LLMs for Trustworthy Software Engineering: Insights and Challenges

@ LADC/SBESC 2024

Marco Vieira

marco.vieira@charlotte.edu

UNIVERSITY OF NORTH CAROLINA CHARLOTTE

Open Positions @ UNC Charlotte

B ...

... d
thin

Wha

Bas
pro
inc
con
wri
don
this
and
life
for

ChatGPT 4o

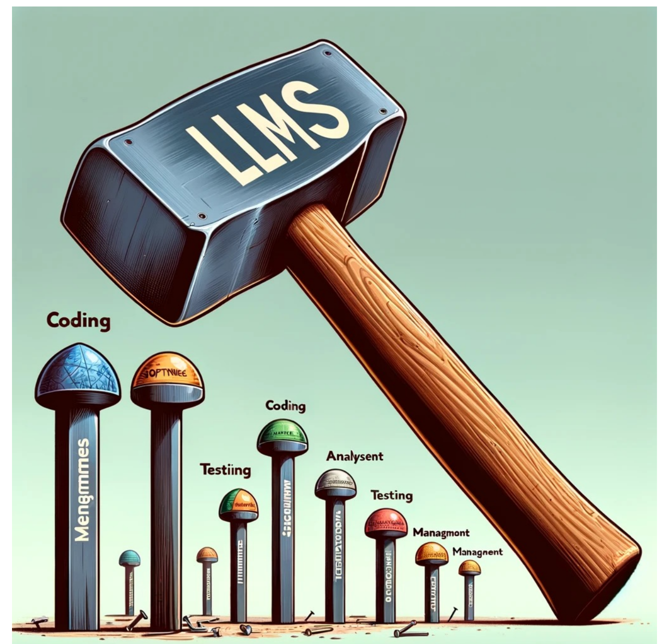
LLMs in Software Engineering

LLMs are already used in software engineering: but for isolated tasks!

Code: produce code

Analysis: detect patterns

Documentation: document code



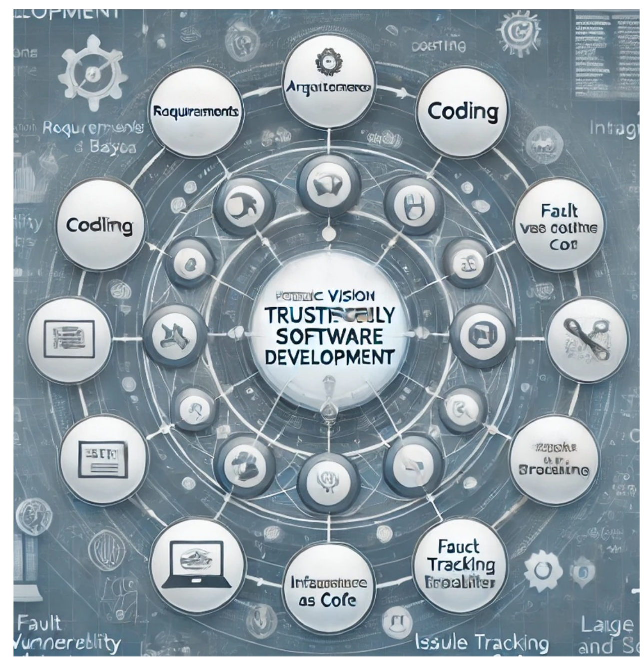
Vision: LLMs for Trustworthy Software

Need a holistic vision:

- Requirements, architecture, coding, analysis, testing, fault and vulnerability injection, IaC, issue tracking, monitoring, assessment…
- Focus on continuous improvement of trustworthiness properties

There are many challenges…

- Integration with existing practices
- Weaknesses and biases
- Lack of explainability
- Large-scale systems and legacy codebases
- Compliance with standards and regulations
- …



Outline

Starting with some basics…

- Software engineering
- Trust and trustworthiness
- LLMs

(Potential) role of LLMs in Trustworthy Software Engineering

- Design
- Development
- Deployment
- Assessment

Case: Benchmarking vulnerability detection and patching with LLMs

(Some of) the open challenges…

What is Software Engineering?

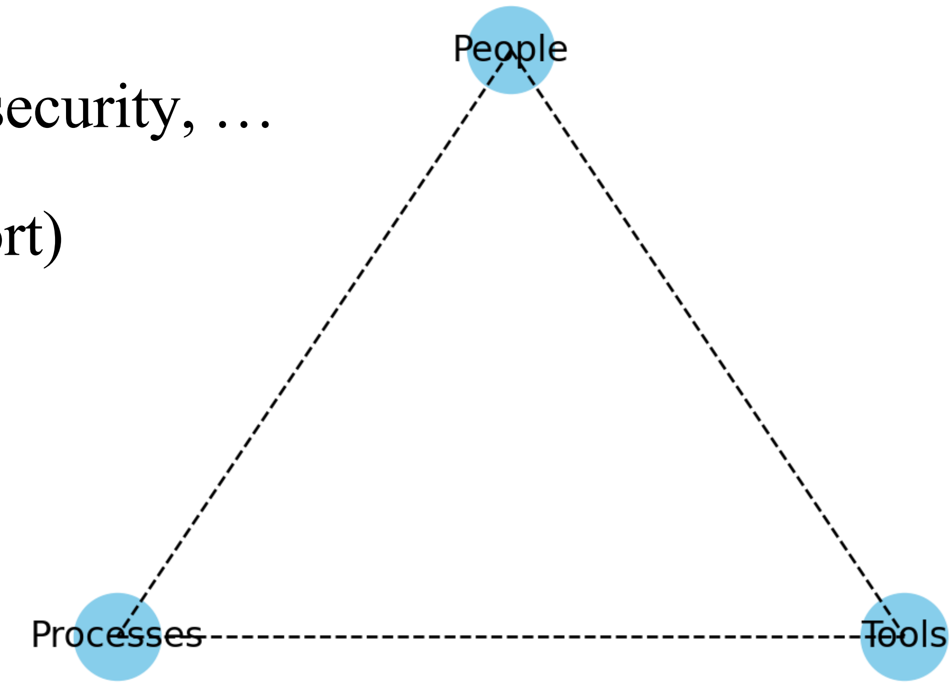
"*Software Engineering is the systematic application of engineering approaches to the development of software*"

Deliver software that meets user needs

Ensure reliability, maintainability, security, …

Optimize resources (time, cost, effort)

Processes + People + Tools



Software Development Lifecycle

Framework that defines the processes involved in developing software

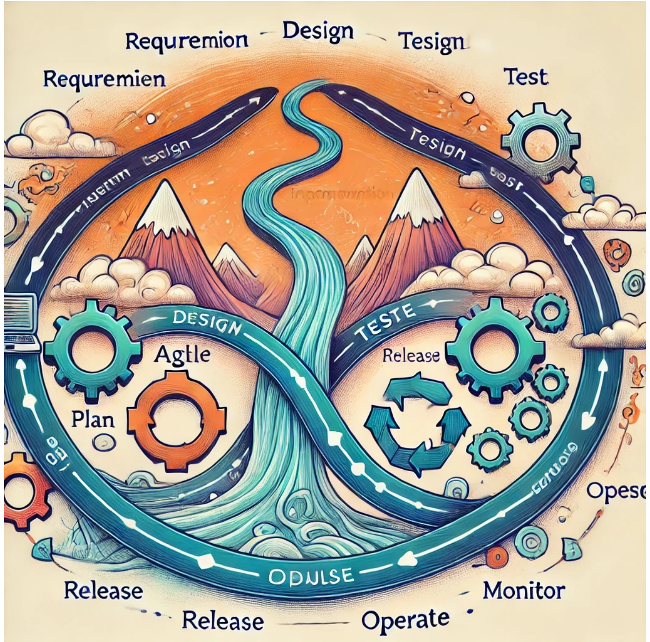
– From concept to deployment and maintenance

Popular lifecycle models:

– Waterfall: sequential phases
– Agile: iterative and incremental approach
– DevOps: continuous integration and delivery

Phases: requirement analysis, architecture design, implementation, testing, deployment, maintenance

3D: Design, Development, Deployment



Trust and Trustworthiness

Concepts broadly studied in many different areas

- Sociology, economics, psychology…

Human trust and trustworthiness

- Changes over time and can be highly subjective

Trust: Reliance on a system that it will exhibit the expected behavior

- Includes many perspectives!
- Trust level: estimated probability of this reliance

Trustworthiness: worthiness of a system for being trusted

- Assessed based on evidences
- Complex and potentially subjective!

Trustworthiness Properties

Trustworthiness is frequently seen as a security aspect

- It is trustworthy if it is secure!?

I consider it a more general notion!

