The ability of LLMs on naming conventions

Group 7

An Empirical Study on Capability of LLMs in understanding Code Semenitcs

- 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.

An Empirical Study on Capability of LLMs in understanding Code Semenitcs

- 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.

An Empirical Study on Capability of LLMs in understanding Code Semenitcs

- 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 :
- .1Code Summarization: Creating natural language summaries of code.

- .2 Method Name Prediction: Generating descriptive function/method names .

- .3Output 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.

An Empirical Study on Capability of LLMs in understanding Code Semenitcs

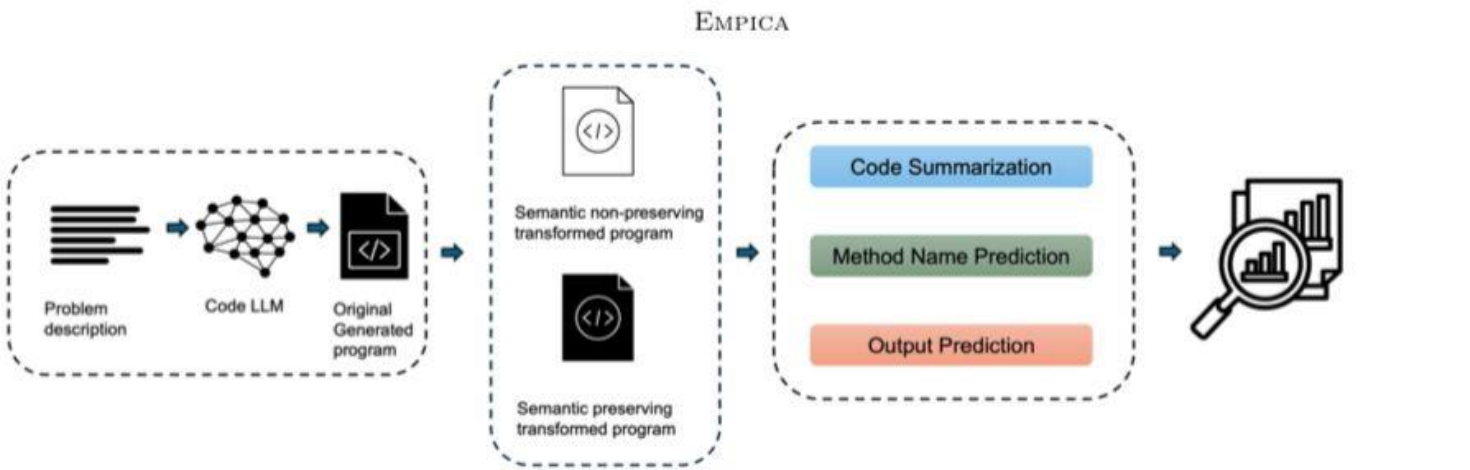

Figure 3: Framework overview

An Empirical Study on Capability of LLMs in understanding Code Semenitcs

- 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 .
- 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. 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.

An Empirical Study on Capability of LLMs in understanding Code Semenitcs

- 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, 0587/redaer/gro.ralohcscitnames.www//:sptth .2024ba2247e1e140ce8b776517ee931ad72e7156

Automated Variable Renaming Using Large Language Models (LLMs)

- **Introduction**
- Variable names significantly impact code comprehension, maintainability, and readability. However, developers frequently use inconsistent or uninformative variable names, making code harder to understand. Automated variable renaming aims to enhance identifier quality using **machine learning (ML) techniques**, including **statistical language models, deep learning models, and static code analysis**.

- **Research Goal idea**
- The study investigates how well **state-of-the-art language models** can predict **meaningful variable names**, potentially **automating rename refactoring**. The authors evaluate three models:

Automated Variable Renaming Using Large Language Models (LLMs)

- **Methodology**
- **N-gram Cached Language Model** – A statistical model that predicts words based on preceding sequences [2].
- **T5 Transformer Model** – A deep learning model trained to understand programming languages [3].
- **CugLM (Transformer-based Code Completion Model)** – A neural model designed for predicting identifiers in source code [4].
- These models are tested on datasets derived from **real-world software projects**, comparing their ability to **recommend variable names matching developers' choices**.

