IMGTB: A Framework for Machine-Generated Text Detection Benchmarking

Michal Spiegel[1,2] and **Dominik Macko**[1]
[1] Kempelen Institute of Intelligent Technologies
[2] Faculty of Informatics, Masaryk University
michal.spiegel@intern.kinit.sk, dominik.macko@kinit.sk

Abstract

In the era of large language models generating high quality texts, it is a necessity to develop methods for detection of machine-generated text to avoid their harmful use or simply for annotation purposes. It is, however, also important to properly evaluate and compare such developed methods. Recently, a few benchmarks have been proposed for this purpose; however, integration of newest detection methods is rather challenging, since new methods appear each month and provide slightly different evaluation pipelines. In this paper, we present the IMGTB framework, which simplifies the benchmarking of machine-generated text detection methods by easy integration of custom (new) methods and evaluation datasets. In comparison to existing frameworks, it enables to objectively compare statistical metric-based zero-shot detectors with classification-based detectors and with differently fine-tuned detectors. Its configurability and flexibility makes research and development of new detection methods easier, especially their comparison to the existing state-of-the-art detectors. The default set of analyses, metrics and visualizations offered by the tool follows the established practices of machine-generated text detection benchmarking found in state-of-the-art literature.

1 Introduction

Due to indistinguishability between human-written texts and high-quality texts generated by modern large language models (LLMs) (Sadasivan et al., 2023), the machine-generated text detection (MGTD) belongs to the key challenges identified by (Kaddour et al., 2023). MGTD methods are needed in many areas, such as prevention of disinformation spreading, plagiarism, impersonation and identity theft, automated scams and frauds, or even prevention of unintentional inclusion of lesser quality generated texts in future models' training data (Kaddour et al., 2023; Weidinger et al., 2021; Zellers et al., 2019; Wahle et al., 2022; Vykopal et al., 2023).

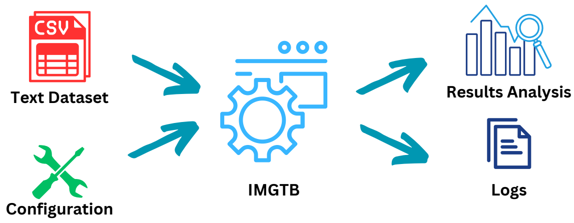

Figure 1: IMGTB framework exemplar usage overview.

Regardless of the area of application, we are witnessing a race of new MGTD methods competing with new *generation* methods and appearing monthly. This presents a challenge to efficiently evaluate and benchmark the new methods. The problem is twofold, missing the uniform implementation of the methods and standardized evaluation. Even when source codes for experiment replication are released, they are usually too specific and not flexible for reuse. Moreover, across application areas, domains, text lengths, or topics, the performance of different MGTD methods varies. Therefore, a flexible way of comparison over various datasets (even custom ones) is currently missing. These problems are usually addressed by common benchmarking frameworks.

There is a lack of flexibility, configurability, and extensibility in the current MGTD benchmarking frameworks; therefore, we have focused on refining the most recent one, MGTBench (He et al., 2023), by integrating missing features and extending support to new types of detection pipelines. The key contributions of the proposed extended framework IMGTB[1] are as follows:

- *objective comparison* of statistical metric-based zero-shot methods with others,

[1] https://github.com/kinit-sk/IMGTB

Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), pages 172–179
August 11-16, 2024 ©2024 Association for Computational Linguistics

- *integration* of the newest MGTD methods (e.g., MFD, Binoculars, S5) and fine-tuning processes (e.g., PEFT, per-language)[2],
- *simplified ability* to implement custom MGTD methods (plug-in by abstract classes and templates),
- more flexible usage of custom evaluation *datasets* (multi-format support),
- increased *configurability* of the benchmark settings (e.g., classifier selection),
- benchmark results *analysis* (configurability, automated charts generation).

