AIpom at SemEval-2024 Task 8: Detecting AI-produced Outputs in M4

Alexander Shirnin, **Nikita Andreev**
Vladislav Mikhailov, **Ekaterina Artemova**
HSE University, CAIT and Applied AI Institute
University of Oslo, Toloka AI
Correspondence: ashirnin@hse.ru

Abstract

This paper describes AIpom, a system designed to detect a boundary between human-written and machine-generated text (SemEval-2024 Task 8, Subtask C: Human-Machine Mixed Text Detection). We propose a two-stage pipeline combining predictions from an instruction-tuned decoder-only model and encoder-only sequence taggers. AIpom is ranked second on the leaderboard while achieving a Mean Absolute Error of 15.94. Ablation studies confirm the benefits of pipelining encoder and decoder models, particularly in terms of improved performance.

1 Introduction

SemEval-2024 Task 8 (Wang et al., 2024a) focuses on multigenerator, multidomain, and multilingual machine-generated text detection based on the M4 corpus (Wang et al., 2024b). The shared task offers three subtasks, which correspond to standard task formulations in the rapidly developing field of artificial text detection (Jawahar et al., 2020; Uchendu, 2023): (A) classifying if a given text in a particular language is human-written or machine-generated, (B) attributing the author of a given text, and (C) detecting a boundary between human-written and machine-generated text. Developing generalizable solutions to these problems helps mitigate the risks of misusing generative language models (LMs) for malicious purposes (Weidinger et al., 2022) and improve human performance in identifying AI-produced content (Gehrmann et al., 2019).

This paper proposes AIpom[1], a novel method for human-machine mixed text detection (Subtask C). The boundary detection setup aligns with common user scenarios for applying generative LMs in practice, e.g., text continuation, creative writing, and story generation. The standard approach to this task is training a linear classifier or a regression model over encoder representations (Cutler et al., 2021; Dugan et al., 2023). In contrast, AIpom leverages a pipeline of decoder and encoder models to detect machine-generated text, utilizing them sequentially. AIpom takes second out of 33 participating teams on the Subtask C leaderboard by achieving a Mean Absolute Error (MAE) of 15.94 on the official evaluation set. After the official evaluation phase, we develop a better-performing solution with an MAE score of 15.21.

Our ablation studies confirm that using decoder or encoder models individually leads to lower performance. Thus, employing the pipeline of decoder and encoder models proves to be an effective solution. Additionally, these studies highlight domain shift issues, as there is a significant score disparity between the development and official evaluation sets. Future efforts should focus on enhancing the AIpom robustness with respect to the text domain and text generator. The codebase and models are publicly released[2].

2 Background

The M4 corpus consists of human-written and machine-generated texts in six languages (English, Chinese, Russian, Urdu, Indonesian, and Arabic) across various domains, ranging from Wikipedia to academic peer reviews. The generative LMs include the OpenAI models (ChatGPT and text-davinci-003), LLaMA-1 (Touvron et al., 2023), FLAN-T5 (Chung et al., 2022), Cohere, Dolly-v2[3], and BLOOMZ (Muennighoff et al., 2022). The organizers provide 3649, 505, and 11123 dataset instances in Subtask C training, development, and official evaluation sets, respectively.

[1] AIpom is named after a simian pokémon *aipom*, and stands for detecting **AI**-**p**roduced **o**utputs in **M**4.

[2] github.com/25icecreamflavors/AIpom

[3] hf.co/databricks/dolly-v2-12b

Proceedings of the 18th International Workshop on Semantic Evaluation (SemEval-2024), pages 1667–1672

Task Formulation Human-machine mixed text detection requires predicting the index corresponding to the first machine-generated word, as shown below:

- `text:` "👨‍💼We have added a 2+ page 🤖discussion on the experimental results, highlighting the superiority of the ARC-based models and their impact on the field of deep learning."
- `label: 6`

Performance Metric MAE measures the absolute distance between the predicted word and the word where the human-machine transition occurs.