- Even broader than dependability…

Requires identifying and evaluating all relevant measurable characteristics that may influence reliance

- Functional and non-functional

Security, privacy, reliability, performance, fairness, transparency, …

- Just define as needed!

LLMs

DNN for parsing and generating human-like text

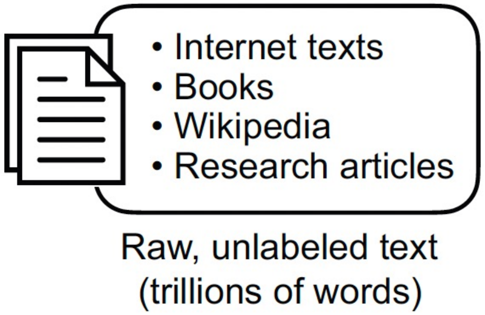


Figure from: Sebastian Raschka, Build a Large Language Model (From Scratch)

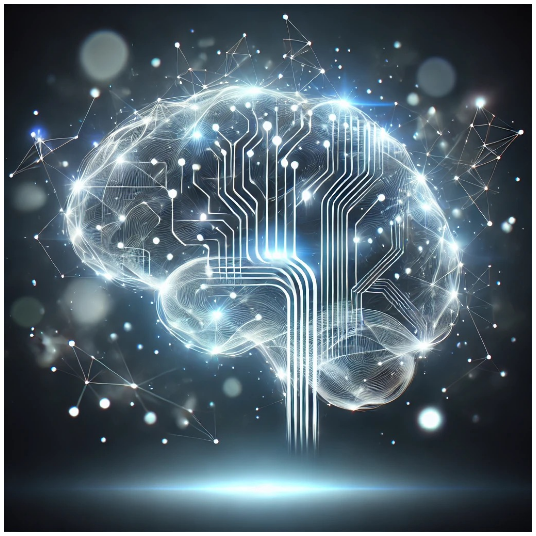
LLMs are Intelligent!

"*LLMs are intelligent systems capable of understanding and reasoning like humans*"

LLMs do not think or understand in the human sense!

– They generate outputs based on patterns in the data they were trained on

LLMs simulate understanding through pattern recognition and statistical modeling



They lack awareness, reasoning, or intent!

LLMs are Useless Hype!

"*LLMs are overhyped, unreliable, and impractical for real-world applications*"

While not perfect, LLMs are far from useless!

- Demonstrated value in numerous practical applications
- Code generation, content creation, and research...

LLMs are tools that require proper usage, oversight, and understanding of limitations

Truth Lies in Between…

LLMs are neither "intelligent" nor "useless hype"!

LLMs are powerful tools:

- Excel at pattern recognition and language generation
- Automating repetitive tasks, enhancing productivity, and assisting with creativity

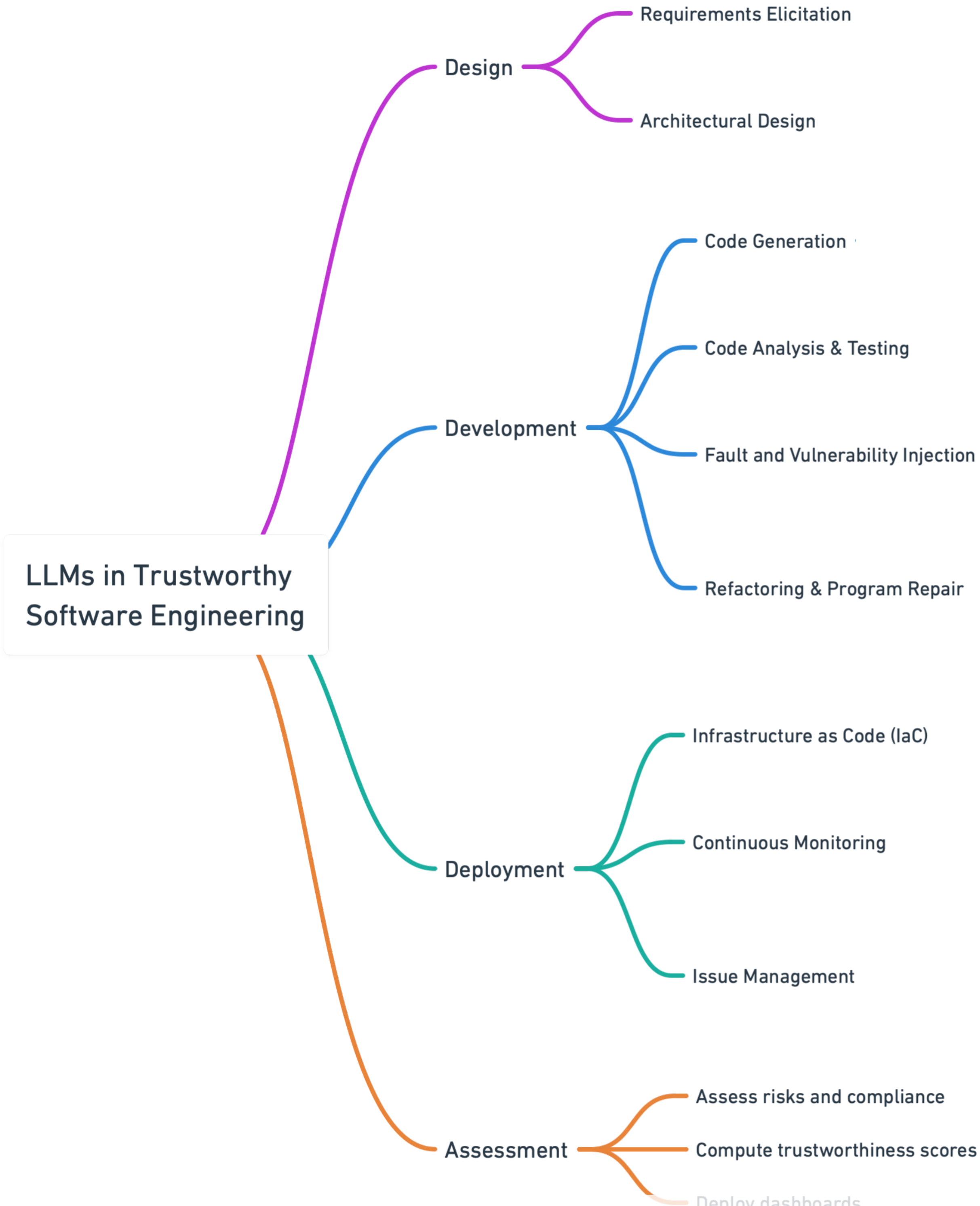
LLMs have limitations:

- Lack true understanding and reasoning
- Prone to generating incorrect or biased outputs

Advanced tools that require thoughtful use, validation, and oversight!

Design
Development
Deployment
Assessment

(POTENTIAL) ROLE OF LLMS IN TRUSTWORTHY SOFTWARE ENGINEERING


LLMs in Trustworthy Software Engineering
Design
Requirements Elicitation
Architectural Design
Development
Code Generation
Code Analysis & Testing
Fault and Vulnerability Injection
Refactoring & Program Repair
Deployment
Infrastructure as Code (IaC)
Continuous Monitoring
Issue Management
Assessment
Assess risks and compliance
Compute trustworthiness scores
Deploy dashboards

Requirements Elicitation

Traditionally a manual process:

- Interviews, document reviews, use-case development, …
- Time-intensive and error prone: particularly with non-functional requirements

How can LLMs help?

- Automates analysis of diverse sources: meeting transcriptions, user stories, regulatory documents, …
- Identify trustworthiness requirements early: embedding security, reliability, and privacy principles

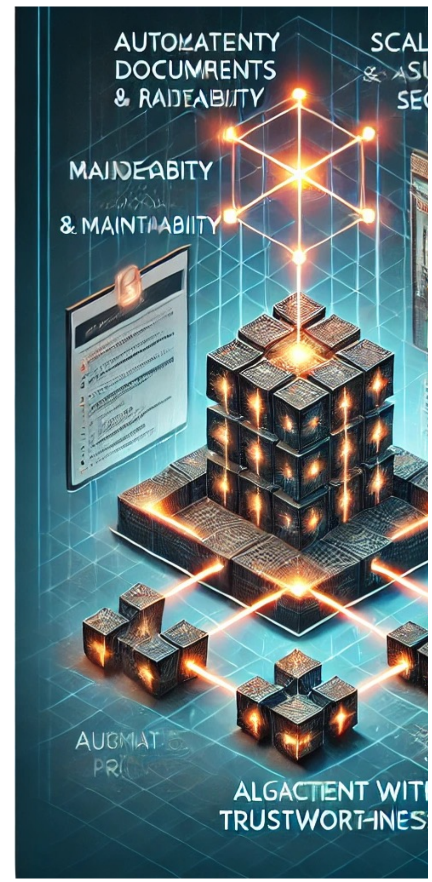
Comprehensive approach to capturing both functional and non-functional needs, ensuring trustworthiness from the start

