TeleAI at SemEval-2026 Task 13: Data-Centric Full-Parameter Fine-Tuning with Multi-Level Ensembling for Generated Code Detection

Shiquan Wang[1†], Ruiyu Fang[1†], Shuangyong Song[1], Yongxiang Li[1], Xuelong Li[2*]

[1]Xingchen AGI Lab, China Telecom Artificial Intelligence Technology (Beijing) Co., Ltd
[2]Institute of Artificial Intelligence (TeleAI), China Telecom
wangsq23,fangry,songshy,liyx25@chinatelecom.cn, xuelong_li@ieee.org

Abstract

This paper presents our top-ranking system for SemEval-2026 Task 13 on code generation detection under multi-lingual and distribution-shift settings. Our approach achieved 1st place in Subtasks A and B, and 2nd place in Subtask C in the official evaluation.

Our framework integrates data-centric analysis, full-parameter model adaptation, and multi-level ensemble learning. We first analyze label and length distributions and apply repeated oversampling to address class imbalance. We then optimize prompts in a data-driven manner to improve inference stability. Based on Qwen3-30B-A3B-Instruct, we conduct full-parameter fine-tuning with diverse training configurations and integrate multiple checkpoints using soft voting, hard voting, logits-based voting, and LightGBM stacking.

Experimental results demonstrate substantial improvements over zero-shot baselines and consistent gains from ensemble strategies, validating the effectiveness of systematic adaptation and ensembling for robust code generation detection.

1 Introduction

In recent years, large language models (LLMs) have not only achieved remarkable performance in natural language understanding and generation(He et al., 2024; Li et al., 2024b; Wang et al., 2024; Li et al., 2024a; Wang et al., 2025b; Liu et al., 2025), but have also demonstrated strong capabilities in code generation. Modern generative models can produce syntactically correct and semantically meaningful code across multiple programming languages, which has been shown to assist in software development, automated debugging, and programming education. However, this rapid progress also brings new challenges: distinguishing between human-written and machine-generated code is becoming increasingly difficult, yet critically important for software quality assurance, academic integrity, intellectual property protection, and secure code analysis(Yang et al., 2025).

To address this emerging problem, SemEval-2026 Task 13: Detecting Machine-Generated Code was introduced as a shared evaluation benchmark focusing on the forensic analysis of code authorship (Orel et al., 2026). This task challenges the community to design systems that identify whether a given code snippet was authored by humans or by generative models, and to attribute code to specific generation sources when possible. The competition consists of three subtasks:

Subtask A — Binary classification of code as human-written or machine-generated, with test distributions intentionally exhibiting out-of-distribution shifts to test generalization robustness.

Subtask B — Multi-class classification to identify which particular Large Language Model produced a code sample, exploring stylistic and structural patterns across generators.

Subtask C — Four-way hybrid authorship classification that not only distinguishes human and machine code, but also identifies hybrid cases where human code has been edited by an LLM or where an LLM intentionally tries to evade detection.

These subtasks together form a comprehensive evaluation of code provenance detection, requiring systems to balance accuracy, robustness to unseen languages and contexts, and nuanced understanding of hybrid generation patterns.

We participated in this task using a framework centered on systematic data processing, full-parameter model fine-tuning, and multi-stage post-processing and ensemble strategies, designed to handle the challenge of diverse programming languages and generator characteristics. By fully fine-tuning large pretrained models on task-specific code representations, we enable deep adaptation

[†] Equal contribution.
[*] Corresponding author.

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 263–269

to the nuances of machine vs. human code patterns. Through targeted cleaning, augmentation, and class imbalance mitigation in preprocessing, we further improve downstream classifier stability. At inference, we combine multiple model predictions and inference views via ensemble mechanisms to synthesize complementary strengths. Our system achieved first place in Subtask A and Subtask B, and second place in Subtask C, demonstrating the effectiveness of this integrated approach in large-scale code authorship detection benchmarks.

2 Related Work

With the rapid advancement of large language models (LLMs), the ability of these models to generate high-quality source code has significantly improved. As a consequence, detecting LLM-generated code has emerged as an important and challenging research problem at the intersection of natural language processing and software engineering. Compared to natural language text, source code exhibits stronger syntactic constraints, explicit structural patterns, and language-specific semantics, which pose both opportunities and challenges for detection methods. Existing approaches related to LLM-generated content detection can be broadly categorized into the following directions.

