Optimizing Reasoning LLMs for Automated Feature Engineering via Multi-Stage Direct Preference Optimization

Mingzhi Hu[1] **Menghai Pan**[1] **Yuzhong Chen**[1] **Junpeng Wang**[1]
Yingtong Dou[1] **Vineeth Rakesh**[2] **Hoang Anh Duy Le**[3] **Uday Singh Saini**[1]
Xin Dai[1] **Xiran Fan**[1] **Yuqing Liu**[4] **Yiwei Cai**[1]

[1]**Visa Research** [2]**Adobe** [3]**Rice University** [4]**University of Illinois at Chicago**

{mingzhu,menpan,yuzchen,junpenwa,yidou,udasaini,xidai,xirafan,yicai}@visa.com
vineethm@adobe.com, el72@rice.edu, yliu363@uic.edu

Abstract

Feature engineering is central to tabular machine learning, but useful transformations must be both statistically predictive and semantically valid with respect to feature names, task descriptions, and observed data patterns. Although large language models can exploit such context, prompt-based feature engineering remains unreliable: generated transformations may be redundant, schema-invalid, non-executable, or weakly aligned with downstream performance. We propose MSDPO, a two-stage Direct Preference Optimization framework for automated feature engineering. Proposal-DPO aligns the model toward useful, diverse, and non-redundant feature proposals, while Function-DPO aligns it toward executable feature-construction code using execution and validation feedback. After alignment, the frozen model directly generates transformation functions for unseen tabular tasks. Across 20 classification datasets, MSDPO achieves the best overall mean rank among strong classical and LLM-based feature-engineering baselines. Under a budget-matched comparison inside the same feature-engineering harness, MSDPO also attains a better mean rank than Claude Sonnet 4.6, despite using an 8B open-weight backbone. Moreover, preferences constructed from classification signals transfer to four regression tasks. Ablation results show that the two DPO stages are complementary, improving both feature design and executable implementation.

1 Introduction

Tabular data remains one of the most widely used modalities in practical machine learning, supporting applications in healthcare, finance, risk modeling, operations, and e-commerce (Shwartz-Ziv and Armon, 2022; Huang et al., 2020; Borisov et al., 2022). Despite recent progress in tabular deep learning and foundation models, including Transformer-based and attention-based architectures for tabular prediction (Hollmann et al., 2022; Gorishniy et al., 2021; Somepalli et al., 2021), strong tabular performance still often depends on feature engineering: constructing informative transformations such as ordinal mappings, ratios, interactions, aggregations, discretizations, and nonlinear functions. Effective feature engineering is challenging because useful transformations must be both statistically predictive and semantically valid with respect to the task, schema, and feature meanings.

Conventional automated feature-engineering methods reduce manual effort by searching over predefined transformation spaces, such as relational transformations, symbolic compositions, or operator-based feature candidates (Kanter and Veeramachaneni, 2015; Khurana et al., 2016; Katz et al., 2016; Horn et al., 2019; Zhang et al., 2023). These methods can be effective when useful transformations fall within their operator libraries, but they primarily operate on column types and values rather than natural-language semantics. As a result, they have limited ability to exploit feature names, task descriptions, domain knowledge, or example rows, which are often important for constructing semantically meaningful features.

Large language models (LLMs) offer a natural way to exploit such semantic context. Given feature names, task descriptions, and a few demonstration rows, an LLM can propose transformations that reflect domain meaning rather than only syntactic operator search. Recent work uses LLMs for tabular prediction and feature engineering, including Python transformations, feature rules, decision-tree-style expressions, or candidate features refined through execution and validation feedback (Hegselmann et al., 2023; Hollmann et al., 2023; Abhyankar et al., 2025; Han et al., 2024; Nam et al., 2024; Dong et al., 2026). However, directly prompted LLMs remain unreliable feature

Findings of the Association for Computational Linguistics: IJCNLP-AACL 2026, pages 4599–4618
November 6-10, 2026 ©2026 Association for Computational Linguistics

engineers: their generations may be redundant, weakly predictive, schema-invalid, non-executable, or poorly aligned with downstream model performance. Existing methods often mitigate this through inference-time scaffolds such as iterative prompting, execution filters, validation feedback, or constrained search, leaving the model's underlying preference for useful and executable transformations unchanged.

We instead formulate LLM-based feature engineering as a preference-learning problem. Rather than relying only on per-dataset prompting and filtering, we train the generator to prefer transformations that are valid, executable, interpretable, non-redundant, and useful for downstream prediction. We propose MSDPO, a multi-stage Direct Preference Optimization framework that aligns a reasoning LLM at two levels. Proposal-DPO teaches the model what feature transformations to prefer, while Function-DPO teaches it how to implement them as executable Python feature-construction code. After alignment, the frozen model directly generates a `modify_features` function for unseen datasets using task descriptions, feature metadata, and a few demonstration rows.

This design leads to a single aligned generator that can produce executable feature-construction functions for unseen tabular tasks without per-dataset fine-tuning. Our contributions are:

- We formulate LLM-based feature engineering as a preference-learning problem, training the generator to prefer diverse, valid, executable, non-redundant, and useful transformations.
- We introduce MSDPO, a two-stage DPO framework that aligns the model at both the feature-proposal and function-code levels for cross-dataset feature generation.
- We evaluate MSDPO on 24 tabular datasets spanning classification and regression. MSDPO achieves the best mean rank on both the 20-dataset classification benchmark and the four-dataset regression benchmark. Under the controlled, budget-matched feature-engineering harness, MSDPO achieves a better mean rank than Claude Sonnet 4.6.