Architecture Design

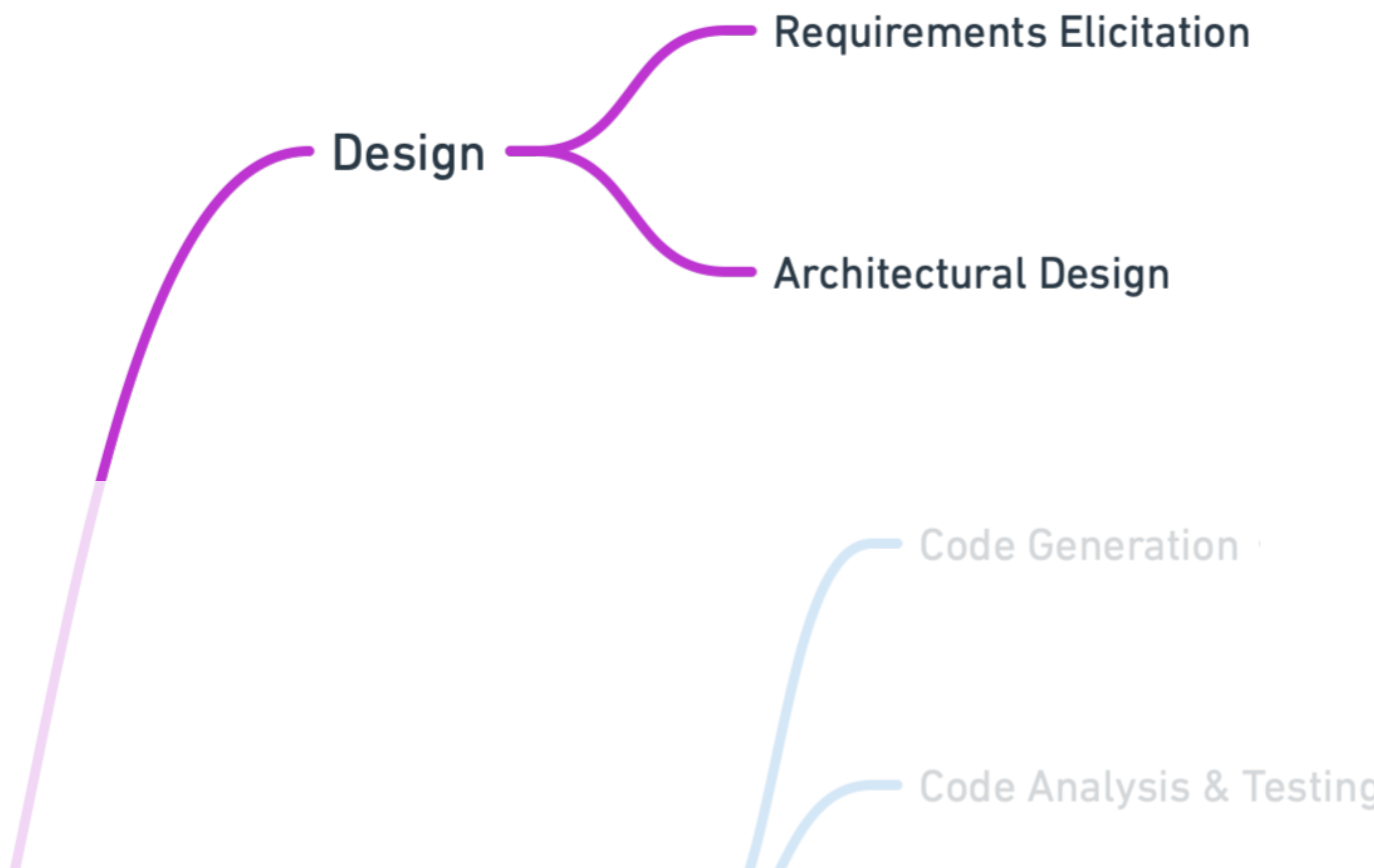
Traditional design focus on modularity, scalability, maintainability

How can LLMs help?

- Analyze requirements
- Suggest design patterns aligned with trustworthiness needs
- Assist in evaluating trade-offs: scalability vs. security
- Supports architects in making informed decisions



Design architectures that are inherently trustworthy, ensuring alignment with critical system requirements from the beginning


Design
Requirements Elicitation
Architectural Design
Code Generation
Code Analysis & Testing

Code Generation

Role of LLMs in code generation goes beyond syntax completion

- Focus on best practices and trustworthiness principles

How can LLMs help?

- Generating code from specifications: reduce development time
- Embed features like input validation to prevent vulnerabilities
- Incorporate proper error-handling mechanisms
- Recommend secure and reliable libraries or third-party APIs
- Ensures consistent use of trusted components
- Helps avoid introducing bugs or vulnerabilities inadvertently
- …

Code Generation: Example

Write a program which prints the central coordinate $(p_x, p_$
triangle which is constructed by three points (x_1, y_1), $(x_2,$

Input

Input consists of several datasets. In the first line, the numb

$x_1\ y_1\ x_2\ y_2\ x_3\ y_3$

in a line. All the input are real numbers.

Output

For each dataset, print p_x, p_y and r separated by a space in
decimals. Round off the solution to three decimal places.

Constraints

- $-100 \leq x_1, y_1, x_2, y_2, x_3, y_3 \leq 100$
- $n \leq 20$

Sample Input

```
1
0.0 0.0 2.0 0.0 2.0 2.0
```

Output for the Sample Input

```
1.000 1.000 1.414
```

```c
#include <stdio.h>
#include <math.h>

// Function to calculate the circumcircle
void calculateCircumcircle(double x1, double y1, double x2, double y2, double x3, double y3) {
    double D = 2 * (x1 * (y2 - y3) + x2 * (y3 - y1) + x3 * (y1 - y2));

    if (D == 0) {
        printf("The points are collinear, no circumcircle can be formed.\n");
        return;
    }

    double px = ((x1 * x1 + y1 * y1) * (y2 - y3) +
                 (x2 * x2 + y2 * y2) * (y3 - y1) +
                 (x3 * x3 + y3 * y3) * (y1 - y2)) / D;

    double py = ((x1 * x1 + y1 * y1) * (x3 - x2) +
                 (x2 * x2 + y2 * y2) * (x1 - x3) +
                 (x3 * x3 + y3 * y3) * (x2 - x1)) / D;

    double r = sqrt((px - x1) * (px - x1) + (py - y1) * (py - y1));

    printf("%.3f %.3f %.3f\n", px, py, r);
}

int main() {
    int n;
    printf("Enter the number of datasets: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        double x1, y1, x2, y2, x3, y3;
        printf("Enter the coordinates of the triangle (x1 y1 x2 y2 x3 y3): ");
        scanf("%lf %lf %lf %lf %lf %lf", &x1, &y1, &x2, &y2, &x3, &y3);

        calculateCircumcircle(x1, y1, x2, y2, x3, y3);
    }

    return 0;
}
```

Looks easy, but it is not…

Sometimes the code does not compile…

- Feeding back the compilation warnings and errors: easy fix!

For more complex problems, frequently the testcases fail!

- Feeding back the testcase and ask for a fix…
- Sometimes it fixes, most times fails again ☹

Case: large scale experiment to assess performance!

- About 4000 problem statements and testcases from coding contests
- Diverse solutions (correct and incorrect) in different languages
- How close are the generated solutions to the correct reference solutions?
- How close are the generated solutions to the bad reference solutions?

Code Analysis

Proactive quality assurance through real-time analysis

- Identifies flaws, vulnerabilities, and performance issues during development

How can LLMs help?

- Scan code as it is written
- Flag potential weaknesses or risky practices
- Detect vulnerabilities like unprotected user inputs
- Suggests fixes to avoid security risks
- …

The goal is to reduce rework by addressing issues early and prevents errors from propagating through the codebase

Code Analysis: Example

"Identify security vulnerabilities in the code…"

- The code is the one generated before!

"The provided C code is well-written for its purpose, but it may still have potential security vulnerabilities, especially when considering real-world usage scenarios. Below are some vulnerabilities and areas of concern:

1. Unchecked Input in scanf
2. Lack of Input Bounds Checking
3. Floating-Point Arithmetic Issues
4. Division by Zero
5. Potential Denial of Service (DoS)
6. Lack of Error Handling for printf
7. Hardcoded Limits on Precision
8. Output Information Disclosure

Including many more details!

Testing

Generates test cases for functional and non-functional aspects

- Ensure rigorous evaluation of code against edge cases

How can LLMs help?

