MatchFixAgent: Language-Agnostic Autonomous Repository-Level Code Translation Validation and Repair

Ali Reza Ibrahimzada[1], Brandon Paulsen[2], Reyhaneh Jabbarvand[1], Joey Dodds[2], Daniel Kroening[2 3]

[1]University of Illinois Urbana-Champaign [2]Amazon [3]University of Oxford

International Conference on Machine Learning (ICML)
July 6-11, 2026, Seoul, South Korea

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

amazon

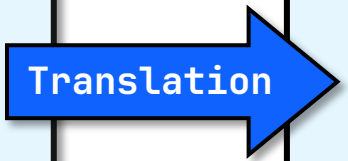
alirezai@illinois.edu

What is Code Translation?

- Code translation converts source code from one programming language (PL) to another

Java Code

```java
public class Main {
  public static void main(String[] args) {
      int max = 5;
      for (int i = 0; i < max; i++) {
          System.out.println(i);
      }
  }
}
```



Python Code

```python
max = 5
for i in range(max):
    print(i)
```

Why Code Translation?

- Use cases of code translation:
  - Application modernization
  - Migrating legacy software to cloud-native applications
- Transpiler techniques, e.g., C2Rust, CxGO, Java2C#
  - Produces **non-idiomatic code** and **does not generalize**
- LLMs, e.g., GPT-4, Claude Sonnet 
  - **Generalizes well** and **generates more natural code** but **does not scale**

Why Code Translation?

THE WHITE HOUSE

Administration Priorities The Record Briefing Room Español

FEBRUARY 26, 2024

Statements of Support for Software Measurability and Memory Safety

ONCD ▸ BRIEFING ROOM ▸ PRESS RELEASE

Read the full report here

Read the fact sheet here

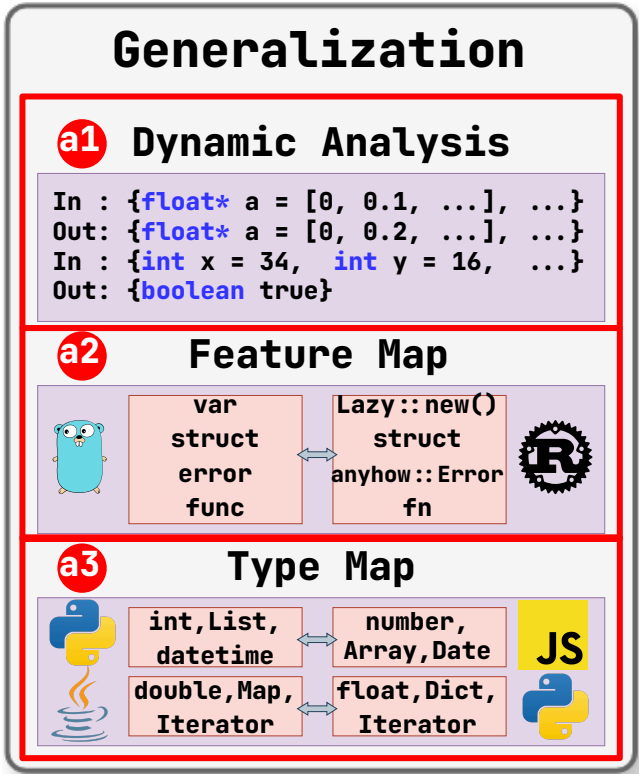
Today, the Office of the National Cyber Director released a new Technical Report titled *"Back to the Building Blocks: A Path Toward Secure and Measurable Software."* This report builds upon the President's National Cybersecurity Strategy, addressing the technical community to tackle undiscovered vulnerabilities that malicious actors can exploit.

Existing Translation & Validation Techniques

- AlphaTrans [FSE'25]
  - Java to Python
  - Uses GraalVM for validation
- Oxidizer [PLDI'25]
  - Go to Rust
  - Uses mocking for validation
- Skel [PLDI'25]
  - Python to JavaScript
  - Uses program execution for validation