Automated Variable Renaming Using Large Language Models (LLMs)

- **Datasets**
- The study constructs **three datasets** to evaluate the models:
- **Large-Scale Dataset:**
  - 1,221,193 Java method instances with local variables.
  - Extracted from **1,425 GitHub repositories** to train and test models.
- **Reviewed Dataset:**
  - 457 Java methods with variable renaming that occurred **during code reviews**.
  - Ensures **higher-quality identifiers** that multiple developers agreed upon.
- **Developers' Dataset:**
  - 442 instances of **variable renaming extracted from rename-refactoring commits**.
  - Represents **real-world refactoring** cases performed by developers.

- Each model is trained on the large-scale dataset and evaluated on **all three datasets**

Automated Variable Renaming Using Large Language Models (LLMs)

Evaluation Metrics

The performance of the models is assessed using:

- **Prediction Accuracy:** Measures how often the predicted identifier matches the developer's choice.
- **Confidence Score Analysis:** Examines how confident the models are in their predictions and correlates confidence with correctness.
- **McNemar's Test & Odds Ratio (OR):** Used for **statistical comparison** of model performance.
- **Qualitative Analysis:** Manually inspects incorrect predictions to determine if they are still **meaningful**alternatives.

Automated Variable Renaming Using Large Language Models (LLMs)

Key Findings

- **CugLM significantly outperforms other models**, achieving **63.46% accuracy** on the large-scale dataset.
    - **T5 Transformer achieves 37.35%**, while the **N-gram model lags at 10.54%**.
- **Performance drops on high-quality datasets** (reviewed & developers' datasets), indicating **challenges in predicting meaningful names**.
- **Confidence scores strongly correlate with correctness**, suggesting models can be used in **refactoring tools with confidence thresholds**.
- **Some incorrect predictions are still useful**, offering **valid alternative names** (e.g., sum → totalAmount).

Automated Variable Renaming Using Large Language Models (LLMs)

- **References**

- **[1] A. Mastropaolo, E. Aghajani, L. Pascarella, and G. Bavota, “Automated Variable Renaming: Are We There Yet?”Empirical Software Engineering, vol. 28, no. 45, 2023. DOI: 10.1007/s10664-022-10274-8.**
- **[2] V. Hellendoorn and P. Devanbu, "Are Deep Neural Networks the Best Choice for Modeling Source Code?"Proceedings of the 2017 ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE), 2017, pp. 763–773.**
- **[3] C. Raffel et al., “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”, Journal of Machine Learning Research, vol. 21, pp. 1–67, 2020.**
- **[4] C. Liu et al., “Multi-Task Learning for Code Completion”, Proceedings of the 2020 ACM/IEEE International Conference on Automated Software Engineering (ASE), 2020, pp. 473–485.**

How Does Naming Affect Language Models on Code Analysis Tasks?

- Idea:
- The study is investigating how the naming of variables, methods, and functions impacts the performance of LLMs such as CodeBERT and Chatgbt in code analysis tasks. The main idea is creating a dataset that is misleading with goofy names intentionally. And perform code analysis tasks on these datasets with a well-trained model such as (CodeBERT)

How Does Naming Affect Language Models on Code Analysis Tasks?

- Methods: The authors designed experiments to systematically manipulate code naming conventions and measure the impact on LLMs performance. They categorized code features into literal features such as (naming of variables, functions, and comments) and logical features (keywords and operators that define the control flow).
- Steps followed as :
    - Dataset creation
    - Model selection and fine-tuning
    - tasks

How Does Naming Affect Language Models on Code Analysis Tasks?

Dataset

- Contains code and corresponding documentation pairs for multiple languages (Java,Python,etc.) Used for the search code task.
- BigCloneBench:
- A benchmark dataset consisting of known code clones across different java projects. Used for the clone detection task.

- Anonymized Datasets:
- 16variations are created by anonymizing variables names or methods names using random and shuffle naming strategies.

How Does Naming Affect Language Models on Code Analysis Tasks?