2 Related Works

Due to increasing quality of texts generated by modern LLMs, the research around detection of machine-generated text increased its importance. However, a common way to properly compare several detection methods was missing, mainly due to missing publicly available datasets. Few years ago, MGTD researchers mostly used data generated by a single LLM, such as GPT-2[3] or Grover (Zellers et al., 2019), results on which could not be properly generalized. Later on, larger-scale multi LLM benchmarks for MGTD task have been proposed, such as TuringBench (Uchendu et al., 2021), DeepfakeTextDetect (Li et al., 2023), M4 (Wang et al., 2023), or MULTITuDE (Macko et al., 2023). As a result, MGTD methods can now be evaluated on such benchmark datasets and compared to each other. However, these datasets do not share common class labels, structure, or form, what makes the evaluation on multiple of them complicated and unnecessarily prolongs the research.

The other issue significantly prolonging the research is a missing unified implementation of existing MGTD methods. It leaves on the researchers a burden to either reuse the published source codes of individual methods (if there is some), which are different among each other and require customization, or implement them completely into their evaluation framework to be evaluated in a unified way with their newly proposed MGTD method. Some of the proposed MGTD methods, such as DetectGPT (Mitchell et al., 2023), released the full source code including implementation of other existing SOTA methods, enabling complete replication of experiments and providing a good basis to build upon.

[2] see section 3.2.4 for a description of the implemented methods

[3] https://github.com/openai/gpt-2-output-dataset

The result is a faster advancement by extension of the original method, in the form of DetectLLM (Su et al., 2023) or Fast-DetectGPT (Bao et al., 2023), proving the benefits of full replication possibilities.

However, these methods focused on zero-shot statistical-based detection of machine-generated text, comparing various statistical metrics to distinguish between human-written and machine-generated samples, not providing the classification prediction. Thus, the implementations do not allow easy comparison with supervised high-performing pretrained LLMs finetuned for MGTD task, such as the popular OpenAI detector (Solaiman et al., 2019). The proposed MGTBench framework (He et al., 2023) attempted to solve the problem, by implementation of these methods in a common framework. Using a dedicated classifier trained individually for metric-based statistical MGTD methods, it provides a class prediction, enabling a direct comparison to LLMs-based MGTD classifiers. MGTBench has already accelerated MGTD research, such as (Wu and Xiang, 2023) or (Macko et al., 2023).

However, MGTBench provides a quite complicated way to use custom datasets or to integrate new MGTD methods. Moreover, the used classifier-based (must be trained) evaluation of metric-based statistical detectors does not enable their true zero-shot evaluation (without training, as reported in the corresponding papers). Therefore, a more flexible and configurable benchmarking framework is needed.

3 IMGTB – Integrated MGTD Benchmark Framework

In this section, we introduce the central design principles, IMGTB was built with, as well as its architecture and the functionality of the main components. We use a term *experiment* to denote a single run of the specified detection method on data from the specified dataset.

3.1 Design Principles

The IMGTB framework was designed with several main principles in mind. We consider them important to mention because they encompass what was missing in other similar works and why this tool was developed in the first place.

[P1: Modularity] All the subtasks and responsibilities, such as configuration parsing, data loading, and runnning experiments, were divided and

