UNIVERSITY OF SALERNO

DEPARTMENT OF COMPUTER ENGINEERING, ELECTRICAL ENGINEERING AND APPLIED MATHEMATICS

Master's Degree Course in Computer Engineering

Natural Language Processing and
Large Language Models

Automatic Detection of LLM-Generated Source Code

Supervisor:
Ch.mo Prof.
Nicola Capuano

Candidate::
Nunzio Del Gaudio
Mat. 0622702277

ACADEMIC YEAR 2024/2025

Contents

Chapter 1

Introduction

Generative Artificial Intelligence (AI) is a rapidly advancing branch of AI that, in recent years, has made tremendous progress, largely due to the widespread adoption of deep learning techniques. Generative models are now powerful deep learning architectures capable of producing highly realistic text, images, and even video content.

The famous Turing Test is frequently referenced in this context. In this test, a human judge had to determine whether they were communicating with another human or with a machine. This milestone was already surpassed in 2014, with early systems such as Eugene Goostman [1].

Today, it is widely accepted that much of the content generated by generative models is indistinguishable from human-produced material and indeed, for humans, **this distinction is increasingly difficult to make**.

The introduction of the Transformer architecture in 2017 [2] revolutionized the field and led to the development of Large Language Models (LLMs), which quickly became part of everyday life. This wave began with the public availability of GPT-3 in June 2020 [3], followed by a growing ecosystem of LLMs including GPT-4 [4], PaLM [5], LLaMA [6], Claude [7], and Mistral [8].

Generative AI, however, is not limited to natural language processing. It has also enabled powerful models for image generation, such as Stable Diffusion [9], and more recently for video synthesis, with systems like Veo-3 [10].

Although the usefulness and extraordinary capabilities of these tools are undeniable, there are many scenarios in which it becomes necessary to have user-friendly tools to detect whether text, images, or videos were generated by an AI.

This thesis presents a more modest, yet equally important objective: **detecting artificially generated source code** produced by code generation models such as OpenAI Codex [11], GPT-4 [4], and Code LLaMA [12].

While natural language detectors are well-established and widely studied in scientific literature, the detection of AI-generated code remains a far less consolidated and more recent research area. The contribution of this work is to **analyze the most relevant scientific advancements** made in the past year, which are often fragmented and inconsistent due to the unique challenges involved in distinguishing machine-generated code, a task that is arguably more complex than natural text detection.

1.1 Historical Overview of Code Generation by LLMs

The history of code generation can be explored from several perspectives.

1.1.1 Natural Language Processing

One possible starting point possible perspective for analysing the evolution of code generation is to trace the development of Natural Language Processing (NLP), the field that studies how machines process and interact with human language. NLP comprises two major subdomains: Natural Language Understanding (NLU), which focuses on a machine's ability to interpret and "understand" human language, and Natural Language Generation (NLG), which concerns the generation of natural-sounding text.

This distinction is particularly relevant because the technologies currently used for code generation are essentially the same as those employed for natural language generation. In fact, Transformer-based models trained on source code data approach code generation in the same way they would handle natural text generation by predicting sequences of tokens in context using learned statistical patterns [2].

From this point of view we can start in the 1943 during World War II with the invention of Colossus, one of the first digital electronic computers, developed in order to analyse encrypted communications from the German military.

In parallel, the discipline of Natural Language Processing (NLP) began to take shape as early as the 1940s, culminating in the 1954 Georgetown experiment the first public demonstration of machine translation. Another milestone came in 1966 with the creation of ELIZA [13], considered the first chatbot in history, which simulated the behaviour of a psychotherapist using pattern-matching rules.

During the 1970s and 1980s, symbolic approaches to NLP became popular. These early attempts aimed to enable machines to "understand" language through manually encoded rules and logic-based systems. In the 1990s, the importance of statistical methods became evident, marking a shift from rule-based to probabilistic models for language processing.