2 Overview

2.1 Problem Definition

We study automated feature engineering for supervised tabular prediction. Given a dataset $\mathcal{D} = (X, y)$ with raw feature columns $\mathcal{C}$ and labels y, the goal is to construct an engineered table $\tilde{X}$ that improves the performance of a downstream predictor h. In our setting, each dataset is accompanied by a natural-language task description τ, feature metadata m, and a small set of demonstration rows E. We denote the resulting dataset context as

$$c = (\tau, m, E).$$

The feature-engineering model receives c and produces executable Python code G implementing `modify_features(df)`. Executing G induces a dataframe transformation g_G, which maps the raw table to the engineered table:

$$\tilde{X} = g_G(X).$$

The engineered table $\tilde{X}$ is then used to train and evaluate the downstream learner h. The objective is not merely to generate plausible feature ideas, but to produce transformations that are semantically meaningful, schema-consistent, executable, and beneficial for downstream prediction. This formulation reflects two requirements of practical tabular feature engineering. First, useful transformations often depend on feature semantics rather than data types alone, such as ratios between related measurements, grouped counts over related binary indicators, ordinal recodings of categorical variables, or nonlinear transformations of numerical variables. Second, generated transformations must be executable code that can be applied consistently across training, validation, and test splits without accessing labels or post-outcome information.

2.2 Structured LLM-based Feature Generation

We formulate LLM-based feature engineering as conditional generation from the dataset context c. Given c, the LLM policy π_θ generates a structured response

$$z = (S, G),$$

where $S = \{s_1, \ldots, s_K\}$ is a set of proposed feature transformations, represented by feature names

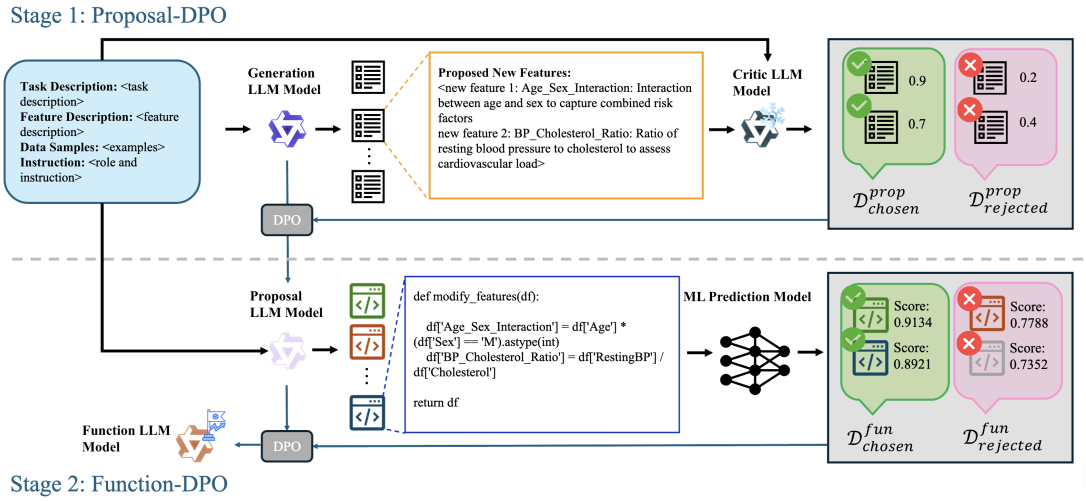


Figure 1: Two-stage DPO framework (MSDPO). **Stage-1 (Proposal-DPO)** scores the feature-proposal span S using a Critic LLM, based on usefulness, diversity, and non-redundancy, and optimizes the full response including both S and the function-code span G. **Stage-2 (Function-DPO)** uses execution validity and validation AUC against a raw-feature baseline to form implementation-level preferences.

and natural-language descriptions, and G is executable Python code implementing these transformations as `modify_features(df)`. This decomposition separates two complementary decisions. The proposal S specifies *what* transformations should be constructed, such as a clinically meaningful ratio, a grouped count over related indicators, or a nonlinear numerical transformation. The code span G specifies *how* those transformations are realized as an executable dataframe transformation g_G. Both are necessary: a feature idea may be semantically useful but fail if implemented incorrectly, while valid code may be unhelpful if the underlying transformation is redundant or irrelevant. We require generated outputs to satisfy three practical constraints. First, they must avoid using variables unavailable at prediction time. Second, they must be schema-consistent, using only valid input columns and behaving deterministically across data splits. Third, they must execute successfully as dataframe transformations. These requirements motivate our two-stage alignment procedure: Proposal-DPO aligns the model toward useful and non-redundant feature proposals, while Function-DPO aligns it toward executable implementations with strong downstream validation performance.

3 Methodology

Building on Section 2, MSDPO applies preference optimization in two stages. Proposal-DPO derives preferences from the feature-proposal span S and optimizes the full response (S, G), teaching the model what transformations to prefer. Function-DPO then uses execution-grounded preferences to align the function-code span G, teaching the model how to implement them reliably. The two stages share the same generator but use different preference signals and loss masks.