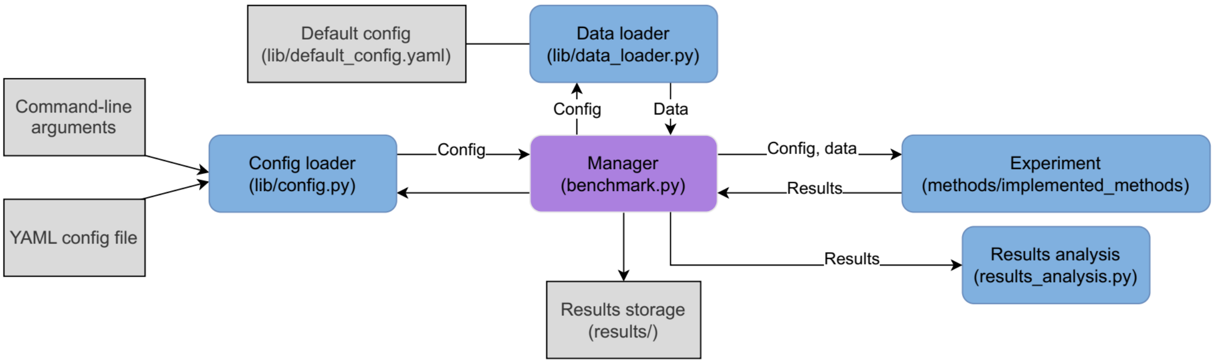

Figure 2: Figure displays an overview of the framework architecture. The Manager acts as the coordinator, interconnecting all other components. In a usual workflow, the Manager requests user-specified configurations from the Configuration Parser. Given the configurations, it requests the train and test datasets from the Data Loader. Furthermore, it forwards the configurations and datasets for evaluation using user-specified detection methods (Experiment component). Finally, it stores the results returned by experiments and calls a Results Analysis component which provides a basic evaluation and visualization of the results.

assigned to their respective modules that only communicate between themselves through a very general interface. Such a decreased inter-module coupling makes the framework very robust and resistant to changes and easy to update, which is useful in order to utilize all the technologies that are yet to be discovered.

[P2: Ease of use] The issue and a main blocker when testing and experimenting with new MGTD methods and new datasets seems to be the need to manually set up and integrate a new method, which often does not work out-of-the-box, to manually parse each dataset and then write one's own analysis tools. This framework was designed to mitigate this issue. Simple experiments can be running in seconds just using the terminal via command-line arguments or, for more complex experiments, using a YAML configuration file. Any dataset or detector can be easily accessed from the Hugging Face Hub without the need to manually download it. The framework also includes many parsing utility functions that enable to load and parse almost any dataset without any need to provide a custom code. Additionally, with built-in analysis tools, it is possible to have basic analysis done right after the experiment has finished.

[P3: Customizability] The structure of input data can vary significantly, detection methods often need different resources, and although we do try to provide utility functions to provide for most of them, it is not possible to cover all such possible cases. Therefore, we have put great emphasis on making the customization of our codebase and extending our functionalities as simple and straightforward as possible.

3.2 Architecture Overview

Figure 2 overviews the main components of the framework architecture, further described in the following subsections.

3.2.1 Manager

The Manager, interconnecting all the other components, serves as the user interface. Its main task is to orchestrate the other components. It calls the data loader, forwards configurations, runs experiments and so on.

3.2.2 Configuration Parser

Configuration Parser provides the functionality to specify configurations directly in the terminal via command-line arguments for quick experiment setup or via a YAML configuration file for more complex experiments. However, command-line arguments offer only a subset of the options the YAML configurations system offers. For convenience, user-specified configurations are always merged with a system default configurations (see *lib/default_config.yaml*). To add a new parameter to the configurations is as easy as adding it to one's YAML configurations file, or to the system default (*lib/default_config.yaml*), no changes to the code itself are needed.

3.2.3 Data Loader

Data loader's main responsibility is to offer functionalities to parse as many different dataset formats and structures as possible. Currently, it is possible to specify column names, labels, a Hugging Face Hub dataset just by providing its identifier, use different subsets, splits, test on machine or human only text data, and much more. In the case that these predefined functionalities would not be sufficient, we try to make it as easy as possible to integrate custom parser functions.

