DuoLens: A Framework for Robust Detection of Machine-Generated Multilingual Text and Code

Shriyansh Agrawal*
shriyanshag0411@gmail.com

Aidan Lau*
aidan3616@outlook.com

Sanyam Shah
sanyam21in@gmail.com

Sunishchal Dev
dev@algoverseairesearch.org

Vasu Sharma
sharma.vasu55@gmail.com

Kevin Zhu
kevin@algoverseacademy.com

Ahan M R†
ahanmr@microsoft.com

Abstract

The prevalence of Large Language Models (LLMs) for generating multilingual text and source code has only increased the imperative for machine-generated content detectors to be accurate and efficient across domains. Current detectors either incur high computational cost or lack sufficient accuracy, often with a trade-off between the two, leaving room for further improvement. To address these gaps, we propose the fine-tuning of encoder-only Small Language Models (SLMs), in particular, the pre-trained models of RoBERTA & CodeBERTa using specialized datasets on source code and other natural language to prove that for the task of binary classification, SLMs outperform LLMs by a huge margin whilst using a fraction of compute. Our encoders achieve AUROC = 0.97 to 0.99 and macro-F1 = 0.89 to 0.94 while reducing latency by 8-12$\times$ and peak VRAM by 3–5$\times$ at 512-token inputs. Under cross-generator shifts and adversarial transformations (paraphrase, back-translation; code formatting/renaming), performance retains $\geq 92\%$ of clean AUROC. We release training and evaluation scripts with seeds and configs; a reproducibility checklist is also included.

1 Introduction

As generative AI models continue to advance, their outputs permeate many domains, from written prose to computer code. AI-assisted content creation can boost productivity—for instance, code generation with LLMs has been reported to improve developer efficiency by up to 33% [3]. However, the rise of machine-generated text and code also brings critical concerns in academia and industry, such as plagiarism, misinformation, and unfair advantages in assessments or job interviews questioned by Benke and Szőke [4]. Furthermore, this issue extends toward source code, exacerbated by the wide availability of specialized LLMs (Qwen Coder, Anthropic Claude 3.7/4 Sonnet) and coding 'Agents' which are capable in acting autonomously in a workflow. They exhibit a track record for inefficient code generation, poor security practices, partial execution and the long running issue of LLM hallucination from Ambati et al. [2]. This has sparked a growing need for reliable and accessible detection of machine-generated content to maintain integrity and trust. Current detectors often exhibit one notable weaknesses: many are primarily effective only on English text and struggle

*Equal contribution
†Project lead

39th Conference on Neural Information Processing Systems (NeurIPS 2025): 4th Workshop on Deep Learning for Code.

with other languages, leading to false positives on non-English inputs. Additionally, individual detection approaches (e.g., based on perplexity or stylistic cues alone) can be evaded, and ensemble agent-based methods can be prohibitively slow or resource-intensive, such as Li et al. [15]'s method. Similar challenges apply to source code detection—detectors must distinguish machine-generated code from human-written code across different programming languages, but research in this area is quite narrow but readily emerging [23].

In this work, we propose the use of SLMs rather than LLMs for binary classification of code into machine-generated or human-written labels. Our rationale for the use of SLMs is since that: 1) LLMs may have a propensity or underlying bias to viewing AI generated text as being of higher quality; 2) LLMs are prone to hallucinate and, when fine-tuned, may experience catastrophic failure.

Contributions:

- **Datasets for Evaluating Detection in Source Code and Multilingual Text:** We present a large-scale dataset that aligns well with the optimal input range of SLMs (with chunking employed when necessary). The datasets encompass samples of source code spanning multiple programming languages, as well as multilingual text from diverse domains, all annotated with binary classification labels.
- **Comprehensive Evaluation:** We conduct benchmarking experiments on multilingual text, as well as source code in Python, Java, JavaScript, C, C++, C#, and Go. Across these evaluations, DuoLens consistently outperforms baseline detectors in terms of accuracy and macro-F1, including under challenging scenarios such as cross-model and cross-language settings. These findings support our hypothesis that SLMs can surpass LLMs in the binary classification of human-written versus machine-generated code and multilingual text.

2 Related Work

Pre-existing Datasets. Although the field remains relatively nascent, a number of datasets have been developed to distinguish between human-written and machine-generated code. **AIGCodeSet** [8] consists exclusively of Python samples, exemplifying a wider trend also observed in **CodeMirage** [1] and **DroidCollection** [20], where Python constitutes the majority of instances while other languages (C, C++, C#, Java, JavaScript, Go) have a sparse representation. These resources also incorporate task-specific samples and enhance stylistic diversity by including outputs from multiple LLMs. In contrast, resources for multilingual text detection exhibit greater variation. **Multitude** [16] compiles news articles but lacks syntactic alignment, while **Multisocial** [17] captures authentic social media content yet is limited to informal registers. **M4** [26] offers a substantially larger corpus, albeit predominantly formal in style. Finally, **HC3** [11] provides multilingual QA pairs but remains constrained in its linguistic coverage.

RoBERTa as a detector. Early detector efforts fine-tuned RoBERTa on model-specific outputs (e.g., OpenAI's RoBERTa-based GPT-2 detector from Solaiman et al. [22]; community releases such as "roberta-large-openai-detector"), demonstrating high accuracy on the generator(s) seen during training but poor generalization to unseen or larger models. For example, Chen et al. [6]'s GPT-Sentinel, had similar results where it was unable to generalise to other LLMs. However, recent research has moved away from enhancing or enriching RoBERTa and BERT models in favor of using LLMs for text detection, which includes methods such as Binoculars [12], AGENT-X [15], Ghostbuster [25], EAGLE [5], and RAIDAR [18]. However, we advocate for the use of BERT-based models, particularly for binary classification tasks, where LLMs demand substantially greater computational resources while yielding inferior or, at best, comparable performance.

