AutoJudge: Automated Task Complexity Evaluation using Efficiently Fine-Tuned Transformer Models

Abhiraj Bharangar
Enrollment No: 24117002
Department of Computer Science and Engineering
Indian Institute of Technology Roorkee (IITR)
abhiraj_b@cs.iitr.ac.in

January 8, 2026

Abstract

Assessing the complexity of programming tasks is a critical challenge for educational platforms and competitive programming repositories. This paper introduces *AutoJudge*, a multi-task learning system designed to automatically predict the difficulty class (Easy, Medium, Hard) and a continuous complexity score for coding problems. We explore and evaluate three transformer-based architectures: ModernBERT-Large, RoBERTa-Large, and DeBERTa-v3-Large. Utilizing Low-Rank Adaptation (LoRA) for parameter-efficient fine-tuning, we demonstrate that smaller, specialized BERT-based models significantly outperform larger general-purpose LLMs (e.g., Llama-3.2-3B) on this specific task. Our best-performing model, DeBERTa-v3-Large, achieved a classification accuracy of 57.11% and a regression Mean Absolute Error (MAE) of 1.6300. The system is deployed via a Streamlit web interface for real-time analysis. The complete code and dataset are available at: `https://github.com/Always-Exploring-exe/AutoJudge_Project_24117002`.

1 Introduction

The proliferation of online coding platforms has led to a massive influx of programming problems. Categorizing these problems by difficulty is traditionally a manual, subjective process prone to inconsistency. An automated system capable of analyzing problem statements—including descriptions, input/output constraints, and sample cases—to predict complexity would significantly streamline content curation and personalization for learners.

In this work, we propose a deep learning approach to map problem text to complexity metrics. We define the problem as a multi-task learning objective:

1. **Classification**: Categorizing problems into three tiers: Easy, Medium, and Hard.
2. **Regression**: Predicting a fine-grained difficulty score ranging from 1.0 to 10.0 (based on platform-specific ratings).

We hypothesize that encoder-only transformer models (like BERT and its variants) are structurally better suited for this discriminative task compared to decoder-only generative Large Language Models (LLMs), which often struggle with non-generative regression tasks without extensive instruction tuning.

2 Dataset and Methodology

2.1 Dataset

We utilized the *TaskComplexityEval-24* dataset, consisting of approximately 4,112 programming tasks scraped from platforms such as Kattis and LeetCode. Each sample includes:

- **Text Features:** Title, Problem Description, Input Description, Output Description.
- **Targets:**
  - *Problem Class:* Ground truth label (Easy, Medium, Hard).
  - *Problem Score:* A continuous difficulty rating (normalized 1.0–9.7).

2.2 Preprocessing

The raw text fields were concatenated into a single `full_text` input string to maximize context for the model. Special separation tokens (e.g., `[SEP]`) were inserted between the title, description, and constraints to help the model distinguish between different semantic sections of the problem statement.

2.3 Model Architectures

We evaluated three state-of-the-art encoder models:

- **ModernBERT-Large:** A recently optimized BERT variant with extended context capabilities.
- **RoBERTa-Large:** A robustly optimized BERT model trained on larger corpora [3].
- **DeBERTa-v3-Large:** Utilizing disentangled attention and enhanced mask decoding, often state-of-the-art for NLU tasks [2].

To manage computational constraints while fine-tuning these large models, we employed **Low-Rank Adaptation (LoRA)** [1]. LoRA freezes the pre-trained model weights and injects trainable rank decomposition matrices into the layers of the Transformer architecture, significantly reducing the number of trainable parameters.

2.4 The LLM Linear Probe Experiment

Prior to fine-tuning, we conducted a baseline experiment using a modern medium-sized LLM, **Llama-3.2-3B-Instruct** [4]. We attached linear probes to the residual stream of the final token to regress the complexity score and classify difficulty.

