Beyond Accuracy: Evaluating Source Code Capabilities in Large Language Models for Software Engineering

Alejandro Velasco
svelascodimate@wm.edu
William & Mary
Williamsburg, Virginia, USA

ABSTRACT

This dissertation aims to introduce interpretability techniques to comprehensively evaluate the performance of Large Language Models (LLMs) in software engineering tasks, beyond canonical metrics. In software engineering, Deep Learning techniques are widely employed across various domains, automating tasks such as code comprehension, bug fixing, code summarization, machine translation, and code generation. However, the prevalent use of accuracy-based metrics for evaluating Language Models trained on code often leads to an overestimation of their performance. Our work seeks to propose novel and comprehensive interpretability techniques to evaluate source code capabilities and provide a more nuanced understanding of LLMs performance across downstream tasks.

CCS CONCEPTS

• **Computing methodologies** → **Cross-validation**; **Knowledge representation and reasoning**; • **Software and its engineering** → *General programming languages*; **Source code generation**.

KEYWORDS

Large Language Models, Interpretability, DL4SE, Category Theory, Causal Inference

ACM Reference Format:
Alejandro Velasco. 2024. Beyond Accuracy: Evaluating Source Code Capabilities in Large Language Models for Software Engineering. In *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3639478.3639815

1 PROBLEM AND RESEARCH STATEMENT

With the advent of Large Language Models (LLMs), code generation has been addressed as a *machine learning problem* in downstream tasks such as code completion [11], program repair [10], and test case generation [26]. Moreover, industry interest in leveraging LLMs for code completion has also grown as evidenced by tools such as Microsoft's IntelliCode [18], Tabnine [2], OpenAI's Codex [28], and GitHub's Copilot [12]. Adopting LLMs for code in practical settings might potentially bring risks associated with the efficacy and trustworthiness of these machines [17]. LLMs, which are described often in terms of probabilistic distributions, can sometimes generate correct outputs with high confidence even when the input features do not contain semantically meaningful information (*i.e.,* predicting identifiers from indentation tokens in the prompt). This previous erroneous condition of LLMs is formally known as *over-interpretation* [15]. The intricate nature of LLMs, combined with the way information is learned and encoded in the hidden layers, makes the evaluation process of LLMs particularly complex.

Although scaling up LLMs have been shown to enhance their performance significantly [27] by exhibiting ***emergent abilities***, LLMs for code have been not rigorously evaluated on those emergent abilities. In fact, such emergent abilities have not been fully explored yet by software researchers. A key unresolved problem in the field of Deep Learning for Software Engineering (DL4SE) is the understanding of specific conditions by which a LLM learns complex semantics from mere observational data. This lack of understanding represents a significant gap in the evaluation of NCMs, particularly in explaining these emergent behaviors. Despite the significant progress in the development of techniques for assessing LLMs, the evaluation of code generation models in software engineering remains limited. In other words, current canonical metrics fall short of offering insights into the extent to which the evaluated models learn *meaningful semantic information* from the source code.

Current evaluation approaches omit coarse-granular semantic properties such as *Reliability*, *Maintainability*, *Correctness*, *Security*, and *Robustness*. These semantic properties not only reflect the true quality of code predictions but also represent a crucial distinguishing factor between the capabilities of human programmers and those of trained models. By considering these enriched semantic aspects in the evaluation of proposed architectures, we can improve future research in deep learning for software engineering.

In this dissertation, we aim to extend the boundaries of *DL4SE evaluation* by developing techniques to assess the behavior of LLMs beyond accuracy. We also seek to bridge the gap between canonical accuracy metrics (*e.g.,* BLEU [22], ROUGE [16], METEOR [4], CodeBLEU [1]) and meaningful semantic information. Our approach adopts elements from Control and Data-Flow analysis and incorporates mathematical frameworks such as Causal Inference (CI) and Category Theory.

2 PROPOSED RESEARCH

Considering the recent literature on interpretability and the inherent challenges in comprehensively evaluating the decisions made by LLMs in software engineering tasks, our approach is threefold. Firstly, our aim is to gain a comprehensive understanding of the nature of the semantic information these models are expected to capture. Second, we will evaluate how well current architectures learn such knowledge and the underlying reasons behind their decision-making processes. Finally, we propose developing error detection and tools, to provide software practitioners and researchers with effective means to assess the reliability of selected models for specific downstream tasks. The following subsections will explain the rationales of each point.

2.1 Modeling Semantics

In simpler terms, Language Models are probability distributions $P(x_t|x_1, x_2, \ldots, x_{t-1})$, where the output x_t at time step t, given the input sequence $(x_1, x_2, .., x_{t-1})$, is inferred through the conditional probability $P(x_t|h_t)$. The hidden state h_t encapsulates the properties of the preceding context. LLMs are trained with a huge amount of data, and the text used in the training is totally *unstructured.* In other words, no grammatical or semantic rules or information are explicitly given to the model, yet such complex architectures seem to produce coherent outputs. Producing coherent outputs implies that some semantic information of the programming language (PL) must also be learned (*e.g.,* a return statement should be always at the end of a function). LLMs learn this complex information just by seeing samples of coherent text.

