Fixing and Generating Programs for Fun and Profit

José Cambronero

PROSE, Microsoft

josepablocam @ (gmail, twitter, github)

PROSE Team

Gustavo Soares
Principal Research Manager

Ananya Singha
Research Fellow

Danny Simmons
Principal Software Engineer

Sumit Gulwani
Partner Research Manager

Vu Le
Principal Research Manager

Abishai Ebenezer
Research Fellow

José Cambronero
Senior Researcher

Priyanshu Gupta
Associate Researcher

Daniel Perelman
Senior Researcher

Sherry Shi
Senior Software Engineer

Avishree Khare
Research Fellow

Ashish Tiwari
Principal Researcher

Chris Parnin
Principal Researcher

Austin Henley
Senior Program Manager

Anirudh Khatry
Research Fellow

Bhavya Chopra
Research Fellow

Arjun Radhakrishna
Principal Researcher

Harshit Joshi
Research Fellow

Clint Simon
Senior Software Engineer

Mukul Singh
Associate Researcher

Yasharth Bajpai
Associate Researcher

Het Shah
Research Fellow

Saksham Gupta
Research Fellow

Gust Verbruggen
Researcher

https://www.microsoft.com/en-us/research/group/prose/

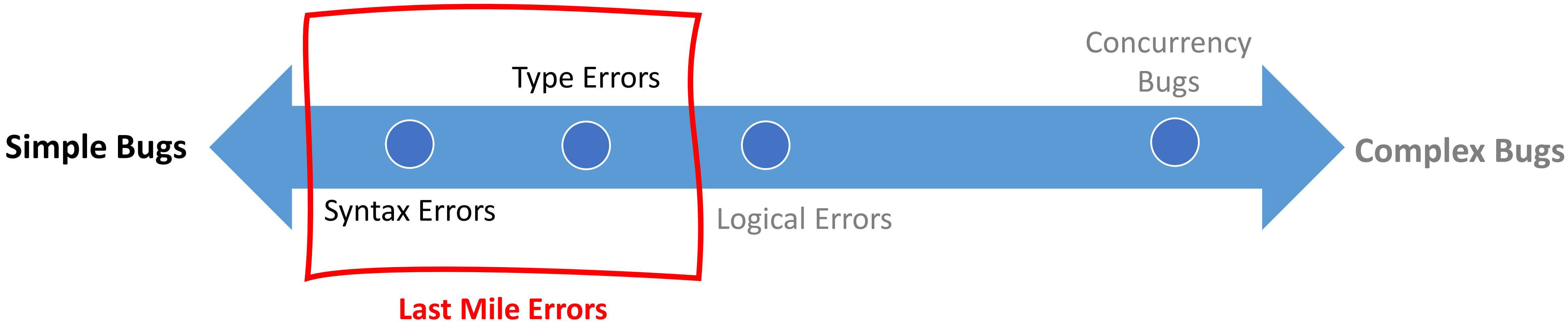
Agenda

- Fixing Last Mile Errors
  - LaMirage: Neurosymbolic approach
  - RING: LLM-based approach
  - FLAME: Custom Excel-Specific LLM trained on formulas
- Domain-Specific Synthesis (maybe…based on time)
  - FormaT5: Natural language to conditional formatting rules
- Open Discussion
  - Building software in PROSE
  - Grad school vs Industry
  - Career changes
  - Anything else on your mind

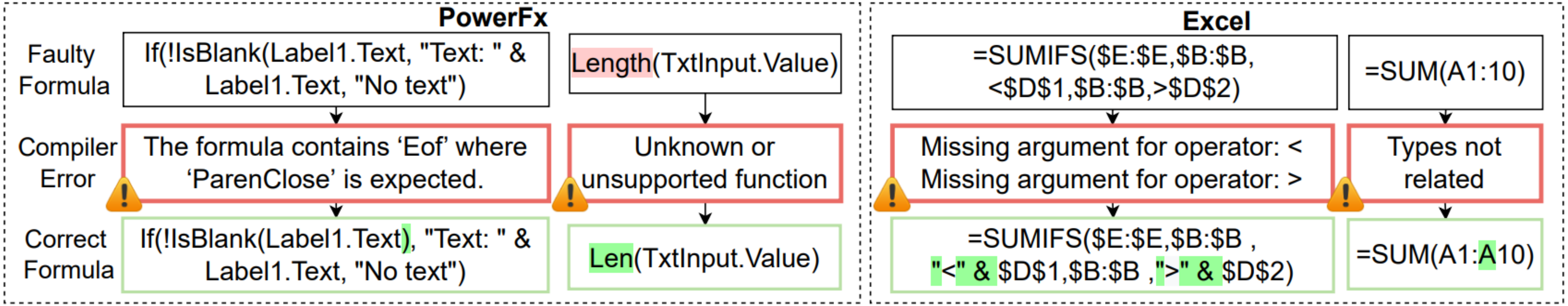
Fixing Programs

Last-Mile Errors: Syntax++

- Wide range of spectrum of errors
  - from simple, e.g., syntax errors, to complex, e.g., concurrency bugs
- We call errors that require few edits to fix, **Last Mile errors**.
- They are hard for low-code users to even identify them.



