DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS
NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

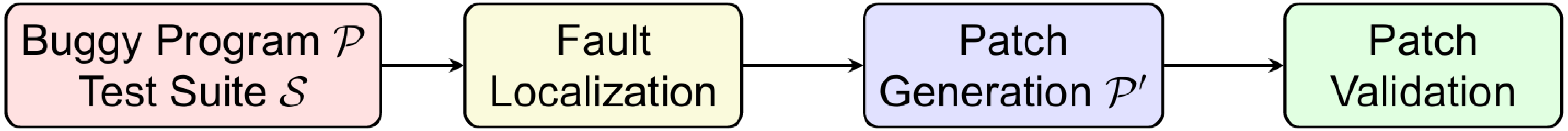
24 OCTOBER 2025

PAUL: Patch Automation Using LLMs

Michael-Raphael Kostagiannis
Undergraduate student
(not for long!)

Software Repair

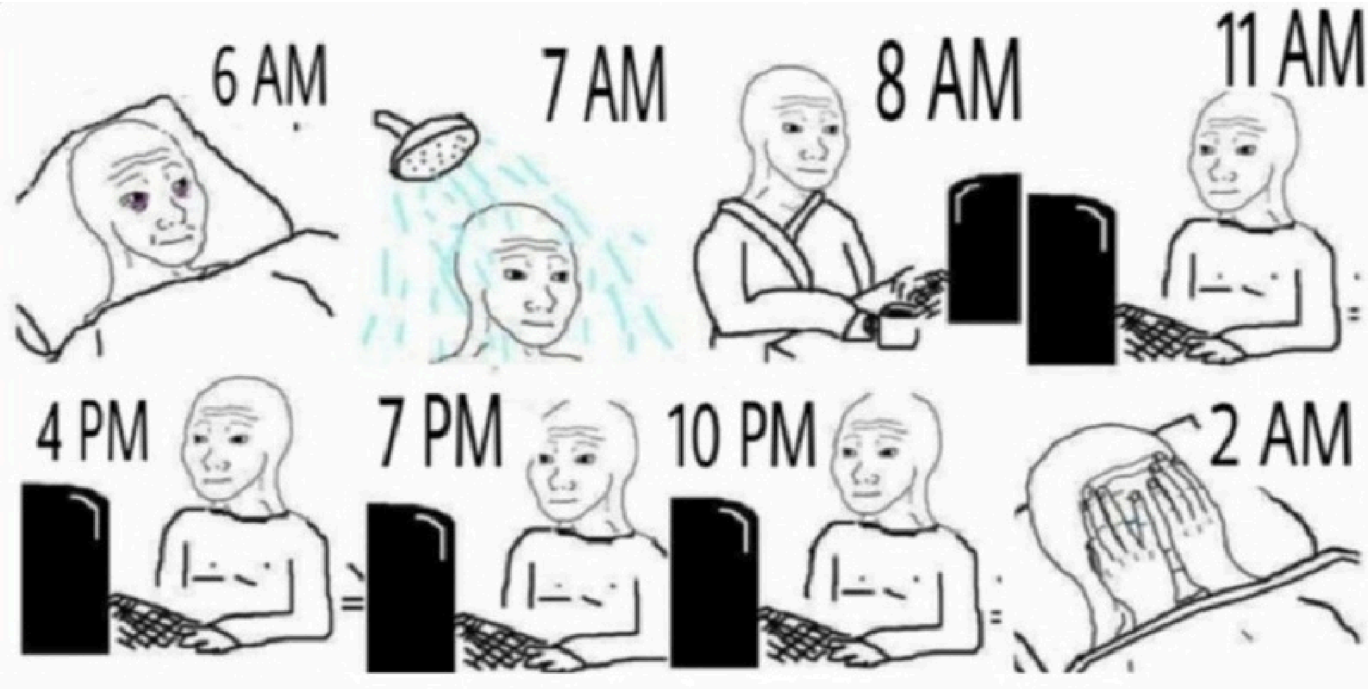
- ”Software systems become legacy systems when they begin to resist modification and evolution.” - Robert C. Seacord
- **Software repair** is the process of detecting software failures and applying fixes at the source code level.
- Typically represented as a multi-step pipeline:



- Is tiring and boring.

Why automate software repair?