3 Methodology

Multilingual Dataset. Many prior multilingual datasets are heavily topic-focused, for example, concentrating on singular domains such as social media or news, which limits the models ability to generalize across different styles of writing. Additionally, they also contain heavy language imbalances (most towards English), which leads to worse performance on other languages.
To address these issues, our dataset is comprised of 54,520 text samples across eight languages: English, Russian, Arabic, Dutch, German, Spanish, Portuguese, and Romanian, achieved through extending previous datasets, including **MULTITuDE** [16] (news focused), **MultiSocial** [17] (social

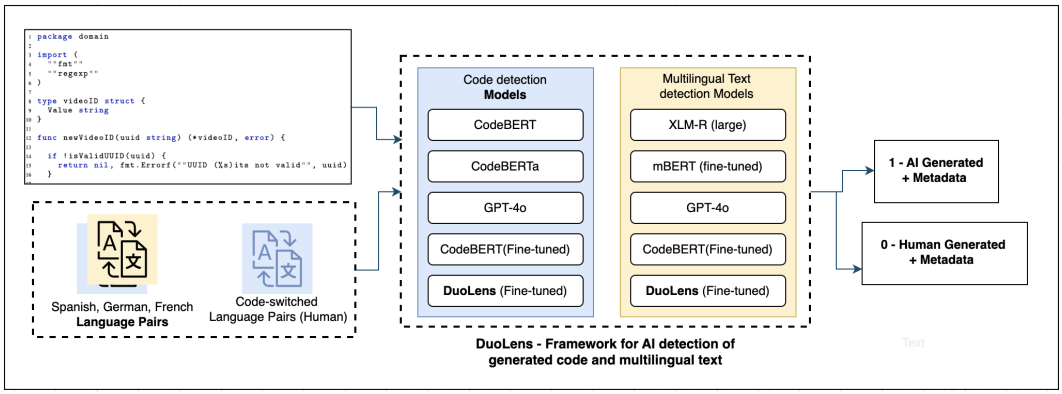


Figure 1: Overview of the experimental setup where each sample from the dataset is processed by the models and classified as either 0 (human-written) or 1 (machine-generated)

media focused), **M4** [26] (general but slightly outdated), and **HC3** (question-answer pair focused). Additional details about the dataset are available in Appendix B.2. Our dataset is more balanced in terms of style and domain, making it better suited for fine-tuning. Similarly to the code dataset, it has an equal balance of human-written and machine-generated samples (27,260 samples for each) to prevent any bias from arising. However, the languages are not balanced in this dataset since the datasets used to create ours all had different languages with varying amounts of samples in shared languages between them, leading to language imbalances in our dataset.

Code Dataset. Existing datasets for code generation evaluation exhibit two notable limitations: **1)** they are often narrow in scope, with samples predominantly drawn from sources such as Puri et al. [21]'s IBM CodeNet or competitive programming platforms thereby offering limited representation of production-level code; **2)** they tend to be heavily imbalanced across languages, with a disproportionate emphasis on Python, which hinders the ability to generalize across syntactically complex languages such as the C family of languages as well as Java, and JavaScript. Our dataset was constructed by curating samples from previous datasets, including **AIGCD** [8], **DroidCollection** [20], and **CodeMirage** [1]. To solve the aforementioned problems, our dataset contains code samples in 7 programming languages: Python, Java, JavaScript, C, C#, C++, and Go. Each language has an identical number of human-written and machine-generated samples (6000 samples for each class in its respective language), to ensure balance not only between the classes but also between the languages, in turn solving both the problems that hindered previous datasets. Shown in Figure 3a and explained in Appendix B.1.

Dataset Creation Both datasets were constructed by first enumerating the available samples for each label (machine-generated and human-written) within each language, in order to determine the maximum number of samples that could be retained per language while preserving label balance. This procedure ensured that no label or language was overrepresented, thereby mitigating potential sources of bias that could adversely affect fine-tuning. Once these maxima were established, the final samples were drawn from the aggregated pool of all available datasets for the particular dataset (multilingual or source code).

Architecture. For the task of detecting machine-generated code, we employed **CodeBERT** by Feng et al. [10] and **CodeBERTa** as baseline models. CodeBERT is trained on natural language and source code, thereby possessing capabilities in both domains. In contrast, CodeBERTa is half the size of CodeBERT and is exclusively trained on source code (from CodeSearchNet Husain et al. [14]) without official support for NLP tasks. Both models were fine-tuned through a classification head attached to each model at initialization that was subsequently trained and fine-tuned, using samples from all the languages in the dataset, to distinguish between machine-generated and human-written code. We propose **DuoLens**, a dual-encoder detector that fuses complementary representations from **CodeBERT** and **CodeBERTa** to identify AI-generated content in both natural language and source code. Given an input sequence x, we obtain hidden states from each encoder and derive pooled vectors via **[CLS]** or mean pooling; a lightweight fusion head then combines these signals using a learned gate that down-weights redundant features and emphasizes encoder-specific cues (e.g., lexical/semantic alignment from CodeBERT's NL–code pretraining versus syntactic/structural

regularities from CodeBERTa's code-centric pretraining). The fused representation feeds a single linear classifier trained with class-balanced binary cross-entropy.

Multilingual encoders. For multilingual text classification, we fine-tune three widely used pre-trained encoders: **XLM-RoBERTa-base** and **XLM-RoBERTa-large** [7], and **Multilingual BERT** (mBERT) [9]. The two XLM-RoBERTa variants are trained on the same CommonCrawl-derived multilingual corpus and differ only in model capacity, whereas mBERT is trained primarily on multilingual Wikipedia. All models are fine-tuned using the protocol described above with identical tokenization, maximum sequence length, optimizer, and learning-rate schedule; checkpoints are selected on the development set, and probabilities are calibrated with temperature scaling.

4 Experiments

4.1 Experimental Setup

Baselines. We evaluated the base models of **CodeBERT** and **CodeBERTa** using a probing approach described in Tenney et al. [24]'s work. An analogous procedure was employed for the non-fine-tuned multilingual models. We also employed a chunking strategy for the classification of source code, as many samples exceeded the maximum input length of 512 tokens for BERT-based models. In addition, we incorporated **GPT 4o** (OpenAI et al. [19]) as a baseline for both domains and **Qwen2.5 Coder 3B** for source code (Hui et al. [13]), which was assessed through few-shot prompting with six examples (3 pairs of machine-generated and human-written samples in different languages). We choose to use this prompt engineering method as it reflects a more realistic scenario which then better facilitates reproducibility and further work.

Metrics. We report overall accuracy (class and language specific as well), AUCROC, F1-Macro, and samples per second for all models.

