UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Artifacts Available V1.1

Artifacts Evaluated Functional V1.1

Artifacts Evaluated Reusable V1.1

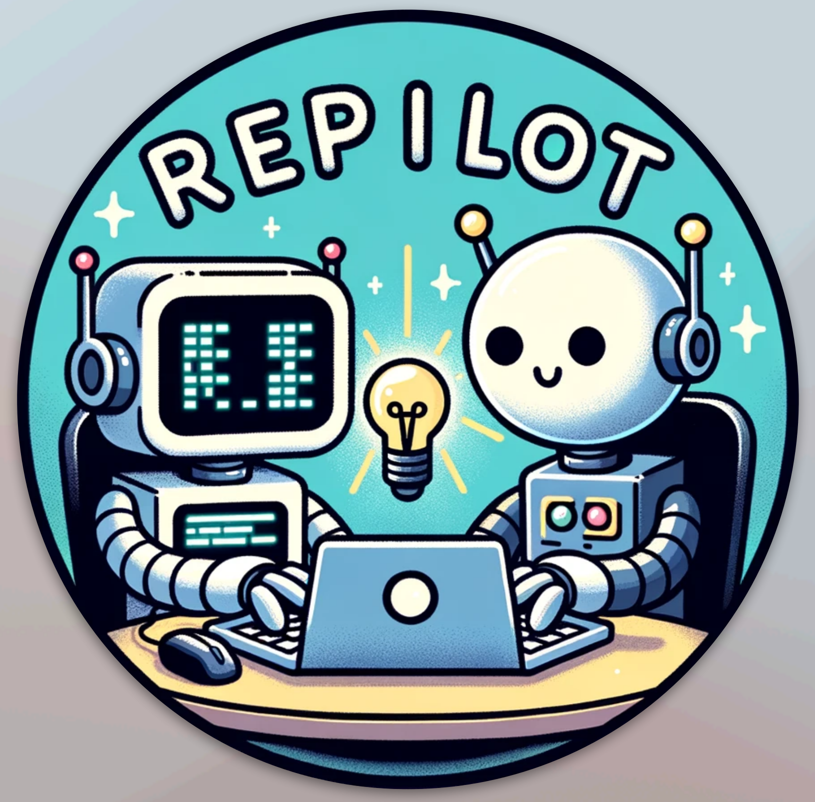
Copiloting the Copilots (Repilot)

Fusing Large Language Models with Completion Engines for Automated Program Repair

Yuxiang Wei
@YuxiangWei9

Chunqiu Steven Xia
@steven_xia_

Lingming Zhang
@LingmingZhang

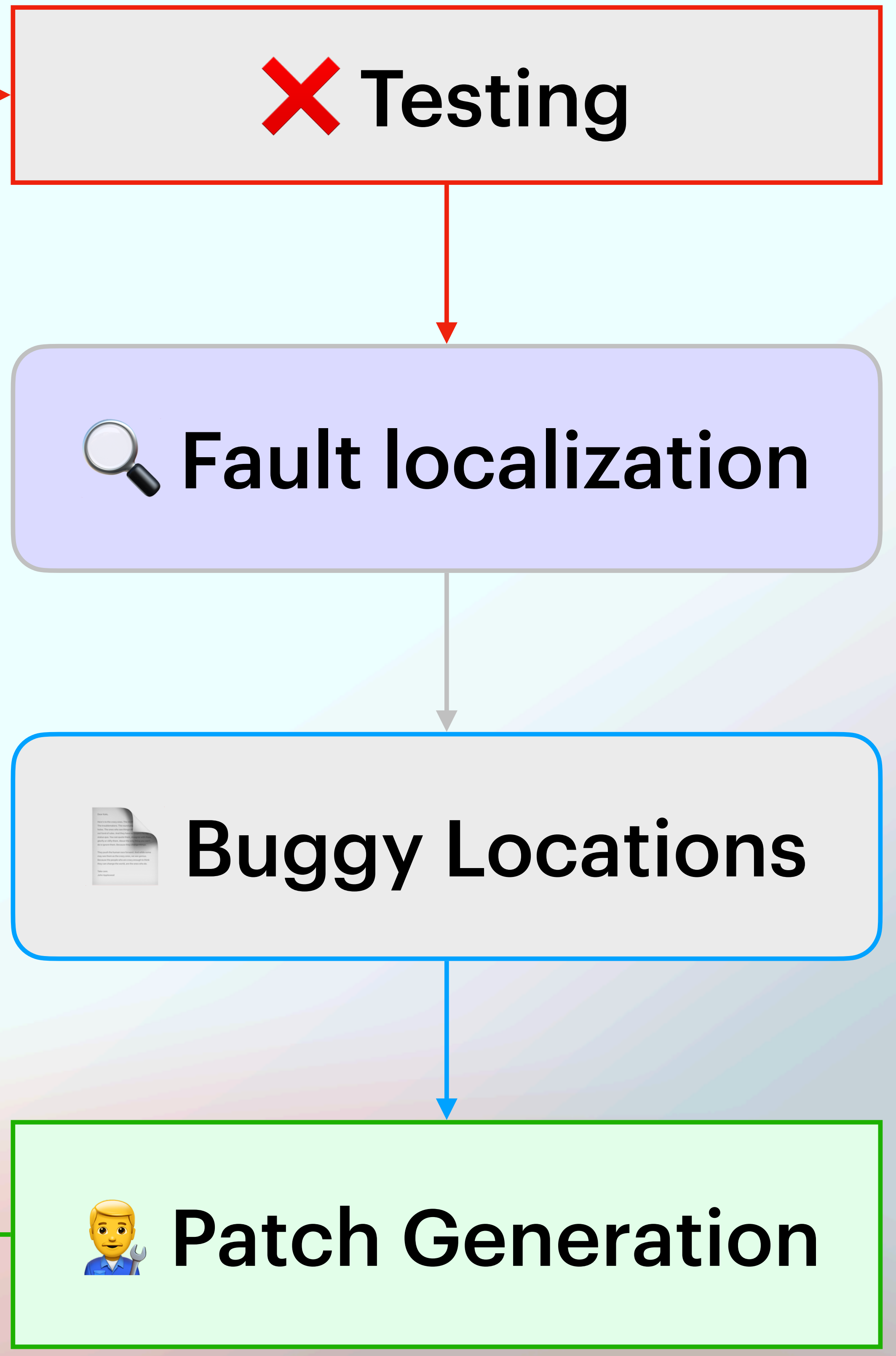


Powered by DALL·E 3

Automated Program Repair (APR)
The two stages

```java
private String getRemainingJSDocLine() {
   String result = stream.getRemainingJSDocLine();
   return result;
}
```

Buggy program (part of a project)




```diff
private String getRemainingJSDocLine() {
   String result = stream.getRemainingJSDocLine();
+  unreadToken = NO_UNREAD_TOKEN;
   return result;
}
```