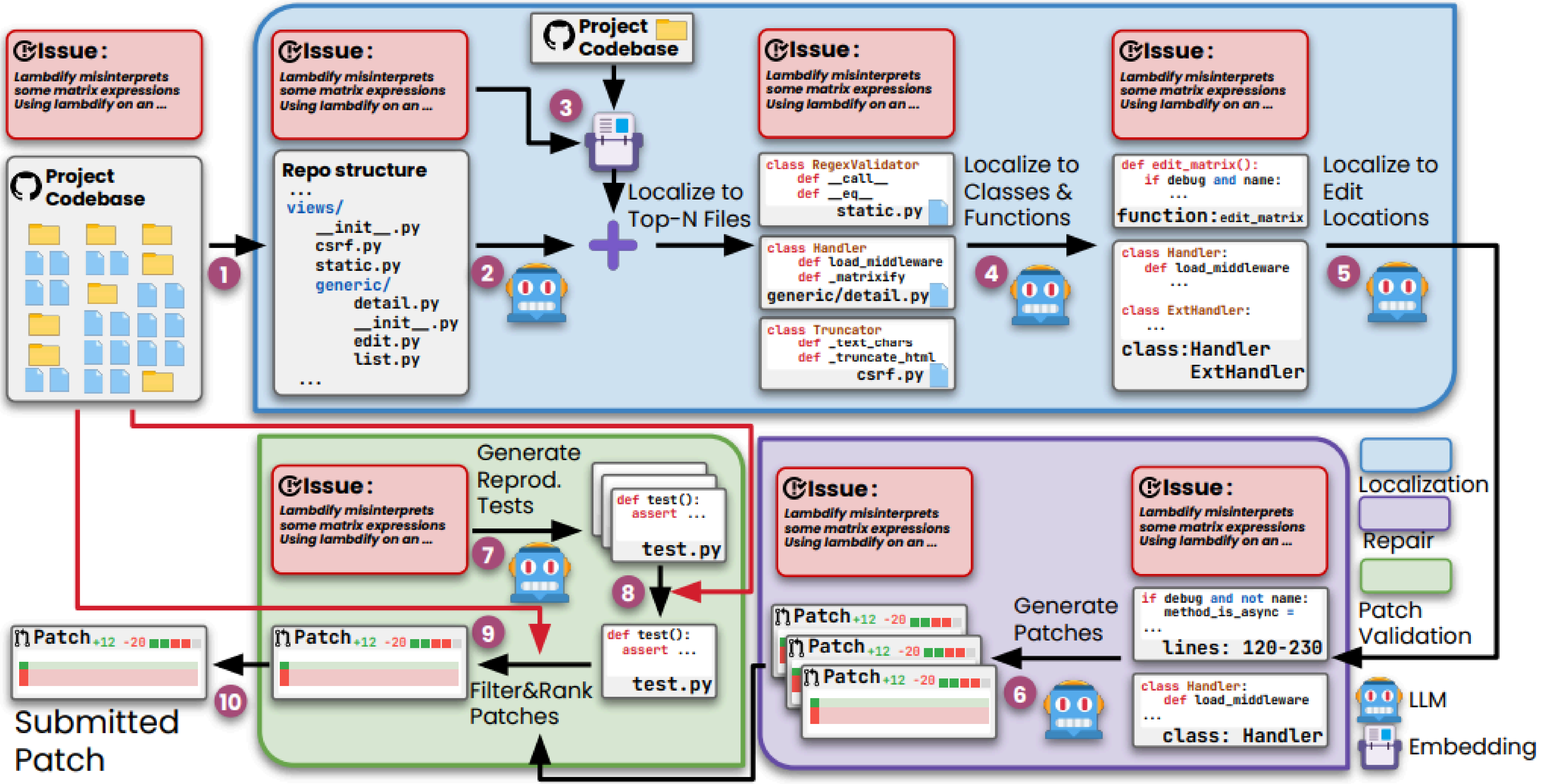
- **Bugs are expensive** – Software failures cost companies billions (e.g., NASA's Mars Climate Orbiter failure due to a unit conversion bug).
- **Time-consuming process** – Developers spend 50%+ of their time debugging. Ubuntu lists more than 140,000 open bugs as of today.
- **Security vulnerabilities** – Delayed bug fixes can be exploited.
- **Human error in fixes** – Manually written patches can introduce new bugs.
- Is **really** tiring and boring.