3.1 Preference Learning with DPO

DPO aligns a policy π_θ to preferences over outputs without training an explicit reward model (Rafailov et al., 2023; Xu et al., 2024; Pan et al., 2026). Given a context c and two candidate outputs z^+ and z^-, where z^+ is preferred to z^-, DPO minimizes

$$\mathcal{L}_{\mathrm{DPO}}(\theta) = -\mathbb{E}[\log \sigma(\beta\, \Delta(\theta))]\,,$$
$$\Delta(\theta) = \log \frac{\pi_\theta(z^+ \mid c)}{\pi_\theta(z^- \mid c)} - \log \frac{\pi_{\mathrm{ref}}(z^+ \mid c)}{\pi_{\mathrm{ref}}(z^- \mid c)}, \quad (1)$$

where π_{ref} is a fixed reference policy, $\sigma(\cdot)$ is the sigmoid function, and β controls the strength of the preference signal.

We instantiate DPO sequentially in two stages. In both stages, candidates are full structured responses $z = (S, G)$, including both the feature-proposal span and function-code span. Stage 1 derives preferences from the feature-proposal span S while optimizing the full response, using the initial generator as the reference policy. The Stage 1 checkpoint is then used to sample new responses for Stage 2, where preferences are derived from execution and validation feedback on the function-code span G. Stage 2 uses the Stage 1 checkpoint

as the reference policy and applies the DPO loss only to the function-code span.

3.2 Stage-1: Critic-based Proposal-DPO

Stage-1 aligns the model at the *feature-proposal* level. Given a dataset context c, the generator produces a candidate proposal $S = \{s_1, \ldots, s_K\}$ using the generation template in Listing 1 (Appendix B). Each generated response contains both a feature-proposal span, consisting of feature names and descriptions, and a function-code span that implements the proposed transformations.

To construct proposal-level preferences, we use a separate Critic LLM to score only the feature-proposal span S, conditioned on the task description, feature metadata, and demonstration rows. The Critic does not inspect the function-code span when assigning the Stage-1 score. It evaluates S along three axes: (i) **usefulness** for the prediction task, (ii) **diversity** across the proposed features, and (iii) **non-redundancy** with respect to both raw and newly proposed features. The Critic returns a scalar score $s_{\text{crit}}(S \mid c) \in [0, 1]$, where higher is better. The exact Critic prompt is shown in Listing 2 (Appendix C). For two structured responses $z_i = (S_i, G_i)$ and $z_j = (S_j, G_j)$ sampled from the same dataset context, we construct a Stage-1 preference pair when the Critic score of S_i exceeds that of S_j by a margin δ_{feat}:

$$s_{\text{crit}}(S_i \mid c) - s_{\text{crit}}(S_j \mid c) > \delta_{\text{feat}}.$$

In this case, we set $z^+ = z_i$ and $z^- = z_j$. Although the preference label is determined only from the feature-proposal span S, the Stage-1 DPO loss is computed over the full structured response $z = (S, G)$, including both the feature proposal and its executable realization. This encourages the model to produce coherent feature ideas together with compatible implementations.

3.3 Stage-2: Execution-grounded Function-DPO

Stage-2 aligns the model at the *function-realization* level. After Stage-1 training, the aligned checkpoint is used to sample new structured responses for the same preference-training contexts, from which candidate functions are extracted and evaluated.

For each dataset context, we define a raw-feature baseline by training the downstream learner on the original features only, yielding validation performance $\mathcal{M}_{\text{base}}$. Given a structured response $z = (S, G)$, we execute the generated function-code span G to obtain the induced transformation g_G, apply it to the raw table, and train the same downstream learner on the resulting engineered features to compute validation performance $\mathcal{M}(G)$. For example, for the classification tasks used in preference-training dataset construction, the performance metric $\mathcal{M}$ can be validation AUC.

Candidates that cannot be parsed, cannot be executed, or reference columns outside the declared schema are assigned to the rejected pool. For executable and schema-consistent functions, we use validation performance to construct preference pairs within the same dataset context. In particular, a function that matches or improves the raw-feature baseline is preferred over a non-executable, schema-invalid, or lower-performing function; when two executable code spans are compared, the one with higher validation performance is preferred:

$$\mathcal{M}(G_i) > \mathcal{M}(G_j).$$

The corresponding full responses are then assigned as $z^+ = z_i$ and $z^- = z_j$. We then apply DPO to these function-level pairs, masking the loss to the `<function_start>`–`<function_end>` span. This stage teaches the model *how* to realize feature designs as executable code that is both structurally valid and useful for downstream prediction.

3.4 Training Procedure

The two stages use the same DPO objective but different preference signals and loss masks. Stage-1 uses the initial generator as the fixed reference policy and optimizes full structured responses using proposal-level preferences derived from the `<features>` span. After Stage-1 training, the aligned checkpoint is used to sample new structured responses for the same preference-training contexts, from which candidate functions are extracted and evaluated. Stage-2 then uses this Stage-1 checkpoint as the fixed reference policy and optimizes execution-grounded preferences with loss applied to the function-code span. Implementation details, including backbone model, reasoning-mode configuration, loss masking, sampled-candidate counts, preference-pair counts, optimizer settings, learning rates, batch sizes, and decoding parameters, are summarized in Section 4.1 and reported in full in Appendix A.3. Algorithm 1 in Appendix A.4 summarizes the two-stage training procedure.

3.5 Transfer to Unseen Datasets

After both DPO stages, we freeze the resulting MSDPO model and apply it to previously unseen datasets without additional fine-tuning. For each unseen dataset, we construct a dataset-specific prompt containing the task description, feature metadata, and four demonstration rows, following the generation template in Listing 1 (Appendix B). Only the prompt content changes across datasets; the model parameters remain fixed. The generated `modify_features` function is executed to produce the engineered table, and the downstream learner is trained on the resulting features. We evaluate MSDPO on both classification and regression datasets.

