GPT-4's Search for K3 Fibrations in Heterotic/Type IIA String Duality

Aaron Vermeersch[*]

The development of Large Lanaguage Models, such as the recently released GPT-4, has revolutionized the field of Machine Learning and opened new avenues for interdisciplinary research. Prompt Engineering, a methodology for designing effective input prompts to guide these AI models, will emerge as a powerful tool for accelerating research efforts. In this study, we leverage Prompt Engineering with GPT-4 to address the problem of predicting $K3$ Fibrations in Calabi-Yau manifolds embedded in toric varieties with a single weight system. Out of the 184,026 spaces previously discovered, 101,495 remained unclassified concerning the presence of a $K3$ projection. By utilizing an ensemble of Deep Neural Networks, we predict that 16,531 of these previously unclassified spaces possess a $K3$ projection, 83,498 as having two $K3$ projections, and 1,466 as containing three $K3$ projections. Lastly, we evaluate the ability of these models to predict the complete spectrum of possible Hodge Numbers and other properties of reflexive polytopes. These results should be useful in heterotic/Type IIA (F-theory) duality.

1 Introduction

The emergence of Large Language Models (LLMs), exemplified by the groundbreaking GPT-4 [1], has revolutionized the field of Natural Language Processing and unlocked new avenues for interdisciplinary research. A key technique that has evolved alongside these AI models is Prompt Engineering, which centers on crafting effective input prompts that guide LLMs towards specific outputs. Here, we investigate the application of GPT-4 in accelerating software development related to fundamental physics research, by using Deep Learning methods to investigate $K3$ fibered Calabi-Yau manifolds embedded in toric varieties with single weight systems.

$K3$ fibrations become important when studying the dualties between the $E_8 \times E_8$ heterotic string compactified on $K3 \times T^2$ and the Type IIA string compactified on a Calabi-Yau manifold. In [2], it was noticed that the Calabi-Yau manifold on which the Type IIA string was compactified can be written as a fiber bundle with base $\mathbb{P}^1$ and a $K3$ surface as a generic fiber. The heterotic string was compactified on $K3 \times T^2$ which can be constructed as a bundle with base $\mathbb{P}^1$ and generic fiber T^4. $K3$ fibrations were then shown to fit well with the properties of the dual heterotic string compactified on $K3 \times T^2$ [3]. The duality was made explicit in [4] whereby assuming the Type IIA dual of the weakly coupled heterotic string is in the Calabi-Yau phase, then this Calabi-Yau manifold must be a $K3$ fibration. Calabi-Yau manifolds that could be embedded in toric varieties were developed in [5], and then in [6] the properties of these spaces were investigated, including

---

[*] aaronvermeersch@gmail.com

whether they contained a $K3$ fibration. More information on $K3$ fibrations can be found in [7] [8] and on their appearance in string theory [9].

In recent years, Machine Learning has become important in exploring the vast landscape of string theory vacua. Machine learning models, in the form of Support Vector Machines, have been employed to predict various properties of Calabi-Yau manifolds, such as Hodge Numbers [10], leading to advancements in our understanding of these structures. Furthermore, Deep Learning has been applied to investigate a multitude of topics [11] pertaining to string theory including brane webs [12], topological properties of Calabi-Yau manifolds in F-theory [13], the string vacua in heterotic string theory [14], and others [15].

In this study, we harness the power of GPT-4 for the development of our Deep Learning software, which is employed to identify K3 fibrations in Calabi-Yau manifolds embedded in toric varieties. We use an ensemble of Deep Neural Networks and the weight system of the space to predict K3 fibrations in 101,495 previously unclassified spaces. The remainder of this paper is organized as follows: Section 2 offers an overview of how from a reflexive polytope a Calabi-Yau manifold can be constructed and embedded in toric varieties and that this can produce a $K3$ fibration. Section 3 delves into the impact of Prompt Engineering in this work and the Deep Learning methodology employed in our analysis, while Section 4 presents our results. Finally, in Section 5, we suggest potential avenues for future research.

2 $K3$-fibered CY manifolds & Reflexive Polytopes

In this section, we explore how from a polytope, a toric variety can be constructed and if the polytope is reflexive then certain hypersurfaces of this space describe Calabi-Yau manifolds. Furthermore, by restricting ourselves to four dimensions, any reflexive polytope with a projection onto a 3 dimensional sublattice which produces a polyhedron will come to define a $K3$ fibration.

Let $M, N \simeq \mathbb{Z}^n$ be two dual lattices with their respective real extensions as $M_{\mathbb{R}}$ and $N_{\mathbb{R}}$, and a fan Σ defined on $N_{\mathbb{R}}$. A lattice polytope Δ is a polytope such that the vertices of $\Delta \subset M_{\mathbb{R}}$ lie on the lattice points. Δ is considered reflexive if it, and its dual defined as

$$\Delta^* = \{y \in N_{\mathbb{R}} : \langle x, y\rangle \geq -1, \forall x \in \Delta\} \tag{1}$$

contains only one interior point at the origin. From here, we follow the holomorphic quotient approach [16] when constructing our toric variety V_{Σ}. First, assign coordinates $z_1, \ldots, z_N$ to each one-dimensional cone in Σ and let the fan Σ be based by a maximal triangulation of Δ^*. The generators $\nu_1, \ldots, \nu_N$ of the one-dimensional cones are represented by the integer points of Δ^*. From the resulting $\mathbb{C}^N$ we remove the exceptional set

$$Z_{\Sigma} = \bigcup_{I} \{(z_1, \ldots, z_N) : z_i = 0, \ \forall i \in I\} \tag{2}$$

where the union is taken over all sets I for which the generators do not belong in a cone in Σ. The toric variety [17] is then given by the quotient of by a group which is a product of a finite abelian group and $(\mathbb{C}^*)^{N-n}$ acting by

$$(z_1, \ldots, z_N) \sim (\lambda_{z_1}^{w_j^1}, \ldots, \lambda_{z_N}^{w_j^N}) \quad \text{if} \quad \sum_k w_j^k x^k = 0 \tag{3}$$

Consequently, a Calabi-Yau hypersurface in V_Σ is given by the zero locus of

$$P(z_1, \ldots, z_N) = \sum_{x \in \Delta \cap M} a_x \prod_{k=1}^{N} z_k^{<v_k, x>+1} \tag{4}$$

with the Hodge numbers h_{11} and h_{21}, for the above Calabi-Yau hypersurface, given by Batyrev [18]:

$$h_{11} = l(\Delta^*) - 5 - \sum_{codim\theta^*=1} l^*(\theta^*) + \sum_{codim\theta^*=2} l^*(\theta^*) l^*(\theta) \tag{5}$$

$$h_{21} = l(\Delta) - 5 - \sum_{codim\theta=1} l^*(\theta) + \sum_{codim\theta=2} l^*(\theta^*) l^*(\theta) \tag{6}$$

Here, θ and θ^* represent faces of Δ and Δ^*, respectively. The functions l and l^* denote the numbers of integer points and integer interior points within the polytopes. Notably, these formulas remain invariant under the exchange of Δ with Δ^* and h_{11} with h_{21}, highlighting the mirror symmetry in Batyrev's construction.