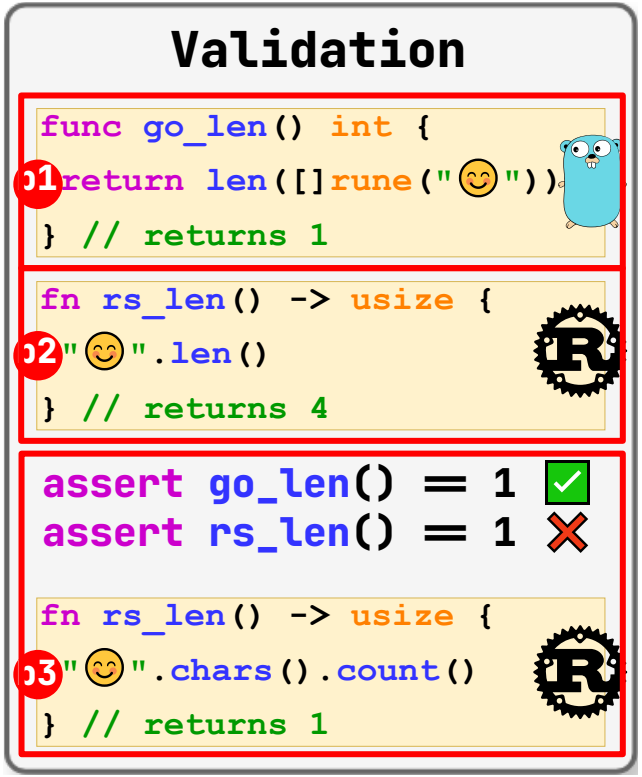
Limitation 1: Generalization

- High engineering overhead and limited generalizability
- Language-specific design decisions
- Development time
  - **8 months** for AlphaTrans
  - **19K** LoC in Oxidizer



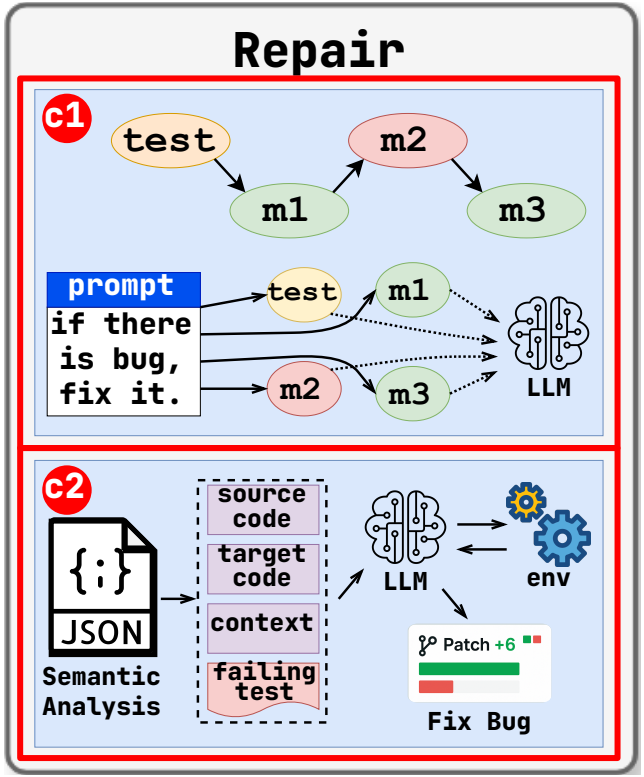
Limitation 2: Validation

- Existing test suites can be incomplete and not rigorous
- Code coverage is limited to developer-written tests