Unhelpful compiler messages



Approaches to Last-Mile Repair

Symbolic

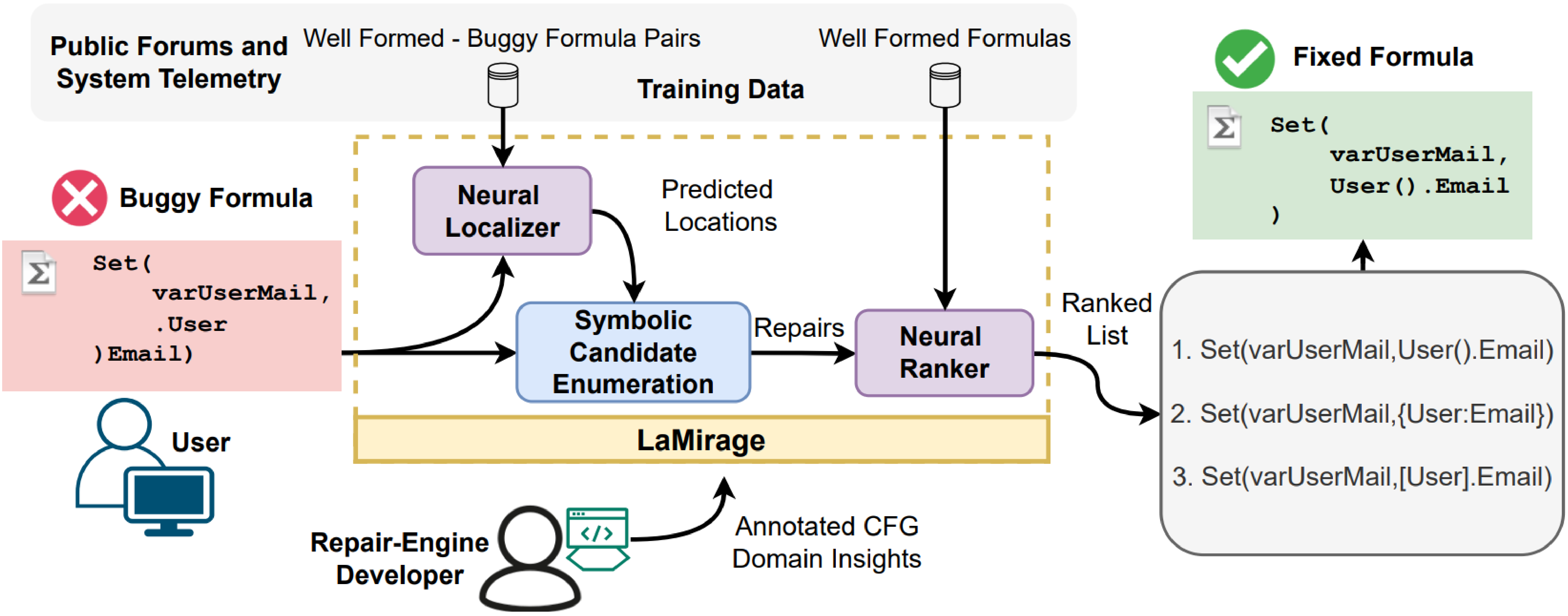
Neurosymbolic

Neural

LaMirage

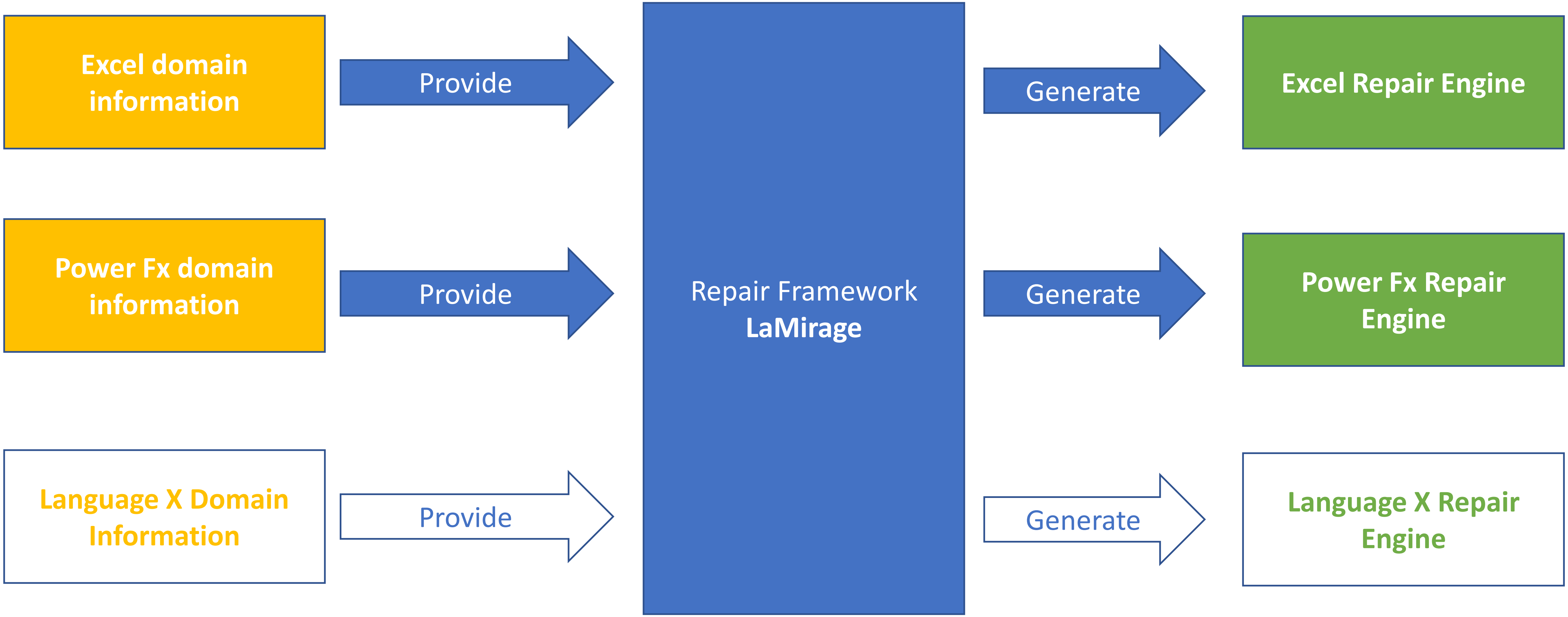
https://aka.ms/lamirage-arxiv

LaMirage: LAst-MIle RepAir-engine GEnerator



We should also avoid repeating work!

- Implement as a repair engine **generator**



Performance

- Neural methods are better than error recovery parser.
- LaMirage, a neurosymbolic method, outperforms neural models
- Performance degradation for neural models in PowerFx

System	Type	Excel (200 benchmarks)				Power Fx (200 benchmarks)			
		Top-1	Top-3	Top-5	Time (ms)	Top-1	Top-3	Top-5	Time (ms)
Excel Desktop	Symbolic	83	83	83	-	-	-	-	-
GRMTOOLS	Symbolic	97	104	108	**13.6**	98	110	113	**17.2**
BIFI	Neural	115	130	134	363.1	34	45	48	592.8
CODEX	Neural	111	156	160	1651.8	86	117	132	1997.9
CODEX-EDIT	Neural	147	163	165	5806.6	106	137	140	6417.6
LAMIRAGE	**Neurosymbolic**	**174**	**182**	**182**	32.1	**170**	**177**	**177**	134.4

RING

https://aka.ms/ring-paper

Domain-Specific Repair Engines

- Symbolic: substantial engineering for new domain
- Neural: need new data and retraining for new domain
- Neurosymbolic: both challenges mitigated but still there

New languages pose a significant investment

LaMirage Framework (Neurosymbolic)	BIFI (Neural)	Dr. Repair (Neural)	Tfix (Neural)	?	?

Large Language Models Trained on Code (LLMC)

Can we use an LLMC(e.g., Codex) to repair programs in all these domains?

Yes, and we'll show how with RING

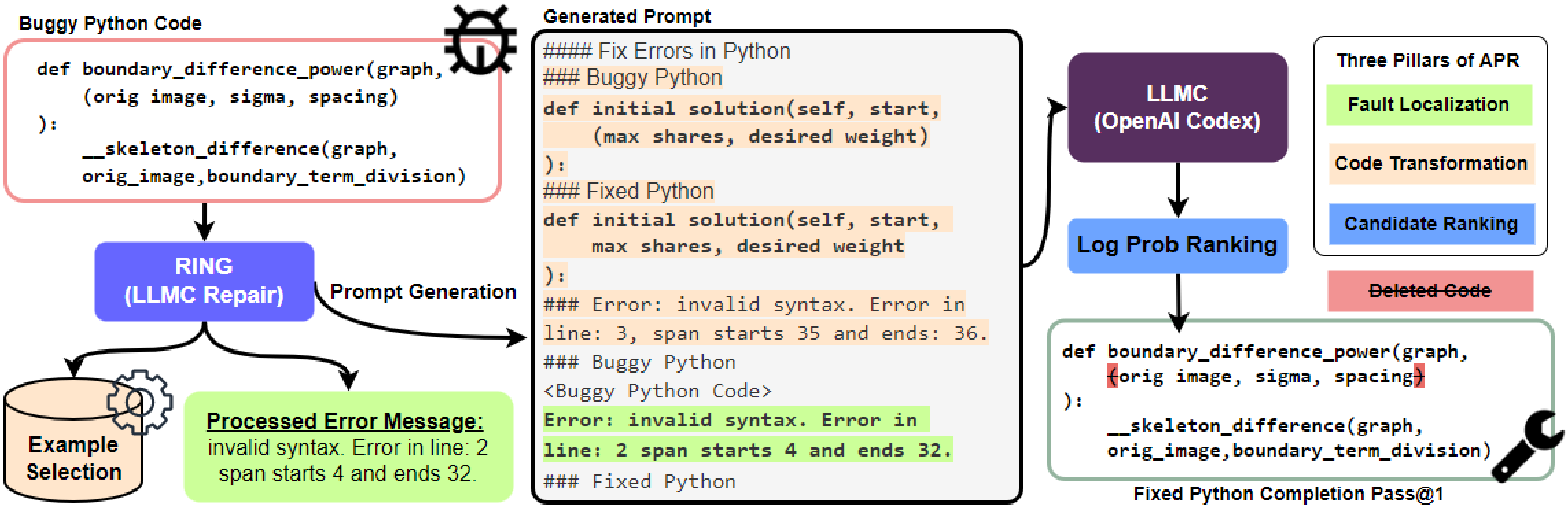
JS

?