PLs are not only algebraic (*i.e.,* elements are combined together to form coherent text) but also statistical [6]. For example, the expression $'variable{+}{+}'$ may occur more frequently than $'variable{+}=1'$, and the ***meaning*** of the token 'variable' is defined by the totality of expressions where it appears. With this premise, we will incorporate elements from category theory [14], to represent PLs as categories. Given the functor $PL('variable', -)$, we can obtain the totality of expressions in the dataset that define the meaning of $'variable'$. We plan to model the semantics of every expression in the datasets used to train Language Models by obtaining functors through the *Yoneda Embedding* [8]. This approach has been exprored by Bradley et al. [7] to model the semantics of Natural Language (English).

2.2 Evaluating Capabilities

In recent years, a growing interest has emerged in not only evaluating LLMs for code but also providing interpretations about how these models arrive at generated predictions [5, 13, 19–21, 24, 25]. Unfortunately, the current evaluation process overly relies on accuracy leaving no consensus as to what other properties or SE settings are impacting the code generation process.

We start by adapting the concept of capability introduced by Ribeiro et al. [23] in our approach. A ***source code capability*** refers to a feature that is inherent in a programming language and represents a concept that can be easily understood by a software engineer. Based on this premise, we define three types of *source code capabilities*: Linguistic, Semantic, and Quality.

Linguistic capabilities are evaluated at a fine-grained level, focusing on the model's ability to understand hidden syntax and grammatical features in the training data. For instance, we seek to understand what types of elements in grammar most influence code predictions.

Capability	Rationale
Very Busy Expressions	An expression E is very busy at Point P, if no matter what path is taken from P, the expression E is evaluated before any of its operands are defined.
Dead Code	A statement $x = value$ can be considered as dead code if the value of x is not used in the future.
Reduntant Expressions	An expression is redundant if the value computed by the expression is available on some\all paths through a program to that expression.
Code Smells	Code Duplicates, Primitive Obsession, Long Parameter List, Message Chain, etc.
Vulnerabilities	Source Code vulnerabilities as defined in Common Weakness Enumeration CWE [3].

Table 1: Examples of Quality Capabilities to be addressed in this dissertation.

Semantic capabilities are related to the domain knowledge encoded in the generated output. We propose to incorporate sophisticated frameworks such as Category Theory [14] and Mechanistic Interpretability [9] to model the semantics of PLs and reverse engineering the internals of a LLM in order to explain and evaluate the efficacy of these models beyond accuracy.

Finally, ***Quality capabilities*** aims to assess the ability of LLMs to produce non-buggy and high-quality optimized code (*e.g.,* Code Smells, Vulnerabilities, and Compiler Optimizations). Table 1 summarizes the most relevant sub-capabilities from this category.

2.3 Building Evaluation Tools

Our final goal is to design, develop, and publish usable tools to incorporate the previous elements for error detection and evaluation of LLMs for code. A key feature of our tools will be their ability to translate complex model decisions into human-understandable knowledge in terms of source code capabilities (*i.e.,* Linguistic, Semantic, and Quality).

3 ANTICIPATED CONTRIBUTIONS

Through conducting the proposed research, we aim to provide a comprehensive framework to help understand why Large Language Models (LLMs) make specific predictions in the context of source code capabilities for software engineering downstream tasks. Our goal is to design a novel evaluation framework based on mathematical theory, incorporating elements from category theory and syntax static analysis. We anticipate that our contribution will empower software practitioners with a deeper understanding of the dynamics of Large Language Models (LLMs), thereby enhancing their ability to ensure and assess reliability in practical scenarios. All datasets, software tools, and overall results from our study will be made publicly available to ensure verifiability.

REFERENCES

[1] [n. d.]. [2009.10297] CodeBLEU: a Method for Automatic Evaluation of Code Synthesis. https://arxiv.org/abs/2009.10297

[2] [n. d.]. Code faster with ai code completions. https://www.tabnine.com/

[3] [n. d.]. CWE - Common Weakness Enumeration. https://cwe.mitre.org/index.html

[4] Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, Jade Goldstein, Alon Lavie, Chin-Yew Lin, and Clare Voss (Eds.). Association for Computational Linguistics, Ann Arbor, Michigan, 65–72. https://aclanthology.org/W05-0909

[5] Jasmijn Bastings and Katja Filippova. 2020. The elephant in the interpretability room: Why use attention as explanation when we have saliency methods? (2020), 149–155. https://doi.org/10.18653/v1/2020.blackboxnlp-1.14 arXiv: 2010.05607.

[6] Tai-Danae Bradley, Juan Luis Gastaldi, and John Terilla. [n. d.]. The structure of meaning in language: parallel narratives in linear algebra and category theory. ([n. d.]).

