CARIAD

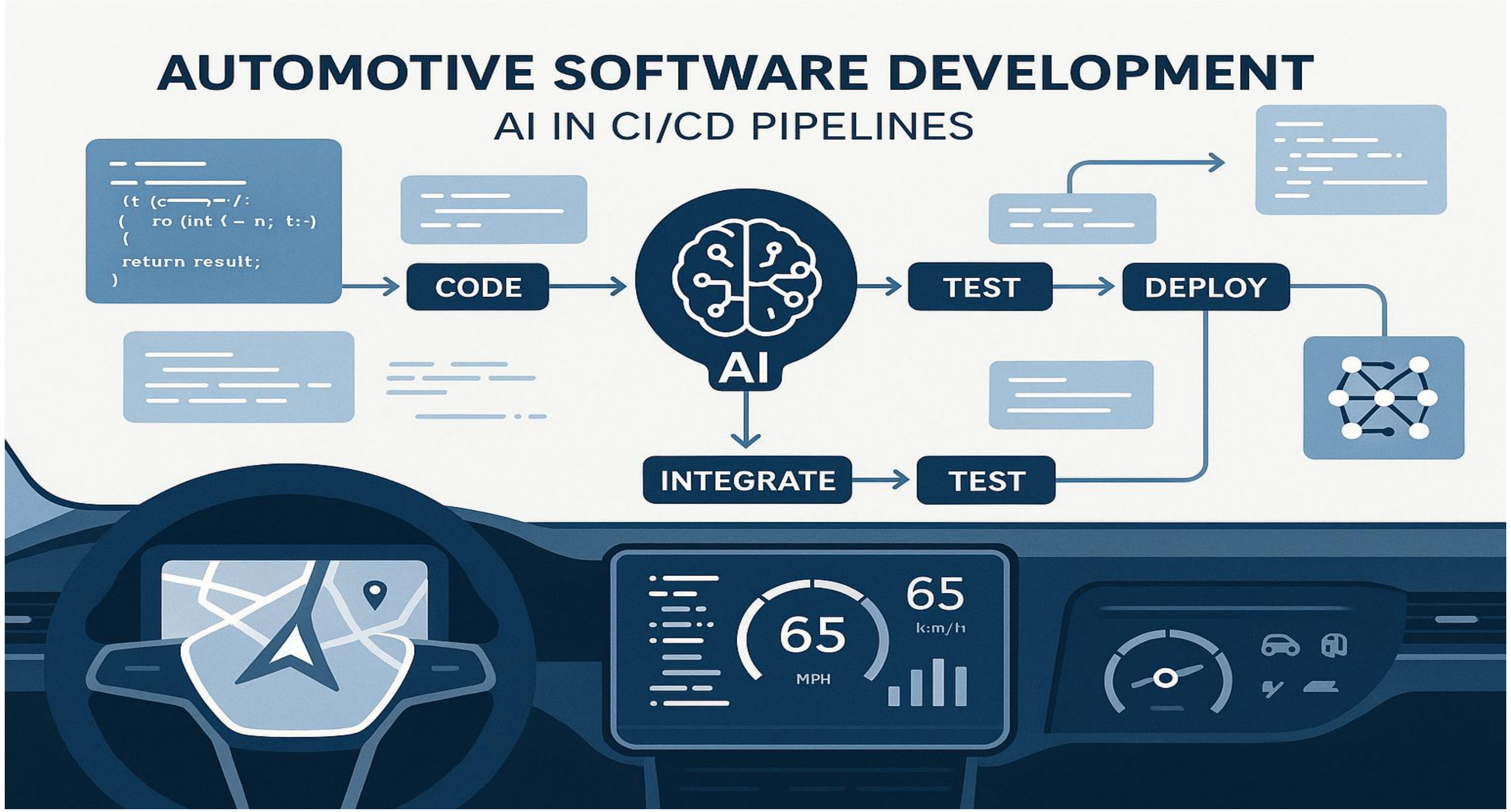
Master Thesis: AI Usage in CI/CD/CT Pipelines for Compute Platforms in Automotives

