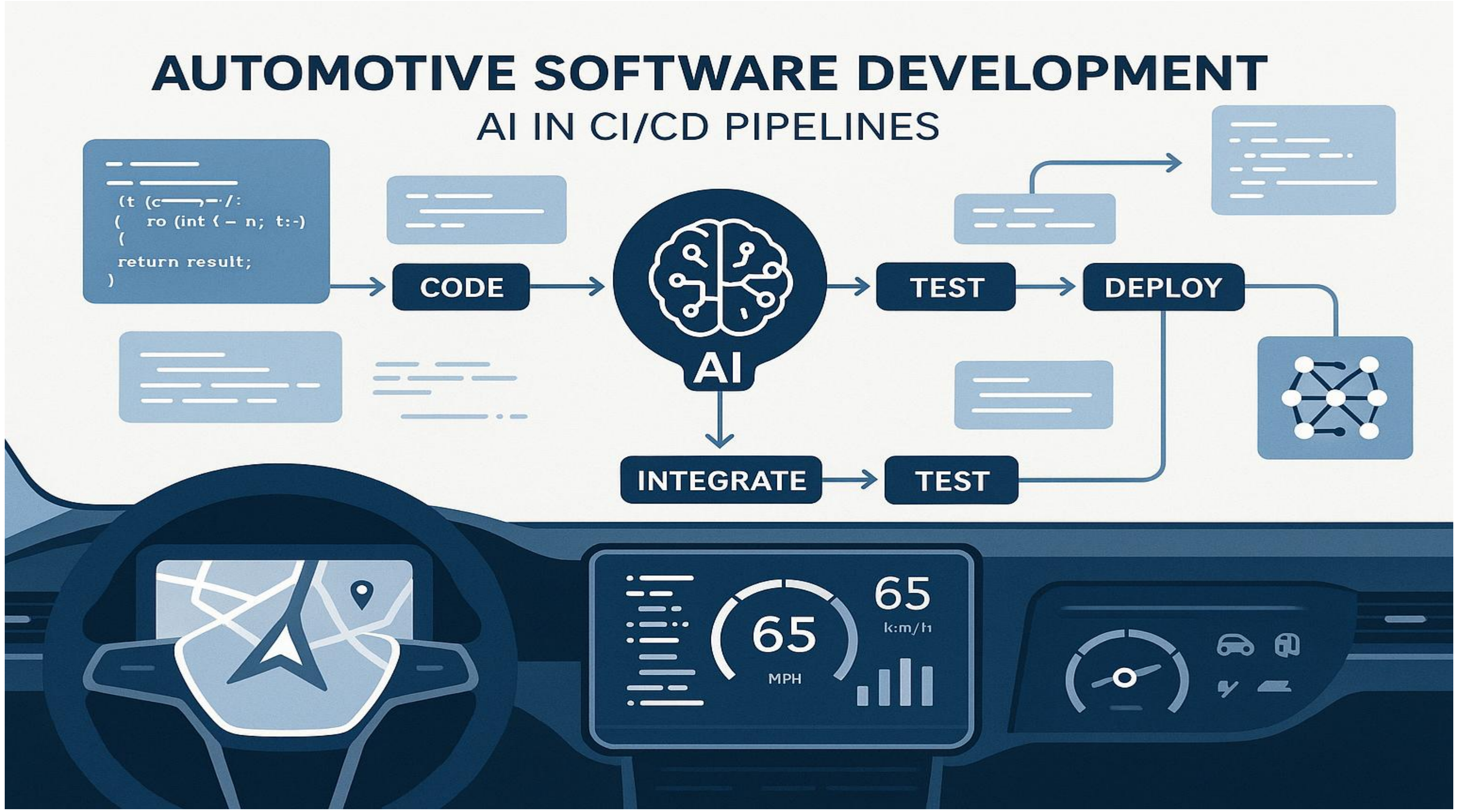
Master Thesis: AI Usage in CI/CD/CT Pipelines for Compute Platforms in Automotive

Morris Darren Babu
Friedrich-Alexander-Universität Erlangen-Nürnberg, Hardware-Software-Co-Design
Sept 22, 2025