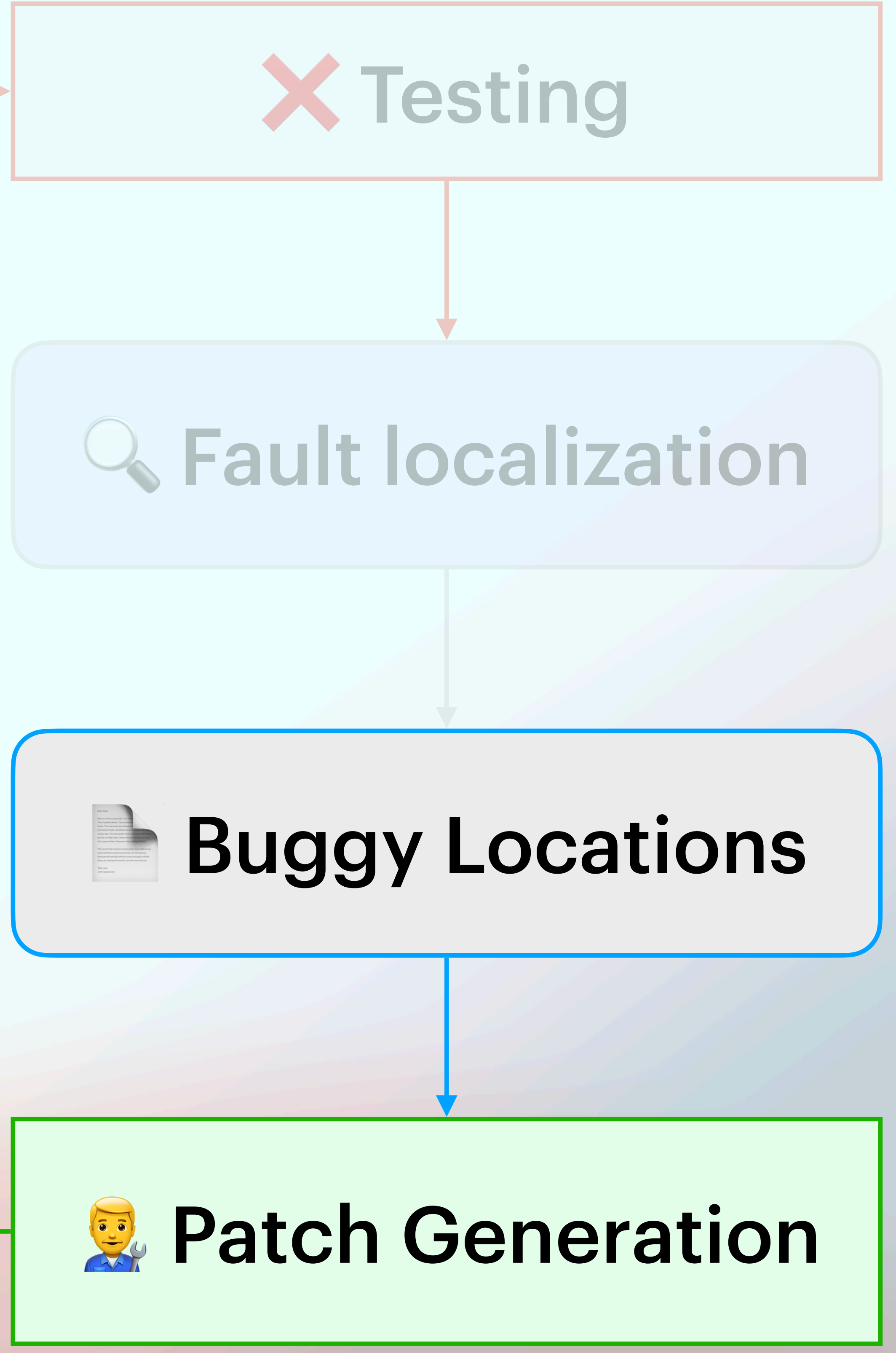
Candidate patches

Automated Program Repair (APR)

The two stages

```java
private String getRemainingJSDocLine() {
   String result = stream.getRemainingJSDocLine();
   return result;
}
```

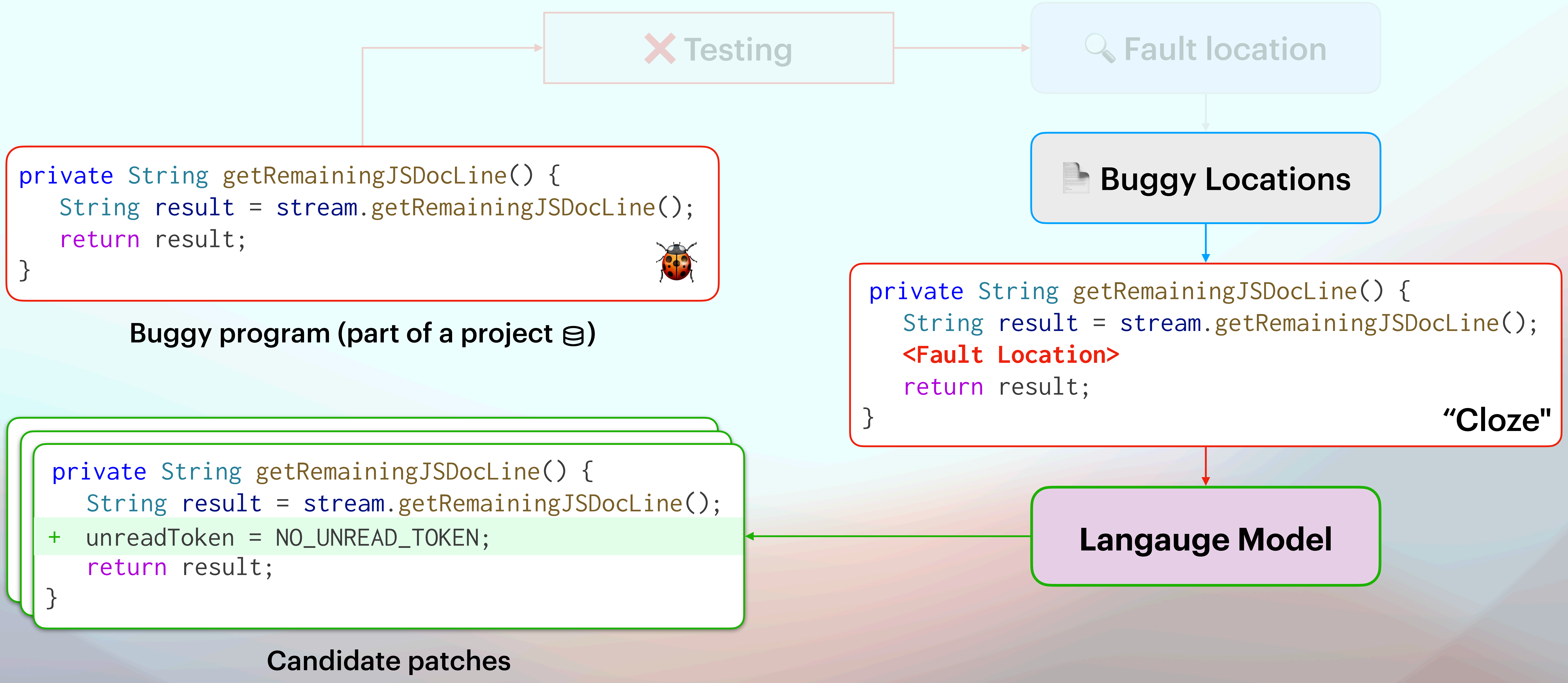
Buggy program (part of a project)




```diff
private String getRemainingJSDocLine() {
   String result = stream.getRemainingJSDocLine();
+  unreadToken = NO_UNREAD_TOKEN;
   return result;
}
```