In 2006 Google launched its now ubiquitous Google Translate service. The following decade saw the rise of voice-based assistants: Apple's Siri (2011), Microsoft's Cortana (2014), Amazon's Alexa (2014), and Google Assistant (2016).

A major breakthrough came with the introduction of distributed word representations, especially word2vec [14] and GloVe [15], published in 2013 and 2014 respectively. These methods enabled dense vector representations that captured semantic relationships between words in large corpora.

The most significant leap, however, occurred in 2017 with the publication of the now seminal paper "Attention is All You Need" [2], which introduced the Transformer architecture. This architecture remains the dominant framework in NLP and underpins nearly all modern LLMs. Transformers enabled major advances in tasks such as text generation, machine translation, question answering, text summarization and, more recently, code generation.

The first LLMs capable of code generation began to appear around 2019–2020, notably with the release of GPT-2 [16]. In 2021, OpenAI released Codex [11], a GPT-3 derivative trained specifically for code generation and explanation, which was later integrated into GitHub Copilot.

Between 2022 and 2024, a wave of new LLMs for code generation was released, including CodeT5 [17], CodeGen [18], CodeGeeX [19], and Code LLaMA [12].

1.1.2 Code Generator

Another possible point of view is the one focused on code generation itself. This is not a recent development at all: it dates back to 1957 with FORTRAN. FORTRAN is both a programming language and a compiler, developed by IBM. A compiler can be seen as a code generation tool, since it translates source code into machine code — the only ”language” truly understandable by a computer [20].

Compilers have a long history and have been continuously improved over time, but they are not the only tools for code generation. Already in 1976, the concept of intelligent editors emerged with Emacs, thanks to its support for custom macros [21]. Later, in 1996, Visual Basic 5 introduced symbolic completion, namely the ability of the IDE to suggest code based on the context and the symbols already present.

In 1999, with XSLT, one of the first standardized tools for automatic transformation between markup languages was introduced [22]. During the same years, the work of Zelle and Mooney (1996–1999) proposed methods based on Natural Language processing to generate database queries from natural language expressions [23, 24].

At the same time, template-engine-based tools spread, such as Jinja2 (2005) and Mako (2006), which allowed the generation of dynamic code by combining data with predefined structures [25, 26].

In 2017, with research on AST-guided code generation, LSTM models began to be used to generate code in a more structured way, guided by the syntax of the programming language [27].

Finally, in 2021, with GitHub Copilot, one of the most advanced code completion tools was introduced: so efficient that it is capable of generating entire code sections from simple textual prompts, thanks to the use of generative AI models based on Transformer architectures [11].

1.1.3 LLM code generator

1.2 Motivations Behind LLM-Generated Code Detection

1.3 Challenges in LLM-Generated Code Detection

1.4 Thesis Objectives

Chapter 2

Overview of current state of the art

Study of the state of the art on detection systems for LLM-generated text, with particular reference to early work focused on source code, for example,

Chapter 3

Dataset Analysis

Let us recall that the ultimate goal is to build a binary classifier capable of determining whether a piece of source code was written by a human or generated by a Large Language Model (LLM). For binary classifiers, **accuracy** is typically the default evaluation metric, unless one classification outcome, such as true positives or false positives, is considered more critical than the other. Given that this classifier can be used in scenarios where a programmer could be 'accused' of not having authored the code themselves, it is essential that such claims be made with a high degree of confidence. While it is certainly desirable to allow end users to configure the decision threshold, the default setting should prioritize a **low FPR** (false positive rate) to prevent unjust accusations.

3.1 Dataset Evaluation Criteria

3.2 Datasets Proposed in the Scientific Literature

3.3 Final Dataset

Chapter 4

Detection Methods

4.1 Evaluation Metrics

4.2 Evaluation of Existing Methods

4.3 Proposed Improvements

Chapter 5

User Interface

5.1 Requirements Analysis

5.2 Design Choices

5.3 Implementation and Features

Chapter 6

Conclusion

6.1 Final Evaluation and Perceived Quality

6.2 Future Work

Bibliography

[1] Kevin Warwick and Huma Shah. “Can Machines Think? A Report on Turing Test Experiments at the Royal Society”. In: *Journal of Experimental & Theoretical Artificial Intelligence* 28.6 (2016), pp. 989–1007.

