Phase 1: Token-Based Code Similarity Analysis

Phase 1 analyzes programs at the token level. The primary goal is to detect exact matches and differences in lexical elements of the code, such as identifiers, keywords, operators, and literals. This phase provides a quick initial similarity measurement and helps detect direct copying of code without structural transformations.

1. Tokenization Process

Each input file is read and processed using ANTLR's lexer, converting source code into a list of tokens. For each token, the system records type, text, line number, and column. Whitespace and comments are ignored.

2. Edit Distance for Token Comparison

Similarity is computed using a Levenshtein-like edit distance on tokens, considering insertion, deletion, and substitution. Similarity = 1 - (Token Edit Distance / Max Token Count). This produces a percentage similarity score.

3. Token Comparison Visualization

The system provides a token-level diff, highlighting [MATCH] for identical tokens and [DIFF] for differences. This visualization helps identify copied or modified lines.

4. Advantages

- Fast and computationally light - Detects exact token copying - Provides visual representation of differences - Useful for initial screening before deeper AST analysis Limitations: - Sensitive to variable/function renaming - Cannot detect logical equivalence if structure is rewritten

5. Example

Input 1: a = 1; b = 2; Input 2: b = 2; { a + 1; } Token-level diff shows differences in identifiers and numbers, but matches in operators and semicolons.

6. Conclusion

Phase 1 provides a fast and straightforward way to measure code similarity at the lexical level. It detects copied code and superficial changes effectively, producing a similarity percentage and token-level visualization. However, it is limited in handling variable/function renaming, statement

reordering, and expression rewriting. Phase 2 complements this phase by detecting deeper structural similarity and logical equivalence.