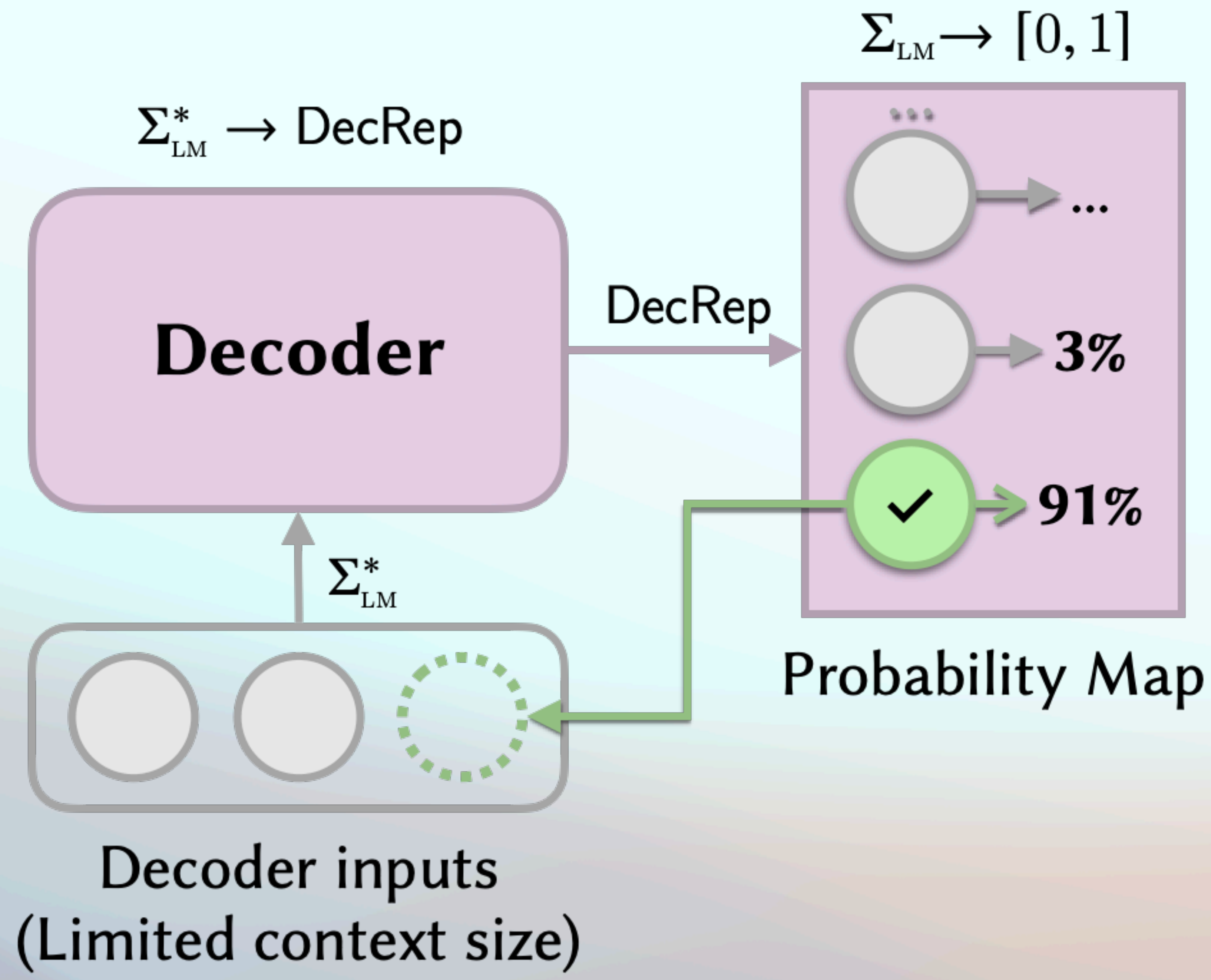
Candidate patches

Patch Generation with Large Language Models (LLMs)



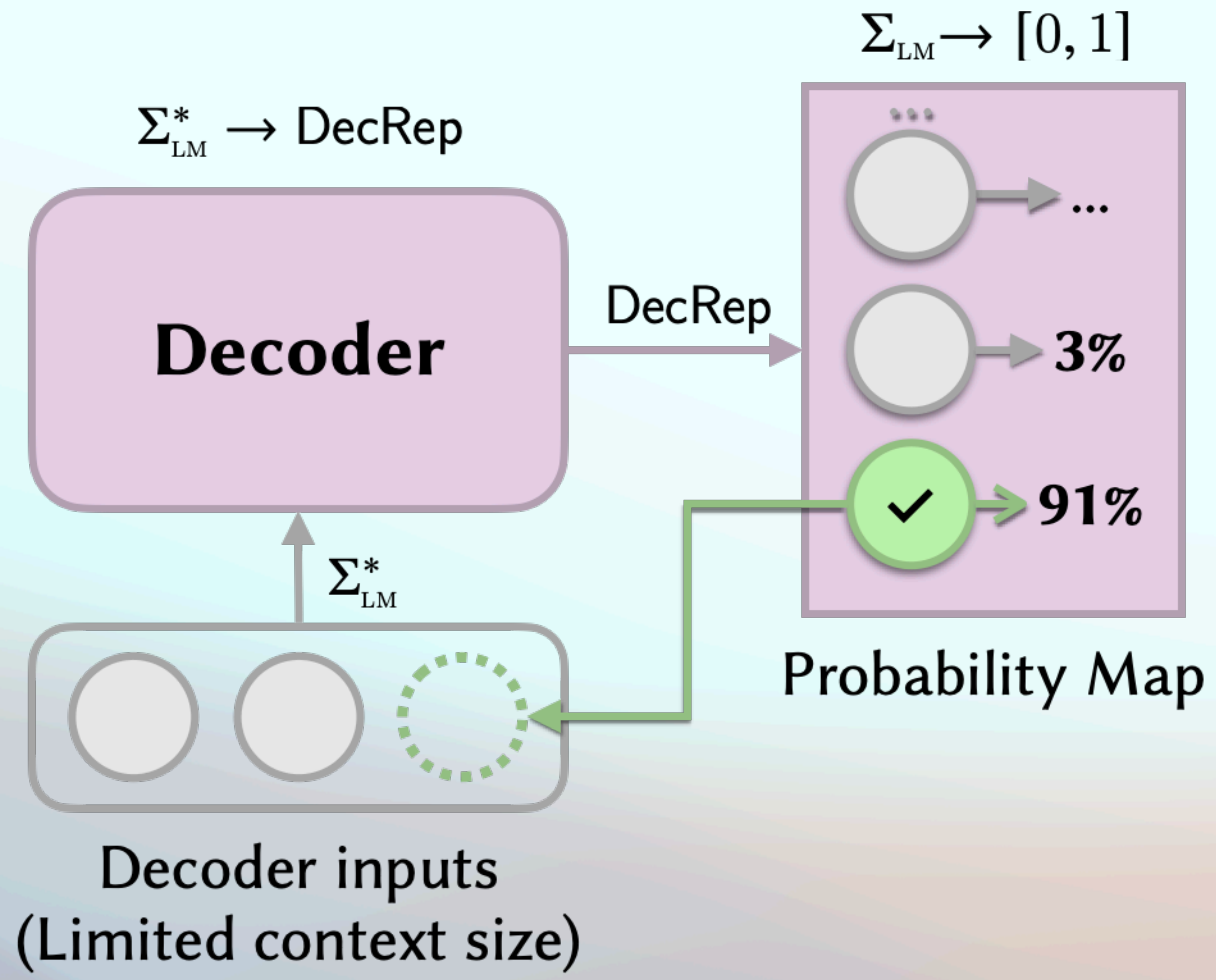
AlphaRepair: Less Training, More Repairing Please: Revisiting Automated Program Repair via Zero-shot Learning