[7] Tai-Danae Bradley, John Terilla, and Yiannis Vlassopoulos. 2021. An enriched category theory of language: from syntax to semantics. http://arxiv.org/abs/2106.07890 arXiv:2106.07890 [cs, math].

[8] Tai-Danae Bradley, John Terilla, and Yiannis Vlassopoulos. 2022. An Enriched Category Theory of Language: From Syntax to Semantics. *La Matematica* 1, 2 (June 2022), 551–580. https://doi.org/10.1007/s44007-022-00021-2

[9] Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E Burke, Tristan Hume, Shan Carter, Tom Henighan, and Christopher Olah. 2023. Towards Monosemanticity: Decomposing Language Models With Dictionary Learning. *Transformer Circuits Thread* (2023). https://transformer-circuits.pub/2023/monosemantic-features/index.html.

[10] Zimin Chen, Steve James Kommrusch, Michele Tufano, Louis-Noël Pouchet, Denys Poshyvanyk, and Martin Monperrus. 2019. SEQUENCER: Sequence-to-Sequence Learning for End-to-End Program Repair. *IEEE Transactions on Software Engineering* (2019), 1–1. https://doi.org/10.1109/TSE.2019.2940179

[11] Matteo Ciniselli, Nathan Cooper, Luca Pascarella, Denys Poshyvanyk, Massimiliano Di Penta, and Gabriele Bavota. 2021. An Empirical Study on the Usage of BERT Models for Code Completion. *CoRR* abs/2103.07115 (2021). arXiv:2103.07115 https://arxiv.org/abs/2103.07115

[12] GitHub. [n. d.]. GitHub Copilot · Your AI pair programmer. https://copilot.github.com/

[13] Sarthak Jain and Byron C. Wallace. 2019. Attention is not Explanation. https://doi.org/10.48550/arXiv.1902.10186 arXiv:1902.10186 [cs].

[14] Tom Leinster. 2016. Basic Category Theory. https://doi.org/10.48550/arXiv.1612.09375 arXiv:1612.09375 [math].

[15] Yao Li, Tao Zhang, Xiapu Luo, Haipeng Cai, Sen Fang, and Dawei Yuan. 2022. Do Pre-trained Language Models Indeed Understand Software Engineering Tasks? https://doi.org/10.48550/arXiv.2211.10623 arXiv:2211.10623 [cs].

[16] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*. Association for Computational Linguistics, Barcelona, Spain, 74–81. https://aclanthology.org/W04-1013

[17] David Lo. 2023. *Trustworthy and Synergistic Artificial Intelligence for Software Engineering: Vision and Roadmaps*.

[18] Microsoft. [n. d.]. Visual Studio IntelliCode | Visual Studio - Visual Studio. https://visualstudio.microsoft.com/services/intellicode/

[19] Ahmad Haji Mohammadkhani and Hadi Hemmati. [n. d.]. Explainable AI for Pre-Trained Code Models: What Do They Learn? When They Do Not Work? ([n. d.]).

[20] Akash Kumar Mohankumar, Preksha Nema, Sharan Narasimhan, Mitesh M. Khapra, Balaji Vasan Srinivasan, and Balaraman Ravindran. 2020. Towards Transparent and Explainable Attention Models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, 4206–4216. https://doi.org/10.18653/v1/2020.acl-main.387

[21] David N. Palacio, Nathan Cooper, Alvaro Rodriguez, Kevin Moran, and Denys Poshyvanyk. 2023. Toward a Theory of Causation for Interpreting Neural Code Models. https://doi.org/10.48550/arXiv.2302.03788 arXiv:2302.03788 [cs, stat].

[22] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Pierre Isabelle, Eugene Charniak, and Dekang Lin (Eds.). Association for Computational Linguistics, Philadelphia, Pennsylvania, USA, 311–318. https://doi.org/10.3115/1073083.1073135

[23] Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond Accuracy: Behavioral Testing of NLP models with CheckList. https://doi.org/10.48550/arXiv.2005.04118 arXiv:2005.04118 [cs].

[24] Sofia Serrano and Noah A. Smith. 2019. Is Attention Interpretable? https://doi.org/10.48550/arXiv.1906.03731 arXiv:1906.03731 [cs].

[25] Yao Wan, Wei Zhao, Hongyu Zhang, Yulei Sui, Guandong Xu, and Hai Jin. 2022. What Do They Capture? – A Structural Analysis of Pre-Trained Language Models for Source Code. https://doi.org/10.48550/arXiv.2202.06840 arXiv:2202.06840 [cs].

[26] Cody Watson, Michele Tufano, Kevin Moran, Gabriele Bavota, and Denys Poshyvanyk. 2020. On Learning Meaningful Assert Statements for Unit Test Cases. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 1398–1409. https://doi.org/10.1145/3377811.3380429

[27] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent Abilities of Large Language Models. https://doi.org/10.48550/arXiv.2206.07682 arXiv:2206.07682 [cs].

[28] Wojciech Zaremba, Greg Brockman, and OpenAI. 2021. OpenAI Codex. https://openai.com/blog/openai-codex/