2.1 Watermarking Techniques

Watermarking techniques aim to embed imperceptible signals into content generated by LLMs, enabling downstream identification of its origin. These approaches include data-driven, model-driven, and post-processing watermarking methods. Data-driven methods introduce controlled perturbations during training so that generated outputs exhibit identifiable statistical regularities (Gu et al., 2022). Model-driven approaches directly modify model parameters or decoding strategies to inject generation signatures into the output (Kirchenbauer et al., 2023; Liu et al.). Post-processing methods apply detectable patterns after generation (Yoo et al., 2023).

While watermarking has shown promise for tracing the provenance of LLM-generated text, its applicability to code detection remains limited. Code generation quality is highly sensitive to syntactic correctness and formatting, and watermarking signals may be easily disrupted by minor edits, refactoring, or formatting changes. Moreover, watermarking requires cooperation from the code generator, which is often unrealistic in adversarial or open-world detection scenarios such as those considered in SemEval tasks.

2.2 Statistical Methods

Statistical methods identify LLM-generated content by analyzing surface-level or distributional properties of the output. Feature-based approaches rely on handcrafted metrics such as lexical diversity, token frequency distributions, or structural complexity (Arase and Zhou, 2013; Hamed and Wu, 2023). White-box statistical methods exploit internal generation probabilities or model-specific information to detect anomalies indicative of machine generation (Su et al., 2023; Wu et al., 2023). In contrast, black-box methods operate solely on the generated output without requiring access to the underlying model (Yu et al., 2023).

When applied to code detection, these methods typically examine token repetition patterns, indentation styles, abstract syntax tree statistics, or language-specific constructs. Although statistical approaches are computationally efficient and interpretable, they often struggle to generalize across programming languages, coding styles, and unseen generation models. Their performance also degrades when code is partially edited by humans, a scenario explicitly addressed in hybrid authorship settings.

2.3 Neural Network-Based Approaches

Neural network-based approaches leverage deep learning models to automatically learn discriminative representations for detecting LLM-generated content (He et al., 2024). Feature-based classifiers extract embeddings from pretrained models and feed them into downstream classifiers (Aich et al., 2022; Xiong et al., 2024; Song et al., 2017; Song and Meng, 2015; Song et al., 2020a,b). Pretrained language model classifiers fine-tune large encoders directly for detection tasks (Fagni et al., 2021; Lu et al., 2022; Wang et al., 2025a). More recently, several studies explore using LLMs themselves as detectors, exploiting their strong language and reasoning capabilities to identify machine-generated patterns (Bhattacharjee and Liu, 2023; Li et al., 2024b; Rodriguez et al., 2022; Shi et al., 2026; Li et al., 2025; Xiong et al., 2025b; Zhao et al., 2025; Xing et al., 2025; Xiong et al., 2025a).

For code detection tasks, pretrained code models and large language models fine-tuned on code corpora have become the dominant approach. These

models can capture both syntactic and semantic properties of source code, making them particularly suitable for distinguishing human-written code from LLM-generated code. However, neural approaches generally require substantial labeled data and computational resources, and their robustness to out-of-distribution generators and hybrid editing scenarios remains an open challenge.

2.4 Human-Aided Approaches

Human-aided detection methods incorporate human expertise to assist in identifying LLM-generated content. Human-in-the-Loop frameworks design tasks that encourage annotators to evaluate coherence, correctness, or domain-specific knowledge (Ippolito et al., 2020). Other approaches rely on crowdsourcing to aggregate multiple independent judgments into a consensus decision (Dugan et al., 2023; Weng et al., 2023).

In the context of code detection, human expertise can be particularly valuable for assessing logical correctness, coding conventions, and subtle stylistic cues. However, such approaches are inherently limited by scalability concerns.

3 Method

This section presents our overall approach for SemEval 2026 Task 13. The proposed framework consists of four main components: data analysis and augmentation, prompt optimization, full-parameter model fine-tuning, and multi-strategy ensemble voting. By jointly optimizing data distribution, inference behavior, and prediction aggregation, our method achieves strong performance and robustness across multiple subtasks.

3.1 Data Analysis and Data Augmentation

Before model training, we conduct a systematic analysis of the official training data, focusing on label distribution, sample quantity distribution, and code length distribution. The analysis results are used to guide subsequent data augmentation and training strategies.