3 AIpom

First, we overview the AIpom pipeline. Next, we detail the fine-tuning procedures for encoder and decoder models.

Overview The AIpom pipeline (see Figure 1) consists of multiple consecutive steps of fine-tuning language models:

(a) The decoder is fine-tuned on the training set to predict the change point from a human-written text to a machine-generated text.
(b) The decoder makes predictions and outputs the source texts with predicted change points.
(c) The first encoder model is fine-tuned on the texts with predicted change points from step (b).
(d) The second encoder model is fine-tuned on the mixture of texts from the training set and the texts with predicted change points from step (b).
(e) Two encoders are used to predict the indices of change points in test texts.
(f) The predicted change points from step (e) are aggregated by averaging.

Decoder The decoder is fine-tuned as follows: the input comprises the prompt and the training text. We experimented with various prompts, including instructing the model to output only the human-written text, the text with an inserted symbol representing the change point, and the machine-generated text alone. Our preliminary experiments suggest that instructing the decoder to output only the machine-generated text yields better results. Therefore, we use this option in subsequent experiments. Table 1 describes the prompt, and Figure 2 illustrates fine-tuning the decoder. The decoder is used in the first step of the AIpom pipeline: we utilize it to generate initial predictions, which are then further processed by two encoders.

```
As an output, write only the machine-generated part of the provided text. Output must start with "Answer: ". Separate tokens by " ". If the whole text is human-written, output "None". Here is the text: example["text"]
```

Table 1: The prompt used for fine-tuning the decoder.

After receiving the predicted text from the decoder, we post-process the original training text and insert a special token <BREAK> directly before the first machine-generated word predicted by the decoder. This allows us to pass the prediction further to the encoder.

Encoder The encoder is fine-tuned to label input texts on a token-wise way. Each token in the human-written segment is labeled with a zero, while each machine-generated token is labeled with one. In our final prediction, we determine the position of the word in which the first "1" label appears, indicating machine-generated text. See Figure 3 for illustration.

The AIpom pipeline involves fine-tuning two encoders. The first encoder is trained on a dataset consisting of texts labeled by the decoder. In contrast, the second encoder is fine-tuned using a dataset that includes both the decoder's predictions and the original source texts from the training set. we receive the predicted change point from each encoder, which we aggregate by averaging.

4 Experiments

Overview We design a series of experiments using two recent language models, a decoder *Mistral-7B-OpenOrca*[4] (Jiang et al., 2023) and an encoder *DeBERTaV3-Large*[5] (He et al., 2023), selected based on their performance on standard NLP benchmarks and computational requirements.

First, we establish a baseline for the decoder model by zero-shot prompting and then compare it to Low-Rank Adaptation (LoRA) tuning (Hu et al., 2021), which yields significantly better results. Second, we look into improving the performance of the