3.2.4 Experiment

Experiment is an abstract class defining a single abstract method *run(data, config)* that runs the experiment on the provided data and given configurations and returns results (ideally in the standardized format). In regards to detectors, the framework offers many already implemented (*methods/implemented_methods*), such as single metric-based methods (e.g., Entropy by Lavergne et al., 2008, or Binoculars by Hans et al., 2024), multi-metric-based methods (e.g., GLTR by Gehrmann et al., 2019, MFD by Wu and Xiang, 2023, or S5 by Spiegel and Macko, 2024), or perturbation-based methods (e.g., DetectGPT by Mitchell et al., 2023, or DetectLLM-NPR by Su et al., 2023). Single metric methods (even perturbation based) can be run with a to-be-trained classifier on top or in zero-shot manner using a predefined or a calibrated classification threshold. To run a Sequence Classification Hugging Face Hub model, only its identifier needs to be specified in the methods configurations as a file path. In addition to running such models directly, they can be fine-tuned using three different configurable processes: full, PEFT (QLoRA based parameter-efficient fine-tuning by Dettmers et al., 2023), or per-language based multilingual fine-tuning (Spiegel and Macko, 2024). Although there are many MGTD methods already implemented in the framework, the true feature of this component is the possibility to quickly implement new custom experiments. By using some of the predefined experiment templates for metric-based or perturbation-based methods, it is possible to implement experiments in just a few lines of code. There is, however, still a possibility to implement a fully custom experiment by implementing the *run()* method from scratch.

3.2.5 Results Analysis

Results analysis can be run either right after a benchmark run, can be specified in the configurations, or later by loading the results from a file. We implement several analysis methods ourselves, such as detection performance (Accuracy, Precision, Recall, F1-score, ROC - receiver operating characteristic), false positives/negatives, or runtime performance. But it is ensured for easy integration of new analysis methods.

4 Case Study

To better illustrate the use of the framework in practice, in this section we showcase a few example use case scenarios. We look at:

> **A.** How to quickly run and evaluate simple experiments using CLI
>
> **B.** How to run complex experiments using YAML configuration files

For a more visual version of this demonstration, see the video[4]. For more detailed and runnable version, see the Jupyter notebook[5].

4.1 Example Scenario A

Let's assume we obtained a completely new never-before-seen dataset of texts generated by one of the latest SOTA large language models. In a similar manner, we could also use existing datasets, even from completely unrelated domains, such as AI-powered text summarization, translation, question answering, or disinformation detection.

Out of curiosity, we'd like to see how the current SOTA detection methods roughly (i.e., default settings) perform on this new data.

Starting from scratch, this would probably take a significant amount of effort to preprocess the data, find the source code of the detectors, integrate the detectors, evaluate and plot the results, as well as considerable knowledge about tools like pandas, numpy or transformers, not to mention the time spent browsing the documentation of said tools.

This all seems a little bit too much. But with our framework we could accomplish the same just by running one CLI command as follows:

```
python benchmark.py --dataset
↪   xzuyn/futurama-alpaca huggingfacehub
↪   machine_only output --methods
↪   roberta-base-openai-detector
↪   Hello-SimpleAI/chatgpt-detector-roberta
↪   andreas122001/roberta-mixed-detector
```

In the command, the option *--dataset* is used for specification of *xzuyn/futurama-alpaca* dataset, available at HuggingFace (see the *huggingfacehub* keyword), which contains only machine-generated

[4] https://www.youtube.com/watch?v=NlHIC4HDQrc
[5] https://colab.research.google.com/drive/15C7kzpnDnx_zqwplCpc949xVJ4Bhdnjl?usp=sharing

texts (see the *machine-only* keyword), and the data field/column to be used for texts being *output*. For the full description of the *dataset* parameters see the GitHub repository[6]. The option *--methods* is followed by identifiers of the methods to be evaluated and compared in the benchmark. If such identifiers are not found in the local implementations of the MGTD methods, the HuggingFace is used as a repository of the models.

When the benchmark run finishes, we are able to find all the results in the latest *results/logs* log entry. It contains a JSON file storing all the benchmark results and the output plots (examples in Figure 3 and 4) of the results analysis component. Using the provided plots, per-detection-method performance is easily comparable.

Regarding Figure 3, only machine-class samples were included in the Scenario A dataset; therefore, the precision of all detectors is 1.0 (i.e., no false positives) and the accuracy is the same as the recall. Based on Figure 4, the last detection method clearly has problems in identifying machine texts from the provided dataset, due to prevalence of false negatives (with a high certainty, based on machine-class probability score).

