IPC: Intelligent Programming Companion

Yosef Madboly   
*Computer Science*  
*Ain Shams University*Cairo, Egypt  
20201701037@cis.asu.edu.eg

Salma Mahdy   
*Computer Science*  
*Ain Shams University*Cairo, Egypt  
20201700353@cis.asu.edu.eg

Mohamed Hesham   
*Computer Science*  
*Ain Shams University*Cairo, Egypt  
20201701245@cis.asu.edu.eg

Salma Ayman   
*Computer Science*  
*Ain Shams University*Cairo, Egypt  
20201700345@cis.asu.edu.eg

Omar Yasser  
*Computer Science*  
*Ain Shams University*Cairo, Egypt  
20201701195@cis.asu.edu.eg

Martina Angelos  
*Computer Science*  
*Ain Shams University*Cairo, Egypt  
20201700624@cis.asu.edu.eg

**Abstract—**Competitive programmers and software engineers preparing for interviews encounter several critical challenges, such as uncertainty on how to prepare, problem repetition, inconsistent difficulty assessments, platform fragmentation, and ambiguous problem descriptions, all of which impede effective learning and fair competition. To tackle these issues comprehensively, a centralized platform is proposed, featuring four main modules: problem summarization, technique-based and semantic similarity for problems and solutions, problem difficulty-level and tags prediction, and a solver for standard problems that can appear in an interview. The Summarization Module leverages Google’s Gemini 1.5 architecture, with advanced prompt engineering techniques. The Similarity Module uses the nearest-vector approach, utilizing state-of-the-art vector databases to find the most similar problems/solutions, achieving a remarkable 97.5% recall@10 on problem statements and 99.6% recall@10 on solutions. Meanwhile, the Difficulty/Tags Prediction Module applies a BigBird BERT-based architecture to accurately predict problem difficulty levels and assign relevant tags, demonstrating robust performance with an ROC AUC score of 76.37%. Lastly, the solver module is presented as a helpful interactive chatbot for dynamic problem-solving, built on a RAG architecture with best-in-class open-weight CodeQwen LLM. The presented results underscore the efficacy and practicality of our proposed solutions. Through the implementation of this platform, our aim is to foster a more cohesive and supportive environment for competitive programmers and software engineer interview candidates. By encouraging innovation, promoting knowledge sharing, and enhancing problem-solving experiences, our system aims not only to elevate current standards in competitive programming and interview preparation but also to lay the groundwork for future advancements in Computer Science education and problem-solving methodologies.

***Keywords—Competitive programming, Interview Preparation, Algorithmic Problem Uniqueness, Algorithmic Problem summarization, Algorithmic problem Tags prediction***

# **Introduction**

The field of Computer Science (CS) is known for its rapid evolution and the continuous learning required from students, employees, and employers. Among the various domains within CS, problem-solving is particularly challenging and popular, requiring ongoing practice and upskilling. A key question in this field is: “How can one best prepare for interviews and competitive programming contests?” Despite numerous platforms hosting diverse problems, several significant issues impact both interview preparation and competitive programming:

Repetition and Uniqueness: Problem setters often create similar problems unknowingly, leading to repeated problems. This repetition gives an unfair advantage to those familiar with these problems and reduces the overall value of the problem sets.

Difficulty Assessment: Problem setters frequently struggle to determine the appropriate difficulty level for their problems, leading to inconsistent assessments that hinder effective preparation and fair competition.

Platform Fragmentation: The absence of a centralized platform for problem setters to share and collaborate results in problems being scattered across multiple websites, causing confusion and making it difficult for individuals to find problems for practice.

Clarity of Problem Descriptions: Many problem descriptions lack clarity, posing challenges for both human solvers and AI tools. Unclear problem statements hinder accurate comprehension and effective problem-solving.

Addressing these challenges will enhance the competitive programming ecosystem and improve interview preparation strategies. By grouping problems, reducing repetition, and refining problem descriptions, we aim to create a more cohesive and supportive environment for all participants.

# **Literature Review**

## Problem Summarization Module

Text summarization condenses lengthy text into a shorter version while retaining essential information. Methods include extractive, which selects key sentences, and abstractive, which generates new text. Advanced models improve summarization accuracy and reduce repetition but struggle with competitive programming problems due to excessive details and lack of annotated datasets.

