Journal of Industrial Engineering and Applied Science
Journal Home: jieas.suaspress.org | CODEN: JIEAAE
Vol. 2, No. 5, 2024 | ISSN 3005-6071 (Print) | ISSN 3005-608X (Online)

# Style-Adaptive Optimization in Software Development Using MOE Structure

**ZHANG, Dutao [1*] LI, Boyi [1]**

[1] ITMO University, Russia

**** ZHANG, Dutao is the corresponding author, E-mail: zh19980811@gmail.com***

**Abstract:** The following literature review and research is devoted to Models of human-AI interaction in software development. The following literature review and research focuses on human-computer interaction modeling in software development. It focuses on the problems of AI's inability to reasonably recognize the user's style, and its inability to complete large and complex AI projects in human-computer AI interaction applications.

**Keywords:** LLM, Machine Learning, MOE, Random Forest, Ensemble Model.

**DOI:** https://doi.org/10.5281/zenodo.13845335 **ARK:** https://n2t.net/ark:/40704/JIEAS.v2n5a06

## 1 INROTRODUCTION

The aim of the practical training is to create a comprehensive review of the chosen problem. Thus, the goals are:

Find and analyze articles, books and researches on the practice topic;

Choose a baseline, implement several experiments using baseline algorithm;

Propose their own modeling framework based on experiments as theoretical support and discuss implementation options, economics and practical implication.

## 2 BACKGROUND AND PROBELM

### 2.1 BACKGROUND

Today is an era of rapid AI development, and with the times, more and more people are starting to use AI mod for work, study and life.AI has been an omnipresent theme over the past year, pushing boundaries and redefining several industries in a short space of time. An astounding number of over 24 billion visits were generated from September 2022 to August 2023 by just the top 50 AI tools, with an average monthly growth of 236.3 million visits. Out of this massive number, ChatGPT alone accounted for 14 billion traffic, covering a massive 60% of the traffic analyzed. [1] In a recent study of the behavior of the AI sector from September 2022 to August 2023, traffic patterns, demographics, trends and consumer behavior were analyzed.[1]

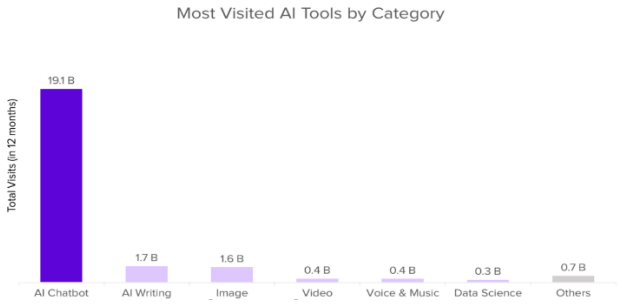


**FIGURE 1. MOST VISITED AI TOOLS BY CATEGORY**

Figure 1.1 — Most Visited AI Tools by Category shows that AI Chatbot and AI Writing have 30 percent of current mainstream AI tools. ChatGPT is the most popular model with 14.6B visits, accounting for 56% of all AI user visits. Let's explore ChatGPT, as we all know ChatGPT is the most popular model, it can master 10 programming languages, in addition to the fact that he can successfully pass the Google intermediate test, these facts are enough to prove that today's AI models do have good potential in programming.

### 2.2 CODE GENERATION AI

AI code generation technology has undergone significant advancements in recent years, leading to a variety of practical applications and tools in the software development field. The core technology behind most AI code generation tools involves deep learning algorithms and large neural networks that are trained on extensive collections of existing source code, often from open-source projects. These tools are designed to understand and generate code, facilitate pattern matching, and even refactor existing code for better efficiency.[3]

Notable examples of AI code generation tools include

GitHub Copilot, launched in 2021, which is powered by algorithms from OpenAI and trained by parsing real application code bases. This enables Copilot to make coding suggestions for software in almost any language or framework. Similarly, Amazon's CodeWhisperer and other similar tools aim to auto-generate complete snippets of code for a wide range of languages and frameworks [12].

These AI-assisted development tools have been increasingly adopted across various industries, proving advantageous for fast-moving startups, enterprises, and freelancers by speeding up product development and allowing lean teams to deliver high-quality products efficiently. These tools don't just assist in creating basic websites but can also be instrumental in building complex web and mobile applications.