RING: Multilingual Program Repair with LLMs

OpenAI

RING



RING: Repair Is Nearly Generation

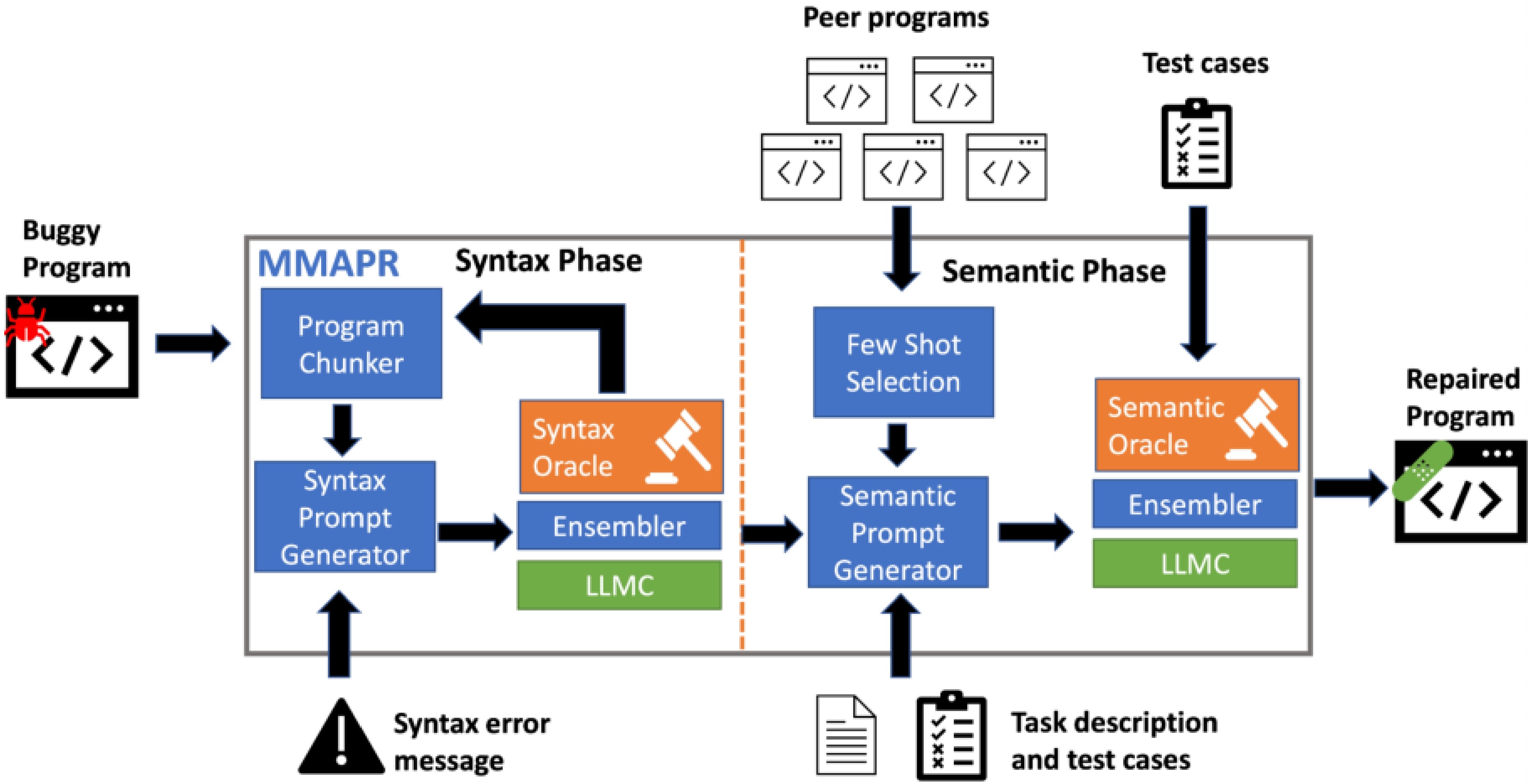
RING Results

Language	Approach	Top@1	Top@3	Top@50*	Metric	Avg. Tokens
Excel	RING (Abstracted Message, Error Vector)	**0.82**	**0.89**	**0.92**	Exact Match	26 ±14
	LaMirage (Bavishi et al. 2022)	0.71	0.76	-		
	Codex (Chen et al. 2021)	0.60	0.77	0.88		
Power Fx	RING (Compiler Message, Message Embedding)	0.71	0.85	**0.87**	Exact Match	29 ±19
	LaMirage (Bavishi et al. 2022)	**0.85**	**0.88**	-		
	Codex (Chen et al. 2021)	0.47	0.68	0.84		
Javascript	RING (Compiler Message, Error Vector)	0.46	0.59	**0.64**	Exact Match	163 ±106
	TFix (extended code snippets) (Berabi et al. 2021)	0.09	-	-		
	TFix (original dataset) (Berabi et al. 2021)	**0.59**	-	-		
	Codex (Chen et al. 2021)	0.19	0.28	0.39		
Python	RING (Compiler Message, Message Embedding)	**0.94**	**0.97**	0.97	Passes Parser Edit Distance < 5	104 ±150
	BIFI (Yasunaga and Liang 2021)	0.92	0.95	0.96		
	Codex (Chen et al. 2021)	0.87	0.94	**0.98**		
C	RING (Compiler Message, Message Embedding)	**0.63**	**0.69**	**0.70**	Passes Parser Edit Distance < 5	223 ±72
	Dr Repair (Yasunaga and Liang 2020)	0.55	-	-		
	Codex (Chen et al. 2021)	0.40	0.56	0.61		
Powershell	RING (Compiler Message, Message Embedding)	**0.18**	**0.25**	**0.28**	Exact Match	24 ±30
	Codex (Chen et al. 2021)	0.10	0.15	0.18		

PyDex: Fixing Intro Programming Assignments

https://arxiv.org/abs/2209.14876

PyDex



Peer programs
Test cases
Buggy Program
MMAPR
Syntax Phase
Semantic Phase
Program Chunker
Syntax Oracle
Syntax Prompt Generator
Ensembler
LLMC
Few Shot Selection
Semantic Prompt Generator
Semantic Oracle
Ensembler
LLMC
Repaired Program
Syntax error message
Task description and test cases