Agenda

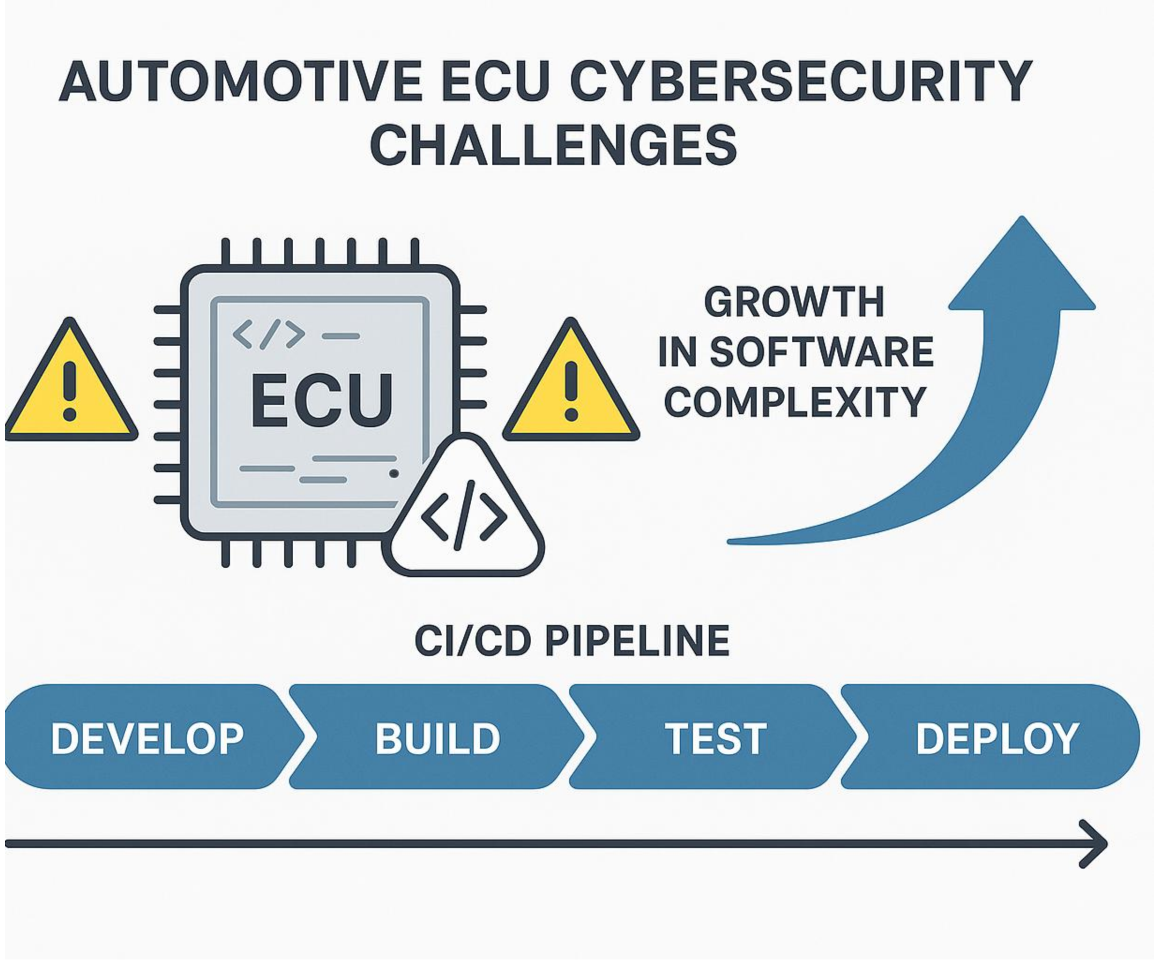


AUTOMOTIVE SOFTWARE DEVELOPMENT
AI IN CI/CD PIPELINES
CODE
AI
TEST
DEPLOY
INTEGRATE
TEST
return result;
65
MPH
65
k:m/h

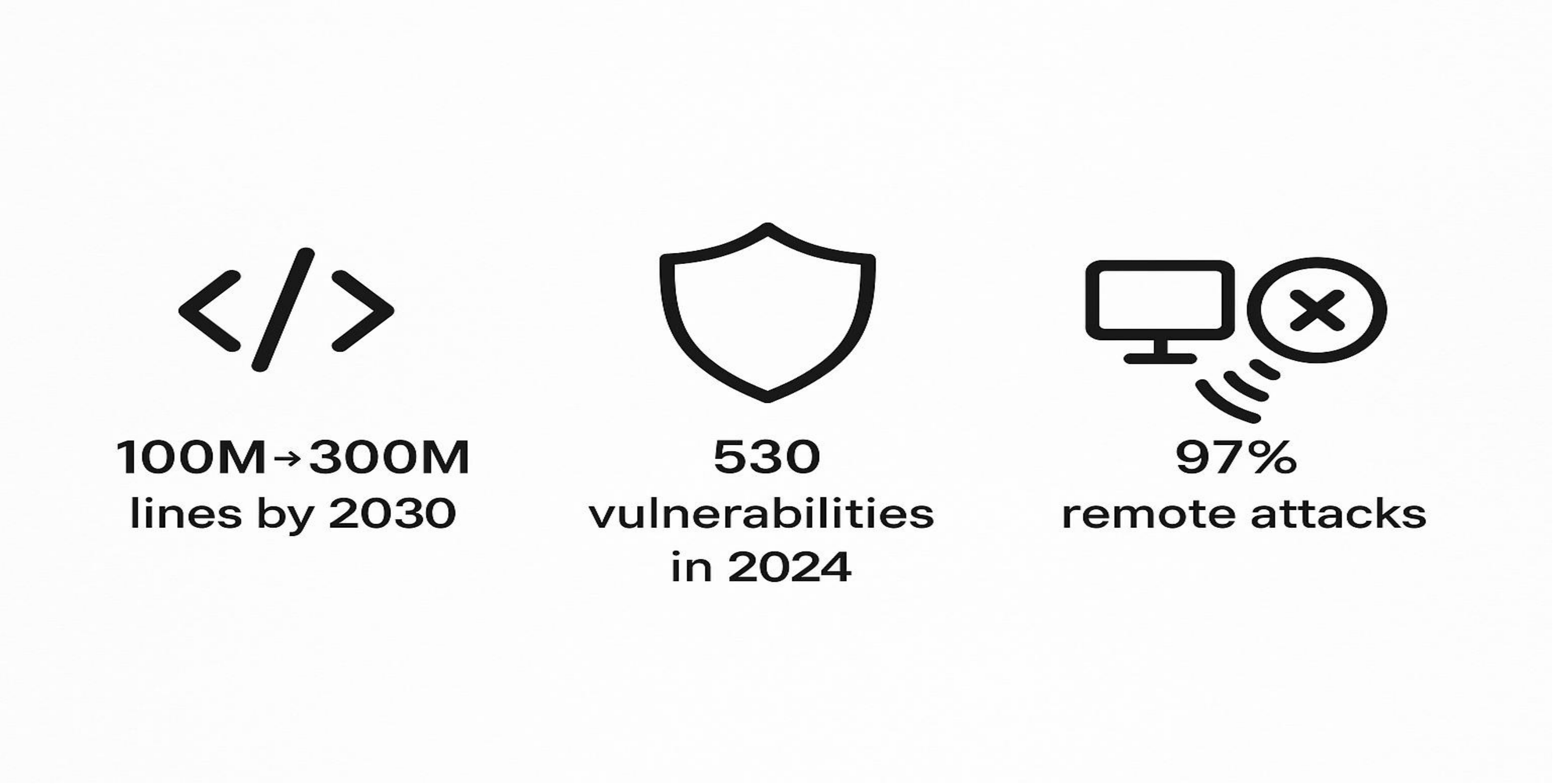
Introduction & Problem Statement

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



The Challenges

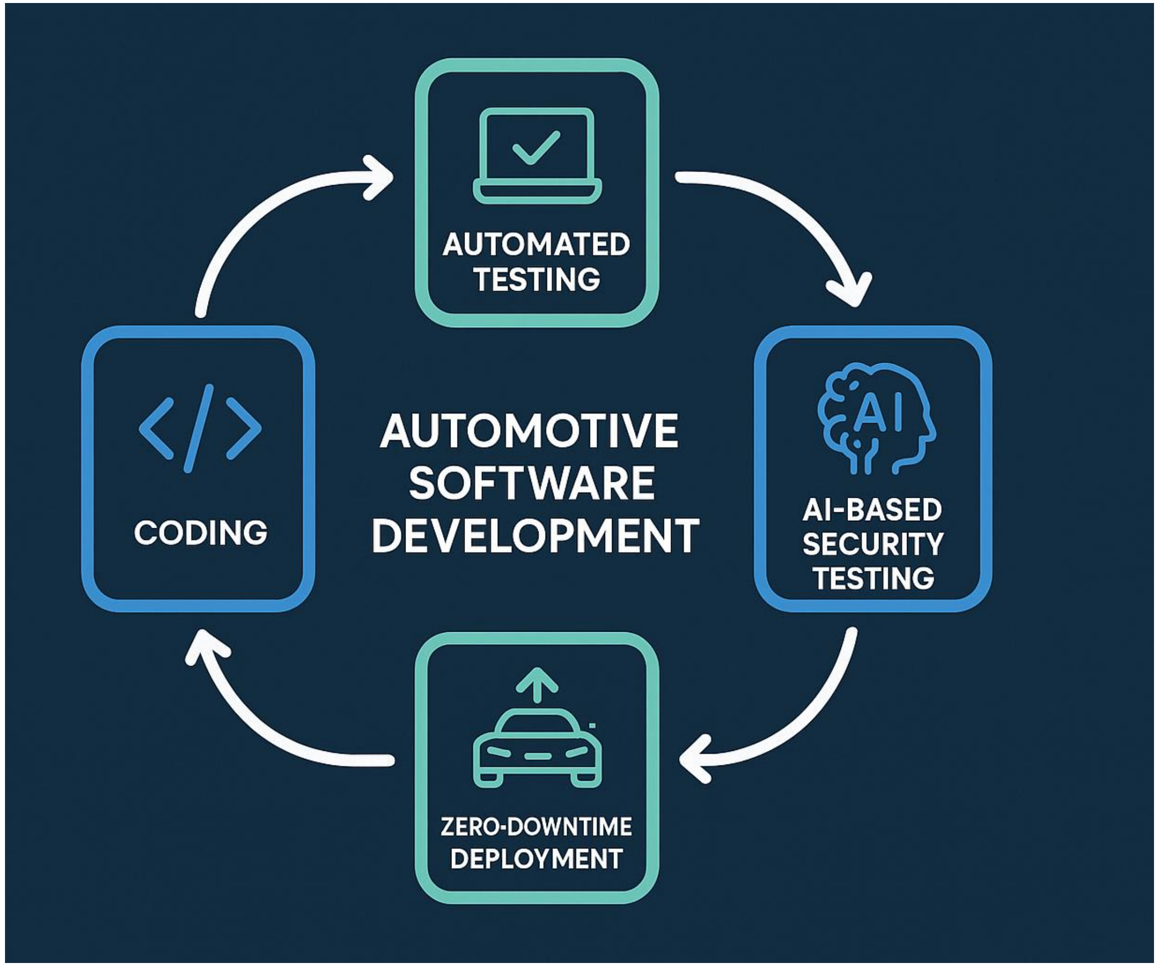


- Current white-box fuzzing & testing are manual or slow to scale
- Vulnerabilities slip through nightly CI builds due to time constraints
- Manual fuzz driver creation requires deep expertise - bottleneck for development teams
- Growing attack surface with connected and autonomous vehicle features

Research Objectives

- **AI-assisted white-box fuzzing** for automotive compute platforms
- **Novel security testing methods** integrating LLMs with fuzzing capabilities
- **Automated test artifact generation** - test cases, procedures, reports, quality matrices