3.1.1 Label and Quantity Distribution

Our analysis reveals that the training data exhibits significant class imbalance across different subtasks, with some categories having substantially fewer samples than the dominant classes. Directly training on the original data may cause the model to bias toward high-frequency labels, leading to unstable performance on minority classes, particularly in fine-grained classification and hybrid authorship scenarios.

To alleviate this issue, we adopt a repeated oversampling strategy for minority classes. Without introducing synthetic samples or external data, low-frequency samples are duplicated to increase their effective presence during training, enabling the model to better learn discriminative features across different categories.

3.1.2 Code Length Distribution

In addition to label imbalance, we analyze the length distribution of code samples. The results show a wide range of code lengths, from short code snippets and individual functions to long program implementations. Such variability may affect the model's ability to consistently capture contextual information and may negatively impact training stability.

To address this issue, we apply unified truncation and padding strategies during input construction, constraining extreme-length samples and reducing distributional discrepancies among training instances.

3.2 Prompt Optimization

Beyond supervised fine-tuning, we introduce an iterative, data-driven prompt optimization strategy to improve model output stability and task alignment during inference. This process is conducted without modifying model parameters and serves as an effective complement to model training.

Specifically, we design and optimize prompt templates separately for each subtask. Initial prompts are constructed based on task definitions, label semantics, and output format constraints. We further enhance prompt diversity through contextual augmentation, such as incorporating explicit task instructions, category descriptions, and prediction constraints.

An iterative evaluation procedure is then applied to assess different prompt candidates on the validation set. Combined with pruning and early stopping mechanisms, poorly performing or redundant prompts are gradually eliminated, resulting in the selection of optimal prompt configurations. This strategy effectively controls the search space while preventing overfitting to specific prompt patterns.

Experimental results indicate that prompt optimization significantly improves the stability and task relevance of model predictions across multiple subtasks.

3.3 Model Training

We select Qwen3-30B-A3B-Instruct as the backbone model and perform full-parameter fine-tuning for all subtasks.

Compared to parameter-efficient tuning methods, full fine-tuning allows the model to more thoroughly adapt its internal representations to the diverse generation patterns, programming styles, and hybrid authorship characteristics present in the code detection task. To encourage model diversity and improve ensemble effectiveness, we adopt different training hyperparameter settings and save multiple checkpoints during training.

Each subtask is fine-tuned independently to ensure that the model sufficiently captures task-specific decision boundaries. Cross-entropy loss is used as the training objective, and the repeated oversampling strategy described earlier is applied to mitigate class imbalance.

3.4 Ensemble and Voting Strategies

To further enhance prediction robustness, we employ multiple ensemble and voting strategies at inference time, aggregating outputs from models trained with different configurations and checkpoints.

3.4.1 Soft Voting

Soft voting aggregates the predicted class probability distributions from multiple models by averaging or weighted averaging, and selects the class with the highest final probability. This approach effectively leverages confidence information and provides stable performance across subtasks.

3.4.2 Hard Voting

Hard voting performs majority voting over discrete model predictions, selecting the most frequently predicted label as the final output. Although probability information is ignored, this method can significantly improve robustness when model predictions are complementary.

3.4.3 LightGBM-Based Stacking

Finally, we introduce a LightGBM-based stacking strategy as a second-level ensemble method. Validation-set predictions from multiple models, including probability vectors or logits, are used as input features to train a LightGBM classifier. The stacking model learns to assign adaptive weights to different base models and produces final predictions. This learning-based fusion strategy consistently improves performance across subtasks.

4 Experiments

In this section, we present the experimental settings and results of our system on SemEval 2026 Task 13. We first report the official test set performance, followed by detailed analysis of model fine-tuning and ensemble strategies on development and test sets.

4.1 Evaluation Metric

All subtasks are evaluated using Macro-F1 as the official metric. Macro-F1 computes the unweighted average of F1 scores across categories, making it particularly suitable for imbalanced classification scenarios. Given the noticeable class imbalance in Subtasks B and C, Macro-F1 provides a fair assessment of model performance across both majority and minority classes.

Table 1: Official Test-B Results for SemEval-2026 Task 13.

Subtask	Macro-F1	Rank
Subtask A	0.99708	1st
Subtask B	0.50819	1st
Subtask C	0.73332	2nd

4.2 Official Results

Table 1 presents our official test set performance. Our system achieved first place in Subtask A and Subtask B, and second place in Subtask C.