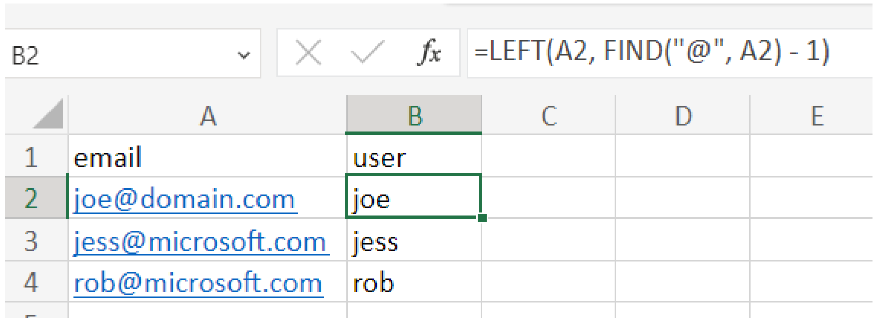
PyDex Results

Method		PyDex (without few-shot)		PyDex (with few-shot)		BIFI + Refactory		PyDex(syntax) + Refactory		PyDex(syntax) + GenProg	
ID	# Sub	RR (%)	Mean TED (SD)	RR (%)	Mean TED (SD)	RR (%)	Mean TED (SD)	RR (%)	Mean TED (SD)	RR (%)	Mean TED (SD)
2865	11	100.00	6.45 (4.74)	100.00	6.45 (4.74)	100.00	16.45 (7.00)	100.00	20.55 (6.08)	90.91	16.10 (6.08)
2868	28	85.71	8.79 (8.94)	100.00	8.64 (8.49)	82.14	36.35 (19.26)	96.43	35.15 (19.24)	96.43	26.00 (9.06)
2869	23	95.65	16.68 (18.47)	100.00	10.30 (10.99)	69.57	47.75 (20.27)	100.00	42.35 (19.77)	30.43	20.29 (12.85)
2870	27	74.07	10.00 (13.33)	100.00	15.00 (19.35)	85.19	39.48 (31.38)	92.59	35.72 (31.78)	33.33	20.22 (21.35)
2872	18	100.00	8.33 (15.15)	100.00	7.39 (13.01)	72.22	105.08 (34.58)	100.00	103.06 (38.65)	88.89	15.94 (6.56)
2873	32	78.13	12.00 (16.18)	90.63	12.93 (15.47)	84.38	75.00 (19.75)	100.00	71.41 (20.37)	25.00	18.63 (5.48)
2874	16	100.00	9.56 (12.50)	100.00	8.50 (11.76)	87.50	35.79 (18.63)	100.00	38.94 (31.43)	75.00	15.83 (5.48)
2875	23	86.96	14.75 (19.97)	100.00	11.52 (12.52)	78.26	63.22 (28.97)	100.00	58.65 (28.55)	47.83	17.09 (7.94)
2877	21	100.00	9.71 (16.82)	100.00	9.14 (16.79)	80.95	67.47 (27.87)	100.00	57.95 (32.19)	85.71	19.44 (11.49)
2878	25	100.00	37.00 (60.16)	100.00	36.32 (59.53)	68.00	138.18 (44.17)	88.00	167.50 (66.11)	52.00	21.46 (15.49)
2879	21	76.19	131.19 (51.62)	85.71	132.78 (52.61)	52.38	183.45 (40.90)	71.43	195.33 (55.24)	4.76	229.00 (N/A)
2882	23	60.87	90.64 (71.76)	91.30	106.57 (77.57)	0.00	N/A	0.0	N/A	17.39	42.00 (18.30)
2883	5	100.00	17.40 (14.67)	100.00	17.40 (14.67)	40.00	141.00 (8.49)	100.00	103.60 (39.37)	60.00	46.00 (19.47)
2920	10	80.00	84.38 (67.62)	80.00	53.50 (66.05)	0.00	N/A	10.00	69.00 (N/A)	20.00	42.00 (5.66)
2921	3	100.00	28.00 (3.61)	100.00	28.00 (3.61)	0.00	N/A	0.0	N/A	0.0	N/A
Overall		86.71	28.59	96.50	29.68	67.13	70.39	83.57	73.53	49.30	22.82

FLAME

https://aka.ms/flame-arxiv

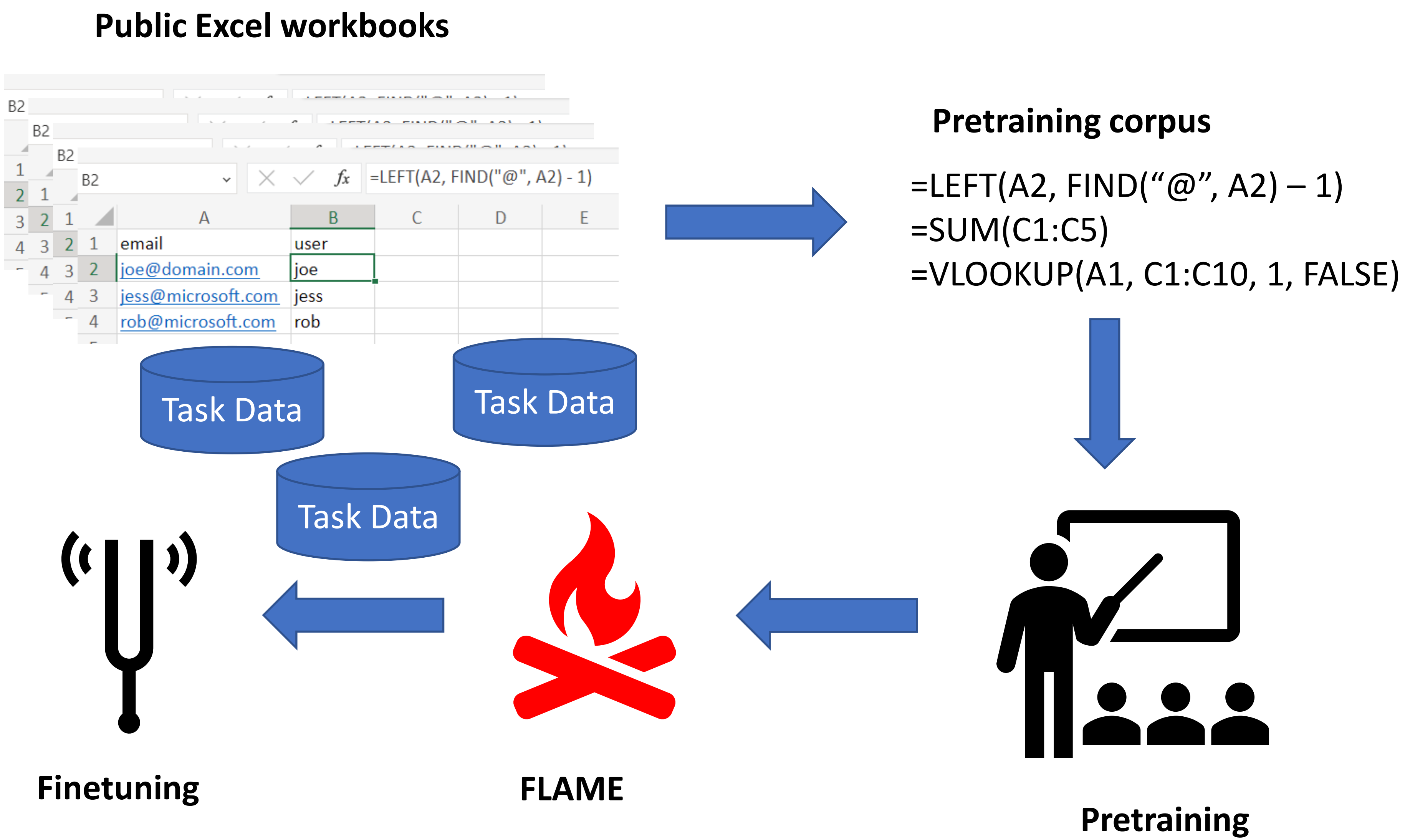
Why a domain-specific model for formulas?



