Fixing breaking dependency updates with LLM

Speaker: Frank Reyes

Supervisors: Martin Monperrus and Benoit Baudry

Introduction

Client application

Dependency update

breaking

Break client application

Challenges

- Identify the source of the error from large volumes of log data.
- Distinguish between warnings and critical errors.

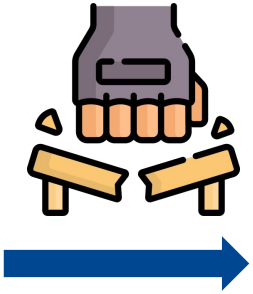
Breaking Update

1 file changed +1 -1 lines changed

pom.xml +1 -1

```
@@ -19,7 +19,7 @@
19 19        <dependency>
20 20            <groupId>org.A</groupId>
21 21            <artifactId>A</artifactId>
22    -          <version>1.0.0</version>
   22 +          <version>2.0.0</version>
23 23        </dependency>
24 24        <dependency>
25 25            <groupId>org.B</groupId>
```

One single change in the pom.xml file



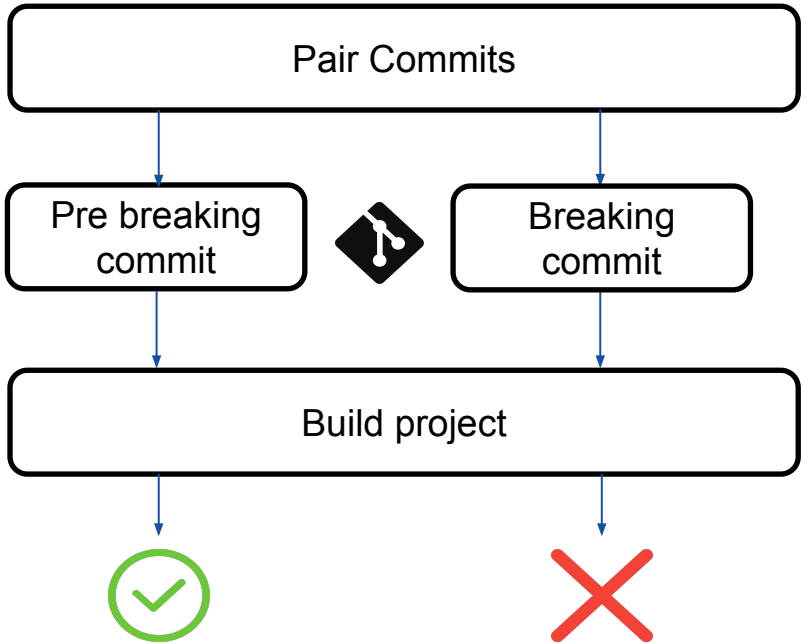
Dependency update

Break client application

- External dependencies are essential, but upgrades can be a risk.
- We need to understand and reproduce the failures in order to resolve them effectively.

Breaking Dependency Update

What is a Breaking dependency update?



1 file changed +1 -1 lines changed

pom.xml +1 -1

```
@@ -19,7 +19,7 @@
19  19          <dependency>
20  20              <groupId>org.A</groupId>
21  21              <artifactId>A</artifactId>
22      -           <version>1.0.0</version>
    22  +           <version>2.0.0</version>
23  23          </dependency>
24  24          <dependency>
25  25              <groupId>org.B</groupId>
```

F. Reyes, Y. Gamage, G. Skoglund, B. Baudry and M. Monperrus, "BUMP: A Benchmark of Reproducible Breaking Dependency Updates," *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Rovaniemi, Finland, 2024, pp. 159-170, doi: 10.1109/SANER60148.2024.00024.

