CSC311: Introduction to Machine Learning

Project Report

Submitted by

Yuxuan Liu, Yi Pan, Ruoxin He , Geyu Liu

University of Toronto
Fall 2025

1 Executive Summary

This project evaluates three machine learning model families—Random Forests, Gradient Boosting (XGBoost), and a Neural Network—to classify survey responses describing experiences with ChatGPT, Claude, and Gemini. Among the three, XGBoost achieved the best performance, reaching a projected validation accuracy of approximately 0.7515. Its advantage comes from its strong ability to model non-linear interactions in our structured feature set while maintaining good generalization. Compared with Random Forests and the Neural Network, XGBoost produced more consistent performance across classes, making it the most robust and reliable model for this task, leading to a projected test accuracy of approximately 0.64.

2 Data Exploration

- ***Feature Summarization***
  The raw dataset contains three feature types: (1) four 1–5 Likert-scale rating questions, which behave as ordinal numerical variables and mostly cluster around values 3–4; (2) two multi-select categorical fields describing which tasks each model performs well or poorly, with writing, summarization, and coding tasks appearing most frequently; and (3) three free-text responses with substantial variation in length. The dataset also includes a label indicating whether the student evaluated ChatGPT, Claude, or Gemini, with the classes being roughly balanced. Overall, the data combines ordinal ratings, multi-hot categorical selections, and unstructured text, forming a heterogeneous feature set relevant for model development.

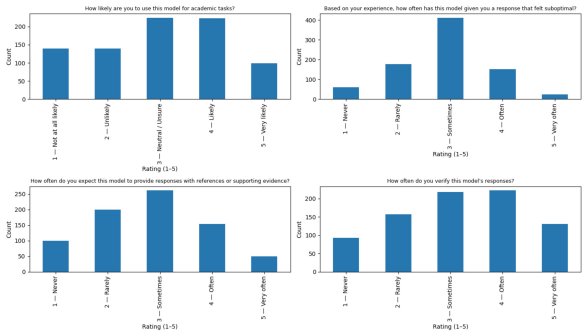

Figure 1: Rating Distributions

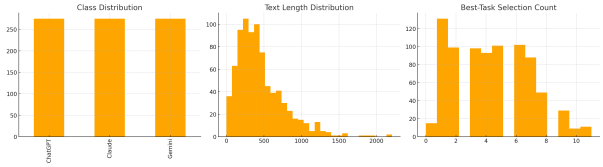

Figure 2: Class, Text Length, Best-Task Selection Count Distribution

- ***Data Issues***
  The preprocessing workflow addressed several inconsistencies in the raw data. For the four rating questions, descriptive strings such as "3 – Sometimes" were converted to their leading integer values, and missing entries were imputed using the column-wise mode. These ordinal ratings were then one-hot encoded. The two multi-select fields required additional cleaning because of commas inside parentheses; we applied a protected splitting procedure to correctly recover each selected

option and remove malformed tokens, producing reliable multi-hot vectors. For the free-text responses, we normalized the text through lowercasing, removal of non-alphanumeric characters, and whitespace compression. We then constructed five topic indicators by matching normalized text against predefined keyword lists. After inspecting rating distributions and text-length histograms, we did not observe extreme outliers requiring removal.

- ***Preprocessing Summary***
  These transformations resulted in a unified 33-dimensional feature representation composed of one-hot encoded rating indicators, multi-hot encodings of the "best task" selections, and binary topic features derived from the combined free-text fields. This standardized representation captures both structured and unstructured information in a form suitable for the model families explored in this project.

- ***Preventing Data Leakage***
  To prevent data leakage, we applied a grouped train–validation–test split based on student_id (60%/20%/20%). Because each student provides three highly related responses, grouping ensures that no student appears in multiple splits. Without this constraint, the model could memorize a student's writing style from the training set and gain advantage on validation or test samples. Keeping each student entirely within one partition ensures that our evaluation reflects true generalization.

3 Methodology

- ***Model Families***
  XGBoost: We included XGBoost because it handles our structured features very well and often gives strong performances. It builds trees one after another, with each tree trying to correct the errors of previous ones. This lets the model pick up patterns that simple trees might miss.

  Random forest: It is a bagging-based ensemble method that trains many decision trees on bootstrapped samples and random subsets of features. It fits our dataset well because it naturally models nonlinear interactions among the one-hot ratings, multi-hot task indicators, and topic features. Its randomness and averaging make the model robust to noise in student-written text, enabling stable performance without heavy tuning.