[6] https://github.com/kinit-sk/IMGTB/tree/main#dataset-parameters

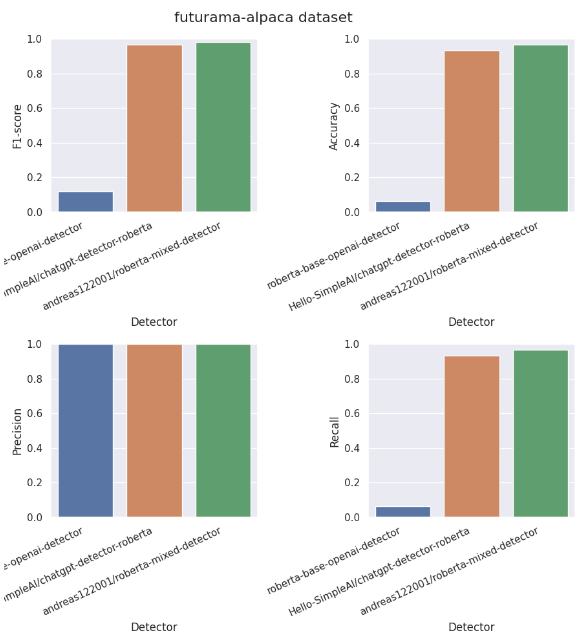

Figure 3: Automatically generated chart for detection-performance metrics analysis.

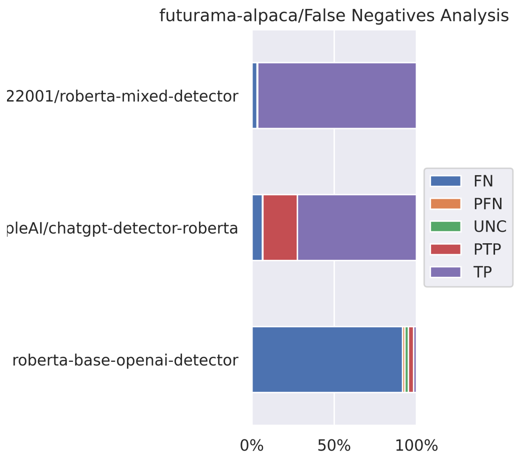

Figure 4: Automatically generated chart for false-negatives analysis (inspired by Weber-Wulff et al., 2023), where FN, PFN, UNC and PTP represent false negatives, potential false negatives, uncertainty, potential true positives and true positives, respectively.

4.2 Example Scenario B

In this scenario, let's assume that we have developed and integrated a new metric-based MGT detection method called *MiracleMetric*. IMGTB implements a number of state-of-the-art detection methods. Implementing new methods is streamlined by the use of template abstract classes that allow fast prototyping of new statistical and fine-tuned methods. To make a complex evaluation on multiple datasets, comparing with multiple different detection methods, and with different parameters, we can design a very compact and readable YAML configuration file.

Firstly, in Figure 5 we specify the data to be used. After that we can specify multiple methods (including our *MiracleMetric*) with different parameters, models, etc. in Figure 6. As opposed to other benchmarks (e.g. MGTBench (He et al., 2023)), IMGTB enables users to specify custom datasets, detection methods and various other parameters by simply creating a configuration file, without the need to modify the codebase. This crucial advantage enables fast prototyping and eliminates unnecessary, repetitive tasks that often hinder researchers in this field.

