**Project Proposal: Semantic Classification of Math Problems Using NLP**

**Team:** Deepika Reddygari, Phanindra Kumar Kalaga, Prudhvi Chekuri

**Motivation:**While discussing study challenges, one of our teammates recalled the difficulty their younger sibling faced when preparing for a math test. Despite having access to dozens  
of resources, there was no easy way to filter problems by topic - Algebra, Geometry, or Calculus. That observation quickly resonated with all of us.

It made us wonder: Could we build a system that understands the language of math problems and classifies them just like a teacher would?  
This project is our attempt to explore that question using Natural Language Processing.

**Problem Statement:**

We aim to build a model that classifies word-based math problems into one of eight mathematical domains: Algebra, Arithmetic, Calculus, Geometry, Number Theory, Probability, Statistics, and Trigonometry.

Our goal is to use the foundational and state-of-the-art methods from class and contribute to improving domain-specific (Maths) natural language processing problems.

**Dataset and Relevance:**

We'll use the dataset provided in the [Kaggle Competition: Classification of Math Problems by Kaust Academy](https://www.kaggle.com/competitions/classification-of-math-problems-by-kasut-academy/overview).

The dataset includes labeled examples of real-world math problems, offering rich, diverse text for training and testing a robust classification model. Its structure and balance across topics make it an ideal fit for this type of semantic analysis.

**NLP Tasks:**

To begin, we’ll implement a classical machine learning pipeline using TF-IDF + Logistic Regression or SVM to set a meaningful baseline. From there, we will fine-tune modern transformer-based models, such as BERT or DeBERTa - to capture deeper semantic patterns and context within problem statements.

If time allows, we may also experiment with lightweight custom architectures or prompt-based methods.

**Tools and Packages:**

Our project will rely on:

* Hugging Face Transformers – for easy access to pretrained language models
* PyTorch – for model training and experimentation
* Scikit-learn – for classical baselines, metrics, and preprocessing
* Pandas and NumPy – for efficient data handling

These tools give us flexibility for rapid prototyping, scaling, and comparative analysis.

**NLP Tasks We'll Tackle:**

The primary task is multi-class text classification, with supporting tasks that include:

* **Tokenization** using Hugging Face’s AutoTokenizer tailored for MathBERT
* **Contextual Embedding Generation** using AutoModelForSequenceClassification
* **Fine-tuning** the transformer model with Trainer and TrainingArguments API for efficient training and evaluation
* **Stratified K-Fold Cross-Validation** to improve robustness and reduce overfitting
* **Dataset Wrapping and Label Encoding** for compatibility with the Hugging Face Trainer

**Evaluation:**

We will evaluate our model using the macro-averaged F1-score, which is especially important given that the dataset contains multiple classes with varying frequencies. Macro-F1 ensures that underrepresented classes are treated with equal importance as the more common ones.

* **Validation across Stratified K-Folds** to maintain balanced class distribution across training and validation sets
* **Per-class performance breakdown** via confusion matrices (optional) to understand misclassifications
* **Final selection of best-performing fold based on F1-score** for model checkpointing

These metrics will help us optimize not only for correctness but also for educational value.

**Tentative Project Timeline:**

|  |  |
| --- | --- |
| April 15–17 | Divide dataset tasks, clean and explore the data, implement classical models |
| April 18–22 | Fine-tune transformer models and analyze performance |
| April 23–25 | Conduct error analysis, finalize experiments, tune hyperparameters |
| April 25–29 | Draft final report, create visualizations, and prepare for submission |

We’ll collaborate closely throughout using GitHub and shared notebooks to divide the workload efficiently.