Example of breaking dependency updates

Dependency updates can introduces **incompatible changes**:

- Renamed or removed methods
- Visibility changes (e.g., public → private)
- Parameter changes

Causes: Compilation errors, test failures

```
237 ...
238 try {
239     GHCompare.Status status = GitHub.connect().getRepository(ghc.owner + '/' + ghc.
          repo).getCompare(branch, ghc.hash).status;
240     return status == GHCompare.Status.identical || status == GHCompare.Status.behind;
241 } catch (FileNotFoundException x) {
242     // For example, that branch does not exist in this repository.
243     return false;
244 }
245 ...
246
```

```
[ERROR]/incrementals-tools/lib/src/main/java/io/jenkins/tools/incrementals/lib/UpdateChecker.java:[239,126] status has private access in org.kohsuke.github.GHCompare
```

Our Goal

Fix Breaking Dependency updates using LLM

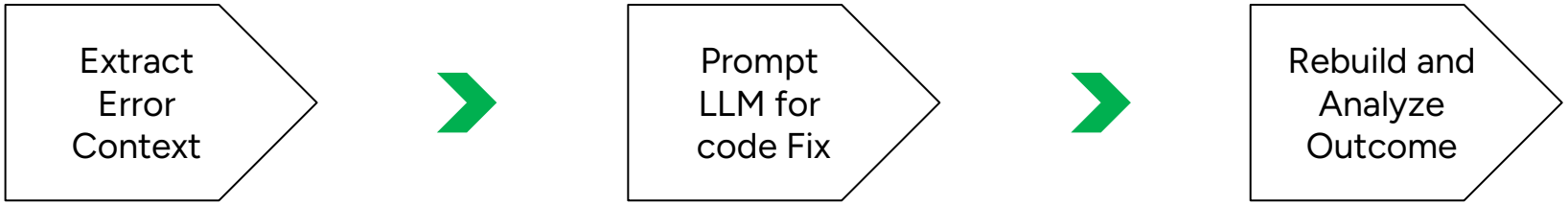
- LLMs offer adaptive, contextual code generation

Previous approaches:

- Rule-based transformations, rigid and limited
- Static analysis, lacks flexibility

Byam: an approach to fix breaking dependency updates using LLM

Methodology

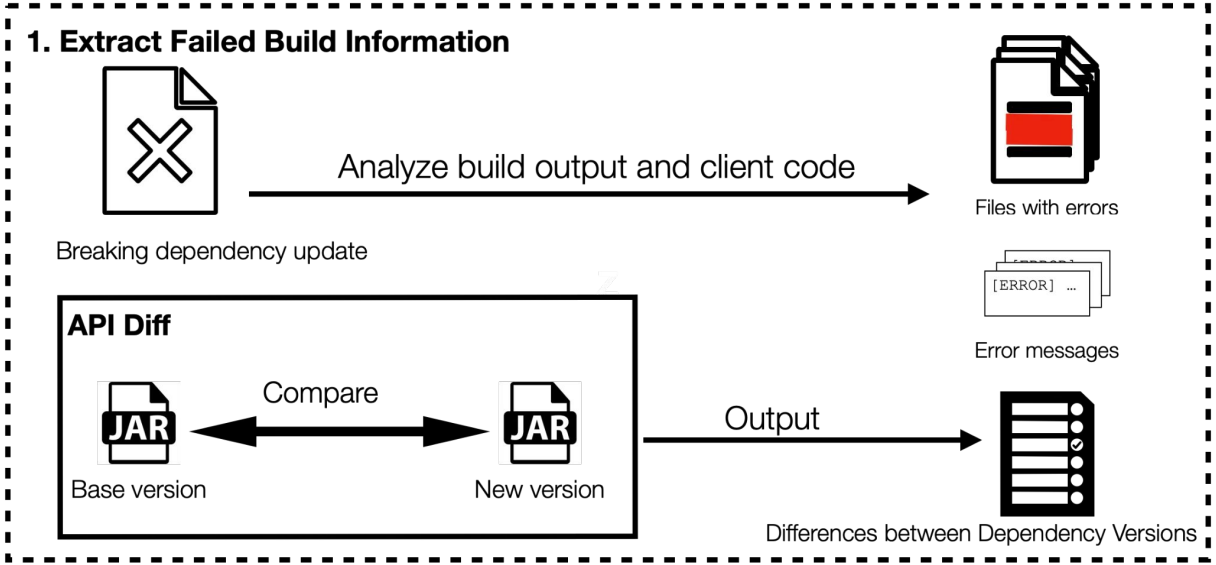


- Extraction of error lines from the logs
- Extract differences between dependency version