Scenarios. We evaluate: (1) Detection of source code in all the 7 programming languages included in our dataset (2) Detection of multilingual text in the 8 languages provided in our dataset (3) Cross language performance for language specific fine-tuned checkpoints of the model.

4.2 Results and Analysis

Multilingual Text Detection. Table 1 presents the performance of the models on multilingual text detection. Similar to the code domain, GPT-4o underperforms with an AUCROC of 0.573 and F1-Macro of 0.490, demonstrating limited binary classification capabilities in this specific setting. Pretrained multilingual models, especially XLM-RoBERTa-large, show reasonably strong performance out of the box, with AUCROC values ranging from 0.891 to 0.937.
When fine-tuned, all the SLMs achieve very similar performance among each other, with XLM-RoBERTa-large achieving an especially high F1-Macro score, which could be explained due to its much larger size. Additionally, the results from the cross-language scenario are included in Table 2

Code Detection. Table 1 showcases the performance of models on the task of detecting machine-generated code. The baseline models include CodeBERT [10], CodeBERTa, and GPT-4o [19], with their respective fine-tuned variants. Particularly, GPT-4o, a powerful LLM, performs significantly worse than SLMs on both metrics, achieving only 0.535 AUCROC and 0.414 F1-Macro. In contrast, CodeBERT and CodeBERTa, even without fine-tuning, achieve substantially higher AUCROC scores of 0.953 and 0.948, respectively, all while requiring substantially less compute.
Upon fine-tuning, both CodeBERT and CodeBERTa achieve near-perfect performance. CodeBERTa (fine-tuned) achieves very similar results to CodeBERT (while being half the size of CodeBERT) with an AUCROC of 0.985 and F1-Macro of 0.937, surpassing CodeBERT but(fine-tuned) not by much. Additionally, the results from the cross-language performance scenario are included in Table 3

5 Conclusion

We presented **DuoLens**, a dual-encoder system that unifies detection of AI-generated text and code via the combination of CodeBERT [10]and CodeBERTa using a fine-tuned fusion head. DuoLens outperforms strong baselines on multilingual text and source code across the languages in our datasets, and shows improved cross-language generalization (specifically, in Java). Further details regarding directions for future work are included in Appendix D. By building upon the approach of fine-tuning BERT models, but with a keen view towards effective and balanced dataset creation,

Table 1: Detection performance: (i) *Code detection* and (ii) *Multilingual text detection* reported as AUC-ROC and F1-Macro. Highest score per column is in **bold**.

Model	AUC-ROC	F1-Macro
CodeBERT	0.953	0.888
CodeBERTa	0.948	0.883
Qwen2.5 Coder 3B	0.492	0.388
GPT-4o	0.535	0.414
CodeBERT (fine-tuned)	0.98	0.93
DuoLens (fine-tuned)	**0.985**	**0.937**

(i) Code detection

Model	AUC-ROC	F1-Macro
XLM-R-large	0.937	0.865
XLM-R	0.915	0.836
mBERT	0.891	0.816
GPT-4o (few-shot)	0.573	0.49
XLM-R-large (fine-tuned)	0.974	**0.924**
XLM-R (fine-tuned)	0.974	0.899
DuoLens (fine-tuned)	**0.975**	0.911

(ii) Multilingual text detection

we then effectively address the pertinent issue of detecting AI generated content across domains – making this more accessible regardless of compute limitations or available models.

6 Limitations

At this stage, DuoLens focuses on the use of SLMs in binary classification of AI-generated natural language across modalities. However, due to the inherent limitation of BERT based models being encoder-only, there is no output visible to the user. This potentially would leave room for future work, in which an LLM could be integrated for sentence-level classification. The datasets we created can also be improved, specifically the multilingual text dataset which has language imbalances as illustrated in Figure 4a. Additionally, DuoLens inherits biases of its underlying models; while we try our best to train on diverse data, fairness is not guaranteed. A similar issue is apparent where even with robust evaluation, results may be constrained to specific languages and models within their respective training and test sets.

Furthermore, we must also acknowledge that due to limited computational resources and for the purpose of efficacy our baselines are largely constrained to open-weight LLMs, with only GPT 4o used as a closed source AI generator.

References

[1] Vibhor Agarwal, Yulong Pei, Salwa Alamir, and Xiaomo Liu. Codemirage: Hallucinations in code generated by large language models, 2025. URL https://arxiv.org/abs/2408.08333.

[2] Sri Haritha Ambati, Norah Ridley, Enrico Branca, and Natalia Stakhanova. Navigating (in)security of ai-generated code. In *2024 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 1–8, 2024. doi: 10.1109/CSR61664.2024.10679468.

[3] Brett A. Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. Programming is hard – or at least it used to be: Educational opportunities and challenges of ai code generation, 2022. URL https://arxiv.org/abs/2212.01020.

[4] Eszter Benke and Andrea Szőke. Academic integrity in the time of artificial intelligence: Exploring student attitudes. *International Journal of Sociology of Education*, 16:91–108, 10 2024.

[5] Amrita Bhattacharjee, Raha Moraffah, Joshua Garland, and Huan Liu. Eagle: A domain generalization framework for ai-generated text detection, 2024. URL https://arxiv.org/abs/2403.15690.

[6] Yutian Chen, Hao Kang, Vivian Zhai, Liangze Li, Rita Singh, and Bhiksha Raj. Gpt-sentinel: Distinguishing human and chatgpt generated content, 2023. URL https://arxiv.org/abs/2305.07969.

[7] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale, 2020. URL https://arxiv.org/abs/1911.02116.

[8] Basak Demirok and Mucahid Kutlu. Aigcodeset: A new annotated dataset for ai generated code detection, 2025. URL https://arxiv.org/abs/2412.16594.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

[10] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. Codebert: A pre-trained model for programming and natural languages, 2020. URL https://arxiv.org/abs/2002.08155.

[11] Biyang Guo, Xin Zhang, Ziyuan Wang, Minqi Jiang, Jinran Nie, Yuxuan Ding, Jianwei Yue, and Yupeng Wu. How close is chatgpt to human experts? comparison corpus, evaluation, and detection, 2023. URL https://arxiv.org/abs/2301.07597.