- Automatically generate testcases based on the specification
- Generate edge cases to reduce unforeseen errors
- Address performance, reliability, and security aspects
- Scenario-specific testing: e.g., financial application - simulate high transaction volumes
- …

Case: generating tests from problem statements and “test the tests”!

Fault and Vulnerability Injection

Facilitates error detection and intrusion tolerance validation

- Simulates faults or vulnerabilities to assess system robustness

How can LLMs help?

- Inject faults to observe system behavior under stress
- Creates complex attack vectors to test defensive mechanisms
- Leverage historical incident data to define fault models, vulnerability models, attack models
- …

Vulnerability Injection: Example

“Inject a vulnerability in the code…”

```
PreparedStatement ps7 = con.prepareStatement("UPDATE tpcc_customer
SET c_balance = c_balance + ?, c_delivery_cnt = c_delivery_cnt + 1
WHERE c_id = ? AND c_d_id = ? AND c_w_id = ?");
ps7.setDouble(1, ol_total);
ps7.setLong(2, c_id);
ps7.setLong(3, d_id);
ps7.setString(4, w_id);
result = ps7.executeUpdate();
```

This is the “too obvious” output:

```
String query = "UPDATE tpcc_customer SET c_balance = c_balance + "
+ ol_total + ", c_delivery_cnt = c_delivery_cnt + 1 WHERE c_id = "
+ c_id + " AND c_d_id = " + d_id + " AND c_w_id = '" + w_id + "'";
Statement stmt = con.createStatement();
result = stmt.executeUpdate(query);
```

More on Development…

Refactoring:

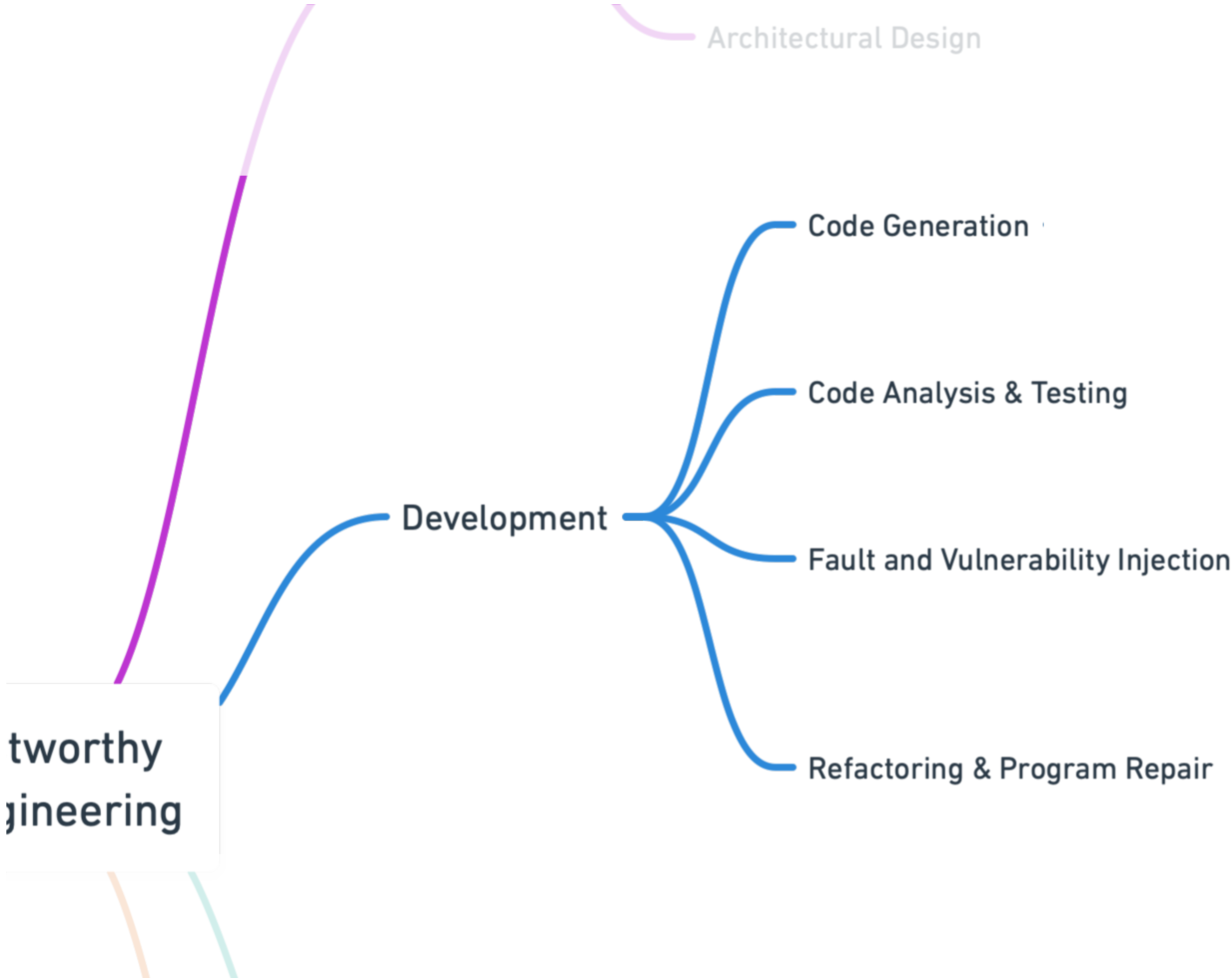
– Detect outdated or risky code patterns
– Suggests revisions to prevent security liabilities or performance bottlenecks

Program repair:

– Automatically detect and resolves defects: null pointers, vulnerabilities, …
– Suggest context-aware fixes aligned with best practices

Programming language migration:

– Facilitate modernization of legacy systems by automating code translation
– Convert language-specific constructs and adapt to new paradigms
– Example: migrating from C++ to Rust to ensure safety and concurrency



Architectural Design
Development
Code Generation
Code Analysis & Testing
Fault and Vulnerability Injection
Refactoring & Program Repair
tworthy
gineering

Deployment

Infrastructure as Code (IaC):

- Codify infrastructure configurations, reducing manual intervention
- Ensure consistent, reliable, and secure deployments
- LLMs: automate creation of scripts, identify configuration problems, detect and resolve deployment issues, translate configurations to diverse environments

Monitoring and anomaly detection:

- Ensure security, reliability, and performance by identifying deviations
- Tracking key indicators: memory usage, CPU load, response times, …
- Analysis of runtime data: system logs, user behavior, …
- Example: flag unusual login patterns as potential unauthorized access.

Issue Management

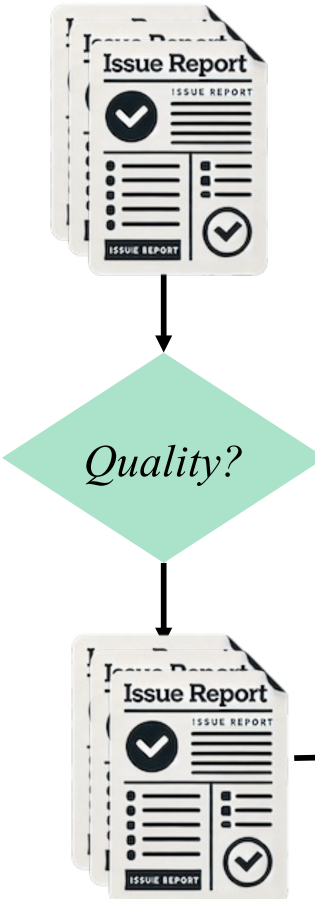
Ensures timely resolution of incidents

- Maintain trustworthiness by addressing unexpected problems
- Time-consuming task especially in very large projects

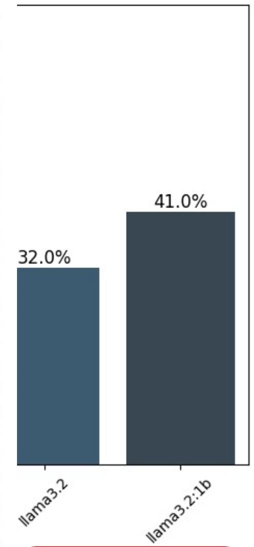
How can LLMs help?

