Exploring Existing Machine Learning Approaches Discerning the Quality of Source Code Identifiers

Marcel Simader
AI for Software Engineering, Topic 3
k11823075 / SKZ 521
marcel.simader@jku.at

Melissa Frischherz
AI for Software Engineering, Topic 3
k12011649 / SKZ 521
melissa.frischherz@gmail.com

***Abstract*—The quality of source code identifiers has a direct, and measurable impact on program comprehension, indirectly influencing validity, maintainability, and security of software. Using Language Models (LMs) to extract semantic information from an identifier is an important but challenging task, that is fundamental to assessing its quality. This paper proposes a small-scale survey of existing approaches to quantify how "suitable" a given identifier is in some program context. Such a measure facilitates the evaluation of naming conventions in big code bases, suggestions of context-aware variable, method, or type names, or even finding semantic relationships between identifiers across languages and programs. By leveraging promising advancements in Natural Language Processing (NLP) and machine learning, particularly the recent Large Language Models (LLMs), to analyze the information found in identifiers, we could greatly improve the code analyst's toolkit.**

***Index Terms*—Machine Learning, Natural Language Processing, Code design, Maintainability, Software Quality/SQA.**

I. Introduction

Identifiers are the smallest unit of semantic information in program source code, yet they make up the majority of it [1]. This makes them both important to developers, who need them to comprehend software systems, and practically indispensable to nearly all analytical tools [2]. An empirical study published by Butler et al. shows that making poor choices for method, class, and type names correlates with less readable, maintainable, and overall worse quality code [1]. Recently, Machine Learning (ML) and Natural Language Processing (NLP) approaches have lead to the development of useful code analysis tools, such as the "Anti-Pattern" detector by Arnaoudova et al. [3]. At the heart of such modern tools lies the extraction of meaning behind identifiers chosen by imperfect developers, who may introduce questionable naming schemes, spelling errors, or semantically faulty names. The information found in good identifiers, on the other hand, can be used to gather a rich understanding of a program's structure and intended function.

In the following sections, we present a survey of various papers with a focus on identifiers in the context of software engineering and Artificial Intelligence (AI). We pay special attention to papers that attempt to qualitatively assess identifier names, which opens the path to generating context-dependant identifier suggestions, or finding failures to adhere to (implicitly) established naming conventions, for example. With this study, we aim to provide an entry point into this area of research in the hopes of sparking new ideas, highlighting current gaps in knowledge, and inspiring connections to be made across different fields of research. At the end of this paper, the reader should have a baseline understanding of relevant techniques and technologies, from which new research could be conducted.

Section II will give some background on the theoretical and technical frameworks, section III will discuss how we searched for, and selected papers, section IV will present the findings of the survey, and finally sections V and VI will conclude it with a discussion and ideas for future work.

II. Background

A. Natural Language Processing

NLP has been an active area of research since the nineteen nineties, and was proposed as possible solution to machine translation as early as the fifties. The foundation for the grammar model used in most NLP today is the modelling of probabilities of a word s_i occurring given the context of the previous words in some sentence $s_1 s_2 \dots s_i$. In essence, the problem is to find a probability distribution over these sentences, so that $P(s_1 s_2 \dots s_i) = P(s_1)\, P(s_2 \mid s_1) \cdots P(s_i \mid s_1 \dots s_{i-1})$ aligns with the behavior we want from the system, e.g. summarizing a text. Historically, resources were limited, and even in ignoring the exponential nature of the problem, computing a history of hundreds of conditional probabilities was simply not feasible. This is why the n-gram model was developed, which "chops off the tail end" of the context, thereby reducing it to $n-1$ conditional probabilities [4].

As Allamanis et al. [5] note, an issue with the n-gram model for modelling identifiers is that they often contain neologisms. That is, words that do not appear anywhere in the training set, also called the "vocabulary" of the model. This is self-evident when looking at an example of a getter-method name `getState`, which could also be written as `state`, `retrieveState`, `getsState`, etc. An identifier model needs to be capable of handling *out-of-vocabulary words*: This is known as the Out of Vocabulary (OOV) problem [6].

B. Machine Learning

Classical statistical techniques paved the way for neural networks in NLP in the 21st century. The basis of a Neural

Network (NN) is the neuron, which is found in layers, and sums up inputs and applies some activation function to get an output. The intent with neurons is to model human brains to the best of our current knowledge. Deep learning is the method used to train Deep Neural Networks (DNNs), which typically contain more than four internal layers of neurons, also called "hidden layers" [7]. Naturally, recognition of language is well-suited for DNNs, as they can be trained to provide the probabilities discussed in subsection II-A.