  Neural Network: We implemented a hybrid neural architecture that combines a pre-trained DistilBERT encoder with a small multilayer perceptron (MLP) for the tabular features. DistilBERT processes the concatenated free-text responses and produces a 768-dimensional embedding, while the tabular MLP projects the 33 structured features into a 128-dimensional representation; these two embeddings are concatenated and fed into a feed-forward classifier. We partially fine-tuned only the last several transformer layers while keeping earlier layers frozen to avoid over-

fitting on our relatively small dataset. This model family is appropriate for our data because it can directly exploit the semantic information in the free-text responses—something tree-based models cannot fully capture—while still incorporating one-hot and multi-hot structured features. The hybrid design allows the network to model interactions between textual cues and tabular signals, making it well suited for a dataset that mixes descriptive text with structured ratings and task indicators

- ***Validation Method***
  To ensure a fair comparison and prevent leakage, we used a grouped train/validation/test split based on the student_id column. THis guarantees that responses from the same student never appear in more than one split and thus does not leak across the train, validation and test. We used approximately 60 percent of the students for training, 20 percent for validation and 20 percent for testing. We only used the validation set to choose hyperparameters. The test set was kept separate and used later to check how well the final model generalizes.

- ***Optimization***
  XGBoost: We did not use optimizers like SGD or Adam because they are mainly used for neural networks. Instead we controlled how the model learns by tuning some key settings such as the learning rate, max depth and reg_lambda etc. These settings act like regularization and help prevent the trees from growing too large or too specific. We did not use early stopping so, we relied on a large search of 7500 different hyperparameters combinations to find the model that worked best on the validation set. THis was our main method for improving the model's performance.

  Random Forest: Random Forest does not use gradient-based optimization or learning rate schedules; instead, performance is controlled through structural regularization. The effective optimization consists of:Bootstrap sampling for each tree (bagging), providing the variance reduction effect.Random feature subsampling at each split.Tree-depth constraints that prevent overfitting. These mechanisms serve roles analogous to regularization in gradient-based models: Maximum depth limits model complexity, Minimum samples per leaf, Max features controls randomness and decorrelates trees(more trees reduce variance).

  Neural Network: For the Neural Network, we used the AdamW optimizer with separate learning rates: a low learning rate for the DistilBERT encoder (1e-5) and a higher learning rate for the tabular MLP and classifier (1e-3). This reflects the different learning dynamics of pre-trained transformers versus newly initialized layers. We applied dropout in both the tabular branch and classifier to reduce overfitting, and trained for up to 20 epochs with early stopping based on validation accuracy.

- ***Hyperparameter list, choices, tuning***
  XGBoost: We sampled 7,500 different combinations of hyperparameters from fixed ranges. For each trial, we trained the model on the training set and measured ac-

curacy on the validation set. The hyperparameters we varied were maximum tree depth, learning rate, number of trees, subsample rate, column sampling rate, L2 regularization strength, and minimum child weight. These ranges are standard for XGBoost and cover the most common settings while keeping the search manageable. After running all trials, we selected the model with the highest validation accuracy. We also recorded test accuracy for analysis, but we did not use the test set to pick hyperparameters. In addition to accuracy, we checked precision, recall, and macro F1 to make sure the model performed reasonably across all three classes.

Random Forest: We performed a structured grid search over four key hyperparameters that control tree depth, leaf-size regularization, feature subsampling, and ensemble size. Specifically, we varied the number of trees (100, 150, 200, 250), maximum depth (10, 15, 20, 25, 30 and the unconstrained option), minimum samples per leaf (1, 2, 4, 6, 8), and the number of features considered at each split ("sqrt", "log2", None, and 0.5). These ranges follow standard practice for Random Forest classifiers and were chosen to balance expressive capacity with computational feasibility. Each combination was trained on the training set and evaluated on the validation split, allowing us to select configurations that best trade off bias and variance. After completing all experiments, we select the hyperparameters that achieve the highest validation accuracy and evaluate this model only once on the test set.

Neural Network: For the Neural Network, we tuned a smaller but focused set of hyperparameters that meaningfully affect performance. These included the number of unfrozen transformer layers, the MLP hidden dimension (fixed at 128 after early experiments), dropout rate (0.4), learning rates for the encoder and classifier, and the maximum sequence length for tokenization. We evaluated each configuration by training on the training set and selecting the model with the highest validation accuracy. As with the other model families, the test set was never used for model selection. In addition to accuracy, we examined precision, recall, and macro-F1 to ensure that the neural network performed consistently across all three labels and was not overfitting to only one model type.

- ***Avoid winner-only tuning***
  To avoid the "winner-only tuning" mistake mentioned in the instructions, we tuned all three model families before choosing the final one. We did not pick a model first and only tuned that one. Instead we gave each model a fair chance by testing reasonable hyperparameter ranges for XGBoost, Random Forest, and our Neural Network. Only after comparing their validation performance did we select the final model. At the same time, all three models were trained and evaluated on the same 33 feature sets, which allowed us to compare them fairly.

- ***Evaluation metrics***
  We mainly used accuracy as our main evaluation metric because it is the simplest way to give us a clear sense of how often the model predicts the correct LLM. But accuracy alone doesn't show how well the model does on each class, so we also looked

at precision, recall and macro-F1. Some models performed better on one class than the others and these extra metrics helped us see that . Macro-F1 is especially useful because it gives equal weight to all three classes. By using both accuracy and additional metrics, we understand how each model actually works better.