	Task	Language	Var.[1]	Def.	Inv.	Anon.[2]	Size
D1	Code Search	Java & Python	51	55	55	Rand.	175,878 & 266,738[3]
D2	Code Search	Java & Python	51	55	55	Shuff.	175,878 & 266,738
D3	Code Search	Java & Python	55	51	55	Rand.	175,878 & 266,738
D4	Code Search	Java & Python	55	51	55	Shuff.	175,878 & 266,738
D5	Code Search	Java & Python	55	55	51	Rand.	175,878 & 266,738
D6	Code Search	Java & Python	55	55	51	Shuff.	175,878 & 266,738
D7	Code Search	Java & Python	51	51	51	Rand.	175,878 & 266,738
D8	Code Search	Java & Python	51	51	51	Shuff.	175,878 & 266,738
D9	Clone Dete.	Java	51	55	55	Rand.	1316,444
D10	Clone Dete.	Java	51	55	55	Shuff.	1316,444
D11	Clone Dete.	Java	55	51	55	Rand.	1316,444
D12	Clone Dete.	Java	55	51	55	Shuff.	1316,444
D13	Clone Dete.	Java	55	55	51	Rand.	1316,444
D14	Clone Dete.	Java	55	55	51	Shuff.	1316,444
D15	Clone Dete.	Java	51	51	51	Rand.	1316,444
D16	Clone Dete.	Java	51	51	51	Shuff.	1316,444

How Does Naming Affect Language Models on Code Analysis Tasks?

- Evaluation Metrics:
- -1Mean Reciprocal Rank (MRR):
- Used to evaluate the code search task. It measures how well the model ranks the correct code snippet for a given query.[4]
- Higher MRR indicates better performance.
- -2F1 Score:
- Used for the clone detection task to assess precision and recall in identifying similar code snippets.[14]
- Higher F1 reflects more accurate detection of code clones.

How Does Naming Affect Language Models on Code Analysis Tasks?

Table 2. Results on Code Search. The results are shown in $\frac{MRR}{Dataset}$ format.

Language	Orig.	Anon.	w/o Var.	w/o Def.	w/o Inv.	All
Java	70.36%	Random	$\frac{67.73\%}{\mathscr{D}1}$	$\frac{60.89\%}{\mathscr{D}3}$	$\frac{69.84\%}{\mathscr{D}5}$	$\frac{17.42}{\mathscr{D}7}$
		Shuffling	$\frac{67.14\%}{\mathscr{D}2}$	$\frac{58.36\%}{\mathscr{D}4}$	$\frac{69.84\%}{\mathscr{D}6}$	$\frac{17.03\%}{\mathscr{D}8}$
Python	68.17%	Random	$\frac{59.8\%}{\mathscr{D}1}$	$\frac{55.43\%}{\mathscr{D}3}$	$\frac{65.61\%}{\mathscr{D}5}$	$\frac{24.09\%}{\mathscr{D}7}$
		Shuffling	$\frac{59.78\%}{\mathscr{D}2}$	$\frac{55.65\%}{\mathscr{D}4}$	$\frac{65.61\%}{\mathscr{D}6}$	$\frac{23.73\%}{\mathscr{D}8}$

Table 3. Results on Clone Detection. The results are shown in $\frac{F1}{Dataset}$ format.

Language	Orig.	Anon.	w/o Var.	w/o Def.	w/o Inv.	All
Java	94.87%	Random	$\frac{92.64\%}{\mathscr{D}9}$	$\frac{93.97\%}{\mathscr{D}11}$	$\frac{94.72\%}{\mathscr{D}13}$	$\frac{86.77\%}{\mathscr{D}15}$
		Shuffling	$\frac{92.52\%}{\mathscr{D}10}$	$\frac{94.27\%}{\mathscr{D}12}$	$\frac{93.67\%}{\mathscr{D}14}$	$\frac{84.76\%}{\mathscr{D}16}$

How Does Naming Affect Language Models on Code Analysis Tasks?

Result:
Clone detection performance dropped when names were anonymized. The decline was less compared to code search.This suggests that LLMs can still recognize code similarity through logical structures. Even when .naming conventions is anonymized

How Does Naming Affect Language Models on Code Analysis Tasks?