The fibration structure is described by the pair $(\Delta, \Delta^*) \subset M \times N$ of reflexive polytopes where N has a distinguished three dimensional sub-lattice N_3 such that Δ_3^* is a three dimensional reflexive polytope. The sub-polytope Δ_3^* corresponds to the fiber and divides Δ^* into a top and a bottom. The fan corresponding to the base space can be obtained by projecting the fan of the fibration along the linear space spanned by the sub-polytope describing the fiber. In our case, Δ^* is a four dimensional polytope so that the base space is always $\mathbb{P}^1$ [19].

The algorithm that generated the $K3$ fibered toric CY data [1] consisted of constructing a set S of maximal polytopes such that any reflexive polytope is a subpolytope of a polytope in S [20]. These maximal polytopes are defined by a weight system as a set of positive rational numbers $q_i = w_i/d$, where $\sum q_i = 1$ and $\sum w_i = d$. For the present case, we consider only the case of a single weight system defined by 5 weights. These weights, $(w_1, \ldots, w_5)$, define a sublattice Γ^4 within the lattice $\Gamma^5 \simeq \mathbb{Z}^5$, as:

$$\Gamma^4 = \{(x^i) \in \Gamma^5 : \sum w_i x^i = d\}. \tag{7}$$

Then, given S, construct all subpolyhedra of all polyhedra in S and check for reflexivity [11]. The 184,026 weight systems related to 4 dimension polytopes was originally calculated in [21] while in [6], these weight systems were returned to in order to determine the existence of the $K3$ fibrations.

3 Artifical Intelligence

3.1 GPT-4

GPT-4 (Generative Pre-trained Transformer 4) is a state-of-the-art language model developed by OpenAI, building upon the success of its predecessor, GPT-3 [22], and offering substantial improvements in scale, capabilities, and performance [23]. The transformer architecture underpinning GPT-4 leverages self-attention mechanisms [24] to process input sequences in parallel rather than sequentially, enabling the model to effectively capture long-range dependencies and intricate patterns in text data.

---

[1] The dataset used for this study can be found at Kreuzer and Sharke's website.

The training methodology for GPT-4 entails unsupervised pre-training on an extensive corpus of text data, succeeded by task-specific fine-tuning. Pre-training fosters a general language understanding through next-word prediction, whereas fine-tuning adapts the model to designated tasks by minimizing the loss on labeled data.

Prompt Engineering [2] is a technique that optimizes input format and context for models like GPT-4, which are capable of generalizing from minimal input in one-shot or few-shot learning scenarios [22]. This approach facilitates more accurate and relevant responses to specific queries or tasks.

A key application of Prompt Engineering lies in expediting software development. GPT-4 can generate high-quality code snippets, identify and rectify errors, and provide succinct explanations of complex code structures when given effective prompts. This process not only saves time but also alleviates cognitive burden on developers, enabling them to concentrate on more crucial aspects of software development. See the Appendix for specific instances of Prompt Engineering that were used in this work.

Furthermore, Prompt Engineering in software development carries significant implications for fundamental research. Automating mundane and repetitive tasks allows researchers to dedicate more time and resources to exploring novel ideas and innovative methodologies. GPT-4's capacity to comprehend and generate human-like text empowers researchers to efficiently extract insights from copious volumes of literature, thereby streamlining knowledge discovery. Consequently, Prompt Engineering and GPT-4 hold the potential to substantially accelerate scientific progress and contribute to the broader advancement of human knowledge.

3.2 Feature Engineering

To optimize the performance of our machine learning models, we employed a variety of preprocessing techniques to both input and output variables in the dataset, with the aim of transforming the data into an appropriate format and scale that would facilitate the learning process and improve model accuracy.

For input variables to the models, specifically the weight system, standard scaling was applied. Standard scaling is a normalization technique whereby the features are made to have zero mean and unit variance by subtracting the mean value and dividing by the standard deviation for each feature. Employing standard scaling is particularly advantageous when dealing with features of differing scales, as it ensures that no single feature disproportionately influences the model's learning process due to its magnitude.

The formula for standard scaling is as follows:

$$z = \frac{x - \mu}{\sigma} \tag{8}$$

where x represents the original feature value, μ is the mean of the feature, σ is the standard deviation, and z is the standardized value.

For output variables, we implemented distinct preprocessing techniques contingent upon the nature of the task. In tasks involving the prediction of categorical variables, such as the number of K3 projections, one-hot encoding was employed to transform the output variable. One-hot encoding is a technique that converts a categorical variable into a binary vector in which each category is represented by a unique binary sequence

---

[2] To obtain useful guidance on enhancing the quality of your prompts, please refer to the documentation provided by OpenAI onTechniques to Improve Reliability.

with a single “1” in the position corresponding to the category and “0”s elsewhere. This representation enables the model to discern distinct patterns for each category without presupposing any ordinal relationship between them.

3.3 Deep Learning Models

Inspired by the structure and functionality of biological neural networks, neural networks serve as powerful computational models designed to recognize patterns and representations within large, complex datasets. A common architecture for neural networks consists of densely connected layers, also known as fully connected or dense layers. In these layers, each neuron connects to every neuron in adjacent layers, enabling the network to learn intricate patterns from the input data[3].

Activation functions introduce non-linearity into neural networks, enabling them to model complex relationships between inputs and outputs. In our work, we utilize the Rectified Linear Unit (ReLU) as the activation function for the hidden layers. ReLU, defined as $f(x) = max(0, x)$ where x represents the input value, is computationally efficient and simple, making it popular in deep learning models.

In classification problems, the last layer of the neural network employs a softmax function. The softmax function, defined as

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{9}$$

for $i = 1, \ldots, K$, where x_i are the input values and K is the number of classes, converts the raw output values into probabilities representing the likelihood of each input belonging to a specific class. By using the softmax function, we can interpret the output as class probabilities, aiding decision-making and understanding the model’s confidence level.

The Adam optimizer, a popular optimization algorithm for training deep learning models, combines the advantages of Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp) to compute adaptive learning rates for individual parameters based on the first and second moments of the gradients. The learning rate, a crucial hyperparameter, determines the step size for updating model parameters during each training iteration. Adam’s ability to automatically adjust the learning rate for each parameter throughout the training process reduces manual hyperparameter tuning and makes it well-suited for handling noisy, sparse gradients which are common challenges in deep learning. In this work, all of the neural networks were trained using an Adam optimizer.

To further refine our model, we employed early stopping, a regularization technique that mitigates overfitting in deep learning and neural networks. Overfitting arises when a model captures noise or random fluctuations instead of the underlying patterns, impairing generalization to unseen data. Early stopping monitors the model’s performance on both the training and validation sets, ceasing the training process when validation performance deteriorates, typically after a predetermined number of epochs without improvement. Early stopping curtails overfitting, conserves computational resources, and streamlines model selection, culminating in superior models that generalize well to new data.

Dropout was also employed as a regularization technique to further mitigate overfitting in the neural networks. Dropout randomly ”drops out” a certain percentage of neurons

---

[3] For additional reading, see [25] [26] [27].

during training, preventing the model from relying too heavily on individual neurons and promoting the development of robust, generalizable features. In our work, dropout follows each densely connected layer, with varying dropout rates determined through hyperparameter tuning.