- ***Additional techniques***
  Feature engineering played a major role in this project. First, we converted the four 1–5 rating questions into one-hot vectors, producing 20 features. Next, we processed the two multi-select questions ("best tasks" and "suboptimal tasks") by splitting and cleaning the text and mapping the eight possible options into binary columns, producing multi-hot vectors. Because the "suboptimal tasks" features added noise and did not improve validation accuracy, we kept only the eight "best tasks" features in the final representation. Finally, we combined and cleaned the three free-text responses and extracted five topic indicators (code, math, writing, research, and chat) by checking for predefined keywords. These steps yielded the final 33-dimensional feature set.

- ***Implementation details***
  We implemented the neural network using PyTorch and Hugging-Face Transformers. DistilBERT was loaded through AutoTokenizer and AutoModel, and a small MLP processed the 33 tabular features. For XGBoost, we used the standard xgboost library and exported the trained model with get_booster().get_dump(). For the Random Forest model, we used the sklearn.ensemble.RandomForestClassifier implementation. We also wrote custom code to export the trained Random Forest into a JSON format, including tree structures, split rules, thresholds, and class counts pred.py could be performed with pure NumPy.

4 Results

- ***Overall Model Performance***
  Table 1 reports validation accuracy, precision, recall, and macro-F1 for the three model families. XGBoost achieved the highest validation accuracy (0.7515), followed by Random Forest (0.7394) and the Neural Network (0.6788).

Table 1: Validation Performance Across Model Families

Model	Accuracy	Precision	Recall	Macro-F1
XGBoost	0.7515	0.7599	0.7515	0.7495
Random Forest	0.7394	0.7507	0.7394	0.7339
Neural Network	0.6788	0.6787	0.6788	0.6753

- ***Confusion Matrix and Error Analysis***
  Table 2 shows the confusion matrix for the final XGBoost model on the test set (165 samples). Claude and Gemini are the most frequently confused pair, indicating overlap in task-selection patterns and topic keywords. Claude has one of the lowest

recalls (tied with Gemini), making it one of the hardest classes to classify. Its responses tend to resemble both ChatGPT and Gemini in ratings and topics, causing the model to misclassify a large portion of Claude samples.

Table 2: Confusion Matrix for Final XGBoost Model on Test Set

	Pred: ChatGPT	**Pred: Claude**	**Pred: Gemini**
True: ChatGPT	45	4	6
True: Claude	7	30	18
True: Gemini	8	17	30

- ***Comparison Across Model Families***
  XGBoost outperformed both Random Forest and the Neural Network on both validation and test sets. Random Forest performed reasonably well but lacked a boosting mechanism to correct systematic errors. The Neural Network underperformed due to limited dataset size and highly variable text quality.

- ***Estimated Test Performance***
  Table 3 shows the test accuracy for the three model families. XGBoost achieved a test accuracy of 0.6364. Random Forest and the Neural Network reached 0.6061 and 0.6303 respectively.

- ***Justification of Performance Estimate***
  We used a held-out test split where the students never appear in the training or validation sets. This prevents leakage and makes the test accuracy a fair measure of how the model performs on new users. All hyperparameter tuning was done using only the validation set.

  XGBoost reached 0.7515 accuracy on the validation split and 0.6364 on the test split. The drop is expected because the test set contains different students, but the two numbers are still close enough to be consistent. This suggests that the model's performance is not tied to a particular split and that the test result is not unusually high or low. Since the training and validation results were also fairly close, we treat 0.6364 as a reasonable estimate of how well the model performs on new data.

5 Contributions and Learning

- ***Ruoxin He***
  I contributed by implementing and coding part of the Neural Network model, including components of the hybrid architecture and the training pipeline. I also wrote major portions of the report, specifically the data exploration, results analysis, and Neural Network sections of the methodology. Through this work, I gained experience in model construction, data preprocessing, and presenting technical findings clearly.

- ***Geyu Liu***
  I implemented the hybrid neural network model by integrating the BERT-based text encoder with the tabular MLP into a unified training pipeline. I learned how to fine-tune DistilBERT safely on a small dataset by freezing early layers, adjusting learning rates, and using dropout and early stopping to prevent overfitting. This helped me understand how semantic text embeddings and structured features interact in hybrid architectures.

- ***Yuxuan Liu***
  I built the full preprocessing pipeline that transformed the raw survey data into the final 33-feature representation and experimented with several feature-set variants. I also trained the XGBoost models, implemented the grouped train/validation/test split, and carried out the hyperparameter search. Through this process, I learned how stable validation splits and careful feature engineering strongly affect model performance and generalization.

- ***Yi Pan***
  I implemented and trained the Random Forest model, conducted the grid-search hyperparameter tuning and generated evaluation plots. I also verified the experiment outputs and compared with other model families to ensure consistency for the final report. I learned how proper validation-based tuning and grouped data splitting prevent data leakage and significantly improve model reliability.

References

[1] XGBoost Developers. Xgboost documentation. `https://xgboost.readthedocs.io/`.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

[3] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

[4] R. Tang, Y. Lu, L. Liu, L. Mou, O. Vechtomova, and J. Lin. Distilling task-specific knowledge from bert into simple neural networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

[1] [3] [4] [2]