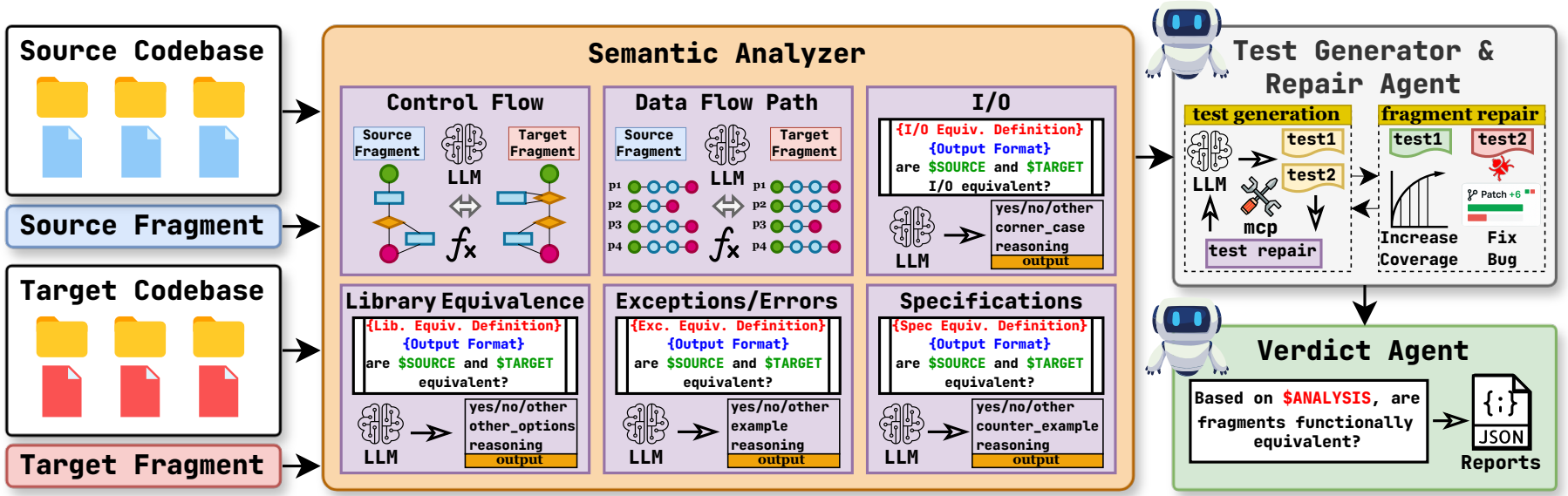


Limitation 3: Repair

- All existing tools fail to repair translation bugs
- Re-prompting fragments separately can introduce new bugs