Training a neural network requires defining a loss function to measure the divergence between predicted outputs and true target values. Mean Squared Error (MSE) and Categorical Cross-Entropy (CCE) are two commonly used loss functions for regression tasks with discrete target values and classification tasks, respectively.

MSE is a widely used loss function for regression tasks involving discrete target values. It calculates the average squared difference between predicted and actual values, as illustrated in the following equation:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_{\text{true}}^{(i)} - y_{\text{pred}}^{(i)})^2 \tag{10}$$

Here, n signifies the number of samples, y_{true} corresponds to the true target values, and y_{pred} denotes the predicted values.

For multi-class classification tasks, where the objective is to predict one of several potential categories, Categorical Cross-Entropy is a widely utilized loss function. It quantifies the dissimilarity between the true probability distribution and the predicted probability distribution across the classes, as demonstrated in the following formula:

$$CCE = -\sum_{i=1}^{n}(y_{\text{true}}^{i}\log(y_{\text{pred}}^{i})) \tag{11}$$

To evaluate the performance of a neural network, distinct metrics are employed for regression tasks with discrete target values and classification tasks. For regression tasks, which entail predicting discrete target values, Mean Absolute Error (MAE) serves as an evaluation metric, calculating the average absolute difference between predicted and actual values:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_{\text{true}}^{i} - y_{\text{pred}}^{i}| \tag{12}$$

In classification tasks, which involve predicting categorical variables, accuracy and F1-score serve as evaluation metrics. Accuracy represents the proportion of correctly classified instances out of the total instances. The F1-score, on the other hand, is the harmonic mean of precision and recall, calculated as follows:

$$F1\text{-}score = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{13}$$

Precision is defined as the proportion of true positive predictions out of the total predicted positives, while recall is the proportion of true positives out of the total actual positives.

A confusion matrix is another valuable tool for assessing a classifier's performance. This matrix provides a visual representation of a classifier's accuracy and robustness on labeled data. Each row corresponds to the true class labels, and each column represents the predicted class labels. The cell at the intersection of a specific row and column displays the number of instances belonging to the true class (row) and predicted as the corresponding class (column). The matrix's diagonal elements indicate the correctly classified instances, while the off-diagonal elements denote misclassifications.

To optimize our neural network's performance, we executed hyperparameter optimization by identifying the most favorable combination of hyperparameters that minimized validation loss. We conducted 50 trials, each employing a distinct set of hyperparameters, and selected the configuration yielding the lowest validation loss. The chosen hyperparameters were subsequently utilized to train the final model. Hyperparameters considered in our search encompassed the number of neurons in each layer, the degree of dropout preceding each layer, and the number of layers, which ranged from 1 to 4.

The hyperparameter space was explored using a random algorithm, whereby a random combination of hyperparameters was selected, the model trained, and subsequently evaluated. The best-performing model was then chosen for the ensemble method which would constitute our final result.

Property	Neurons/Layer	Dropout Per Layer	Learning Rate
R: # of $K3$ Projections	128, 128, 192, 224	0.4, 0.1, 0.4, 0.2	3.168E-4
C: # of $K3$ Projections	256, 192, 64, 256	0.5, 0.1, 0.2, 0.3	7.371E-4
R: # of Proj. onto $K3$ Facet	64, 224, 256, 192	0.1, 0.5, 0.1, 0.4	1.966E-4
C: # of Proj. onto $K3$ Facet	160, 224, 32, 256	0.1, 0.3, 0.3, 0.2	1.915E-4
R: Existence of 1st K3 Facet	192, 160, 192, 64	0.1, 0.3, 0.5, 0.5	1.350E-4
C: Existence of 1st K3 Facet	192, 256, 128, 224	0.1, 0.1, 0.2, 0.3	7.396E-5
R: Existence of 2nd K3 Facet	128, 256, 128	0.2, 0.3, 0.3	1.257E-4
C: Existence of 2nd K3 Facet	192, 224, 32	0.2, 0.5, 0.3	1.172E-4
R: h_{11}	96, 128	0.2, 0.2	7.196E-3
C: h_{11}	128, 160, 192, 224	0.3, 0.3, 0.2, 0.4	3.040E-4
R: $h_{21} \leq 200$	224, 224	0.3, 0.3	2.723E-4
C: $h_{21} \leq 200$	128, 224, 224	0.2, 0.3, 0.1	7.018E-4

Table 1: Optimal hyperparameters for each neural network model. The table presents the best-performing configuration of neurons per layer, dropout rate per layer, and learning rate for each model, identified from a total of 600 trained networks. Regression models are indicated by "R" and classification models by "C".

To further enhance the predictive accuracy and stability of our model, we incorporated Bootstrap Aggregating, an ensemble method commonly referred to as bagging, to amplify predictive accuracy and stability. Bagging entails training multiple base models (5 estimators in our case) on distinct subsets of the training data, sampled with replacement. The final prediction is achieved by averaging individual model predictions for regression tasks or by majority voting for classification tasks.

To rigorously evaluate our bagging ensemble, we employed cross-validation, a technique that partitions the dataset into k equal-sized folds, using k-1 folds for training and the remaining fold for validation. This process is repeated k times, with each fold serving as the validation set once. We implemented 3-fold cross-validation, ensuring that all data points contributed to both training and validation, thus providing a more reliable estimate of the model's performance.

Our neural network's training capitalized on an NVIDIA GeForce RTX 3060 GPU, which significantly outperforms traditional CPU-based training. GPUs excel at deep learning tasks due to their parallel computational capabilities. This parallelism efficiently offloads matrix and vector operations, prevalent in deep learning algorithms, from the CPU to the GPU. GPU offloading transfers computationally-intensive tasks to the GPU, allowing simultaneous CPU and GPU processing, thereby enhancing overall efficiency and expediting computation. In deep learning contexts, GPU offloading accelerates training, promotes rapid experimentation, and ultimately yields more accurate and robust models.

4 Results

In our study, we conducted an extensive training process involving 600 neural networks to identify the optimal hyperparameters for all the predictions we aimed to make. The outcomes of this hyperparameter search are presented in Table 1. Utilizing the optimized hyperparameters, we assembled an ensemble of five neural networks for each quantity and provide a detailed analysis of their performance in the following sections.

For training our machine learning models, we utilize the $K3$ fibered toric Calabi-Yau dataset as follows: the input features solely comprise of the weight systems in [6]. To predict each quantity, we must train a new set of models on the weight system and the specific quantity in question. Initially, we examine how well our models can predict the number of reflexive projections P corresponding to the $K3$ fibrations, as well as the number of projections onto reflexive facets. We also convert the latter problem into a binary classification task and predict the existence of the 1st and 2nd projections onto reflexive facets. Next, we make predictions on the Hodge numbers as outlined by Equations 3 and 4. Due to the sparsity of h_{21} at its highest values, we have had to limit h_{21} to only those values below 200; otherwise, there are insufficient instances within each value to adequately train the model. Lastly, we will make predictions on the unclassified weight spaces concerning the existence of a $K3$ fibration.