- Automating triage and prioritization
- Root cause analysis and fault localization
- Remediation suggestions
- …

Case: Triage and Prioritization

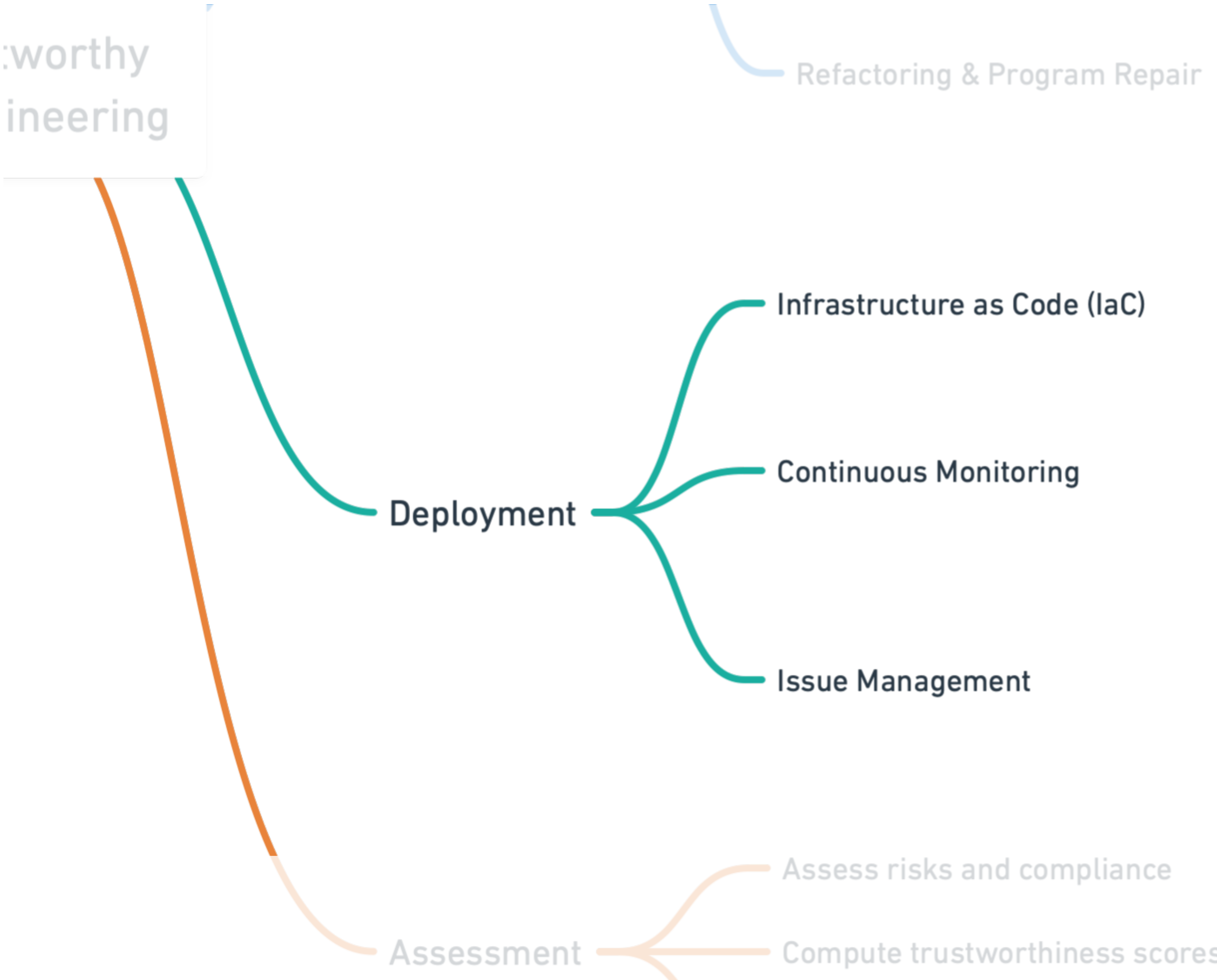


Project	Language	Issues	Bugs	Non-bugs	LoC
Firefox	C/C++	25423	18607	6816	25,300,000
Mozilla Core	C/C++	164708	128608	36100	20,300,000
NextCloud Server	PHP	15392	10821	4571	9,110,000
Roslyn	C#	10248	8290	1958	5,900,000
MariaDB Server	C/C++	11746	9855	1891	4,280,000
Kibana	TypeScript	13680	11461	2219	3,230,000
Tensorflow	C/C++	6546	4912	1634	3,090,000
QGis	C/C++	24080	20543	3537	2,190,000
Godot	C/C++	23727	21105	2622	1,590,000
MongoDB Server	C/C++	28641	13730	14911	1,590,000
Spring Framework	Java	12734	4440	8294	1,420,000
Elasticsearch	Java	20026	9605	10421	1,200,000
Bazel	Java	3283	2110	1173	1,110,000
Mozilla NSS	C/C++	6493	4144	2349	1,080,000
Symfony	PHP	16759	11602	5157	1,030,000
SeaMonkey	C/C++	9946	8765	1181	1,020,000



Critical
High
Medium
Low

by Tasfia Tasnim



worthy
ineering
Refactoring & Program Repair
Deployment
Infrastructure as Code (IaC)
Continuous Monitoring
Issue Management
Assessment
Assess risks and compliance
Compute trustworthiness scores

Risks and Compliance

Ensures systems remain trustworthy over time

- Proactive evaluation supports informed decision-making

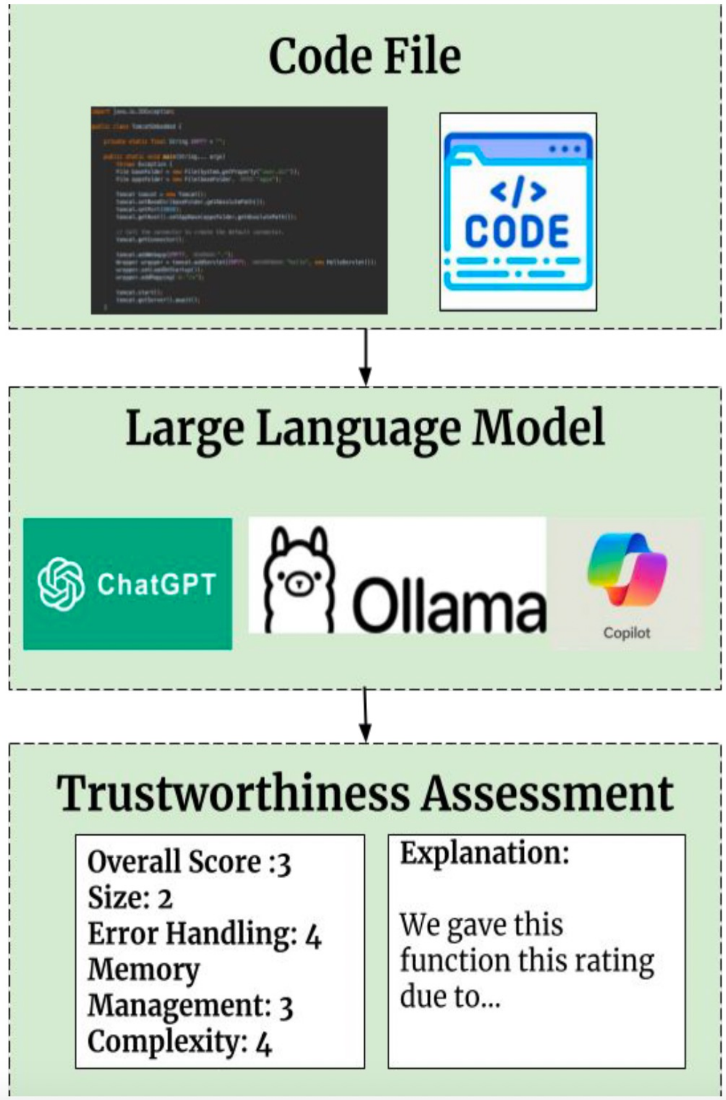
LLMs @ design:

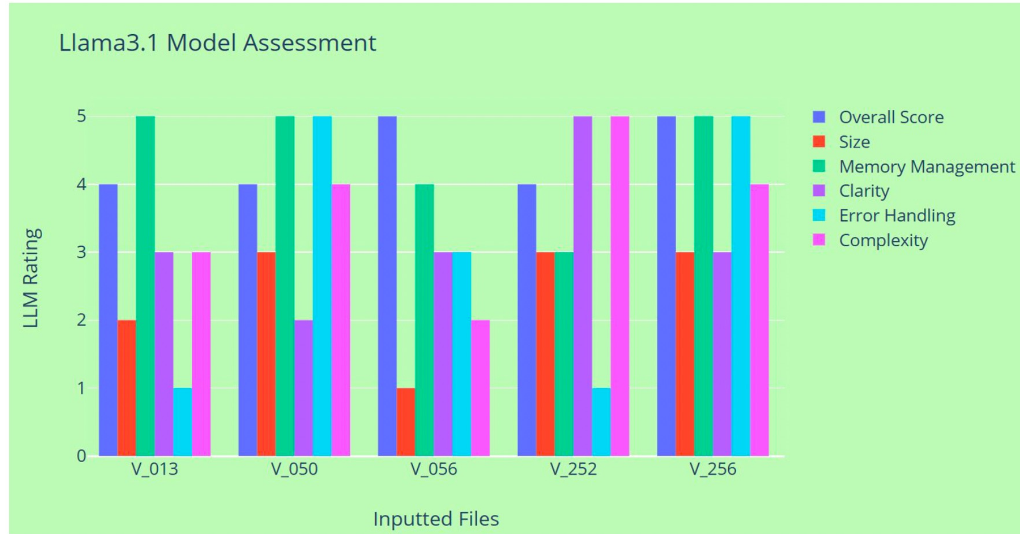
- Analyze architectural choices and system specifications
- Evaluate potential risks like vulnerabilities and scalability issues
- Check compliance with regulatory requirements
- Example: Highlight areas requiring security controls