The results demonstrate the effectiveness and robustness of our approach across different task formulations, including binary classification, multi-class classification, and hybrid authorship detection.

4.3 Impact of Full Fine-Tuning

To analyze the contribution of supervised fine-tuning, we compare the performance of the base model (Qwen3-30B-A3B-Instruct), the fully fine-tuned model, and the ensemble-enhanced system.

Subtask A As shown in Table 2, the base model achieves only 0.4185 Macro-F1 on the test-A set, indicating that zero-shot performance is insufficient for reliable code authorship detection. After full-parameter fine-tuning, performance dramatically improves to 0.9476, demonstrating that supervised

Table 2: Performance comparison on Subtask A.

Method	Macro-F1
Qwen3-30B-A3B	0.4185
+ finetune	0.9476
+ ensemble	0.9956

adaptation is essential for aligning the model with task-specific decision boundaries. Further applying ensemble strategies increases performance to 0.9956, showing additional gains from prediction aggregation.

Table 3: Performance comparison on Subtask B.

Method	Macro-F1
Qwen3-30B-A3B	0.0859
+ finetune	0.5146
+ ensemble	0.5236

Subtask B Table 3 reports results on the test-A set. The base model achieves only 0.0859 Macro-F1, reflecting the difficulty of multi-class generator attribution without supervision. After full fine-tuning, performance increases substantially to 0.5146. The ensemble strategy further improves the score to 0.5236, indicating that multi-checkpoint fusion helps capture complementary discriminative patterns across generators.

Table 4: Performance comparison on Subtask C.

Method	Macro-F1
Qwen3-30B-A3B	0.1475
+ finetune	0.8959
+ ensemble	0.9100

Subtask C Subtask C is the most challenging setting due to hybrid authorship scenarios. As shown in Table 4, the base model obtains 0.1475 Macro-F1 on the test-A set. Full fine-tuning leads to a significant improvement to 0.8959, confirming the necessity of task-specific parameter adaptation. The ensemble method further boosts performance to 0.9100, highlighting the effectiveness of multi-model aggregation under complex classification settings.

4.4 Effect of Ensemble Strategies

Across all subtasks, ensemble methods consistently improve performance over single fine-tuned checkpoints. The improvements, although smaller than those from fine-tuning, are stable and meaningful.

The ensemble gains can be attributed to the diversity introduced by: (1) different training hyperparameter configurations; (2) multiple saved checkpoints; (3) complementary decision boundaries learned during optimization.

By aggregating predictions through soft voting, hard voting, logits-based fusion, and LightGBM-based stacking, the final system effectively reduces variance and improves robustness, especially in challenging multi-class and hybrid detection scenarios.

4.5 Discussion

The results highlight three points. First, full-parameter fine-tuning drives the largest gains; zero-shot or instruction-following alone is inadequate. Second, ensembles yield consistent improvements, especially on Subtask C with hybrid authorship. Third, combining data balancing, prompt optimization, and model ensembling improves generalization and leads to top-ranked performance, validating our framework for code generation detection.

5 Conclusion

This paper presents our solution to SemEval-2026 Task 13 under a multi-lingual and distribution-shift setting. We build a systematic framework consisting of data analysis with repeated oversampling for class imbalance, iterative prompt optimization without parameter modification, full-parameter fine-tuning of Qwen3-30B-A3B-Instruct, and multi-strategy ensemble learning.

Experimental results show substantial improvements over the zero-shot baseline across subtasks. In particular, full-parameter adaptation significantly enhances task alignment, while ensemble strategies further improve robustness and stability.

Overall, our approach demonstrates that combining data-centric preprocessing, controlled prompt optimization, and diversified model ensembling is highly effective for large language model-based code generation detection.

References

Ankit Aich, Souvik Bhattacharya, and Natalie Parde. 2022. Demystifying neural fake news via linguis-

tic feature-based interpretation. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 6586–6599.

Yuki Arase and Ming Zhou. 2013. Machine translation detection from monolingual web-text. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1597–1607.

Amrita Bhattacharjee and Huan Liu. 2023. Fighting fire with fire: Can chatgpt detect ai-generated text? *arXiv preprint arXiv:2308.01284*.

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.

Tiziano Fagni, Fabrizio Falchi, Margherita Gambini, Antonio Martella, and Maurizio Tesconi. 2021. Tweepfake: About detecting deepfake tweets. *Plos one*, 16(5):e0251415.