With this done, the only step keeping us from the results is running the benchmark using these configurations:

```
python benchmark.py
↪ --from_config=example_config.yaml
```

This will output similar results to the previous

```yaml
data:
  global:
    filetype: auto
  list:
  - filepath: WxWx/ChatGPT-Detector-Bias
    filetype: huggingfacehub
    text_field: text
    label_field: kind
    human_label: Human-Written

  - filepath: yaful/DeepfakeTextDetect
    filetype: huggingfacehub
    train_split: test_ood_gpt
    test_split: test_ood_gpt_para
    human_label: "1"
```

Figure 5: Data configurations in YAML format.

```yaml
methods:
  global:
    base_model_name: gpt2-medium
    mask_filling_model_name: t5-large
    DEVICE: cuda
  list:
  - name: MiracleMetric
  - name: MiracleMetric
    clf_algo_for_threshold:
      name: MLPClassifier
      hidden_layer_sizes: [64, 32, 16]
  - name: LoglikelihoodMetric
  - name: LogRankMetric
  - name: EntropyMetric
  - name: DetectLLM_LLR
  - name: EntropyMetric
  - name: roberta-base-openai-detector
```

Figure 6: Methods configurations in YAML format.

example scenario. However, this time we might not be satisfied with the simple automatic analysis provided to us by the framework and we might want to do a more complex and custom-made analysis fitting to the specific needs of our benchmark run. For a demonstration on this exact issue, see the provided Jupyter notebook with the full demo.

5 Discussion

The usefulness of the IMGTB framework has been evaluated in practice by its usage in (Macko et al., 2024b; Spiegel and Macko, 2024; Macko et al., 2024a), where it proved to be valuable especially for its implementation of the statistical detectors.

In comparison to the SOTA MGTD framework called MGTBench (He et al., 2023), IMGTB enables an objective comparison of statistical zero-shot detectors (i.e., without classifier training) by using ROC curve. Further, IMGTB integrates the newest detectors, such as MFD, Binoculars, or S5. It directly enables multiple fine-tuning processes for language models. Most of all, in MGTBench (He et al., 2023), configurations, datasets and detection methods are often hard coded and cannot be easily changed or reconfigured, IMGTB simplifies usage of custom datasets and detectors by supporting plug-in like extension. Faster evaluation is provided by the implemented results analysis and automated comparison charts generation.

To fine-tune a model for binary classification, a more generic SOTA framework, such as Ludwig (Molino et al., 2019), could be used. However, such a framework would not be usable to compare the fine-tuned detectors to statistical detection methods or online services. Therefore, a specialized MGTD framework, such as MGTBench or IMGTB is needed for such a comparison. IMGTB enables such a fine-tuning process by itself and also includes a unique multilingual MGTD specific version of per-language fine-tuning, not available in other frameworks. Similarly, the Evaluate framework[7] could be directly used to compare pre-trained classification models. However, to compare also to statistical and other custom detection methods, a significant effort would be required to implement the custom pipelines, loosing flexibility, configurability (especially concerning the detectors training), and tailor-made analysis and visualization tools in comparison to the proposed IMGTB.

5.1 Extensions & Enhancements Possibilities

There are various limitations and many possible extensions of the current version of the framework which can be targeted to increase its usability even more. Multiple MGTD methods with vastly different evaluation pipelines are not yet compatible with the framework such as Grover by Zellers et al., 2019, FAST by Zhong et al., 2020 or many zero-shot online services (usually paid), available by a custom API (application programming interface), of which only GPTZero[8] is currently supported by the framework. We are continuously working on extension for these additional features.

To speed up the experiments, *bitsandbytes* library has already been utilized for quantized inference and fine-tuning of LLMs for some methods. This can be further extended to be used also for metric computation in metric-based methods. Further speed-ups can be achieved by eliminating redundant tasks (e.g., loading of the same base models

[7] https://huggingface.co/docs/evaluate/en/index

[8] https://gptzero.me/

for multiple methods, calculating the same metrics or generating the same perturbations for multiple methods).

There are also possibilities for significant extension of the framework beyond the current scope. Similarly to detection methods, authorship obfuscation methods (i.e., evading detection) can be integrated into the framework to offer automated evaluation of adversarial robustness of the detection methods in the benchmark. The extension can be also focused to methods for detection of AI content in other modalities (or mixed modalities), such as images, voice or videos, which would make it even more universal.