## Problem Similarity Module

Semantic textual similarity measures how similar two texts are. BERT [5] introduced a bidirectional language representation model, improving semantic similarity tasks significantly. Sentence-BERT [1] (SBERT) further reduced computational overhead and maintained accuracy, making it more suitable for similarity search and clustering.

## Solution Similarity Module

An unsupervised voting algorithm was developed for detecting algorithmic strategies in competitive programming solutions [2], leveraging multiple code embedding techniques. CNNs were used to classify C++ solutions by identifying algorithmic approaches [3]. Another method predicted user rank and country based on C++ submissions using ANNs. Synthesizing short Python programs from natural language descriptions was also explored. Despite these advances, competitive programming problems often obscure necessary algorithms, impacting performance.

## Predicting Problem Tags

A dataset of 4,019 problems annotated with algorithm tags showed that CNN-based classifiers can achieve near-human performance in tag prediction. Transformer-based architectures are proposed for handling long sequences typical of problem descriptions.

## Solver Module

Large language models (LLMs) were investigated for programming questions, achieving notable performance on benchmark datasets. LLMs like Codex [4] have shown success in program synthesis from natural language queries. However, reinforcement learning (RL) methods for enhancing Code LLMs face limitations in implementation and computational cost. A system for generating innovative solutions to coding challenges achieved top performance but faced limitations in model training and evaluation processes.

# **System Architecture**

The system architecture consists of three layers: Presentation, Logic, and Data.

## Presentation Layer

Users interact with the system via the user interface (UI), which sends queries to the Solver Module for responses, problems to the Summarization Module for summaries, problems to the Difficulty/Tags Prediction Module for determining difficulty levels and tags, and problems or solutions to the Similarity Module to find similar items.

## Logic Layer

This layer processes information from the presentation layer using four modules: the Problem Summarization Module, the Problem/Solution Similarity Module, the Difficulty/Tag Prediction Module, and the Solver Module. Each module has a specific role and collectively contributes to the overall functionality of the system.

## Data Layer

This layer manages and stores all data related to problems and solutions, including problem texts, inputs/outputs, summaries, difficulty levels, tags, solutions, and other metadata. It ensures that all necessary information is readily available for retrieval and processing by the modules in the logic layer.

# **Proposed Method**

## Problem Summarization Module

This module strips extraneous details from problem statements. Two approaches were explored: pre-LLM methods, utilizing architectures like BERT [5], BART, T5 [6], and PEGASUS [7]; and LLM with prompt engineering, using LLMs like Gemma-7b [8], Mixtral-8x7b [9], Gemini 1.5 flash [10], and Vicuna-33b, with detailed prompts to generate concise statements.

## Problem/Solution Similarity Module

This module identifies similar problems or solutions using two approaches: the Clustering Approach, which groups similar items into clusters using algorithms and embeddings like all-mpnet-base-v2, bge-small-en-v1.5, and all-MiniLM-L6-v2 for problems, and voyage-code-2 for solutions; and the Nearest Vectors Approach, which represents items as vectors and finds the nearest ones using the same embeddings as the clustering approach.

## Difficulty/Tags Prediction Module

This module predicts tags and difficulty levels using a multi-task learning approach with a BigBird BERT-based architecture. The model processes problem statements, feeding them into a multi-label classifier and difficulty predictor.

## Solver Module

Built on a Retrieval-Augmented Generation (RAG) architecture with CodeQwen [11] LLM capabilities, this module functions as an interactive chatbot. It uses indexing and retrieval-generation processes to provide accurate, contextually relevant responses to user queries. The model architecture incorporates un-tied embeddings and rotary positional embeddings for improved code generation and understanding.

# **System Implementation**

## Dataset

Our research utilizes a comprehensive dataset of programming problems and their corresponding solutions, scraped from various online judges and GitHub repositories. **Table 1** presents the distribution of problems and solutions across different sources.

| Online Judge | Number Of Problems | Number Of Solutions |
| --- | --- | --- |
| AtCoder | 2307 | 1,587,756 |
| CodeChef | 3201 | 53,961 |
| CSES | 296 | 1,175 |
| Codeforces | 8636 | 16,229 |
| HackerEarth | 985 | 36,179 |
| LeetCode | 2424 | 3,570 |
| Uva | 1461 | 1,347 |
| Yosupo | 193 | 195 |

**Table 1: Problems and Solutions Dataset details**

The dataset is divided into two primary parts: Problems and Solutions. The Problems dataset includes the problem link, problem text, sample input/output, time limit, memory limit, tags, and difficulty level. The Solutions dataset includes the problem link, submission link, problem name, source, language (all solutions are in C++), time, memory, and the solution code.

