.77	63.75	2.54	77.02	**1.60**	**78.69**	3.26	45.49	67.37	52.66	4.32	62.78	1.71	76.92	3.91	81.66	2.72	86.28
BPE NLM (512) + cache	-	-	-	77.42	-	-	-	50.49	68.16	56.30	-	-	-	-	-	-	-	-
BPE NLM (2048)	4.77	**64.27**	**2.08**	77.30	-	-	**3.60**	48.22	69.79	55.37	**4.22**	**64.50**	**1.59**	**78.27**	**3.66**	81.71	**2.69**	**86.67**
BPE NLM (2048) + cache	-	-	-	**78.29**	-	-	-	**52.44**	**70.12**	**58.30**	-	-	-	-	-	-	-	-
Large Train																		
Nested Cache	-	-	2.49	75.02	2.17	77.38	-	52.20	72.37	59.09	-	-	1.67	**84.33**	-	-	**1.45**	71.22
BPE NLM (512)	3.15	70.84	1.72	79.94	**1.04**	**81.16**	4.92	51.41	74.13	59.03	3.11	70.94	1.56	77.59	3.04	84.31	2.14	87.06
BPE NLM (512) + cache	-	-	-	80.29	-	-	-	55.68	**74.30**	61.94	-	-	-	-	-	-	-	-
BPE NLM (2048)	**2.40**	**75.81**	**1.23**	82.41	-	-	**5.98**	57.54	72.18	62.91	**2.38**	**80.17**	**1.36**	83.24	**2.09**	**86.17**	1.90	**87.59**
BPE NLM (2048) + cache	-	-	-	**83.27**	-	-	-	**60.74**	73.76	**65.49**	-	-	-	-	-	-	-	-

top 2% of submitted papers
(distinguished paper)

90+ citations in 2 years
data, code, models available

Needed expertise in both **ML & SE**, willingness to **revisit assumptions**

Research Program:
ML4SE in Scarce Data Scenarios

Source Code NLMs are getting … larger

Ours (2020)

1.8 GB

OpenAI's Codex (2021)

GitHub Copilot

159 GB

Trained on 55 Million
GitHub code repositories
Runs only in the cloud
Closed source

DeepMind's AlphaCode (2022)

715 GB

Are there 55 million repositories of COBOL?

MANTIS?

NCL?

4D?

Legacy systems, in orphan languages

Legacy Systems: old, but **critical infrastructure** for important institutions

Complex code, many **quality issues**
Missing documentation and tests

Knowledge loss over decades
Constant need for software evolution

Orphan Languages: decades-old proprietary languages with **little or no support**