We transform automotive mobility

CARIAD
A VOLKSWAGEN GROUP COMPANY

Agenda

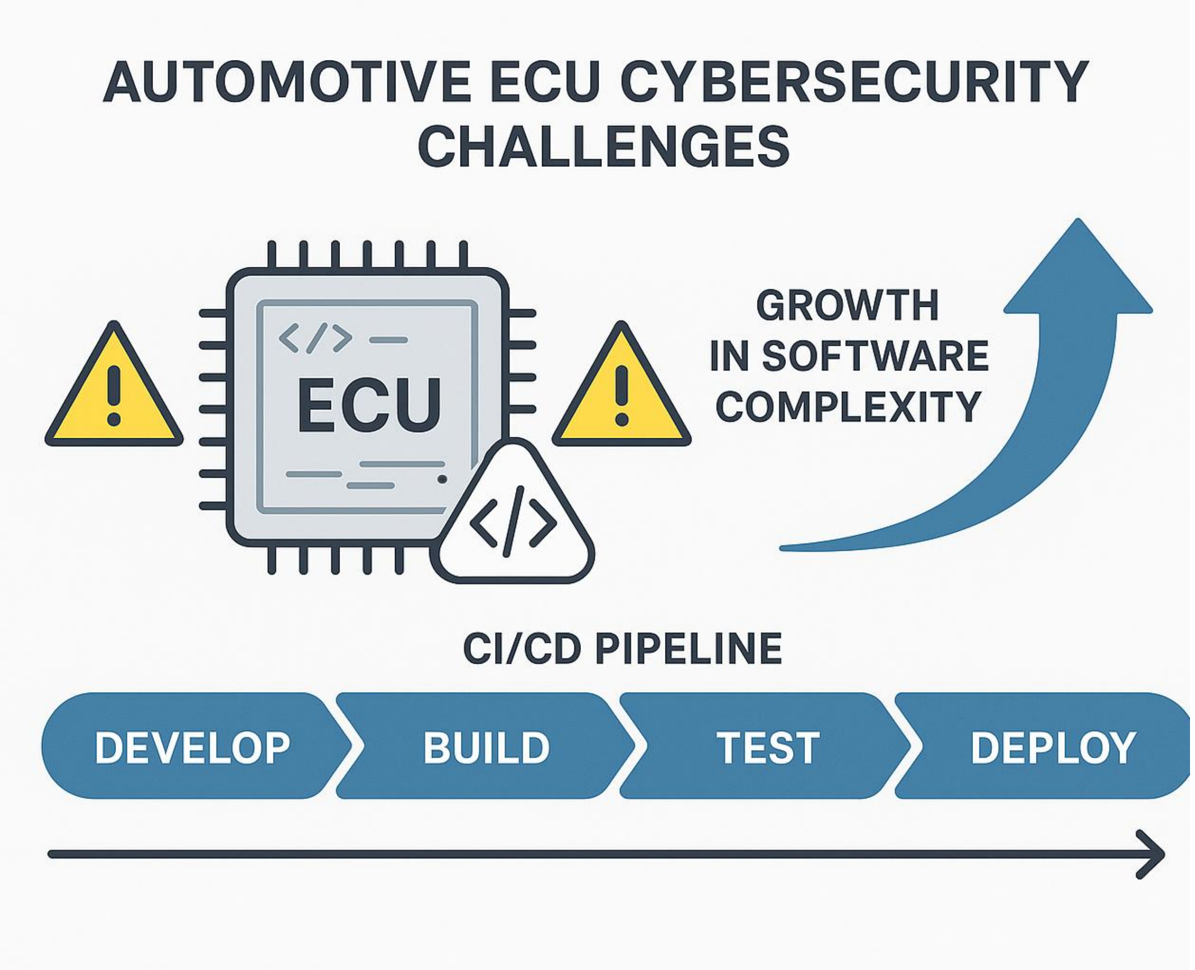
// Introduction
// Method
// Result
/7 Status
// Next Steps



Introduction

Introduction

- *Rapidly growing software complexity & shorter release cycles*
- *Automotive ECUs are safety-critical, hence zero tolerance*
- *Traditional security tests cannot keep pace with CI/CD demand*



Problem Statement

- Current white-box fuzzing & testing are manual or slow to scale
- Vulnerabilities may slip through nightly CI due to time limits
- Need an AI-guided approach integrated into CI/CD/CT to

– boost path coverage

– reduce manual fuzz test case creation

– auto-generate actionable test artifacts

Research Objectives

Technique Design

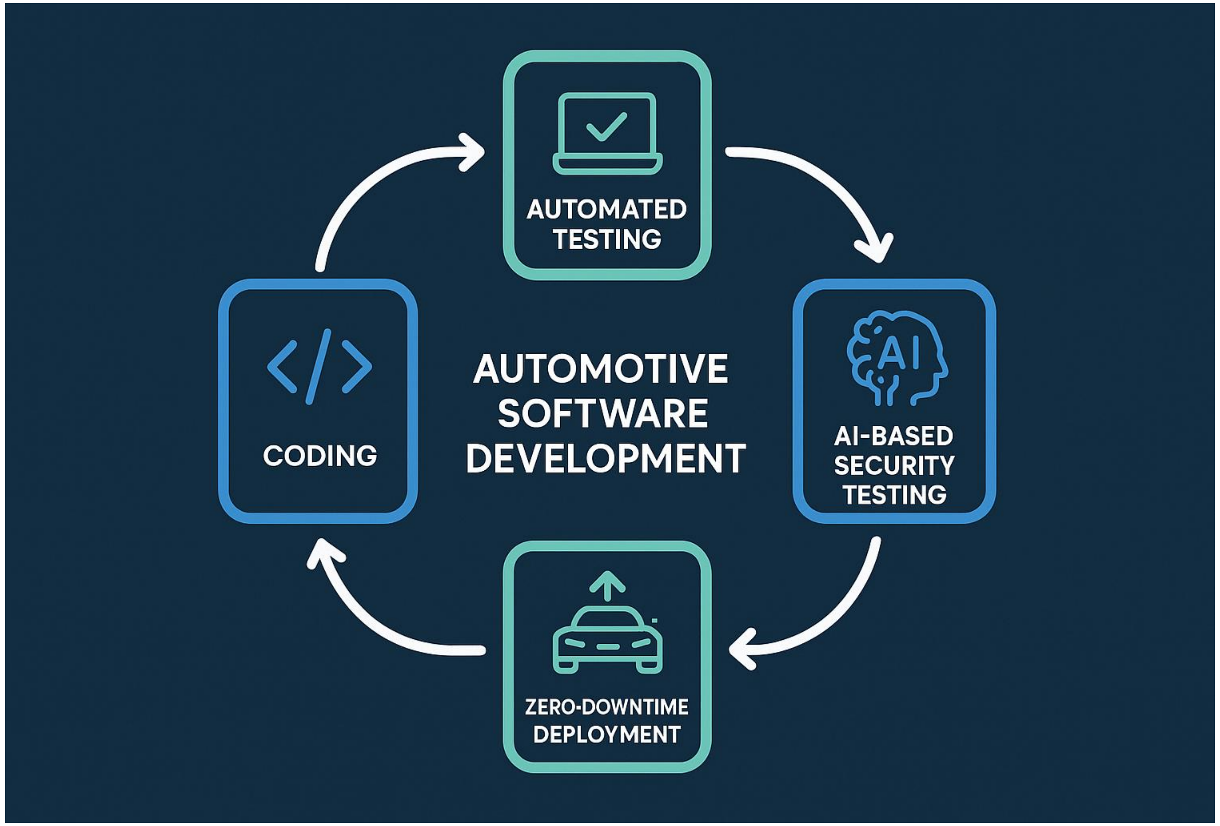
- AI-assisted white-box fuzzing for automotive targets