Chenxi Gu, Chengsong Huang, Xiaoqing Zheng, Kai-Wei Chang, and Cho-Jui Hsieh. 2022. Watermarking pre-trained language models with backdooring. *arXiv preprint arXiv:2210.07543*.

Ahmed Abdeen Hamed and Xindong Wu. 2023. Improving detection of chatgpt-generated fake science using real publication text: Introducing xfakebibs a supervised-learning network algorithm.

Zhongjiang He, Zihan Wang, Xinzhang Liu, Shixuan Liu, Yitong Yao, Yuyao Huang, Xuelong Li, Yongxiang Li, Zhonghao Che, Zhaoxi Zhang, and 1 others. 2024. Telechat technical report. *arXiv preprint arXiv:2401.03804*.

Daphne Ippolito, Daniel Duckworth, Chris Callison-Burch, and Douglas Eck. 2020. Automatic detection of generated text is easiest when humans are fooled. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1808–1822.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. 2023. A watermark for large language models. In *International Conference on Machine Learning*, pages 17061–17084. PMLR.

Xiang Li, Yiqun Yao, Xin Jiang, Xuezhi Fang, Chao Wang, Xinzhang Liu, Zihan Wang, Yu Zhao, Xin Wang, Yuyao Huang, and 1 others. 2024a. 52b to 1t: Lessons learned via tele-flm series. *arXiv preprint arXiv:2407.02783*.

Xiang Li, Yiqun Yao, Xin Jiang, Xuezhi Fang, Chao Wang, Xinzhang Liu, Zihan Wang, Yu Zhao, Xin Wang, Yuyao Huang, and 1 others. 2024b. Tele-flm technical report. *arXiv preprint arXiv:2404.16645*.

Zhongqiu Li, Shiquan Wang, Ruiyu Fang, Mengjiao Bao, Zhenhe Wu, Shuangyong Song, Yongxiang Li, and Zhongjiang He. 2025. Mr-uie: multi-perspective reasoning with reinforcement learning for universal information extraction. *Vicinagearth*, 2(1):17.

Aiwei Liu, Leyi Pan, Xuming Hu, Shiao Meng, and Lijie Wen. A semantic invariant robust watermark for large language models, 2024. *URL https://arxiv.org/abs/2310.06356*.

Xinzhang Liu, Chao Wang, Zhihao Yang, Zhuo Jiang, Xuncheng Zhao, Haoran Wang, Lei Li, Dongdong He, Luobin Liu, Kaizhe Yuan, and 1 others. 2025. Training report of telechat3-moe. *arXiv preprint arXiv:2512.24157*.

Yaojie Lu, Qing Liu, Dai Dai, Xinyan Xiao, Hongyu Lin, Xianpei Han, Le Sun, and Hua Wu. 2022. Unified structure generation for universal information extraction. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5755–5772.

Daniil Orel, Dilshod Azizov, Indraneil Paul, Yuxia Wang, Iryna Gurevych, and Preslav Nakov. 2026. SemEval-2026 task 13: Detecting machine-generated code with multiple programming languages, generators, and application scenarios. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Juan Diego Rodriguez, Todd Hay, David Gros, Zain Shamsi, and Ravi Srinivasan. 2022. Cross-domain detection of gpt-2-generated technical text. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1213–1233.

Lingling Shi, Haoyu Jin, Ruiyu Fang, Shuangyong Song, Jinsong Su, Yongxiang Li, and Xuelong Li. 2026. Cemtcontrollable element-oriented machine translation via structured linguistic reasoning. In *Findings of the association for computational linguistics ACL 2026*.

Shuangyong Song, Haiqing Chen, and Zhiwei Shi. 2017. Intention classification of user queries in intelligent customer service system. In *2017 International Conference on Asian Language Processing (IALP)*, pages 83–86. IEEE.

Shuangyong Song and Yao Meng. 2015. Classifying and ranking microblogging hashtags with news categories. In *2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS)*, pages 540–541. IEEE.

Shuangyong Song, Chao Wang, Haiqing Chen, and Huan Chen. 2020a. Tcnn: Triple convolutional neural network models for retrieval-based question answering system in e-commerce. In *Companion Proceedings of the Web Conference 2020*, pages 844–845.

Shuangyong Song, Chao Wang, Siyang Liu, Haiqing Chen, Huan Chen, and Hongyun Bao. 2020b. Sentiment analysis technologies in alime—an intelligent assistant for e-commerce. *International Journal of Asian Language Processing*, 30(04):2050016.