[2] Ashish Vaswani and et al. “Attention is all you need”. In: *Advances in Neural Information Processing Systems* 30 (2017).

[3] Tom Brown and et al. “Language Models are Few-Shot Learners”. In: *NeurIPS* 33 (2020), pp. 1877–1901.

[4] OpenAI. *GPT-4 Technical Report*. https://openai.com/research/gpt-4. 2023.

[5] Aakanksha Chowdhery et al. “PaLM: Scaling Language Modeling with Pathways”. In: *arXiv preprint arXiv:2204.02311* (2022).

[6] Hugo Touvron et al. “LLaMA: Open and Efficient Foundation Language Models”. In: *arXiv preprint arXiv:2302.13971* (2023).

[7] Anthropic. *Claude: The Anthropic Language Model*. https://www.anthropic.com/index/introducing-claude. 2023.

[8] Jianfeng Jiang and et al. “Introducing Mistral: A High-Performance Language Model”. In: *Mistral AI* (2023).

[9] Robin Rombach and et al. “High-Resolution Image Synthesis with Latent Diffusion Models”. In: *CVPR* (2022).

[10] Google DeepMind. *Veo: Advancing Generative Video Models*. https://deepmind.google/technologies/veo. 2024.

[11] Mark Chen and et al. “Evaluating Large Language Models Trained on Code”. In: *arXiv preprint arXiv:2107.03374*. 2021.

[12] Baptiste Roziere and et al. “Code LLaMA: Open Foundation Models for Code”. In: *arXiv preprint arXiv:2308.12950* (2023).

[13] Joseph Weizenbaum. “ELIZA—a computer program for the study of natural language communication between man and machine”. In: *Communications of the ACM* 9.1 (1966), pp. 36–45.

[14] Tomas Mikolov and et al. “Efficient Estimation of Word Representations in Vector Space”. In: *arXiv preprint arXiv:1301.3781* (2013).

[15] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. “GloVe: Global Vectors for Word Representation”. In: *EMNLP*. 2014.

[16] Alec Radford and et al. “Language Models are Unsupervised Multitask Learners”. In: *OpenAI Blog* 1.8 (2019).

[17] Yue Wang and et al. “CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation”. In: *arXiv preprint arXiv:2109.00859* (2021).

[18] Erik Nijkamp and et al. “CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis”. In: *arXiv preprint arXiv:2203.13474* (2022).

[19] Guolin Zeng and et al. “CodeGeeX: A Pre-trained Model for Code Generation with Cross-Programming-Language Evaluation”. In: *arXiv preprint arXiv:2203.13474* (2022).

[20] John Backus and et al. “The FORTRAN automatic coding system”. In: *Proceedings of the Western Joint Computer Conference* (1957), pp. 188–198.

[21] Richard M. Stallman. *EMACS: The extensible, customizable self-documenting display editor*. https://www.gnu.org/software/emacs/. 1981.

[22] W3C. *XSL Transformations (XSLT) Version 1.0*. https://www.w3.org/TR/xslt. 1999.

[23] John M. Zelle and Raymond J. Mooney. “Learning to parse database queries using inductive logic programming”. In: *AAAI/IAAI, Vol. 2*. 1996, pp. 1050–1055.

[24] Raymond J. Mooney. “Learning semantic parsers: An important but understudied application of machine learning”. In: *Proceedings of the AAAI Spring Symposium on Natural Language Processing for the World Wide Web*. 1997.

[25] Pallets Projects. *Jinja2 Documentation*. https://jinja.palletsprojects.com/. 2005.

[26] Mike Bayer. *Mako Templates for Python*. https://www.makotemplates.org/. 2006.

[27] Pengcheng Yin and Graham Neubig. “A syntactic neural model for general-purpose code generation”. In: *ACL*. 2017.