The results were notably poor:

- **Classification Accuracy:** 40.22%
- **Regression MAE:** 3.28
- **Regression MSE:** 3.96

This experiment highlighted a key insight: while LLMs are powerful generators, their pre-trained representations for direct regression on specific domain metrics (like competitive programming difficulty) are not inherently aligned without significant fine-tuning. Furthermore, specific keywords or "tags" in problem statements can be misleading; a problem with simple tags can be extremely difficult due to subtle logical constraints, nuances that generic LLM embeddings often miss. This justified our decision to fine-tune specialized encoder models instead.

3 Experimental Setup

All models were trained using a unified multi-task loss function combining Cross-Entropy Loss (for classification) and Mean Squared Error Loss (for regression). The hyperparameters used across all experiments are detailed in Table 1.

Table 1: Hyperparameters for Fine-Tuning

Parameter	Value
Max Sequence Length	350
Batch Size	8
Epochs	10
Learning Rate	5×10^{-5}
Weight Decay	0.1
LoRA Dropout	0.1
LR Scheduler Patience	3

4 Results

We evaluated the models on a 20% held-out test set. The DeBERTa-v3-Large model demonstrated superior performance across all metrics, validating the efficacy of its disentangled attention mechanism for understanding complex problem statements.

Table 2: Performance Comparison of Fine-Tuned Models

Model	Accuracy (%)	MAE	RMSE
Llama-3.2-3B (Linear Probe)	40.22	3.2800	3.9600
ModernBERT-Large	52.49	1.6600	2.0400
RoBERTa-Large	57.00	1.5950	1.9444
DeBERTa-v3-Large	**57.11**	**1.6300**	**1.9700**

(Note: Accuracy plots and training curves for the BERT-based models are attached below in Figure 1.)

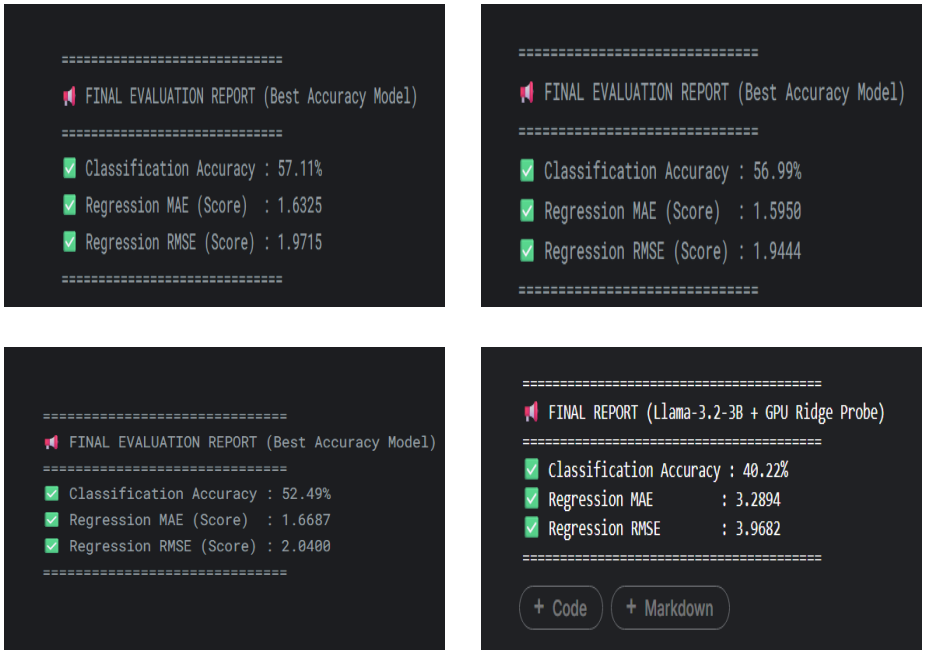

Figure 1: Training Loss and Accuracy Curves. Top Row: DeBERTa-v3 (left), RoBERTa-Large (right). Bottom Row: ModernBERT (left), LLM_probing_LLama-3.2-3b (right).