[12] Abhimanyu Hans, Avi Schwarzschild, Valeriia Cherepanova, Hamid Kazemi, Aniruddha Saha, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Spotting llms with binoculars: Zero-shot detection of machine-generated text, 2024. URL https://arxiv.org/abs/2401.12070.

[13] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

[14] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. *arXiv:1909.09436 [cs, stat]*, September 2019. URL http://arxiv.org/abs/1909.09436. arXiv: 1909.09436.

[15] Jiatao Li, Mao Ye, Cheng Peng, Xunjian Yin, and Xiaojun Wan. Agent-x: Adaptive guideline-based expert network for threshold-free ai-generated text detection, 2025. URL https://arxiv.org/abs/2505.15261.

[16] Dominik Macko, Robert Moro, Adaku Uchendu, Jason Lucas, Michiharu Yamashita, Matúš Pikuliak, Ivan Srba, Thai Le, Dongwon Lee, Jakub Simko, and Maria Bielikova. Multitude: Large-scale multilingual machine-generated text detection benchmark. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, page 9960–9987. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.616. URL http://dx.doi.org/10.18653/v1/2023.emnlp-main.616.

[17] Dominik Macko, Jakub Kopal, Robert Moro, and Ivan Srba. Multisocial: Multilingual benchmark of machine-generated text detection of social-media texts, 2025. URL https://arxiv.org/abs/2406.12549.

[18] Chengzhi Mao, Carl Vondrick, Hao Wang, and Junfeng Yang. Raidar: generative ai detection via rewriting. *ArXiv*, abs/2401.12970, 2024. URL https://api.semanticscholar.org/CorpusID:267095281.

[19] OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Madry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, Alex Paino, Alex Renzin, Alex Tachard Passos, Alexander Kirillov, Alexi Christakis, Alexis Conneau, Ali Kamali, Allan Jabri, Allison Moyer, Allison Tam, Amadou Crookes, Amin Tootoochian, Amin Tootoonchian, Ananya Kumar, Andrea Vallone, Andrej Karpathy, Andrew Braunstein, Andrew Cann, Andrew Codispoti, Andrew Galu, Andrew Kondrich, Andrew Tulloch, Andrey Mishchenko, Angela Baek, Angela Jiang, Antoine Pelisse, Antonia Woodford, Anuj Gosalia, Arka Dhar, Ashley Pantuliano, Avi Nayak, Avital Oliver, Barret Zoph, Behrooz Ghorbani, Ben Leimberger, Ben Rossen, Ben Sokolowsky, Ben Wang, Benjamin Zweig, Beth Hoover, Blake Samic, Bob McGrew, Bobby Spero, Bogo Giertler, Bowen Cheng, Brad Lightcap, Brandon Walkin, Brendan Quinn, Brian Guarraci, Brian Hsu, Bright Kellogg, Brydon Eastman, Camillo Lugaresi, Carroll Wainwright, Cary Bassin, Cary Hudson, Casey Chu, Chad Nelson, Chak Li, Chan Jun Shern, Channing Conger, Charlotte Barette, Chelsea Voss, Chen Ding, Cheng Lu, Chong Zhang, Chris Beaumont, Chris Hallacy, Chris Koch, Christian Gibson, Christina Kim, Christine Choi, Christine McLeavey, Christopher Hesse, Claudia Fischer, Clemens Winter,