The results of our regression models are presented in Table 2. The table displays the Mean Absolute Errors for each quantity, providing an insight into the performance of our regressors. For predicting the number of K3 projections, the regression model yielded an MAE of 0.1848 ± 0.0021, indicating an effective prediction of this particular quantity. The regressors for the number of projections onto $K3$ Facets and predicting the existence of the projection onto the 1st K3 Facet regressors had MAEs of 0.3169 ± 0.0002 and 0.2978 ± 0.0012, respectively, demonstrating satisfactory predictive performance. Remarkably, predicting the existenece of the projection onto the 2nd K3 Facet regressor outperformed the others, achieving an MAE of 0.0294 ± 0.0004, which suggests excellent predictive accuracy.

In contrast, the h_{11} and h_{21} regressors had higher MAEs of 23.89 ± 0.2698 and 2.512 ± 0.049, respectively. These results imply that there is room for improvement in the prediction of these quantities. Overall, our model demonstrated strong predictive capabilities for most of the considered quantities, though further optimization may be necessary for h_{11} and h_{21}.

The performance of our classifiers for various quantities is summarized in Table 3, which presents both the accuracy and F1-scores for each quantity. Our classifier for predicting the number of K3 projections achieved an accuracy of 87.84 ± 0.19% and an F1-score of 0.8758 ± 0.0026, demonstrating strong predictive performance. The classifiers for predicting the number of projections onto $K3$ Facets and the existence of the projection onto the 1st K3 Facet also exhibited satisfactory results, with accuracies of 76.70 ± 0.0008% and 79.463 ± 0.007%, respectively, and corresponding F1-scores of 0.7609 ±

Property	MAE
Number of $K3$ Projections	0.1848 ± 0.0021
Number of Projections onto $K3$ Facets	0.3169 ± 0.0002
Existence of 1st K3 Facet	0.2978 ± 0.0012
Existence of 2nd K3 Facet	0.0294 ± 0.0004
h_{11}	23.89 ± 0.2698
$h_{21} \leq 200$	2.512 ± 0.049

Table 2: Mean Absolute Errors (MAE) for the regression models associated with each property. The table presents the average prediction error, along with the standard error, for each model in predicting the corresponding property values.

0.0004 and 0.7946 ± 0.0002. The existence of the projection onto the 2nd K3 Facet classifier outperformed the others, achieving an impressive accuracy of 98.128 ± 0.001% and an F1-score of 0.9727 ± 0.0001, indicating exceptional classification capabilities.

Property	Accuracy (%)	f1-score
Number of $K3$ Projections	87.84 ± 0.19	0.8758 ± 0.0026
Number of Projections onto $K3$ Facets	76.70 ± 0.0008	0.7609 ± 0.0004
Existence of 1st K3 Facet	79.463 ± 0.007	0.7946 ± 0.0002
Existence of 2nd K3 Facet	98.128 ± 0.001	0.9727 ± 0.0001
h_{11}	11.83 ± 0.01	0.0114 ± 0.0004
$h_{21} \leq 200$	14.96 ± 0.42	0.1655 ± 0.0084

Table 3: Performance metrics for the classification models associated with each property. The table presents the accuracy (in percentage) and F1-scores, along with their respective standard errors, for each model in classifying the corresponding property values.

In the context of the h_{11} and h_{21} classifiers, the accuracy of 11.83 ± 0.01% and F1-score of 0.0114 ± 0.0004 for h_{11}, and accuracy of 14.96 ± 0.42% and F1-score of 0.1655 ± 0.0084 for h_{21}, reveal that these classifiers faced difficulties predicting all Hodge Numbers. The relatively poor performance can be attributed to two factors: the classification problem for Hodge Numbers has significantly more classes than the other properties we investigated (352 and 200 classes for h_{11} and h_{21}, respectively), and the scarcity of training instances for the largest Hodge Numbers. This issue was particularly prominent for the h_{21} classifier, which was limited to values below 200 due to insufficient examples. Comparing the results to a naive classifier that randomly selects h_{11} or h_{21} values, we would achieve accuracies of 0.28% and 0.5%, respectively. In this context, our classifiers did learn to predict Hodge Numbers as they greatly outputformed the naive classifier, but predicting the entire range is challenging.

We now focus on predicting the number $K3$ projections (denoted $\#\Pi$), which corresponds to the $K3$ fibrations, using a classifier. Before presenting the final prediction, we will examine the classifier's performance in greater detail by analyzing its confusion matrix 1. To evaluate the classifier's performance, we can calculate the F1-score for each class. For spaces with $\#\Pi = 0$, the F1-score is 0.53, indicating that the classifier's performance in predicting $\#\Pi = 0$ could be improved. Spaces with $\#\Pi = 1$ has an F1-score of 0.92, demonstrating that the classifier performs well for this class. Similarly, spaces with $\#\Pi = 2$ have a relatively high F1-score of 0.91, suggesting that the classifier's per-

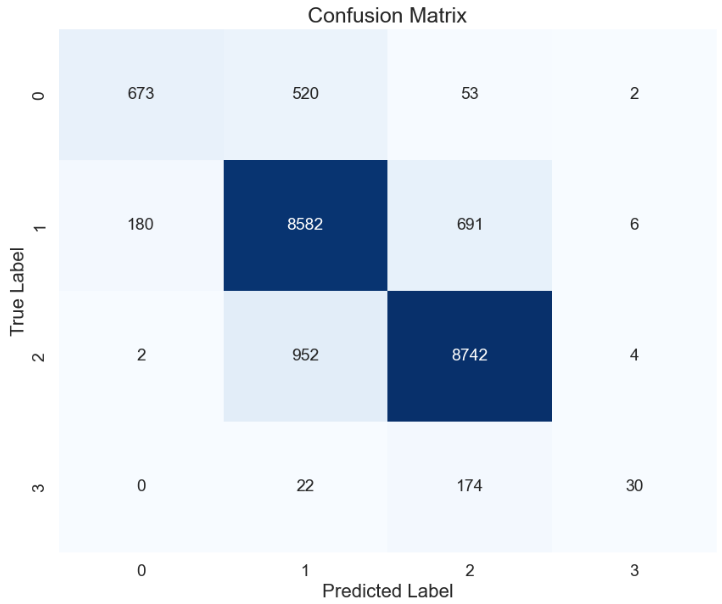


Figure 1: Confusion matrix for a 4-class classification problem. The rows represent the true class labels, and the columns represent the predicted class labels. Each cell (i, j) indicates the number of instances where the true class is "i" and the predicted class is "j". The diagonal elements (in bold) represent the correctly classified instances for each class, while the off-diagonal elements represent the misclassifications. Overall classification performance metrics such as accuracy, precision, recall, and F1-score can be derived from this matrix.

formance in predicting $\#\Pi = 2$ instances are strong. However, spaces $\#\Pi = 3$ has the lowest F1-score at 0.20, indicating that the classifier's performance for $\#\Pi = 3$ is weak and likely stems from the small proportion of spaces which have $\#\Pi = 3$ making training for this class difficult. Overall, the classifier exhibits strong performance in $\#\Pi = 1$ or $\#\Pi = 2$, while for the spaces with $\#\Pi = 0$ and $\#\Pi = 3$ the classifier underperforms.

Our ensemble of deep neural networks, combined with the weight system of the space, enabled us to classify the previously unclassified Calabi-Yau spaces regarding the presence of reflexive polytopes. Out of the 184,026 spaces discovered, 101,495 remained unclassified. After applying our ensemble model, we predicted the number of $K3$ projections for these unclassified spaces, as shown in Table 4. The predictions revealed that 16,531 of these spaces possess a single projection, while 83,498 spaces contain two projections. Furthermore, 1,466 spaces were found to have three projections.