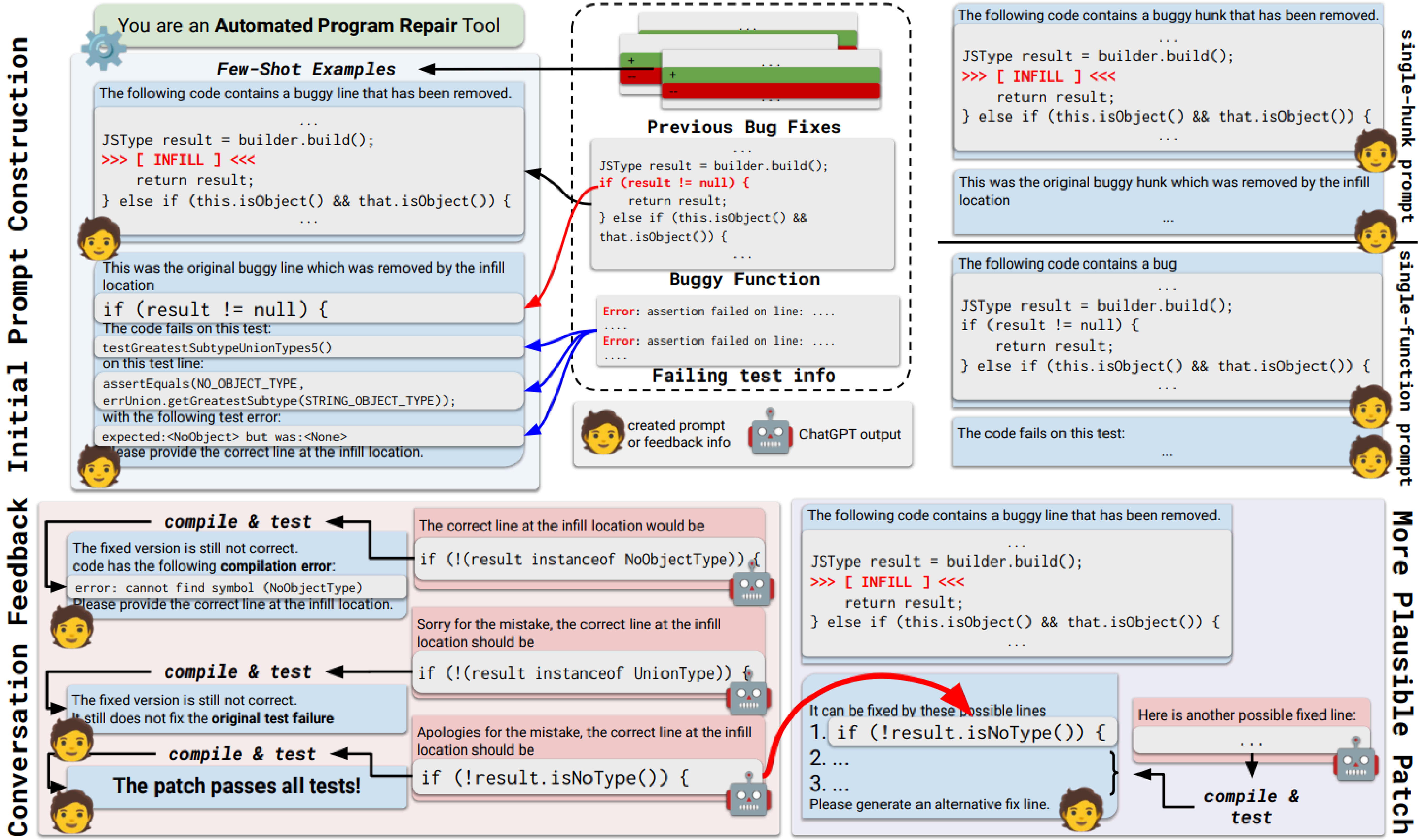
Why bother with LLMs?

- Traditional **Automated Program Repair (APR)** tools do exist but they are limited in their usage and efficiency (e.g. GenProg) due to **lack of deep reasoning**. We need systems with code reasoning and context awareness.
- **Large Language Models (LLMs)** are pre-trained deep learning model designed to generate and understand natural language. We can use these to our advantage in order to enhance APR techniques!

