# RQ4: Error Analysis

## 1 ERROR ANALYSIS

We conduct a thorough error analysis over three rounds of experiments, ultimately identifying the following common errors in repository-level code translation, which are classified into five categories: E1: Configuration File (“`pom.xml`”) Issues, E2: Limited Understanding, E3: Incomplete Code Generation, E4: Language Feature (“Next Code Token” Predictors) Issues, E5: Encoding Issues.

**E1. Configuration File (“`pom.xml`”) Issues:** These errors arise from misconfigurations in the project’s “`pom.xml`” file, causing failures in dependency resolution, Java version mismatches, or other configuration-related issues.

- **E1.1. Unresolved Dependency Error:** This error occurs because the artifact can not be found in the central Maven repository. As a result, Maven is unable to resolve the necessary dependencies for the project.
- **E1.2. Unsupported Java Version Error:** Since the “`pom.xml`” file does not explicitly specify the Java version, Maven defaults to a version. Historically, Maven’s default source and target version was 1.5, which might not match the required version. To resolve this issue, the Java version must be explicitly set in the “`pom.xml`” file, for example, specifying version 1.8 as shown in the figure.
- **E1.3. Missing or Misconfigured Dependency:** The error message indicates that the package `org.apache.http.client.fluent` is not found in the project. This happens when the required dependency for Apache HttpClient Fluent API is missing or improperly configured in the “`pom.xml`”).
- **E1.4. Inappropriate Content in config file:** The issue arises when content that does not belong in the “`pom.xml`” file is added, leading to errors during project setup.

**E2. Limited Understanding:** These errors are caused by a lack of familiarity with class structures, constructor arguments, and Java syntax, leading to issues such as incorrect class instantiation or naming conventions.

- **E2.1. Calling a Non-existent Constructor:** The error message shows that the EventEmitter constructor `EventEmitter ee = new EventEmitter(true);` does not match any available constructors in the EventEmitter class. Though the class has multiple constructors, none accepts a single boolean argument.
- **E2.2. Incorrect Constructor Arguments:** The code attempts to instantiate an `FMFTRLMachine` object with a constructor whose argument list does not match the expected parameters. The `FMFTRLMachine` constructor requires 12 parameters of types `int, double, double, double, double, double, int, double, double, double, double, double`. However, only 11 arguments are provided, causing a mismatch in the constructor signature and leading to a compile-time error.
- **E2.3. Mismatch Between File Name and Class Name:** The class “`Demoji`” is declared as “`public`”, but the file name is incorrectly spelled as “`DeMoji.java`”. In Java, the file name must exactly match the class name, including case sensitivity, for public classes. Hence, the file should be renamed to “`Demoji.java`” to prevent the compiler from throwing an error.
- **E2.4. Method Redefinition:** The error occurs because the class “`ShortUrl`” contains two methods with the same name “`encodeUrl(int, int)`”, leading to a conflict in the method signature. Java allows method overloading if the parameter types or number of parameters

Author’s address:

E1.1

```
<dependency>
    <groupId>com.github.rjeschke</groupId>
    <artifactId>txtmark</artifactId>
    <version>0.16</version>
</dependency>
```

[ERROR] Could not resolve dependencies for project com.example:readtime-java:jar:1.0-SNAPSHOT: Could not find artifact com.github.rjeschke:txtmark:jar:0.16 in central (https://repo.maven.apache.org/maven2)

E1.2

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.8.1</version>
    <configuration>
        <source>1.8</source>
        <target>1.8</target>
    </configuration>
</plugin>
```

[ERROR] Source option 5 is no longer supported. Use 6 or later.
[ERROR] Target option 1.5 is no longer supported. Use 1.6 or later.

E1.3

```
import org.apache.http.client.fluent.Request;
```

[ERROR] src/main/java/StackAPI.java:[1,37] package org.apache.http.client.fluent does not exist

E1.4

```
@Override
protected T crossover(T parent1, T parent2) {
    double randDouble = 0.5; // Fix: Determine a sensible
legacy strategy (default 0.1)
```

[ERROR] Some problems were encountered while processing the POMs:
[FATAL] Non-parseable POM /home/repos/100SolidJavaDebug2/pom.xml: only whitespace content allowed before start tag and not @ (position: START_DOCUMENT seen @... @1:2) @ line 1, column 2 @

E2.1