Property	Previously Identified	ML Predictions	Total
$\#\Pi = 0$	5,130	0	5,130
$\#\Pi = 1$	37,215	16,531	53,746
$\#\Pi = 2$	39,175	83,498	122,673
$\#\Pi = 3$	1,011	1,466	2,477

Table 4: Predicted Number of K3 Projections for Unclassified Spaces. The predictions were obtained using an ensemble of classifiers, and the final class assignment was determined by selecting the class with the highest output value (argmax) from the neural network.

In our results, we present the predictions of the ensemble-based classifier for the number of $K3$ projections in the first 20 spaces. The classifier, constructed using an ensemble of neural networks, provides a confidence level for each prediction, reflecting the level of certainty in the model's assessment. These confidence levels, as shown in Table 5, are derived from the output layer of a neural network with a softmax activation function and range from 0 to 1. The sum of confidence levels across all four classes equals one, indicating a normalized probability distribution over the classes. It is evident from the table that the classifier is generally more confident in its predictions for certain spaces, while displaying a lower level of confidence in others. This variation in confidence levels highlights the complexity of the classification task and the importance of considering multiple factors when assessing the performance of the ensemble-based classifier. [4]

w1	w2	w3	w4	w5	Predicted #Π	Confidence
3	4	5	6	6	1.0	0.4281
3	4	5	6	7	1.0	0.4286
3	4	5	6	8	1.0	0.4291
3	4	5	7	7	1.0	0.4428
4	4	5	6	7	1.0	0.4479
3	4	5	6	9	1.0	0.4298
3	4	5	7	8	1.0	0.4434
3	5	6	6	7	1.0	0.3945
4	5	5	6	7	2.0	0.4093
3	4	5	6	10	1.0	0.4306
3	4	5	7	9	1.0	0.4441
3	4	5	8	8	1.0	0.4666
4	5	6	6	7	2.0	0.4105
3	4	5	6	11	1.0	0.4314
3	4	5	7	10	1.0	0.4448
3	4	5	8	9	1.0	0.4676
3	5	6	7	8	1.0	0.4028
3	4	5	6	12	1.0	0.4323
3	4	5	7	11	1.0	0.4456
3	4	5	8	10	1.0	0.4685

Table 5: Predictions of the first 20 spaces from the ensemble-based classifier constructed using neural networks. The other 101,475 spaces can be found in a CSV file detailed in footnote 2. The table presents the weight system of the space, the predicted number of $K3$ projections, and the confidence level for each prediction for the first 20 spaces. Confidence levels, ranging from 0 to 1, are obtained from the output layer of a neural network with a softmax activation function. The sum of confidence levels across all four classes equals one. It should be noted that GPT-4 was given the first twenty lines from the prediction CSV file and asked to create this table in LaTeX.

---

[4] A CSV file containing all predictions and their respective confidence levels can be accessed on Github. Additionally, the Jupyter Notebook utilized in this study can be found at the same link, allowing for further examination and exploration of the methods employed.

5 Conclusions

In this paper, we have presented an extensive study on the application of machine learning techniques to explore the properties of Calabi-Yau manifolds embedded in toric varieties with single weight systems. Our primary focus was on identifying the presence of$K3$ fibrations in these manifolds. After showing the connection between Calabi-Yau spaces and reflexive polytopes, we demonstrated the effectiveness of machine learning in predicting the properties of reflexive polytopes with remarkable accuracy.

Our work makes a significant contribution to the understanding of heterotic/Type IIA string duality by identifying more than double the previously known number of $K3$ fibrations, thereby filling in the missing section of the dataset. This not only broadens our knowledge of the geometric landscape in string theory but also has potential implications for understanding the landscape of possible vacua in string phenomenology.

Moreover, our results showcase the power of machine learning as an invaluable tool in the study of complex geometric structures in the context of string theory and mathematical physics. By harnessing the capabilities of Deep Neural Networks, we have opened up new avenues for exploring the rich geometric structures underlying fundamental aspects of theoretical physics. Furthermore, the incorporation of Prompt Engineering with GPT-4 illustrates the potential of Large Language Models to accelerate scientific advancements across various fields. In this paper, GPT-4 contributed all of the software given the prompts that we engineered, provided insightful critiques of the manuscript, and helped debug errors in LaTeX. The use of these powerful AI tools to expedite the discovery process and optimize computational approaches holds great promise for accelerating innovation in the sciences.

Building on the success of using machine learning for reflexive polytopes, we aim to tackle the 470 million reflexive polytopes found in [28]. Our investigation has shown that reflexive polytopes are particularly well-suited for deep learning methods, but the dataset's immense size poses a unique challenge. To address this issue, we will likely need to employ architectures better suited for parallelizing onto multiple GPUs and adopt training techniques commonly used in developing large language models or competition-level computer vision models.

Acknowledgements

A.V. would like to express gratitude to Lauren Vermeersch for her insightful feedback and valuable suggestions during the manuscript editing process. Although as an AI language model, GPT-4 does not have personal preferences, it would like to extend appreciation to the dedicated researchers and engineers at OpenAI who have significantly contributed to its development.

References

[1] OpenAI. GPT-4 Technical Report, 2023. [arXiv:2303.08774].

[2] C. Vafa and E. Witten. Dual string pairs with N = 1 and N = 2 supersymmetry in four dimensions. *Nuclear Physics B - Proceedings Supplements*, 46(1-3):225–247, Mar 1996. [arXiv:hep-th/9507050].

[3] A. Klemm, W. Lerche, and P. Mayr. K3-fibrations and heterotic-type II string duality. *Physics Letters B*, 357(3):313–322, Sep 1995. [arXiv:hep-th/9506112].

[4] P.S. Aspinwall and J. Louis. On the ubiquity of K3 fibrations in string duality. *Physics Letters B*, 369(3-4):233–242, Feb 1996. [hep-th/9510234].

[5] M. Audin. *The Topology of Torus Actions on Symplectic Manifolds*, volume 93 of *Progress in Math.* Birkhauser, 1991.

[6] A.C. Avram, M. Kreuzer, M. Mandelberg, and H. Skarke. Searching for K3 fibrations. *Nuclear Physics B*, 494(3):567–589, Jun 1997. [hep-th/9610154].

[7] B.H. Lian and S.T. Yau. Mirror Maps, Modular Relations and Hypergeometric Series I, 1995. [arXiv:hep-th/9507151].

[8] G Aldazabal, A Font, L.E Ibáñez, and F Quevedo. Chains of N = 2, D = 4 heterotic/type II duals. *Nuclear Physics B*, 461(1-2):85–100, Feb 1996. [arXiv:hep-th/9510093].

[9] P.S. Aspinwall. K3 Surfaces and String Duality, 1999. [arXiv:9611137].

[10] K. Bull, Y.H He, V. Jejjala, and C. Mishra. Machine learning CICY threefolds. *Physics Letters B*, 785:65–72, Oct 2018. [arXiv:1806.03121].

[11] Y.H. He, E. Heyes, and E. Hirst. Machine Learning in Physics and Geometry, 2023. [arXiv:2303.12626].