6 Conclusion

The machine-generated text detection belongs to the key challenges connected with the advancements of large language models for prevention of misuse of high-quality text generation capability. The proposed IMGTB framework unifies the evaluation of the existing detection methods and simplifies comparison of new detection methods to the state-of-the-art. With a plug-and-play testing ability of new methods, research hypotheses can be easily examined. The framework can also be used for evaluation of state-of-the-art detection methods on custom data to identify the best performing one to be further used for some specific application. Automated results analysis and methods comparison also enables less proficient users to interpret the results and make a selection.

The framework reduces unnecessarily redundant work of researchers and enables them to focus their effort towards development of more effective detection methods. This can eventually accelerate the research in machine-generated text detection to catch up with the text generation, currently in the lead.

Ethical Considerations

We believe that there is only a limited possibility of **misuse of our framework**. By easily identifying the most successful detection methods, the focus of malicious actors can be moved towards them in order to find ways to avoid detection. Although the mentioned risk is serious, the benefits of the provided framework mentioned in the introduction surpass such risks. The detection methods are already available, we just provide means to compare their performance.

There are additional potential ethical risks associated with the MGTD in general, such as difficulty to differentiate between malicious and legitimate use of machine-generated texts, potential harm caused by false positives or over-reliance on the results of an automated detection methods. However, these pertain more to the deployment of an MGTD service rather than to the benchmarking framework, and are therefore deemed out of scope of this work.

Acknowledgements

This work was partially supported by the *VIGILANT - Vital IntelliGence to Investigate ILlegAl DisiNformaTion*, a project funded by the European Union under the Horizon Europe, GA No. 101073921, and by *vera.ai - VERification Assisted by Artificial Intelligence*, a project funded by European Union under the Horizon Europe, GA No. 101070093.

References

Guangsheng Bao, Yanbin Zhao, Zhiyang Teng, Linyi Yang, and Yue Zhang. 2023. Fast-DetectGPT: Efficient zero-shot detection of machine-generated text via conditional probability curvature. *arXiv preprint arXiv:2310.05130*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized llms. *arXiv:2305.14314*.

Sebastian Gehrmann, Hendrik Strobelt, and Alexander M. Rush. 2019. GLTR: Statistical detection and visualization of generated text. *arXiv:1906.04043*.

Abhimanyu Hans, Avi Schwarzschild, Valeriia Cherepanova, Hamid Kazemi, Aniruddha Saha, Micah Goldblum, Jonas Geiping, and Tom Goldstein. 2024. Spotting LLMs with Binoculars: Zero-shot detection of machine-generated text. *arXiv:2401.12070*.

Xinlei He, Xinyue Shen, Zeyuan Chen, Michael Backes, and Yang Zhang. 2023. MGTBench: Benchmarking machine-generated text detection. *arXiv preprint arXiv:2303.14822*.

Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. 2023. Challenges and applications of large language models. *arXiv preprint arXiv:2307.10169*.

Thomas Lavergne, Tanguy Urvoy, and François Yvon. 2008. Detecting fake content with relative entropy scoring. In *Proceedings of the 2008 International Conference on Uncovering Plagiarism, Authorship and Social Software Misuse - Volume 377*, PAN'08, page 27–31, Aachen, DEU. CEUR-WS.org.

Yafu Li, Qintong Li, Leyang Cui, Wei Bi, Longyue Wang, Linyi Yang, Shuming Shi, and Yue Zhang. 2023. Deepfake text detection in the wild. *arXiv preprint arXiv:2305.13242*.

Dominik Macko, Jakub Kopal, Robert Moro, and Ivan Srba. 2024a. MultiSocial: Multilingual benchmark of machine-generated text detection of social-media texts. *arXiv:2406.12549*.

Dominik Macko, Robert Moro, Adaku Uchendu, Jason Lucas, Michiharu Yamashita, Matúš Pikuliak, Ivan Srba, Thai Le, Dongwon Lee, Jakub Simko, and Maria Bielikova. 2023. MULTITuDE: Large-scale multilingual machine-generated text detection benchmark. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9960–9987, Singapore. Association for Computational Linguistics.