```
void testOnWildcards() {
    EventEmitter ee = new EventEmitter(true);
    List<Object> stack = new ArrayList<>();
    Object token = new Object();
```

[ERROR] src/test/java/SyncTestCase.java:[77,27] no suitable constructor found for EventEmitter(boolean)
constructor EventEmitter.EventEmitter() is not applicable (actual and formal argument lists differ in length)
constructor EventEmitter.EventEmitter(boolean,boolean,int) is not applicable (actual and formal argument lists differ in length)
constructor EventEmitter.EventEmitter(java.lang.String,boolean,boolean,int) is not applicable (actual and formal argument lists differ in length)

E2.2

```
FMFTRLMachine learner = new FMFTRLMachine(fmDim,
fmInitDev, L1, L2, L1Fm, L2Fm, D, alpha, beta, alphaFm,
betaFm);
```

[ERROR] src/main/java/RunModelExample.java:[28,33] constructor FMFTRLMachine in class FMFTRLMachine cannot be applied to given types;
required: int,double,double,double,double,double,int,double,double,double,double,double
found: int,double,double,double,double,double,int,double,double,double,double
reason: actual and formal argument lists differ in length

E2.3

```
// Filename: DeMoji.java
public class Demoji {
    public static Pattern emojiPattern;
    public static Map<String, String> codeToDescMap;
    ...
```

[ERROR] src/main/java/DeMoji.java:[25,8] class Demoji is public, should be declared in a file named Demoji.java

E2.4

```
    public static String encodeUrl(int n, int min_length) {
        UrlEncoder encoder = new UrlEncoder(DEFAULT_ALPHABET, DEFAULT_BLOCK_SIZE);
        return encoder.encodeUrl(n, min_length);
    }
    public String encodeUrl(int n, int min_length) {
        return enbase(encode(n), min_length);
    }
```

[ERROR] src/main/java/ShortUrl.java:[45,19] method encodeUrl(int,int) is already defined in class ShortUrl

E2.5

```
import RegexExpr;
```

[ERROR] src/main/java/impl/SequenceExpr.java:[3,17] '.' expected

E3.1

```
Iterator<Map.Entry<String, JsonNode>> fields =
jsonNode.fields();
```

[ERROR] src/main/java/Forecast.java:[49,25] package Map does not exist

E3.2

```
public static void setDerivedUnitsAndConstants() {
    mS = 1e-3 * S;
    uS = 1e-6 * S;
    nS = 1e-9 * S;
    // Magnetic fields and fluxes
```

[ERROR] src/main/java/NumericalUnits.java:[264,23] reached end of file while parsing

E4.1

```
public static String encodeUrl(int n, int min_length) {
    UrlEncoder encoder = new UrlEncoder(DEFAULT_ALPHABET, DEFAULT_BLOCK_SIZE);
    return encoder.encodeUrl(n, min_length);
}
private class UrlEncoder {
    ...
```

[ERROR] src/main/java/ShortUrl.java:[36,30] non-static variable this cannot be referenced from a static context

E4.2

```
System.out.print(clui.c.lightBlue(frame) + " " +
clui.c.grey(message) + "\r");
public class Clui {
    private Colorize c;
    ...
```

[ERROR] src/main/java/Spinner.java:[29,30] c has private access in Clui
[ERROR] src/main/java/Spinner.java:[29,62] c has private access in Clui

E4.3

```
root = new UnitNode(new Position("root", "", -1), 0, "");

public abstract class UnitNode {
    protected Position position;
    ...
```

[ERROR] src/main/java/Tree.java:[12,16] UnitNode is abstract; cannot be instantiated

E4.4

```
// Filename: SteamAPI.java
public abstract class SteamAPI {
    public abstract String getJson(String url);
}
// Filename: SteamUser.java
public class SteamUser extends SteamAPI {
    // Missing the override of abstract method getJson
}
```

[ERROR] src/main/java/SteamUser.java:[1,8] SteamUser is not abstract and does not override abstract method getJson(java.lang.String) in SteamAPI

E4.5

```
public static RegexExpr repeated(RegexExpr expr, int
exactly) {
    return new RepeatedExpr(expr, exactly, exactly, true);
}
```

[ERROR] src/main/java/RegexExpr.java:[49,16] incompatible types: impl.RepeatedExpr cannot be converted to RegexExpr

E4.6

```
server.send("HTTP/1.0 " + httpCode + " " +
HTTP_CODES.get(httpCode) + "\r\n");
```

[ERROR] src/main/java/Utils.java:[19,20] unreported exception java.io.IOException; must be caught or declared to be thrown

E4.7