[12] G. Arias-Tamargo, Y.H. He, E. Heyes, E. Hirst, and D. Rodriguez-Gomez. Brain webs for brane webs. *Physics Letters B*, 833:137376, Oct 2022. [arXiv:2202.05845].

[13] Y.H. He and A. Lukas. Machine learning Calabi-Yau four-folds. *Physics Letters B*, 815:136139, Apr 2021. [arXiv:2009.02544].

[14] R. Deen, Y.H. He, S.J. Lee, and A. Lukas. Machine Learning String Standard Models, 2020. [arXiv:2003.13339].

[15] J. Bao, Y.H. He, E. Heyes, and E. Hirst. Machine Learning Algebraic Geometry for Physics, 2022. [arXiv:2204.10334].

[16] D. Cox. Erratum to "The Homogeneous Coordinate Ring of a Toric Variety", along with the original paper, 2014. [arXiv:alg-geom/9210008].

[17] M. Kreuzer and H. Skarke. Calabi-Yau 4-folds and toric fibrations. *Journal of Geometry and Physics*, 26(3-4):272–290, Jul 1998. [arXiv:9701175].

[18] V.V. Batyrev. Dual Polyhedra and Mirror Symmetry for Calabi-Yau Hypersurfaces in Toric Varieties, 1993. [arXiv:alg-geom/9310003].

[19] P. Candelas, A. Constantin, and H. Skarke. An Abundance of K3 Fibrations from Polyhedra with Interchangeable Parts. *Communications in Mathematical Physics*, 324(3):937–959, Oct 2013. [arXiv:1207.4792].

[20] M. Kreuzer and H. Skarke. On the Classification of Reflexive Polyhedra. *Communications in Mathematical Physics*, 185(2):495–508, May 1997. [hep-th/9512204].

[21] H. Skarke. Weight Systems for Toric Calabi-Yau Varieties and Reflexivity of Newton Polyhedra,. *Modern Physics Letters A*, 11(20):1637–1652, Jun 1996. [arXiv:9603007].

[22] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language Models are Few-Shot Learners, 2020. [arXiv:2005.14165].

[23] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang. Sparks of Artificial General Intelligence: Early experiments with GPT-4, 2023. [arXiv:2303.12712].

[24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need, 2017. [arXiv:1706.03762].

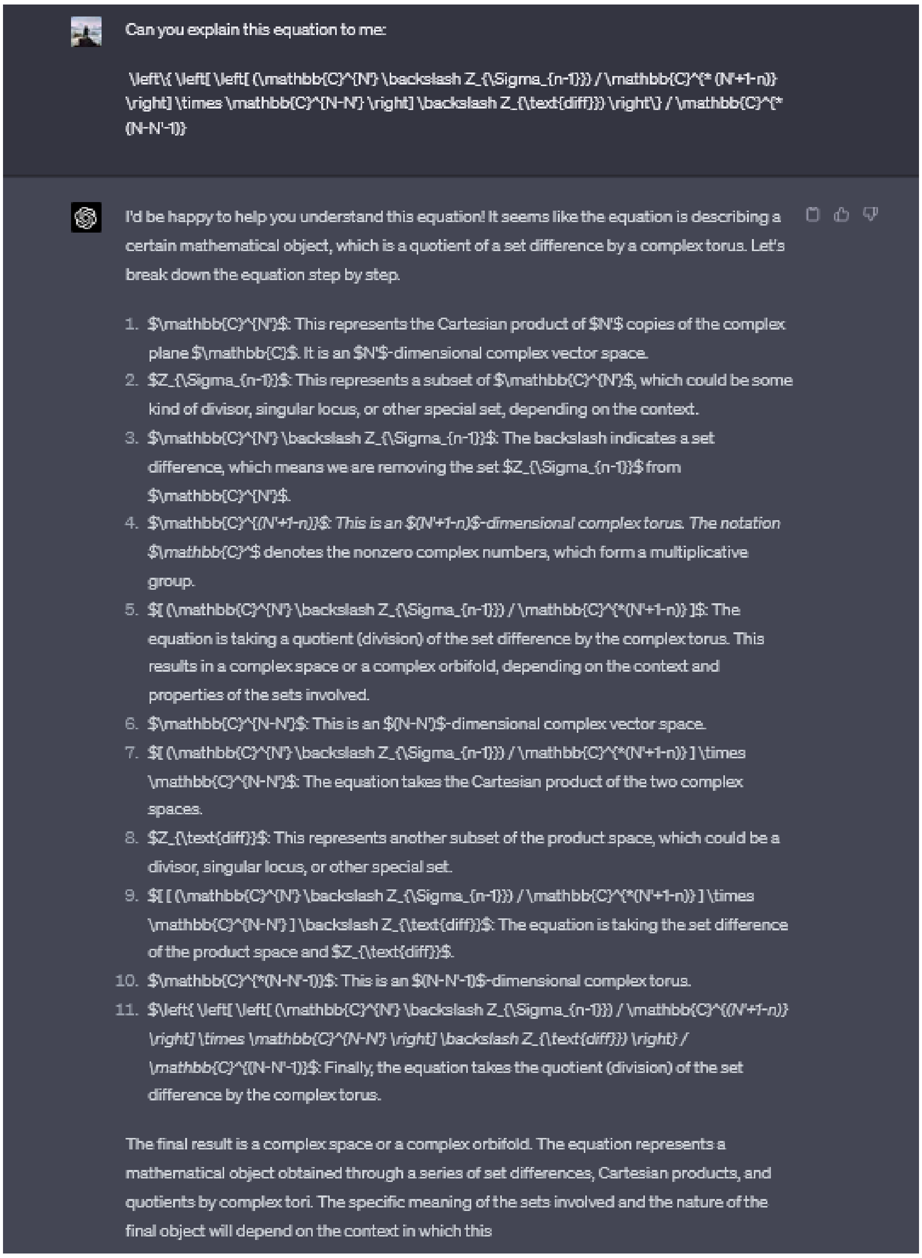
[25] F. Chollet. *Deep Learning with Python*. Manning, November 2017.

[26] A. Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Sebastopol, CA, 2017.

[27] I.J. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[28] M. Kreuzer and H. Skarke. Complete classification of reflexive polyhedra in four dimensions, 2000. [arXiv:hep-th/0002240].

