Machine Learning and LLM-Boost Symbolic Regression for Predicting $\mathbb{Q}$-Gonality of Modular Curves

Xu Zhuang [*1] **Xiaokang Wang** [*1] **Yuxiang Yao** [*1] **Po-Chu Hsu** [*2] **Peikai Qi** [*3]

Abstract

We aim to predict the $\mathbb{Q}$-gonality of modular curves, an invariant measuring the minimal degree of a nonconstant rational map to $\mathbb{P}^1$. Three machine-learning architectures—Extrem gradient-boosted trees, feedforward neural networks, and transformer-based models—achieve over 90% exact-match accuracy on existing curves, with more than 89% of predictions falling within known theoretical bounds. To improve interpretability, we employ an LLM-guided boost symbolic regression pipeline that proposes nonlinear feature combinations and uncovers concise analytic formulas. These expressions match the predictive power of our models while revealing how core arithmetic invariants interact. Our results highlight the effectiveness of combining data-driven prediction with LLM-enhanced symbolic discovery in arithmetic geometry.

1. Introduction

Machine learning and artificial intelligence are increasingly used to study mathematical objects, including in pure mathematics. Prior works have applied machine learning to number theory (He, 2023; He et al., 2024a; Alessandretti et al., 2023; Babei et al., 2025), Calabi-Yau topology (He et al., 2024b), affine Deligne-Lusztig varieties (Dong et al., 2024), and combinatorial constructions (Charton et al., 2024). Transformers have also been used in enumerative geometry (Hashemi et al., 2025). Large Language Models (LLMs) have shown promise in mathematical discovery, such as finding constructions in extremal combinatorics (Romera-Paredes et al., 2024; Ellenberg et al., 2025).

[*]Equal contribution [1]Department of Mathematics, University of California, Irvine,United States [2]Department of Computer Science, University of California, Irvine,United States [3]Department of Mathematics, Michigan State University, East Lansing,United States. Correspondence to: Xu Zhuang <xzhuang8@uci.edu>.

The second AI for MATH Workshop at the 42nd International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

Modular curves are important in arithmetic geometry, with invariants like the $\mathbb{Q}$-gonality, which measures the minimal degree of a rational map to $\mathbb{P}^1(\mathbb{Q})$. Computing $\mathbb{Q}$-gonality is challenging for general modular curves.

This work explores whether machine learning can predict $\mathbb{Q}$-gonality from known invariants of modular curves. We analyze the LMFDB dataset to select features most relevant for prediction, aiming for a minimal and non-redundant set. We then evaluate models including XGBoost, feedforward neural networks, and FT-transformers, assessing their accuracy and ability to respect known bounds.

Recent works have also used LLMs for symbolic regression (SR) (Shojaee et al., 2024; Merler et al., 2024). We investigate whether LLMs can help discover symbolic expressions for $\mathbb{Q}$-gonality in terms of other invariants. We introduce LLM-Boost, a symbolic regression method that combines LLM-generated features with Adaboost, building an interpretable additive model. Our approach outperforms SR baselines and suggests new relationships between invariants and $\mathbb{Q}$-gonality.

2. Preliminaries

2.1. Dataset Description

This study utilizes modular curve data from the L-functions and Modular Forms Database (LMFDB) (LMFDB Collaboration, 2025), which catalogs arithmetic and geometric invariants including `genus`, `level`, `rank`, `q_gonality`, `cusps`, `rational_cusps`, `conductor`, and `log_conductor`.

The LMFDB provides `q_gonality_bounds` containing constraints on `q_gonality` values. For curves with exact values, this field shows '$[a, a]$'. For partial information, it shows '$[a, b]$', where a and b are lower and upper bounds respectively.

For machine learning purposes, interval notations were transformed into numerical values. Curves with exact `q_gonality` values (denoted as '$[a, a]$') were assigned their precise values. For curves with only bounds ('$[a, b]$'), the data was primarily used for model evaluation. In the case of feedforward neural networks, we also experimented

with weakly supervised learning techniques using the data that only have bounds.

2.2. Definitions and Mathematical Motivation

Here we collect basic definitions used in this work. For more detailed mathematical background, see Section A.

Given an open subgroup $H \subset \mathrm{GL}_2(\hat{\mathbb{Z}})$, we can define the associated modular curve X_H (LMFDB Collaboration, 2025). Several important invariants are associated with modular curves, including the `level`, `genus`, `cusps`, `index`, and `psl2index`.

- The `level` of a modular curve X_H is the smallest positive integer N, such that H is the inverse image of its projection to $\mathrm{GL}_2(\mathbb{Z}/N\mathbb{Z})$.
- The `genus` of X_H is the genus of any of its geometric components.
- The `cusps` of X_H are the points whose image under the canonical morphism $j\colon X_H \to X(1) \simeq \mathbb{P}^1$ is ∞.
- The `index` of X_H is $[\mathrm{GL}_2(\widehat{\mathbb{Z}}) : H]$.
- The `psl2index` of X_H is defined as $[\mathrm{PSL}_2(\mathbb{Z}) : \pm H \cap \mathrm{SL}_2(\mathbb{Z})]$.

These indices satisfy the following relation:

$$\texttt{psl2index} = \begin{cases} \texttt{index} & \text{if } -1 \in H, \\ \texttt{index}/2 & \text{if } -1 \notin H. \end{cases}$$

If $\mathbb{K}$ is a number field, then the $\mathbb{K}$-gonality of a modular curve X_H is defined as the minimal degree of a dominant morphism $X_H \dashrightarrow \mathbb{P}^1$ defined over $\mathbb{K}$. The value `qbar_gonality` refers to the $\overline{\mathbb{Q}}$-gonality, while `q_gonality` refers to the $\mathbb{Q}$-gonality. It is clear that `q_gonality` $\geq$ `qbar_gonality`. Gonality is a subtle and intriguing arithmetic invariant that measures the geometric complexity of a modular curve. In this project, our goal is to investigate whether machine learning can effectively predict `q_gonality` using other known invariants of modular curves.

Computing the gonality of a modular curve directly is often challenging, but bounds can frequently be established. For instance, `psl2index` represents the degree of the map $j\colon X_H \to X(1) \cong \mathbb{P}^1$ over the complex field $\mathbb{C}$, which implies `psl2index` $\geq$ `qbar_gonality` by definition. Additionally, Abramovich (Abramovich, 1996) established a linear lower bound: $\frac{7}{800} \cdot$ `psl2index` $\leq$ `qbar_gonality`. Various other techniques exist for determining upper and lower bounds on gonality (see, e.g., (Derickx & van Hoeij, 2014), (Orlić, 2025)). Below, we summarize some general properties of the $\mathbb{K}$-gonality of a curve X of genus g, denoted $\gamma_{\mathbb{K}}(X)$, as outlined in Proposition A.1 of (Poonen, 2006).

Proposition 2.1. *(Proposition A.1 of (Poonen, 2006))*

1. *If $\pi\colon X \dashrightarrow Y$ is a dominant rational map of curves over a field $\mathbb{K}$, then $\gamma_{\mathbb{K}}(X) \leq \deg(\pi)\gamma_{\mathbb{K}}(Y)$.*
2. *If $g > 1$, then $\gamma_{\mathbb{K}}(X) \leq 2g - 2$.*
3. *If $X(\mathbb{K}) \neq \emptyset$ and $g \geq 2$, then $\gamma_{\mathbb{K}}(X) \leq g$.*

The LMFDB database collects such bounds for `q_gonality` for modular curves and records them as `q_gonality_bounds`.

We now recall the definitions of other key features used in our training data. Roughly speaking, the `rank` of a modular curve X_H refers to the order of vanishing at the central point of the L-function associated with its Jacobian. The `conductor` of X_H is the conductor of this L-function. The `canonical_conjugator` is an element that conjugates the subgroup generated by the generators to the subgroup generated by the canonical generators.

The features mentioned above are not strongly correlated in the sense that one invariant can not be determined from the others. Although we generally avoid using features that are strongly related during training, including additional (even correlated) features in the model can sometimes improve prediction accuracy. We now introduce several such features. In mathematics, there is currently no known formula that directly relates these features to the gonality of a modular curve. Some of them are self-explanatory. For instance, `contains_negative_one` is a boolean value indicating whether H contains -1, and `pointless` is a boolean value indicating whether $X_H(\mathbb{Q})$ is empty. The LMFDB also includes many features labeled as "`coarse_invariant`", which refer to the corresponding invariant of the group generated by ± 1 and H. For example, the `coarse_level` of X_H denotes the level of the group $\langle \pm 1, H \rangle$.

3. Feature Selection

Feature selection is a crucial step in building effective machine learning models. In this section, we outline our systematic approach to identifying the most relevant features for predicting the $\mathbb{Q}$-gonality of modular curves.

3.1. Methodology and Approach

We began by sampling 100,000 modular curves from the LMFDB dataset and computed the correlation matrix for all available numerical features. Features exhibiting strong correlation with `q_gonality` were initially prioritized, but

we carefully considered multicollinearity to avoid redundancy, as high correlation between predictors can lead to model instability and reduced interpretability.

Most numerical features in the dataset are integers, and for effective prediction, we required a feature set that could uniquely characterize `q_gonality` values. To achieve this, we employed an inductive approach, incrementally adding features until we identified a minimal set that uniquely determined `q_gonality` across the dataset.

3.2. Feature Selection Process

The LMFDB dataset provides many numerical features for modular curves, such as `coarse_level`, `coarse_index`, `genus`, `coarse_class_num`, `level`, `index`, `fine_num`, `psl2index`, `nu2`, `nu3`, `cusps`, `psl2level`, `sl2level`, `rational_cusps`, `rank`, `genus_minus_rank`, `trace_hash`, `log_conductor`, `num_bad_primes`, `level_radical`, `q_gonality`, and `qbar_gonality`.

We visualized feature relationships using a correlation matrix heatmap (Figure 13). Based on this, we selected `genus`, `cusps`, `rank`, `log_conductor`, `level`, and `rational_cusps` as the most relevant features for predicting `q_gonality`.

However, these features alone could not uniquely determine `q_gonality`—some entries had the same feature values but different `q_gonality`. By adding `coarse_class_num` and `coarse_level`, we obtained a feature set that uniquely determined `q_gonality` for all data points.

Further analysis showed that while `genus`, `log_conductor`, and `rank` are strongly correlated with `q_gonality`, none alone is sufficient for accurate prediction. Details are in Appendix E.

4. Machine Learning $\mathbb{Q}$-Gonality Prediction

For a regression model $\mathcal{M}$ such as XGBoost or Feedforward neural network, we use mean squared error (MSE) as the loss function. For a test set (X_{test}, Y_{test}), we denote predictions by $\mathcal{M}(X_{test})$. Since `q_gonality` is a non-negative integer, we define modified predictions as the closest integer to $\mathcal{M}(X_{test})$:

$$\lfloor \mathcal{M}(X_{test}) + 0.5 \rfloor$$

This modification is necessary because regression models produce continuous outputs, whereas `q_gonality` must be a non-negative integer, requiring appropriate rounding.