## Preprocessing

For solutions, preprocessing steps included removing #include directives, using namespace declarations, and comments; removing all non-ASCII characters; removing unused functions using cpp-check; tokenizing the source code using pygments; and attempting to replace all macro directives, although this approach was eventually discarded due to an excessive number of tokens.

For problems, preprocessing steps included substituting exponential notation, inserting spaces between dollar signs, converting text to lowercase, calculating expressions, removing stopwords using the nltk library, and performing lemmatization to reduce words to their base form.

These preprocessing steps ensured the data fed into our models was clean, consistent, and focused on the functional and relevant aspects of the problems and solutions, enhancing the performance and accuracy of our modules.

# **Results and Discussion**

*A. Problem Summarization Module*

Several low-cost or open-weight LLMs were prompt-engineered to summarize problem statements. Metrics used included:

* General Ability (Chatbot Arena) [12]
* Reasoning & Knowledge (MMLU)
* Reasoning & Knowledge (MT Bench)

**Table 2** summarizes the performance of different LLMs on these benchmarks.

| **Model** | **General Ability (Chatbot Arena)** | **Reasoning & Knowledge (MMLU)** |
| --- | --- | --- |
| Gemini 1.5 Flash | **1231** | **79%** |
| Mixtral 8x7B | 1114 | 71% |
| Gemma-7B | 1037 | 64% |

**Table 2: Performance of Different LLMs**

Results indicate that Gemini 1.5 Flash performs best for its size.

*B. Problem/Solution Similarity Module*

The Silhouette score is used to evaluate the quality of the clusters formed. It measures how similar an object is to its own cluster compared to other clusters. A high Silhouette score indicates that the objects are well-matched to their own cluster and poorly matched to neighboring clusters.

**Tables 3** and **4** show the silhouette scores for problem statements, and solutions respectively, showcasing different combinations of clustering algorithms with different embedding methods.

| Embeddings  Model | Clustering Algorithm | Reasoning & Knowledge (MMLU) | Reasoning &  Knowledge  (MT Bench) |
| --- | --- | --- | --- |
| Sentence-  transformers  /all-  mpnet-base-v2 | K-Means | 30 | 0.14809753 |
| Agglomerative | 30 | 0.12610750 |
| Special Clustering | 30 | 0.10872551 |
| DBscan | - | -0.10904756 |
| Sers/  all-  MiniLM-L6-v2 | K-Means | 30 | 0.13519753 |
| Agglomerative | 30 | **0.18610651** |
| Special Clustering | 30 | 0.10872551 |
| Dbscan | - | -0.18904756 |

**Table 3: Silhouette scores for different clustering algorithms with different embeddings for problem statements**

| Embeddings Model | Clustering Algorithm | Reasoning & Knowledge (MMLU) | Reasoning & Knowledge (MT Bench) |
| --- | --- | --- | --- |
| tf-idf | K-Means | 30 | **0.52416289** |
| Agglomerative | 30 | 0.12610750 |
| Special Clustering | 30 | 0.10872551 |
| DBscan | - | -0.10904756 |
| Voyage-  Code-2 | K-Means | 30 | 0.13519753 |
| Agglomerative | 30 | 0.18610651 |
| Special Clustering | 30 | 0.10872551 |
| Dbscan | - | -0.18904756 |
| sentence-transformers/all-mpnet-base-v2 | K-Means | 23 | 0.13519753 |
| Agglomerative | 23 | 0.113510504 |
| Special Clustering | 23 | -0.040572744 |
| Dbscan | - | -0.06446284 |

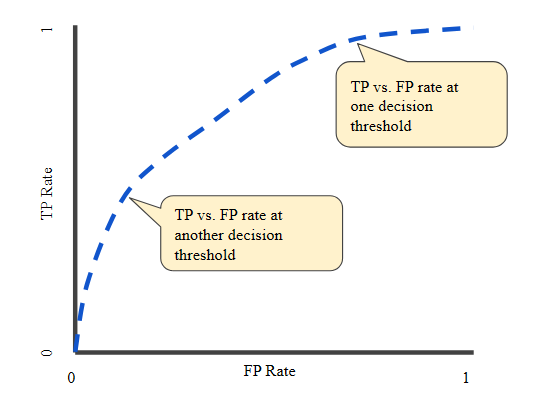
**Table 4: Silhouette scores for different clustering algorithms with different embeddings for problem statements**

The Agglomerative clustering along with all-MiniLM-L6-v2 embedding achieved the highest score for problem statements.

