**Literature Review: Capability of Large Language Models in Understanding Code Semantics**

**1. Introduction**

Large Language Models (LLMs) have made significant progress in software engineering tasks such as **code generation, summarization, refactoring, and documentation**. However, one critical question remains: **Do these models truly understand code semantics, or do they rely solely on pattern recognition?**

In this literature review, we look at Thu-Trang Nguyen et al.'s *"An Empirical Study on the Capability of Large Language Models in Understanding Code Semantics,"* which introduces **EMPICA**, a framework for evaluating LLMs' ability to comprehend code semantics. The study investigates whether LLMs can distinguish between functionally equivalent and non-equivalent code by performing controlled transformations and assessing their impact on model predictions.

**2. Summary of the Paper**

**2.1 Objective**

The paper investigates the **robustness and sensitivity** of state-of-the-art LLMs for code semantics. The goal is to see if these models can distinguish between **semantically equivalent and non-equivalent** code modifications.

**2.2 Methodology**

The authors present **EMPICA**, a systematic framework that performs **eight controlled transformations** on input code:

* **Semantic-Preserving Transformations (SP)**: Retain program behavior (e.g., renaming variables and reordering parameters).
* **Semantic-Non-Preserving Transformations (SNP)**: Modify program behavior (e.g., removing conditional statements or negating relational conditions).

**EMPICA evaluates LLMs** for three software engineering tasks:

1. **Code Summarization**: Creating natural language summaries of code.
2. **Method Name Prediction**: Generating descriptive function/method names.
3. **Output Prediction**: Specifying the expected output of code execution.

The study compares four **cutting-edge LLMs**:

* **DeepSeek-Coder, Code Llama, MagicCoder, and GPT-3.5**

The benchmark datasets used for evaluation include:

* **HumanEval**: A widely used dataset in code generation research.
* **MBPP (Mostly Basic Python Problems)**: A dataset containing Python problems for evaluating code synthesis and understanding.
* **CodeContests**: A dataset comprising competitive programming problems to test model generalization in coding tasks.

**2.3 Key Findings**

**2.3.1 Robustness versus Sensitivity**

* **LLMs are more robust than sensitive**: They perform consistently on SP transformations but struggle to distinguish SNP transformations.
* **Code summarization is extremely robust**: Models produce similar summaries even when semantic changes are introduced, indicating a reliance on structural patterns rather than true semantic understanding.
* **Output prediction is the most sensitive task**: Models are better at detecting semantic changes when predicting code execution outcomes.

**2.3.2 Effects of Specific Transformations**

* **Variable renaming significantly impacts predictions**, despite preserving semantics, suggesting that LLMs rely on superficial cues rather than deep comprehension.
* **Removing conditional statements results in the most noticeable prediction changes**, indicating that control structures are critical to LLM decision-making processes.

**2.3.3 Model Size and Performance**

* **Increasing the model size does not significantly improve semantic sensitivity**: The 33B-parameter model did not outperform smaller models in terms of code semantics.
* **Control dependencies (e.g., if-else structures) are more accurately captured than data dependencies.**

**3. Reflection and Analysis**

**3.1 Contributions**

* The study takes a **quantitative and empirical approach** to assessing LLMs' code comprehension capabilities.
* The **EMPICA framework** provides a **novel method** for systematically assessing robustness and sensitivity in LLM predictions.
* The findings highlight the **limitations of LLMs in true semantic understanding**, which is critical for developing **AI-assisted coding tools**.

**3.2 Limitations**

* The analysis is **limited to three software engineering tasks**; additional tasks such as **bug detection and automated debugging** may provide more insights.
* The **datasets used may not accurately reflect diverse real-world coding practices**, necessitating a more comprehensive evaluation across multiple datasets.
* The study does not investigate the **effect of fine-tuning** on semantic sensitivity improvements.

**3.3 Future Research Directions**

* **Enhancing semantic sensitivity**: Future models should prioritize **deep code reasoning** over structural patterns.
* **Creating more comprehensive benchmarks**: Broadening evaluation to include **real-world repositories** (such as GitHub) can improve generalizability.
* **Developing hybrid models**: Combining LLMs with **static and dynamic analysis techniques** may improve their ability to capture true code semantics.

**4. Conclusion**

The study sheds light on the **strengths and weaknesses** of current LLMs for understanding code semantics. While these models are effective at **code summarization and method name prediction**, they struggle with **semantic comprehension and functional code differentiation**. The findings highlight the **need for better training techniques** to improve **AI-powered software development tools**.

**صورة تحتوي على نص, لقطة شاشة, رسم بياني, موازِ

تم إنشاء الوصف تلقائياً**

**References**

* **HumanEval Dataset**: <https://github.com/openai/human-eval>
* **MBPP (Mostly Basic Python Problems) Dataset**: <https://github.com/google-research/google-research/tree/master/mbpp>
* Thu-Trang Nguyen, Thanh Trong Vu, Hieu Dinh Vo, & Son Nguyen. *An Empirical Study on Capability of Large Language Models in Understanding Code Semantics*. arXiv preprint, 2024. <https://www.semanticscholar.org/reader/0587ba2247e1e140ce8b776517ee931ad72e7156>