Then we can define the accuracy for the regression model $\mathcal{M}$ on (X_{test}, Y_{test}) as

$$Acc(\mathcal{M}(X_{test})) := \frac{1}{|Y_{test}|} \sum_{i=1}^{|Y_{test}|} \mathbf{1}[\lfloor \mathcal{M}(X_{test}^{(i)}) + 0.5 \rfloor = Y_{test}^{(i)}]$$

where $\mathbf{1}[\cdot]$ is the indicator function and $Y_{test}^{(i)}$ denotes the true `q_gonality` value for the i-th test sample.

An interesting direction for future work would be to investigate how bounded conditions on certain features could constrain predicted `q_gonality` values to specific ranges. Such an approach might yield more precise predictions by incorporating known mathematical constraints from modular curve theory.

4.1. Data Preparation

Based on our feature selection analysis, we identified eight numerical features that uniquely determine `q_gonality`: `genus`, `level`, `rank`, `log_conductor`, `cusps`, `rational_cusps`, `coarse_class_num`, and `coarse_level`. Additionally, we incorporated `canonical_conjugator` and `conductor` as supplementary features in our custom dataset.

Figure 1 illustrates the distribution of `q_gonality` values, revealing that the majority are small, with a few outliers at larger values. This presents a challenge for regression models, as they tend to perform well on small `q_gonality` values but consistently struggle to predict larger ones.

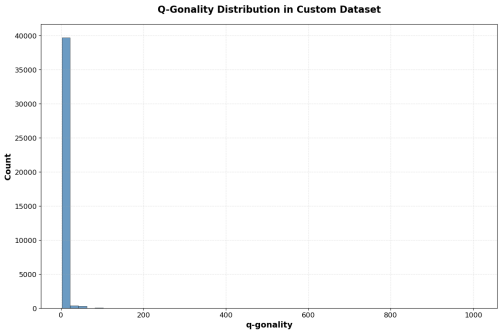


Figure 1. Distribution of `q_gonality` values in the dataset. Most values are small, with a few outliers at larger values.

Tables 1 and 2 summarize the five most and least frequent `q_gonality` values, respectively. The results show that the values 4, 8, 16, 2, and 6 together constitute a substantial portion of the dataset, indicating that most modular curves have low `q_gonality` in the custom dataset, while higher values are comparatively rare.

After extracting data from LMFDB and removing entries with null values, we obtained 46,260 unique modular curves. We split the dataset into D_{exact} (with precise `q_gonality` values) and D_{bounds} (with only bounds available). The

Table 1. Top 5 `q_gonality` Value Distributions

Value	Count	Percentage
4	17,268	42.679%
8	8,068	19.941%
16	6,544	16.174%
2	5,960	14.731%
6	788	1.948%

Table 2. Last 5 `q_gonality` Value Distributions

Value	Count	Percentage
336	7	0.017%
252	4	0.001%
1008	2	0.005%
504	2	0.005%
324	2	0.005%

D_{exact} dataset was further divided into a fixed test set D_{test} (10%) for final evaluation, and the remaining data was split into training (80%) and validation (10%) sets. For cross-validation, we performed 5-fold splits on the non-test portion, denoting the resulting training and validation sets as $D_{train}^{(i)}$ and $D_{val}^{(i)}$ for the i-th fold, where $i = 1, \ldots, 5$. The test set D_{test} remains fixed and is not included in the cross-validation folds.

4.2. Experiments

4.2.1. Hardware and Software Configuration

All experiments were conducted on a workstation equipped with an AMD Ryzen 9 7900X3D CPU, 128GB DDR5 RAM, and an NVIDIA GeForce RTX 4080 GPU. The software environment consisted of Ubuntu 24.04 LTS and Python 3.11.11.

4.2.2. XGBoost

XGBoost (Chen & Guestrin, 2016) is a gradient boosting algorithm widely used for regression tasks due to its speed and accuracy. We used XGBoost with squared error loss, max depth 10, learning rate 0.02, 700 trees, subsample 0.6, colsample 0.8, and GPU acceleration. The model was trained on D_{train}^{i} and evaluated on D_{val}^{i}, D_{test}, and D_{bounds}. Table 3 shows the mean and standard deviation of metrics over 5 cross-validation folds.

Table 3. Performance metrics for the XGBoost regression model across different datasets

Dataset	Acc. (%)	R^2	RMSE
D_{train}^{i}	95.12 ± 0.08	0.9995 ± 0.0001	0.258 ± 0.021
D_{val}^{i}	93.91 ± 0.28	0.9971 ± 0.0028	0.687 ± 0.541
D_{test}	93.84 ± 0.11	0.9988 ± 0.0001	0.302 ± 0.012
D_{bounds}	89.14 ± 0.36	–	–

4.2.3. Feedforward Neural Network

We implemented a Feedforward Neural Network (FNN) using PyTorch with an architecture optimized for this regression task. In this model, we used a simple neural network with the architecture: $12 - 128 - 32 - 1$, where 12 is the input dimension, 128, 32 describe the hidden layer shape, and 1 is the output dimension, each followed by LeakyReLU activation functions with the negative slope 0.01. At the end of the network, we round the output to the nearest integer. We discussed its necessity at the beginning of this section. Note that the feature, `canonical_conjugator`, is considered as a 4-dimensional input to the neural network.

Proposition 2.1 indicates that `q_gonality_bounds` are easier to obtain than exact `q_gonality` values. To utilize all available data, we trained the FNN using weakly supervised learning, which constructs predictive models when labeling resources are limited or labels are noisy/uncertain (Zhou, 2018; Scudder, 1965; Blum & Mitchell, 1998; Collins & Singer, 1999). This approach is particularly suitable here since computing exact `q_gonality` for many modular curves is difficult, while `q_gonality_bounds` provide interval constraints rather than precise outputs. Weakly supervised learning has proven effective in physics (Dery et al., 2017) and medical imaging (Ren et al., 2023).

To implement this weakly supervised learning strategy, we split the D_{bounds} dataset into three subsets: D_{bounds_train} (80%), D_{bounds_val} (10%), and D_{bounds_test} (10%). Similarly, D_{exact} is split into D_{train} and D_{val} using the same ratio as D_{train}^{i}.

Training proceeds in two phases: during the first 2000 epochs, the FNN is trained solely on D_{train} (accurately labeled data) using the standard MSE loss. In the subsequent 2000 epochs, we combine D_{train} and D_{bounds_train} into D'_{train}, and D_{val} and D_{bounds_val} into D'_{val}, training and evaluating the FNN on these merged datasets.

For the combined datasets, the loss function generalizes the MSE to accommodate interval labels from `q_gonality_bounds`. Specifically, for data points with bounds $[a_i, b_i]$, we define

$$\ell(\mathbf{y}, [\mathbf{a}, \mathbf{b}]) = \sum_{i=1}^{N} \text{distance}(y_i, [a_i, b_i])^2$$

where $\mathbf{y}$ is the vector of predicted `q_gonality` values,

$[\mathbf{a}, \mathbf{b}]$ is the list of interval bounds, and N is the number of samples. The function $\text{distance}(y_i, [a_i, b_i])$ computes the shortest distance from y_i to the interval $[a_i, b_i]$, which is zero if $y_i \in [a_i, b_i]$. The learning rate is fixed at 0.0001 and the batch size is 32 throughout training.

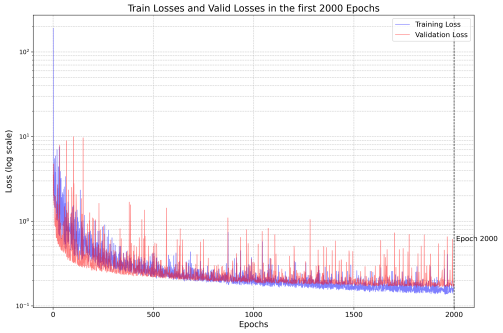


Figure 2. Example of Training and validation loss curves for neural network model in the first 2000 epochs

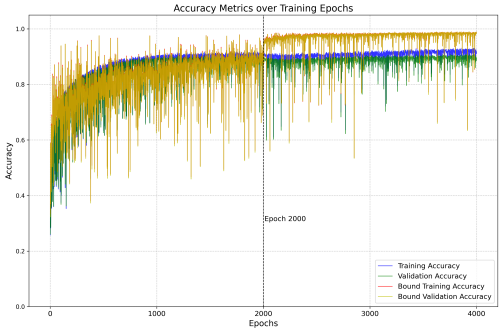


Figure 3. Example of Accuracy curves for for all 4000 epochs

We choose the model with the best valid loss (the above loss function ℓ) among the last 2000 epochs, which happened in 3878-th epoch. The performance of this model is given by Table 4.

Table 4. Performance metrics for the Feedforward Neural Network model across different datasets using weakly supervised learning strategy

Dataset	**Accuracy**	R^2	**RMSE**
D_{train}	91.55%	0.9992	0.3259
D_{val}	89.17%	0.9981	0.3899
D_{test}	87.94%	0.9997	0.4020
D_{bounds}	98.53%	–	–
D_{bounds_train}	98.79%	–	–
D_{bounds_val}	98.86%	–	–
D_{bounds_test}	98.21%	–	–

As a comparison, Table 5 shows the model trained only on D_{train} for 2000 epochs. This shows that the second half of the training roughly kept the accuracy on D_{test}, and significantly improved the accuracy on D_{bounds_test} (see Figure 3), while it did not significantly affects the performances on other datasets.

Table 5. Performance metrics for the Neural Network model across different datasets

Dataset	**Acc. (%)**	R^2	**RMSE**
D^i_{train}	92.70 ± 0.30	0.9991 ± 0.0001	0.360 ± 0.007
D^i_{val}	90.79 ± 0.44	0.9983 ± 0.0012	0.422 ± 0.019
D_{bounds}	89.29 ± 1.29	–	–

The classical neural network model above has better accuracies on D_{bounds} and on its splitted subsets, even if without applying weakly supervised learning strategy, which suggests that the neural network has good generalization performance. Unfortunately, the neural network model performs less well than the XGBoost model on the accurately labeled datasets, which also inspired our experiments on other models.

4.2.4. Feature Tokenizer Transformer (FT-Transformer)

The Feature Tokenizer Transformer (FT-Transformer) was introduced by (Gorishniy et al., 2021) as an adaptation of the transformer architecture specifically designed for tabular datasets. This approach combines the representation learning capabilities of transformers with specialized tokenization techniques to effectively process heterogeneous feature types commonly found in structured data.

The FT-Transformer architecture has demonstrated strong performance across various tabular prediction tasks, particularly those involving complex nonlinear relationships between features. In this work, we employ an FT-Transformer specifically adapted for predicting arithmetic invariants in modular curve datasets, leveraging its ability to handle the mixed numerical, categorical, and list-valued features present in our LMFDB data.

Figure 4 illustrates the FT-Transformer architecture used in our implementation.