DNNs in their basic form are still incapable of handling arbitrary context lengths, not unlike the n-gram. A type of DNN, the Recurrent Neural Network (RNN) was developed to handle sequential data of arbitrary length. It achieves this by feeding its own hidden layer output back into itself, mimicking a basic form of memory [2]. One variant of the RNN, the Long Short-Term Memory (LSTM) model, takes advantage of restrictions on the information flow through itself to improve retention [8].

There are many more variants of these models, arranged in different architectures. The encoder-decoder RNN, and the transformer are two such examples. The transformer appears to be the best neural model for language at the moment, and is also represented in our study. Instead of using recurrence, it relies on the attention mechanism: This allows the model to weigh the importance of different parts of a sentence in relation to the currently computed probabilities. The breakthrough achieved by Vaswani et al. [9] was to utilize self-attention, a mechanism by which the model can also weigh the importance of weights within itself. In praxis, this means that the model learns better approximations of how to *derive* the semantics of a sentence, not just which features to focus on.

III. Methodology

We started by establishing an initial overview of the topic landscape, which led us to a more systematic search of papers pertaining to the ideas discussed in Section I. This was done by crafting search strings with relevant keywords for IEEEXplore[1] and Google Scholar[2]. The search queries we used for both services are as follows:

IEEEXplore: "(AI OR LLM OR Language Model OR NLP) AND Identifiers AND (Completion OR Analysis OR Suggestion)"

Google Scholar: "Large Language Models for measuring quality of identifiers in source code"

Of the 190 papers the IEEEXplore query yielded, we hand selected 30 to include in broader analysis. For the Google Scholar search query, we hand selected 4 papers directly. In total, we used 34 studies to establish the landscape, and looked further into 7 of them. Any other papers cited in the work were used for supplemental information.

IV. Survey

Research into the significance of identifiers in software engineering has been ongoing since at least 1999, which is the oldest paper in our survey [10], but has only recently gained in popularity. A histogram of the years of publication for our survey papers can be seen in figure 1. We suspect this is due to the major improvements in NLP around the time Google released its paper on the transformer in 2017 [9], Peters et al. presented ELMo in the following year [11], and finally, BERT was proposed by Devlin et al. in 2019 [12]. This aligns with the big spike in publications that same year.

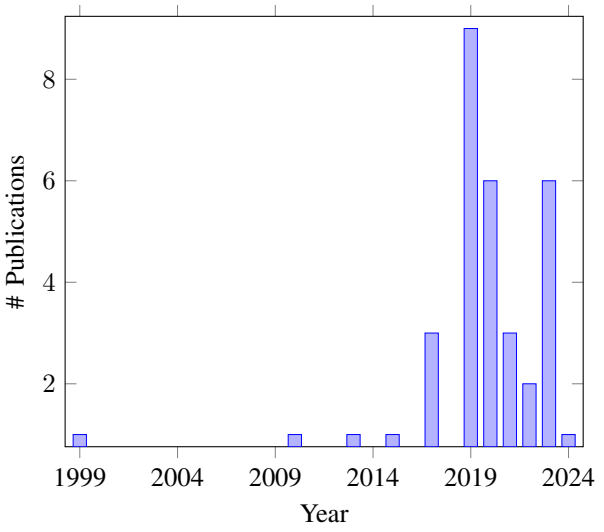

Fig. 1. Histogram of Survey Paper Publications

We have divided the survey into three subsections, one on "traditional" algorithmic approaches, one on approaches using RNNs and LSTMs, and one on very recent advancements like transformers, and Large Language Models (LLMs).

A. Algorithmic Approaches

The simplest approach to quantifying the correct naming of identifiers in our survey is found in the paper by Charitsis et al. [13], which focuses on the skills of computer science students. A reduced syntax backed by the Java programming language named Karel is used along with student submissions to create a simple, manually-annotated dataset of function name quality. Students submitted their source code solutions for a problem statement in Karel. The submitted code was compiled in-memory and instrumented, wherein the program state was compared before and after every function execution to produce a pairwise similarity score of each student function. This similarity score, along with the manually annotated quality is used to train a discriminator model producing scores on unseen identifiers. Charitsis et al. report classifier accuracies between 89.36% and 76.16%, depending on the problem hardness.

We also examined a paper which aims to tackle the OOV problem (see subsection II-A) by means of identifier splitting [6]. Shi et al. show that by modifying the Byte Pair Encoding (BPE) algorithm, which was originally developed for compression but later adopted by Karampatsis et al. [14], the performance of language models can be improved slightly.

[1]https://ieeexplore.ieee.org
[2]https://scholar.google.com

In their study, they argue that the semantics of identifiers, e.g. `getHeader`, is lost in the BPE scheme. A better approach would be to split the identifier by certain heuristics – the Ronin approach in this case – and turn the previous example into `get` and `Header`. Splitting identifiers with high accuracy is crucial in this domain, which makes solutions to this problem particularly sought after: Allamanis et al. present another approach in which they split a token into subtokens [5].