Coley Czarnecki, Colin Jarvis, Colin Wei, Constantin Koumouzelis, Dane Sherburn, Daniel Kappler, Daniel Levin, Daniel Levy, David Carr, David Farhi, David Mely, David Robinson, David Sasaki, Denny Jin, Dev Valladares, Dimitris Tsipras, Doug Li, Duc Phong Nguyen, Duncan Findlay, Edede Oiwoh, Edmund Wong, Ehsan Asdar, Elizabeth Proehl, Elizabeth Yang, Eric Antonow, Eric Kramer, Eric Peterson, Eric Sigler, Eric Wallace, Eugene Brevdo, Evan Mays, Farzad Khorasani, Felipe Petroski Such, Filippo Raso, Francis Zhang, Fred von Lohmann, Freddie Sulit, Gabriel Goh, Gene Oden, Geoff Salmon, Giulio Starace, Greg Brockman, Hadi Salman, Haiming Bao, Haitang Hu, Hannah Wong, Haoyu Wang, Heather Schmidt, Heather Whitney, Heewoo Jun, Hendrik Kirchner, Henrique Ponde de Oliveira Pinto, Hongyu Ren, Huiwen Chang, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian O'Connell, Ian Osband, Ian Silber, Ian Sohl, Ibrahim Okuyucu, Ikai Lan, Ilya Kostrikov, Ilya Sutskever, Ingmar Kanitscheider, Ishaan Gulrajani, Jacob Coxon, Jacob Menick, Jakub Pachocki, James Aung, James Betker, James Crooks, James Lennon, Jamie Kiros, Jan Leike, Jane Park, Jason Kwon, Jason Phang, Jason Teplitz, Jason Wei, Jason Wolfe, Jay Chen, Jeff Harris, Jenia Varavva, Jessica Gan Lee, Jessica Shieh, Ji Lin, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joanne Jang, Joaquin Quinonero Candela, Joe Beutler, Joe Landers, Joel Parish, Johannes Heidecke, John Schulman, Jonathan Lachman, Jonathan McKay, Jonathan Uesato, Jonathan Ward, Jong Wook Kim, Joost Huizinga, Jordan Sitkin, Jos Kraaijeveld, Josh Gross, Josh Kaplan, Josh Snyder, Joshua Achiam, Joy Jiao, Joyce Lee, Juntang Zhuang, Justyn Harriman, Kai Fricke, Kai Hayashi, Karan Singhal, Katy Shi, Kavin Karthik, Kayla Wood, Kendra Rimbach, Kenny Hsu, Kenny Nguyen, Keren Gu-Lemberg, Kevin Button, Kevin Liu, Kiel Howe, Krithika Muthukumar, Kyle Luther, Lama Ahmad, Larry Kai, Lauren Itow, Lauren Workman, Leher Pathak, Leo Chen, Li Jing, Lia Guy, Liam Fedus, Liang Zhou, Lien Mamitsuka, Lilian Weng, Lindsay McCallum, Lindsey Held, Long Ouyang, Louis Feuvrier, Lu Zhang, Lukas Kondraciuk, Lukasz Kaiser, Luke Hewitt, Luke Metz, Lyric Doshi, Mada Aflak, Maddie Simens, Madelaine Boyd, Madeleine Thompson, Marat Dukhan, Mark Chen, Mark Gray, Mark Hudnall, Marvin Zhang, Marwan Aljubeh, Mateusz Litwin, Matthew Zeng, Max Johnson, Maya Shetty, Mayank Gupta, Meghan Shah, Mehmet Yatbaz, Meng Jia Yang, Mengchao Zhong, Mia Glaese, Mianna Chen, Michael Janner, Michael Lampe, Michael Petrov, Michael Wu, Michele Wang, Michelle Fradin, Michelle Pokrass, Miguel Castro, Miguel Oom Temudo de Castro, Mikhail Pavlov, Miles Brundage, Miles Wang, Minal Khan, Mira Murati, Mo Bavarian, Molly Lin, Murat Yesildal, Nacho Soto, Natalia Gimelshein, Natalie Cone, Natalie Staudacher, Natalie Summers, Natan LaFontaine, Neil Chowdhury, Nick Ryder, Nick Stathas, Nick Turley, Nik Tezak, Niko Felix, Nithanth Kudige, Nitish Keskar, Noah Deutsch, Noel Bundick, Nora Puckett, Ofir Nachum, Ola Okelola, Oleg Boiko, Oleg Murk, Oliver Jaffe, Olivia Watkins, Olivier Godement, Owen Campbell-Moore, Patrick Chao, Paul McMillan, Pavel Belov, Peng Su, Peter Bak, Peter Bakkum, Peter Deng, Peter Dolan, Peter Hoeschele, Peter Welinder, Phil Tillet, Philip Pronin, Philippe Tillet, Prafulla Dhariwal, Qiming Yuan, Rachel Dias, Rachel Lim, Rahul Arora, Rajan Troll, Randall Lin, Rapha Gontijo Lopes, Raul Puri, Reah Miyara, Reimar Leike, Renaud Gaubert, Reza Zamani, Ricky Wang, Rob Donnelly, Rob Honsby, Rocky Smith, Rohan Sahai, Rohit Ramchandani, Romain Huet, Rory Carmichael, Rowan Zellers, Roy Chen, Ruby Chen, Ruslan Nigmatullin, Ryan Cheu, Saachi Jain, Sam Altman, Sam Schoenholz, Sam Toizer, Samuel Miserendino, Sandhini Agarwal, Sara Culver, Scott Ethersmith, Scott Gray, Sean Grove, Sean Metzger, Shamez Hermani, Shantanu Jain, Shengjia Zhao, Sherwin Wu, Shino Jomoto, Shirong Wu, Shuaiqi, Xia, Sonia Phene, Spencer Papay, Srinivas Narayanan, Steve Coffey, Steve Lee, Stewart Hall, Suchir Balaji, Tal Broda, Tal Stramer, Tao Xu, Tarun Gogineni, Taya Christianson, Ted Sanders, Tejal Patwardhan, Thomas Cunninghman, Thomas Degry, Thomas Dimson, Thomas Raoux, Thomas Shadwell, Tianhao Zheng, Todd Underwood, Todor Markov, Toki Sherbakov, Tom Rubin, Tom Stasi, Tomer Kaftan, Tristan Heywood, Troy Peterson, Tyce Walters, Tyna Eloundou, Valerie Qi, Veit Moeller, Vinnie Monaco, Vishal Kuo, Vlad Fomenko, Wayne Chang, Weiyi Zheng, Wenda Zhou, Wesam Manassra, Will Sheu, Wojciech Zaremba, Yash Patil, Yilei Qian, Yongjik Kim, Youlong Cheng, Yu Zhang, Yuchen He, Yuchen Zhang, Yujia Jin, Yunxing Dai, and Yury Malkov. Gpt-4o system card, 2024. URL https://arxiv.org/abs/2410.21276.

[20] Daniil Orel, Indraneil Paul, Iryna Gurevych, and Preslav Nakov. Droid: A resource suite for ai-generated code detection, 2025. URL https://arxiv.org/abs/2507.10583.

[21] Ruchir Puri, David S. Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, Veronika Thost, Luca Buratti, Saurabh Pujar, Shyam Ramji, Ulrich Finkler, Susan Malaika, and Frederick Reiss.

Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks, 2021. URL https://arxiv.org/abs/2105.12655.

[22] Irene Solaiman, Miles Brundage, Jack Clark, Amanda Askell, Ariel Herbert-Voss, Jeff Wu, Alec Radford, Gretchen Krueger, Jong Wook Kim, Sarah Kreps, et al. Release strategies and the social impacts of language models. *arXiv preprint arXiv:1908.09203*, 2019.

[23] Hyunjae Suh, Mahan Tafreshipour, Jiawei Li, Adithya Bhattiprolu, and Iftekhar Ahmed. An empirical study on automatically detecting ai-generated source code: How far are we?, 2024. URL https://arxiv.org/abs/2411.04299.

[24] Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline, 2019. URL https://arxiv.org/abs/1905.05950.

[25] Vivek Verma, Eve Fleisig, Nicholas Tomlin, and Dan Klein. Ghostbuster: Detecting text ghostwritten by large language models, 2024. URL https://arxiv.org/abs/2305.15047.

[26] Yuxia Wang, Jonibek Mansurov, Petar Ivanov, Jinyan Su, Artem Shelmanov, Akim Tsvigun, Chenxi Whitehouse, Osama Mohammed Afzal, Tarek Mahmoud, Toru Sasaki, Thomas Arnold, Alham Aji, Nizar Habash, Iryna Gurevych, and Preslav Nakov. M4: Multi-generator, multi-domain, and multi-lingual black-box machine-generated text detection. In Yvette Graham and Matthew Purver, editors, *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1369–1407, St. Julian's, Malta, March 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.eacl-long.83.