- Build prompt template
  - Buggy Line
  - Error Messages
  - API Differences
  - CoT

- Update LLM generated code into the project
- Re-run the build and tests

Step 1: Extract build information




```
[ERROR]/incrementals-tools/lib/src/main/java/io/jenkins/tools/incrementals/lib/UpdateChecker.java:[239,126] status has private access in org.kohsuke.github.GHCompare
```

File path

Buggy line

Step 1: Extract build information

```
[ERROR]/incrementals-tools/lib/src/main/java/io/jenkins/tools/incrementals/lib/UpdateChecker.java:[239,126] status has private access in org.kohsuke.github.GHCompare
```

File path

Buggy line

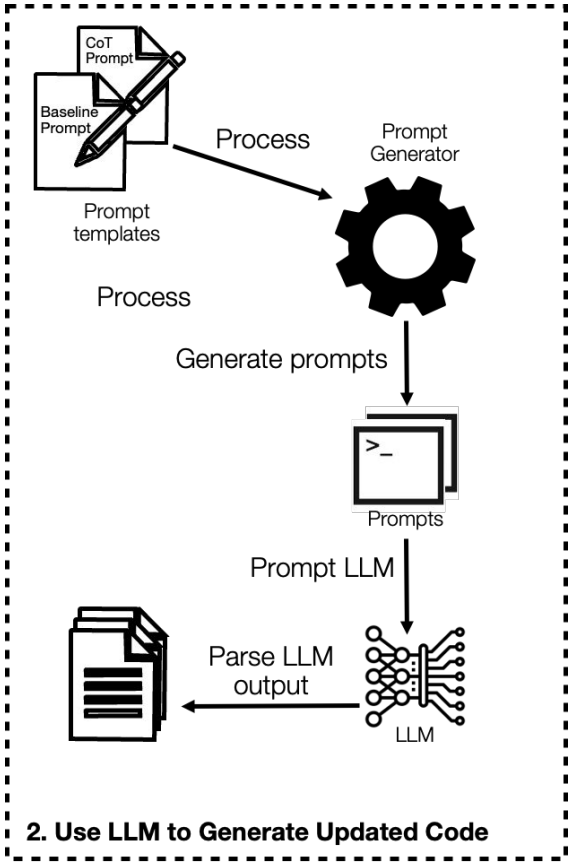
API Changes

```
The error is caused by a change in the API of the dependency. The new library version includes the following changes:
    - ***! MODIFIED CLASS: PUBLIC org.kohsuke.github.GHCompare  (not serializable)
    ***! CLASS FILE FORMAT VERSION: 52.0 <- 49.0
    ***! MODIFIED FIELD: PRIVATE (<- PUBLIC)org.kohsuke.github.GHCompare$Status status
```

Buggy Line

```
GHCompare.Status status = GitHub.connect().getRepository(ghc.owner + '/' + ghc.repo).getCompare(branch, ghc.hash).status;
```

Step 2: Prompt LLM for code fix



Variation	Description
Baseline Prompt	Includes client code and error message but excludes additional context.
Buggy Line Inclusion	Adds the specific line of code causing the compilation error.
API Differences (API Diff)	Includes details of API differences between dependency versions.
Chain of Thought (CoT) Prompting	Guides LLM reasoning by incorporating structured reasoning steps.