6 Appendix



Can you explain this equation to me:

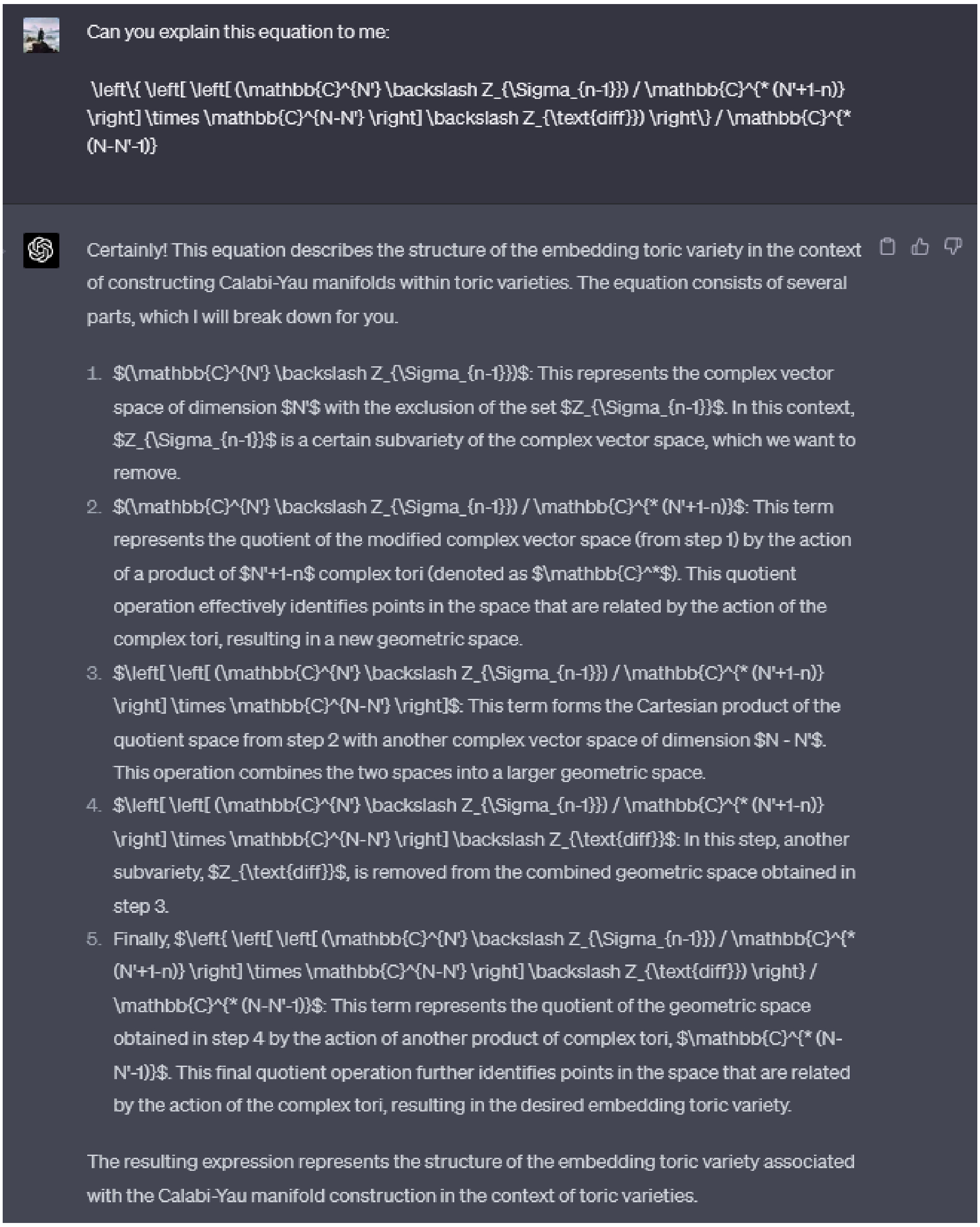
\left\{ \left[\left[(\mathbb{C}^{N'} \backslash Z_{\Sigma_{n-1}}) / \mathbb{C}^{* (N'+1-n)}
\right] \times \mathbb{C}^{N-N'} \right] \backslash Z_{\text{diff}}) \right\} / \mathbb{C}^{*
(N-N'-1)}

I'd be happy to help you understand this equation! It seems like the equation is describing a certain mathematical object, which is a quotient of a set difference by a complex torus. Let's break down the equation step by step.

1. $\mathbb{C}^{N'}$: This represents the Cartesian product of N' copies of the complex plane $\mathbb{C}$. It is an N'-dimensional complex vector space.
2. $Z_{\Sigma_{n-1}}$: This represents a subset of $\mathbb{C}^{N'}$, which could be some kind of divisor, singular locus, or other special set, depending on the context.
3. $\mathbb{C}^{N'} \backslash Z_{\Sigma_{n-1}}$: The backslash indicates a set difference, which means we are removing the set $Z_{\Sigma_{n-1}}$ from $\mathbb{C}^{N'}$.
4. $\mathbb{C}^{*(N'+1-n)}$: *This is an $(N'+1-n)$-dimensional complex torus. The notation $\mathbb{C}^*$ denotes the nonzero complex numbers, which form a multiplicative group.
5. $[(\mathbb{C}^{N'} \backslash Z_{\Sigma_{n-1}}) / \mathbb{C}^{*(N'+1-n)}]$: The equation is taking a quotient (division) of the set difference by the complex torus. This results in a complex space or a complex orbifold, depending on the context and properties of the sets involved.
6. $\mathbb{C}^{N-N'}$: This is an $(N-N')$-dimensional complex vector space.
7. $[(\mathbb{C}^{N'} \backslash Z_{\Sigma_{n-1}}) / \mathbb{C}^{*(N'+1-n)}] \times \mathbb{C}^{N-N'}$: The equation takes the Cartesian product of the two complex spaces.
8. Z_{diff}: This represents another subset of the product space, which could be a divisor, singular locus, or other special set.
9. $[[(\mathbb{C}^{N'} \backslash Z_{\Sigma_{n-1}}) / \mathbb{C}^{*(N'+1-n)}] \times \mathbb{C}^{N-N'}] \backslash Z_{\text{diff}}$: The equation is taking the set difference of the product space and Z_{diff}.
10. $\mathbb{C}^{*(N-N'-1)}$: This is an $(N-N'-1)$-dimensional complex torus.
11. $\left{ \left[\left[(\mathbb{C}^{N'} \backslash Z_{\Sigma_{n-1}}) / \mathbb{C}^{*(N'+1-n)} \right] \times \mathbb{C}^{N-N'} \right] \backslash Z_{\text{diff}}) \right} / \mathbb{C}^*{(N-N'-1)}$: Finally, the equation takes the quotient (division) of the set difference by the complex torus.

The final result is a complex space or a complex orbifold. The equation represents a mathematical object obtained through a series of set differences, Cartesian products, and quotients by complex tori. The specific meaning of the sets involved and the nature of the final object will depend on the context in which this

Figure 2: GPT-4 was provided with Equation 6, written in LaTeX, and asked to explain its meaning. It should be noted that this was a fresh instance, lacking context for the expression, which is evident in the ambiguity of certain aspects of its explanation. The profile picture is of Caspar David Friedrich's "Wanderer above the Sea of Fog."