5 Web Interface and Deployment

To bridge the gap between model research and practical utility, we developed a responsive web application using **Streamlit**. The interface leverages Streamlit's reactive programming model to provide real-time complexity analysis.

5.1 Architecture and Workflow

The application backend loads the fine-tuned DeBERTa-v3 model weights using `st.cache_resource`. This ensures that the heavy model is loaded into GPU memory only once, enabling low-latency inference for subsequent queries.

- **Input Processing:** Users paste raw problem text into separate fields for Title, Description, and I/O constraints. The backend dynamically concatenates these inputs, mimicking the training data structure.
- **Tokenization:** The text is processed using the DeBERTa tokenizer (with a max length of 350 tokens) before being passed to the LoRA-adapted model.
- **Visualization:** The predicted difficulty class is displayed with color-coded alerts (Green for Easy, Yellow for Medium, Red for Hard), while the continuous complexity score is visualized on a normalized progress bar, offering immediate visual feedback.
- **Transparency:** An expandable "Technical Debug Info" section displays the raw logits and token counts, providing transparency into the model's decision-making process.

Deployment: The application is containerized and can be launched locally via a simple shell script (`streamlit run app.py`), making it easily deployable on local machines or cloud instances with GPU support.

A demonstration video showcasing the model approach and the working web UI with real-time predictions can be viewed here:

`[https://drive.google.com/file/d/1hLyrOIZDFwqaW6CssQRFXUfmJQw9xCtS/view?usp=drive_link]`.

6 Limitations

Despite achieving a significant improvement over baseline models, AutoJudge faces inherent limitations rooted in the complexity of algorithmic problem solving.

6.1 The Reasoning Gap

A critical bottleneck in automated judging is the gap between textual description and algorithmic depth. High-difficulty problems—specifically those rated 1600+ on platforms like Codeforces—often rely on obscure mathematical properties,

complex dynamic programming states, or specific graph theory observations that are not explicitly stated in the text [5].

For example, a problem statement might be short and syntactically simple (e.g., "Find the number of ways to tile a grid"), but the solution requires advanced combinatorics or matrix exponentiation. Our text-based regressors, including DeBERTa, primarily rely on semantic patterns and keywords. They lack the logical reasoning capabilities to "solve" the problem to determine its difficulty. Consequently, purely text-based regression hits a performance ceiling on high-rating problems where difficulty is derived from logical nuance rather than linguistic complexity.

6.2 Data Constraints

Our dataset consists of ≈4,112 samples. While sufficient for fine-tuning via LoRA, this is relatively small compared to the diversity of competitive programming tasks. Rare topics or novel problem types may not be well-represented, leading to lower confidence predictions on out-of-distribution tasks.

7 Conclusion

This project presented *AutoJudge*, a robust system for automating the difficulty assessment of programming tasks. Through a comparative study of modern encoder architectures, we demonstrated that **DeBERTa-v3-Large**, fine-tuned with LoRA, provides the most accurate complexity estimations, achieving a classification accuracy of 57.11% and a regression MAE of 1.6300.

Our results highlight a key finding: for specific discriminative tasks like complexity regression, efficiently fine-tuned encoder models significantly outperform general-purpose decoder LLMs (like Llama-3.2-3B) when prompted via linear probes. The accompanying web interface successfully operationalizes this model, offering a practical tool for educators and platform maintainers to categorize content efficiently. Future work will explore integrating code snippets (solution files) into the input context to help bridge the reasoning gap identified in high-difficulty problems.

References

[1] Hu, E. J., et al. (2021). LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint arXiv:2106.09685*.

[2] He, P., Gao, J., & Chen, W. (2021). DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing. *arXiv preprint arXiv:2111.09543*.

[3] Liu, Y., et al. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv preprint arXiv:1907.11692*.

[4] Touvron, H., et al. (2023). Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv preprint arXiv:2307.09288*.