Step 2: Prompt LLM for code fix

Prompt ID	Prompt Name	Client Code	Error Message	Buggy Line	APIDiff	CoT
P1	Baseline Prompt	✓	✓			
P2	Buggy Line	✓	✓	✓		
P3	APIDiff	✓	✓		✓	
P4	Buggy Line + APIDiff	✓	✓	✓	✓	
P5	CoT Prompt	✓	✓			✓
P6	CoT + Buggy Line	✓	✓	✓		✓
P7	CoT + API Diff	✓	✓		✓	✓
P8	CoT + Buggy Line + APIDiff	✓	✓	✓	✓	✓

Step 2: Prompt LLM for code fix

Act as an Automatic Program Repair (APR) tool, reply only with code, without explanation.
You are specialized in breaking dependency updates, in which the failure is caused by an external dependency.
To solve the failure you can only work on the client code.

the following client code fails:
“‘java
<client_code>
“‘

with the following error message:
<error_message>

- Propose a patch that can be applied to the code to fix the issue.
- Return only a complete and compilable class in a fenced code block.
- You CANNOT change the function signature of any method but may create variables if it simplifies the code.
- You CAN remove the @Override annotation IF AND ONLY IF the method no longer overrides a method in the updated dependency version.
- If fixing the issue requires addressing missing imports, ensure the correct package or class is used in accordance with the newer dependency version.
- Avoid removing any existing code unless it directly causes a compilation or functionality error.
- Return only the fixed class, ensuring it fully compiles and adheres to these constraints.

Step 2: Prompt LLM for code fix

Buggy Line

the error is triggered in the following specific lines in the previous code: <buggy line>

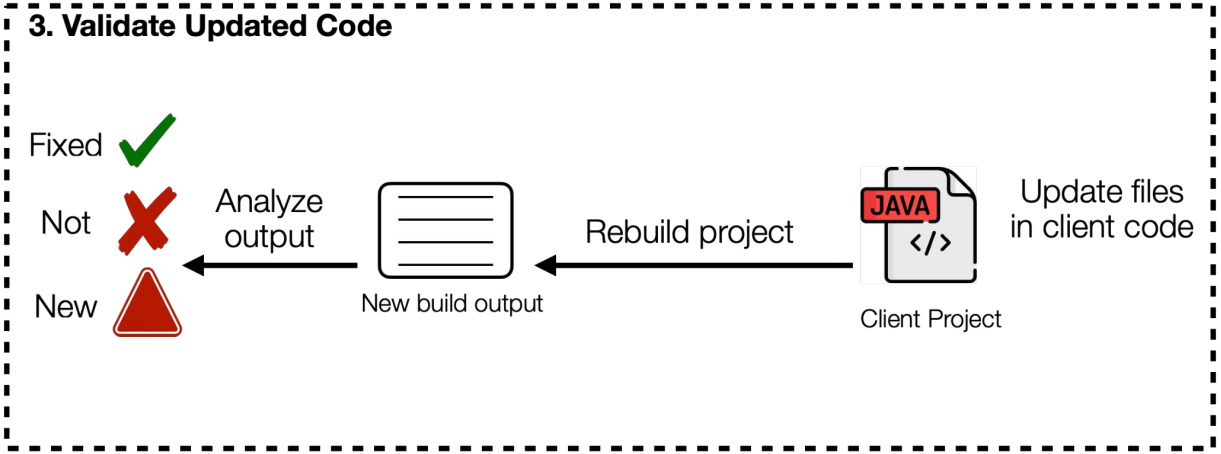
API DIff

The error is caused by a change in the API of the dependency. The new library version includes the following changes: <api diff>

CoT

Before proposing a fix, please analyze the situation and plan your approach within <repair strategy > tags:
– Identify the specific API changes that are causing the failure in the client code.
– Compare the old and new API versions, noting any changes in method signatures, return types, or parameter lists.
– Determine which parts of the client code need to be updated to accommodate these API changes.
– Consider any constraints or requirements for the fix (e.g., not changing function signatures, potential import adjustments).
– Plan the minimal set of changes needed to fix the issue while keeping the code functional and compliant with the new API.
– Consider potential side effects of the proposed changes on other parts of the code.
– Ensure that the planned changes will result in a complete and compilable class.
– If applicable, note any additional imports that may be needed due to the API changes.