Appendix: Code Samples, Prompt Templates, and Multi-LLM Protocol

This appendix provides representative code samples (human-written vs. AI-generated) for each programming language (Python, Java, Go, C++, and JavaScript). Additionally, an overview of the methodology is presented in Figure 2

A Code Samples from our Dataset

A.1 Python Samples

Human-Written

```python
N=int(input())
Z=[0]*N
W=[0]*N

for i in range(N):
    x,y=map(int,input().split())
    Z[i]=x+y
    W[i]=x-y

alpha=max(Z)-min(Z)
beta=max(W)-min(W)
print(max(alpha,beta))
```

AI-Generated

```python
def flatten(m, p=()):
    """"""
    Flattens a mapping tree so that all leaf nodes appear
    as tuples in a list containing a path and a value.

    Parameters:
    m (dict): A dictionary that may contain other dictionaries.

    Returns:
    list of tuples: A list of tuples where the first item is
                    a tuple representing the path to the leaf node
                    and the second item is the value of the leaf node.
    """"""
    result = []
    for k, v in m.items():
        if isinstance(v, dict):
            result.extend(flatten(v, p + (k,)))
        else:
            result.append((p + (k,), v))
    return result
```

A.2 Java Samples

Human-Written

```java
private DataList fillDataList(List<Global> results, long records,
    Query query, Map<String, QueryParameter> parameterMap) throws
    AWException {
    DataListBuilder builder = getBean(DataListBuilder.class);
    boolean paginate = query == null || !query.isPaginationManaged();
    builder.setEnumQueryResult(results)
      .setRecords(records)
      .setPage(parameterMap.get(AweConstants.QUERY_PAGE).getValue().
          asLong())
      .setMax(parameterMap.get(AweConstants.QUERY_MAX).getValue().
          asLong())
      .paginate(paginate)
      .generateIdentifiers();

    // If query is defined, fill with query data
    if (query != null) {
```

```
      // Add transformations & translations
      builder = processDataList(builder, query, parameterMap);
    }

    // Sort datalist
    builder = sortDataList(builder, parameterMap);

    return builder.build();
  }
```

AI-Generated

```
public void scheduleDelayedTask() {
    mLoginStarted = false;
    Handler handler = new Handler(Looper.getMainLooper());
    Runnable task = new Runnable() {
        @Override
        public void run() {
            // Execute the task
            // You can add any code here that needs to run after the delay
        }
    };

    Log.d(""TaskScheduler"", ""Scheduling task with a delay of "" + WAIT_FOR_LOGIN_START_MS + "" milliseconds."");
    handler.postDelayed(task, WAIT_FOR_LOGIN_START_MS);
}
```

A.3 Go Samples

Human-Written

```
package domain

import (
  ""fmt""
  ""regexp""
)

type videoID struct {
  Value string
}

func newVideoID(uuid string) (*videoID, error) {

  if !isValidUUID(uuid) {
    return nil, fmt.Errorf(""UUID (%s)its not valid"", uuid)
  }

  vid := &videoID{
    Value: uuid,
  }

  return vid, nil
}

func isValidUUID(uuid string) bool {
  r := regexp.MustCompile(""^[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-4[a-fA-F0-9]{3}-[8|9|aA|bB][a-fA-F0-9]{3}-[a-fA-F0-9]{12}$"")
  return r.MatchString(uuid)
}
```

AI-Generated

```
package ifname

```

```go
import (
  ""sync""
  ""time""
)

// TTLValType represents a value with a timestamp
type TTLValType struct {
  time time.Time // when entry was added
  val  valType
}

// timeFunc is a function that returns the current time
type timeFunc func() time.Time

// TTLCache is a cache with a time to live
type TTLCache struct {
  mu            sync.RWMutex
  validDuration time.Duration
  lru           LRUCache
  now           timeFunc
}

// NewTTLCache returns a new TTL cache
func NewTTLCache(valid time.Duration, capacity uint) *TTLCache {
  return &TTLCache{
    lru:           NewLRUCache(capacity),
    validDuration: valid,
    now:           time.Now,
  }
}

// Get returns the value associated with the given key and its age
func (c *TTLCache) Get(key keyType) (valType, bool, time.Duration) {
  c.mu.RLock()
  v, ok := c.lru.Get(key)
  c.mu.RUnlock()
  if!ok {
    return valType{}, false, 0
  }
  age := c.now().Sub(v.time)
  if age < c.validDuration {
    return v.val, ok, age
  } else {
    c.mu.Lock()
    c.lru.Delete(key)
    c.mu.Unlock()
    return valType{}, false, 0
  }
}

// Put inserts or updates the value associated with the given key
func (c *TTLCache) Put(key keyType, value valType) {
  c.mu.Lock()
  v := TTLValType{
    val:  value,
    time: c.now(),
  }
  c.lru.Put(key, v)
  c.mu.Unlock()
}

// Delete removes the value associated with the given key
func (c *TTLCache) Delete(key keyType) {
  c.mu.Lock()
  c.lru.Delete(key)
```

```
  c.mu.Unlock()
}
```

A.4 C++ Samples

Human-Written

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
 public:
  vector<vector<string>> printTree(TreeNode* root) {
    int d = depth(root);
    int n = 0;
    for (int i = 1, cnt = 0; cnt < d; i *= 2, ++cnt) n += i;
    vector<vector<string>> res(d, vector<string>(n));
    build(root, res, 0, 0, n);
    return res;
  }

 private:
  int depth(TreeNode *root) {
    if (!root) return 0;
    return max(depth(root->left), depth(root->right)) + 1;
  }

  void build(TreeNode *root, vector<vector<string>> &res, int d, int begin,
             int end) {
    if (!root) return;
    int mid = (begin + end) / 2;
    res[d][mid] = to_string(root->val);
    build(root->left, res, d + 1, begin, mid);
    build(root->right, res, d + 1, mid + 1, end);
  }
};
```

AI-Generated

```
#include <iostream>
#include <string>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

// Define the packet struct
struct packet {
    int id;
    int sequenceNumber;
    int priority;
    char payload[1024];
};

// Function to send a packet
void sendPacket(int id, int sequenceNumber, int priority, int port, int index, int sock) {
    // Initialize the packet struct
    struct packet packet;
    packet.id = id;
```

```
    packet.sequenceNumber = sequenceNumber;
    packet.priority = priority;

    // Populate the packet with a character payload based on the index
    if (index >= 0 && index < 1024) {
        packet.payload[index] = 'A';
    }