4 Experiments

4.1 Experiment Setup

Evaluation datasets. We evaluate on 24 public tabular datasets, including 20 classification datasets and 4 regression datasets. The datasets are drawn from the UCI Machine Learning Repository, Kaggle competitions, OpenML, TabZilla, and public research datasets. They cover binary classification, multi-class classification, and continuous regression tasks, with dataset sizes ranging from hundreds to hundreds of thousands of rows and raw feature dimensions ranging from 4 to 102. Appendix A.2 lists every dataset with row counts, feature counts, class cardinalities, and the natural-language task descriptions passed to the LLM.

Preference-training setup. We construct preference data from eight public binary classification datasets reserved for DPO training (Ko et al., 2025). Detailed dataset statistics and task descriptions are provided in Appendix A.1. These datasets are used only to construct preference pairs and are not part of the evaluation benchmark. For Stage-1, we sample 50 feature proposals per training context and score each proposal using a Qwen3-32B Critic LLM, as described in Section 3.2. We construct Proposal-DPO pairs using threshold-based score separation: a proposal is preferred over another proposal from the same context if its Critic score is higher by more than $\delta_{\text{feat}} = 0.2$. This produces 1,378 Proposal-DPO preference pairs. For Stage-2, we start from the Stage-1 aligned checkpoint and sample 50 new structured responses per training context. We extract the generated `modify_features(df)` function from each response and evaluate it using validation AUC against a raw-feature XGBoost baseline. This process produces 2,757 Function-DPO preference pairs.

Training details. Both DPO stages use full fine-tuning of Qwen3-8B (Yang et al., 2025). Stage-1 uses the base Qwen3-8B model as the reference policy and trains for 2 epochs on the 1,378 Proposal-DPO pairs. Stage-2 starts from the Stage-1 aligned checkpoint and uses that checkpoint as the reference policy, training for 2 epochs on the 2,757 Function-DPO pairs. In both stages, we set $\beta = 0.5$. We enable Qwen3-8B's reasoning mode during candidate generation and inference. Generated `<think>` blocks are retained during parsing but excluded from DPO loss computation. The loss is applied only to the supervised response spans: Stage-1 computes loss on both the feature-proposal span and the function-code span; Stage-2 computes loss only on the function-code span. A compact summary of the training hyperparameters and evaluation-time generation protocol is provided in Appendix A.3.

Evaluation protocol. We use a fixed 80/20 train/test split over five seeds (42–46). For each dataset, each LLM-based method generates 20 candidate feature-construction outputs, each corresponding to a set of engineered features. Each output is evaluated by cross-validation on the training split, and the best output is then evaluated once on the held-out test split. For classification tasks, candidate selection uses validation AUC; for regression tasks, candidate selection uses validation R^2. All Qwen3-based LLM methods are evaluated with the thinking mode enabled.

Metrics. For classification, we report ROC-AUC, and macro one-vs-rest ROC-AUC is used for multi-class datasets. For regression, we report R^2, where higher values indicate better performance. For cross-dataset comparison, we report mean rank: for each dataset, methods with valid results are ranked from 1 to K, with 1 indicating the best performance. We use the min-rank convention for ties: all methods tied at the best value receive rank 1, and ties at lower values receive the smallest rank occupied by the tied group. N/A entries are excluded from the method's mean-rank denominator.

Baselines. We compare MSDPO with a no-feature-engineering XGBoost reference and seven automated feature-engineering baselines, covering classical search-based methods and LLM-

Table 1: Classification results over five seeds (42–46) on the 20 held-out datasets. Cells give mean test ROC-AUC with the across-seed standard deviation as a subscript; higher is better. Best per row is bolded, best non-tied runner-up underlined. N/A marks a method that produced no valid executable output on every seed. Ties use min-rank; Rank-1 includes ties; mean rank is lower-better over each method's valid datasets. Mean AUC is averaged over each method's valid datasets.