The final publication foregoing neural networks in our survey is the extensive article on identifier renaming and prediction suggestion by Zhang et al. [15]. This is the only study we analyzed which uses a decision-tree based model. More accurately, it uses several decision trees in an arrangement called a "Random Forest", from which an aggregate result is drawn. A particularly interesting facet of this work is an emphasis on the fact that similar identifiers usually change in similar ways, which lead the authors to utilize file histories to aid in improving accuracy.

B. DNNs and RNNs

DNNs seem to be by far the most common approach in modern papers on identifier quality assessment. Allamanis et al. present a variation on the Logbilinear (LBL) Language Model (LM), the logbilinear context model [5]. This neural network extends the long-distance context of the LBL LM to encompass what is called the local and global context. This allows the model to more efficiently learn from an identifier's surroundings at *every occurrence* of said identifier, along with a feature vector of general properties (e.g. "constant value", "integer type"). This, in combination with the exploitation of the semantic substructures of identifiers mentioned in the previous subsection, allows the model to far surpass the predictive qualities of the n-gram. Allamanis et al. state that they introduce a "[...] model that is to [their] [...] knowledge, the first that can propose neologisms [...]" [5].

Due to the sequential nature of text, the identifier naming problem is also particularly well-suited for RNNs and LSTMs, although we do not include a study on the latter. The paper by Gao et al. focuses on an encoder-decoder RNN model with the promising addition of the attention and copy mechanisms [2]. Instead of reading the surrounding source code, documentation comments are used to train the model in this study. The addition of attention makes it possible for the model to learn which words are most important for each subtoken of the result. Neologisms are handled by copying, which makes it possible for the model to select subtokens from the data directly, without ever having learned them. Gao et al. go on to show that an RNN with both attention and copying surpasses simpler RNNs and tf-idf, a widespread method of information retrieval, in every metric that was measured.

C. Transformers, LLMs and Beyond

The most modern, and in turn, most sophisticated and performant LMs utilize transformer models. Villmow et al. propose such a transformer to discriminate between identifiers that adhere to, or conversely, break naming conventions [16]. Both a generative and a discriminative model was trained: The generative model tries to fill in gaps in presented source code, while the discriminative model uses a "weak AI" to generate naming suggestions which are then classified. The dataset used to train their LLM was manually annotated and contains over 6000 data points, which are freely available for future research. The model that performed the best, dubbed CODEDOCTOR by the authors, was the generative model with 247 million parameters. For comparison, the GRAPHCODEBERT model [17] based on BERT which was mentioned in section IV, ranked third and contains 125 million parameters.

Another model based on BERT, and by far the most complex in terms of scope and execution of the papers presented thus far, is the one presented by Ju et al. [18]. This study delves into the scarcely researched topic of tracking identifiers across programming languages and projects – in fact, this is the only paper in our survey to attempt this feat. They achieved cross-language identifier binding by string-matching, and then passing the potential match through stacked discriminators along with a pre-trained BERT model to apply contextual and framework knowledge. Interestingly, this was also the only paper to compare their results with the extremely popular IntelliJ IDEA: Whereas the IDE could correctly classify 564 unique and 1203 duplicate identifier pairs, the novel model could respectively detect 3361 and 8454 pairs.

V. DISCUSSION AND FUTURE WORK

Our survey shows that a lot of ongoing research in this field looks promising, and could pose massive benefits for quality, maintainability, and validity of programs. Identifier quality assessment seems to be a rather niche, but surprisingly deep topic. It is not hard to imagine the potential for continuous integration pipelines to flag potentially inappropriate identifier names, or for IDEs to support fully cross-language refactoring tools. Such applications are still scarce, but do exist, as seen in the rename analysis tool by Peruma et al., which uses source code and commit messages in an attempt to explain the decision making process behind name changes [19].

One subfield with increasing importance appears to be the splitting of identifiers into subtokens that capture the meaning *intended* by the developer with accuracy. Besides the papers we have already presented [2], [5], [6] there are clearly still improvements to be made, e.g. Liu et al. [20], making this a potentially fruitful future research topic.

We acknowledge the potential of having missed some publications, as this survey was far from extensive or rigorous. Furthermore, there is a possibility that literature on tools already utilizing this research or on dedicated hobby projects has simply not been published or reviewed through reputable outlets. We propose that further research is needed to get a full grasp on how extensive tooling for this application of LMs is at this point in time.

Perhaps this is the reason we were not able to find any publications involving the recent Generative Pre-Trained Transformers (GPTs) with billions of parameters developed mostly by OpenAI. There seems to be a big gap in understanding

between the "small LLMs" and the incomprehensibly complex models currently being trained. The efficacy of OpenAI's ChatGPT[3] has recently come under more scrutiny, and our conjecture is that targeted approaches will remain more effective for this use case [21].