- **Seamless CI/CD/CT integration** with existing automotive development workflows
- **Continuous fuzzing** embedded into daily development cycles
- **Zero-downtime deployment** of AI-enhanced security testing



- **Measure coverage, MTTV, and CI latency** vs. baseline traditional methods
- **Comprehensive cost-benefit analysis** for enterprise adoption
- **Real-world validation** within CARIAD's development environment

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

Research Questions

RQ1: Effectiveness of LLM-Generated Fuzz Drivers

Can Large Language Models generate effective fuzz drivers that achieve code coverage and vulnerability detection performance comparable to manually-written drivers?

RQ2: Model Selection and Optimization

Which LLM architectures, training approaches, and optimization techniques are most effective for automotive fuzzing applications?

RQ3: CI/CD Integration Feasibility

Can AI-driven fuzzing be integrated into automotive CI/CD/CT pipelines without significantly impacting development velocity or resource utilization?

RQ4: Automated Test Artifact Generation
Can LLMs automatically generate comprehensive test artifacts (test procedures, reports, quality matrices) that meet automotive development and compliance requirements?

RQ5: Continuous Learning and Adaptation
How can AI-driven fuzzing systems continuously improve their effectiveness through learning from fuzzing campaigns, bug discoveries, and developer feedback?

RQ6: Cost-Effectiveness and ROI
What are the economic implications of AI-enhanced fuzzing deployment, and under what conditions does it provide positive return on investment for automotive organizations?

Literature Review & State of the Art

Literature review with quantitative analysis

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 crashes**than 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

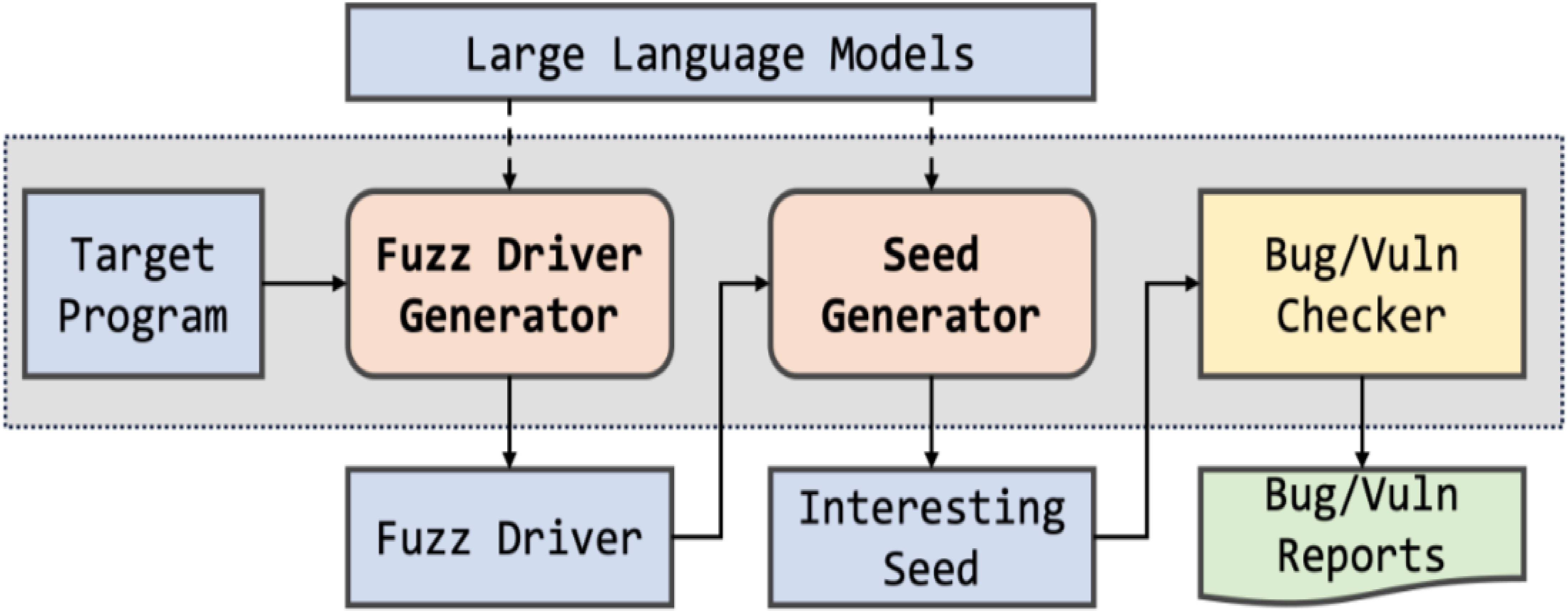
Future Research Priorities:

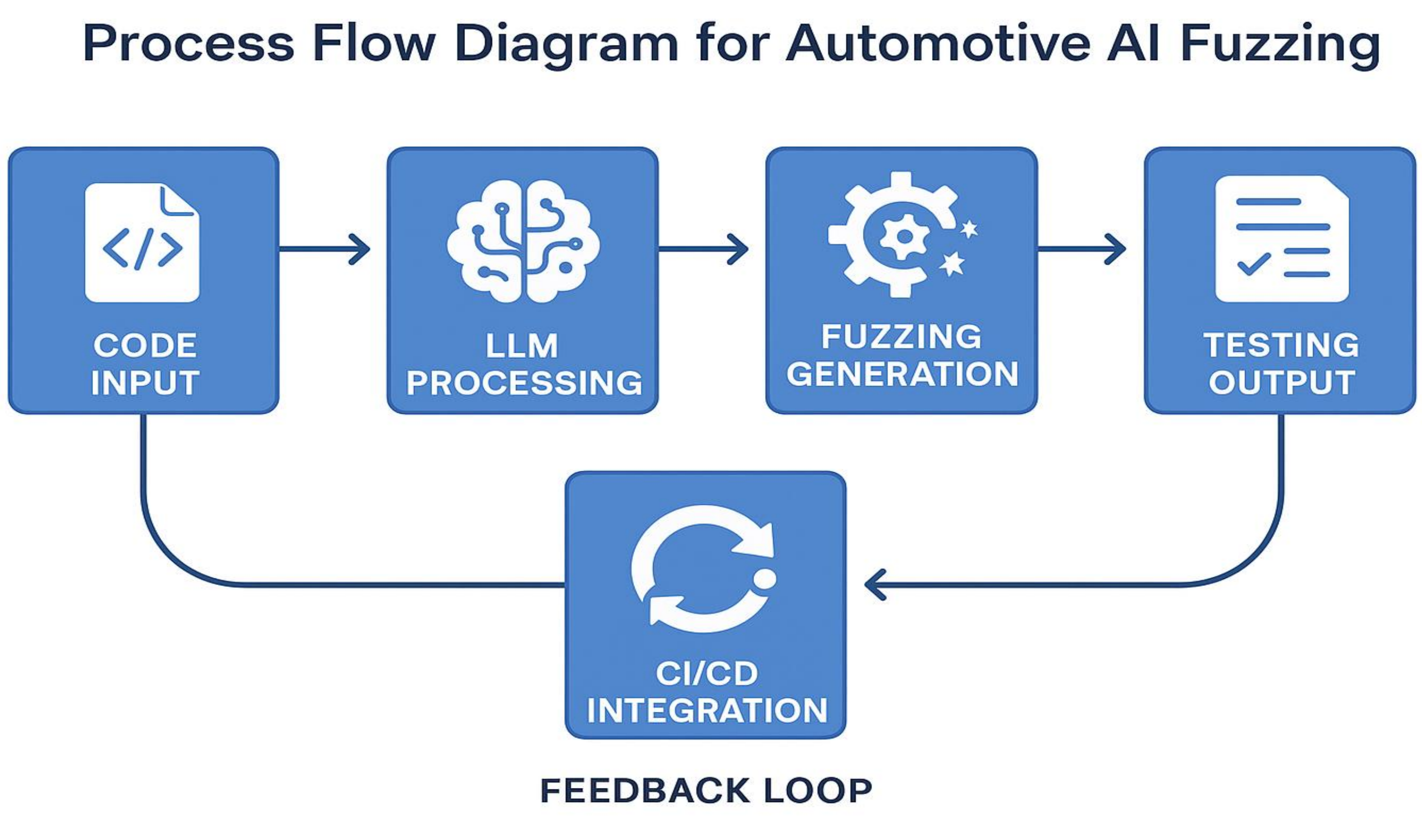
- Automotive-specific AI fuzzing frameworks
- Integration with safety standards (ISO 26262, ISO 21434)
- Real-time aware fuzzing techniques