Step 3: Rebuild



LLM-generated fixed code replaces buggy files

- Rebuild project (e.g., `mvn test`)

Track success:

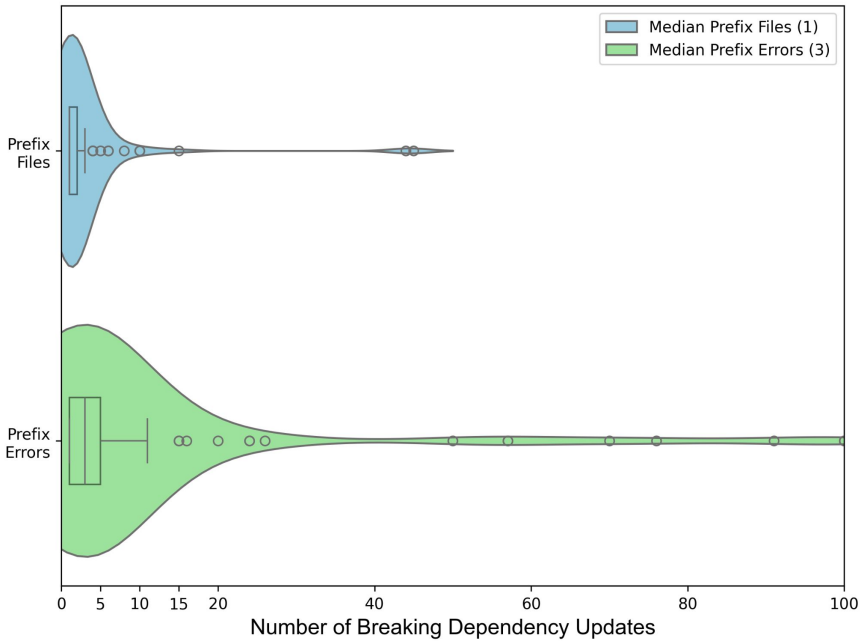
- Build success?
- Fixed files?
- Fixed errors?
- Any new errors introduced?

```
try {
-       GHCompare.Status status = GitHub.connect().getRepository(ghc.owner + '/' + ghc
        .repo).getCompare(branch, RHilightghc.hash).status;
+       GHCompare compare = GitHub.connect().getRepository(ghc.owner + '/' + ghc.repo)
        .getCompare(branch, ghc.hash);
+       GHCompare.Status status = compare.getStatus();
            return status == GHCompare.Status.identical || status == GHCompare.Status.
    behind;
} catch (FileNotFoundException x) {
        // For example, that branch does not exist in this repository.
        return false;
}
```

Experimental Setup

Dataset: **BUMP** – 571 real-world Breaking dependency updates

- 243 (43%) Compilation failures
- 103 breaking dependency updates



LLMs evaluated:

- OpenAI o3-mini
- GPT-4o-mini
- Google Gemini Flash
- DeepSeek V3
- Qwen2.5-32B-instruct

Evaluation metrics

1. **Build Success Rate (BSR)** – full builds fixed
2. **File Fix Success Rate (FFSR)** – % files with no errors, from failed repairs
3. **Compilation Error Fix Rate (CEFR)** – % of errors fixed, from failed repairs
4. **Relative Error Fix (REF)** – new errors introduced