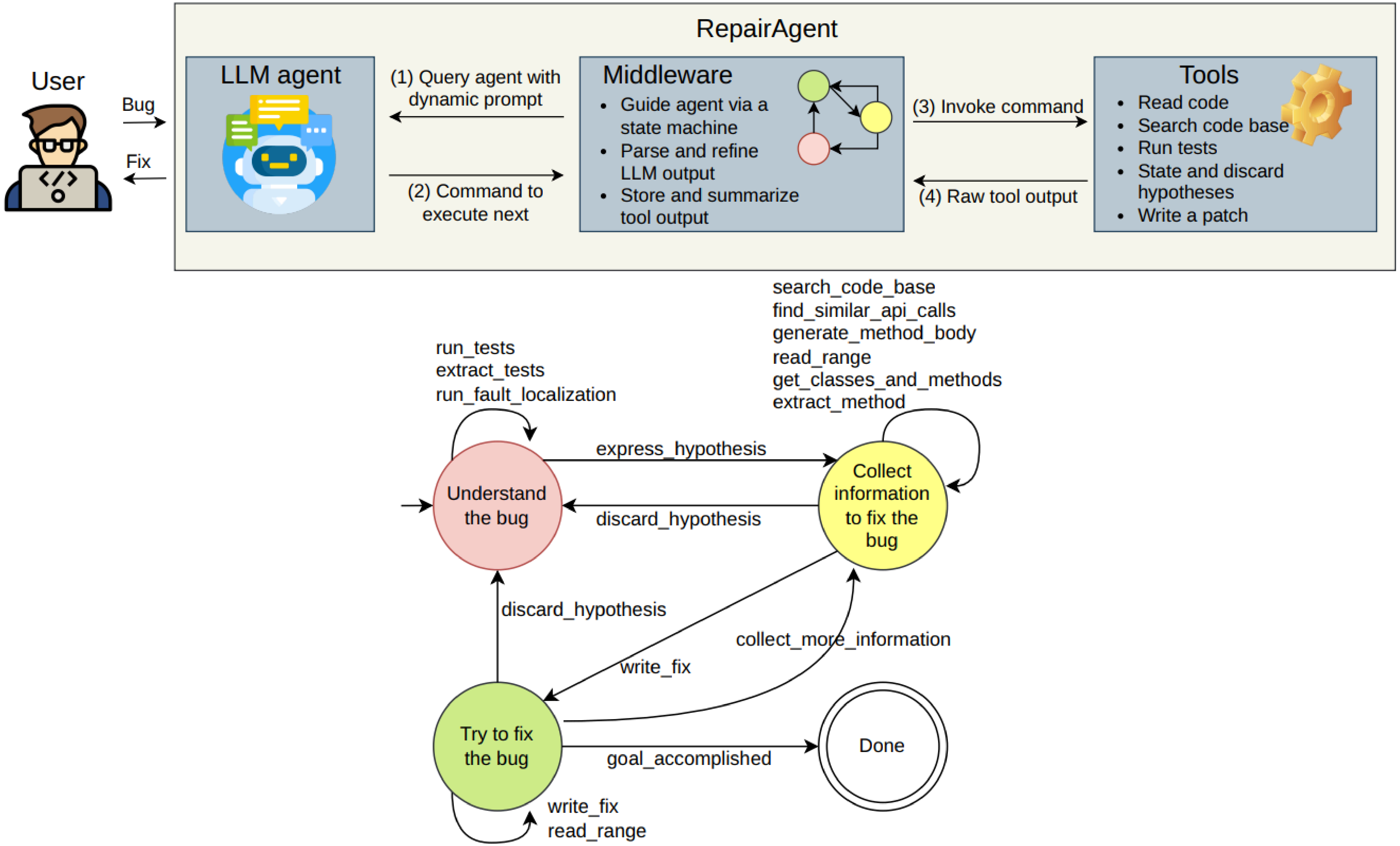
LLM-based APR: AGENTLESS



LLM-based APR: ChatRepair



LLM-based APR: RepairAgent



Are they good?

- **Absolutely!** LLM-driven software repair systems solve real-world problems that no previous system could solve. Such systems, however are:
  - **Hard to use**: Need rigorous, manual setup.
  - **Strictly reaserch-based**: Little to no everyday integration.