Dataset	XGBoost	DFS	AutoFeat	OpenFE	CAAFE	OCTree	FeatLLM	LLM-FE	MSDPO
ad-click	$0.7590_{.007}$	$0.7617_{.006}$	$0.7590_{.007}$	$\mathbf{0.8088}_{.006}$	$0.7607_{.008}$	$0.7500_{.004}$	N/A	$0.7137_{.066}$	$\underline{0.7793}_{.006}$
adult	$\underline{0.9292}_{.002}$	$0.9266_{.003}$	$\underline{0.9292}_{.002}$	$0.9265_{.002}$	$0.9287_{.002}$	$0.9278_{.002}$	$0.8938_{.007}$	$0.9286_{.003}$	$\mathbf{0.9294}_{.003}$
balance-scale	$0.9299_{.023}$	$0.9951_{.006}$	$0.9299_{.023}$	$\underline{0.9966}_{.005}$	$\mathbf{1.0000}_{.000}$	$0.9254_{.027}$	$\mathbf{1.0000}_{.000}$	$\mathbf{1.0000}_{.000}$	$\mathbf{1.0000}_{.000}$
bank	$0.9275_{.001}$	$0.9261_{.001}$	$0.9275_{.001}$	$\underline{0.9299}_{.001}$	$0.9275_{.001}$	$0.9274_{.001}$	N/A	$0.9284_{.001}$	$\mathbf{0.9311}_{.002}$
banknote	$\underline{0.9999}_{.000}$	$\underline{0.9999}_{.000}$	$\underline{0.9999}_{.000}$	$\mathbf{1.0000}_{.000}$	$\underline{0.9999}_{.000}$	$\underline{0.9999}_{.000}$	$0.9996_{.001}$	$\underline{0.9999}_{.000}$	$\underline{0.9999}_{.000}$
breast-w	$0.9919_{.004}$	$0.9959_{.002}$	$0.9919_{.004}$	$\mathbf{0.9964}_{.003}$	$0.9941_{.005}$	$0.9928_{.002}$	N/A	$0.9940_{.005}$	$\underline{0.9962}_{.001}$
car	$\underline{0.9998}_{.000}$	$\mathbf{1.0000}_{.000}$	$\underline{0.9998}_{.000}$	$\mathbf{1.0000}_{.000}$	$\underline{0.9998}_{.000}$	$0.9996_{.000}$	$0.9661_{.037}$	$0.9539_{.063}$	$\mathbf{1.0000}_{.000}$
cdc	$\mathbf{0.8219}_{.002}$	$0.8204_{.003}$	$\mathbf{0.8219}_{.002}$	$0.8179_{.002}$	$\underline{0.8218}_{.002}$	$\underline{0.8218}_{.002}$	$0.8199_{.004}$	$\underline{0.8218}_{.002}$	$0.8216_{.003}$
cmc	$0.6933_{.016}$	$0.6892_{.020}$	$0.6933_{.016}$	$0.6908_{.011}$	$0.6963_{.014}$	N/A	$0.6954_{.015}$	$\underline{0.6968}_{.009}$	$\mathbf{0.6971}_{.013}$
communities	$\underline{0.8740}_{.004}$	$0.8682_{.003}$	$\underline{0.8740}_{.004}$	$0.8725_{.009}$	$0.8729_{.004}$	$0.8709_{.003}$	$0.8100_{.013}$	$0.8725_{.002}$	$\mathbf{0.8755}_{.003}$
credit-g	$0.7762_{.029}$	$0.7760_{.032}$	$0.7736_{.027}$	$\underline{0.7832}_{.023}$	$0.7762_{.029}$	$0.7826_{.025}$	N/A	$\mathbf{0.7843}_{.020}$	$0.7796_{.024}$
cultivar	$0.7367_{.041}$	$0.7666_{.035}$	$0.7367_{.041}$	$0.7438_{.031}$	$0.7137_{.048}$	$0.6879_{.047}$	$0.6341_{.022}$	$\underline{0.8104}_{.066}$	$\mathbf{0.8252}_{.058}$
customer	$0.8263_{.010}$	$0.8241_{.010}$	$0.8263_{.010}$	$0.8233_{.012}$	$0.8249_{.011}$	$0.8212_{.011}$	N/A	$\mathbf{0.8274}_{.012}$	$\underline{0.8271}_{.013}$
heart	$0.9222_{.011}$	$0.9154_{.015}$	$0.9213_{.010}$	$\underline{0.9247}_{.010}$	$0.9233_{.011}$	$0.9192_{.013}$	$0.8787_{.028}$	$0.9170_{.014}$	$\mathbf{0.9253}_{.013}$
junglechess	$0.9770_{.001}$	$0.9875_{.001}$	$0.9770_{.001}$	$\underline{0.9879}_{.000}$	$0.9829_{.001}$	$0.9761_{.001}$	$0.9811_{.001}$	$0.9822_{.008}$	$\mathbf{0.9988}_{.001}$
myocardial	$0.6938_{.028}$	$\underline{0.7009}_{.028}$	N/A	$0.6915_{.023}$	$0.6938_{.025}$	$\mathbf{0.7144}_{.035}$	N/A	$0.6909_{.025}$	$0.6918_{.033}$
pc1	$0.8619_{.034}$	$\underline{0.8727}_{.013}$	$0.8619_{.034}$	$\mathbf{0.8927}_{.042}$	$0.8670_{.026}$	$0.8470_{.033}$	$0.8641_{.033}$	$0.8631_{.030}$	$0.8464_{.029}$
titanic	$0.8390_{.019}$	$0.8389_{.015}$	$0.8390_{.019}$	$\mathbf{0.8555}_{.020}$	$0.8180_{.030}$	$0.8399_{.019}$	N/A	$0.8445_{.021}$	$\underline{0.8551}_{.020}$
vehicle	$0.9314_{.006}$	$\mathbf{0.9468}_{.004}$	$0.9314_{.006}$	$\underline{0.9391}_{.006}$	$0.9367_{.004}$	$0.9266_{.004}$	$0.9084_{.017}$	$0.9338_{.007}$	$0.9350_{.007}$
water	$0.6370_{.023}$	$0.6494_{.015}$	$0.6414_{.026}$	$0.6540_{.014}$	$\mathbf{0.6624}_{.015}$	$0.6284_{.011}$	N/A	$0.6593_{.028}$	$\underline{0.6621}_{.020}$
Valid datasets	20/20	20/20	19/20	20/20	20/20	19/20	12/20	20/20	20/20
Mean AUC	0.8564	0.8631	0.8650	0.8668	0.8600	0.8610	0.8709	0.8611	0.8688
Mean rank	4.50	4.85	4.74	3.55	3.75	6.16	7.08	4.10	**2.45**
Rank-1	1/20	2/20	1/20	6/20	2/20	1/20	1/20	3/20	**9/20**

based feature-generation methods. Classical baselines include DFS, AutoFeat, and OpenFE (Kanter and Veeramachaneni, 2015; Horn et al., 2019; Zhang et al., 2023). LLM-based baselines include CAAFE, OCTree, LLM-FE, and FeatLLM (Hollmann et al., 2023; Nam et al., 2024; Abhyankar et al., 2025; Han et al., 2024). We also use Claude Sonnet 4.6 as a frontier baseline, with results reported in Appendix D. All methods use the same 80/20 split; for methods that generate multiple candidate feature sets or functions, we use the same validation-based candidate-selection protocol. Additional details are provided in Appendix A.5.