Jinyan Su, Terry Zhuo, Di Wang, and Preslav Nakov. 2023. Detectllm: Leveraging log rank information for zero-shot detection of machine-generated text. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 12395–12412.

Shiquan Wang, Ruiyu Fang, Mengxiang Li, Zhongjiang He, and Shuangyong Song. 2025a. When less is more: Minimal prompts with lora for llm text detection. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 275–283. Springer.

Zihan Wang, Xinzhang Liu, Yitong Yao, Chao Wang, Yu Zhao, Zhihao Yang, Wenmin Deng, Kaipeng Jia, Jiaxin Peng, Yuyao Huang, and 1 others. 2025b. Technical report of telechat2, telechat2. 5 and t1. *arXiv preprint arXiv:2507.18013*.

Zihan Wang, Yitong Yao, Li Mengxiang, Zhongjiang He, Chao Wang, Shuangyong Song, and 1 others. 2024. Telechat: An open-source billingual large language model. In *Proceedings of the 10th SIGHAN Workshop on Chinese Language Processing (SIGHAN-10)*, pages 10–20.

Luoxuan Weng, Minfeng Zhu, Kam Kwai Wong, Shi Liu, Jiashun Sun, Hang Zhu, Dongming Han, and Wei Chen. 2023. Towards an understanding and explanation for mixed-initiative artificial scientific text detection. *arXiv preprint arXiv:2304.05011*.

Kangxi Wu, Liang Pang, Huawei Shen, Xueqi Cheng, and Tat-Seng Chua. 2023. Llmdet: A third party large language models generated text detection tool. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2113–2133.

Hongrui Xing, Xinzhang Liu, Zhuo Jiang, Zhihao Yang, Yitong Yao, Zihan Wang, Wenmin Deng, Chao Wang, Shuangyong Song, Wang Yang, and 1 others. 2025. Llmsr@ xllm25: A language model-based pipeline for structured reasoning data construction. In *Proceedings of the 1st Joint Workshop on Large Language Models and Structure Modeling (XLLM 2025)*, pages 342–350.

Sishi Xiong, Mengxiang Li, Dakai Wang, Yu Zhao, Jie Zhang, Changzai Pan, Haowei He, Xiangyu Li, Wenhan Chang, Zhongjiang He, and 1 others. 2025a. Teleai at semeval-2025 task 8: Advancing table reasoning framework with large language models. In *Proceedings of the 19th International Workshop on Semantic Evaluation (SemEval-2025)*, pages 1828–1841.

Sishi Xiong, Dakai Wang, Yu Zhao, Jie Zhang, Changzai Pan, Haowei He, Xiangyu Li, Wenhan Chang, Zhongjiang He, Shuangyong Song, and 1 others. 2025b. Tablereasoner: Advancing table reasoning framework with large language models. *arXiv preprint arXiv:2507.08046*.

Sishi Xiong, Yu Zhao, Jie Zhang, Li Mengxiang, Zhongjiang He, Xuelong Li, and Shuangyong Song. 2024. Dual prompt tuning based contrastive learning for hierarchical text classification. In *Findings of the association for computational linguistics ACL 2024*, pages 12146–12158.

Jian Yang, Xianglong Liu, Weifeng Lv, Ken Deng, Shawn Guo, Lin Jing, Yizhi Li, Shark Liu, Xianzhen Luo, Yuyu Luo, and 1 others. 2025. From code foundation models to agents and applications: A comprehensive survey and practical guide to code intelligence. *arXiv preprint arXiv:2511.18538*.

KiYoon Yoo, Wonhyuk Ahn, Jiho Jang, and Nojun Kwak. 2023. Robust multi-bit natural language watermarking through invariant features. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2092–2115.

Xiao Yu, Yuang Qi, Kejiang Chen, Guoqiang Chen, Xi Yang, Pengyuan Zhu, Weiming Zhang, and Nenghai Yu. 2023. Gpt paternity test: Gpt generated text detection with gpt genetic inheritance. *CoRR*.

Deji Zhao, Donghong Han, Jia Wu, Zhongjiang He, Bo Ning, Ye Yuan, Yongxiang Li, Chao Wang, and Shuangyong Song. 2025. Enhancing math reasoning ability of large language models via computation logic graphs. *Knowledge-Based Systems*, 325:113905.