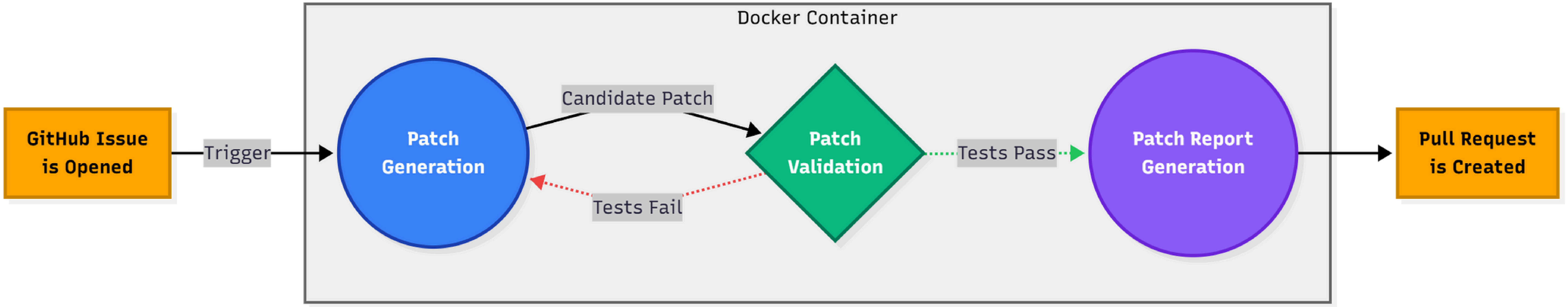
System	Benchmark	Success Rate	Avg. Cost	Availability
AGENTLESS	SWE-bench Lite	32% (96 bugs)	$0.70	Open-source
ChatRepair	Defects4J	12.7% (114 + 48 bugs)	$0.42	Closed-source
RepairAgent	Defects4J	12.9% (164 bugs)	$0.14	Open-source

- Ideally, we want an LLM-based APR system with minimal setup and high success rate. But that is simply impossible... right?

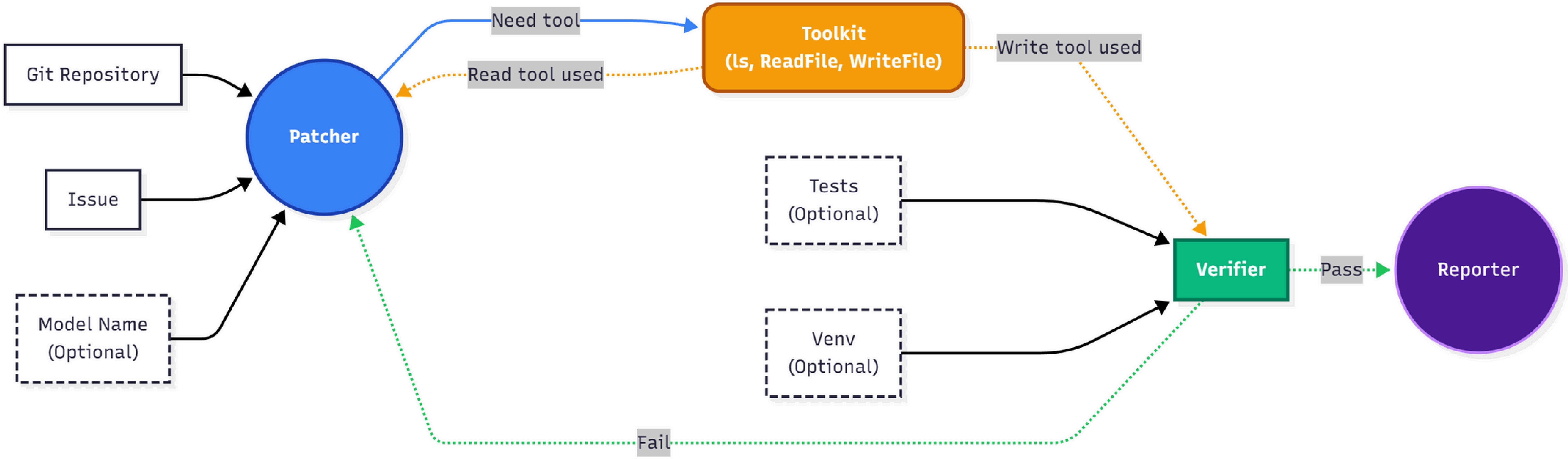
Meet PAUL!