### 2.3 PROBLEM

However, there are some important considerations and challenges associated with AI-generated code. We found that AI models are still inadequate when it comes to styling code and completing large, complex projects. In most cases it is difficult for code to recognize a program's coding style and this is a problem in complex projects with strong contextual relationships. The main reason is that for stylized code or complex projects they have strong contextual relationships, and when AIs work on programming code, they can't get more contextual relationships or when they do get such contextual relationships, they don't train this part of the code very well, so it's very difficult for them to work on this kind of programming. In the following article we focus on this question.

## 3 CURRENT METHODS

### 3.1 PROBLEM ANALYSIS

We observe that both the issues of stylization and the inability to handle large coding projects are closely linked to context. To gain a deeper understanding of this, the problem can be broken down into two main parts:

Lack of Labeling: This refers to the situation where AI, during its training, did not include certain types of problems as part of its training dataset. Consequently, when AI encounters specific types of problems or tasks, it may not be able to effectively generate correct code or responses due to the lack of appropriate labels or samples.

Insufficient Context: This is particularly common when dealing with complex projects. User inputs may require the AI to understand the entire context of a code project, but the AI might only have access to partial code snippets. As a result, the AI lacks a comprehensive understanding of the context, which can lead to limitations in generating code or solutions.

### 3.2 MISTRAL AI MODEL

Mistral is a Mixture of Expert (Moe) model, featuring a decoder-based large language model (LLM) architecture. It includes 8 experts per multi-layer perceptron (MLP), with a total of 85 billion parameters. Despite this large number, its computational requirement is comparable to a 14B model due to efficient processing. Each token from the hidden states is dispatched twice, ensuring more efficient computation.[2]its performance is far more than ChatGPT's latest model in performance is far more than the most popular AI language interaction model ChatGPT, the principle of which is shown in the figure below he is obtained from the weighted average of multiple models, he accepts inputs through the router, and according to the inputs to find the corresponding expert model to achieve the response to the inputs automatically. The corresponding expert model is used to implement the response to the input. Mistral model is weighted to correctly generalize the effect of the previous parameters of the model to improve the accuracy of the model.

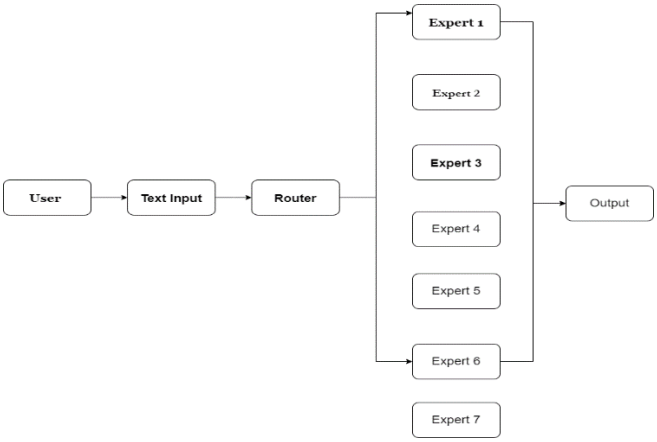


**FIGURE 2. MISTRAL AI FLOW CHART**

The described flow chart for Mistral AI, as in figure 2, illustrates a system where the router receives text input from the user and then selects the most appropriate expert models from a group of seven for output. This indicates a form of a Mixture of Experts (Moe) model, where different parts of the network specialize in different types of tasks or data. In such systems, a router or gating network determines which expert model is best suited for a given input.

### 3.3 SOLUTION

In Problem 2.1, both the lack of context and the missing labels are certainly factors in the selection of the dataset during training. It is true that we can allow the model to be fine-tuned on different data. However, the same model seems to be more bloated and bulkier; in fact, today's large language models have very large parameters in order to prevent knowledge loss, e.g., the LLM model has about 7B of parameters, while ph-2 has almost 2B, which is hardly large enough to allow LLM engineers to train in hardware-poor conditions. Thus, inspired by the Mistral model, we can consider a new approach: training specialized small-scale code models to fit specific programming styles or requirements. The key to this approach is to focus on a

specific programming task or style, rather than trying to build a single, large-scale model that can handle all situations. By taking this approach, we can utilize limited computational resources more efficiently while improving our ability to understand and handle specific code styles or complex structures. This not only makes the model more efficient and practical, but also better adapts it to specific programming environments and project requirements, thus opening up new possibilities in the field of AI programming.