Almost no open source presence:
Not enough data for out of the box ML approaches

Two challenges: scientific and practical

Learn with less data

In practical settings

For orphan languages

Relevant for legacy systems

Learn with less data, but with more …

“Big data” ➡ Small preprocessing

`source code` **== natural text**

Large but generic NLMs

“Small data” ➡ SE-specific preprocessing

`source code !=` **natural text**

more structure

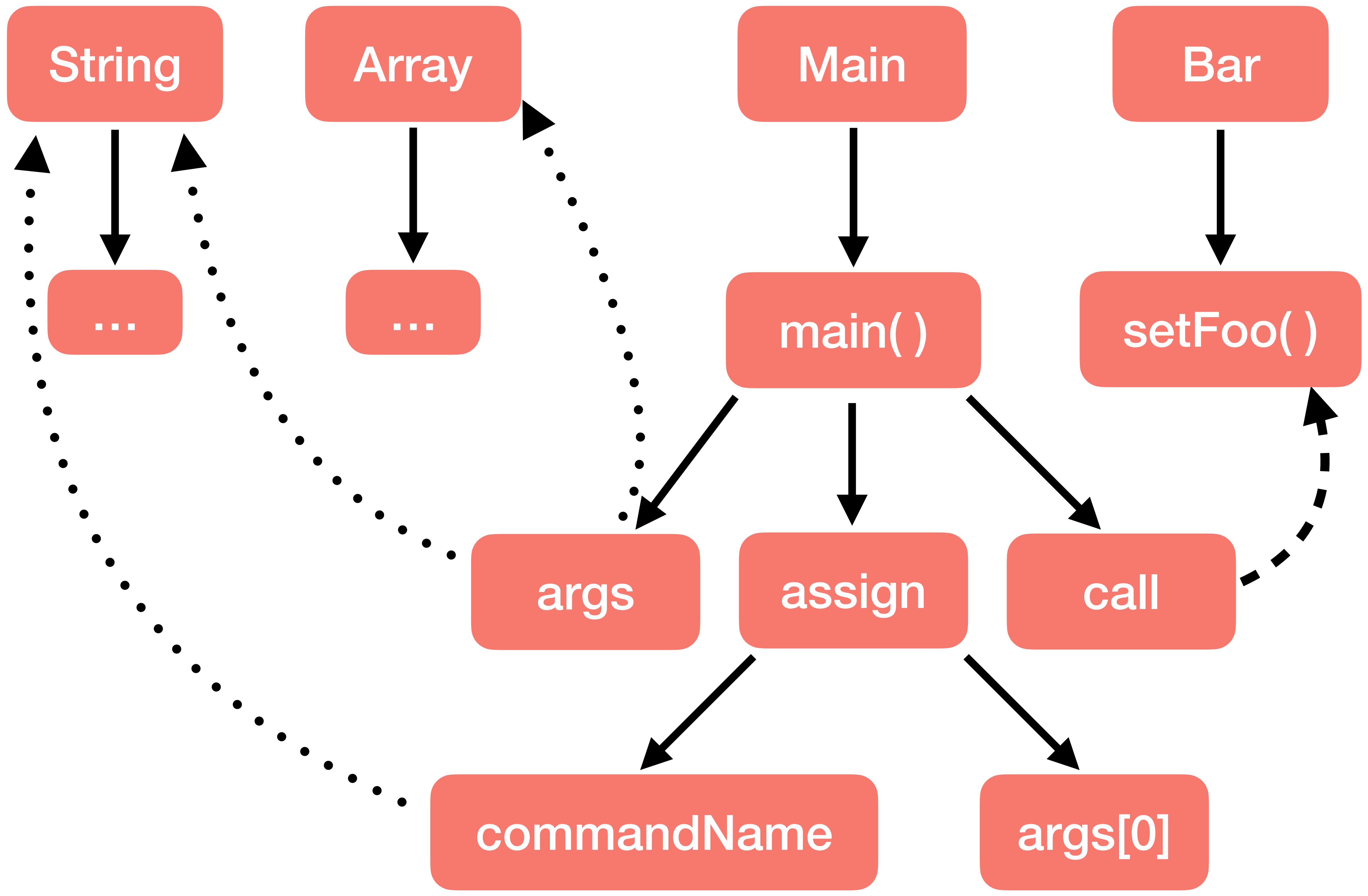
more information

better transfer

Learn from less with more structure

```
public static void main(String [ ] args) {
      String commandName = args [0] ;
      String uncommonFlag = args [1] ;
      myBar.setFoo(args[2])
```