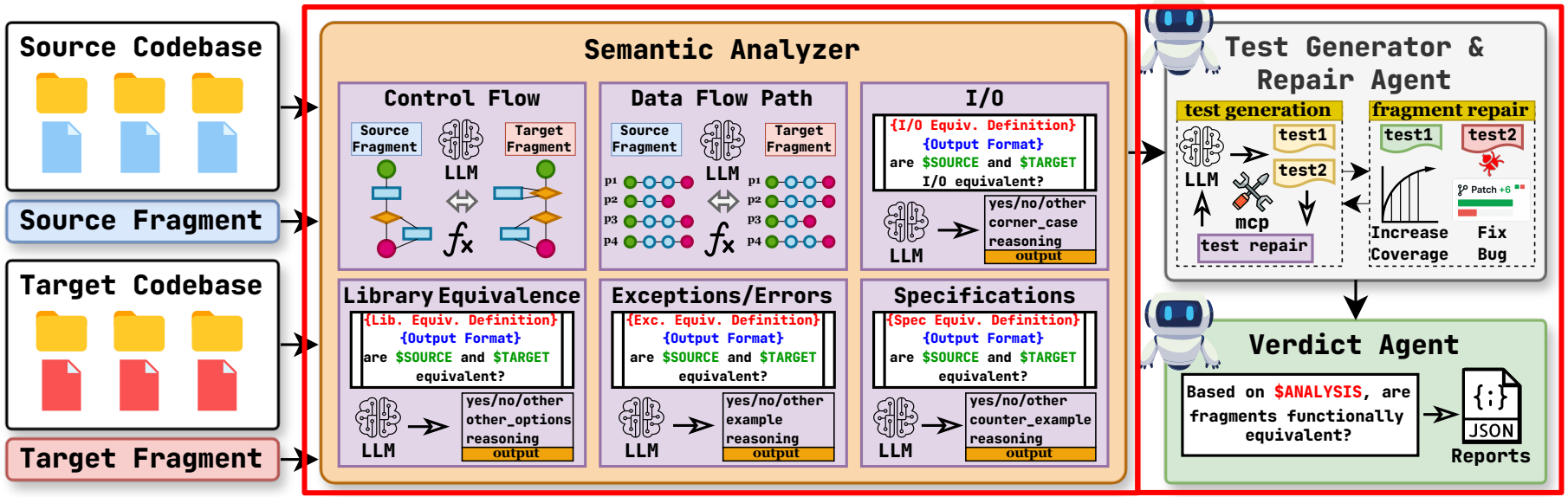


MatchFixAgent



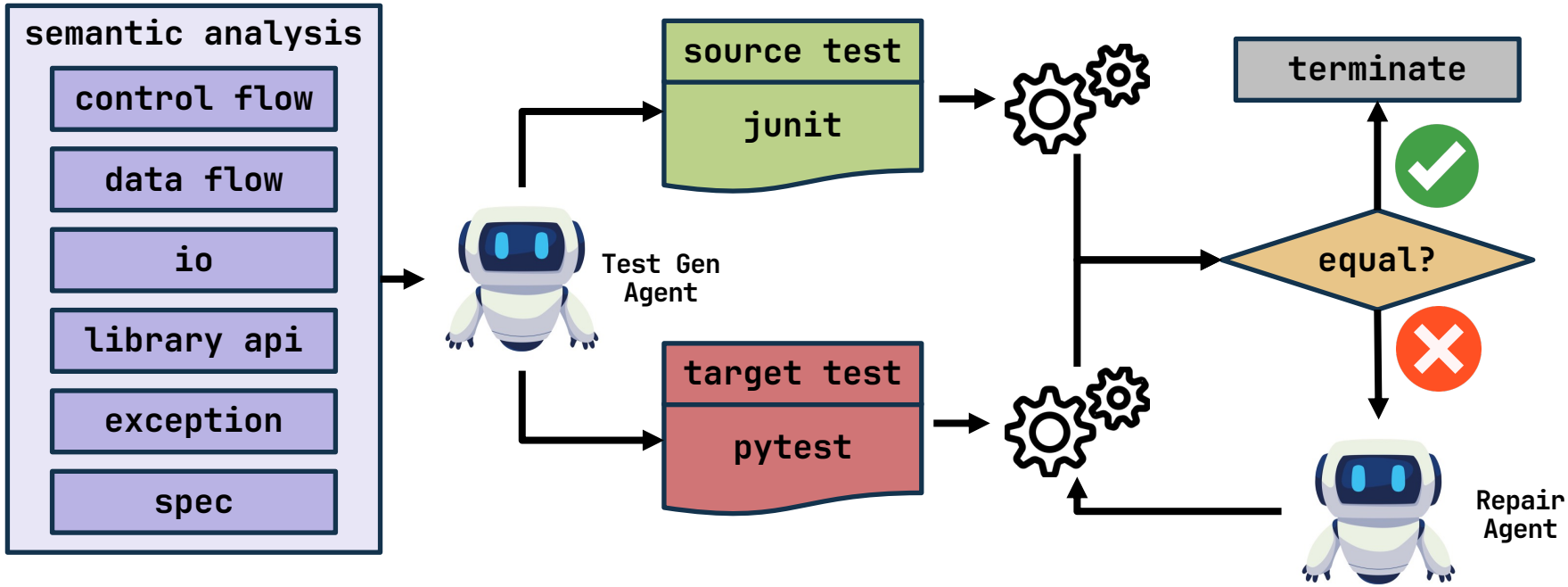
- Modular design and interpretable
- LLMs guided by program analysis
- Supports multiple PLs, LLMs and CLI Agents

Semantic Analyzer



- Six-dimension semantic analysis
- Natural language reasoning
- Control flow and data flow analysis

Test Generator and Repair Agent



Evaluation

- **4** existing tools and **6** language pairs
- **2219** fragments
- **900K** lines of code
- **89.6%** test coverage

Tools	Source Language	Target Language	# Translation Pairs	LoC	Test Coverage (%)
Oxidizer	Go	Rust	192	4167	92.1
RustRepoTrans	Python | C | Java	Rust	344	786069	99.4
AlphaTrans	Java	Python	1346	116085	66.3
Skel	Python	JavaScript	337	1206	91.9
Total			**2219**	**910398**	**89.6**

Effectiveness in Translation Validation

- Validates **99.2%** of fragments
- **72.8%** of verdicts agree with existing techniques
- **60.7%** of investigations in favor of MatchFixAgent

Tools	# Trans. Pairs	Tool Validation			MatchFixAgent			Agreement	Disagreement	
		EQ	NEQ	VF	EQ	NEQ	VF		Tool	Ours
Oxidizer	192	138 (71.9)	53 (27.6)	1 (0.5)	132 (68.8)	59 (30.7)	1 (0.5)	121 (63.7)	15.9	84.1
RustRepoTrans	344	153 (44.5)	188 (54.7)	3 (0.9)	104 (30.2)	228 (66.3)	12 (3.5)	264 (80.2)	87.5	12.5
AlphaTrans	1346	573 (42.6)	189 (14)	584 (43.4)	1022 (75.9)	320 (23.8)	4 (0.3)	525 (69.3)	26.5	73.5
Skel	337	276 (81.9)	19 (5.6)	42 (12.5)	261 (77.4)	75 (22.3)	1 (0.3)	233 (79.3)	46.5	53.5
Total	**2219**	**1140 (51.4)**	**449 (20.2)**	**630 (28.4)**	**1519 (68.5)**	**682 (30.7)**	**18 (0.8)**	**1143 (72.8)**	**39.3**	**60.7**

EQ: Equivalent, NEQ: Not-Equivalent, VF: Validation Failure

Qwen3 experiments with cost analysis in the paper!