LLMs vs. Autocompletion

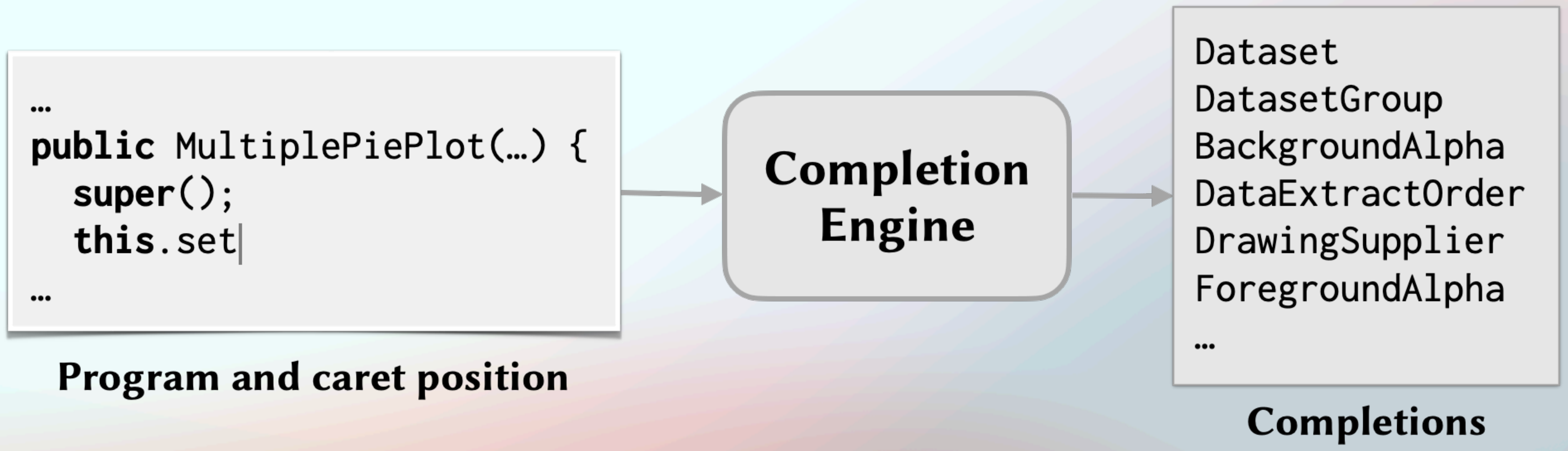


Code generation with LLMs

LLMs vs. Autocompletion



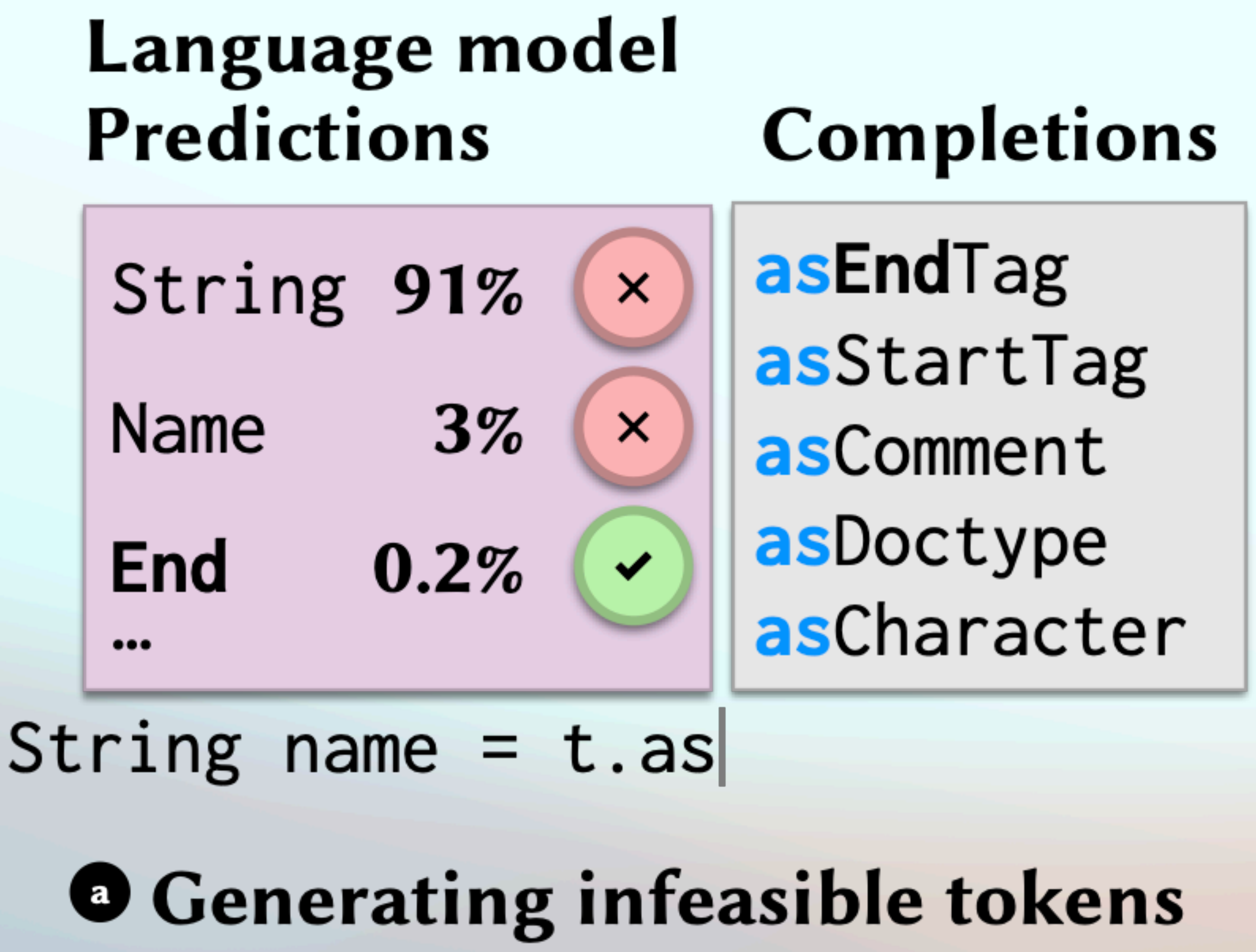
Code generation with LLMs



Semantics-based autocompletion

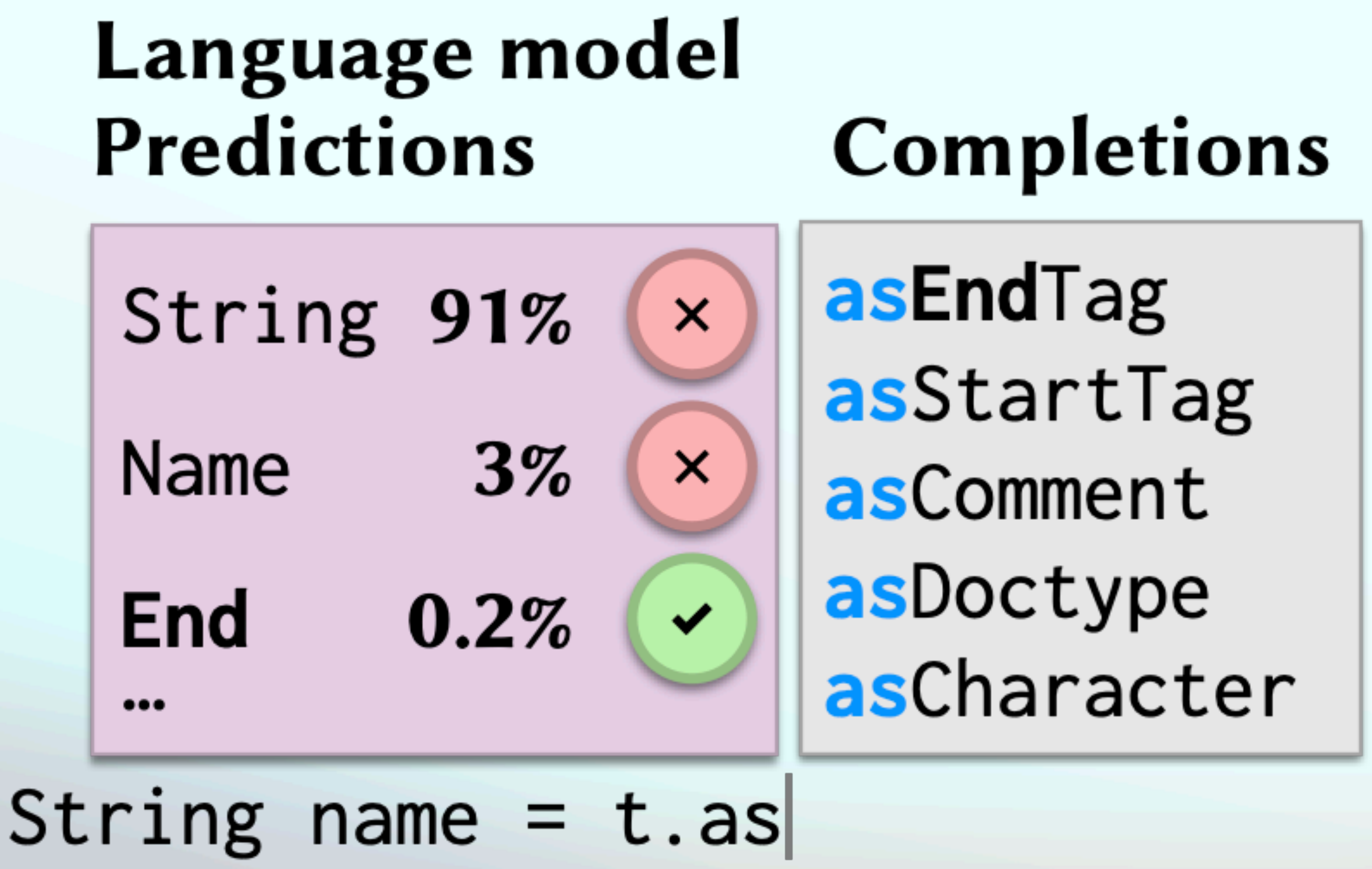
Problems with LLM-only code generation

The autoregressive (token-by-token) nature

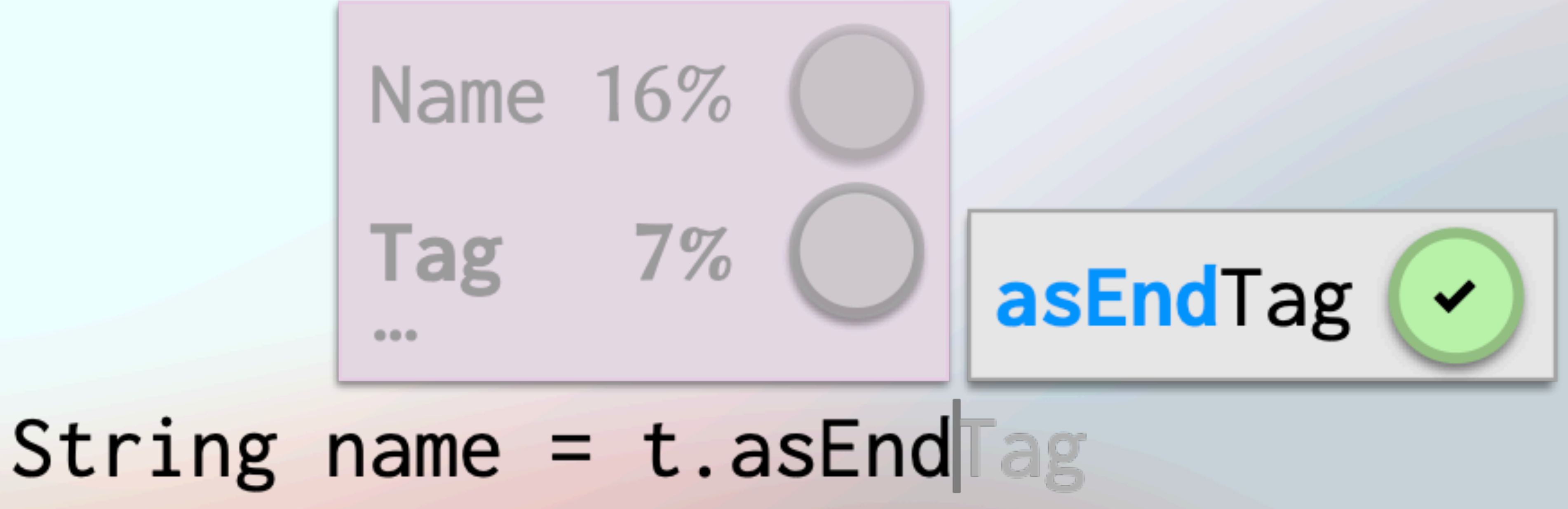


