Paper: Vagavolu, D., Swarna, K. C., & Chimalakonda, S. (2021). *A Mocktail of Source Code Representations*. Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE).

This paper studies the problem of source code representation for software engineering tasks. The authors investigate whether combining multiple representations of source code can improve the performance of models used in tasks such as method naming, code classification, and code clone detection. The paper focuses specifically on integrating syntactic and semantic program representations. The authors examine whether combining these representations provides better semantic understanding of programs compared to using only syntactic representations, which is common in existing approaches. The central research question of the paper is whether paths extracted from multiple code representations can improve the performance of software engineering tasks. In particular, the authors explore whether integrating semantic structures such as Control Flow graph (CFG) and Program Dependence graph (PDG) with syntactic structures like Abstract Syntax Trees (AST) can improve code representation models.

Problem: Most existing approaches to representing source code for machine learning rely primarily on AST-based representations. While AST captures syntactic structure, it does not fully represent semantic information such as control flow and data dependencies. This limitation may reduce the effectiveness of machine learning models for software engineering tasks.

Claims: The authors claim that combining syntactic and semantic representations can significantly improve the quality of learned code representations and improve performance in downstream tasks such as method name prediction.

Evidence: To evaluate this claim, the authors extend the existing *code2vec* model to include path contexts extracted from AST, CFG, and PDG. They construct a dataset consisting of approximately 730,000 C methods collected from 16 open-source GitHub projects and on some individual projects. Paths are extracted from these graph representations and used as inputs to an attention-based neural network that generates code embeddings.

Statistical Analysis: The model is evaluated on the *method naming* task, where the goal is to predict the name of a method from its source code. The authors use the F1 score as the primary evaluation metric, computed using precision and recall over predicted subwords in the method names. The results show that integrating AST, CFG, and PDG paths improves performance compared to the baseline *code2vec* model. Specifically, the proposed approach improves the F1 score by approximately 11% on the full dataset and up to 100% improvement on some individual projects.