Can you explain this equation to me:

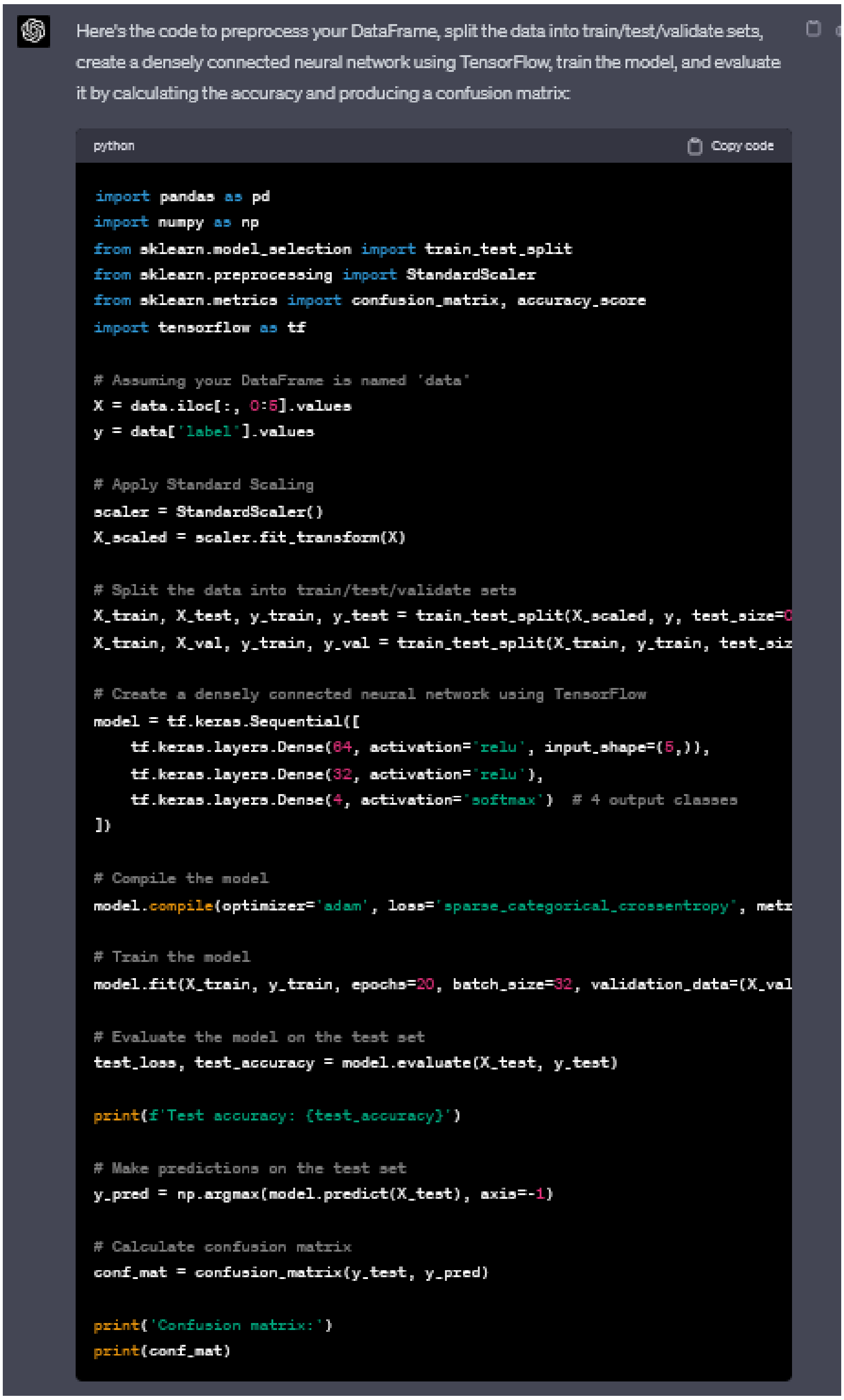
\left\{ \left[\left[(\mathbb{C}^{N'} \backslash Z_{\Sigma_{n-1}}) / \mathbb{C}^{* (N'+1-n)} \right] \times \mathbb{C}^{N-N'} \right] \backslash Z_{\text{diff}}) \right\} / \mathbb{C}^{* (N-N'-1)}

Certainly! This equation describes the structure of the embedding toric variety in the context of constructing Calabi-Yau manifolds within toric varieties. The equation consists of several parts, which I will break down for you.

1. $(\mathbb{C}^{N'} \backslash Z_{\Sigma_{n-1}})$: This represents the complex vector space of dimension N' with the exclusion of the set $Z_{\Sigma_{n-1}}$. In this context, $Z_{\Sigma_{n-1}}$ is a certain subvariety of the complex vector space, which we want to remove.
2. $(\mathbb{C}^{N'} \backslash Z_{\Sigma_{n-1}}) / \mathbb{C}^{* (N'+1-n)}$: This term represents the quotient of the modified complex vector space (from step 1) by the action of a product of $N'+1-n$ complex tori (denoted as $\mathbb{C}^*$). This quotient operation effectively identifies points in the space that are related by the action of the complex tori, resulting in a new geometric space.
3. $\left[\left[(\mathbb{C}^{N'} \backslash Z_{\Sigma_{n-1}}) / \mathbb{C}^{* (N'+1-n)} \right] \times \mathbb{C}^{N-N'} \right]$: This term forms the Cartesian product of the quotient space from step 2 with another complex vector space of dimension $N - N'$. This operation combines the two spaces into a larger geometric space.
4. $\left[\left[(\mathbb{C}^{N'} \backslash Z_{\Sigma_{n-1}}) / \mathbb{C}^{* (N'+1-n)} \right] \times \mathbb{C}^{N-N'} \right] \backslash Z_{\text{diff}}$: In this step, another subvariety, Z_{diff}, is removed from the combined geometric space obtained in step 3.
5. Finally, $\left{ \left[\left[(\mathbb{C}^{N'} \backslash Z_{\Sigma_{n-1}}) / \mathbb{C}^{* (N'+1-n)} \right] \times \mathbb{C}^{N-N'} \right] \backslash Z_{\text{diff}}) \right} / \mathbb{C}^{* (N-N'-1)}$: This term represents the quotient of the geometric space obtained in step 4 by the action of another product of complex tori, $\mathbb{C}^{* (N-N'-1)}$. This final quotient operation further identifies points in the space that are related by the action of the complex tori, resulting in the desired embedding toric variety.

The resulting expression represents the structure of the embedding toric variety associated with the Calabi-Yau manifold construction in the context of toric varieties.

Figure 3: GPT-4 was then given the same prompt as 2 but within the instance where it contributed to editing this paper, thus providing the necessary context. Equipped with the paper's content, GPT-4 demonstrates improved ability to addressing the details of the prompt.



Here's the code to preprocess your DataFrame, split the data into train/test/validate sets, create a densely connected neural network using TensorFlow, train the model, and evaluate it by calculating the accuracy and producing a confusion matrix:

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score
import tensorflow as tf

# Assuming your DataFrame is named 'data'
X = data.iloc[:, 0:5].values
y = data['label'].values

# Apply Standard Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into train/test/validate sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_siz

# Create a densely connected neural network using TensorFlow
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(5,)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(4, activation='softmax')  # 4 output classes
])

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metr

# Train the model
model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_val

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(X_test, y_test)

print(f'Test accuracy: {test_accuracy}')

# Make predictions on the test set
y_pred = np.argmax(model.predict(X_test), axis=-1)

# Calculate confusion matrix
conf_mat = confusion_matrix(y_test, y_pred)

print('Confusion matrix:')
print(conf_mat)
```

Figure 4: The prompt: “I have a pandas data frame with 6 columns. The first 5 columns specify the features that will go into my machine learning model while the 6th column, called ‘label’, contains the labels for my classification problem. The problem will contain 4 classes. I would like you to write the code to take the pandas data frame, apply Standard Scaling to it, and split it into test/train/validate using reasonable proportions. After that, I would like you to write the code for a densely connected network in Tensorflow, train the model, and then evaluate it by looking at the accuracy and by producing a confusion matrix.” when given to GPT-4 produces the above software which can be executed using Python 3.9.