Our FT-Transformer implementation uses a feature tokenizer to convert heterogeneous inputs into unified embeddings. Numerical features including 8 core invariants (`level`, `genus`, `rank`, `cusps`, `rational_cusps`, `log_conductor`, `coarse_class_num`, `coarse_level`) are linearly projected to $d_{model} = 128$ dimensions. Three boolean categorical features are embedded using standard embedding

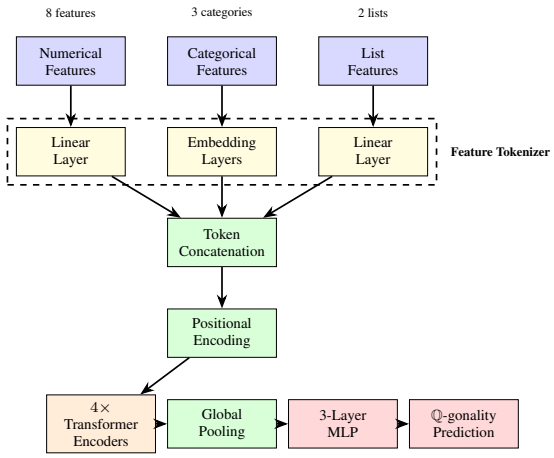


Figure 4. FT-Transformer architecture for q-gonality prediction. The model processes 8 numerical features, 3 categorical features, and 2 list features through a feature tokenizer, followed by positional encoding, transformer encoders, and a multi-layer perceptron (MLP) output head.

layers, while 2 list features (`canonical_conjugator`, `conductor`) are processed through linear projection layers.

The architecture consists of 4 transformer encoder layers with 8 attention heads, feedforward dimension 512, and GELU activation. We use AdamW optimizer with learning rate 5×10^{-4}, train for 270 epochs with batch size 64, and apply early stopping with patience 50.

We also apply dropout with rate 0.08 to prevent overfitting.

The performance results for the FT-Transformer model are summarized in Table 6.

Table 6. Performance metrics for the FT-Transformer model across different datasets

Dataset	**Acc. (%)**	R^2	**RMSE**
D^i_{train}	95.62 ± 0.70	0.9389 ± 0.0325	2.813 ± 0.972
D^i_{val}	93.17 ± 0.58	0.9971 ± 0.1079	3.770 ± 4.001
D_{test}	93.64 ± 0.63	0.9968 ± 0.0007	0.496 ± 0.055
D_{bounds}	93.46 ± 0.82	–	–

4.3. Discussion of Results

Across all splits, **XGBoost** demonstrated the most consistent and reliable point-wise accuracy, achieving the lowest RMSE (0.258 ± 0.021 on D^i_{train} and 0.302 ± 0.012 on D^i_{test}), and near-perfect R^2 scores (≥ 0.9988). The drop in accuracy from train to test was modest ($95.12\% \rightarrow 93.84\%$). The **Feedforward Neural Network** performed slightly worse, with a test-set RMSE approximately 40%

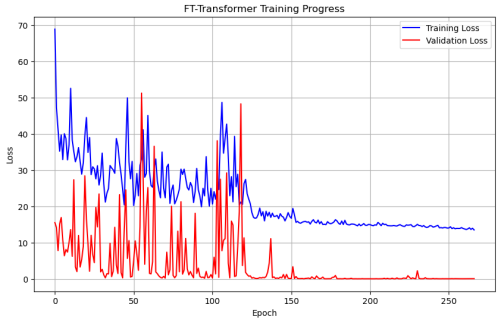


Figure 5. Example of Training and validation loss curves for FT-Transformer model

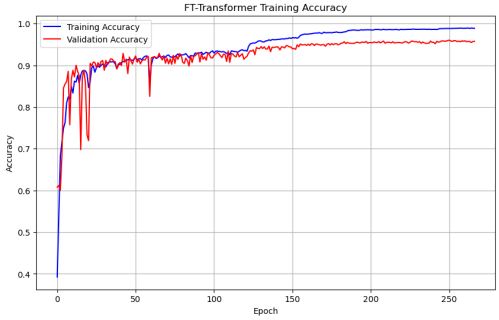


Figure 6. Example of Training and validation accuracy curves for FT-Transformer model

higher (0.425 ± 0.009) and a 2–3% decrease in accuracy, but still maintained strong generalization ($R^2 = 0.9977$). In contrast, the **FT-Transformer** matched the other models in accuracy for in-distribution splits, but exhibited a substantially higher RMSE (2.813 ± 0.972 on D^i_{train} and 3.770 ± 4.001 on D^i_{val}), likely due to difficulty modeling rare, large-magnitude targets in the training data. However, when these extreme cases are excluded—as in D_{test} and D_{bounds}—its RMSE drops sharply (0.496 ± 0.055), and it achieves the highest accuracy on the boundary set (93.46%). This suggests that self-attention may provide robustness near decision boundaries, even if calibration suffers for large `q_gonality` values.

4.4. Out-of-Distribution (OOD) Test

To evaluate the generalization capability of our models, we conduct an out-of-distribution (OOD) assessment. Specifically, we partition D_{exact} into two subsets: $D_{\text{level}\leq 59}$, containing modular curves with level ≤ 59, and $D_{\text{level}>59}$, containing those with level > 59. This split allows us to train models exclusively on $D_{\text{level}\leq 59}$ and then evaluate their performance on the more challenging, higher-level set $D_{\text{level}>59}$, thereby rigorously testing each model's robustness to distributional shifts.

$D_{\text{level}>59}$ comprises 10.04% of D_{exact}. We further divide $D_{\text{level}\leq 59}$ into training and validation sets, denoted D_{train}^{ood} and D_{val}^{ood}, using an 8:1 ratio. For XGBoost and FNN, we use the same hyperparameters as in the previous sections, while for FT-Transformer, we increase dropout to 0.175 to improve generalization.

Table 7. Performance metrics for XGBoost under OOD evaluation

Dataset	Accuracy	R^2	RMSE
D_{train}^{ood}	96.86%	0.9994	0.185
D_{val}^{ood}	95.45%	0.9991	0.2526
$D_{\text{level}>59}$	80.96%	0.7206	16.0031
D_{bounds}	87.57%	–	–

Table 8. Performance metrics for FNN under OOD evaluation

Dataset	Accuracy	R^2	RMSE
D_{train}^{ood}	92.81%	0.9979	0.354
D_{val}^{ood}	91.59%	0.9979	0.377
$D_{\text{level}>59}$	68.71%	0.9950	2.141
D_{bounds}	91.93%	–	–

Table 9. Performance metrics for FT-Transformer under OOD evaluation

Dataset	Accuracy	R^2	RMSE
D_{train}^{ood}	93.00%	0.9398	1.906
D_{val}^{ood}	91.94%	0.9933	0.676
$D_{\text{level}>59}$	82.98%	0.3486	24.435
D_{bounds}	89.74%	–	–

The out-of-distribution (OOD) results show that all models perform well on the training and validation sets, but their accuracy drops on the harder $D_{\text{level}>59}$ set. XGBoost achieves the high accuracy (80.96%) but has a large RMSE, indicating difficulty with high-level curves. The FT-Transformer has the highest accuracy(82.98%) but also struggles with calibration and has high RMSE. The neural network has the lowest accuracy (68.71%) but a lower RMSE, suggesting it captures general trends but not exact values. Figure 7 illustrates these results. Overall, XGBoost is the most robust, while the other models perform well only within the training distribution. Generalizing to rare, high-gonality cases remains challenging.

5. Symbolic Regression for $\mathbb{Q}$-Gonality Prediction Using LLM-Boost

Our second objective is to uncover an explicit symbolic expression that accurately models the $\mathbb{Q}$-gonality of modular curves. While high-precision predictive models—such as those produced by black-box machine learning methods—can capture complex patterns in the data, they often lack interpretability and insight into the underlying mathematical structure. In contrast, symbolic regression (SR) aims to identify concise, human-readable formulas that approximate the true functional relationship. Given a training dataset $D_{\text{train}} = (x_i, y_i)_{i=1}^n$, where each x_i encodes curve-specific features and y_i denotes the associated $\mathbb{Q}$-gonality, our goal is to discover a symbolic function $\tilde{f}$ such that $\tilde{f}(x_i) \approx y_i$ for all i, while preserving interpretability and the potential for generalization to unseen instances. By recovering such an expression, we aim not only to achieve accurate predictions, but also to shed light on the latent mathematical principles governing $\mathbb{Q}$-gonality.

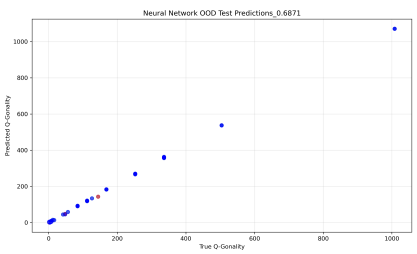
Traditional SR approaches, such as genetic programming and sparse regression, operate over structured equation spaces defined by expression trees or grammars (Cranmer, 2023; Brence et al., 2020), while more recent work has incorporated language models to generate program-like hypotheses that offer greater expressiveness (Biggio et al., 2021; Petersen et al., 1912). Despite progress, these methods often struggle with scalability and efficiency in high-dimensional or scientific domains, where the relevant features are neither obvious nor given. To address this challenge, we introduce LLM-boost, a novel SR framework that combines symbolic regression with LLM-driven feature construction and iterative model refinement. Our method leverages the generative capabilities of LLMs to propose candidate features, evaluates their utility in improving predictive accuracy, and selectively incorporates them into a growing symbolic model. Below, we detail the core components of this algorithm and demonstrate how this iterative boosting framework enables efficient exploration of symbolic feature space while preserving interpretability and accuracy.

5.1. LLM-boost Methodology

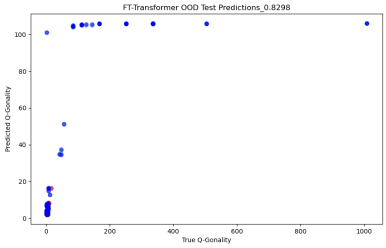
To recover an interpretable symbolic expression, we assume that the true functional relationship can be approximated by an additive model of the form

$$\tilde{f}(x) = \sum_{i=1}^{k} w_i f_i(x),$$

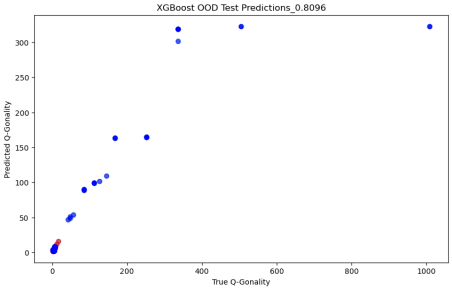
where each f_i is a simple, non-linear feature function. These candidate features are generated by a pre-trained language model π_θ, conditioned on dynamically updated prompts p_i, i.e., $f_i = \pi_\theta(\cdot \mid p_i)$. The prompts are adaptively modified to make sure no repeated feature is produced.



(a) Neural Network OOD Test Predictions



(b) FT-Transformer OOD Test Predictions



(c) XGBoost OOD Test Predictions

Figure 7. Out-of-distribution test predictions for different models. The plots show predicted vs. true $\mathbb{Q}$-gonality values, with accuracy scores displayed in the titles. Red dashed lines indicate perfect predictions. Blue points indicate wrong predictions.

To construct the ensemble of features, we adopt a boosting-inspired approach similar to AdaBoost. The pipeline is listed in figure 8. At each iteration k, we compute the residual of the current model $\tilde{f}_k = \sum_{i=1}^{k} w_i f_i$, and query the LLM to generate additional candidate features. These features are stored in a memory buffer. We then fit the residual using each feature in the buffer and select the one that yields the greatest improvement in predictive accuracy. This selected feature is incorporated into the model, and the process is repeated. This process is equivalent to linear regression on seleted features by Frisch-Waugh-Lovell Theorem if our initial factor model is linear regression model. However, this viewpoint allows us to generalize further. See Appendix C for more details. Unlike standard regression models optimized purely for mean squared error, our final model prioritizes interpretability and accuracy, often resulting in lower bias and more meaningful symbolic expressions. The greedy selection strategy balances the exploitation of promising features with the exploration of novel ones, enabling efficient and interpretable symbolic discovery. A more detailed discussion can be found in Appendix C.