Problems with LLM-only code generation

The autoregressive (token-by-token) nature

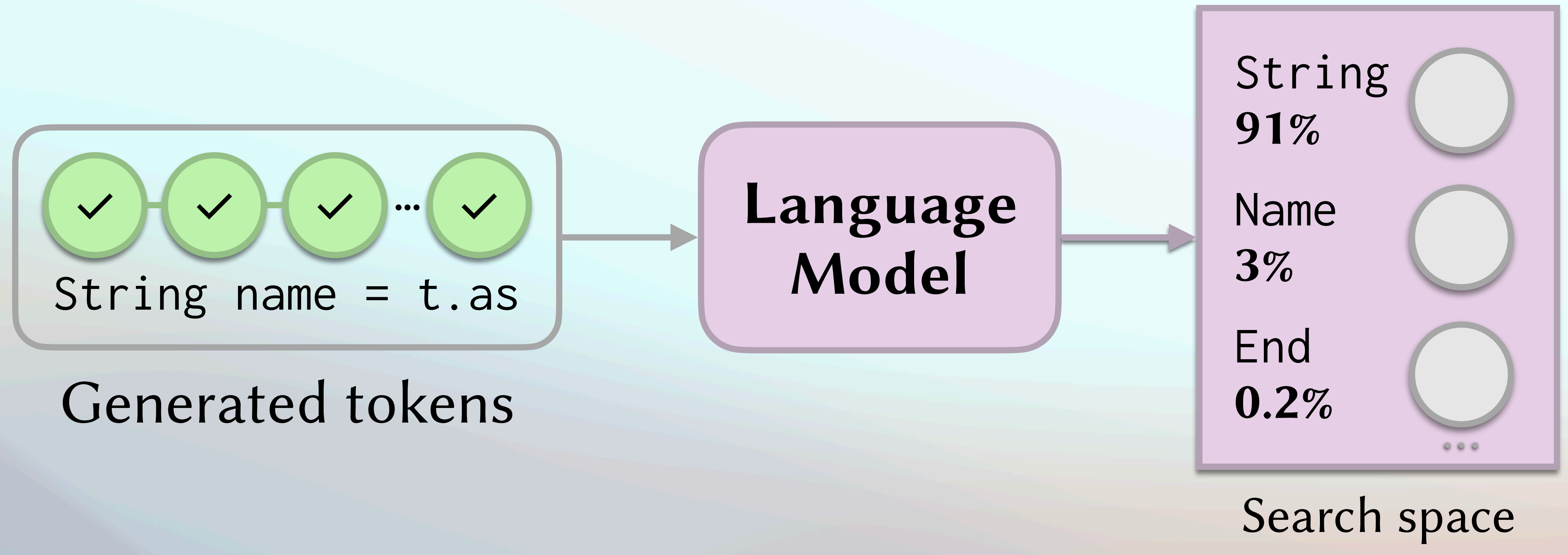


ⓐ Generating infeasible tokens

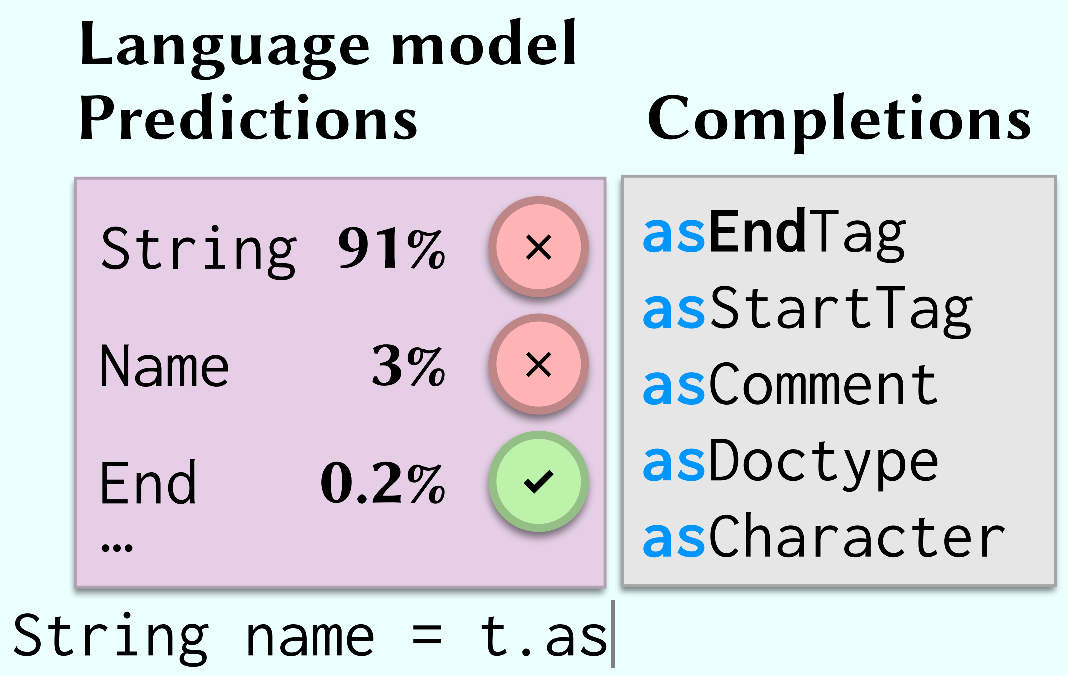


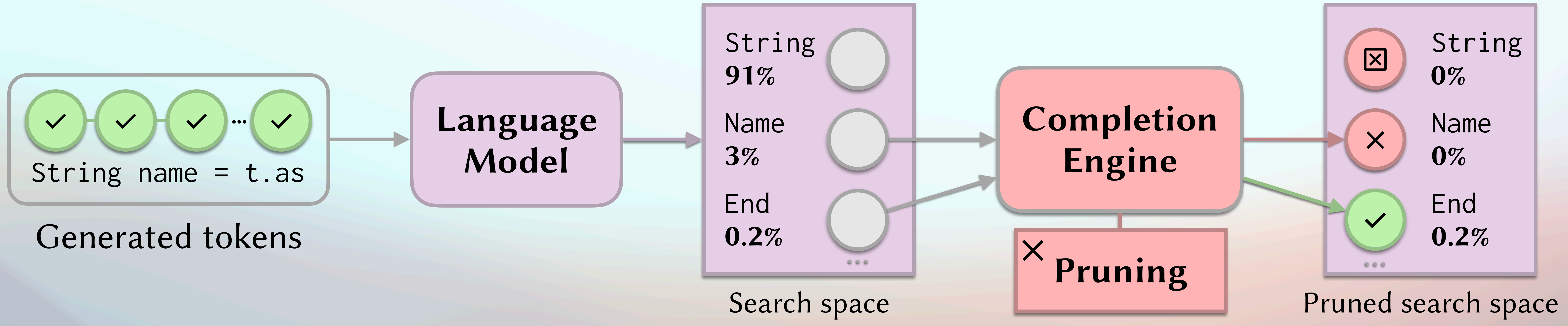
ⓑ Hard to generate rare tokens

How Repilot works

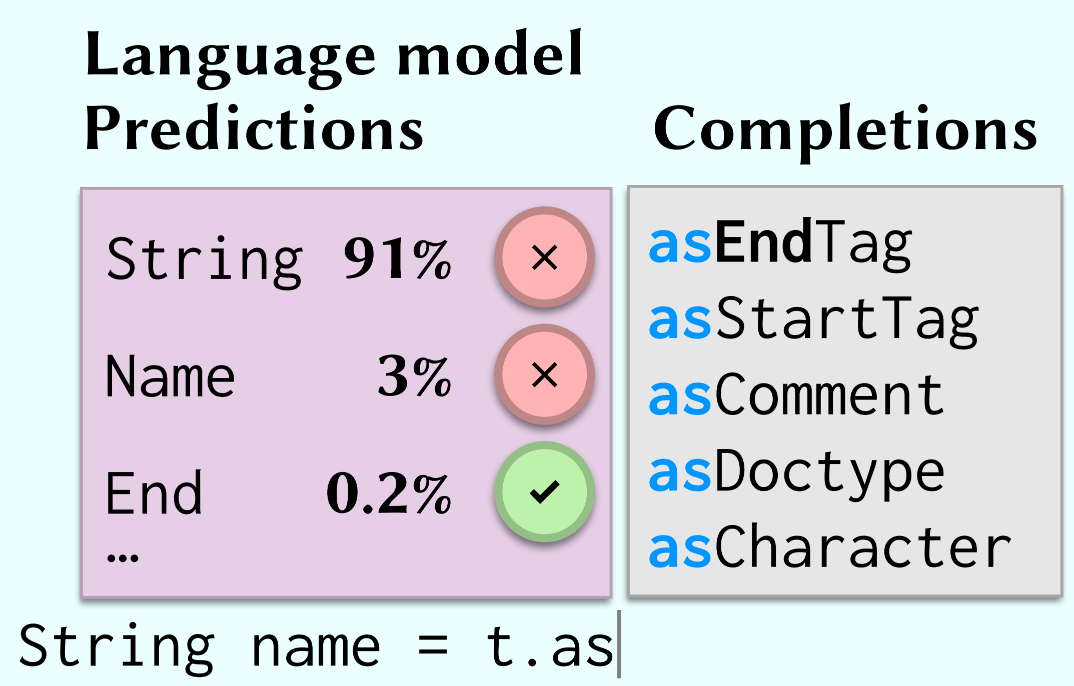


