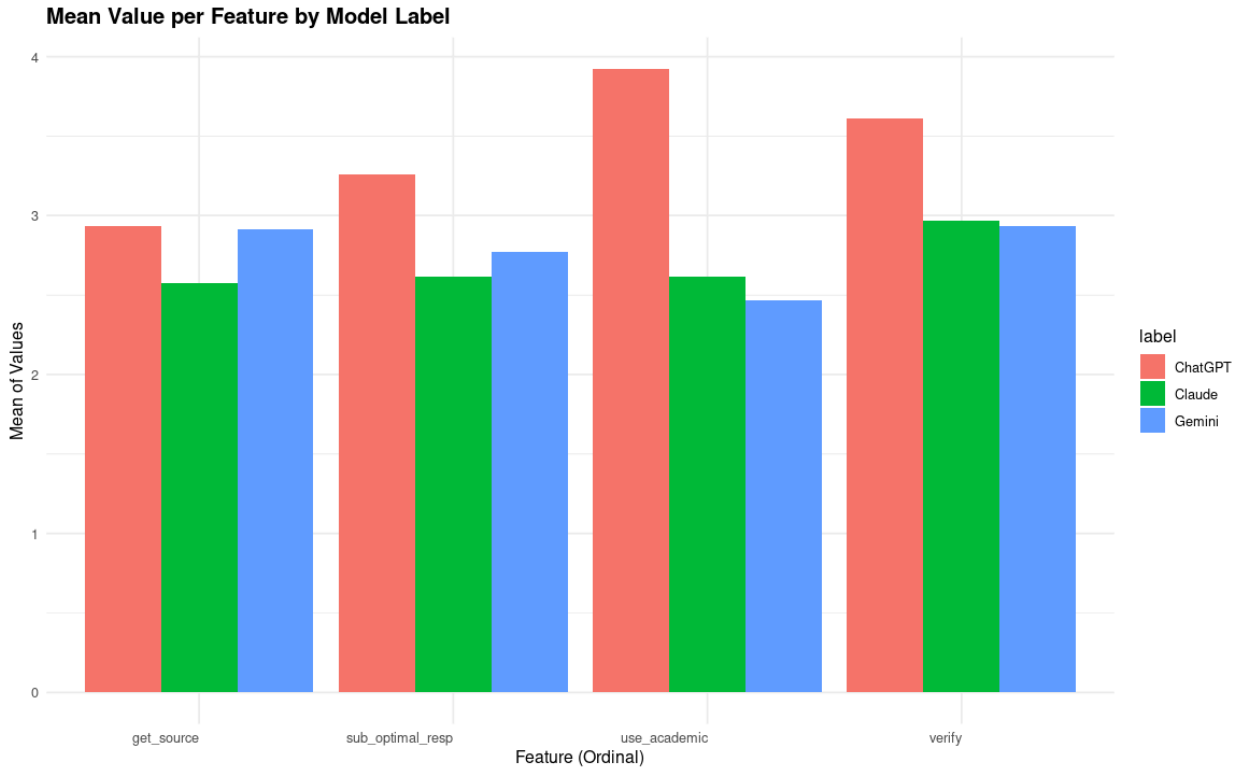
# Project Proposal

## Data Exploration

Aside from the student id column (which we will not consider a feature,) our features compromise text, categorical data, and ordinal data. The distribution of data is given by the following picture.




We have 275 observations for each model. There are some missing data points for the ordinal data for labels with Gemini and Claude. The text responses are filled with no responses and questionable AI responses which will prove to be a problem to deal with. Notably, some students answered “The same as the other model” for some AI responses.

All text data must first be categorized and then encoded to one hot vector as they should have no ordering amongst them. Categorizing is done for us for the question where students pick from fixed choices. For free response data, we are going to look for certain keywords; for example, if the text response contains “math,” then this response will be classified under the category “math,” and so on. The words by which we will categorise with is certainly a choice that has to be made and has no correct decision; however, we are trying to be as accurate as possible. So far, categories will grab responses with “math”, “coding” or “programming” or…, “summary” or “summarise” or…, “academic”, “email” or “resume” or…, and more. At this point, normalisation does not seem necessary.

We will prevent data leakage by sampling student ids. Since a student's response for one AI model may give us an idea for how they would respond for different AI models, our model can peek into the test data using other model responses. Also, this way, we make sure our model is trained on an equal number of ChatGPT, Gemini and Claude labels. However, our method for preventing leakage is not set in stone yet.

# Planned Methodology

### Model Families & Reasons:

The three model families we plan to explore are KNN, Decision Tree, and Neural Networks.

Why?

**KNN**: We decided to include KNN as one of our models because it is simple and works well as a baseline. It classifies each response by looking at its closest examples in the training set. This helps us see how much the answers group naturally based on similarity. Although KNN can be less reliable for high-dimensional text data, it is easy to implement and gives us a good reference point before testing more complex models.

**Decision Tree**: Decision Trees learn simple if-then rules directly from the data, which makes them easy to understand. They work well for our dataset because it includes both survey-style answers and open-ended text. The model can automatically figure out which features, such as certain words or response patterns, best separate ChatGPT, Claude, and Gemini. This helps us see what makes each model's responses different.