5.2. Experiments

In our experiment, we use GPT 4-o mini (via OpenAI api) as our feature generator. We choose the base additive model $\tilde{f}_0$ to be the polynomial model of order at most 3. We set the iteration to be 100 with the memory buffer of 20. The

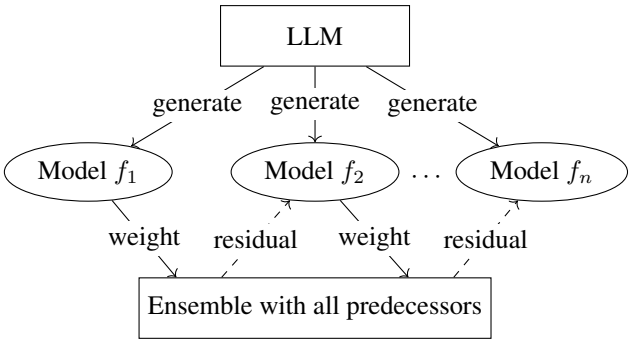


Figure 8. LLM-Boost

training and accuracy can be seen in figure 9. As it is shown in the figure 9, at the training stage, several flat regions indicate the LLM is trying to construct the feature that can significantly improve the accuracy. During 100 iterations, we found 19 significant features, which will be listed in the Appendix D.

The performance results of LLM-Boost are summarized in 10. We also compare the symbolic regression results with the baseline LLM-SR (Shojaee et al., 2024), PySR and GP-learn. The results are summarized in table 16 ,12 and 13. Our LLM-Boost model outperformeds the standard baseline in both accuracy and R^2. In particular, our model shows a

strong predictive power on the D_{bounds}, which even catch up the predictive power of the above "black box" models.

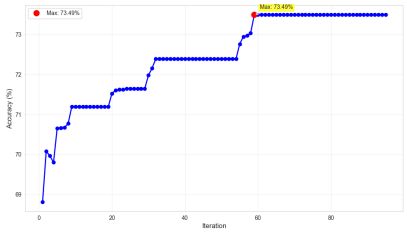


Figure 9. Training and Accuracy figure for LLM-Boost training

Table 10. Performance metrics for the LLM-Boost model across different datasets

Dataset	Accuracy	R^2	RMSE
D_{train}	73.49%	0.9973	0.5913
D_{val}	72.91%	0.9945	0.6718
D_{test}	73.50%	0.9895	1.9814
D_{bounds}	81.52%	–	–

Table 11. Performance metrics for the LLM-SR model across different datasets

Dataset	Accuracy	R^2	RMSE
D_{train}	36.98%	0.9919	1.0153
D_{val}	37.86%	0.9871	0.6718
D_{test}	35.88%	0.9973	0.9874
D_{bounds}	56.47%	–	–

Table 12. Performance metrics for the PySR model across different datasets

Dataset	Accuracy	R^2	RMSE
D_{train}	64.14%	0.9815	1.5359
D_{val}	65.10%	0.9877	1.0095
D_{test}	64.09%	0.9732	3.1606
D_{bounds}	68.00%	–	–

Among the 19 features that is generated by LLM, we found that most features has coefficients roughly 0. However, there is an interesting feature that has a comparatively large linear regression coefficient (around 0.4) $\frac{\log(\text{cusps}+1)\ \log(\text{rank}+1)}{\exp(\text{genus})}$, (around 0.5) $\frac{\log(\text{cusps}+1)\ \log(\text{rank}+1)}{\exp(\text{genus}+1)}$, (around 0.8) $\log(\text{cusps}+1) \times \text{rank}^2 \times \exp(\text{genus}/(\text{level}+1))$. This new finding may give some interesting insight into mathematical conjectures and proofs.

Table 13. Performance metrics for the GP-learn model across different datasets

Dataset	Accuracy	R^2	RMSE
D_{train}	68.48%	0.9293	3.0022
D_{val}	68.07%	0.9421	2.1877
D_{test}	70.32%	0.9208	5.4378
D_{bounds}	62.17%	–	–

6. Conclusion and Future Work

This work shows that machine learning can effectively predict the $\mathbb{Q}$-gonality of modular curves, a challenging invariant in arithmetic geometry. We evaluated XGBoost, feedforward neural networks, and Feature Tokenizer Transformers, all achieving over 90% exact-match accuracy on known cases. Our LLM-Boost method outperformed symbolic regression baselines while remaining interpretable, producing accurate, human-readable equations. The model also identified new features that may shed light on interactions between arithmetic invariants.

Future work includes incorporating more non-numerical features, exploring the predictability of bounded features, and optimizing the LLM-Boost approach as outlined in Appendix C, such as dynamic prompt management and improved search strategies.

Acknowledgements

We thank the anonymous reviewers for their constructive feedback and suggestions, which have improved the quality of this work. The first author gratefully acknowledges Nathan Kaplan, the first author's advisor, for insightful discussions on modular curves and the applications of AI to mathematics, as well as Zeyi Xu for introducing the AI4MATH workshop and providing valuable suggestions on drafting the paper.

Impact Statement

This paper aims to advance the field of Machine Learning as applied to pure mathematics, specifically by employing machine learning techniques to predict arithmetic invariants of modular curves. The research is focused on expanding mathematical knowledge and computational methods within the specialized domain of arithmetic geometry. There are no apparent negative societal consequences or ethical concerns associated with this work.

References

Abramovich, D. A linear lower bound on the gonality of modular curves. *International Mathematics Research Notices*, 1996(20):1005–1011, 01 1996. ISSN 1073-7928. doi: 10.1155/S1073792896000621. URL https://doi.org/10.1155/S1073792896000621.

Alessandretti, L., Baronchelli, A., and He, Y.-H. Machine learning meets number theory: the data science of birch–swinnerton-dyer. In *Machine Learning: In Pure Mathematics And Theoretical Physics*, pp. 1–39. World Scientific, 2023.

Babei, A., Charton, F., Costa, E., Huang, X., Lee, K.-H., Lowry-Duda, D., Narayanan, A., and Pozdnyakov, A. Learning euler factors of elliptic curves. *arXiv preprint arXiv:2502.10357*, 2025.

Biggio, L., Bendinelli, T., Neitz, A., Lucchi, A., and Parascandolo, G. Neural symbolic regression that scales. In *International Conference on Machine Learning*, pp. 936–945. Pmlr, 2021.

Blum, A. and Mitchell, T. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pp. 92–100, 1998.

Brence, J., Todorovski, L., and Džeroski, S. Probabilistic grammars for equation discovery. *arXiv preprint arXiv:2012.00428*, 2020.

Charton, F., Ellenberg, J. S., Wagner, A. Z., and Williamson, G. Patternboost: Constructions in mathematics with a little help from ai. *arXiv preprint arXiv:2411.00566*, 2024.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Collins, M. and Singer, Y. Unsupervised models for named entity classification. In *1999 Joint SIGDAT conference on empirical methods in natural language processing and very large corpora*, 1999.

Cranmer, M. Interpretable machine learning for science with pysr and symbolicregression. jl. *arXiv preprint arXiv:2305.01582*, 2023.

Derickx, M. and van Hoeij, M. Gonality of the modular curve x1(n). *Journal of Algebra*, 417:52–71, 2014. ISSN 0021-8693. doi: https://doi.org/10.1016/j.jalgebra.2014.06.026. URL https://www.sciencedirect.com/science/article/pii/S0021869314003585.

Dery, L. M., Nachman, B., Rubbo, F., and Schwartzman, A. Weakly supervised classification in high energy physics. *Journal of High Energy Physics*, 2017(5):1–11, 2017.

Dong, B., He, X., Jin, P., Schremmer, F., and Yu, Q. Machine learning assisted exploration for affine deligne–lusztig varieties. *Peking Mathematical Journal*, pp. 1–50, 2024.

Ellenberg, J. S., Fraser-Taliente, C. S., Harvey, T. R., Srivastava, K., and Sutherland, A. V. Generative modeling for mathematical discovery. *arXiv preprint arXiv:2503.11061*, 2025.

Fan, W., Liu, K., Liu, H., Wang, P., Ge, Y., and Fu, Y. Autofs: Automated feature selection via diversity-aware interactive reinforcement learning. In *2020 IEEE International Conference on Data Mining (ICDM)*, pp. 1008–1013. IEEE, 2020.

Gorishniy, Y., Rubachev, I., Khrulkov, V., and Babenko, A. Revisiting deep learning models for tabular data. *Advances in neural information processing systems*, 34: 18932–18943, 2021.

Hashemi, B., Corominas, R. G., and Giacchetto, A. Can transformers do enumerative geometry? In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4X9RpKH4Ls.

He, Y.-H. Machine-learning mathematical structures. *International Journal of Data Science in the Mathematical Sciences*, 1(01):23–47, 2023.

He, Y.-H., Lee, K.-H., Oliver, T., and Pozdnyakov, A. Murmurations of elliptic curves. *Experimental Mathematics*, pp. 1–13, 2024a.

He, Y.-H., Yao, Z.-G., and Yau, S.-T. Distinguishing calabi-yau topology using machine learning. *arXiv preprint arXiv:2408.05076*, 2024b.

LMFDB Collaboration, T. The L-functions and modular forms database, home page of modular curve. https://www.lmfdb.org/knowledge/show/modcurve, 2025. [Online; accessed 27 May 2025].

Merler, M., Dainese, N., and Haitsiukevich, K. In-context symbolic regression: Leveraging language models for function discovery. *CoRR*, 2024.

Orlić, P. *Gonality of modular curves and their quotients*. PhD thesis, University of Zagreb. Faculty of Science. Department of Mathematics, 2025.

Petersen, B. K., Landajuela, M., Mundhenk, T. N., Santiago, C. P., Kim, S. K., and Kim, J. T. Deep symbolic regression: Recovering mathematical expressions from data via

risk-seeking policy gradients. arxiv 2021. *arXiv preprint arXiv:1912.04871*, 1912.

Poonen, B. Gonality of modular curves in characteristic p. *arXiv preprint math/0601141*, 2006.

Ren, Z., Wang, S., and Zhang, Y. Weakly supervised machine learning. *CAAI Transactions on Intelligence Technology*, 8(3):549–580, 2023.

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J., Ellenberg, J. S., Wang, P., Fawzi, O., et al. Mathematical discoveries from program search with large language models. *Nature*, 625 (7995):468–475, 2024.

Scudder, H. Probability of error of some adaptive pattern-recognition machines. *IEEE Transactions on Information Theory*, 11(3):363–371, 1965.

Shojaee, P., Meidani, K., Gupta, S., Farimani, A. B., and Reddy, C. K. Llm-sr: Scientific equation discovery via programming with large language models. *arXiv preprint arXiv:2404.18400*, 2024.

Zhou, Z.-H. A brief introduction to weakly supervised learning. *National science review*, 5(1):44–53, 2018.