- Up to billions of parameters, trained on GBs of code
- Costly to train and deploy
- General purpose programming languages – quite different from Excel formulas

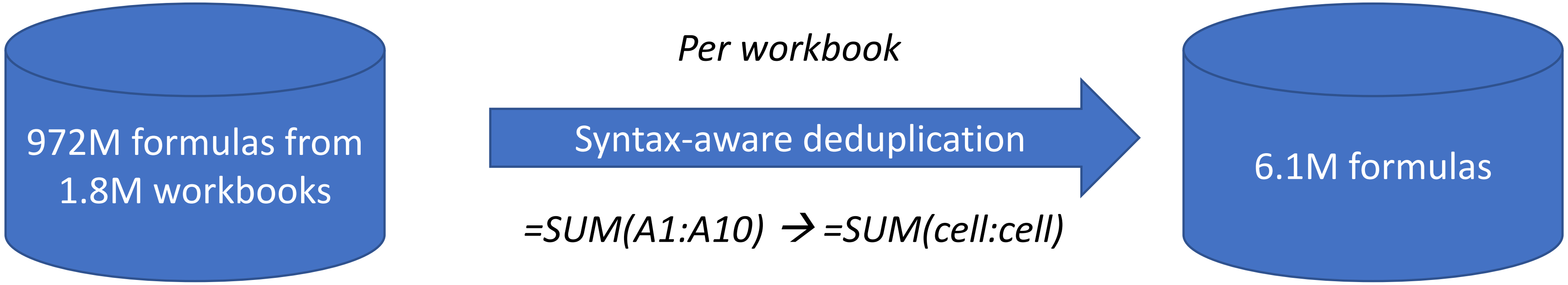
- 60M parameters, trained on 540MB of formulas
- Cheaper to train and deploy
- Tailored to Excel formula language

FLAME Overview



Domain-Specific Data Curation and Tokenization

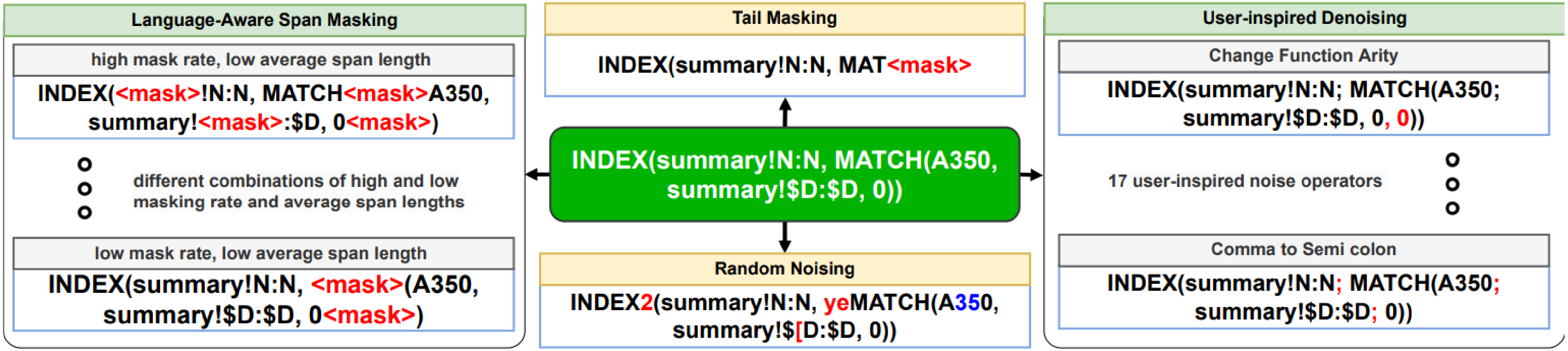
Curation



Tokenization

=SUMIF(B1:B5, "Not available", A1:A5) → = sumif (b 1 : b 5 , ␣ " not ␣ available " , ␣ a 1 : a 5)

Domain-Specific Pretraining

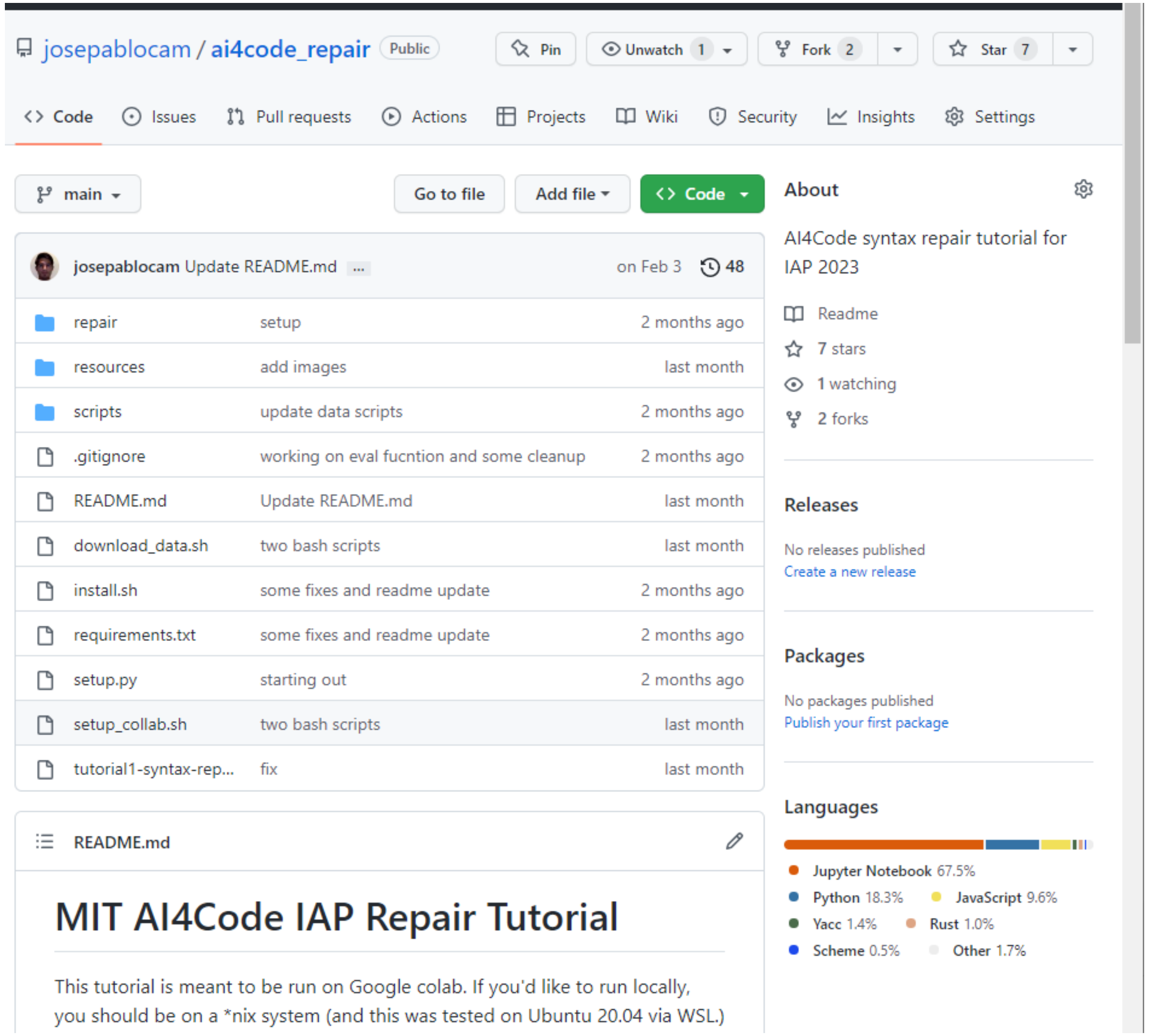


FLAME Results (small snapshot...)