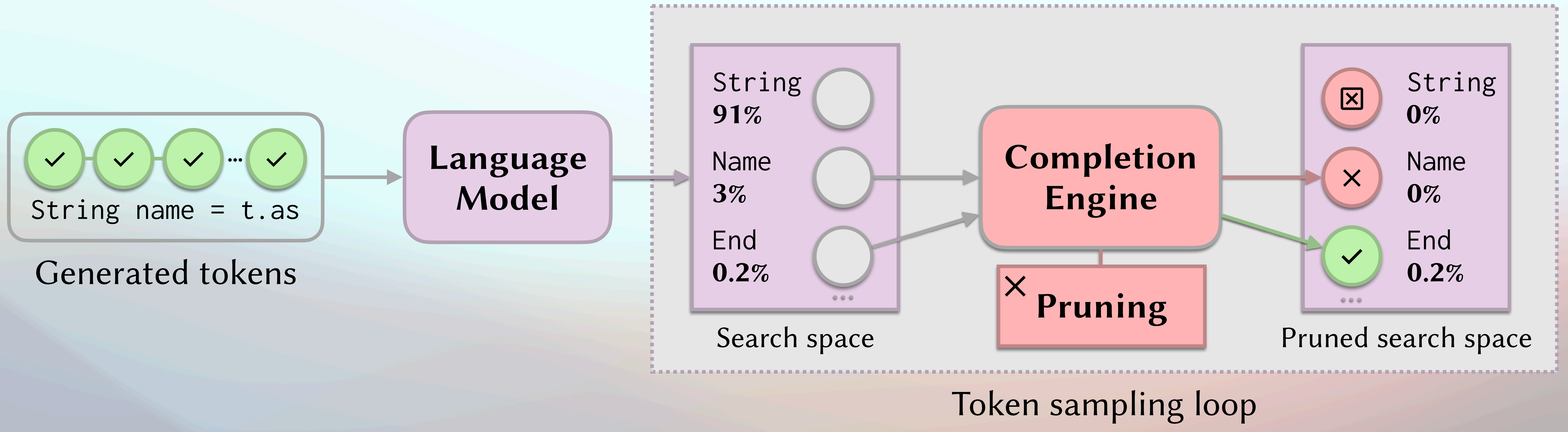
How Repilot works



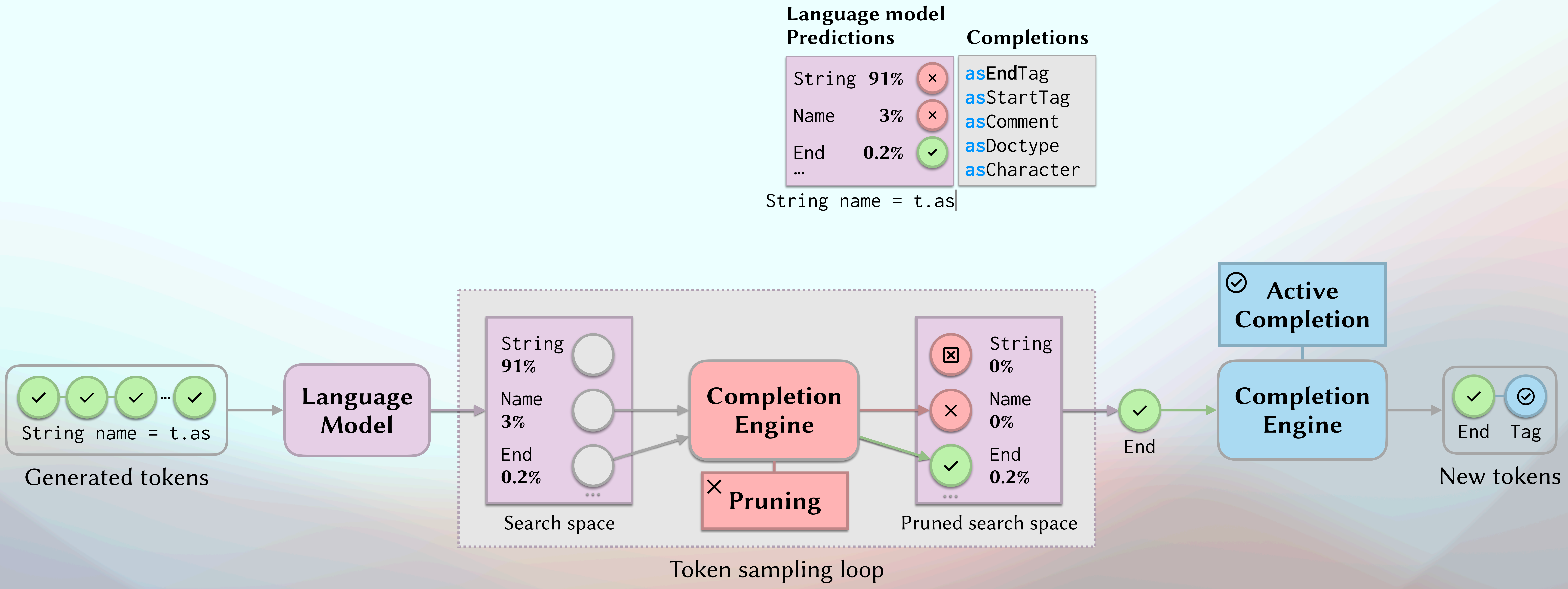


How Repilot works

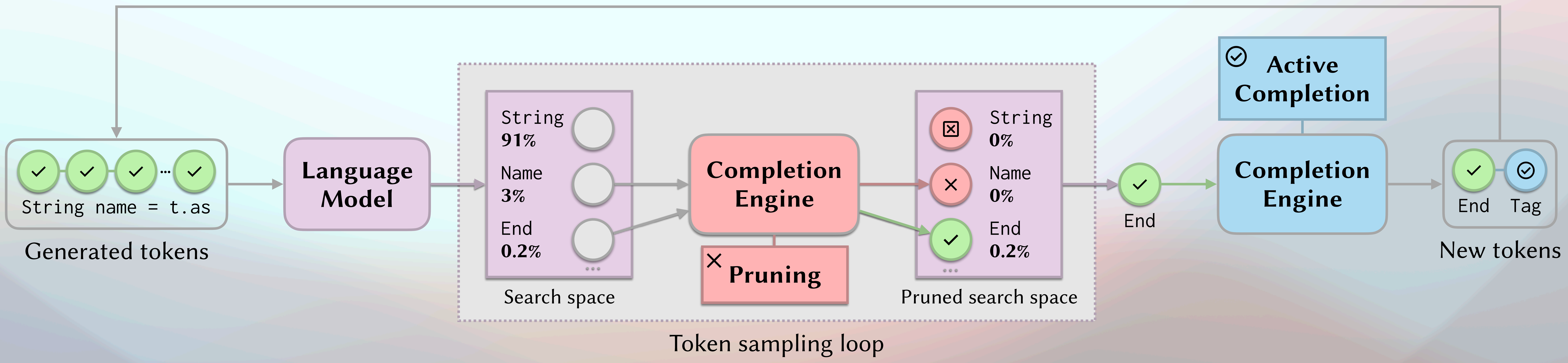




How Repilot works

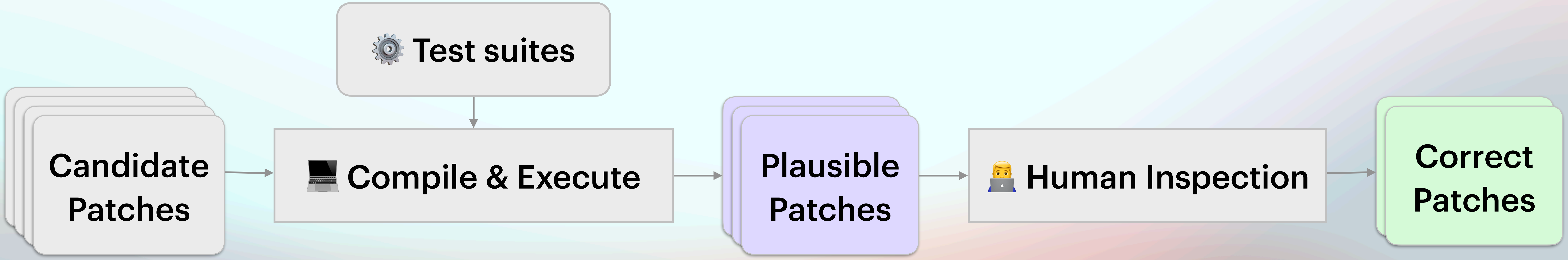


How Repilot works



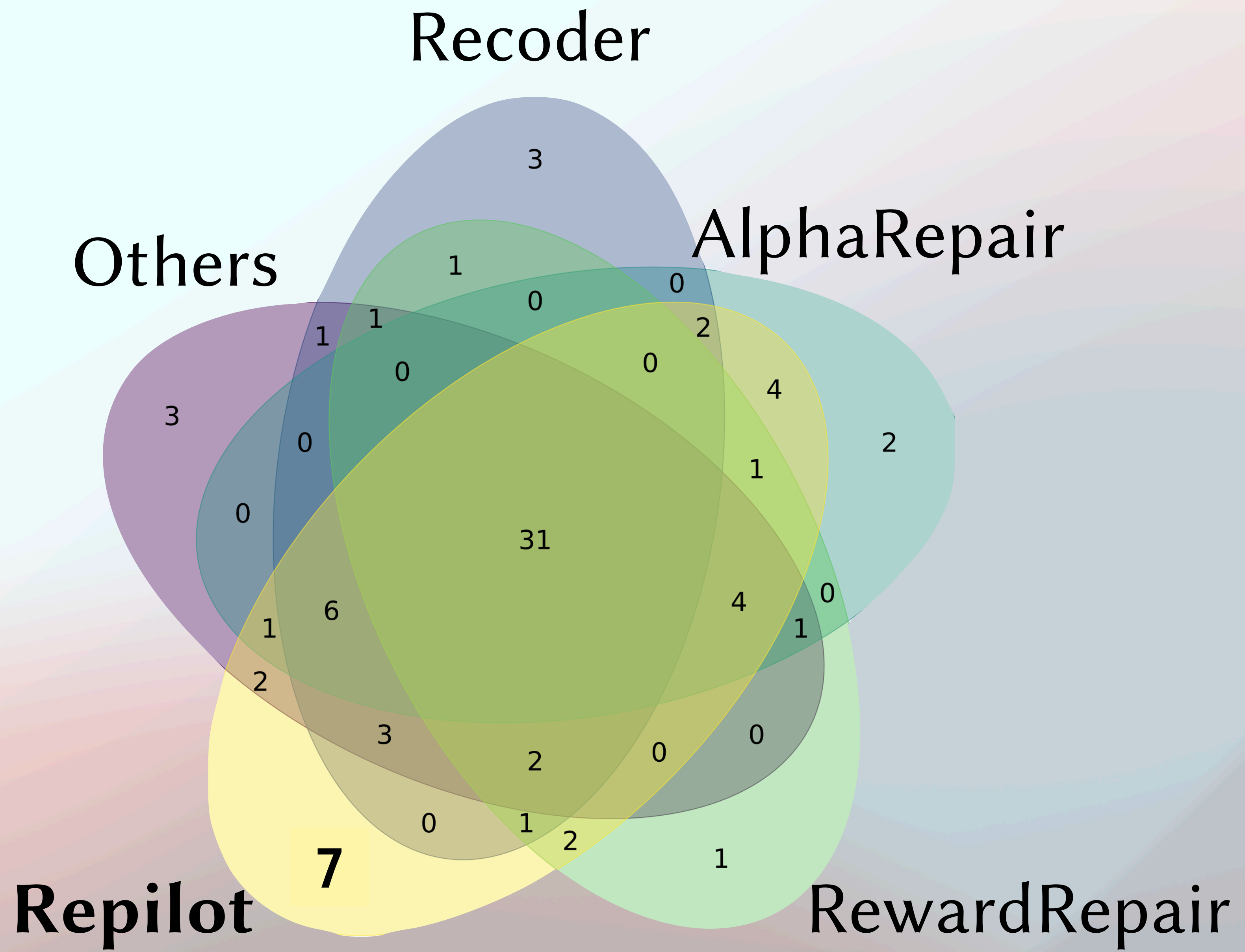
Evaluation

Evaluation pipeline and metrics



Evaluation
Comparison with existing tools