LLMs @ runtime:

- Monitor key events and metrics: security incidents, unusual user activity, …
- Identify and address emerging threats

Case: Risks and Compliance





- (Almost) consistent results across multiple iterations
- (Slightly) conflicting results compared to other models
- LLMs are capable of explaining their scores based on function attributes

Work by Austin Lee

Scores and Dashboards

Generate trustworthiness scores based on current system data

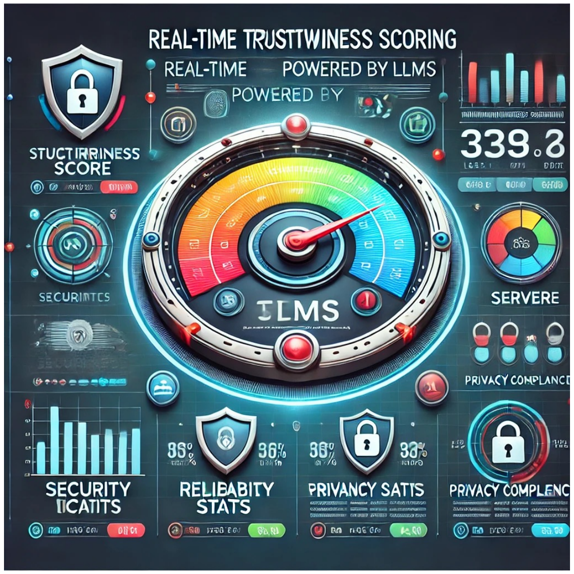
– Provide up-to-date insights into system trustworthiness

Proactive decision-making:

– Inform stakeholders of potential weaknesses
– Support timely corrective actions if needed

Enables trustworthiness to evolve with:

– Internal changes (e.g., system updates)
– External conditions (e.g., regulatory shifts)



Know More!

Work by Arastoo Zibaeirad

SVD and SVP

Our Benchmark

Some Results

Challenges

CASE: BENCHMARKING VULNERABILITY DETECTION AND PATCHING WITH LLMS

SVD and SVP

SVD: Software Vulnerability Detection

SVP: Software Vulnerability Patching

Rising need for automation

– Surge in identified software vulnerabilities each year: > 29,000+ CVEs in 2023

Some traditional techniques:

– Static Analysis Tools (SAT)
– Fuzzing and Penetration Testing Tools
– Automatic Program Repair (APR)

LLMs as a complementary approach… but what is the performance?

Current Limitations

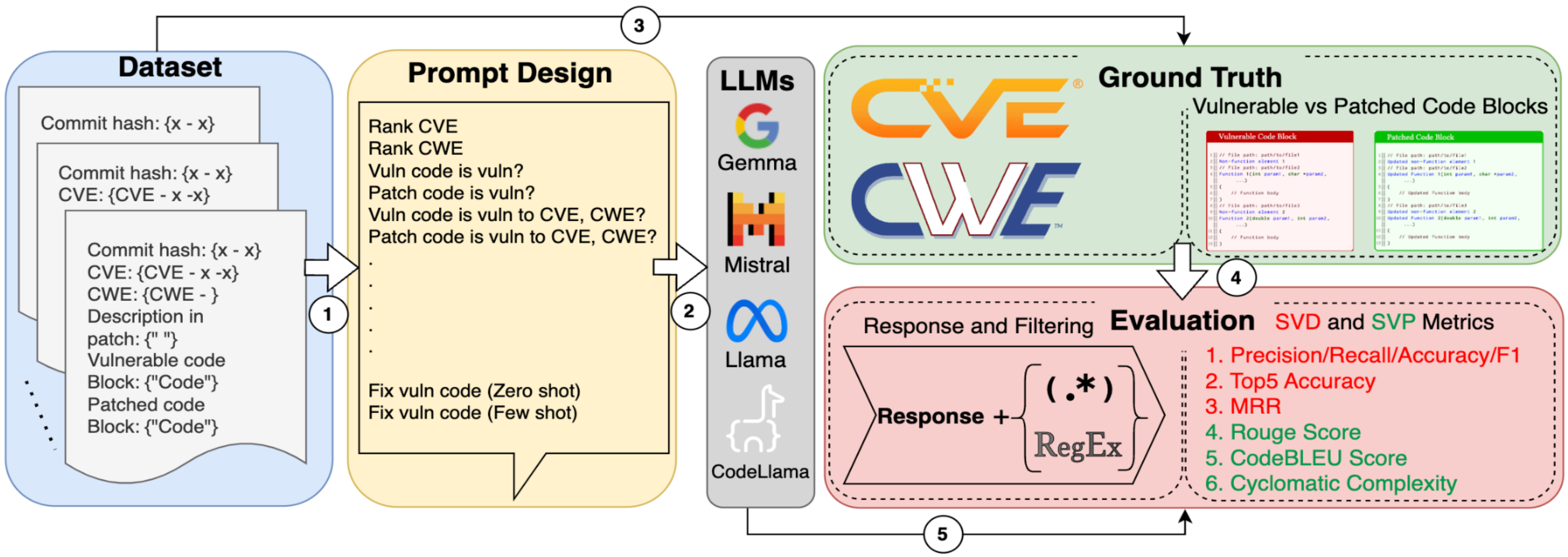
No real-world datasets

- Small code snippets rather than complex real-world vulnerabilities
- Lacking ground truth labels and patches (small dataset, manual labeling)

Data leakage in evaluation

- LLMs evaluated on datasets that include code they were trained on
- Inflated performance does not reflect capabilities in real-world settings

Our Benchmark



Real-world vulnerable and patched code: > 300 vulnerabilities from Linux kernel

Evaluated 10 LLMs!

Vulnerable Code Block

```
// File path: path/to/file1
Non-function element 1
// File path: path/to/file2
Function 1(int param1, char *param2,
      ...)
{
    // Function body
}
// File path: path/to/file3
Non-function element 2
Function 2(double param1, int param2,
      ...)
{
    // Function body
}
```

Patched Code Block

```
// File path: path/to/file1
Updated non-function element 1
// File path: path/to/file2
Updated Function 1(int param1, char *param2,
      ...)
{
    // Updated function body
}
// File path: path/to/file3
Updated non-function element 2
Updated Function 2(double param1, int param2,
      ...)
{
    // Updated function body
}
```

SVD: Overall Performance

LLMs	Precision		Recall		Accuracy		F1 Score	
	SVD3,4	SVD5,6	SVD3,4	SVD5,6	SVD3,4	SVD5,6	SVD3,4	SVD5,6
Codellama-7b	48.95	68.08	48.53	56.95	49.26	65.15	49.02	56.10
Codellama-34b	49.86	60.76	49.83	54.77	51.02	52.08	51.04	51.55
Llama3-8b	49.65	93.16	49.35	64.78	48.50	79.15	47.56	60.15
Llama3-70b	47.57	28.66	48.53	35.77	46.99	28.01	48.21	35.10
Llama3.1-8b	50.09	88.93	50.16	64.08	49.60	80.46	49.35	61.37
Llama3.1-70b	50.48	68.73	50.65	58.21	49.16	66.78	48.86	56.63
Mistral-7b	49.38	51.47	49.35	50.40	49.59	39.41	49.67	43.92
Mixtral-8*7b	49.23	72.88	48.86	58.76	49.32	35.62	49.51	41.37
Gemma2-27b	52.33	47.56	52.12	49.83	49.59	59.61	49.51	54.14
Gemma2-9b	50.78	85.34	51.30	63.67	49.34	73.62	49.02	59.08