    // Convert the packet to a string
    char packetStr[1024];
    sprintf(packetStr, ""%d %d %d %s"", packet.id, packet.
        sequenceNumber, packet.priority, packet.payload);

    // Send the packet using the socket-based send function
    try {
        send(sock, packetStr, strlen(packetStr), 0);
    } catch (const std::exception& e) {
        std::cerr << ""Error sending packet: "" << e.what() << std::
            endl;
    }
}
```

A.5 JavaScript Samples

Human-Written

```
function inspect(value, opts) {
  const ctx = {
    budget: {},
    indentationLvl: 0,
    seen: [],
    currentDepth: 0,
    stylize: stylizeNoColor,
    showHidden: inspectDefaultOptions.showHidden,
    depth: inspectDefaultOptions.depth,
    colors: inspectDefaultOptions.colors,
    customInspect: inspectDefaultOptions.customInspect,
    showProxy: inspectDefaultOptions.showProxy,
    maxArrayLength: inspectDefaultOptions.maxArrayLength,
    breakLength: inspectDefaultOptions.breakLength,
    compact: inspectDefaultOptions.compact,
    sorted: inspectDefaultOptions.sorted,
    getters: inspectDefaultOptions.getters };
  if (arguments.length > 1) {
    if (arguments.length > 2) {
      if (arguments[2] !== undefined) {
        ctx.depth = arguments[2];
      }
      if (arguments.length > 3 && arguments[3] !== undefined) {
        ctx.colors = arguments[3];
      }
    }
    if (typeof opts === 'boolean') {
      ctx.showHidden = opts;
    } else if (opts) {
      const optKeys = Object.keys(opts);
      for (const key of optKeys) {
        if (hasOwnProperty(inspectDefaultOptions, key) || key === '
            stylize') {
          ctx[key] = opts[key];
        } else if (ctx.userOptions === undefined) {
          ctx.userOptions = opts;
        }
      }
    }
  }
```

```
  if (ctx.colors) {
    ctx.stylize = stylizeWithColor;
  }
  if (ctx.maxArrayLength === null) {
    ctx.maxArrayLength = Infinity;
  }
  return formatValue(ctx, value, 0);
}
```

AI-Generated

```
module.exports = function(config) {
  config.set({
    // Base path used to resolve all patterns (e.g., files, exclude)
    basePath: '',

    // Frameworks to use (Jasmine is the testing framework)
    frameworks: ['jasmine'],

    // List of files/patterns to load in the browser
    files: [
      'dist/gainda.js', // Main project file
      'test/spec/**/*.spec.js' // Test files
    ],

    // List of files/patterns to exclude
    exclude: [],

    // Preprocessors to transform files before serving them to the
        browser
    preprocessors: {
      'dist/gainda.js': ['coverage'] // Optional: Add coverage
          preprocessing
    },

    // Test results reporter to use
    reporters: ['progress'],

    // Web server port
    port: 9876,

    // Enable/disable colors in the output (reporters and logs)
    colors: true,

    // Level of logging
    // Possible values: config.LOG_DISABLE || config.LOG_ERROR ||
        config.LOG_WARN || config.LOG_INFO || config.LOG_DEBUG
    logLevel: config.LOG_INFO,

    // Enable/disable watching file and executing tests whenever any
        file changes
    autoWatch: true,

    // Start these browsers
    browsers: ['Chrome'],

    // Continuous Integration mode
    // If true, Karma captures browsers, runs the tests, and exits
    singleRun: false,