Overview

Overview

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


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

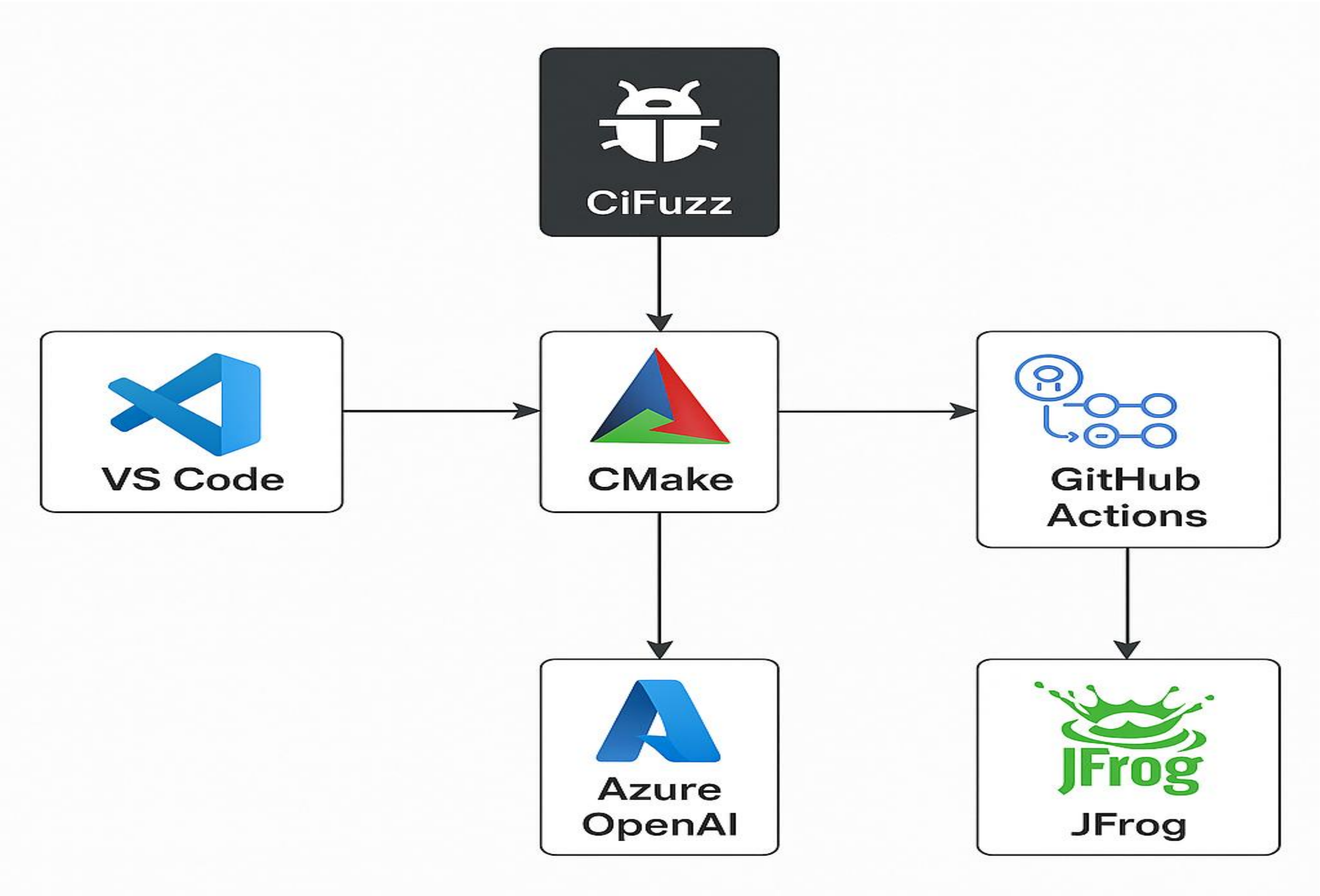


Process Flow Diagram for Automotive AI Fuzzing
CODE INPUT
LLM PROCESSING
FUZZING GENERATION
TESTING OUTPUT
CI/CD INTEGRATION
FEEDBACK LOOP