The K-means clustering along with TF-IDF embedding achieved the highest score for solutions.

*C. Difficulty/Tags Prediction Module*

An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds, as shown in the **figure 1**.



**Figure 1: ROC curve**

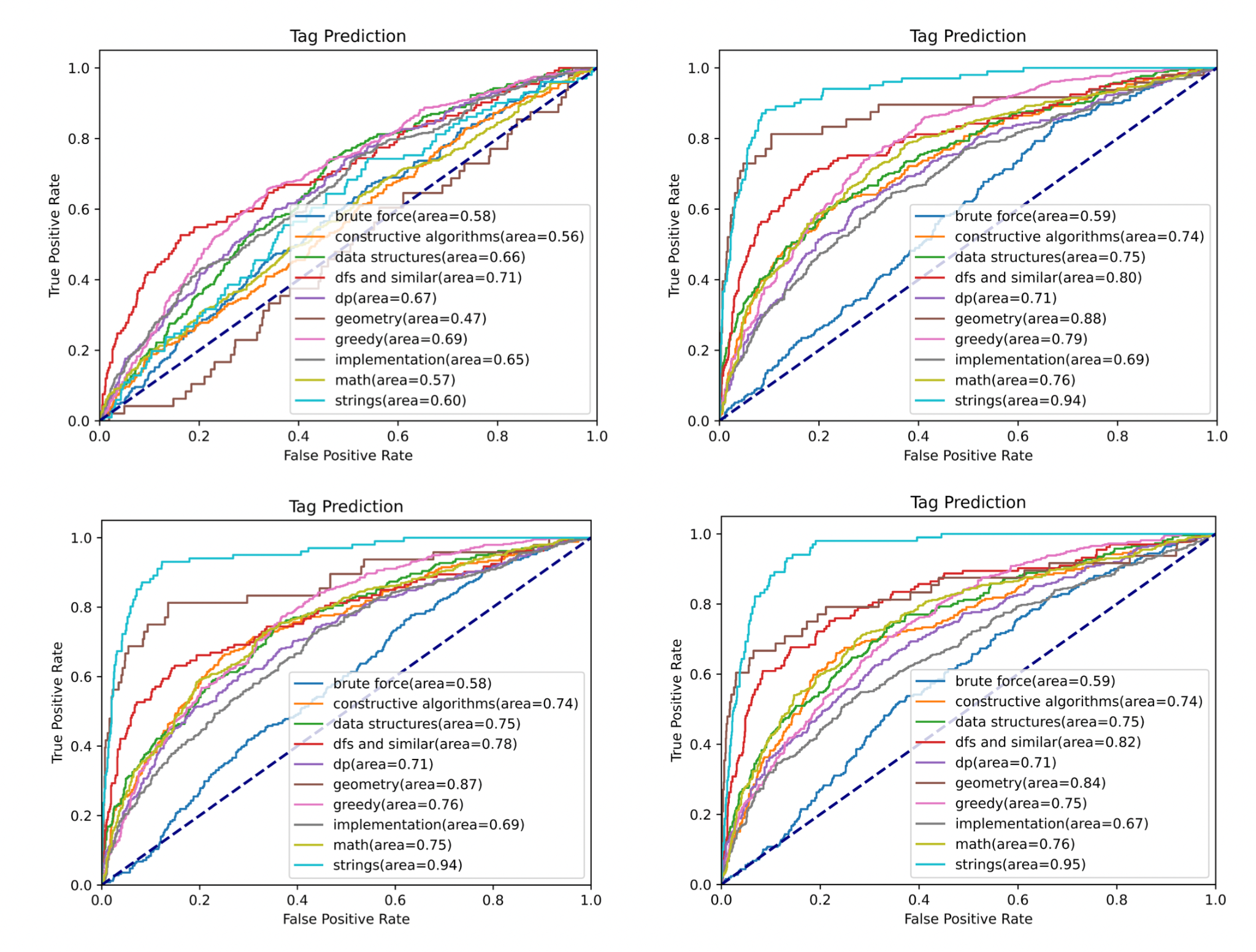
The ROC curve plots two parameters:

* TPR (True Positive Rate): Also known as recall, it shows the share of detected true positives. For example, the share of emails correctly labeled as spam out of all spam emails in the dataset.
* FPR (False Positive Rate): This shows the share of objects falsely assigned a positive class out of all objects of the negative class. For example, the proportion of legitimate emails falsely labeled as spam.