Build Repair

Best result: **o3-mini**

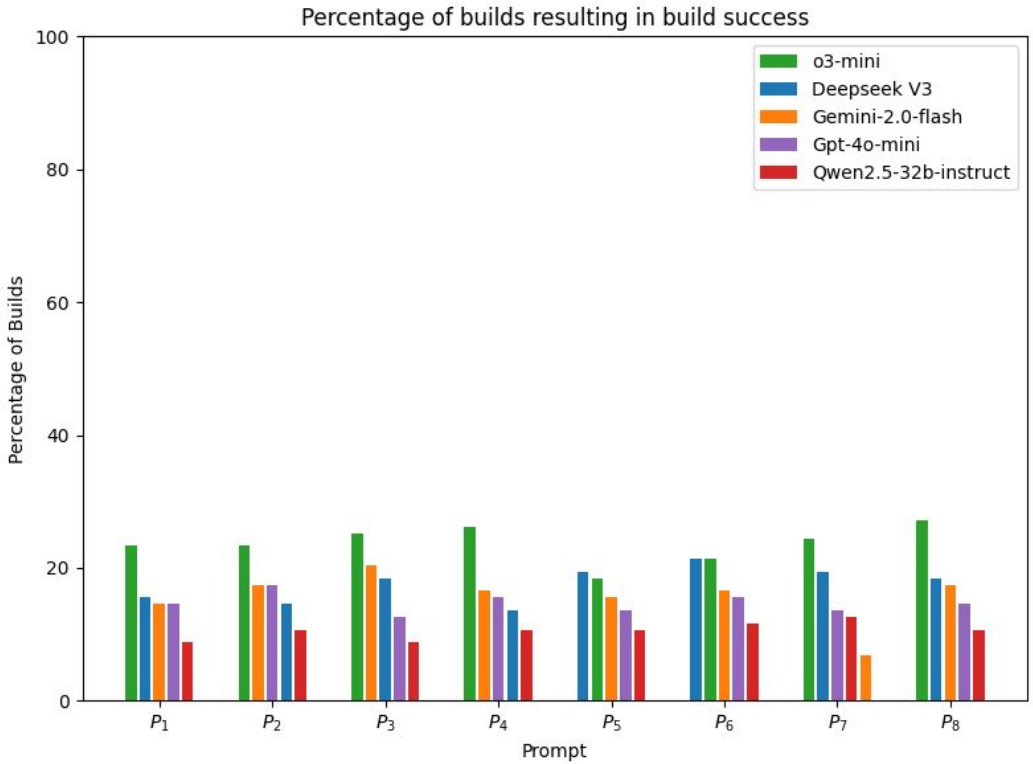
- **27% builds fully repaired**

Other models:

DeepSeek V3: 21%

Gemini: struggled with CoT

Qwen: low success



File Fix Success Rate (FFSR)

Prompt ID	Deepseek V3	Gemini 2.0-flash	Gpt 4o-mini	o3 mini	Qwen2.5 32b-instruct
P1	48/252(19%)	48/251(19%)	40/256(16%)	75/242(31%)	41/262(16%)
P2	61/254(24%)	54/248(22%)	46/253(18%)	66/241(27%)	44/263(17%)
P3	62/249(25%)	82/246(33%)	53/262(20%)	89/246(36%)	61/265(23%)
P4	64/255(25%)	87/251(35%)	53/255(21%)	97/239(41%)	46/263(17%)
P5	52/251(21%)	59/260(23%)	36/254(14%)	72/252(29%)	63/268(24%)
P6	42/248(17%)	39/243(16%)	40/254(16%)	74/243(30%)	62/263(24%)
P7	59/244(24%)	71/253(28%)	57/260(22%)	88/238(37%)	61/264(23%)
P8	71/248(29%)	76/250(30%)	53/256(21%)	92/246(37%)	54/267(20%)