A. Mathematical Background

We begin by recalling the classical definition of a modular curve. Let

$$\mathbb{H} := \{z = x + yi \in \mathbb{C} \mid y > 0\}$$

denote the upper half-plane. An element $\gamma \in \mathrm{GL}_2(\mathbb{R})$ acts on $\mathbb{H}$ by Möbius transformation. Specifically, for $z \in \mathbb{H}$ and

$$\gamma = \begin{bmatrix} a & b \\ c & d \end{bmatrix},$$

the action is given by

$$\gamma \cdot z := \frac{az + b}{cz + d}.$$

Let Γ be a subgroup of the discrete group $\mathrm{SL}_2(\mathbb{Z})$, and consider the quotient space

$$Y(\Gamma) := \Gamma\backslash\mathbb{H},$$

which can be given a complex structure to be a complex manifold. In number theory, one often considers the following standard congruence subgroups of $\mathrm{SL}_2(\mathbb{Z})$ for a positive integer N:

$$\Gamma(N) := \left\{ \gamma \in \mathrm{SL}_2(\mathbb{Z}) \,\middle|\, \gamma \equiv \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \mod N \right\},$$

$$\Gamma_0(N) := \left\{ \gamma \in \mathrm{SL}_2(\mathbb{Z}) \,\middle|\, \gamma \equiv \begin{bmatrix} * & * \\ 0 & * \end{bmatrix} \mod N \right\},$$

$$\Gamma_1(N) := \left\{ \gamma \in \mathrm{SL}_2(\mathbb{Z}) \,\middle|\, \gamma \equiv \begin{bmatrix} 1 & * \\ 0 & 1 \end{bmatrix} \mod N \right\}.$$

These subgroups satisfy the inclusions

$$\Gamma(N) \subset \Gamma_1(N) \subset \Gamma_0(N) \subset \mathrm{SL}_2(\mathbb{Z}).$$

Sometimes, one considers subgroups $\Gamma \subset \mathrm{GL}_2(\mathbb{Z})$ that contain $\Gamma(N)$ for some N. In such cases, the largest integer N for which $\Gamma \supset \Gamma(N)$ is called the *level* of Γ. These special types of subgroups are of particular interest because the quotient space $\Gamma\backslash\mathbb{H}$ often admits a natural interpretation as a moduli space of elliptic curves with additional structure. This moduli-theoretic viewpoint is one of the main reasons why modular curves are central objects in arithmetic geometry.

Although the quotient space $Y(\Gamma) = \Gamma\backslash\mathbb{H}$ is a complex manifold, it is not compact. However, one can compactify it by adding a finite set of additional points, called *cusps*, to obtain a compact complex manifold denoted by $X(\Gamma)$. This compactification $X(\Gamma)$ is known as the *modular curve* associated to Γ. There exists a natural map from the modular curve to the Riemann sphere,

$$j : X(\Gamma) \to \mathbb{P}^1,$$

called the j-invariant map. The preimage of the point at infinity under this map corresponds precisely to the set of cusps. The genus of the compact Riemann surface $X(\Gamma)$ is also referred to as the *genus of the group* Γ.

Up to this point, we viewed modular curves as complex manifolds over the field $\mathbb{C}$. However, modular curves can also be regarded as algebraic varieties defined over number fields. For instance, when $\Gamma = \Gamma_0(N)$ or $\Gamma_1(N)$, the associated modular curve $X(\Gamma)$ admits a model defined over the rational field $\mathbb{Q}$. In other words, $X(\Gamma)$ can be described as the zero locus of a system of polynomial equations with coefficients in $\mathbb{Q}$. Moreover, the j-invariant map

$$j : X(\Gamma) \to \mathbb{P}^1$$

can also be defined over $\mathbb{Q}$. A natural arithmetic question is to determine the minimal degree of non-constant morphisms from $X(\Gamma)$ to $\mathbb{P}^1$ defined over a given number field $\mathbb{K}$. This minimal degree is called the $\mathbb{K}$-*gonality* of $X(\Gamma)$ and serves as a measure of the arithmetic complexity of the modular curve over the field $\mathbb{K}$.

In general, the gonality of a modular curve is difficult to compute explicitly, and there is no known formula that expresses it in terms of invariants of modular forms. However, one can often obtain upper and lower bounds for the gonality using various techniques, and in some cases, these bounds allow for the exact determination of the gonality.

In the above discussion, we considered subgroups $\Gamma \subset \mathrm{SL}_2(\mathbb{Z})$. However, in number theory, it is common to study such objects from a p-adic perspective. The inverse limit $\mathbb{Z}_p = \varprojlim_n \mathbb{Z}/p^n\mathbb{Z}$ defines the ring of p-adic integers. More generally, taking the inverse limit over all positive integers yields the profinite completion of $\mathbb{Z}$: $\widehat{\mathbb{Z}} := \varprojlim_n \mathbb{Z}/n\mathbb{Z}$. One can consider subgroups $\Gamma \subset \mathrm{GL}_2(\widehat{\mathbb{Z}})$, which encode congruence information at all primes simultaneously.

The database LMFDB adopts this adelic viewpoint in its definition of modular curves, by considering subgroups of $\mathrm{GL}_2(\widehat{\mathbb{Z}})$ (LMFDB Collaboration, 2025). Although certain invariants of modular curves may be defined differently in the LMFDB framework, they agree with the classical definitions.

B. LLM-boost Feature Generation Prompt

The following prompt was used for generating features in our LLM-boost approach:

DATA DESCRIPTION
You are given a table with the following features:
• level: the level of the modular curve
• cusps: the number of cusps of the modular curve
• rational_cusps: number of cusps defined over $\mathbb{Q}$
• genus: genus of the modular curve
• rank: Mordell-Weil rank of the Jacobian
• log_conductor: The natural logarithm of the conductor of the modular curve
• coarse_class_num: The number of isomorphism classes in the coarse moduli space associated with the modular curve
• coarse_level: The level parameter associated with the coarse moduli space of the modular curve
• Q-gonality: a property indicating the minimal degree of a nonconstant map to $\mathbb{P}^1$ defined over $\mathbb{Q}$ (your prediction target)

These invariants are known to interact non-linearly. For example, genus and gonality often grow together but not proportionally; cusps and rational cusps relate to modular parametrization; and level is often positively associated with complexity, but not strictly.
Example input row:
`{'level': 2, 'cusps': 1, 'rational_cusps': 0, 'genus': 0, 'rank': 0, 'log_conductor': 0, 'coarse_class_num': 0, 'coarse_level': 0}`
TASK

- Return EXACTLY 1 Python function, ≤ 120 characters, that computes a NEW numeric feature from ONE data row. DO NOT return trivial functions.
- Do NOT return any feature that is already listed in the SAVED FEATURES sections below.
- Do NOT return any feature that is equivalent to the features listed in the SAVED FEATURES sections below.
- Do NOT include any Markdown code block markers in your response.
- For each function you generate, also provide a short reason (1-2 sentences) explaining why this feature could help predict Q-gonality.
- Try to use transcendental functions like log and exp more. Not just rational functions.

FORMAT
`def f1(row): return ...`
Reason: <your reason here>
SAVED FEATURES
`{{saved features}}`

C. Detailed LLM-Boost Methdology

Traditional symbolic regression typically employs genetic programming and transformer-based next-symbol prediction. These token-level prediction methods are useful when the underlying mechanism is simple. However, the limitation is that token-level prediction only captures "local" expression patterns but fails to understand the "global" meaning. An output with good performance may suffer from poor interpretability, especially when the actual underlying relation is unknown and highly complex.

One approach is to use LLMs to generate possible relations and evaluate the output for self-evolution. This idea has appeared in prior work (Shojaee et al., 2024). However, in our case, the relation is highly nonlinear with many variables. Another issue is that `q_gonality` is—although not mathematically proven but demonstrated in the dataset—uniquely determined by the 8 features. Simply using LLMs to generate the true relation tends to use only a few features, and the output is overly simplified compared to the true relation. Another difficulty is that we use accuracy as our metric, which is often zero, as the output resembles "random guessing" with very high MSE. Consequently, the method fails to converge.

To address these problems, we propose LLM-Boost, which combines LLM-generated features with the AdaBoost idea. Instead of using LLMs to predict the complicated equation directly, we assume that the true relation satisfies the additive model:

$$\tilde{f}(x) = \sum_{i=1}^{k} w_i f_i(x),$$

where each f_i is a simple, nonlinear function of features. These candidate features are generated by a pre-trained language model π_θ, conditioned on dynamically updated prompts p_i, i.e., $f_i = \pi_\theta(\cdot|p_i)$. The prompts are adaptively modified to ensure that no repeated features are produced.

To construct the ensemble of features, we adopt a boosting-inspired approach similar to AdaBoost. At each iteration k, the algorithm consists of four stages:

- **Residual stage**: Compute the residual of the current model $\tilde{f}_k = \sum_{i=1}^{k} w_i f_i$:

$$\tilde{\epsilon}_k(x) = y - \tilde{f}_k(x)$$

- **Feature orthogonalization stage**: Extract an LLM-produced feature from a buffer of length 20. For each feature, we first regress it against the previous features:

$$\epsilon_k(x) = f_k(x) - \sum_{i=1}^{k-1} a_i f_i(x)$$

  Then we fit the residual with the orthogonalized feature and update the model:

$$\begin{aligned}\tilde{f}_k(x) &= \tilde{f}_{k-1}(x) + b_k \epsilon_k(x) \\ &= \tilde{f}_{k-1}(x) + b_k \left(f_k(x) - \sum_{i=1}^{k-1} a_i f_i(x) \right)\end{aligned}$$

  where $\tilde{\epsilon}_k(x) = b_k \epsilon_k(x) + \delta$ represents the linear regression model. This eliminates confounding effects from previously generated features and focuses only on the orthogonal direction. Note that this model differs from direct linear regression using features $\{f_i(x)\}$ since all linear coefficients of f_i for $i < k$ are frozen.

- **Feature selection stage**: Compute the accuracy of the ensemble model for each candidate feature in the buffer, and use a greedy method to select the feature that improves accuracy the most. Remove the selected feature from the buffer.

- **Feature generation stage**: Use the LLM to generate additional features to maintain the buffer size.