Tool	Methodology	#Correct Fixes		
		Defects4J 1.2	Defects4J 2.0	Total
CoCoNuT	NMT	30	-	-
DLFix	NMT	32	-	-
PraPR	Template	35	-	-
TBar	Template	41	7	48
CURE	NMT	43	-	-
RewardRepair	NMT	45	24	69
Recoder	NMT	51	10	61
AlphaRepair	LLM	52	34	86
Repilot	LLM	**66**	**50**	**116**



Number of unique fixes

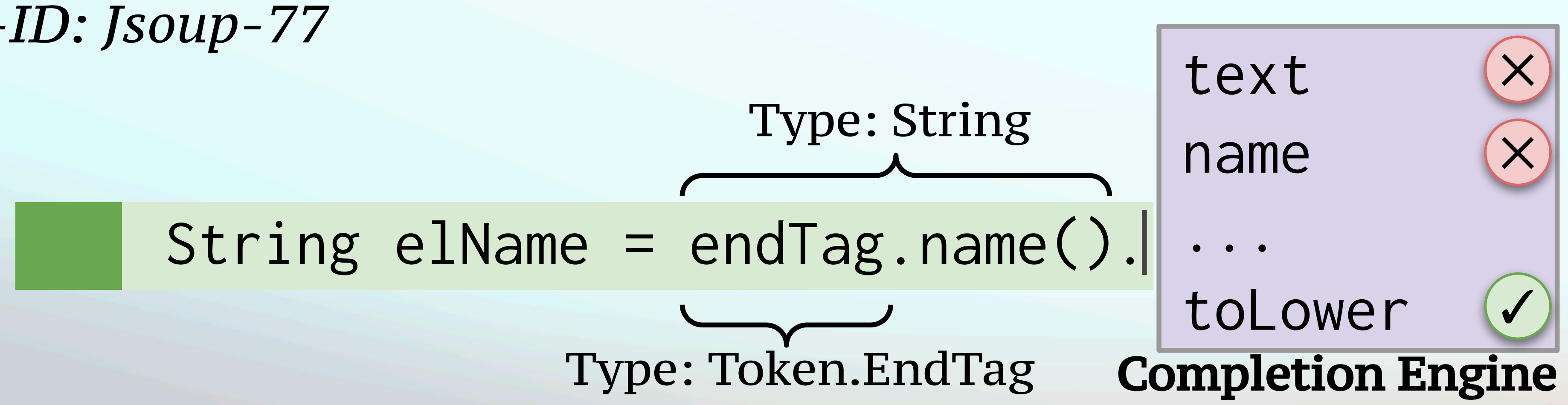
NMT means “Neural Machine Translation”