Tables 1 and 2 report the five seeds (42–46) main results under the 80/20 train/test split and thinking-mode evaluation convention. Classification results are reported using AUC, where higher is better. Regression results are reported using R^2, where a higher value is better. Each LLM-based method generates 20 candidate feature-construction outputs, each corresponding to a set of engineered features, and selects the best output using validation performance. N/A indicates that the method did not produce a valid executable output under the shared parsing and execution protocol. CAAFE, OCTree, and FeatLLM are omitted from regression because their released evaluation pipelines are classification-oriented.

4.2 Experiment results

Classification performance. Table 1 reports the classification results. MSDPO obtains the best overall mean rank, 2.45, outperforming the second-best methods, OpenFE with a mean rank of 3.55. Under the min-rank convention, MSDPO obtains rank-1 performance on 9/20 datasets, including 7 sole-best wins. The gains are especially clear on datasets such as `adult`, `bank`, `cmc`, `communities`, `cultivar`, `heart`, and `junglechess`, where useful transformations often require combining semantic column meaning with executable numerical or categorical transformations.

Classical search methods such as DFS, AutoFeat, and OpenFE can be strong when useful transformations lie within predefined operator spaces. For example, OpenFE performs best on `ad-click`, `breast-w`, `pc1` and `titanic`, suggesting that its downstream-model-guided candidate search can identify effective operator-based transformations. However, these methods do not directly use task descriptions or feature semantics, which limits their ability to form domain-sensitive recodings,

Table 2: Regression results over five seeds (42–46) on four held-out regression datasets. Cells give mean test R^2 with the across-seed standard deviation as a subscript; higher is better. Best per row is bolded, best non-tied runner-up underlined. Ranks are computed from the four-decimal values shown using the min-rank convention; Rank-1 includes ties and mean rank is lower-better.

Dataset	XGBoost	DFS	AutoFeat	OpenFE	LLM-FE	MSDPO
diamonds	$0.9810_{.001}$	$0.9809_{.001}$	$0.9810_{.001}$	$\underline{0.9818_{.001}}$	$0.9811_{.001}$	$\mathbf{0.9820}_{.001}$
housing	$0.8276_{.012}$	$0.8323_{.021}$	$0.8295_{.010}$	$\underline{0.8437_{.007}}$	$0.8404_{.011}$	$\mathbf{0.8488}_{.005}$
insurance	$0.8093_{.041}$	$\mathbf{0.8193}_{.041}$	$0.8093_{.041}$	$0.8114_{.046}$	$0.8129_{.035}$	$\underline{0.8145_{.034}}$
student_score	$\mathbf{0.6521}_{.051}$	$\underline{0.6499_{.052}}$	$\mathbf{0.6521}_{.051}$	$0.6452_{.051}$	$0.6472_{.056}$	$0.6489_{.056}$
Mean R^2	0.8175	0.8206	0.8180	0.8205	0.8204	**0.8236**
Mean rank	4.00	3.50	3.75	3.50	3.50	**2.00**
Rank-1	1/4	1/4	1/4	0/4	0/4	**2/4**

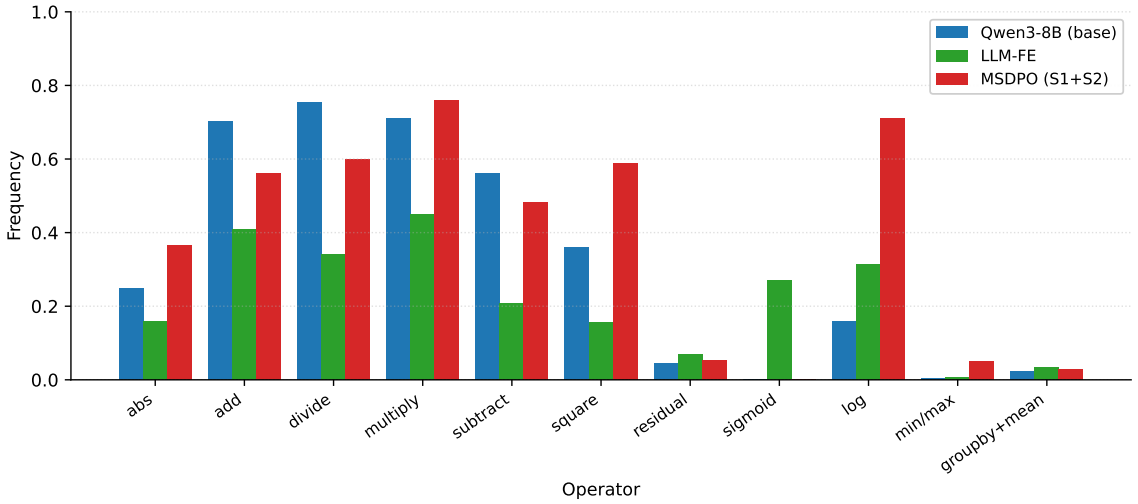


Figure 2: Operator usage, measured as the fraction of generated feature-construction functions containing each operator, for the unaligned Qwen3-8B generator, LLM-FE, and MSDPO. Compared with LLM-FE, MSDPO shows higher usage of several executable numerical transformations.