Effectiveness in Translation Repair

- **265** samples for repair
- Patches **50.6%** of fragments (**32.1%** more)
- **8.5%** increase in code coverage

Tools	# Trans. Pairs	Tool NEQ ∩ MatchFixAgent NEQ	Tool Repaired	MatchFixAgent Repaired	Coverage (Improve) %
Oxidizer	192	21 (10.9)	0 (0)	17 (81)	92.4 (↑0.3)
RustRepoTrans	344	170 (49.4)	49 (28.8)	66 (38.8)	100 (↑0.6)
AlphaTrans	1346	63 (4.7)	0 (0)	46 (73)	99.6 (↑33.3)
Skel	337	11 (3.3)	0 (0)	5 (45.5)	100 (↑8.1)
Total	**2219**	**265 (11.9)**	**49 (18.5)**	**134 (50.6)**	**98.1 (↑8.5)**

NEQ: Not-Equivalent

Illustrative Example

Go

```go
lcs := LCS(str1, str2)
return (len([]rune(str1)) - lcs) +
       (len([]rune(str2)) - lcs)
```

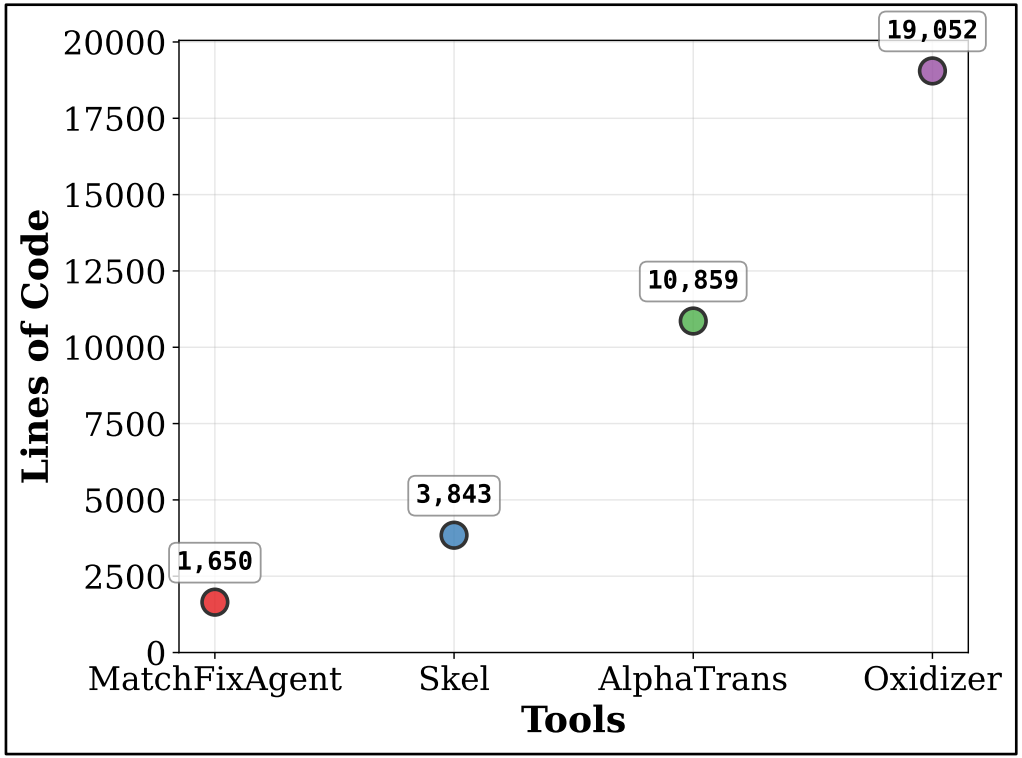
Rust