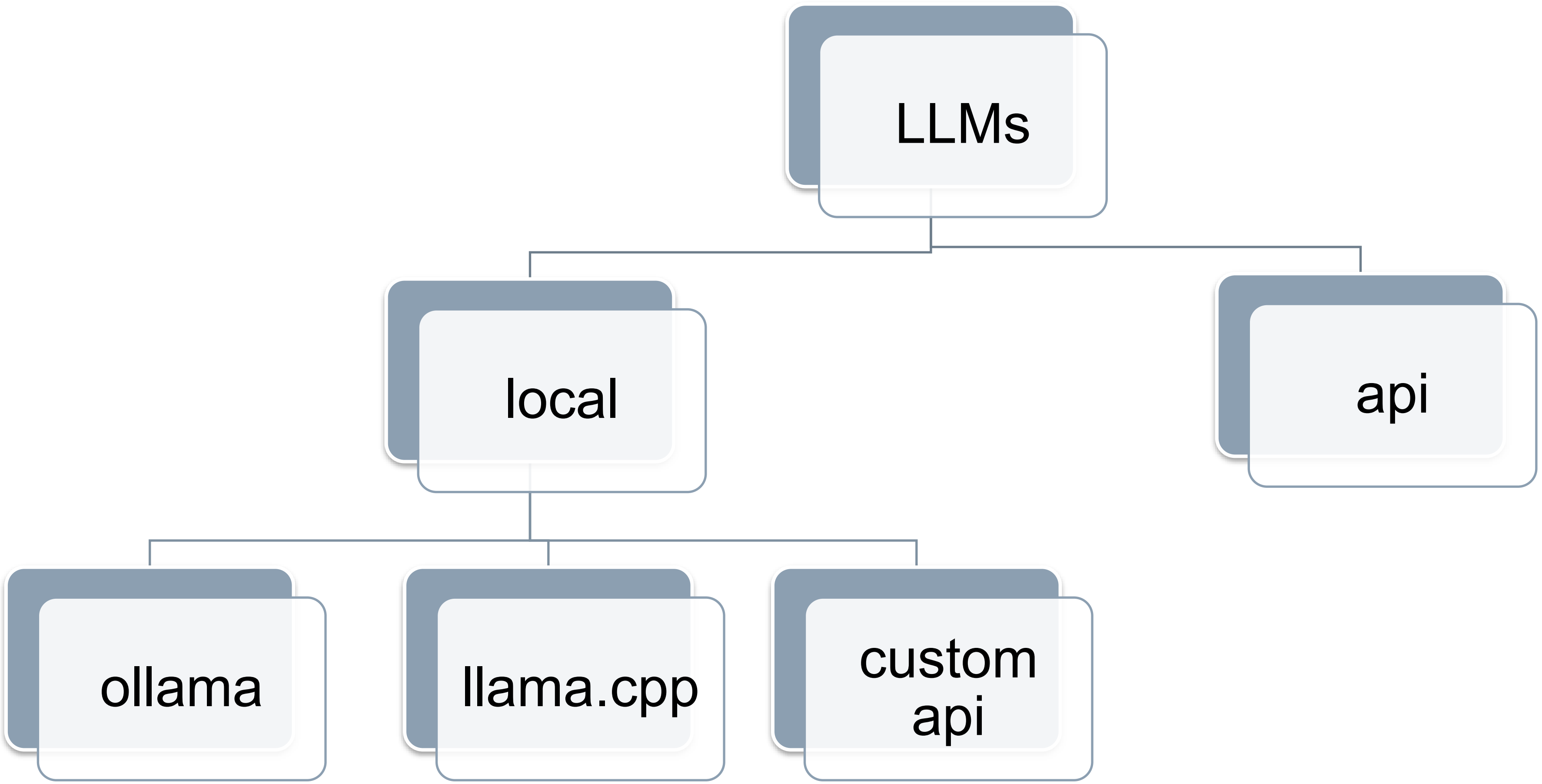
Tools used:

- Visual Studio Code
- CiFuzz (exe or docker image)
- Cmake Build systems and libfuzzer
- Azure Open AI endpoint
- Github Actions (CI/CD) and Jenkins
- Jfrog and Artifactory (docker images)

CI/CD Integration



Method

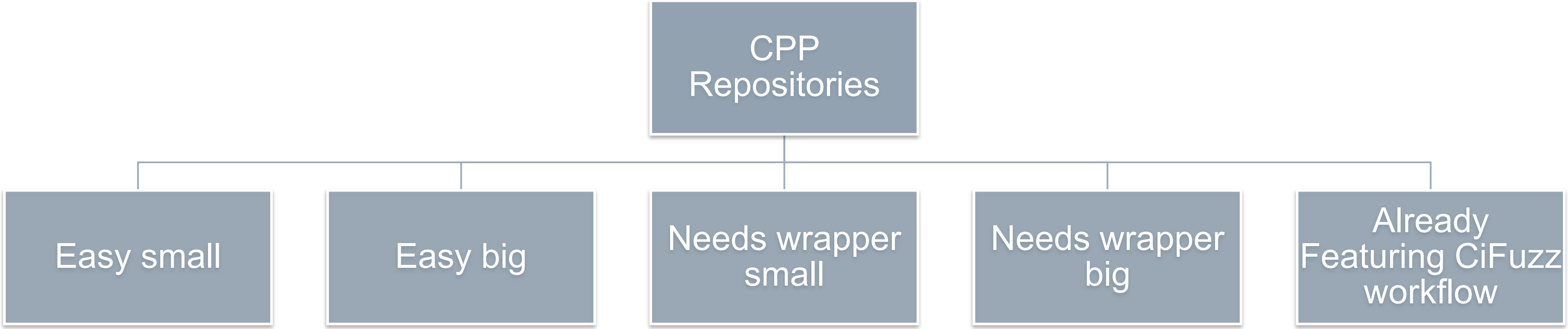

LLMs
local
api
ollama
llama.cpp
custom api

LLMs broad classification

- Code specialized
- General purpose

LLM’S models and performance

- Small < 15B parameters
- Medium 15 – 25B parameters
- Large 25 -35B parameters
- Extra Large >40B parameters


CPP Repositories
Easy small
Easy big
Needs wrapper small
Needs wrapper big
Already Featuring CiFuzz workflow