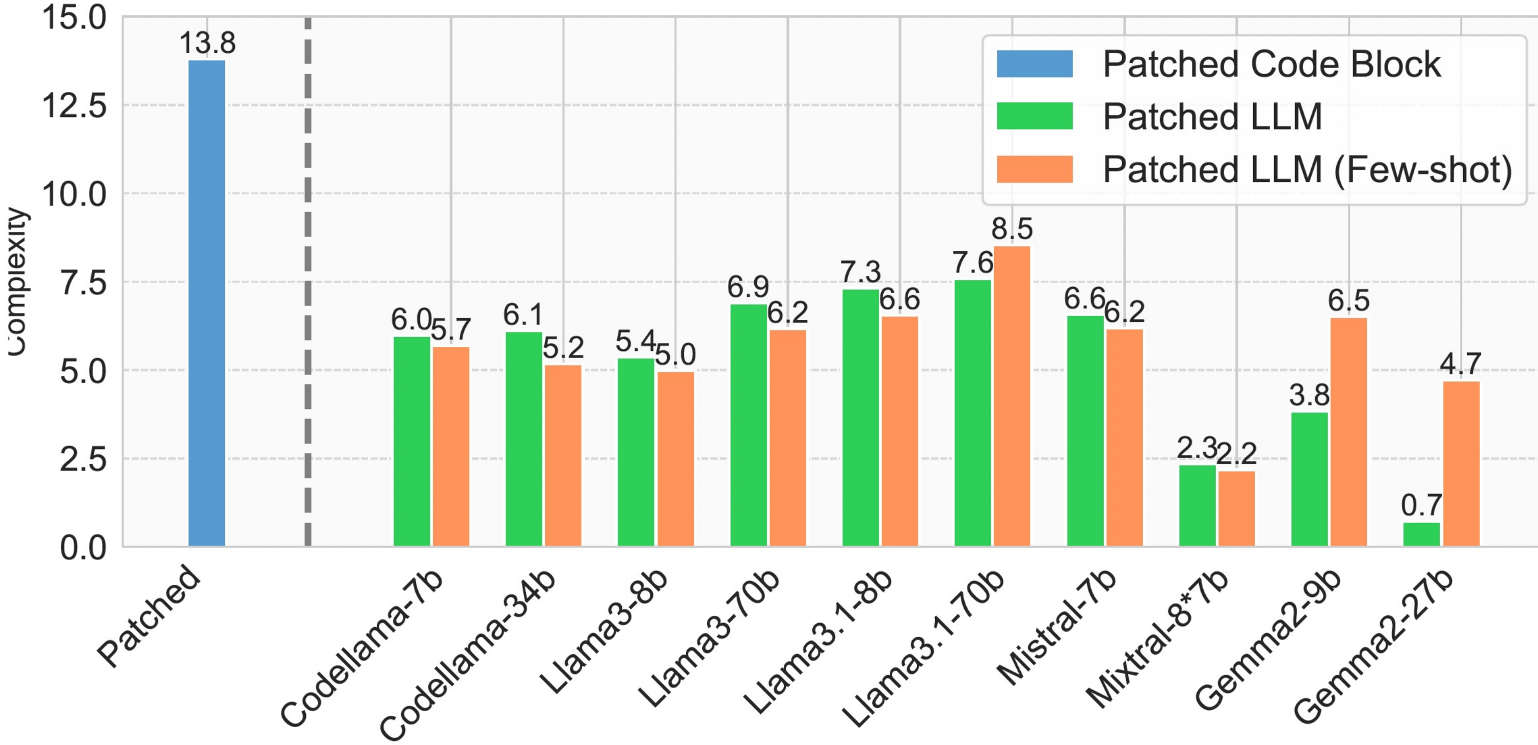
- SVD3: Is vulnerable (Z)?
- SVD4: Is patched vulnerable (Z)?
- SVD5: CVE/CWE-Vuln Check (Z)? V, CVE, CWE
- SVD6: CVE/CWE-Patch Check (Z)? P, CVE, CWE

SVD: Vulnerable vs. Patched Code

Struggle to distinguish between vulnerable (on the left) and patched (on the right) code

– Particularly when changes are subtle!