[4] hf.co/Open-Orca/Mistral-7B-OpenOrca
[5] hf.co/microsoft/deberta-v3-large

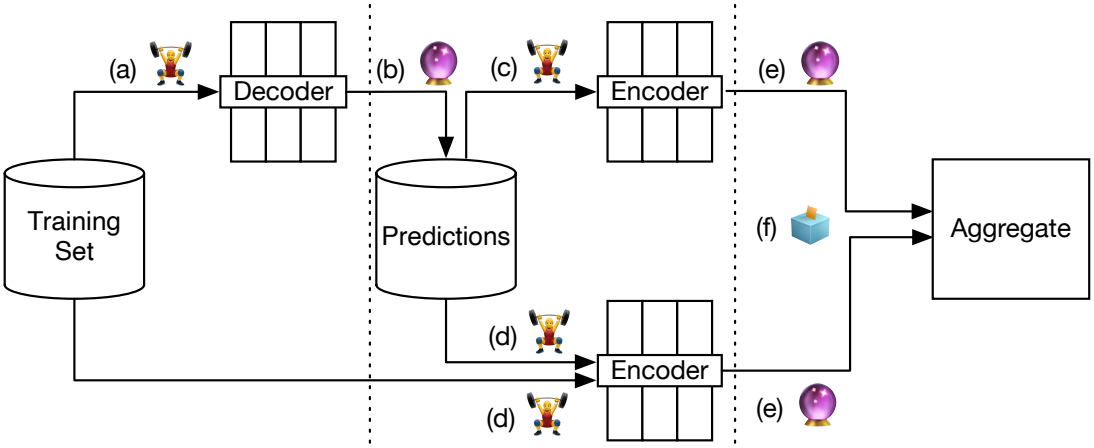

Figure 1: The AIpom pipeline involves fine-tuning decoder and encoder models to predict change points between the human-written and machine-generated text. This process includes fine-tuning the decoder, predicting change points, fine-tuning two encoders, and aggregating predicted change points. 🏋 stands for fine-tuning a language model, 🔮 – predicting with the language model, 📦 – for aggregating the predictions by averaging.

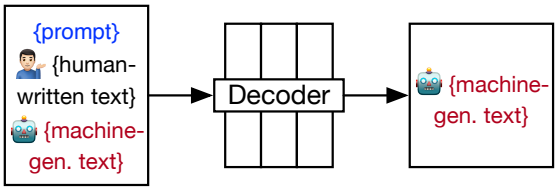

Figure 2: We fine-tune the decoder to output only the machine-written text.

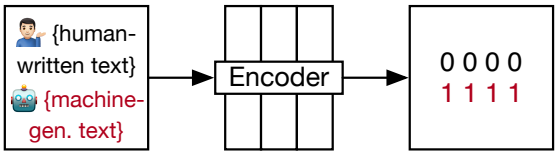

Figure 3: We fine-tune the encoder for token labeling. Human-written tokens are assigned zeros, while machine-generated tokens are assigned ones.

encoder model. We experiment with hyperparameter selection and feeding the encoder model with texts labeled by the decoder model. The combination of the decoder and encoder model outperforms each pipeline component individually.

We only use the development set to evaluate our pipeline and choose our final submission based on the MAE on the development set. In §5, we report the ablation studies results on both development and official test sets[6].

Decoder fine-tuning and inference To fine-tune the Mistral model, we employ LoRA layers tuning with the `SFTTrainer` class from the transformers library (Wolf et al., 2020). The model is fine-tuned on to output machine-generated texts, that is the loss functions are computed only on the model-generated parts. We experimented with learning rates in the range $[1e\text{-}5, 5e\text{-}5]$ with an increment of `1e-5`, and `warmup_ratio` $\in \{0.01, 0.03, 0.05\}$. Based on the results observed on the development set, we select a learning rate of `2e-5`, combined with a `warmup_ratio=0.03` and the `CosineLRScheduler`. For the Parameter-Efficient Fine-Tuning (PEFT) configuration, we adhere to the recommended parameters for Mistral: `rank=32`, `lora_alpha=64`, and `lora_dropout=0.05`. The batch size is set to 4 and the model is fine-tuned for 4 epochs.

We fine-tune the model to start its response with the `"Answer: "` template. This helps improve performance at the inference stage by providing easier-to-clean-up predictions, ensuring they always start the same way. We use the vLLM framework[7] (Kwon et al., 2023) for text generation, with default hyperparameters, the sampling temperature of 1, and `top_p` of 1.

Data labeling with decoder To prepare the training set for the encoder, we split the training set into two folds and perform LoRA tuning on two Mistral models with the same hyperparameters on each fold. Then, each fold is labeled using the decoder fine-tuned on the other fold. This helps us track the decoder's performance and reduce overfitting. During testing, we fine-tune another Mistral model on the entire training set and assess its per-

[6]The shared task organizers have released the gold annotation for the official test set.