Theoretically, suppose at iteration $k-1$, the subspace spanned by the previously generated model is $\mathcal{E}_{k-1}$ and the subspace spanned by the memory buffer $\mathcal{B}_k$ is $\mathcal{F}_k$. Since our model is linear, the residual $\tilde{\epsilon}_k$ has an $\mathcal{E}_{k-1}$ component, which measures how much the model $\tilde{f}_{k-1}$ deviates from the linear regression, and an $\mathcal{E}_{k-1}^{\perp}$ component:

$$\tilde{\epsilon}_k = \tilde{\epsilon}_{\mathcal{E}_k} + \tilde{\epsilon}_k^{\perp}$$

Ideally, a model with 100% accuracy would have zero MSE. As k increases, $\tilde{\epsilon}_k^{\perp}$ dominates and $\tilde{\epsilon}_{\mathcal{E}_k} \to 0$. Thus, our optimization at each step can be formulated as:

$$f_k(x) = \arg\max_{v \in \mathcal{E}_{k-1}^{\perp}} \text{Acc}(\tilde{f}_{k-1}(x) + v)$$

As long as the new LLM-produced feature f_k is not in $\mathcal{E}_{k-1}$, the perpendicular direction ϵ_k contributes to the actual residual, ensuring $\mathcal{F}_k \cap \mathcal{E}_{k-1}^{\perp} \neq \emptyset$. In LLM-boost:

$$f_k(x) \approx \arg\max_{v \in \mathcal{F}_k \cap \mathcal{E}_{k-1}^{\perp}} \text{Acc}(\tilde{f}_{k-1}(x) + v)$$

Based on the knowledge of pre-trained LLMs, we can generate numerous potentially useful highly nonlinear features that human experts might not consider. With a sufficiently large function space $\mathcal{F}_k$ to explore, we can select the feature with the largest projection on the residual. To reduce computational complexity, we use:

$$f_k(x) \approx \arg\max_{f_b \in \mathcal{B}_k} \text{Acc}(\tilde{f}_{k-1}(x) + f_b^{\perp})$$

This approach offers potential for future optimization using methods like steepest descent.

To initialize the additive model, we use the 8 base features and their polynomials up to degree 3. Experimental results show that degree 3 achieves the highest accuracy while maintaining balanced R^2 scores. Beyond degree 3, the model appears to overfit, suggesting the final model should not exceed degree 3.

Table 14. Polynomial regression performance across different degrees

Degree	**Accuracy**	R^2	**RMSE**
1	40.55%	0.9880	1.0924
2	63.59%	0.9927	0.8517
3	68.66%	0.9876	1.1085
4	62.46%	0.8769	3.4970
5	62.70%	0.0288	9.8215
6	51.54%	-42.4290	65.6755

A major challenge affecting performance is that as more features are generated and integrated into the model, it becomes increasingly difficult to find features with large perpendicular components in $\mathcal{E}_k^{\perp}$. Most LLM-generated features (e.g., genus$/(\text{level}+1)^2$) are highly correlated. To address this, when accuracy improvement becomes negligible ($< 0.1\%$), we generate additional features (5 at a time, maximum 3 attempts) and explore among these new candidates. We also increase the size of $\mathcal{B}_k$, though the resulting computational complexity is a drawback requiring the LLM to find effective features through exploration.

Potential solutions include continuously updating prompts to encourage more aggressive feature generation (e.g., incorporating more transcendental functions like $\log$ or $\exp$) and implementing adaptive buffer sizes based on current accuracy and improvement from previous iterations. These optimizations remain for future work.

This process is equivalent to linear regression on seleted features by Frisch-Waugh-Lovell Theorem, if the additive model we start with is a linear regression model. The prove is not hard. Intuitively speaking, suppose $\tilde{f}_{k-1}$ is the linear regression model, then the residual $\tilde{\epsilon}_k$ is orthogonal to $\mathcal{E}_{k-1}$. Everytime we add a new feature, our orthonogonalization process is adjusting the coefficients in the previous generated features as well, make the overall model such that the residual is orthogonal to the $\mathcal{E}_k$, hence it is a linear regression model as long as the linear model we start with is the linear regression

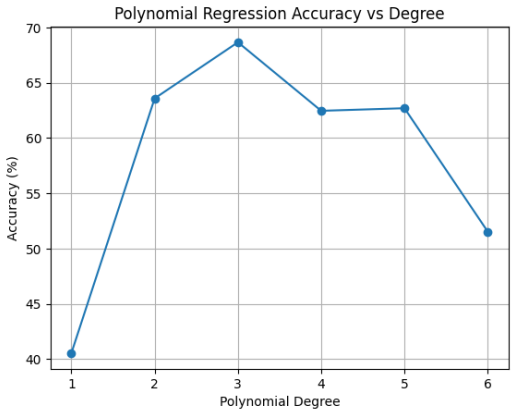


Figure 10. Polynomial degree versus accuracy

model. Although in our case, since the polynomial fitted model is linear regression, makes our final model nothing but the linear regression with greedy feature selecting, the advantage is that we can plug in any linear model as our base model and combine this with LLM-Boost frame work, which will, in the end, deviate from linear regression if our base model is some explainable model which is not linear regression. We will explore this method further for more applications with base model which is not linear regression model.

We explored alternative approaches for integrating LLMs into symbolic regression. One successful variant, **pool boost**, generates a large feature set (approximately 175 features) initially and then applies boosting ensemble. This method achieved higher accuracy (77.20%) on the training set due to having a larger initial memory buffer (175 vs. 20), enabling better optimization directions. However, the computational complexity is prohibitive, as our greedy method requires n^2 linear regressions, and the search space shrinks as good features are removed from the buffer. As shown in Figure 11, accuracy plateaus after approximately 100 iterations, highlighting the insufficient feature generation problem.

Table 15. Performance metrics for Pool Boost across different datasets

Dataset	**Accuracy**	R^2	**RMSE**
D_{train}	77.20%	0.9979	0.5533

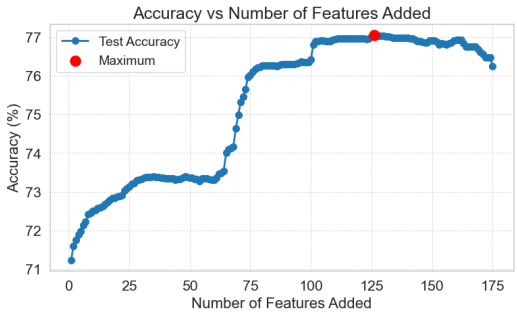


Figure 11. Iteration versus accuracy for pool boost method

One may wonder using the newly generated feature for regression directly, just like the original boosting using the weak learning fitting the residual. We have done the experiments to compare the boosting with orthogonalization (we call it linear regression) and without orthogonalization (we call it residual boosting), we have the following comparison result for the accuracy in Figure 12. We have observed the fact that residual boosting method have a relative slow speed of accuracy increasing compare to the linear regression. One possible explanation is that since the newly added feature is highly correlated to the previous model, the fitted residual will have the component in the space of $\mathcal{E}_k$, which will destroy

the fitted model f_k and make the whole fitting process unstable. So we do not use this way.

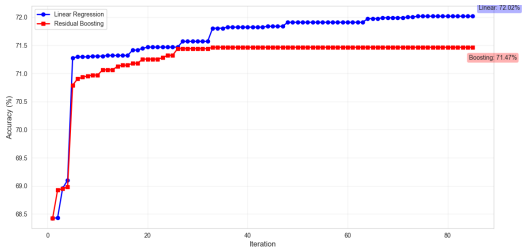


Figure 12. Iteration versus accuracy for pool boost method

Another approach involved LLM-based feature generation combined with reinforcement learning (RL) for feature selection. Optimal subset selection requires 2^n linear regressions, and approximately 100 generated features are needed to avoid bias, making naive feature selection computationally infeasible. We implemented RL-based feature selection following prior work (Fan et al., 2020), incorporating polynomial features up to degree 3 (over 300 total features). This achieved reasonable accuracy (74%), but convergence was slow and scalability limited, leading us to abandon this approach.

We also compare our method with LLM-SR in (Shojaee et al., 2024). The major difference between our method and LLM-SR is that LLM-SR is using LLM to produce the whole equation containing the nonlinear features, with undetermined coefficients, and fit the equation using the training data, while our method use LLM to produce each individual feature at a time, and ensemble them dynamically. We run the LLM-SR 100 rounds use GPT 4-o mini via OpenAI api. The result is as follow:

Table 16. Performance metrics for the LLM-SR model across different datasets

Dataset	**Accuracy**	R^2	**RMSE**
D_{train}	36.98%	0.9919	1.0153
D_{val}	37.86%	0.9871	0.6718
D_{test}	35.88%	0.9973	0.9874
D_{bounds}	56.47%	–	–

The produced equation:

$$\begin{aligned}\text{Q-gonality} = {} & 0.113 \times \text{genus} \\ & - 0.347 \times log(\text{level} + 1) \\ & - 0.760 \times (\text{rational cusps}/(1 + \text{genus})) \\ & + 0.202 \times exp(-\text{rank}) \\ & + 0.027 \times \text{log conductor} \\ & + 0.011 \times (\text{cusps}/(1 + \text{rank})) \\ & - 0.026 \times sqrt(\text{coarse class num} + 1) \\ & + 0.003 \times \text{ coarse level} \\ & + 2.834\end{aligned}$$

We can see that R^2 score and RMSE are within satisfying level, but the accuracy is not that good for this task. There are several causes. One is that, unlike the usual SR task, our task is very complicated, with no certain obvious solution. The LLM-SR is trying to solve the problem as a whole with no other context, so LLM is trying to produce the output out of the pre-trained parameters. In the LLM's pre-trained data, there are no "super-long" equations, so LLM tends to output short and concise equation. The R^2 is high, which indicates it captures the pattern of the data. But our task is subtle, high R^2 score and low RMSE are not enough. Another cause is that even if we have a good output equation at some round, it is hard

to let LLM to recognize it and do the self-evolve. One possible scenario is that one out of ten factor in the equation has the major contribution, but the model is hard to extract that factor out and keep it. Instead, the model just forget the this equation completely and enter the next round. Thus, it makes LLM-SR hard to converge in our task. In addition, LLM often suffers from the hallucination for the long-structured output. Most of the time during the LLM-SR training, the output will have errors, even for the SOTA model.

Our LLM-Boost model partially solved the above shortcomings of LLM-SR: Due to the nature of boosting, our model can produce as- long-as-possible factor models, which is more flexible for complicated problems; We greedily kept all the good produced features and searching for the new features, make LLM to search for the features that we haven't seen before besides the generated good features; Our structured output is relatively short, which makes the LLM makes less mistake and generate more valuable solutions instead of stuck in the hallucination. Another advantage is that our model can leverage the a prior knowledge of the problem. In our example, we combine the polynomial regression up to degree 3 to the LLM-boost and achieve a nice result. We can combine LLM-Boost model with any model that have good performance and improve the model via new feature generating procedure.