Model	Last Mile Repair				Syntax Reconstuction			
	Forum		Test		Forum		Test	
	T@1	T@5	T@1	T@5	T@1	T@5	T@1	T@5
Cushman	**0.79**	0.88	**0.87**	**0.93**	0.70	0.80	<u>0.84</u>	**0.91**
Davinci (FS)	0.76	<u>0.89</u>	0.54	0.77	0.62	0.77	0.61	0.73
CodeT5 (220M)	0.70	0.84	<u>0.84</u>	0.90	<u>0.70</u>	<u>0.84</u>	0.82	0.89
CodeT5 (60M)	0.72	0.83	0.82	0.89	0.65	0.81	0.83	0.89
FLAME	<u>0.76</u>	**0.89**	0.83	<u>0.91</u>	**0.75**	**0.89**	**0.84**	<u>0.89</u>

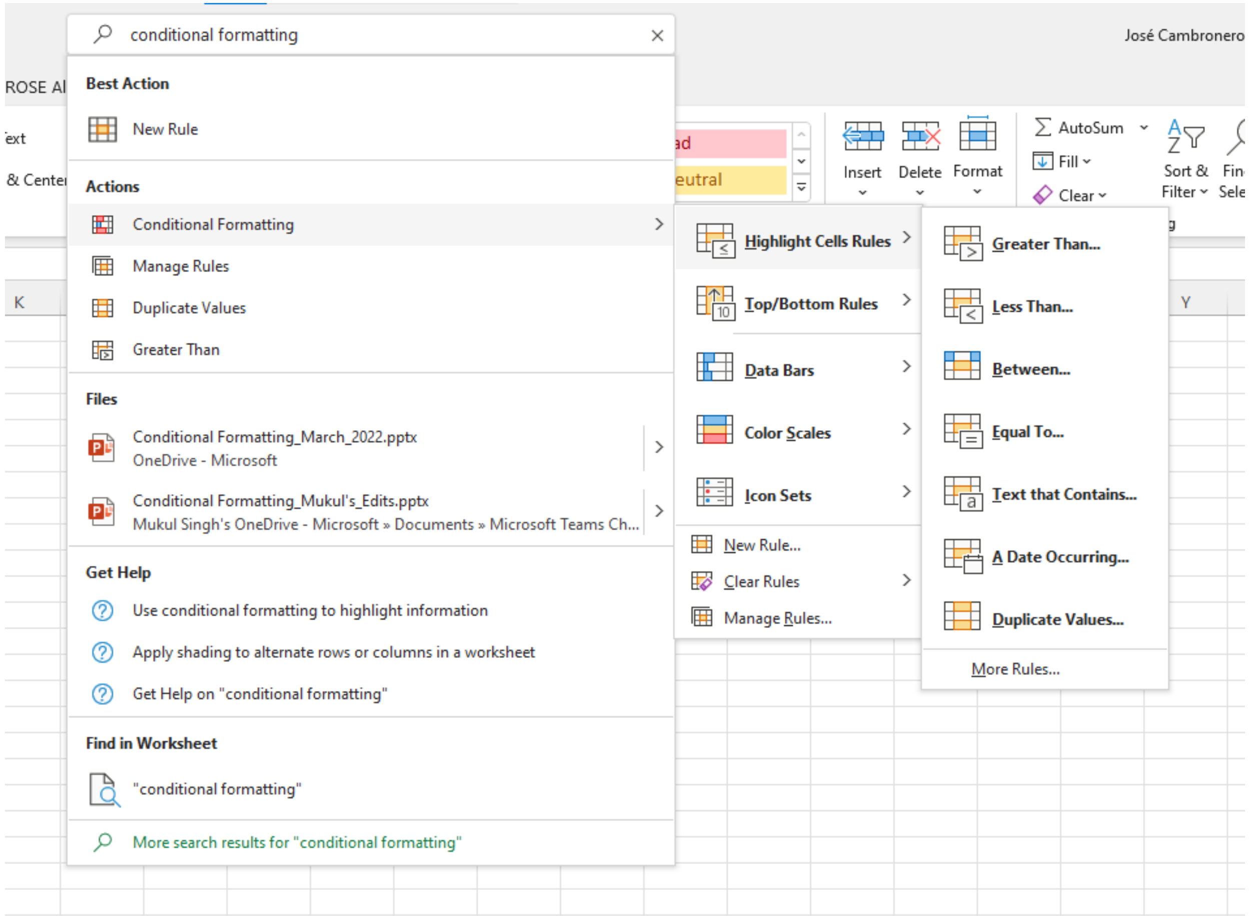
Continue playing with LMR

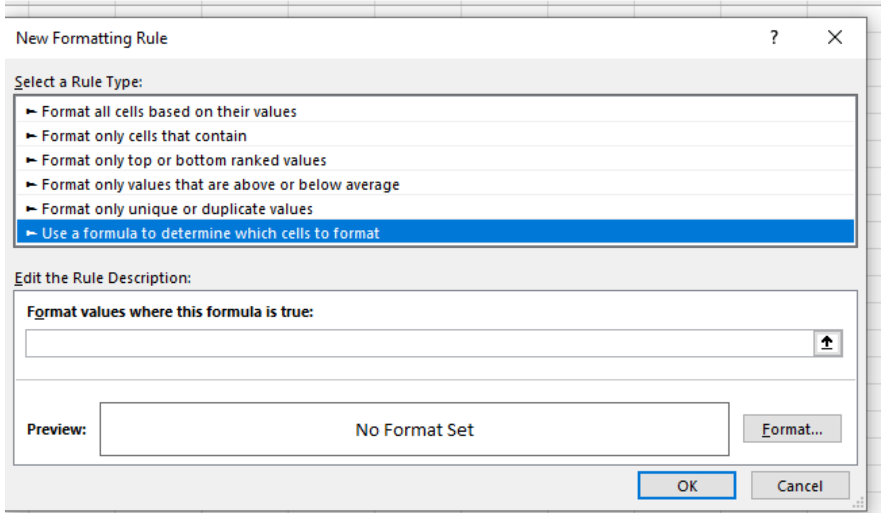
https://aka.ms/lmr-tutorial



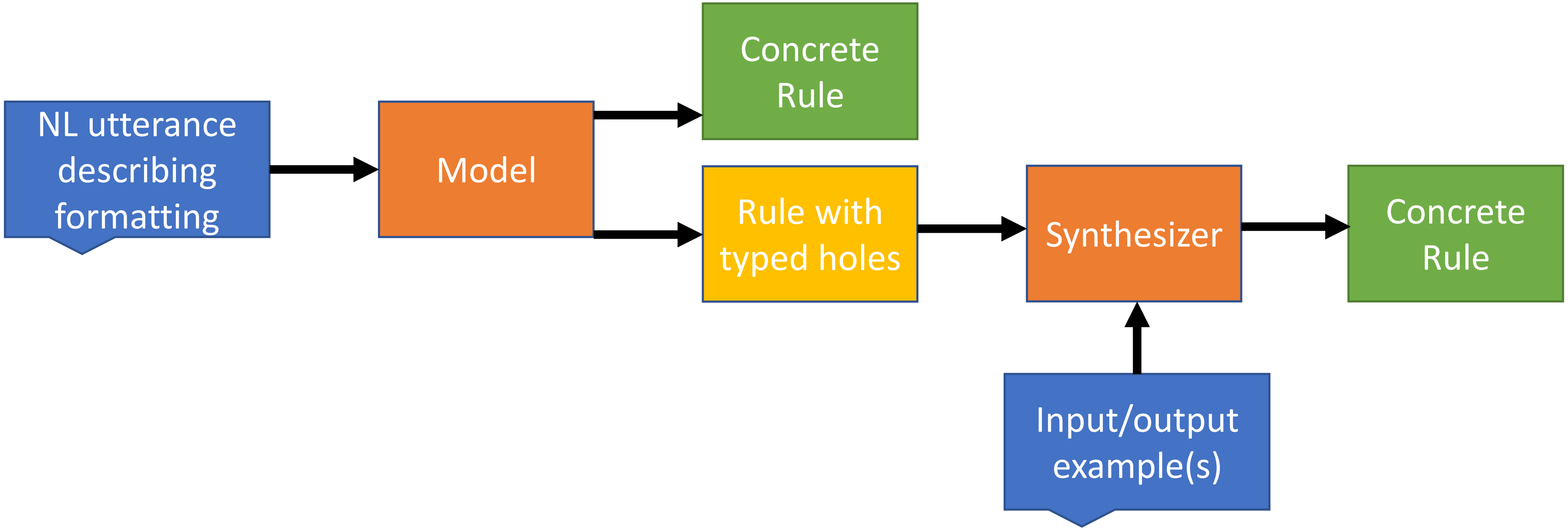
Domain Specific Synthesis

Domain-Specific Tools: An Opportunity

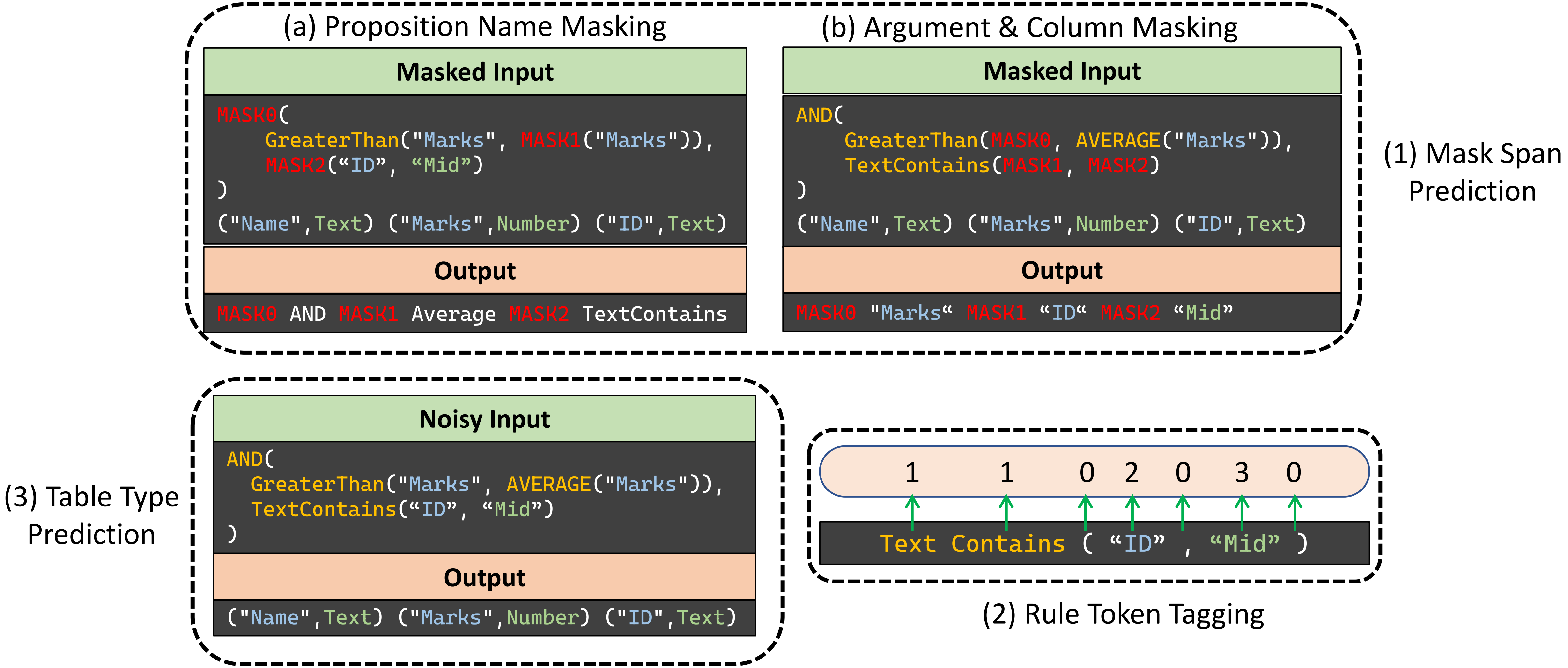




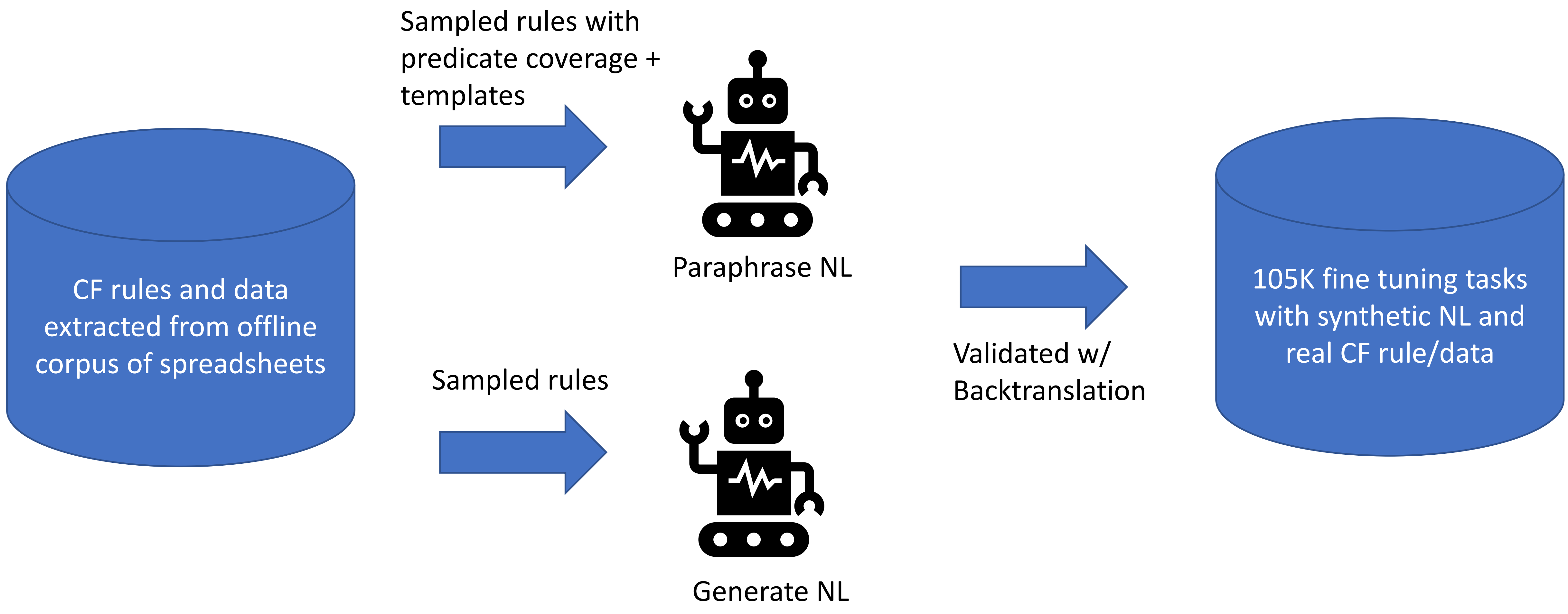
FormaT5: Multimodal Synthesis for Conditional Formatting Rules