Pipeline Integration

- Embed continuous fuzzing into existing CI/CD/CT

Artifact & Impact Automation

- Auto-generate test cases, reports, quality matrix
- Measure coverage, MTTV, and CI latency vs. baseline



Expected Outcomes

AI Generated fuzz test cases and intelligent mutation

Automation of test artifact generation

Integration into the existing CI/CD pipeline

AI-Driven Security Testing

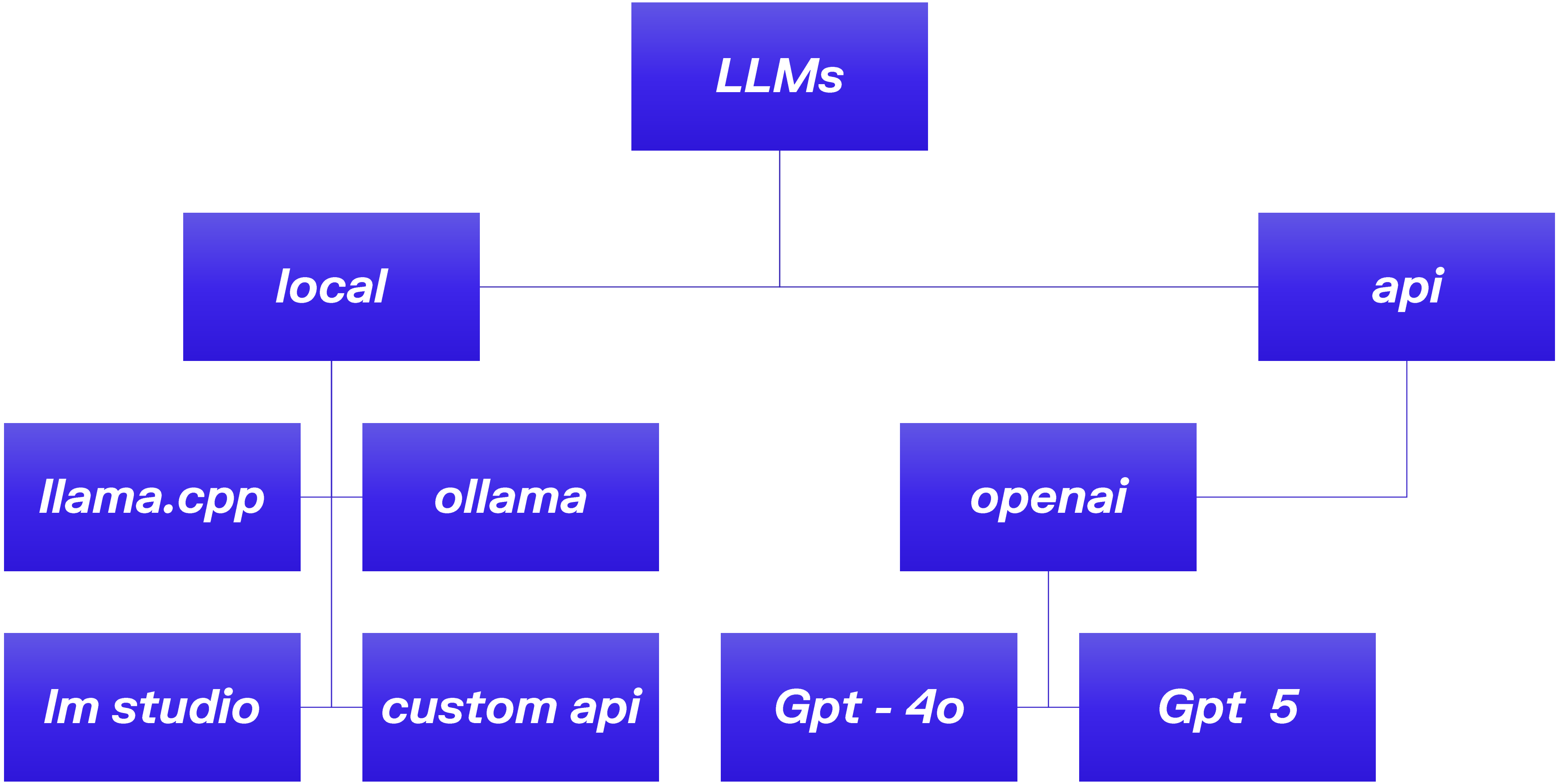
May

Method

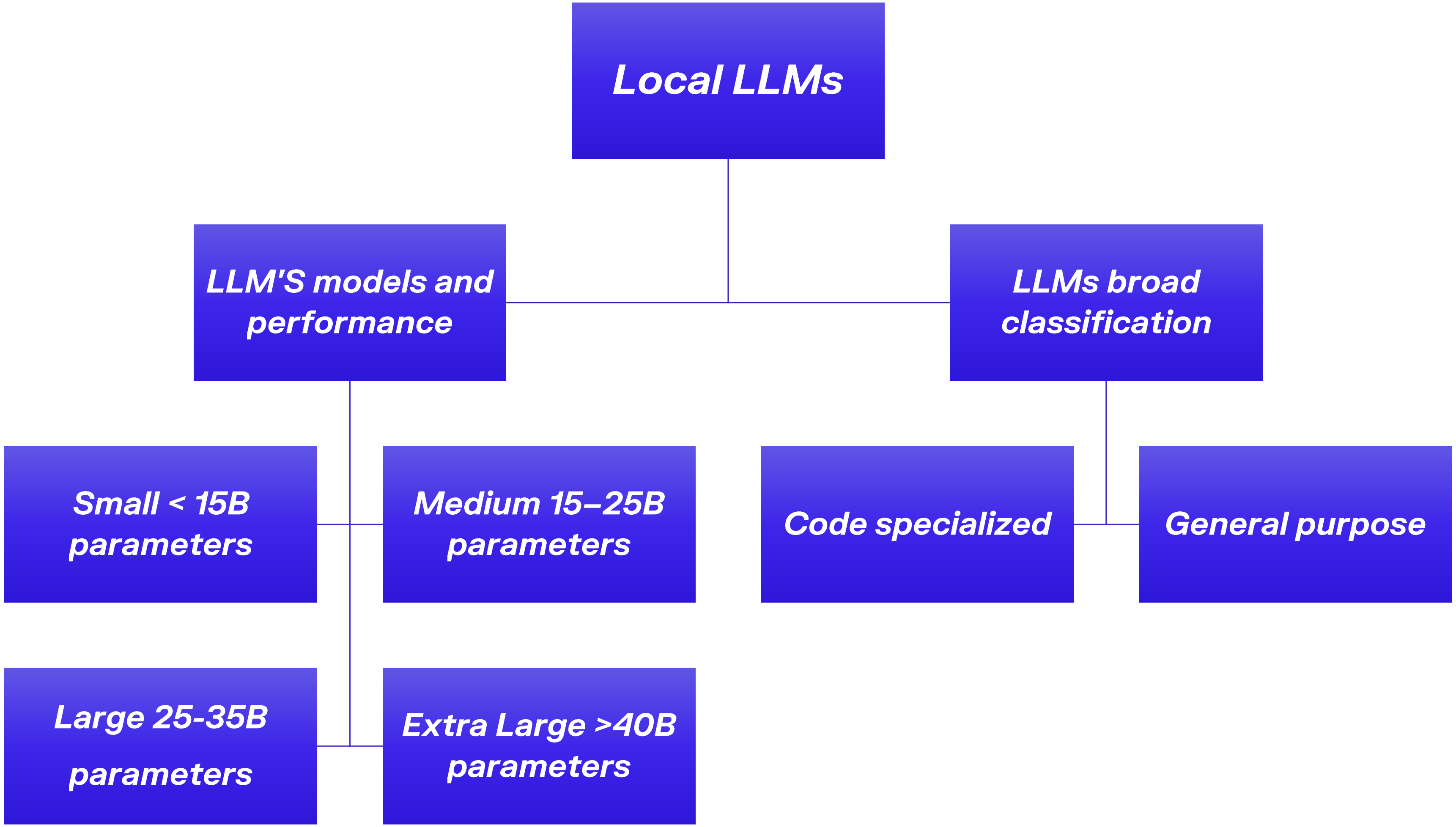
Overview

White box Fuzzing → AI-Integration → Analysis and Evaluation → CI/CD Integration → Artifact Generation → Documentation