D. The Generated Features and Coefficients

```
    poly_level -1.327062015118409963e-05
poly_genus 1.823347690467706349e-04
poly_rank -2.076470839180304150e-05
poly_cusps 1.340799089602361821e-04
poly_rational_cusps -2.609911410237986319e-05
poly_log_conductor 6.248694805191864089e-04
poly_coarse_class_num 5.636849755390863775e-03
poly_coarse_level 6.708807306380486256e-05
poly_level^2 -1.195814052239893305e-03
poly_level genus 1.862740451982679030e-03
poly_level rank -8.841007439528143601e-04
poly_level cusps 9.833585879028660780e-04
poly_level rational_cusps -2.190313095123194850e-04
poly_level log_conductor 2.049228649813721222e-03
poly_level coarse_class_num -4.121849647903544963e-04
poly_level coarse_level 5.097574401708673169e-04
poly_genus^2 -1.534202285711756906e-05
poly_genus rank -4.676882655741077612e-05
poly_genus cusps 1.029784661520296825e-03
poly_genus rational_cusps 1.972805877782021766e-04
poly_genus log_conductor -9.383440531478978936e-04
poly_genus coarse_class_num 1.457099679979013962e-04
poly_genus coarse_level 1.025009020109483649e-03
poly_rank^2 2.181740936330784655e-04
poly_rank cusps -5.042013758272134456e-04
poly_rank rational_cusps -2.044770115067380630e-06
poly_rank log_conductor 9.345158559204242918e-04
poly_rank coarse_class_num 5.232302109540386644e-05
poly_rank coarse_level -1.030793328823869943e-03
poly_cusps^2 3.126149915010396820e-03
poly_cusps rational_cusps -1.107899075650206936e-04
poly_cusps log_conductor 5.590599292031519108e-04
poly_cusps coarse_class_num -2.561388874679910153e-04
poly_cusps coarse_level 7.136032158511422356e-04
poly_rational_cusps^2 -9.910577691374199269e-05
poly_rational_cusps log_conductor 8.547352050544935890e-04
poly_rational_cusps coarse_class_num 6.593289326506223812e-06
poly_rational_cusps coarse_level -2.175898873615241105e-05
poly_log_conductor^2 4.979681423365633799e-05
poly_log_conductor coarse_class_num 7.259466475130400040e-06
poly_log_conductor coarse_level -1.982542652373057816e-03
poly_coarse_class_num^2 -9.475333273964632076e-07
poly_coarse_class_num coarse_level 1.862178310965380214e-04
poly_coarse_level^2 5.611850888484727021e-04
poly_level^3 1.680092543237133446e-05
poly_level^2 genus 2.020787844345689629e-04
poly_level^2 rank 1.771447656564843504e-05
poly_level^2 cusps -1.948633330835962754e-05
poly_level^2 rational_cusps -6.978797251483502117e-05
poly_level^2 log_conductor -5.996978102596058850e-05
poly_level^2 coarse_class_num 4.235857521143931154e-06
poly_level^2 coarse_level -2.782126352434376171e-06
poly_level genus^2 -5.651256831389802606e-04
poly_level genus rank 4.682424996181275842e-04
poly_level genus cusps 2.697470188116504568e-04
poly_level genus rational_cusps -1.234418368317621558e-03
poly_level genus log_conductor 5.966035629675626702e-05
poly_level genus coarse_class_num -6.808476917525402815e-07
poly_level genus coarse_level -3.582118937474009467e-04
poly_level rank^2 -1.522938683644250578e-04
poly_level rank cusps 4.541620674624330124e-05
poly_level rank rational_cusps 7.400183648088708849e-04
poly_level rank log_conductor -6.522603632815835188e-05
poly_level rank coarse_class_num -2.097862978457524022e-06
poly_level rank coarse_level -8.676257791191593067e-06
poly_level cusps^2 -5.136442002745767116e-05
poly_level cusps rational_cusps 2.705536148815437217e-04
poly_level cusps log_conductor -3.245307788265713346e-05
poly_level cusps coarse_class_num 1.494642725522539635e-06
poly_level cusps coarse_level -2.056897306573524402e-05
poly_level rational_cusps^2 3.967030219352915684e-04
poly_level rational_cusps log_conductor 1.476008966897829279e-04
poly_level rational_cusps coarse_class_num 4.219322940853618823e-06
poly_level rational_cusps coarse_level 8.265842031454741559e-05
poly_level log_conductor^2 3.907935106155094665e-06
poly_level log_conductor coarse_class_num -6.713531461182992849e-07
poly_level log_conductor coarse_level 8.265500015080245795e-05
poly_level coarse_class_num^2 1.871361652553034383e-08
poly_level coarse_class_num coarse_level 3.325068389520215596e-06
poly_level coarse_level^2 -1.690626705021128247e-05
poly_genus^3 -6.140611517552104501e-04
poly_genus^2 rank -4.185284753619091663e-04
poly_genus^2 cusps 9.468317629571724580e-04
poly_genus^2 rational_cusps 8.531671128145793459e-04
poly_genus^2 log_conductor 2.614474560909644130e-04
poly_genus^2 coarse_class_num 1.746774292347177916e-06
poly_genus^2 coarse_level 4.161452192520024391e-04
poly_genus rank^2 -8.590321427641646641e-04
poly_genus rank cusps 2.165962950255898392e-04
```

```
poly_genus rank rational_cusps -5.913591116760164848e-04
poly_genus rank log_conductor 2.094172624711568330e-04
poly_genus rank coarse_class_num -1.525260857742961635e-06
poly_genus rank coarse_level -2.017133432759710826e-04
poly_genus cusps^2 -5.166428988814865961e-04
poly_genus cusps rational_cusps 2.535268294424324140e-03
poly_genus cusps log_conductor -2.510163624197777384e-04
poly_genus cusps coarse_class_num -5.417190126548015452e-07
poly_genus cusps coarse_level 1.250320847756399500e-04
poly_genus rational_cusps^2 2.262765426519916740e-04
poly_genus rational_cusps log_conductor -3.133232319251227785e-04
poly_genus rational_cusps coarse_class_num -1.835399585809588917e-06
poly_genus rational_cusps coarse_level 7.167793619273178304e-04
poly_genus log_conductor^2 -3.990199434816409270e-05
poly_genus log_conductor coarse_class_num -4.224972972050467911e-07
poly_genus log_conductor coarse_level -3.164465594356589973e-05
poly_genus coarse_class_num^2 2.2958415330518247 57e-09
poly_genus coarse_class_num coarse_level -3.435154013564254913e-06
poly_genus coarse_level^2 1.557440463059202578e-04
poly_rank^3 -2.206169087159739453e-04
poly_rank^2 cusps 2.517885953645471815e-04
poly_rank^2 rational_cusps 1.435723120960826285e-03
poly_rank^2 log_conductor 1.536868641116519076e-04
poly_rank^2 coarse_class_num -4.049148360572418087e-07
poly_rank^2 coarse_level 2.305645664616038845e-04
poly_rank cusps^2 -1.780908232557274211e-05
poly_rank cusps rational_cusps -1.399488565329560889e-03
poly_rank cusps log_conductor -6.143193556064204745e-05
poly_rank cusps coarse_class_num -5.472420474846961640e-07
poly_rank cusps coarse_level -1.858053573996192997e-04
poly_rank rational_cusps^2 -4.457549453913680834e-04
poly_rank rational_cusps log_conductor 1.713337418581490702e-04
poly_rank rational_cusps coarse_class_num -3.278395076792683244e-06
poly_rank rational_cusps coarse_level -7.0636670053664338 96e-04
poly_rank log_conductor^2 -2.269729923120889252e-05
poly_rank log_conductor coarse_class_num 3.080169304154820042e-07
poly_rank log_conductor coarse_level 8.715710579810013146e-06
poly_rank coarse_class_num^2 9.644428142392691417e-10
poly_rank coarse_class_num coarse_level 6.201416059487864247e-07
poly_rank coarse_level^2 6.670462052678306428e-05
poly_cusps^3 2.426451657160626088e-05
poly_cusps^2 rational_cusps -3.303768496306857486e-04
poly_cusps^2 log_conductor 7.184591038730792599e-05
poly_cusps^2 coarse_class_num 8.815975974839698748e-07
poly_cusps^2 coarse_level -6.091966072038286939e-05
poly_cusps rational_cusps^2 -4.052370387305384396e-04
poly_cusps rational_cusps log_conductor -2.845870018524358390e-04
poly_cusps rational_cusps coarse_class_num -4.755042330045302951e-06
poly_cusps rational_cusps coarse_level -3.669348233298918606e-04
poly_cusps log_conductor^2 1.757027521795363409e-05
poly_cusps log_conductor coarse_class_num -5.731519852930806735e-08
poly_cusps log_conductor coarse_level -1.893331655588952316e-05
poly_cusps coarse_class_num^2 -2.192525030237854611e-09
poly_cusps coarse_class_num coarse_level 5.152260939854922707e-06
poly_cusps coarse_level^2 3.294999944746538007e-05
poly_rational_cusps^3 -5.380117282533443705e-04
poly_rational_cusps^2 log_conductor -1.317512773506196498e-04
poly_rational_cusps^2 coarse_class_num 2.064413742234347086e-06
poly_rational_cusps^2 coarse_level -4.217178836495345134e-05
poly_rational_cusps log_conductor^2 1.826462539568149000e-05
poly_rational_cusps log_conductor coarse_class_num 6.774077077548526504e-07
poly_rational_cusps log_conductor coarse_level -4.273842231197209137e-05
poly_rational_cusps coarse_class_num^2 1.000364584680286939e-09
poly_rational_cusps coarse_class_num coarse_level -4.564056715353407407e-06
poly_rational_cusps coarse_level^2 -5.539825979886093909e-05
poly_log_conductor^3 2.169038422645327025e-06
poly_log_conductor^2 coarse_class_num 2.614113313488541923e-08
poly_log_conductor^2 coarse_level -3.752324467664420668e-06
poly_log_conductor coarse_class_num^2 -2.376287694448673410e-10
poly_log_conductor coarse_class_num coarse_level 6.466201835009970436e-07
poly_log_conductor coarse_level^2 -2.692637075745932642e-05
poly_coarse_class_num^3 -1.759003370477990602e-12
poly_coarse_class_num^2 coarse_level 1.803167395698088496e-09
poly_coarse_class_num coarse_level^2 -5.512427022435909332e-06
poly_coarse_level^3 -2.541701157208304477e-06
poly_intercept 2.270603522010218533e+00
residual_log_conductor_times_rational_cusps_plus_1_pow_genus_plus_1 2.125453516202150744e-81
residual_log_conductor_times_rational_cusps_plus_1_pow_genus_plus_1_intercept -1.560876998090132420e-10
residual_log1p_cusps_times_rank_plus_1_div_genus_plus_1 -9.431426813113684415e-02
residual_log1p_cusps_times_rank_plus_1_div_genus_plus_1_intercept -1.155422786162470794e-15
residual_log_cusps_plus_1_times_rank_plus_1_pow_genus_div_log_cond... 1.663068729889676314e-02
residual_log_cusps_plus_1_times_rank_plus_1_pow_genus_div_log_cond..._intercept 8.170596954779466015e-02
residual_log_level_plus_1_times_cusps_div_1_plus_exp_genus -4.543345200118077687e-02
residual_log_level_plus_1_times_cusps_div_1_plus_exp_genus_intercept -5.874919639991264125e-02
residual_sqrt_rank_plus_1_times_log_cusps_plus_2_times_1_plus_log_... -2.456167637006575078e-03
residual_sqrt_rank_plus_1_times_log_cusps_plus_2_times_1_plus_log_..._intercept 3.080135601525816127e-02
residual_log_rank_plus_2_times_cusps_plus_1_times_exp_log_conducto... 6.254214190697941487e-08
residual_log_rank_plus_2_times_cusps_plus_1_times_exp_log_conducto..._intercept 2.406059646315631545e-02
residual_log_level_plus_1_times_exp_cusps_div_genus_plus_1_times_rank 3.598737095194208248e-05
residual_log_level_plus_1_times_exp_cusps_div_genus_plus_1_times_rank_intercept -1.180706106149719773e-03
residual_log_cusps_plus_1_times_log_rank_plus_1_div_exp_genus_plus_1 -5.550697620205004013e-01
residual_log_cusps_plus_1_times_log_rank_plus_1_div_exp_genus_plus_1_intercept -2.871006324652216734e-03
```