Key Idea 1: Pretraining on Rule+Data Corpus



Key Idea 2: Fine-tune on (synthetic) task data



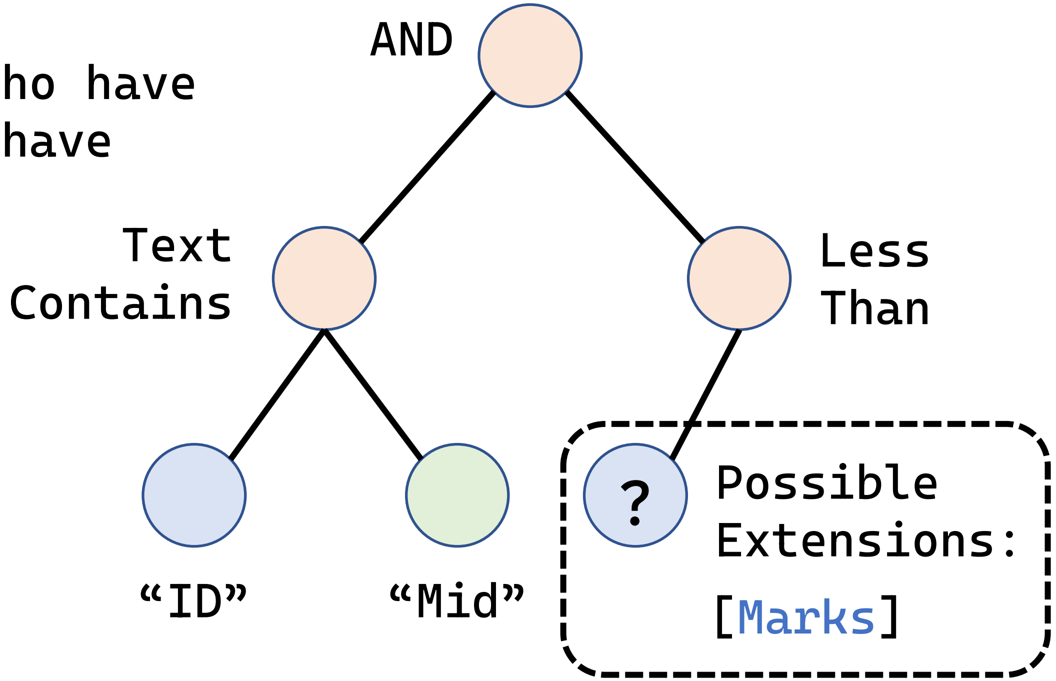
Key Idea 3: Constrained Decoding

Query:

“Highlight Students who have Middle School ID and have scored above Average”

Table Schema:

(Name, Text)
(Marks, Number)
(ID, Text)



Key Idea 4: Abstain when unsure and fill inductively

```
Query:
"You have a list of final year students.
Highlight the students who got clearance
from all departments"
```

Table:

Students	Library	Sports	Lab
Student 1	Yes	No	Yes
Student 2	Yes	Yes	Yes
Student 3	No	Yes	No
Student 4	No	Yes	Yes
Student 5	Yes	No	No

Rule Generation

```
Rule Sketch:
AND(TextEquals("Library",[HOLE]),
  TextEquals("Sports",[HOLE]),
  TextEquals("Lab",[HOLE]))
```

Value Filling

```
Predicted Rule:
AND(TextEquals("Library", "Yes"),
  TextEquals("Sports", "Yes"),
  TextEquals("Lab", "Yes"))
```

FormaT5 Results

Table 2: Comparison of FORMAT5 with baselines on the task of NL based rule generation. We report sketch (SM), exact (EM) and execution match (ExM) of the generated rules for the different task categories. "Model" column denotes the underlying base LLM used by the system. FORMAT5 outperforms all baselines in sketch, execution and exact rule match.

System description			Single			Extended			Multiple		
Method	**Model**	**Param**	**EM**	**SM**	**ExM**	**EM**	**SM**	**ExM**	**EM**	**SM**	**ExM**
T5	T5	770M	74.8	86.8	77.7	63.3	84.7	70.9	48.9	68.2	54.6
CodeT5	T5	770M	75.4	88.0	78.0	65.1	83.8	72.5	50.4	69.3	55.8
Codex	GPT 3	175B	68.5	82.1	70.8	62.9	79.7	70.4	45.1	62.4	50.8
PICARD	T5	770M	75.8	88.7	78.3	67.4	85.6	75.0	52.4	73.5	58.2
Synchromesh	GPT 3	175B	74.5	87.5	76.9	66.0	83.5	73.3	50.9	69.8	56.7
ValueNet	BERT	110M	71.4	79.9	73.9	60.4	74.8	67.6	42.4	63.2	47.9
TAPAS	BERT	110M	73.3	84.7	75.5	65.2	72.6	72.4	43.4	62.6	49.0
TaBERT	BERT	110M	69.6	81.5	72.6	61.6	70.9	68.5	41.8	60.5	47.3
FORMAT5	**T5**	770M	**78.2**	**90.7**	**81.2**	**70.7**	**86.2**	**78.0**	**58.6**	**79.5**	**64.3**

Open Discussion

Building Software in PROSE

- Pull Requests
  - At least 2 approvers
  - Automated checks for style, basic functionality
- Releases
  - Nightly builds with automated error reporting
  - Nightly performance tests (runtime)
- Testing
  - Hierarchical test suites (e.g. some run on PR build, some in release pipeline)
  - Crowd-sourced, synthetic, and manually constructed test cases
- Delivery
  - Azure DevOps Artifacts: npm package, nuget package
- Public Package: https://github.com/microsoft/prose (FlashFill-style framework)

Graduate School vs Industry

- Masters
- PhD
- Researcher
- Product-oriented roles: engineer, product manager, X?

Career Changes

- Personally:
  - Math major → Econ major → Researcher at big bank (mortgages/housing) → Masters in CS → PhD in CS → Industry research
- Uncommon path actually comes with benefits
  - Took some time for me to recognize this
  - Diversity of experiences/roles is a constant source of research ideas