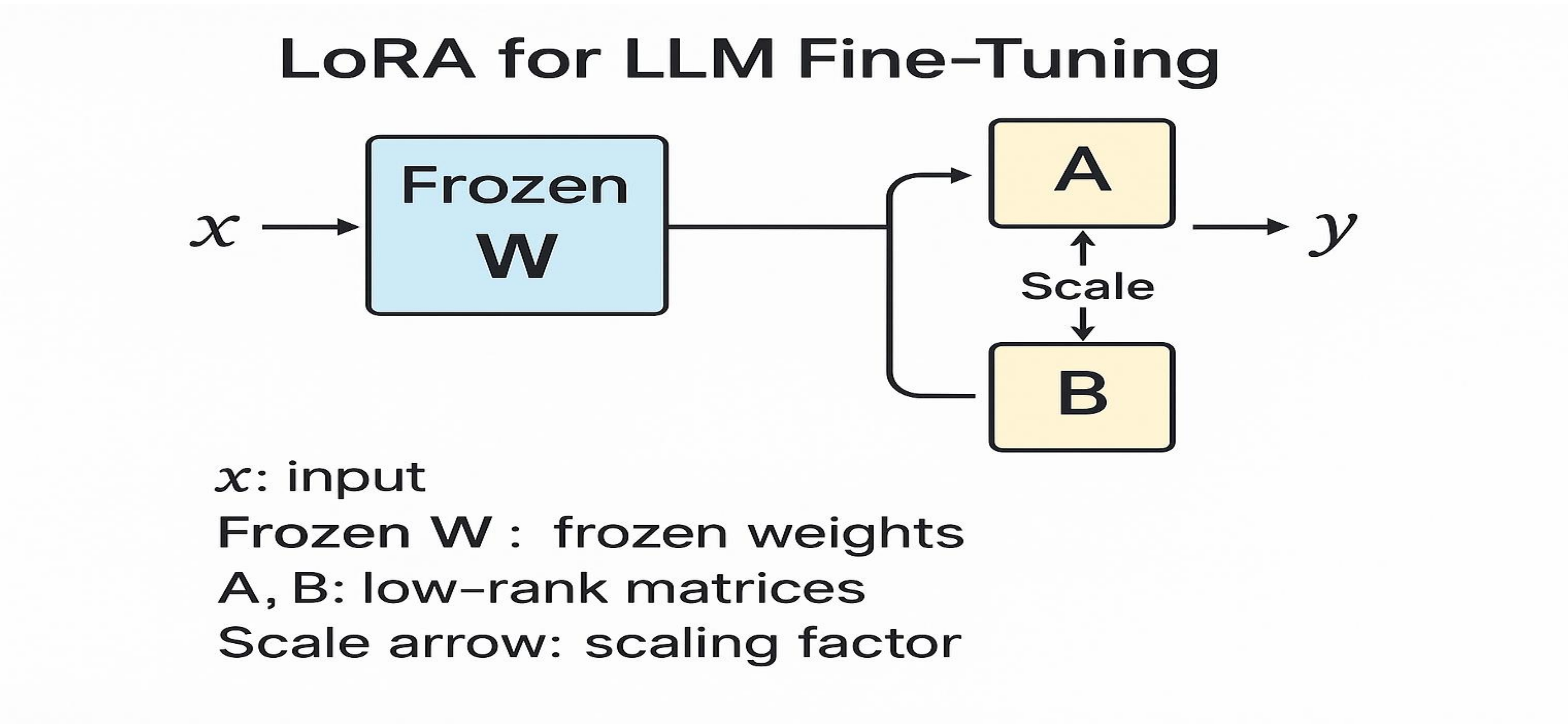
Evaluation

Metrics

Bugs found | **Code coverage %** | **Fuzz test generation time** | **Generated code quality** | **Tokens used**

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)


LoRA for LLM Fine-Tuning
x
Frozen
W
A
Scale
B
y
x: input
Frozen W : frozen weights
A, B: low-rank matrices
Scale arrow: scaling factor

To train open source LLMS requirements

- 32b – needs more than 60gb ram
- 14b – (32 – 37gb) ram
- 7b – 24gb ram without any background process
- 1.5b – 14gb ram

NAME	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

Conclusion

Key Achievements:

- Successfully demonstrated LLM-based fuzzing integration in CI/CD pipelines
- Achieved significant performance improvements over traditional methods
- Developed cost-effective solution using LoRA fine-tuning
- Created practical implementation within automotive constraints

Technical Contributions:

- Novel AI-assisted white-box fuzzing for automotive platforms
- Seamless CI/CD/CT pipeline integration with zero-downtime deployment
- Automated test artifact generation using fine-tuned LLMs
- Comprehensive evaluation framework with multiple metrics

Impact & Validation:

- Real-world validation in CARIAD's development environment
- Demonstrated cost reduction from thousands to under $1 per campaign
- Enhanced security testing without impacting development velocity
- Established foundation for automotive-specific AI fuzzing frameworks

Future Research Directions:

- Integration with safety standards (ISO 26262, ISO 21434)
- Real-time constraint handling for automotive requirements
- Multi-ECU testing approaches for distributed architectures
- Standardization of safety-critical evaluation frameworks

„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!


AI
QUESTIONS?