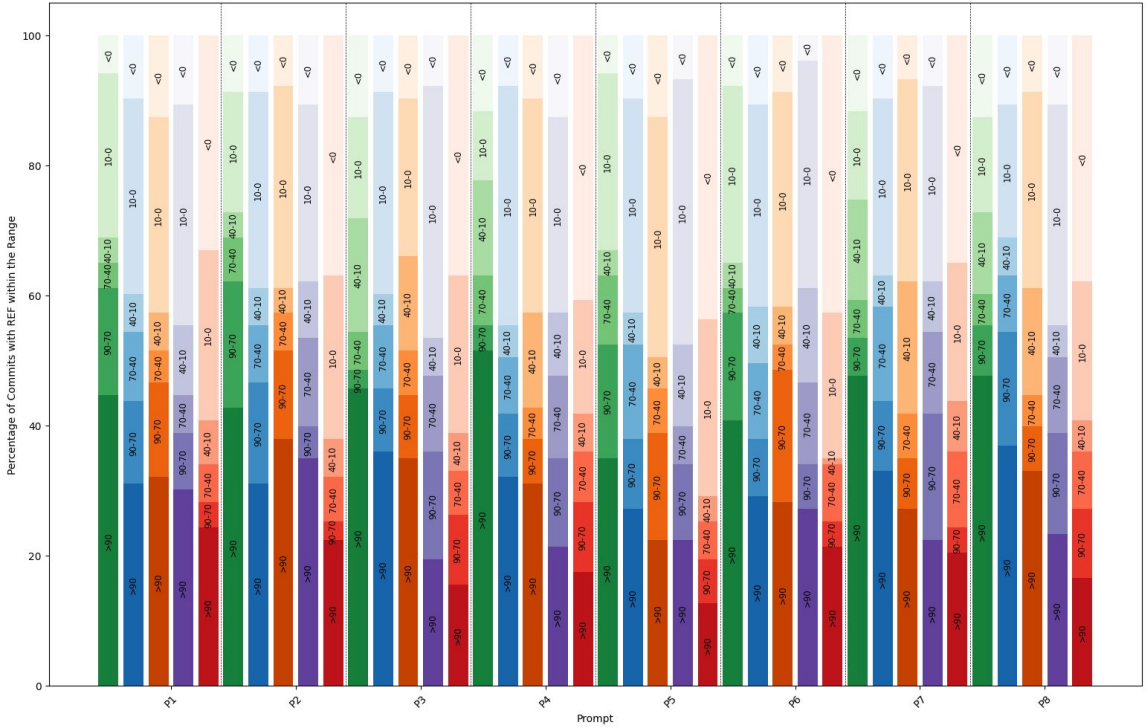
o3-mini : 41% of faulty files fixed

Compilation Error Fix Rate (CEFR)

Prompt ID	Deepseek V3	Gemini 2.0-flash	Gpt 4o-mini	o3 mini	Qwen2.5 32b-instruct
P1	548/938(58%)	680/959(71%)	490/965(51%)	705/941(75%)	529/979(54%)
P2	645/941(69%)	661/955(69%)	548/966(57%)	683/916(75%)	491/983(50%)
P3	553/935(59%)	679/931(73%)	696/979(71%)	736/955(77%)	710/987(72%)
P4	614/942(65%)	687/936(73%)	670/964(70%)	726/938(77%)	622/983(63%)
P5	555/937(59%)	684/978(70%)	531/962(55%)	731/964(76%)	720/994(72%)
P6	534/933(57%)	654/943(69%)	536/973(55%)	712/944(75%)	712/988(72%)
P7	664/921(72%)	711/962(74%)	696/976(71%)	723/937(77%)	669/986(68%)
P8	679/934(73%)	714/959(74%)	680/973(70%)	741/955(78%)	602/997(60%)

Qwen : 9% build success → 72% of original errors fixed in some cases

Relative Error Fix (REF)



o3-mini + P4 → **93.33%**

Some LLMs (e.g. Qwen2.5) introduced more errors than they fixed

Buggy line + API Diff consistently helpful

CoT works only for reasoning capable models

Conclusion

- **LLMs can effectively repair compilation errors caused by breaking dependency updates**, significantly reducing the manual effort required by developers.

- **Including structured context in prompts (buggy line, API differences, and step-by-step reasoning) greatly improves repair success**, leading to accurate fixes with minimal new errors introduced.

Fixing breaking dependency updates with LLM

Frank Reyes (frankrg@kth.se)