Approach


LLMs
local
api
llama.cpp
ollama
openai
lm studio
custom api
Gpt - 4o
Gpt 5

Local LLMs


Local LLMs
LLM'S models and performance
LLMs broad classification
Small < 15B parameters
Medium 15–25B parameters
Large 25-35B parameters
Extra Large >40B parameters
Code specialized
General purpose

Models

Small (<15B parameters)

- phi4:14b (14B)
- llama3:latest (8B)
- qwen2.5-coder:1.5b (1.54B)
- qwen2.5-coder:7b (7B)
- qwen2.5-coder:14b (14B)

Medium (15–25B parameters)

- magistral:24b (24B)
- devstral:latest (24B)
- starcoder2:15b (15B)

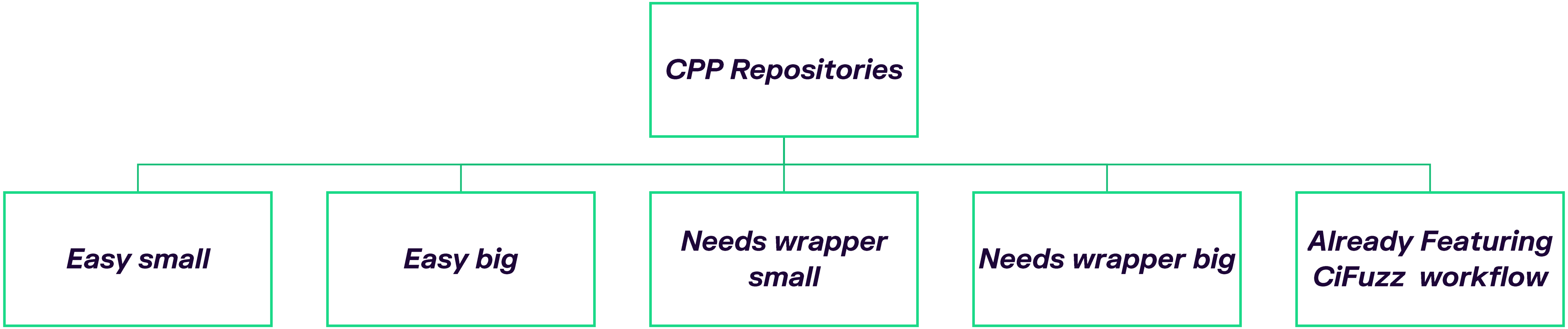
Large (25-35B parameters)