grouped categorical indicators, or task-specific interactions. LLM-based baselines provide complementary strengths but remain constrained by inference-time scaffolds: CAAFE relies on iterative feature generation and validation feedback; OCTree uses decision-tree feedback that may miss broader semantic compositions; LLM-FE achieves the best performance on `credit-g` and `customer`, but its performance across datasets is uneven: it ranks last on `car` and near-last on `ad-click` and `myocardial`, giving a mean rank of 4.10. This is consistent with the operator-usage analysis in Figure 2, where LLM-FE draws on a narrower set of transformations than MSDPO when using 8B model as a backbone; and FeatLLM is less aligned with our setting because it targets class-indicative rules rather than general-purpose `modify_features` functions for a fixed XGBoost pipeline. In contrast, MSDPO internalizes two complementary preferences: useful feature design and valid, performance-improving implementation, yielding the strongest aggregate rank. We also compare MSDPO with a frontier LLM, Claude Sonnet 4.6, in Table 8. In terms of sole-best outcomes, MSDPO wins 8 datasets; Claude Sonnet 4.6 wins 3 datasets. Although Sonnet remains stronger on several datasets, these results show that under the controlled, budget-matched feature-engineering harness, MSDPO achieves a better mean rank than Claude Sonnet 4.6.

Regression transfer. Table 2 reports the performance on regression. MSDPO achieves the best overall mean rank of 2.00, outperforming DFS, OpenFE, and LLM-FE at 3.50, despite Function-DPO being trained with preferences constructed from classification AUC. This cross-task transfer is notable because the preference signal is constructed from a different prediction objective and evaluation metric. The result suggests that Function-DPO

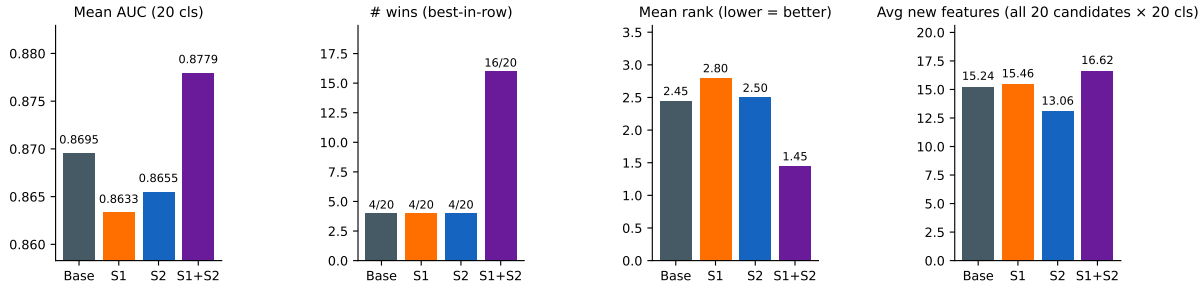


Figure 3: Ablation summary on the 20 classification datasets under the seed-42 80/20 split. The figure reports mean AUC, number of best-in-row wins, mean rank, and average number of generated features. Mean rank is lower-better.

learns more than AUC-specific heuristics: it captures transferable principles of useful feature engineering, including executability, schema consistency, meaningful numerical transformations, ratios, and interactions.

Operator usage analysis. Recent work reports operator biases in LLM-based feature engineering (Küken et al., 2024). To better understand why MSDPO improves aggregate performance, we analyze which transformation operators are used in the generated feature-construction functions. Figure 2 compares the operator-usage frequency of the unaligned Qwen3-8B generator, LLM-FE, and the full S1+S2 MSDPO model. Compared with LLM-FE, MSDPO shows higher usage of several executable numerical transformations, with especially high usage of multiplicative interactions, logarithmic transformations, squared terms, ratios, and absolute-value transformations. These patterns are consistent with the performance results: the two-stage preference alignment does not merely increase the number of generated features, but shifts the generator toward a broader set of commonly useful tabular transformations. While the base model already frequently proposes simple arithmetic operations such as addition/division, MSDPO places more emphasis on nonlinear and interaction-style transformations that are often useful for tabular prediction.

4.3 Ablation: Contribution of Each Stage

We evaluate four variants: Base, the unaligned Qwen3-8B generator; S1-only, using only Critic-based Proposal-DPO; S2-only, using only Function-DPO from the base generator; and the full MSDPO pipeline. Figure 3 summarizes the aggregate ablation results, while the full per-dataset ablation table is provided in Appendix E. The full MSDPO pipeline achieves the strongest aggregate performance across the 20 classification datasets: it obtains the highest mean AUC, 0.8779, the largest number of best-in-row wins, 16/20, and the lowest mean rank, 1.45. Base, S1-only, and S2-only obtain lower mean AUCs of 0.8695, 0.8633, and 0.8655, respectively, and each wins only 4/20 datasets. The feature-count analysis further shows that Stage 2 alone produces the most compact outputs, with 13.06 new features on average, while the MSDPO model produces 16.62 new features on average. This suggests that the two-stage alignment improves both the selection and executable implementation of useful transformations, rather than merely increasing the number of generated features.

Downstream learner	Raw	Base FE	MSDPO FE
XGBoost	2.37	1.74	**1.68**
AutoGluon	1.89	1.95	**1.84**

Table 3: Mean rank (lower is better) of three feature sources under two downstream learners on the common set of 19 held-out classification datasets at seed 42. Per-dataset values for both learners are given in Appendix F.