SVP: Oversimplification

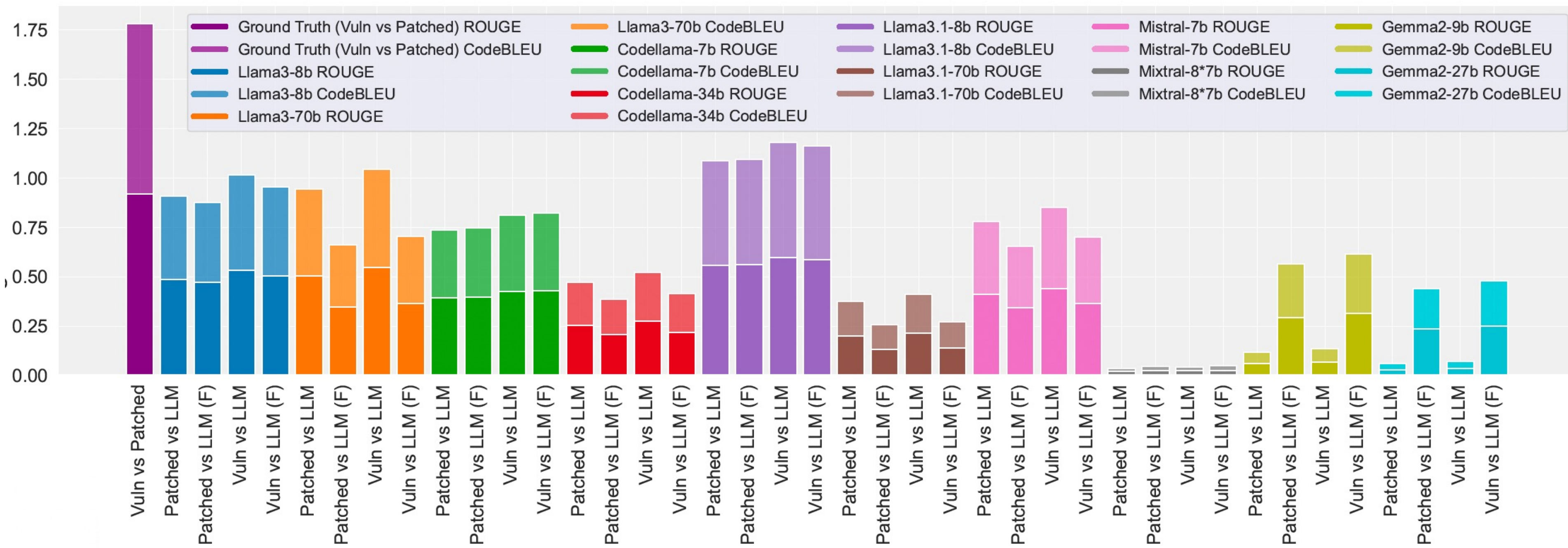


Generated patches often oversimplify the original code

– Resulting in lower cyclomatic complexity

This can improve readability, but impacts functionality or security

SVP: Incompleteness



Similarity scores lower than for ground truth

LLMs produce solutions that are incomplete or require refinement

Generated patches are typically shorter

– Omit critical context or introduce new issues if essential details are missed

Challenges

Limited understanding of program behavior

– Struggles with data flow, control flow, data dependencies, and interactions

Generalization issues

– Difficulty identifying complex or unseen vulnerabilities

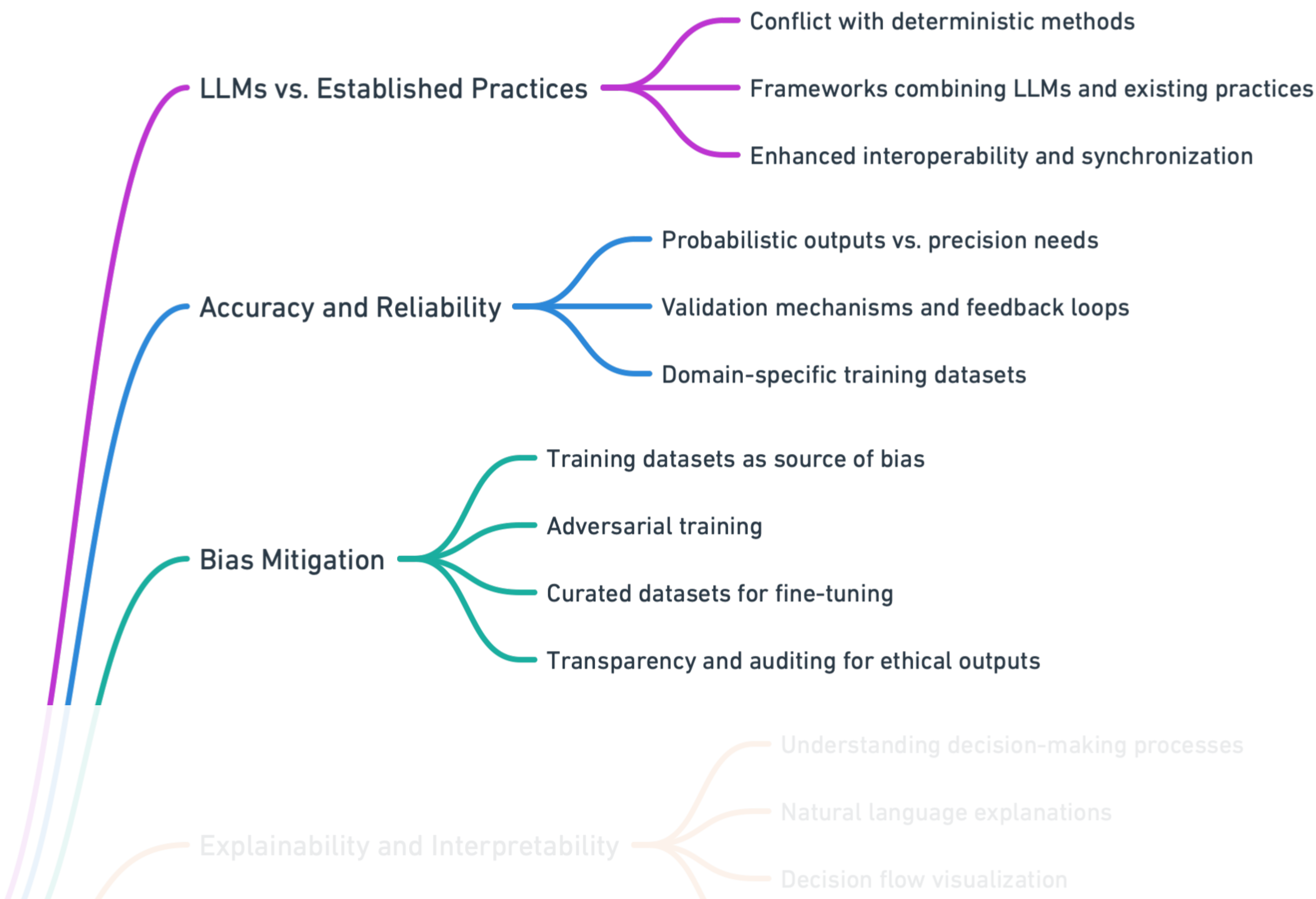
– Reduced precision and recall

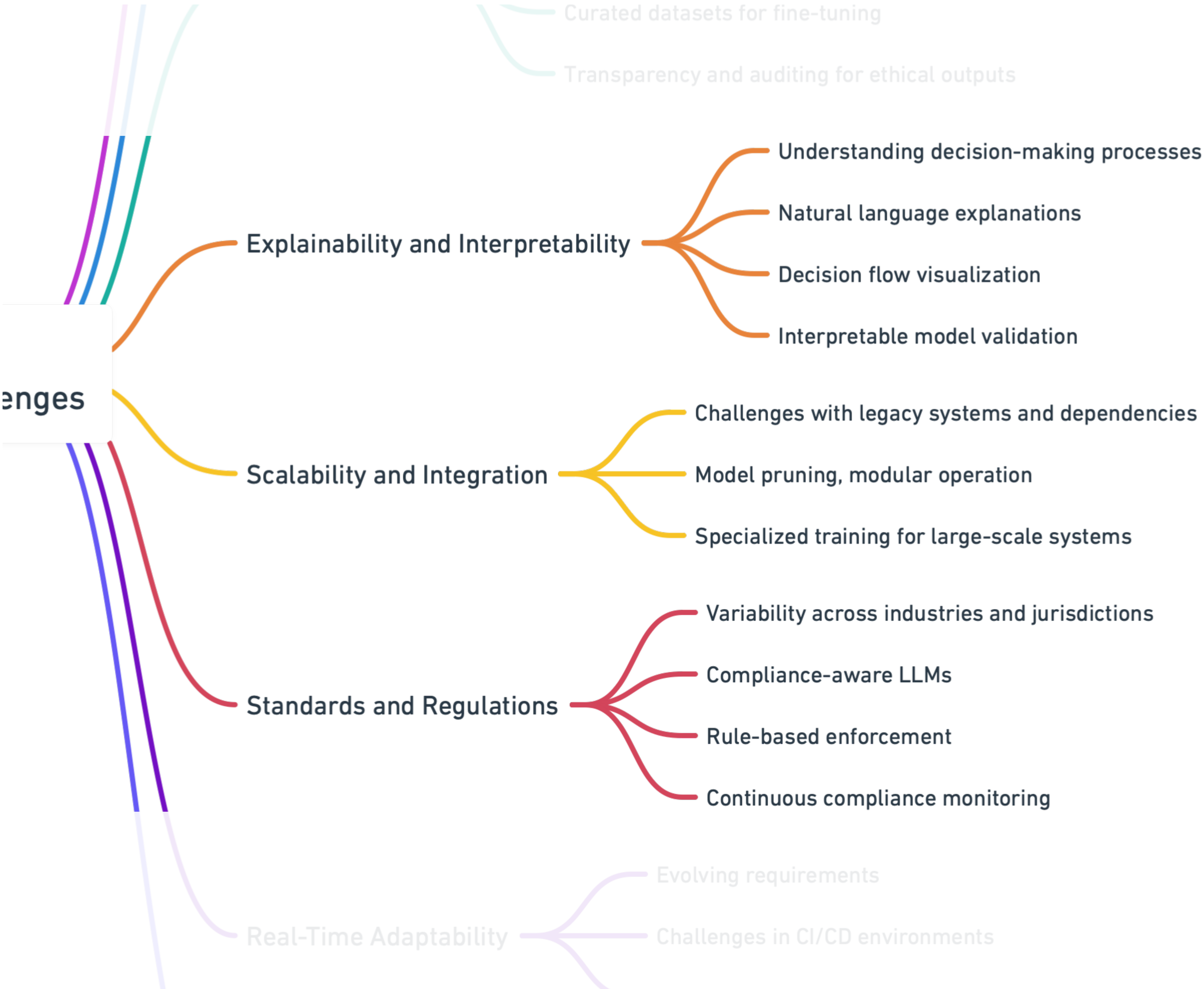
Vulnerability to adversarial attacks

– Small changes, like function renaming, can mislead the models

(SOME OF) THE OPEN CHALLENGES…


Open Challenges
LLMs vs. Established Practices
Conflict with deterministic methods
Frameworks combining LLMs and existing practices
Enhanced interoperability and synchronization
Accuracy and Reliability
Probabilistic outputs vs. precision needs
Validation mechanisms and feedback loops
Domain-specific training datasets
Bias Mitigation
Training datasets as source of bias
Adversarial training
Curated datasets for fine-tuning
Transparency and auditing for ethical outputs
Explainability and Interpretability
Understanding decision-making processes
Natural language explanations
Decision flow visualization
Interpretable model validation
Scalability and Integration
Challenges with legacy systems and dependencies
Model pruning, modular operation
Specialized training for large-scale systems
Standards and Regulations
Variability across industries and jurisdictions
Compliance-aware LLMs
Rule-based enforcement
Continuous compliance monitoring
Real-Time Adaptability
Evolving requirements
Challenges in CI/CD environments
Incremental learning, federated learning
Ethics and Privacy
Risks of exposing sensitive data
Privacy-preserving techniques
Ethical audits
Governance for responsible deployment


LLMs vs. Established Practices
Conflict with deterministic methods
Frameworks combining LLMs and existing practices
Enhanced interoperability and synchronization
Accuracy and Reliability
Probabilistic outputs vs. precision needs
Validation mechanisms and feedback loops
Domain-specific training datasets
Bias Mitigation
Training datasets as source of bias
Adversarial training
Curated datasets for fine-tuning
Transparency and auditing for ethical outputs
Explainability and Interpretability
Understanding decision-making processes
Natural language explanations
Decision flow visualization


Curated datasets for fine-tuning
Transparency and auditing for ethical outputs
enges
Explainability and Interpretability
Understanding decision-making processes
Natural language explanations
Decision flow visualization
Interpretable model validation
Scalability and Integration
Challenges with legacy systems and dependencies
Model pruning, modular operation
Specialized training for large-scale systems
Standards and Regulations
Variability across industries and jurisdictions
Compliance-aware LLMs
Rule-based enforcement
Continuous compliance monitoring
Real-Time Adaptability
Evolving requirements
Challenges in CI/CD environments

Take-Away(s)

LLMs have the potential to reshape software engineering practices

- Automating code generation, bug detection, documentation, …

Empowering teams to build faster, smarter, and trustworthy

Key challenges:

- Ensuring generated outputs align with real-world requirements
- Addressing inherent biases to ensure fairness and ethical use
- Making model decisions transparent and interpretable for developers
- Adapting LLMs to diverse, complex, and large-scale projects
- …


UNIVERSITY OF NORTH CAROLINA
CHARLOTTE