The ROC Curves of Various Model Variations

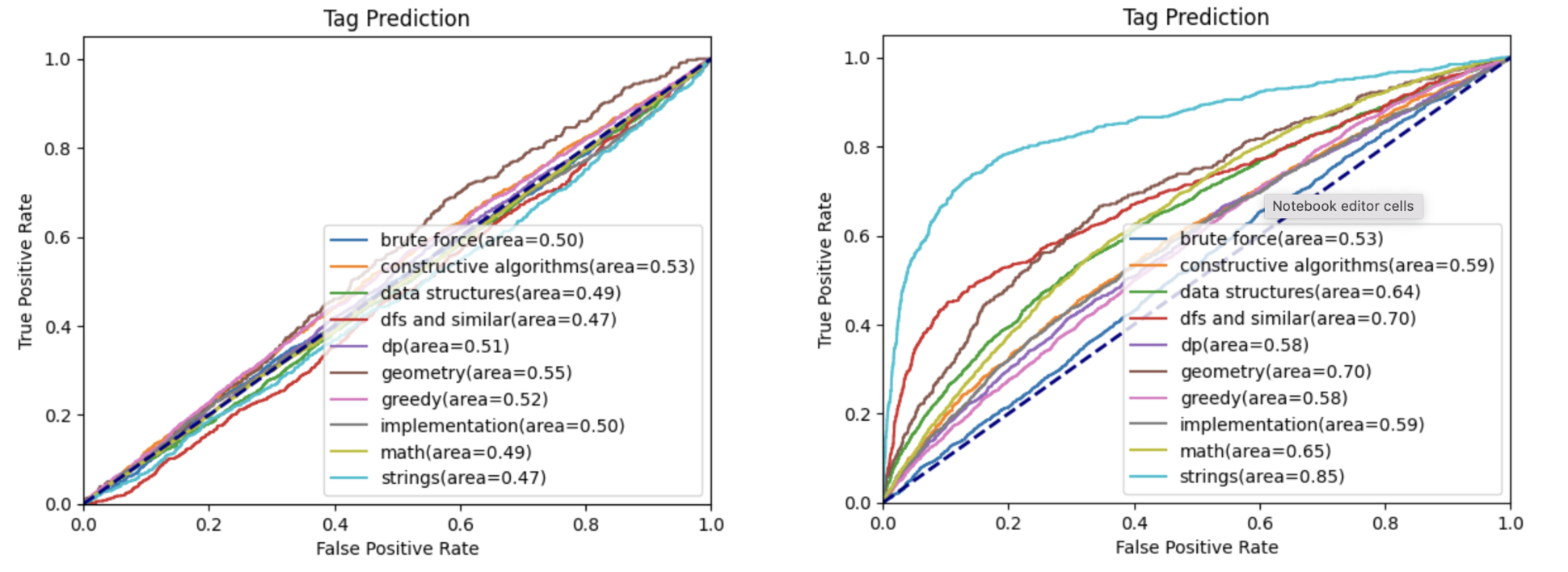
As shown in **figure 2**, the ROC curves for different model variations are depicted:

* Top-left graph: ROC curve of our PSG model trained with λ=1.
* Top-right graph: ROC curve of our PSG model trained with λ=10.
* Bottom-left graph: ROC curve of our PSG model trained with λ=100.
* Bottom-right graph: ROC curve of the single-task BigBird model used to solve only the second task T2.



**Figure 2: ROC curves for different model variations**

shows the ROC curve of the model trained with λ = 10 before vs after fine-tuning.

**Figure 3: ROC curve before and after fine-tuning**

By evaluating the ROC curve and calculating the AUC (Area Under the Curve), we can assess the overall performance of the classification model. The higher the AUC, the better the model is at distinguishing between positive and negative classes. The various model variations and fine-tuning steps help in understanding how different training strategies and hyperparameters affect the model's performance.