Evaluation

Unique fixes generated by Repilot

```
  private void popStackToClose(Token.EndTag endTag) {
-     String elName = endTag.name();
+     String elName = endTag.name().toLowerCase();
      Element firstFound = null;
```

Bug-ID: Jsoup-77

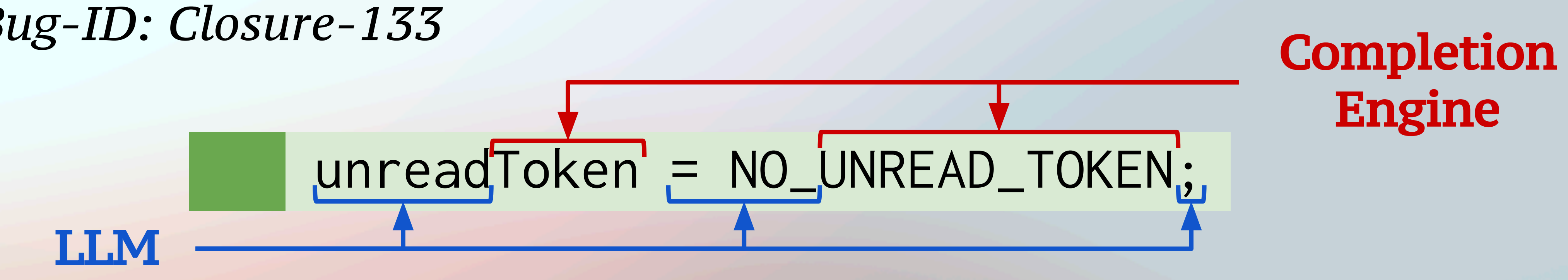


Patch Generation Process

Completion engine filters out invalid tokens

```
  private String getRemainingJSDocLine() {
      String result = stream.getRemainingJSDocLine();
+     unreadToken = NO_UNREAD_TOKEN;
      return result;
  }
```

Bug-ID: Closure-133



Patch Generation Process

Interaction between LLM and completion engine

Evaluation
Generalizability

Repilot is generalizable across bug subjects and models

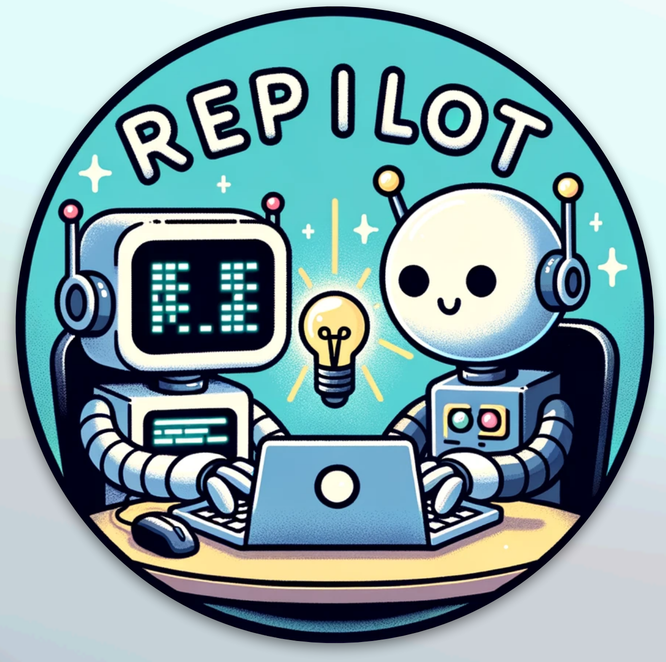
Variant	Model	Subject of Bugs	Generation Time	%Compilable Patches	#Correct Fixes
Base LLM	CodeT5-large	Defects4J 1.2	**0.232s**	43.2%	37
Repilot	CodeT5-large	Defects4J 1.2	0.248s	**63.4%**	**42**
Base LLM	CodeT5-large	Defects4J 2.0	**0.230s**	46.7%	41
Repilot	CodeT5-large	Defects4J 2.0	0.247s	**64.8%**	**45**
Base LLM	INCODER-6.7B	Defects4J 1.2	**1.70s**	32.4%	48
Repilot	INCODER-6.7B	Defects4J 1.2	**1.70s**	**47.2%**	**54**
Base LLM	INCODER-6.7B	Defects4J 2.0	**1.67s**	34.6%	45
Repilot	INCODER-6.7B	Defects4J 2.0	1.69s	**48.0%**	**46**

Repilot

Copiloting the Copilots

Fuses **Large Language Models** with **Completion Engines** for more **effective Patch Generation** in Automated Program Repair

Can be applied to general **program synthesis**



Powered by DALL·E 3

Yuxiang Wei

ywei40@illinois.edu

(⭐104) ise-uiuc/Repilot

Code & Docker & Artifact

arXiv:2309.00608

Paper

@YuxiangWei9

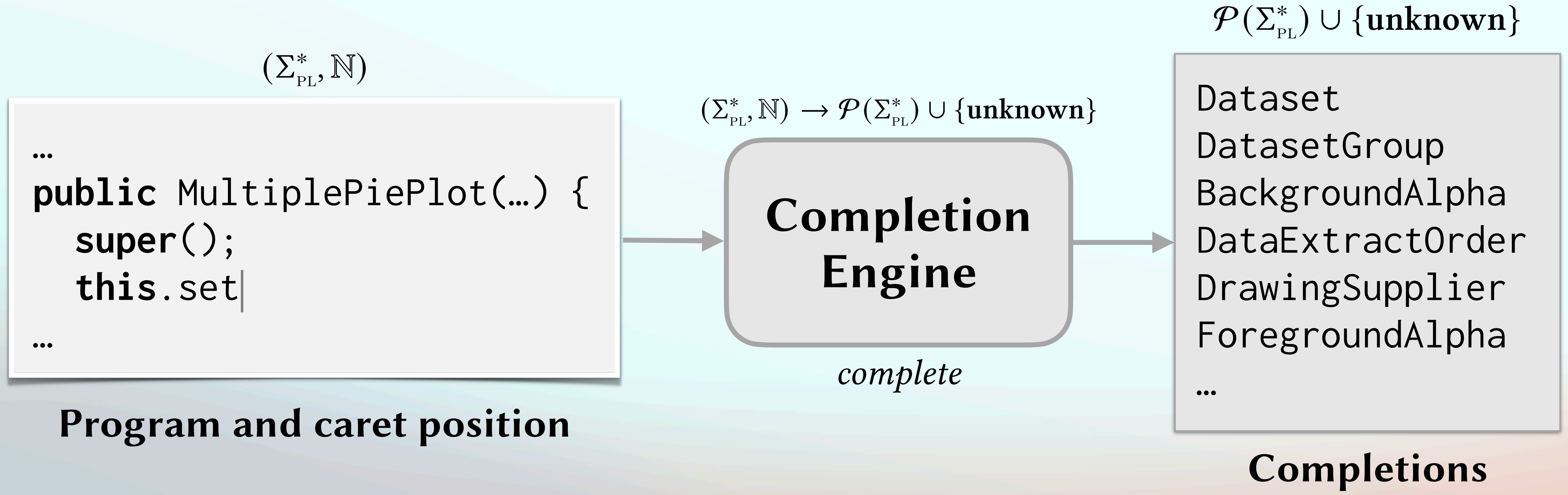
Twitter/X

Supplementaries

Will Repilot go wrong?

Conditional soundness of Repilot

Theorem: Repilot is sound with a **strict completion engine**



Assume

$(\text{prog}, \text{caret}) \models \Phi$

$\text{completions} = \text{complete}(\text{prog}, \text{caret})$

$\text{completions} \neq \textbf{unknown}$

The completion engine is strict if and only if

$\forall c \notin \text{Prefix}(\text{completions}), (\text{prog}', \text{caret}') \not\models \Phi$

$\text{where} \quad \text{prog}' = \text{prog}[\text{caret} \leftarrow c]$

$\qquad\qquad \text{caret}' = \text{caret} + |c|$

$\text{Prefix}(\,\cdot\,) = \{c \mid s \in \cdot \text{ and } c \text{ is a prefix of } s \text{ or vice versa}\}$

Intuitively, the completion engine should capture all possible continuations, but can do over-approximation. **"unknown"** is an over-approximation.

Why resample instead of direct pruning?

In this case, direct pruning cannot do anything, and may result in a wrong path, e.g., `.method_a`