- gemma3:27b (27.4B)
- qwen3:32b (32.8B)
- deepseek-r1:32b (32.8B)
- qwen2.5-coder:32b (32.8B)
- deepseek-coder:33b (33B)
- wizardcoder:33b (33B)
- codellama:34b-instruct (34B)
- yi:34b (34B)

Extra Large (> 40B parameters)

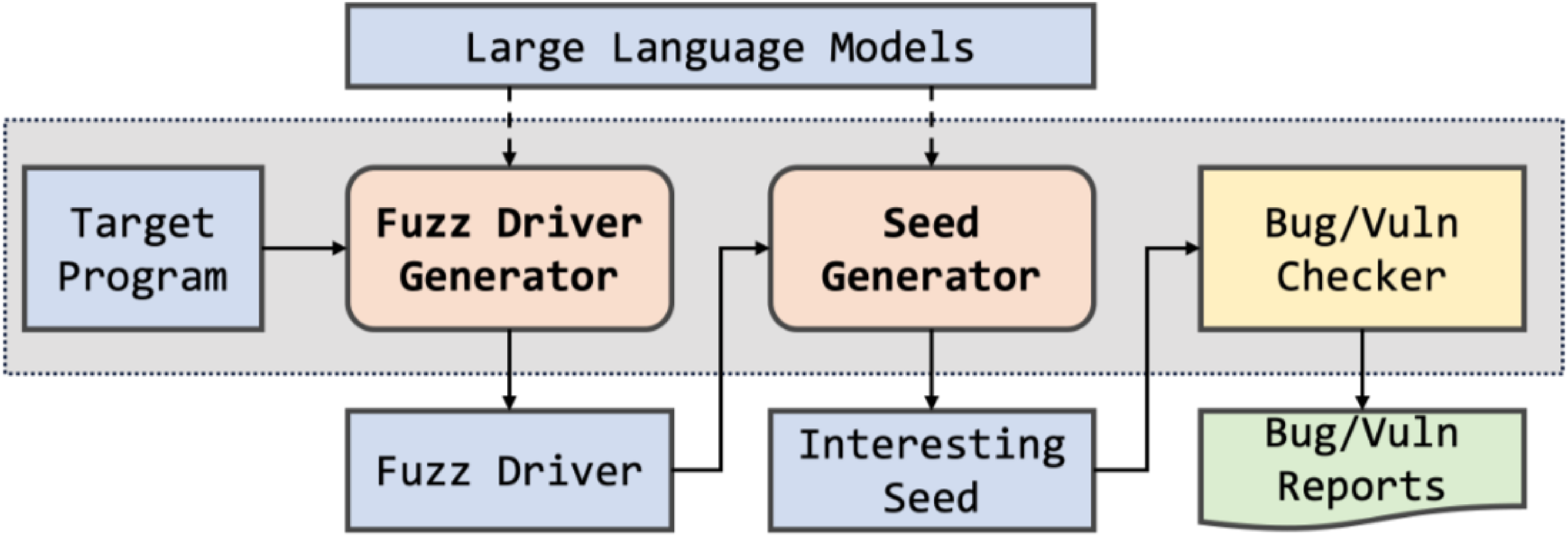
- mixtral:latest (46.7B)

Repositories

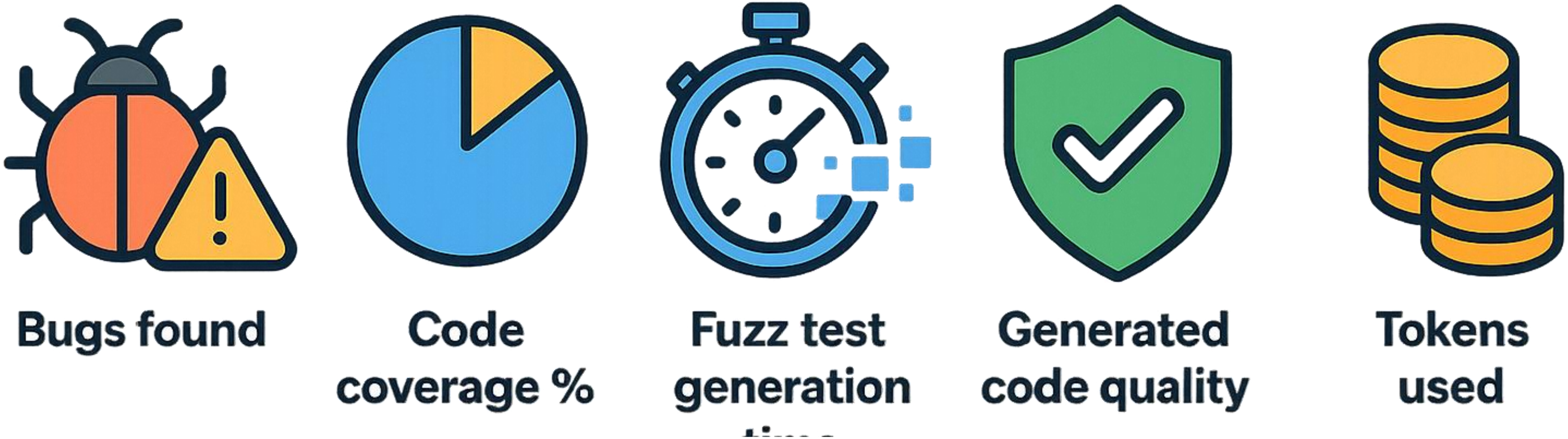


repositories

Working


Large Language Models
Target Program
Fuzz Driver Generator
Seed Generator
Bug/Vuln Checker
Fuzz Driver
Interesting Seed
Bug/Vuln Reports