Dominik Macko, Robert Moro, Adaku Uchendu, Ivan Srba, Jason Samuel Lucas, Michiharu Yamashita, Nafis Irtiza Tripto, Dongwon Lee, Jakub Simko, and Maria Bielikova. 2024b. Authorship obfuscation in multilingual machine-generated text detection. *arXiv:2401.07867*.

Eric Mitchell, Yoonho Lee, Alexander Khazatsky, Christopher D. Manning, and Chelsea Finn. 2023. DetectGPT: Zero-shot machine-generated text detection using probability curvature. *arXiv preprint arXiv:2301.11305*.

Piero Molino, Yaroslav Dudin, and Sai Sumanth Miryala. 2019. Ludwig: a type-based declarative deep learning toolbox. *arXiv:1909.07930*.

Vinu Sankar Sadasivan, Aounon Kumar, Sriram Balasubramanian, Wenxiao Wang, and Soheil Feizi. 2023. Can AI-generated text be reliably detected? *arXiv preprint arXiv:2303.11156*.

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

Michal Spiegel and Dominik Macko. 2024. KInIT at SemEval-2024 task 8: Fine-tuned LLMs for multilingual machine-generated text detection. *arXiv:2402.13671*.

Jinyan Su, Terry Yue Zhuo, Di Wang, and Preslav Nakov. 2023. DetectLLM: Leveraging log rank information for zero-shot detection of machine-generated text. *arXiv preprint arXiv:2306.05540*.

Adaku Uchendu, Zeyu Ma, Thai Le, Rui Zhang, and Dongwon Lee. 2021. TURINGBENCH: A benchmark environment for Turing test in the age of neural text generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2001–2016, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Ivan Vykopal, Matúš Pikuliak, Ivan Srba, Robert Moro, Dominik Macko, and Maria Bielikova. 2023. Disinformation capabilities of large language models. *arXiv preprint arXiv:2311.08838*.

Jan Philip Wahle, Terry Ruas, Frederic Kirstein, and Bela Gipp. 2022. How large language models are transforming machine-paraphrase plagiarism. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 952–963, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Yuxia Wang, Jonibek Mansurov, Petar Ivanov, Jinyan Su, Artem Shelmanov, Akim Tsvigun, Chenxi Whitehouse, Osama Mohammed Afzal, Tarek Mahmoud, Alham Fikri Aji, and Preslav Nakov. 2023. M4: Multi-generator, multi-domain, and multilingual black-box machine-generated text detection. *arXiv:2305.14902*.

Debora Weber-Wulff, Alla Anohina-Naumeca, Sonja Bjelobaba, Tomáš Foltýnek, Jean Guerrero-Dib, Olumide Popoola, Petr Šigut, and Lorna Waddington. 2023. Testing of detection tools for ai-generated text. *International Journal for Educational Integrity*, 19(1).

Laura Weidinger, John Mellor, Maribeth Rauh, Conor Griffin, Jonathan Uesato, Po-Sen Huang, Myra Cheng, Mia Glaese, Borja Balle, Atoosa Kasirzadeh, Zac Kenton, Sasha Brown, Will Hawkins, Tom Stepleton, Courtney Biles, Abeba Birhane, Julia Haas, Laura Rimell, Lisa Anne Hendricks, William Isaac, Sean Legassick, Geoffrey Irving, and Iason Gabriel. 2021. Ethical and social risks of harm from language models. *arXiv preprint arXiv:2112.04359*.

Zhendong Wu and Hui Xiang. 2023. MFD: Multi-feature detection of LLM-generated text.

Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. 2019. Defending against neural fake news. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 9054–9065. Curran Associates, Inc.

Wanjun Zhong, Duyu Tang, Zenan Xu, Ruize Wang, Nan Duan, Ming Zhou, Jiahai Wang, and Jian Yin. 2020. Neural deepfake detection with factual structure of text. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2461–2470, Online. Association for Computational Linguistics.