*D. Solver Module*

## Evaluation Metric

### HumanEval 0-shot

HumanEval is a dataset designed to evaluate the ability of Large Language Models (LLMs) to generate correct and functional code snippets. It comprises a set of programming problems that require the implementation of certain functionalities in Python. The 0-shot evaluation variant of HumanEval assesses the model’s ability to solve these problems without any fine-tuning or additional training examples. This metric directly measures the model's inherent understanding and generation capabilities in the context of programming tasks.

### MBPP 0-shot

MBPP (Model-based Policy Optimization), although not originally a code generation dataset, has been adapted to assess LLMs in code-related tasks. In this context, MBPP contains more complex programming problems that often involve algorithmic thinking and optimization. The 0-shot evaluation in MBPP tests the model's capability to solve these intricate problems from scratch, without any specific training on the problem set. This metric is particularly challenging and provides a measure of the model's advanced programming and logical reasoning skills.

### MBPP+ 0-shot

MBPP+ extends the MBPP dataset by increasing the diversity and complexity of the programming problems. This can include more advanced algorithmic challenges, real-world problems, and multi-language tasks. The 0-shot evaluation in MBPP+ demands that the model demonstrate an even higher level of programming expertise and adaptability across various domains and languages. It serves as a stringent test of the model's ability to reason, learn, and generate code effectively in unfamiliar situations.

### MBPP 3-shot

In contrast to the 0-shot evaluations, MBPP 3-shot provides the model with a small number of examples (typically three) to learn from before generating solutions to unseen challenges. This evaluation setup assesses the model’s capacity for few-shot learning, where it can quickly adapt to new tasks with minimal guidance. The MBPP 3-shot evaluation measures not only the model’s programming ability but also its ability to effectively utilize and generalize from a small number of examples, which is a practical scenario in many real-world applications.

### LiveCodeBench

LiveCodeBench is a dynamic evaluation metric for LLMs, focusing on real-time coding challenges. It aggregates problems from LeetCode [45], AtCoder [40], and Codeforces [41], offering a current and diverse test suite. This benchmark mirrors competitive programming environments, testing LLMs’ adaptability and proficiency in algorithmic problem-solving.

Each of these datasets and evaluation methods offers a unique perspective on the capabilities of LLMs in the context of code generation and algorithmic problem-solving. Different LLMs were assessed according to these evaluations, as shown in **tables 5** and **6**.

| Model | Size | HumanEval 0-shot | HumanEval+ 0-shot | MBPP 0-shot | MBPP+ 0-shot | MBPP3-shot |
| --- | --- | --- | --- | --- | --- | --- |
| CodeLlama-Base | 7B | 33.5 | 25.6 | 52.1 | 41.6 | 38.6 |
| StarCoder2 | 7B | 35.4 | 29.9 | 54.4 | 45.6 | 51.0 |
| DeepSeek-Coder-Base | 6.7B | 47.6 | 39.6 | 70.2 | 56.5 | 60.6 |
| CodeQwen1.5-Chat | 7B | **83.5** | **78.7** | **77.7** | **67.2** | **70.6** |

**Table 5: Different LLMs assessed on various benchmarks**

| Model | Size | LiveCodeBench All Time [P](about:blank)ass@1 | LiveCodeBench 2023/9/1~2024/4/1 Pass@1 |
| --- | --- | --- | --- |
| CodeLLama-Base | 7B | 6.5 | 7.6 |
| StarCode2 | 7B | 11,3 | 12.7 |
| DeepSeek-Coder-Base | 6.7B | 19.1 | 13.7 |
| CodeQwen1.5 | 7B | **21.8** | **19.3** |

**Table 6: Different LLMs assessed on LiveCodeBench**

CodeQwen [11] was chosen due to its state-of-the-art results.

Retrieval-Augmented Generation (RAG) grounds the chatbot to relevant problems and solutions to:

* Provide better context for problems it may not have been trained on or seen before.
* Avoid hallucinations, ensuring accurate and relevant responses.

By integrating RAG, LLM responses are significantly enhanced, improving their effectiveness and reliability in real-world coding scenarios.

##### **Conclusion**

Our solution aimed to address significant challenges in competitive programming and interview preparation through the development of a centralized platform with integrated modules focused on problem uniqueness, solver efficiency, difficulty prediction, and problem clarity.

The implemented modules—Summarization, Similarity, Difficulty/Tags Prediction, and Solver—have demonstrated strong performance metrics. The Similarity Module achieved a remarkable 99.6% recall@10, indicating its efficacy in identifying similar problems and ensuring problem set originality. The Difficulty/Tags Prediction Module showed robust performance with a ROC AUC score of 76.37%, guiding users effectively in problem selection based on difficulty levels and relevant tags.