```
public static int[] findDiffStart(String oldSrc, String newSrc) {
    String[] oldLines = oldSrc.split("
");
```

[ERROR] src/main/java/FindDiff.java:[5,42] unclosed string literal
[ERROR] src/main/java/FindDiff.java:[6,1] unclosed string literal

E4.8

```
public static final String[] VERSION_INFO =.VERSION.split("\\.");
```

[ERROR] src/main/java/AppDirs.java:[6,48] illegal start of expression
[ERROR] src/main/java/AppDirs.java:[6,62] <identifier> expected
[ERROR] src/main/java/AppDirs.java:[6,63] illegal start of type

E5.1

```
System.out.println("🛠 Building image nytimes/blender:" + tag);
```

[ERROR] src/main/java/Build.java:[20,37] unmappable character (0xF0) for encoding US-ASCII

Fig. 1. Error Case Analysis.

differ. However, in this case, both methods have identical signatures, causing confusion for the compiler.

- **E2.5. Incorrect Import Statement:** The error "`"'.' expected"`" in the context of "`import RegexExpr;`" indicates that Java expects a dot ("`.`") to reference a package. Since "`RegexExpr`" is a class and does not need to be imported, the solution is to remove the import statement.

**E3. Incomplete Code Generation:** These errors result from incomplete code generation due to limitations in word count or missing elements, such as import statements, which lead to compilation issues.

- **E3.1. Missing Necessary Import Statement:** The error message "`package Map does not exist`" arises because the Map interface is not recognized by the Java compiler. The Map interface belongs to the "`java.util`" package, and the lack of an import statement for this package causes the error. This phenomenon often occurs when LLMs tend to generate example code while omitting elements they consider unimportant, such as "`import`" statements.
- **E3.2. Premature Termination of Code:** In this code snippet, the generated code is prematurely terminated due to the limitation on the maximum length of token generation. As a result, the complete code is not generated.

**E4. Language Feature ("Next Code Token" Predictor) Issues:** These errors are caused by inconsistent language features, which lead to referencing non-static classes from static contexts or failing to match return types, etc.

- **E4.1. Referencing Non-static Class from Static Context:** The error "`non-static variable this cannot be referenced from a static context`" occurs because the code attempts to create an instance of an inner class ("`UrlEncoder`") inside a static method ("`encodeUrl`"), but inner classes require an instance of the enclosing class to access their non-static members.
- **E4.2. Accessing Private Members:** The error message indicates that the member variable "`c`" in the "`Clui`" class is private and cannot be accessed directly from another class (such as "`Spinner.java`").
- **E4.3. Instantiating Abstract Class:** The error occurs because the code attempts to instantiate an abstract class ("`UnitNode`"). In Java, abstract classes cannot be instantiated directly and must be extended by a subclass that provides implementations for the abstract methods.
- **E4.4. Missing Method Implementation:** The error occurs because the subclass "`SteamUser`" does not provide an implementation for the abstract method "`getJson(String url)`" declared in the abstract class "`SteamAPI`". In Java, if a class extends an abstract class, it must either provide concrete implementations for all abstract methods or be declared as abstract. Since "`SteamUser`" is not declared abstract, the compiler expects it to implement the "`getJson`" method from "`SteamAPI`". To resolve the error, the subclass must override and implement the missing method.
- **E4.5. Incorrect Return Type:** The method "`repeated()`" is expected to return an instance of "`RegexExpr`", but it returns an instance of "`RepeatedExpr`". This mismatch between the expected and actual return types leads to a type incompatibility.
- **E4.6. Missing Expected Exception:** The method "`server.send()`" is expected to throw an "`IOException`", but it does not throw the exception as anticipated.
- **E4.7. Incorrect Newline Representation:** In Java, a newline character inside a string is represented by the escape sequence "`\n`". However, the generated code incorrectly represents this in the "`split()`" method, leading to errors.
- **E4.8. Invalid Syntax:** A dot ("`.`") is incorrectly placed after the equals sign ("="), causing the Java compiler to throw a syntax error. The correct syntax should place the dot immediately after the object or class from which "`VERSION`" is being referenced.

**E5. Encoding Issues:** These errors occur due to incompatible repository encoding formats, which can lead to issues when special characters, such as emojis, are used.

- **E5.1. Encoding Missing:** The error "`unmappable character (0xF0) for encoding US-ASCII`" occurs because the file is being compiled using the US-ASCII encoding, which does not support certain characters, such as emojis. Since emojis are Unicode characters, US-ASCII cannot represent them. The solution is to add "`<encoding>UTF-8</encoding>`" in the "`<configuration>`" tag of the "`pom.xml`" file to resolve this issue.