- **P**atch **A**utomation **U**sing **L**LMs - An open-source, graph based multi-agent system.
- Personal GitHub assistant that automatically analyzes issues and opens pull requests.
- Also supports local execution.
- **No user intervention!**
- Required setup:
  - GITHUB_TOKEN
  - OPENAI_API_KEY

Overview of PAUL



PAUL’s core workflow



Demo

Why prefer PAUL?

- **Full Autonomy with Minimal Setup:** PAUL performs the entire repair cycle automatically. No human interaction. No dependency hell.
- **Feedback-Driven Multi-Agent Worfklow:** Separate agents handle distinct tasks. Improves reasoning and accuracy.
- **Verified Output:** Every patch is tested and validated before submission.
- **Cost-Efficient and Scalable:** Users retain full control on model selection.
- **Open Source:** All resources can be found on the official PAUL repository with reproducible results and artifacts.

PAUL-tests

- Custom benchmark suite specifically designed to validate end-to-end functionality.
- Consists of five deliberately simple Python programs. **90% success rate.**

Python Program	Patcher Tokens	Failed Attempts	Reporter Tokens	Total Tokens	Total Cost ($)	Execution Time (s)
is_anagram	2863	0	1688	4551	0.000921	15.6563
list_deduplicator	2655	0	1493	4148	0.000794	14.6280
middle_element	8316	2	1568	9884	0.001690	22.8476
remove_nth	3517	0	876	4393	0.000787	34.2386
reverse_string	3494	0	869	4363	0.000799	14.3621
Average	4169	0.4	1298	5467	0.001	20.34

QuixBugs

- Classic APR benchmark with known bugs.
- Contains 40 Python and 40 Java defects.
- **87.5-97.5% success rate** across both language subsets depending on the underlying model.

Table 5.2: Average performance metrics on QuixBugs Python subset (3 runs).

Model Name	Programs Repaired	Patcher Tokens	Failed Attempts	Reporter Tokens	Total Tokens	Total Cost ($)	Execution Time (s)
gpt-4o-mini	36	12716	1.00	1693	14409	0.00256	24.45
gpt-4o	39	18221	0.90	1911	20132	0.01341	25.73

Table 5.3: Average performance metrics on QuixBugs Java subset (3 runs).

Model Name	Programs Repaired	Patcher Tokens	Failed Attempts	Reporter Tokens	Total Tokens	Total Cost ($)	Execution Time (s)
gpt-4o-mini	35	15472	1.30	1974	17446	0.00605	19.03
gpt-4o	38	14659	1.10	2128	16787	0.02143	21.52

Table 5.4: Average GPT-5-mini performance metrics (3 runs).

QuixBugs Program	Patcher Tokens	Failed Attempts	Reporter Tokens	Total Tokens	Total Cost ($)	Execution Time (s)
topological_ordering	14918	0	2365	17283	0.000000	56.7874
LCS_LENGTH	19831	0	3191	23022	0.000000	42.2594
WRAP	31506	0	2716	34222	0.000000	45.0498
Average	22085	0.0	2757	24842	0.000000	48.03

SWE-bench Lite

- Benchmark comprising real world GitHub issues from popular open-source Python repositories.
- Over **300,000 lines of code** distributed across hundreds of Python modules.

Model	Patcher Tokens	Failed Attempts	Reporter Tokens	Total Tokens	Total Cost ($)	Execution Time (s)
gpt-4o-mini –file	3970	0	1846	5816	0.000992	10.0566
gpt-4o	492438	0	1881	494319	0.673647	80.7116
Average	248204	0	1863	250067	0.33731	45.38

Future Work

- Fault Localization
- Multi-File Repair
- Build and Test Generalization
- Model Escalation Strategies
- Novel and Contamination-Resistant Benchmarks
- Security and Robustness Against Adversarial Inputs
- Proactive Maintenance and Autonomous Code Reasoning

Conclusion

- PAUL is an open-source, graph-based multi-agent debug assistant.
- Easy integration on any public GitHub and local repository.
- Leverages of LLMs to produce more flexible, context-aware, minimum-cost solutions.
- Achieves high-percent success rates on verified benchmarks.
- As LLMs continue to evolve, so will PAUL.

Thank you!

Michael-Raphael Kostagiannis

Undergraduate student
(not for long!)