The findings of this project are crucial for advancing the field of competitive programming and interview preparation. They underscore the significance of integrated platforms in overcoming existing challenges and improving learning outcomes.

By addressing issues such as problem repetition, inconsistent difficulty assessments, platform fragmentation, and ambiguous problem descriptions, our platform aims to create a more cohesive and supportive environment for competitive programmers and interview candidates. The tools developed not only streamline problem-solving processes but also foster innovation and knowledge sharing within the community.

##### **Acknowledgment**

We are profoundly grateful to God, who provided us with the strength and perseverance to complete this work. We pray that He accepts our efforts and that this work serves its intended purpose. We would like to extend our sincerest gratitude to our supervisors, Prof. Dr. Ahmed Salah and T.A. Razan Bayoumi, for their unwavering support, patience, and invaluable guidance. Their expertise and encouragement were instrumental in the completion of this work. We also wish to express our heartfelt thanks to our families and friends for their continuous support and encouragement. Lastly, we hope that this work contributes positively to the community and serves its intended purpose.

##### **References**

1. Reimers, N., & Gurevych, I. (2019, August 27). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. arXiv.org. https://doi.org/10.48550/arXiv.1908.10084 [1]
2. Stoica, Alexandru & Băbiceanu, Daniel & Rebedea, Traian & Mihaescu, Cristian. (2023). Unsupervised Voting for Detecting the Algorithmic Solving Strategy in Competitive Programming Solutions. 10.21203/rs.3.rs-2834777/v1. [2]
3. Ohashi, Hiroki & Watanobe, Yutaka. (2019). Convolutional Neural Network for Classification of Source Codes. 194-200. 10.1109/MCSoC.2019.00035. [3]
4. Grossman, L. (2005). Codex. Wadsworth Publishing. [4]
5. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019, May 24). Bert: Pre-training of deep bidirectional Transformers for language understanding. arXiv.org. https://arxiv.org/abs/1810.04805 [5]
6. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. arXiv. https://arxiv.org/abs/1910.10683 [6]
7. Zhang, J., Zhao, Y., Saleh, M., & Liu, P. J. (2020). PEGASUS: Pre-training with extracted gap-sentences for abstractive summarization. In Proceedings of the 37th International Conference on Machine Learning (Vol. 119). https://proceedings.mlr.press/v119/zhang20ae/zhang20ae.pdf [7]
8. Team, Gemma, Mesnard, T., Hardin, C., Dadashi, R., Bhupatiraju, S., Pathak, S., Sifre, L., Rivière, M., Kale, M. S., Love, J., Tafti, P., Hussenot, L., Sessa, P. G., Chowdhery, A., Roberts, A., Barua, A., Botev, A., Castro-Ros, A., Slone, A., … Kenealy, K. (2024a, April 16). Gemma: Open models based on Gemini Research and Technology. arXiv.org. https://arxiv.org/abs/2403.08295 [8]
9. Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., Casas, D. de las, Hanna, E. B., Bressand, F., Lengyel, G., Bour, G., Lample, G., Lavaud, L. R., Saulnier, L., Lachaux, M.-A., Stock, P., Subramanian, S., Yang, S., … Sayed, W. E. (2024, January 8). Mixtral of Experts. arXiv.org. https://arxiv.org/abs/2401.04088 [9]
10. Team, G., Georgiev, P., Lei, V. I., Burnell, R., Bai, L., Gulati, A., Tanzer, G., Vincent, D., Pan, Z., Wang, S., Mariooryad, S., Ding, Y., Geng, X., Alcober, F., Frostig, R., Omernick, M., Walker, L., Paduraru, C., Sorokin, C., … Vinyals, O. (2024a, June 14). Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. arXiv.org. https://arxiv.org/abs/2403.05530 [10]
11. QwenLM. (n.d.). QWENLM/Codeqwen1.5: Codeqwen1.5 is the code version of Qwen, the large language model series developed by Qwen team, Alibaba Cloud. GitHub. https://github.com/QwenLM/CodeQwen1.5 [11]
12. Chatbot Arena: Find the best chatbot builder. Chatbot Arena: Find the Best Chatbot Builder. (n.d.). https://chatbotarena.com/ [12]