Evaluation Framework



Finetuning local llms

Qwen 2.5 coder 32b instruct full model 60gb

LoRA (Low-Rank Adaptation)

- LoRA rank (r): 16 – controls adaptation size
- LoRA alpha: 32 – scaling for adaptation
- Dropout: 0.1 – prevents overfitting
- Target modules: q_proj, v_proj, etc. – efficient fine-tuning
- Device: auto – runs on best available hardware
- Dtype: float16 – faster, less memory
- Efficient model loading and saving (safetensors)

Code examples for training

- Jsonl format

{"prompt": "Generate a comprehensive parser fuzzer for structured data\n\nRequirements:\n- Maximize code coverage and edge case detection\n- Generate intelligent, targeted inputs\n- Optimize for vulnerability discovery speed\n- Include comprehensive error handling„

Code examples for training

Qwen 2.5 coder 32b

```
"completion":
"void train_timing_attack(const uint8_t* data, size_t size) {\n    if (size < 20) return;\n    \n    const char* secret = \"supersecretpassword123\";\n    std::string user_input(reinterpret_cast<const char*>(data), \n                           std::min(size, strlen(secret)));\n    \n    auto start = std::chrono::high_resolution_clock::now();\n    \n    // VULNERABLE: Early return creates timing difference\n    bool matches = true;\n    for (size_t i = 0; i < strlen(secret) && i < user_input.length(); i++) {\n        if (secret[i] != user_input[i]) {\n            matches = false;\n            break;  // Early return reveals timing information\n        }\n        // Simulate processing time\n    }\n    \n    auto end = std::chrono::high_resolution_clock::now();\n    auto duration = std::chrono::duration_cast<std::chrono::microseconds>(end - start);\n    \n    printf(\"Password check took %ld microseconds, match: %s\\n\", \n           duration.count(), matches ? \"true\" : \"false\");\n}"}
```