Learning implicit information from scratch requires a large model & lots of data



Explicit information: smaller model, less data

Most papers focus on the ML perspective

… data work

Collect and process artefacts

Everybody wants to do the model work, not the …

Develop approach

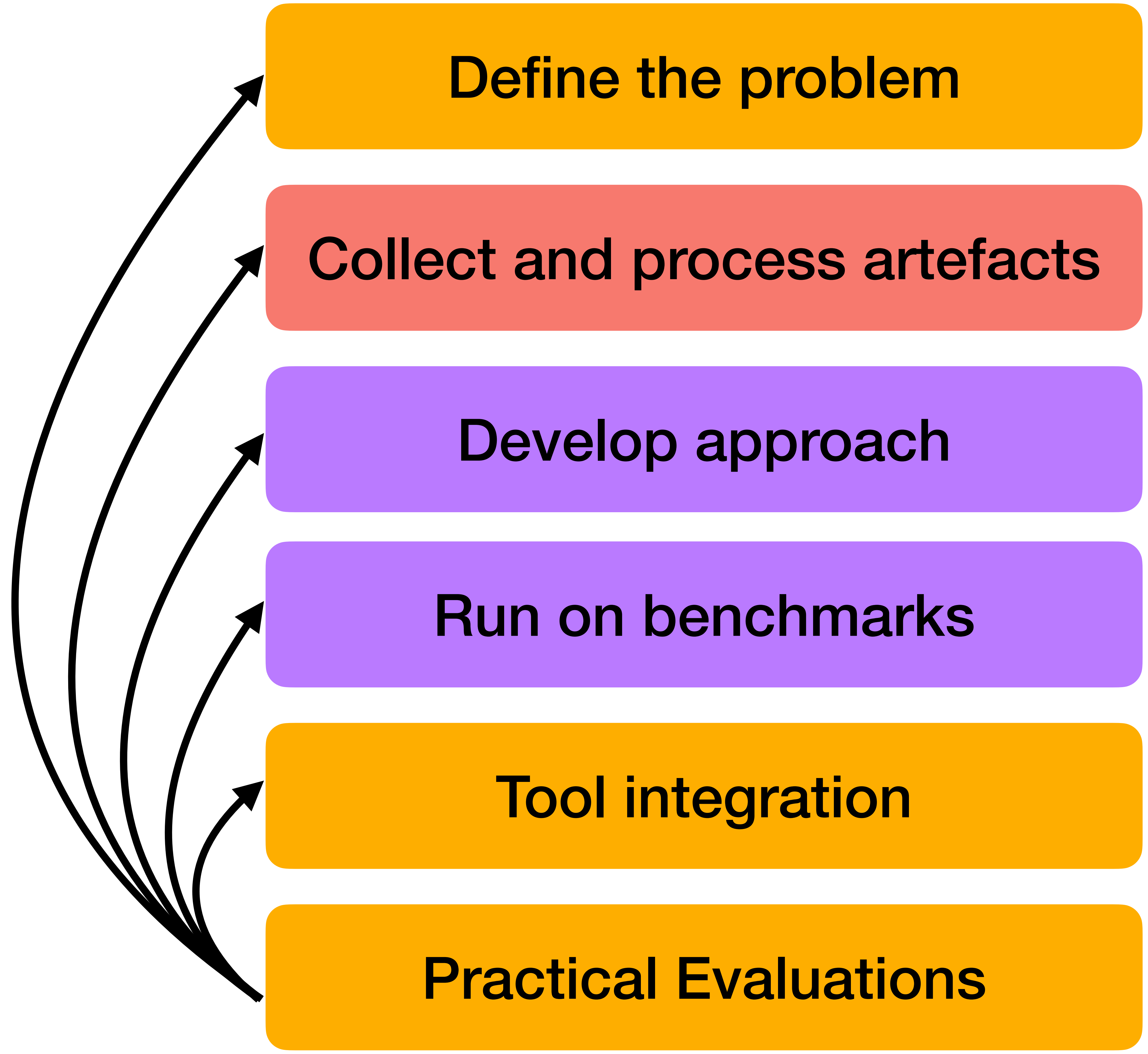
Run on benchmarks

Practical scenarios require a holistic view

… scoping	Define the problem
… data work	Collect and process artefacts
Everybody wants to do the model work, not the …	Develop approach
	Run on benchmarks
… integration	Tool integration
… evaluation	Practical Evaluations

Practical scenarios provide feedback…

… which may lead to **revisiting decisions**



The ideal team for this project is RMoD

RMod

RMoD works with **industrial partners** to help them support their **legacy systems**

Berger Levrault BL SIEMENS arolla Lifeware

Practical problems include test generation, code migration, business rule extraction, bug triage, & many others

Integrating with RMoD

RMod

Reinforce RMoD in: MSR | Human Studies

RMoD seeks: ML4SE

For me, a practical perspective: Berger Levrault · SIEMENS · arolla · Lifeware

RMod & I have extensive previous collaborations: SQJ 2018 · FSE 2016 · ICSME 2015 · ...

SATToSE 2019

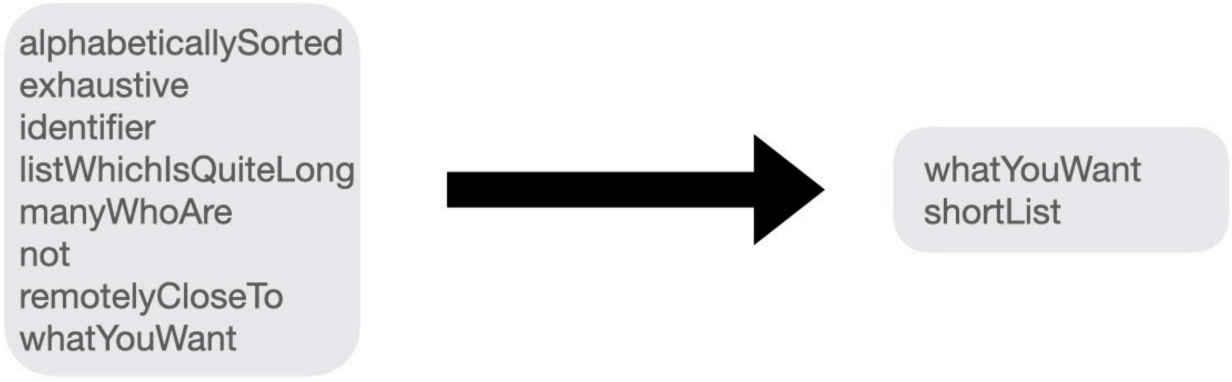
Major SE Contributions

MSR brings **multiple perspectives** on Software Systems

Human Studies bring back the **human perspective**

ML4SE **integrates** these multiple perspectives

Perspectives on code completion



- Defined code completion as a task
- Released and evaluated a tool, still used
- Applied Deep Learning to Code Completion

A research program with two challenges

Learn with less data

In practical settings

For orphan languages

Relevant for legacy systems

RMoD is the ideal team

RMod

Its partners have legacy systems & practical problems

Berger Levrault · SIEMENS · arolla · Lifeware

My skills & interests are a great match with RMoD