[7]`github.com/vllm-project/vllm`

Setup	Model	Fine-tuning setup	<BREAK> in the input	Dev MAE	Test MAE
1.	LoRA Mistral	Training set		2.41	17.00
2.	DeBERTa	Pred. from Mistral	✓	1.74	17.15
3.	DeBERTa	Training set + pred. from Mistral	✓	1.74	**15.21**
4.	AIpom	2. + 3.	✓	**1.68**	15.94
		Ablation experiments			
5.	zero-shot Mistral	Training set		56.51	80.81
6.	DeBERTa	Training set		2.15	19.97
7.	DeBERTa	Training set + pred. from Mistral		1.91	16.49

Table 2: MAE scores on the development and official test sets for different setups and ablation experiments. Setup details include the model used, fine-tuning setup, and presence of <BREAK> in the input data at the inference stage. The top table shows each language model's performance in the AIpom pipeline. The bottom part shows ablation experiments.

formance on the development set. It is worth noting that we apply the post-processing step described in §3, specifically in the "Decoder" paragraph, to the predicted text before passing it to the encoder.

Encoder fine-tuning We build upon the baseline code provided by the task organizers[8], enhancing it to effectively fine-tuning the DeBERTa model. We explore a range of learning rates [1e-5, 5e-5] with an increment of 1e-5 to identify the optimal value for fine-tuning our model. Our final fine-tuning strategy utilizes the Adam optimizer (Kingma and Ba, 2015), with a learning rate of 3e-5 and the default CosineLRScheduler. To ensure consistency across all experiments, we use a maximum sequence length of 1024 for text tokenization, maintain a constant batch size of 64, and limit the maximum number of epochs to 6. To reduce overfitting, we freeze a certain number of bottom DeBERTa layers. Specifically, we experiment with fine-tuning only the top $N \in \{6, 12, 18\}$ layers out of the total 24. Through experiments, we determine that fine-tuning only the top 12 layers produces the best results.

Hardware specification We run experiments on a single GPU TESLA A100 80 GB. Model fine-tuning is conducted using the transformers library. The fine-tuning for DeBERTa requires approximately 3.5 hours to complete, while the inference on the official test set runs within 15 minutes. The LoRA tuning for Mistral lasts approximately 12 hours, with the inference on the official test set taking a few hours. To speed up the prediction phase, we employ the vLLM framework, designed specifically for optimizing the inference. This implementation significantly reduces inference time, with the official test set predictions generated in just 30 minutes.

5 Results

Table 2 provides a detailed comparison of the performance metrics across different models and experimental setups. The key results are:

- Fine-tuning Mistral with LoRA tuning (setup 1) on the training set outperforms zero-shot prompting (setup 5) by a wide margin.
- Fine-tuning DeBERTa using the Mistral predictions (setup 2) leads to higher results than fine-tuning DeBERTa on the training set (setup 6).
- Adding a <BREAK> token to the input at the inference stage improves the performance of the DeBERTa model (setup 3 vs. setup 7).
- Averaging predictions of two DeBERTa models (setup 4) leads to the best results on the development set.

The overall best results on the official test set are achieved with setup 3, where the DeBERTa model is fine-tuned on a dataset consisting of both the training set and predictions from the Mistral model, and the <BREAK> token is added to the input at the inference stage.[9]

Decoder vs. encoder Fine-tuned encoder models exhibit inferior performance compared to LoRA-tuned decoder. Specifically, the decoder struggles to comprehend the task in a zero-shot setting, evidenced by a high MAE of 80.81 on the official test set. However, with LoRA tuning, the decoder achieves a significantly lower MAE of 17.00, outperforming the single encoder model's MAE of

[8] github.com/mbzuai-nlp/SemEval2024-task8

[9] These results are achieved after the shared task submission deadline and hence not submitted for the official evaluation stage.

19.97. The encoder models are adaptable and integrate diverse inputs, including prompts and predictions from additional decoder models.