# 4 METHODOLOGY

## 4.1 Weighted Average Ensemble

In the field of natural language processing, combining multiple models to improve the accuracy of results is a cutting-edge approach. Here we explore the use of weighted combinations of different models XlNet, ALBERT, and ELECTRA to improve the performance of the models in specific tasks. The key metric for evaluation is the F1 score, a commonly used measure in natural language processing tasks, balancing precision and recall. [9]

The experiment will first train and evaluate each independent model to determine their individual performance baselines. Subsequently, these models will be combined using different weighted average strategies to explore whether ensemble methods can further improve the F1 score. This includes testing equal weighted averaging as well as weighting strategies based on the performance of individual models, to find the most effective combination method. The ultimate goal of this research is to provide in-depth insights on how to effectively combine multiple pre-trained models, thereby offering guidance for model selection and integration in natural language processing tasks.

Dataset Selection： Stanford Sentiment Treebank version 2 is a well-known natural language processing dataset widely used for sentiment analysis tasks. SST was originally created by researchers at Stanford University for the purpose of analyzing and understanding sentiment expressions in text. consists of 11,855 single sentences extracted from movie reviews. It was parsed with the Stanford parser and includes a total of 215,154 unique phrases from those parse trees, each annotated by 3 human judges [3]. SST is used for Binary classification experiments It allows us to analyze and test the differences between model integration.

Model Selection: In our model selection, we have chosen XLNet, ELECTRA, and ALBERT. XLNet is commonly used for sentiment classification problems as its permutation-based training method offers a better understanding of the whole text. ELECTRA excels in training on smaller datasets. On the other hand, ALBERT is effective in language understanding but not as suitable for classification tasks; in this context, we consider it more as a source of noise expansion.

**Table 1. Weighted Average Ensemble table**

| Model | F1-score | Model Weights |
|---|---|---|
| Xlnet | 0.912 | |
| ELECTRA | 0.913 | |
| ALBERT （noise） | 0.674 | |
| Xlnet ELECTRA ALBERT | 0.921 | Xlnet: ELECTRA: ALBERT=0.5:0.5:0 |
| Xlnet ELECTRA | 0.921 | Xlnet: ELECTRA=0.5:0.5 |
| ELECTRA ALBERT | 0.910 | ELECTRA: ALBERT=1:0 |
| Xlnet ALBERT | 0.909 | Xlnet: ALBERT=0.5:0.5 |

The content in Table 3suggests that for weighted ensemble, combining higher-quality models can improve model accuracy. When introducing noisy models, a good weighted averaging model can automatically recognize and allocate weights to more reliable models, thereby reducing the impact of noise. This is a valuable feature because it allows the model to adapt to the credibility of different models. When noisy models are added, the accuracy of the ensemble model hardly decreases because during automatic traversal, the weight of the noisy model is automatically set close to zero, minimizing its impact. Additionally, even though the weights of the Xlnet and ALBERT models are nearly equal, the overall F1-Score only decreases by 0.003. This indicates that despite similar weights, the performance differences between the models are relatively small, resulting in a minor decrease in performance after weighted averaging.

## 4.2 Knowledge Distillation

In the last experiment, we have successfully trained a performance-optimized model for solving ELECTRA, a very powerful natural language processing model. Now, we intend to conduct a second set of experiments to study this model in more depth and verify its performance. The focus of this experiment is to distill a student model using the previously trained ELECTRA model as the teacher model and to fuse it with the other models in order to explore the differences between the student model and the teacher model as well as the differences in performance during weighted averaging.

**Table 2. Weighted Average Ensemble table**

| Model | F1-score | Model Weights |
|---|---|---|
| Xlnet | 0.912 | |
| ELECTRA | 0.913 | |
| ALBERT （noise） | 0.674 | |
| Student | 0.910 | |
| Xlnet ELECTRA ALBERT | 0.921 | 0.5:0.5:0 |
| Xlnet ELECTRA | 0.921 | 0.5:0.5 |
| ELECTRA ALBERT | 0.910 | 1：0 |
| Xlnet ALBERT | 0.909 | 0.5:0.5 |
| Xlnet ELECTRA ALBERT Student | 0.921 | 0.5:0:0.5:0 |