**Neural Networks**: For Neural Network, we plan to use a simple feed-forward neural network with one hidden layer. This setup keeps the model easy to train but still allows it to learn patterns that simpler models might miss. Since our dataset includes open-ended text responses and survey-style answers, a neural network is a good fit because it can combine different types of features and capture complex, non-linear relationships between them. A single hidden layer is usually enough when the dataset is not very large.

### The optimization technique(s) we expect to use:

**KNN**: KNN does not use an optimization algorithm since it does not learn weights from the data. Instead, we will tune the number of neighbours k using validation accuracy. We might also try different distance measures like Euclidean or cosine to see which one fits our data better.

**Decision Tree**: For the Decision Tree, we will adjust parameters like maximum depth and minimum samples per split to prevent overfitting. The model will be trained using a greedy algorithm that selects the best feature/split at each step based on information gain.

**Neural Network**: For the Neural Network, we will train the model using the Adam optimizer, which adapts the learning rate during training. We will try a few different learning rates and use techniques such as early stopping to reduce overfitting. The training will run for several epochs, and we will keep the version that performs best on the validation set.

## The validation method we will adopt:

We will employ a straightforward hold-out validation method to evaluate and tune our models. The dataset will be split into three subsets: a training set (70%) for model learning, a validation set (15%) for hyperparameter tuning, and a test set (15%) for the final performance evaluation. This simple approach ensures that we can reliably compare our models' ability to generalize to unseen data. We will not use cross-validation, since it is too computationally expensive for large datasets.

## A list of hyperparameters we plan to tune:

KNN:

| Hyper-parameter | Ranges of values | Justification |
|---|---|---|
| k | 1, 3, 5, 7, 9, 11, 15, 25 | **Small k:** memorizes phrasing (risk of overfitting to specific students).<br>**Larger k:** smoothes across wording variants.<br>**Odd k:** lowers a tie risk for 3 classes. |
| weights | 'uniform', 'distance' | **Uniform:** each neighbor votes equally: faster, less parameter sensitivity.<br>**Distance:** closer neighbors get higher weight; could help when classes share vocabulary. |
| distance metric | cosine, euclidean, manhattan | **Cosine**: How similar the mix of words is between answers, ignores answer length<br>**Euclidean**: Overall difference between answers and highly sensitive to length differences<br>**Manhattan**: Sums small word differences between answers, may capture if categories are separated by many small cues |

Decision Tree:

| Hyper-parameter | Ranges of values | Justification |
|---|---|---|

| | | |
|---|---|---|
| criterion | gini, entropy | We'll compare both impurity reduction and information gain metrics to see which performs better. Entropy may overfit and 'chase' niche words |
| max_depth | 2, 5, 10, 20 | Deeper trees fit more specific rules but can overfit. We will cap at the smallest depth where validation stops improving. |
| min_samples_split | 2, 5, 10, 20 | Larger values force the tree to split only when there's enough data. We plan to increase it if we see many fragile, low-support splits on niche words; decrease it if the tree is too conservative and underfits. |
| min_samples_leaf | 1, 2, 5, 10 | Enforces minimum samples at the leaves so the tree doesn't create single-example rules. Larger leaves smooth predictions and reduce variance. Similar to min_samples_split, we will increase if overfit and decrease if underfit. |

Neural Network:

| **Hyper-parameter** | **Ranges of values** | **Justification** |
|---|---|---|
| n_hidden_layers | 1, 2 | We'll compare a single hidden layer to two hidden layers. One hidden layer already gives non-linear feature learning; two layers may capture more complex interactions without hand-crafting features. Tune to 1 if the validation set doesn't benefit from 2. |
| hidden_units_per_layer | 64, 128, 256 | We'll try widths that are large enough to learn useful features but small enough to avoid overfitting. We'll choose the smallest size that performs well on validation. |
| activation | sigmoid, tanh, ReLU | **ReLU** as a strong default (non-saturating), **tanh** for zero-centered signals, and **sigmoid** as a classical baseline. We'll keep the best performing measure. |
| learning_rate_init | 1e-3, 3e-3, 1e-4 | Set the initial step size. We'll raise it if convergence is very slow and lower it if loss oscillates or diverges. |
| batch_size | 64, 128, 256 | Smaller batches add regularization but can be slow, while larger batches are faster but may overfit. We'll pick the smallest batch that trains stably within time limits. |
| max_iter | 100, 200, 400 | Caps training epochs. We'll use more iterations only if validation keeps improving. |

## The evaluation metrics we will report.

- At least one metric should go beyond accuracy, with justification:

We will use accuracy as our primary performance metric, as it provides an easily interpretable measure of overall success. However, we recognize that accuracy alone offers a limited view, potentially masking significant model weaknesses. For instance, a model could achieve high overall accuracy while performing poorly at identifying a specific, less common chatbot. Hence, we will conduct an analysis using the confusion matrix. This involves calculating precision and

recall for each AI chatbot. Also, we will compute Type I error rates and Type II error rates respectively. By analyzing these patterns, we can identify model weaknesses and select a final model that is not only accurate but also robust and fair across all target classes.