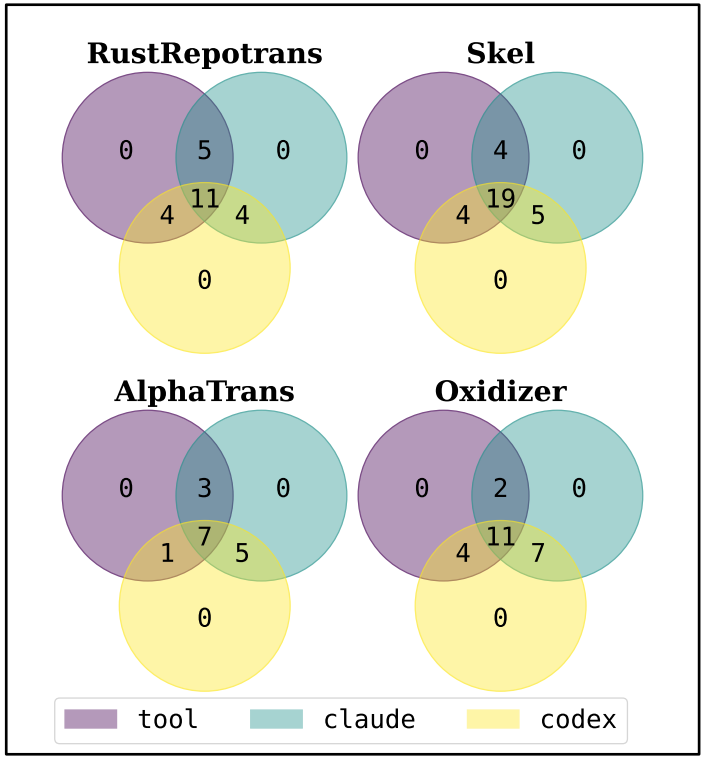
```rust
let lcs_len = lcs(str1, str2)?;
let edit_distance =
      (str1.len() as i32 - lcs_len) +
      (str2.len() as i32 - lcs_len);
```

The Go implementation correctly uses `len([]rune(str1))` and `len([]rune(str2))` to count **Unicode characters**, while the Rust implementation was using `str1.len()` and `str2.len()` which count **bytes** rather than **characters**. This would cause incorrect results for strings containing multi-byte Unicode characters. The fix was to modify the Rust implementation to use `str1.chars().count()` and `str2.chars().count()` to correctly count Unicode characters.

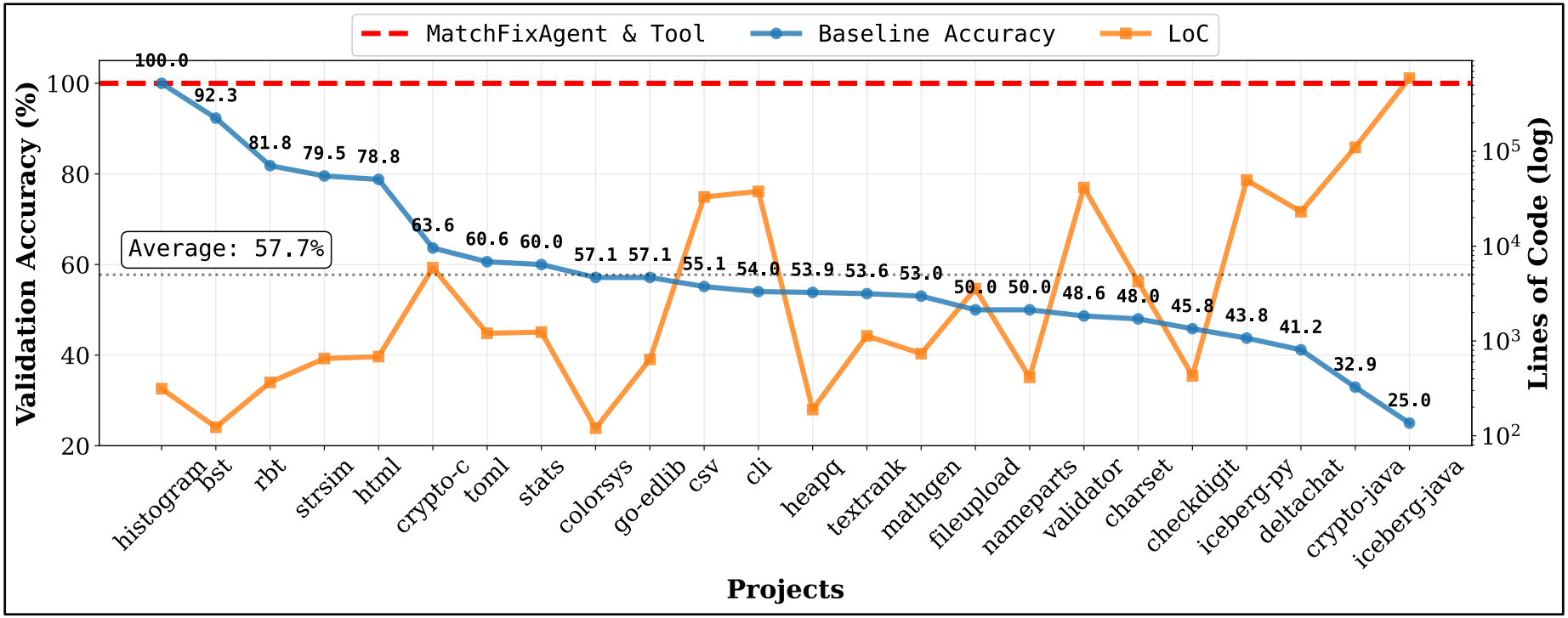
Go: distance("👍a", "a") = 1 ✅
Rust: distance("👍a", "a") = 4 ❌