Benefits of pipelining We hypothesize that encoder models benefit from integrating inputs from a decoder. Our pipeline yields the final MAE of 15.94 while fine-tuning only the single encoder model results in a higher MAE score of 19.97.

Robustness While averaging predictions helps improve the overall performance, we find that the AIPom's robustness has room for improvement. In particular, we observe the performance decrease when comparing the results on the development and official test sets. Setup 3, with an MAE of 1.74 on the development set, performs better than an MAE of 15.21 on the official test set. At the same time, Setup 4 (our final submission) achieves a slightly better development MAE but a worse MAE on the official test set. Setup 3 involves finetuning on a mixture of data, showing how using more data can boost the performance and improve the robustness, especially when the dataset is small. We leave improving the out-of-domain robustness for future work.

6 Conclusion

This paper presents the AIpom system submitted to SemEval-2024 Task 8. Our solution achieves 2nd place out of 33 participating teams in Subtask C. We introduce a novel method that utilizes a pipeline of decoder and encoder models. The advantage of this approach is that the models are exposed to both the original data and the predictions from previous steps. We believe this approach holds significant potential, as it allows for creating pipelines comprising various models, mimicking the transfer of learned knowledge. We plan to further improve our system by exploring different combinations of models and longer pipelines. Additionally, we aim to enhance the system's robustness to handle domain-shift scenarios.

Acknowledgements

AS's work results from a research project implemented in the Basic Research Program at the National Research University Higher School of Economics (HSE University). We acknowledge the computational resources of HSE University's HPC facilities.

References

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling Instruction-Finetuned Language Models. *arXiv preprint arXiv:2210.11416*.

Joseph Cutler, Liam Dugan, Shreya Havaldar, and Adam Stein. 2021. Automatic Detection of Hybrid Human-Machine Text Boundaries.

Liam Dugan, Daphne Ippolito, Arun Kirubarajan, Sherry Shi, and Chris Callison-Burch. 2023. Real or Fake Text?: Investigating Human Ability to Detect Boundaries between Human-written and Machine-Generated Text. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12763–12771.

Sebastian Gehrmann, Hendrik Strobelt, and Alexander Rush. 2019. GLTR: Statistical detection and visualization of generated text. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 111–116, Florence, Italy. Association for Computational Linguistics.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2023. DeBERTav3: Improving deBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing. In *The Eleventh International Conference on Learning Representations*.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Ganesh Jawahar, Muhammad Abdul-Mageed, and Laks Lakshmanan, V.S. 2020. Automatic detection of machine generated text: A critical survey. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2296–2309, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Niklas Muennighoff, Thomas Wang, Lintang Sutawika, Adam Roberts, Stella Biderman, Teven Le Scao, M Saiful Bari, Sheng Shen, Zheng-Xin Yong, Hailey Schoelkopf, et al. 2022. Crosslingual Generalization through Multitask Finetuning. *arXiv preprint arXiv:2211.01786*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models.

Adaku Uchendu. 2023. *Reverse Turing Test in the Age of Deepfake Texts*. Ph.D. thesis, The Pennsylvania State University.

Yuxia Wang, Jonibek Mansurov, Petar Ivanov, Jinyan Su, Artem Shelmanov, Akim Tsvigun, Chenxi Whitehouse, Osama Mohammed Afzal, Tarek Mahmoud, Giovanni Puccetti, et al. 2024a. Semeval-2024 task 8: Multigenerator, multidomain, and multilingual black-box machine-generated text detection. In *Proceedings of the 18th International Workshop on Semantic Evaluation, SemEval*.

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. 2024b. M4: Multi-generator, multi-domain, and multilingual black-box machine-generated text detection. In *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. Association for Computational Linguistics.

Laura Weidinger, Jonathan Uesato, Maribeth Rauh, Conor Griffin, Po-Sen Huang, John Mellor, Amelia Glaese, Myra Cheng, Borja Balle, Atoosa Kasirzadeh, et al. 2022. Taxonomy of Risks Posed by Language Models. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, pages 214–229.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.