Journal of Industrial Engineering and Applied Science
Journal Home: jieas.suaspress.org | CODEN: JIEAAE
Vol. 2, No. 5, 2024 | ISSN 3005-6071 (Print) | ISSN 3005-608X (Online)

| | | |
|---|---|---|
| ELECTRA ALBERT Student | 0.916 | 0.5:0:0.5 |
| ALBERT: Xlnet: Student | 0.915 | 0:0.5:0.5 |
| ELECTRA: Xlnet: Student | 0.921 | 0.5:0.5:0 |
| Xlnet: Student | 0.915 | 0.5:0.5 |
| ALBERT: Student | 0.907 | 0:1 |
| ELECTRA: Student | 0.916 | 0.5:0.5 |

According to the test results in Table 3.2, it is observed that the F1-Score of the student model is slightly lower than that of the teacher model under the same conditions. However, when the student model is fused with other models, its F1-Score decreases slightly compared to fusing the teacher model with other models. Nevertheless, an interesting observation is that the weighted proportions of the distilled student model, when compared to other models, closely resemble those of the teacher model. This indicates that the knowledge distillation process successfully transferred the knowledge from the teacher model to the student model, and the student model plays a significant role in the ensemble.

This finding underscores the potential of knowledge distillation, as it can transform the complex knowledge of the teacher model into a lighter-weight student model while achieving good performance in ensembles, while maintaining consistent weight distribution. This is important for model lightweighting, deployment, and achieving good performance across various tasks.

# 5 PROTOTYPE

Based on the findings from tests conducted in sections 3.1 and 3.2, it's evident that performing a weighted average of model outputs significantly enhances model accuracy. Additionally, post-knowledge distillation, the student model can effectively substitute the teacher model in terms of decision-making, including weight ratios. Consequently, we can propose the following prototype hypothesis: We can establish a comprehensive ensemble of large models, encompassing various stylized models, each proficient in distinct code styles. When users present code-related queries, these queries can be directed through a routing mechanism. This routing process involves distilling sub-models, leading to the identification of the optimal combination of models for decision-making. Subsequently, the results can be further enriched by incorporating the weighted output from the teacher's model. Furthermore, for substantial projects, users have the option to train these stylized models using their own stylized code. The outcomes of this training process can then be employed for code generation.

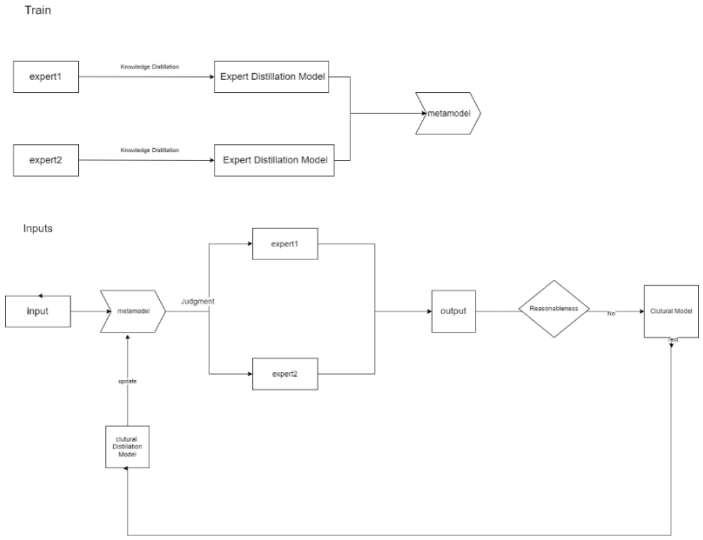


**FIGURE 2. PROTOTYPE**

# 6 CONCLUSION

In summary, the article presents a comprehensive analysis and proposed solutions for advancing AI code generation technology. It begins with an overview of the current AI landscape, highlighting the significant growth in AI tool usage, with a particular emphasis on AI models like ChatGPT that have shown exceptional potential in programming. The article then delves into AI code generation, noting its rapid advancement and the emergence of tools like GitHub Copilot and Amazon's CodeWhisperer, which have revolutionized software development.

The main problem identified is the limitation of AI in generating stylized code and handling large, complex projects. This issue is attributed to a lack of context and insufficient labeling during AI training. To address these challenges, the article introduces Mistral, a Mixture of Experts model, and proposes a prototype that combines large models proficient in various coding styles. This prototype utilizes a routing mechanism to direct user queries to the most suitable models, enhancing decision-making and code generation capabilities.