Development Cost and Adaptability



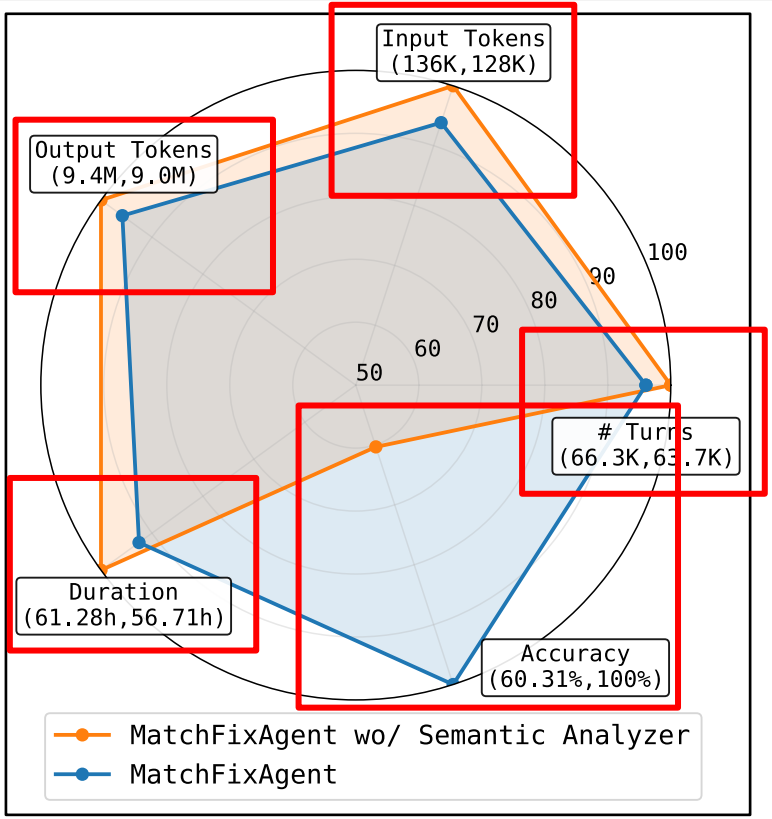


Ablation Study I: Remove SA and TGA



Ablation Study II: Remove SA Only

- **3.9%** more turns
- **6.2%** more input tokens
- **4.2%** more output tokens
- **7.5%** more time
- **39.7%** less accurate



Concluding Remarks

- **MatchFixAgent**: A language-agnostic autonomous system for repository-level code translation validation and repair
- Evaluated on **six** PL pairs and compared with **four** tools
- Validates **99.2%** of translations and repairs **50.6%** of translation bugs
- Human investigation resolves **60.7%** of disputes in favor of MatchFixAgent
- Ablation study to justify the design of MatchFixAgent

Full Paper and Artifacts

http://arxiv.org/abs/2509.16187

https://github.com/Intelligent-CAT-Lab/MatchFixAgent