residual_log_cusps_plus_1_times_log_rank_plus_1_pow_level_div_genu... 1.26155657814037156 3e-03
residual_log_cusps_plus_1_times_log_rank_plus_1_pow_level_div_genu..._intercept 2.78602363154348492 8e-03
residual_log_cusps_plus_1_times_log_rank_plus_2_times_sqrt_log_con... -8.8398382058274031 72e-05
residual_log_cusps_plus_1_times_log_rank_plus_2_times_sqrt_log_con..._intercept -7.9539059436181287 77e-03
residual_log_level_plus_1_times_rational_cusps_plus_1_pow_cusps_di... 3.6385532412824032 04e-07
residual_log_level_plus_1_times_rational_cusps_plus_1_pow_cusps_di..._intercept 7.7330735851721017 77e-03
residual_log_cusps_plus_1_times_rank_pow_2_times_exp_minus_genus_d... 8.2837112548816804 75e-01
residual_log_cusps_plus_1_times_rank_pow_2_times_exp_minus_genus_d..._intercept 4.5870867011580185 66e-03
residual_log_level_plus_1_times_cusps_plus_1_pow_rank_div_genus_pl... 1.1314548762485201 94e-03
residual_log_level_plus_1_times_cusps_plus_1_pow_rank_div_genus_pl..._intercept -1.6303875073431242 65e-02
residual_log_cusps_plus_1_times_exp_rational_cusps_times_level_pow... 3.4665940306751315 57e-06
residual_log_cusps_plus_1_times_exp_rational_cusps_times_level_pow..._intercept -9.0305337162297719 82e-03
residual_log_cusps_plus_1_times_level_pow_2_div_1_plus_exp_genus 1.7196578857792952 95e-05
residual_log_cusps_plus_1_times_level_pow_2_div_1_plus_exp_genus_intercept -4.6862950237921486 34e-04
residual_log_cusps_plus_1_times_log_rank_plus_1_div_exp_genus -4.4570975297878484 56e-01
residual_log_cusps_plus_1_times_log_rank_plus_1_div_exp_genus_intercept -1.5297758410751781 82e-03
residual_log_cusps_plus_1_times_log_rank_plus_2_times_level_pow_1_... 7.6001739183107419 00e-04
residual_log_cusps_plus_1_times_log_rank_plus_2_times_level_pow_1_..._intercept 6.0811238125286423 69e-03
residual_log_cusps_plus_1_times_log_genus_plus_2_times_rank_pow_1_... -6.2234701790411773 27e-04
residual_log_cusps_plus_1_times_log_genus_plus_2_times_rank_pow_1_..._intercept -5.8274113624416177 40e-03

E. Evaluate Feature Importance for `q_gonality`

In this section, we present the results of linear regression for features that exhibit a strong linear relationship with `q_gonality`, as well as the performance of XGBoost when using only coarse features or when excluding them.

Figure 13 shows the correlation matrix of numerical features, while Figure 14 illustrates the relationship between individual features and `q_gonality`. These figures indicate that `genus`, `rank`, `log_conductor`, and `psl2Index` are strongly correlated with `q_gonality`. However, Table 17 demonstrates that linear regression does not achieve high predictive performance, suggesting that these features do not directly encode `q_gonality`.

We also evaluate the performance of XGBoost using only coarse features, as shown in Table 19. The results indicate poor predictive power, implying that coarse features alone do not capture sufficient information about `q_gonality`. In contrast, when coarse features are excluded and only six numerical features are used (Table 18), the predictive performance improves significantly. This demonstrates that coarse features contribute little to the prediction of `q_gonality`.

Table 17. Linear regression performance for individual features predicting `q_gonality`

Feature	R^2	**Accuracy (%)**	**RMSE**	**Accuracy within bounds (%)**
`rank`	0.9171	28.05	2.5468	39.52
`psl2Index`	0.9731	8.06	1.4505	15.74
`genus`	0.9659	22.91	1.6342	10.64
`log_conductor`	0.9635	24.30	1.6895	26.97

Table 18. XGBoost performance using six numerical features (excluding coarse features) for `q_gonality` prediction

Features	R^2	**Accuracy (%)**	**RMSE**
6 numerical features	0.9985	92.61	0.3379

Table 19. XGBoost performance using `coarse_level` and `coarse_class_num` features for `q_gonality` prediction

Features	R^2	**Accuracy (%)**	**RMSE**
coarse_level, coarse_class_num	0.2487	24.69	7.6685

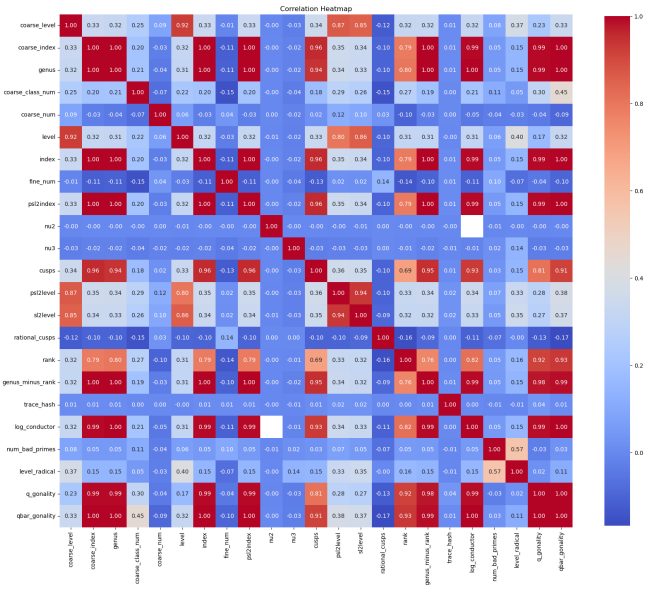


Figure 13. Correlation matrix of numerical features for modular curves from the LMFDB dataset.

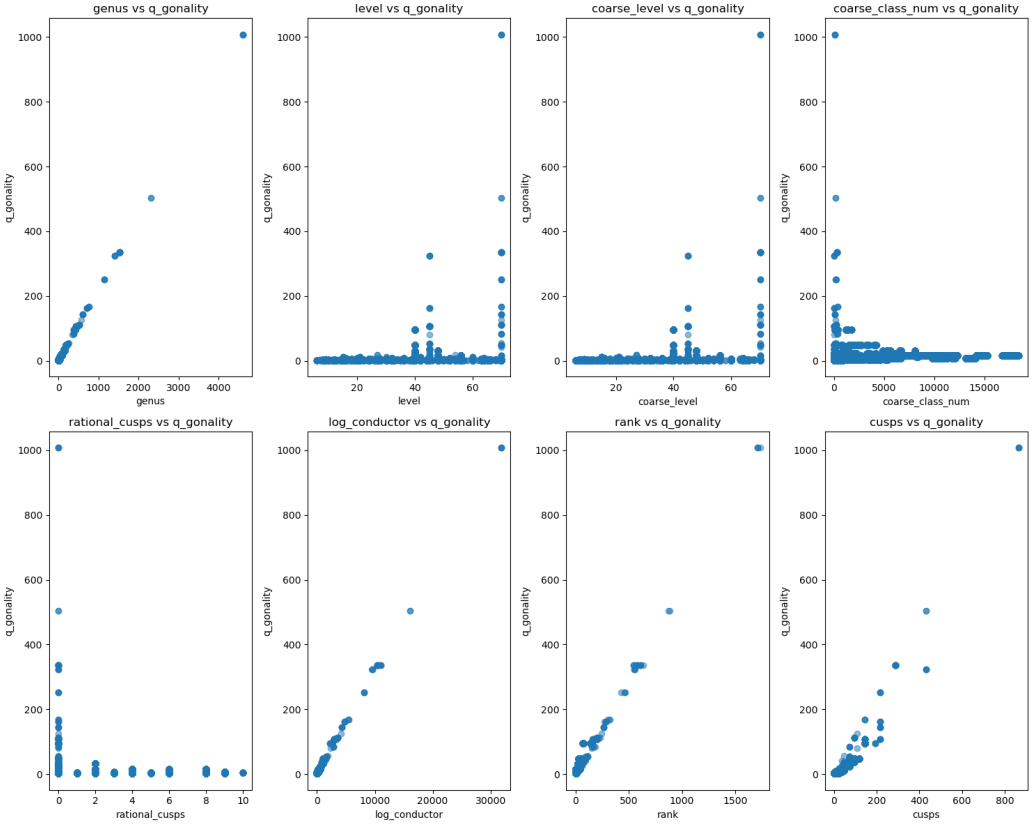


Figure 14. Relationship between individual features and `q_gonality`.

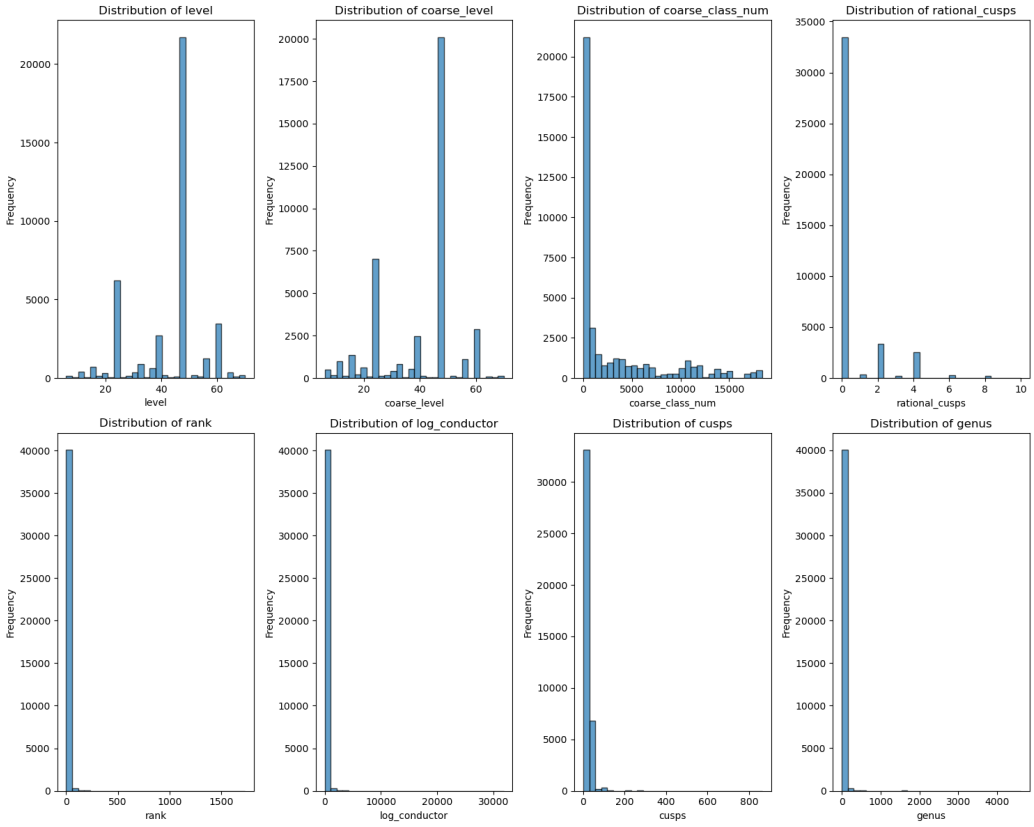


Figure 15. Distribution of feature values.