Finetuning

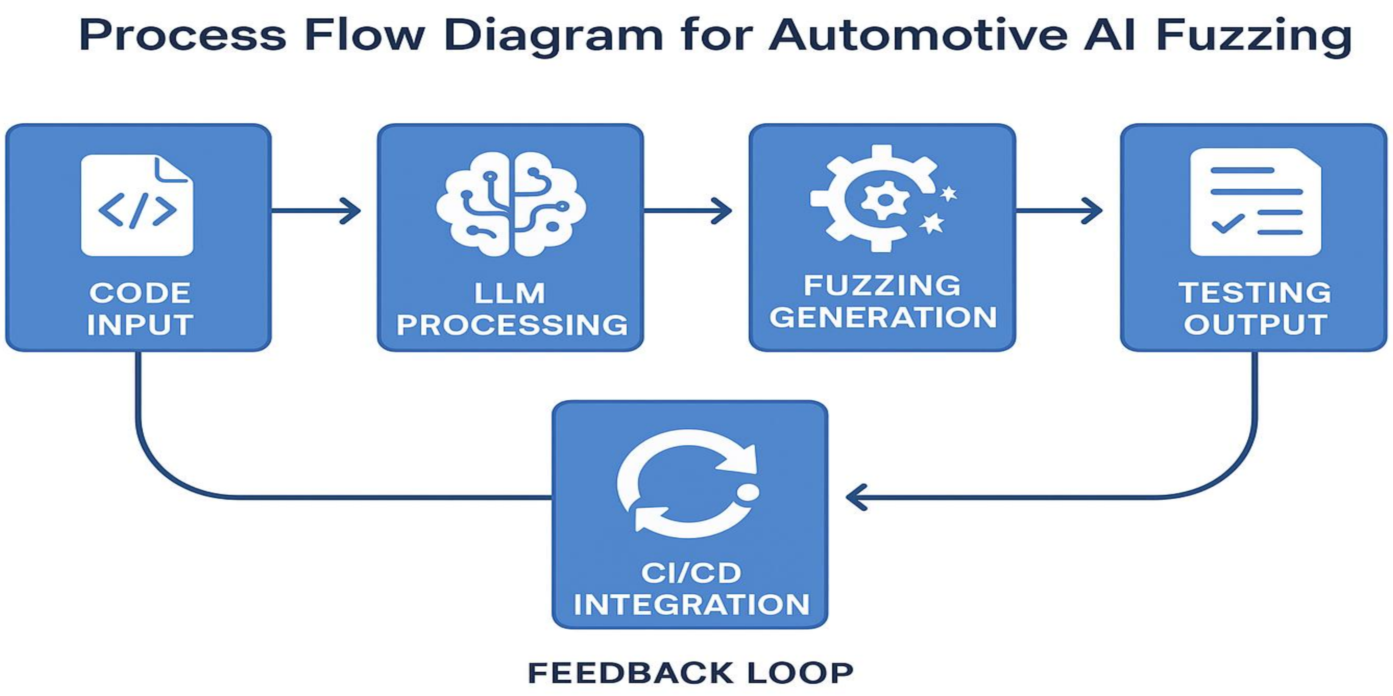
32b – needs more than 60gb ram

14b – (32 – 37gb) ram

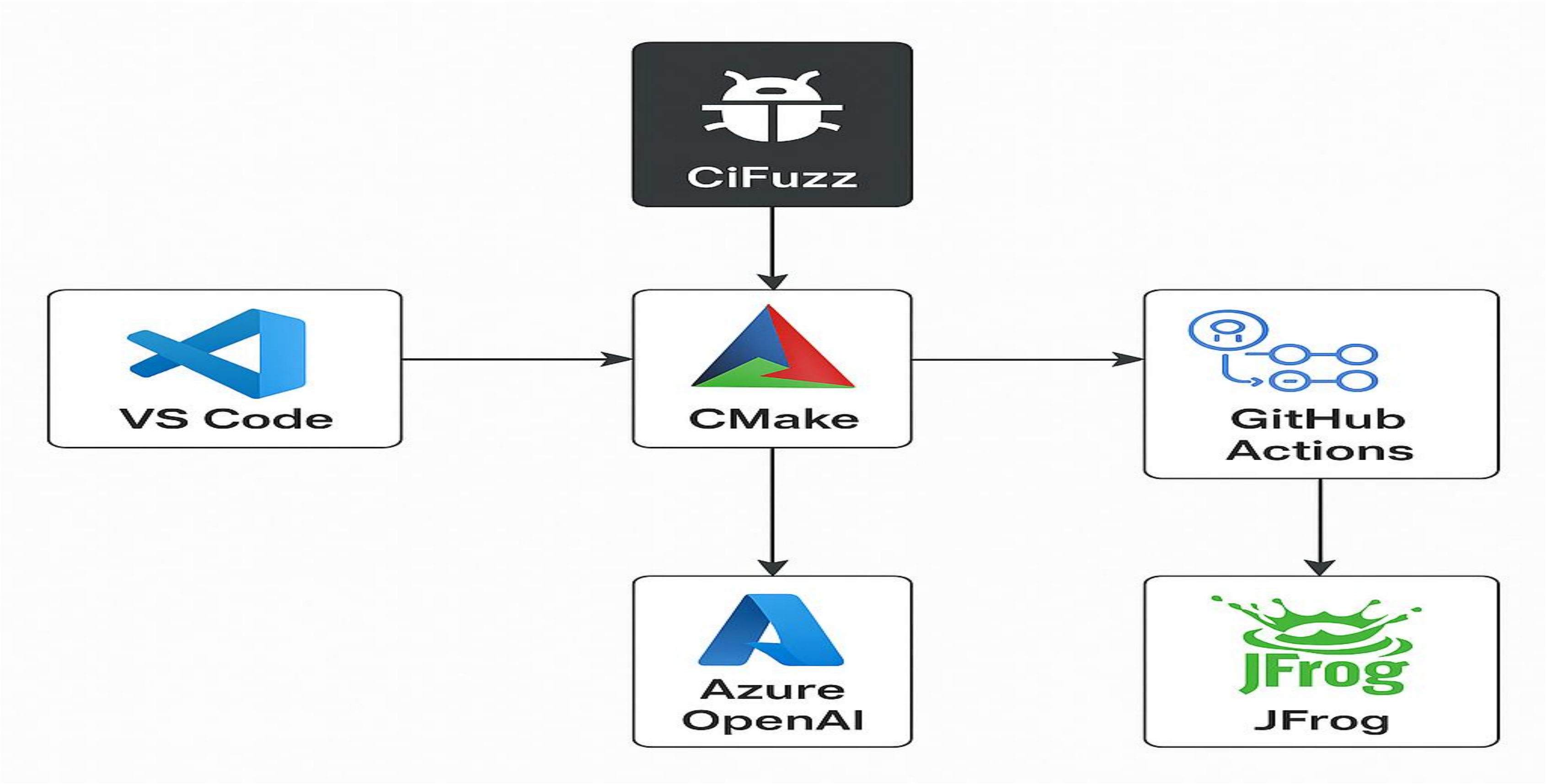
7b – 24gb ram without any background process

1.5b – 14gb ram

CI/CD Process flow



CI/CD Github Actions



buildah

Result

Models - Successful llms

Code intelligence advanced setup

Models	Code Coverage	Time Taken	No of tokens used	Unique test cases	Successful l fuzz tests
Phi 14b	100%	12m 8s	59.8k	0	0
Llama 3	100%	3m 41s	27.6k	0	0
Qwen 1.5 7b	89.47%	6m 9s	50k	0	0

Models - Successful llms

Yaml cpp (35 files, 1061 candidates)

Models	Code Coverage	Time Taken	No of tokens used	Unique test cases	Successfull fuzz tests
Qwen 2.5 coder 32b	43.08%	32m 57s	45.1k	2.04k	2
Gemma 3 27b	45.06%	33m 33s	40.2k	2.05k	2
Phi 14b	34.26%	36m 36s	71.5k	2.22k	1

Models - Unsuccessful llms

Yaml cpp (35 files, 1061 candidates)

Models	Code Coverage	Time Taken	No of tokens used	Unique test cases	Successfull fuzz tests
Codellama 32b	0.00%	50m 43s	74k	0	0
Deepseek r1	0.00%	1h 36m 53s	54.9k	0	0
Deepseek code	0.00%	1h 37m 39s	48.8k	0	0
devstral	0.00%	38m 57s	82.6k	0	0
Llama 3 7b	0.00%	27m 40s	268k	0	0
Starcoder 2 15	0.00%	19m 55s	114k	0	0
Wizardcoder	0.00%	32m 15s	32.5k	0	0
Yi 34b	0.00%	2h 39m 53s	74.9k	0	0
Magistral 24b	0.00%	-	-	0	0
Mixtral	0.00%	-	-	0	0