    // Concurrency level
    // How many browser instances should be started simultaneously
    concurrency: Infinity
  });
};
```

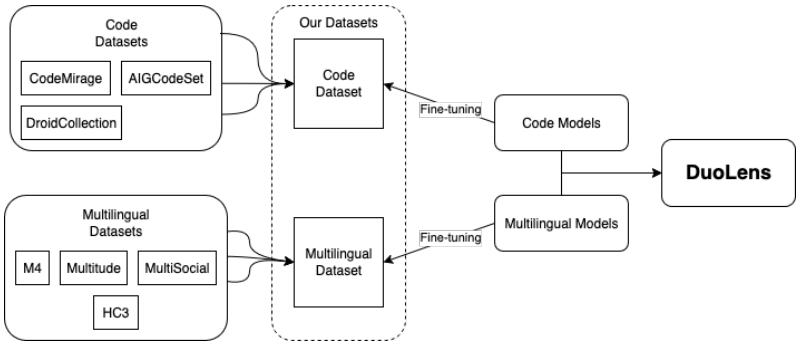
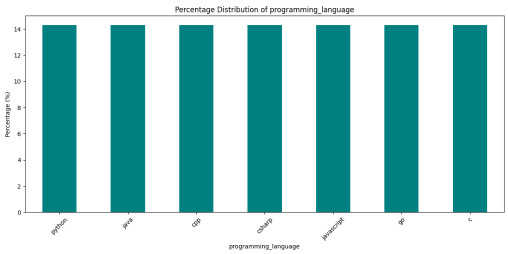
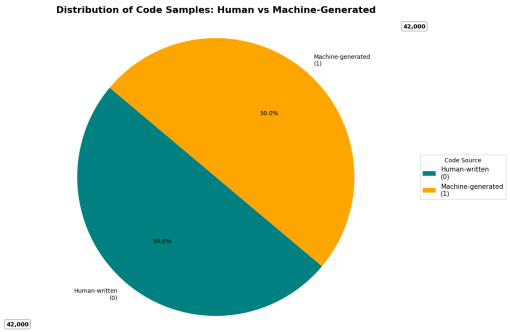


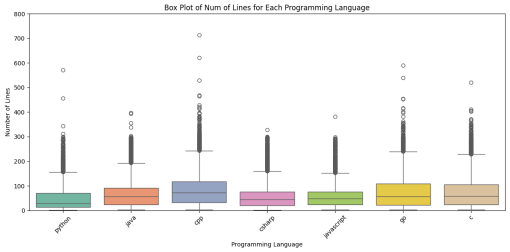
Figure 2: An overview of the methodology adopted in this work is as follows. We first constructed new datasets by curating and extending samples from existing resources. These datasets were subsequently employed to fine-tune and evaluate the models that constitute DuoLens.



(a) The number of samples for each programming language



(b) The percentage distribution of the samples in the dataset between human-written and machine-generated



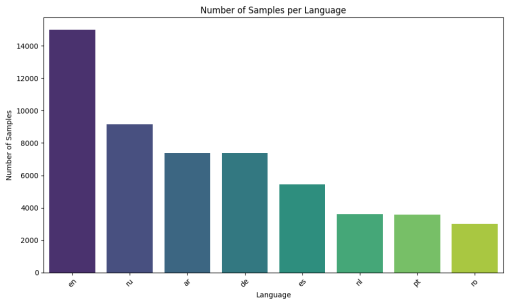
(c) Box plots of the number of lines the code samples have for each language

Figure 3: Source code dataset visualizations

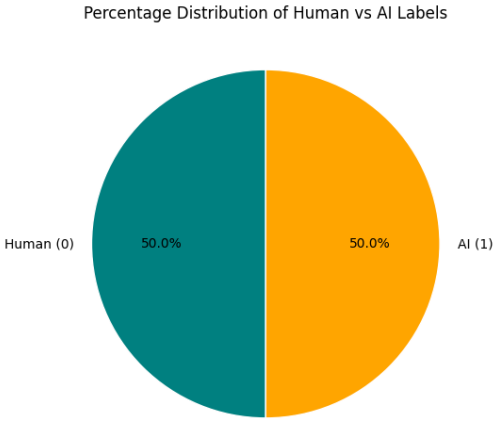
B Dataset Visualisations

B.1 Source Code Dataset

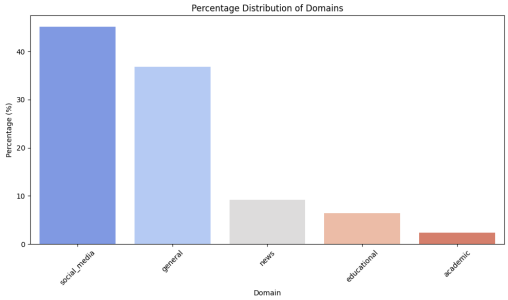
These visualizations illustrate how the proposed source code dataset addresses the two primary limitations observed in prior resources. As shown in Figure 3a, the dataset maintains an equal distribution of samples across all languages, and as depicted in Figure 3b, it achieves a balanced representation of both human-written and machine-generated classes. This design mitigates potential biases during the fine-tuning process. Furthermore, Figure 3c demonstrates that the dataset exhibits a substantial number of outliers with respect to sample length, which is advantageous as it reflects a broader diversity and range of examples rather than a collection of homogenous samples.



(a) Number of samples per language in our dataset



(b) Percentage distribution of machine-generated and human-written samples in our dataset



(c) Percentage distribution of the domain of the samples in our dataset

Figure 4: Multilingual text dataset visualizations

B.2 Multilingual Text Dataset

These visualizations demonstrate how the proposed multilingual text dataset addresses key limitations identified in prior resources. As shown in Figure 4b, the dataset maintains a balanced distribution between human-written and machine-generated samples, thereby reducing the risk of bias during fine-tuning. In addition, Figure 4c illustrates the balanced representation of linguistic styles: while social media samples predominantly contain casual and informal language, often including slang, the remaining domains exhibit more formal registers with comparatively refined diction. This diversity is advantageous, as it ensures broader coverage of linguistic variation while preventing overfitting to a single style. Nevertheless, the effort to simultaneously balance both domain and style necessitated a compromise in language-level balance, as reflected in Figure 4a.

C Language Specific Results

C.1 Natural Language Specific Results

DuoLens was evaluated on specific languages and its accuracy per language was measured using our dataset (besides from the language it was finetuned upon). The results are included in Table 2 where the Spanish-specific DuoLens performed the best among all languages and achieved the highest accuracy out of all.

C.2 Programming Language Specific Results

DuoLens was evaluated on specific languages and its accuracy per language was measured using our dataset (besides from the language it was finetuned upon). The results are included in Table

Model Name	English	Spanish	German	Dutch	Portuguese	Russian	Arabic	Romanian
DuoLens (English)	-	0.6716	0.7953	0.7581	0.7758	0.5714	0.7511	0.8606
DuoLens (Spanish)	0.6406	-	**0.8695**	0.8176	**0.8659**	**0.7461**	**0.8383**	0.8726
DuoLens (German)	**0.6441**	0.7075	-	**0.8191**	0.8114	0.6493	0.8191	0.8933
DuoLens (Dutch)	0.6021	0.6911	0.6667	-	0.8108	0.6134	0.7671	0.8836
DuoLens (Portuguese)	0.6075	0.7352	0.6766	0.7762	-	0.6252	0.7286	0.88
DuoLens (Russian)	0.5917	**0.7642**	0.8334	0.8143	0.8256	-	0.7822	**0.896**
DuoLens (Arabic)	0.6109	0.6512	0.6656	0.7501	0.6919	0.6496	-	0.784
DuoLens (Romanian)	0.6053	0.6761	0.6439	0.7609	0.8016	0.5850	0.7036	-

Table 2: Accuracy in detection for multilingual text samples from each language

Model Name	Python	Java	JavaScript	C	C#	C++	Go
DuoLens (Python)	-	0.791	0.746	0.846	0.775	0.791	0.7
DuoLens (Java)	**0.649**	-	0.778	**0.899**	0.833	**0.895**	**0.864**
DuoLens (JavaScript)	0.504	0.849	-	0.8	0.865	0.856	0.764
DuoLens (C)	0.517	0.755	**0.787**	-	0.745	0.851	0.792
DuoLens (C#)	0.64	0.774	0.711	0.828	-	0.762	0.77
DuoLens (C++)	0.585	**0.867**	0.782	0.884	**0.878**	-	0.807
DuoLens (Go)	0.467	0.677	0.697	0.735	0.69	0.731	-

Table 3: Accuracy in detection for code samples from each language

3 where the Java specific DuoLens performed the best among all the languages and achieved the highest accuracy out of them.

D Future Work

We identify several avenues for future work, including, but not limited to: extending coverage to additional natural and programming languages, incorporating sentence-level detection, developing agent-based or hybrid systems with LLMs to provide explanatory insights for classification outcomes, and extensions to the datasets introduced.