VI. Conclusion

This small-scale survey was compiled by querying two publishers, for which 34 results were found to be relevant. We analyzed 7 of these papers in more detail, and presented a brief overview of the research landscape as it currently stands. We found that the majority of proposed solutions to identifier quality assessment use DNNs like RNNs or transformers, but that simpler solutions like decision trees are still adequate in some circumstances. Table I shows a breakdown of these categories. In our conclusion, we discussed potential applications for this research and future work.

TABLE I
Overview of ML Techniques in our Survey

Technique Used	#
NN	2
DNN	2
RNN	4
Transformer	8
Other	9
N.A.	9

References

[1] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp, "Exploring the influence of identifier names on code quality: An empirical study," in *2010 14th European Conference on Software Maintenance and Reengineering*, 2010, pp. 156–165. [Online]. Available: https://ieeexplore.ieee.org/document/5714430

[2] S. Gao, C. Chen, Z. Xing, Y. Ma, W. Song, and S.-W. Lin, "A neural model for method name generation from functional description," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Feb 2019, pp. 414–421. [Online]. Available: https://ieeexplore.ieee.org/document/8667994

[3] V. Arnaoudova, M. Di Penta, G. Antoniol, and Y.-G. Guéhéneuc, "A new family of software anti-patterns: Linguistic anti-patterns," 03 2013.

[4] P. F. Brown, J. Cocke, S. A. Della Pietra, V. J. Della Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer, and P. S. Roossin, "A statistical approach to machine translation," *Computational Linguistics*, vol. 16, no. 2, pp. 79–85, 1990. [Online]. Available: https://aclanthology.org/J90-2002

[5] M. Allamanis, E. T. Barr, C. Bird, and C. Sutton, "Suggesting accurate method and class names," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 38–49. [Online]. Available: https://doi.org/10.1145/2786805.2786849

[6] J. Shi, Z. Yang, J. He, B. Xu, and D. Lo, "Can identifier splitting improve open-vocabulary language model of code?" in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2022, pp. 1134–1138.

[7] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[8] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6000–6010.

[10] G. Antoniol, G. Canfora, A. Lucia, and E. Merlo, "Recovering code to documentation links in oo system," *Reverse Engineering, Working Conference on*, vol. 0, p. 136, 11 1999.

[11] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," pp. 2227–2237, jun 2018. [Online]. Available: https://aclanthology.org/N18-1202

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Minneapolis, Minnesota: Association for Computational Linguistics, jun 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[13] C. Charitsis, C. Piech, and J. Mitchell, "Assessing function names and quantifying the relationship between identifiers and their functionality to improve them," in *Proceedings of the Eighth ACM Conference on Learning @ Scale*, ser. L@S '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 291–294. [Online]. Available: https://doi.org/10.1145/3430895.3460161

[14] R.-M. Karampatsis, H. Babii, R. Robbes, C. Sutton, and A. Janes, "Big code != big vocabulary: open-vocabulary models for source code," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1073–1085. [Online]. Available: https://doi.org/10.1145/3377811.3380342

[15] J. Zhang, J. Luo, J. Liang, L. Gong, and Z. Huang, "An accurate identifier renaming prediction and suggestion approach," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 6, sep 2023. [Online]. Available: https://doi.org/10.1145/3603109

[16] J. Villmow, V. Campos, J. Petry, A. Abbad-Andaloussi, A. Ulges, and B. Weber, "How well can masked language models spot identifiers that violate naming guidelines?" in *2023 IEEE 23rd International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2023, pp. 131–142. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10356698

[17] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, J. Yin, D. Jiang, and M. Zhou, "Graphcodebert: Pre-training code representations with data flow," *ArXiv*, vol. abs/2009.08366, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:221761146

[18] Y. Ju, Y. Tang, J. Lan, X. Mi, and J. Zhang, "A cross-language name binding recognition and discrimination approach for identifiers," in *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2023, pp. 948–955. [Online]. Available: https://ieeexplore.ieee.org/document/10123604

[19] A. Peruma, M. W. Mkaouer, M. J. Decker, and C. D. Newman, "Contextualizing rename decisions using refactorings and commit messages," in *2019 19th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2019, pp. 74–85.

[20] S. Liu, J. Zhang, J. Liang, J. Luo, Y. Xu, and C. Sun, "Chis: A novel hybrid granularity identifier splitting approach," in *2021 28th Asia-Pacific Software Engineering Conference (APSEC)*, 2021, pp. 192–201.

[21] T. Wu, S. He, J. Liu, S. Sun, K. Liu, Q.-L. Han, and Y. Tang, "A brief overview of chatgpt: The history, status quo and potential future development," *IEEE/CAA Journal of Automatica Sinica*, vol. 10, no. 5, pp. 1122–1136, 2023.

[3] https://chat.openai.com