The article further explores the concept of weighted average ensemble and knowledge distillation. It demonstrates how combining multiple models and distilling knowledge from a teacher model to a student model can significantly improve model accuracy and efficiency. Finally, the article discusses strategies for model acceleration and parallelism, particularly heterogeneous and pipeline parallelism. These methods are proposed to distribute models across multiple servers, optimizing query processing speed and resource utilization in a distributed model environment.

Overall, the article provides valuable insights into the current state and future potential of AI in programming, offering innovative solutions to overcome existing challenges in AI code generation.

## ACKNOWLEDGMENTS

The authors thank the editor and anonymous reviewers for their helpful comments and valuable suggestions.

## FUNDING

Not applicable.

## INSTITUTIONAL REVIEW BOARD STATEMENT

Not applicable.

## INFORMED CONSENT STATEMENT

Not applicable.

## DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

## CONFLICT OF INTEREST

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

## PUBLISHER'S NOTE

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

## AUTHOR CONTRIBUTIONS

Not applicable.

## ABOUT THE AUTHORS

**ZHANG, Dutao**

Faculty of Digital Transformation, ITMO University, St. Petersburg, Russia.

**LI, Boyi**

Faculty of Digital Transformation, ITMO University, St. Petersburg, Russia.

## REFERENCES

[1] Zhao, W. X., et al. (2023). A survey of large language models. *arXiv preprint*. http://arxiv.org/abs/2303.18223

[2] Jiang, A. Q., et al. (2023). Mistral 7B. *arXiv preprint*. http://arxiv.org/abs/2310.06825

[3] Liu, H., Liu, L., Yue, C., Wang, Y., & Deng, B. (2023). AutotestGPT: A system for the automated generation of software test cases based on ChatGPT. *SSRN*. https://www.ssrn.com/abstract=4584792. https://doi.org/10.2139/ssrn.4584792

[4] Nijkamp, E., et al. (2023). CodeGen: An open large language model for code with multi-turn program synthesis. *arXiv preprint*. http://arxiv.org/abs/2203.13474

[5] Melo, G., Alencar, P., & Cowan, D. (2019). Context-augmented software development projects: Literature review and preliminary framework. *arXiv preprint*. http://arxiv.org/abs/1910.08167

[6] Wen, S.-F. (2023). Context-based support to enhance developers' learning of software security. *Education Sciences, 13*(631). https://doi.org/10.3390/educsci13100631

[7] Elyasaf, A. (2021). Context-oriented behavioral programming. *Information and Software Technology, 133*, 106504. https://doi.org/10.1016/j.infsof.2021.106504

[8] Maekawa, A., Kobayashi, N., Funakoshi, K., & Okumura, M. (2023). Dataset distillation with attention labels for fine-tuning BERT. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (pp. 119–127). Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.acl-short.12

[9] Sohail, S. S., et al. (2023). Decoding ChatGPT: A taxonomy of existing research, current challenges, and possible future directions. *Journal of King Saud University - Computer and Information Sciences, 35*, 101675. https://doi.org/10.1016/j.jksuci.2023.101675

[10] Kirk, D., & MacDonell, S. G. (2014). Investigating a conceptual construct for software context. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering* (pp. 1–10). ACM. https://doi.org/10.1145/2601248.2601263

[11] Gu, A., & Dao, T. (n.d.). Mamba: Linear-time sequence modeling with selective state spaces.

[12] Abuhamad, M., Abuhmed, T., Nyang, D., & Mohaisen, D. (2020). Multi-χ: Identifying multiple authors from source code files. *Proceedings on Privacy Enhancing Technologies, 2020*(1), 25–41. https://doi.org/10.2478/popets-2020-0002

[13] Kirk, D. (2021). Software development context: Critiquing often-used terms. In *Proceedings of the 16th International Conference on Evaluation of Novel Approaches to Software Engineering* (pp. 340–347). SCITEPRESS – Science and Technology Publications. https://doi.org/10.5220/0010469903400347

[14] D'Avila, L. F., Barbosa, J. L. V., & Oliveira, K. S. F. (2020). SW-Context: A model to improve developers' situational awareness. *IET Software, 14*(7), 535–543. https://doi.org/10.1049/iet-sen.2019.0345

[15] Wu, S., et al. (n.d.). YUAN 2.0: A large language model with localized filtering-based attention.