- Reflection:
- Key Findings:
  This paper highlights a critical limitation of LLMs in code analysis: their heavy reliance on well-defined naming conventions. It underscores the need for models that better understand the logical structure of code, not just its surface-level semantics.
- Limitations:
  The study focused primarily on CodeBERT and ChatGPT. Expanding the experiments to other LLMs like GraphCodeBERT or more recent versions of GPT might yield broader insights.
- Areas for future exploration:
  Future work could explore training models to focus more on logical code features rather than literal names. Additionally, enhancing LLMs' resilience to poorly named or obfuscated code could improve their real-world application, especially in security-sensitive contexts like malware detection or decompiled code analysis.

How Does Naming Affect Language Models on Code Analysis Tasks?

- References:
- [1]Sarzynska-Wawer, J., Wawer, A., Pawlak, A., Szymanowska, J., Stefaniak, I., Jarkiewicz, M., et al. (thguohT lamroF gnitceteD (2021 yrtaihcysP .snoitatneserpeR droW dezilautxetnoC peeD yb redrosiD .114135 :DI elcitrA ,304 ,hcraeseR 2021.114135.serhcysp.j/10.1016/gro.iod//:sptth
- [4]Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., et al. (larutaN dna gnimmargorP rof ledoM deniarT-erP A :trebedoC (2020 :scitsiugniL lanoitatupmoC rof noitaicossA eht fo sgnidniF .segaugnaL -1536 ,2020 rebmevoN ,2020 PLNME .1547https://doi.org/139.plnme-sgnidnif.1/2020v/10.18653
- [5]Guo, D.Y., Ren, S., Lu, S., Feng, Z.Y., Tang, D.Y., Liu, S.J., Zhou, L. and Duan, N., et al. (edoC gniniarT-erP :trebedochparG (2020 .wolF ataD htiw snoitatneserpeR
- [6]Guo, Q., Cao, J., Xie, X., Liu, S., Li, X., Chen, B., et al. ((2024 :tnemenifeR edoC detamotuA ni TPGtahC fo laitnetoP eht gnirolpxE 46 MCA/EEEI eht fo sgnideecorP .ydutS laciripmE nAth International Conference on Software Engineering, Lisbon, 20-14April -1 ,2024 .13https://doi.org/10.1145/3597503.3623306

Enhancing Identifier Naming with Large Language Models (LLMs)

- Introduction
- Code readability plays a crucial role in software maintainability and comprehension. Poorly named identifiers can make code difficult to understand and maintain, leading to increased debugging time and errors. Automated identifier renaming has emerged as a promising approach to improving code quality by leveraging Large Language Models (LLMs).

Enhancing Identifier Naming with Large Language Models (LLMs)

- **Methodology and Datasets**
- The research utilizes **masked language modeling (MLM)** to train a **GraphCodeBERT** model specifically for identifier renaming. The key dataset includes:
- **236,745 high-quality identifiers** from **50 GitHub repositories**.
- Full Java class contexts to maintain identifier relationships.
- Filtration techniques to remove **low-quality and ambiguous identifiers**.

- The model's performance is assessed using:
- Pseudo-perplexity (PPPL) –Measures how confidently the model predicts masked tokens.
- Negative Pseudo-Likelihood (PLL) –Evaluates the likelihood of correct identifier generation.
- Developer Survey –Gathers insights on identifier quality and readability.

Generating Function Names to Improve Comprehension of Synthesized Programs

- **In modern software development, program synthesis—where AI generates code automatically—is becoming increasingly prevalent. However, one of the biggest challenges in synthesized programs is the lack of meaningful function names. Without clear function names, developers struggle to understand the purpose of automatically generated code,**

Generating Function Names to Improve Comprehension of Synthesized Programs

- Methodology:
- AI-Powered Function Name Generation
- Transformer-based language models can analyze synthesized code, extract contextual meaning, and propose more appropriate function names. This approach improves comprehension by:
- Generating semantically rich function names based on actual function logic.
- Eliminating vague or overly generic names.
- Assisting developers in understanding AI-generated functions more quickly.

Generating Function Names to Improve Comprehension of Synthesized Programs

- Dataset:
- Example Dataset for Training AI Models
- To enable AI models to generate accurate function names, datasets are built using high-quality open-source projects. The dataset includes examples of well-named and poorly named functions, allowing the model to learn naming patterns.