How Does Naming Affect Language Models on Code Analysis Tasks?

**Summary**: 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**) [4].**

**Idea**: The study is stating how variables, methods and functions naming conventions affect the performance of LLMs in code analysis tasks. Since programming languages allow flexible naming conventions, unlike natural languages models use fixed naming conventions for words, the author hypothesizes that poorly or misleading names identifiers could noticeably reduce the performance of LLMs in tasks such as code search or even clone detection. The study aims to measure this effect and evaluate if LLMs depend on literal code features or understanding the logic of the code [5][6].

**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 in methodology as follows:

1. **Dataset Creation:**

* They developed eight datasets where variable names or methods were anonymized such as random naming (e.g. abc123) or shuffle naming (e.g. renaming bubble\_sort to encrypted\_data [5][13][14].

1. **Model selection and Fine-tuning**

* Models used: CodeBERT [4] and GraphCodeBERT [5], which are already pre-trained.
* Fine-tuned these models on the manipulated datasets for two specific code analysis tasks: natural language code search and clone detection.

1. **Tasks**:

* Code Search: Finding the most relevant code snipped for a natural language query [13].
* Clone detection: Identifying if two code fragments perform the same function, despite differences in implementation [14]

**DataSets**:

1. CodeSearchNet Corpus:

* Contains code and corresponding documentation pairs for multiple languages (Java,Python,etc.) Used for the search code task[13].

1. BigCloneBench:

* A benchmark dataset consisting of known code clones across different java projects. Used for the clone detection task[14].

1. Anonymized Datasets:

* 16 variations are created by anonymizing variables names or methods names using random and shuffle naming strategies.

**Evaluation Metrics:**

1- Mean 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.

2- F1 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.

**Results:**

Impact on code search (MRR Scores):

 Original Dataset (No Anonymization):

* Java: 70.36% MRR
* Python: 68.17% MRR

 Anonymizing Variable Names:

* Java (Random): 67.73%
* Python (Random): 59.8%

 Anonymizing Method Names:

* Java (Random): 60.89%
* Python (Random): 55.43%

 Anonymizing All Names:

* Java: 17.42% (Random), 17.03% (Shuffled)
* Python: 24.09% (Random), 23.73% (Shuffled)

**Result Insight:**

The performance dropped drastically when all names were anonymized,

especially in Java where the MRR fell to as low as 17.03%. This indicates LLMs strong reliance on meaningful naming conventions for effective code search.

2- **Impact on Clone Detection (F1 Scores):**

* Original Dataset (No Anonymization): 94.87% F1
* Anonymizing Variable Names: 92.64% (Random), 92.52% (Shuffled)
* Anonymizing Method Names: 93.97% (Random), 94.27% (Shuffled)
* Anonymizing All Names: 86.77% (Random), 84.76% (Shuffled)

**Result insight:**

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.

**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[4][5].
* **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[6][11].
* **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[15]16].

**References:**

[1] Sarzynska-Wawer, J., Wawer, A., Pawlak, A., Szymanowska, J., Stefaniak, I., Jarkiewicz, M., et al. (2021) Detecting Formal Thought Disorder by Deep Contextualized Word Representations. Psychiatry Research, 304, Article ID: 114135. https://doi.org/10.1016/j.psychres.2021.114135

[4] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., et al. (2020) Codebert: A Pre-Trained Model for Programming and Natural Languages. Findings of the Association for Computational Linguistics: EMNLP 2020, November 2020, 1536-1547. <https://doi.org/10.18653/v1/2020.findings-emnlp.139>

[5] Guo, D.Y., Ren, S., Lu, S., Feng, Z.Y., Tang, D.Y., Liu, S.J., Zhou, L. and Duan, N., et al. (2020) Graphcodebert: Pre-Training Code Representations with Data Flow.

[6] Guo, Q., Cao, J., Xie, X., Liu, S., Li, X., Chen, B., et al. (2024) Exploring the Potential of ChatGPT in Automated Code Refinement: An Empirical Study. Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, Lisbon, 14-20 April 2024, 1-13. <https://doi.org/10.1145/3597503.3623306>

[7] Chen, C., Su, J.Z., Chen, J.C., Wang, Y.L., Bi, T.T., Wang, Y.L., et al. (2023) When ChatGPT Meets Smart Contract Vulnerability Detection: How Far Are We?

[9] Katz, D.S., Ruchti, J. and Schulte, E. (2018) Using Recurrent Neural Networks for Decompilation. 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), Campobasso, 20-23 March 2018, 346-356. <https://doi.org/10.1109/saner.2018.8330222>

[11] OpenAI (2023) Introducing ChatGPT.

[12] Feng, Z.Y., Guo, D.Y., Tang, D.Y., Duan, N., Feng, X.C., Gong, M., et al. (2021) Code Pretraining Models.

[13] Husain, H., Wu, H.-H., Gazit, T., Allamanis, M. and Brockschmidt, M. (2019) CodeSearchNet Challenge: Evaluating the State of Semantic Code Search.

[14] Svajlenko, J., Islam, J.F., Keivanloo, I., Roy, C.K. and Mia, M.M. (2014) Towards a Big Data Curated Benchmark of Inter-Project Code Clones. 2014 IEEE International Conference on Software Maintenance and Evolution, Victoria, 29 September-3 October 2014, 476-480. <https://doi.org/10.1109/icsme.2014.77>

[15] Surameery, N.M.S. and Shakor, M.Y. (2023) Use Chat GPT to Solve Programming Bugs. International Journal of Information technology and Computer Engineering, 3, 17-22. https://doi.org/10.55529/ijitc.31.17.22 [16] Cheshkov, A., Zadorozhny, P. and Levichev, R. (2023) Evaluation of ChatGPT Model for Vulnerability Detection.