Models - UnSuccessful llms

fmt

Models	Code Coverage	Time Taken	No of tokens used	Unique test cases	Successful / fuzz tests
Qwen 2.5 coder 32b	0.00%	43m 20s	59.8k	0	0
Gemma 3 27b	0.00%	41m 59s	73.9k	0	0

Models - Successful llms

pugixml

Models	Code Coverage	Time Taken	No of tokens used	Unique test cases	Successful / fuzz tests
Qwen 2.5 coder 32b	34.77%	37m 24s	43.1k	2.5k	1
Gemma 3 27b	0.00%	1h 12m 8s	122k	0	0

Models - Successful llms

Qwen 2.5 coder 32b

Repositories	Code Coverage	Time Taken	No of tokens used	Unique test cases	Successfull fuzz tests
jsoncons	45.64%	37m 24s	43.1k	2.5k	1
glm	0.00%	42m 55s	56.8k	0	0
spdlog	0.00%	49m 33s	62.9k	0	0

Finetuning

NAME	1	SIZE

- qwen2.5-coder:1.5b 986 MB
- qwen-fuzzer-1.5-709-examples:latest 3.1 GB
- qwen-fuzzer-1.5-172-examples:latest 3.1 GB

Finetuning - Successful llms

Yaml cpp (35 files, 1061 candidates)

Models	Code Coverage	Time Taken	No of tokens used	Unique test cases	Successful / fuzz tests
Qwen 2.5 coder 1.5b	same	15 m	112k		
172 examples	same	12 m	65k		
709 examples	same	10 m	50k		

Costs

Literature review

Key Focus Areas:

- AI techniques in software testing
- LLM-based fuzzing approaches
- CI/CD pipeline integration
- Automotive security applications

From Traditional AI to LLMs - The Technical Journey

Key Milestones:

- *NeuFuzz (2019): First neural network-guided fuzzing - 35% improvement*
- *Deep RL Fuzzing (2018): 2.3× more crashesthan AFL, 21 CVEs*
- *TitanFuzz (2022): Watershed moment - First LLM-based fuzzer*
- *Fuzz4All (2023):Universal multi-language fuzzing - 98 bugs found*

Performance Evolution:

- *2018-2020: Traditional ML approaches dominated (60%)*
- *2022-2025: LLM revolution with 50.84% coverage improvements*

The TitanFuzz Revolution & Beyond

TitanFuzz Impact (2022):

- *Zero-shot capability without explicit constraints*
- *30.38% higher coverage on TensorFlow*
- *50.84% higher coverage on PyTorch*
- *65 bugs discovered, 44 previously unknown*

Follow-up Advances:

- *HGFuzzer: 24.8× speedup over traditional approaches*
- *CKGFuzzer: Code knowledge graphs + LLMs*
- *G^2Fuzz: <$0.2 for 24-hour fuzzing campaigns*

Automotive Domain Applications

Safety-Critical Breakthroughs:

- SAFLITE: Autonomous systems fuzzing - 234.5% improvement
- CAN Bus AI Fuzzing: Real-time automotive network security
- ECG Embedded OS: 32 new vulnerabilities in embedded systems
- KernelGPT: 24 unknown bugs, 11 CVE assignments

Automotive Applications:

- Neural network validation for autonomous driving
- ECU firmware testing with real-time constraints
- Multi-ECU system integration testing

Research Gaps & Critical Findings

Identified Gaps:

- Limited automotive-specific research - Most work targets general software
- Real-time constraint handling - Insufficient automotive timing requirements
- Multi-ECU testing - Lack of distributed architecture approaches
- Standardization - No safety-critical evaluation frameworks

Quantitative Evidence:

- 20-90% consistent performance gains across all AI approaches
- 2-24× speed improvements in specialized scenarios
- Hundreds of CVE discoveries with real-world impact
- Cost reduction: From $1000s to <$1 per campaign

Key Takeaways

Major Findings:

- AI-enhanced fuzzing consistently superior to traditional methods
- LLM-based approaches represent paradigm shift
- Industrial adoption accelerating with proven ROI

Future Research Priorities:

- Automotive-specific AI fuzzing frameworks
- Integration with safety standards (ISO 26262, ISO 21434)
- Real-time aware fuzzing techniques

Costs

Costs

Daily Token Usage	Input Tokens	Output Tokens	Daily Cost	Monthly Cost (22 working days)	Annual Cost
Light Usage	5,000	7,500	€0.28	€6.16	€73.92
Moderate Usage	15,000	22,500	€0.83	€18.26	€219.12
Heavy Usage	50,000	75,000	€2.75	€60.50	€726.00
Enterprise Usage	100,000	150,000	€5.50	€121.00	€1,452.00

Costs

Test Scenario	Tokens Consumed	Cost per Run	Runs per Day	Daily Total
Single code file fuzzing	2,000 in + 3,000 out	€0.11	10	€1.10
Module testing	8,000 in + 12,000 out	€0.44	5	€2.20
Full application scan	25,000 in + 35,000 out	€1.30	2	€2.60
CI/CD pipeline integration	15,000 in + 20,000 out	€0.75	8	€6.00

Rapidjson example

Conclusion

This thesis focuses on integrating AI and LLMs into CI/CD/CT pipelines to improve the security testing of automotive software

Timeline

01 Literature Review and Familarizing with softwares

02 Local llm fuzzing and Benchmarking with Evaluation metrics

03 Open source llm training and cloud llm testing (OpenAi)

04 Automation of llm fuzzing locally and adaption to the production code

05 LLM Fuzzing in the ci/cd pipeline with jfrog and azure endpoints in github
.

06 Writing Thesis

Vielen Dank
für Ihre Aufmerksamkeit!

Any Questions