4.4 Robustness to the Downstream Learner

A feature-engineering result obtained with a single downstream model may reflect the limitations of that model rather than the quality of the generated features: if the learner cannot exploit the raw table well, almost any additional feature will appear to help. We therefore repeat the comparison between raw features, features from the unaligned Qwen3-8B backbone, and features from MSDPO, replacing XGBoost with AutoGluon, an automated system that trains and stacks an ensemble of heterogeneous tabular models.

Table 3 shows that the ranking of the two generators is preserved across learners—MSDPO ahead of the unaligned Base model under both—but the advantage of feature engineering over raw features

largely disappears. The informative change is in the raw-feature column, which improves from a mean rank of 2.37 under XGBoost to 1.89 under AutoGluon, while MSDPO stays nearly constant at 1.68 and 1.84. The gap closes because the stronger learner extracts more from the original table, not because the generated features deteriorate. Per-dataset values for both learners are reported in Appendix F.

We interpret this as a ceiling effect. Ensembling and stacking many model families is itself a form of implicit feature construction, and it overlaps with what explicit feature engineering supplies, leaving less headroom for generated features to contribute. The comparison that isolates the effect of preference alignment is MSDPO against the unaligned Base model on the same backbone and prompt, and that ordering survives the change of learner.

We therefore do not claim that the benefit of preference alignment is independent of the downstream model. What these results support is the narrower claim that aligning the generator improves the features it produces, and that this ordering holds under a substantially stronger learner, while the absolute gain over raw features depends on the downstream pipeline in use.

5 Related Work

Automated feature engineering reduces manual design by searching over candidate transformations. Classical methods such as DFS, AutoFeat, and OpenFE construct relational, symbolic, or model-guided features (Kanter and Veeramachaneni, 2015; Khurana et al., 2016; Katz et al., 2016; Horn et al., 2019; Zhang et al., 2023). Recent LLM-based methods exploit task and schema semantics to generate transformations, rules, or operator programs (Hollmann et al., 2023; Abhyankar et al., 2025; Nam et al., 2024; Han et al., 2024). Unlike these inference-time prompting, search, or feedback-based approaches, MSDPO internalizes preferences for useful and executable transformations through Proposal-DPO and Function-DPO. Preference alignment methods such as RLHF and DPO optimize language models from chosen–rejected pairs (Ouyang et al., 2022; Schulman et al., 2017; Rafailov et al., 2023); our work constructs such preferences from feature quality, execution validity, and downstream validation performance. A more detailed discussion is provided in Appendix A.6.

6 Conclusion

We introduced MSDPO, a multi-stage preference-optimization framework for LLM-based automated feature engineering. MSDPO aligns the generator at two levels: Proposal-DPO teaches the model to prefer diverse, useful, and non-redundant feature proposals, while Function-DPO teaches it to produce executable feature-construction code using execution and validation feedback. MSDPO achieves the best mean rank on both the 20-dataset classification benchmark and the four-dataset regression benchmark. The results suggest that preference optimization can move LLM-based feature engineering beyond inference-time prompting and filtering toward a single aligned generator that produces useful, interpretable, and executable transformations for unseen datasets.

Limitations

This study has several limitations. First, MSDPO is trained on preference data from eight binary classification datasets, so the preference signal encodes what makes a feature useful for binary classification performance as measured by validation ROC-AUC; whether the recipe transfers to objectives such as ranking, calibration, or fairness is untested.

Second, our evaluation uses a fixed downstream XGBoost pipeline, and Section 4.4 shows that the size of the end-to-end gain depends on that choice: under AutoGluon the advantage of generated features over the raw table largely disappears, although the ordering of MSDPO above the unaligned backbone is preserved under both learners.

Third, Stage-1 preferences are derived from scores assigned by a separate critic LLM, so the learned notion of a useful proposal inherits that critic's biases, and repeated scorings of the same proposal do not always agree. Discarding pairs whose score gap falls below the margin δ_{feat} and aggregating over many pairs mitigates this, but we do not characterise how the aligned generator would change under a different critic model, prompt, or scoring scale.

Fourth, both preference construction and evaluation sample at non-zero temperature, so an individual run is not reproducible in the strict sense: the candidate pool differs between runs, and so does the candidate that cross-validation selects. We therefore report means and across-seed standard deviations over five independently resampled seeds rather than a single generation.

Our benchmark does not cover extreme class imbalance, multi-label prediction, or time-series feature engineering. In addition, the Claude Sonnet 4.6 comparison is a controlled, budget-matched model-only evaluation and should not be interpreted as an upper bound on richer agentic workflows.

References

Nikhil Abhyankar, Parshin Shojaee, and Chandan K. Reddy. 2025. LLM-FE: Automated feature engineering for tabular data with LLMs as evolutionary optimizers. *arXiv preprint arXiv:2503.14434*.

Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. 2022. Deep neural networks and tabular data: A survey. *IEEE transactions on neural networks and learning systems*, 35(6):7499–7519.

Fengxian Dong, Zhi Zheng, Xiao Han, Wei Chen, Jingqing Ruan, Tong Xu, Yong Chen, and Enhong Chen. 2026. Memory-augmented LLM-based multi-agent system for automated feature generation on tabular data. *arXiv preprint arXiv:2604.20261*.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. 2024. KTO: Model alignment as prospect theoretic optimization. *arXiv preprint arXiv:2402.01306*.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. 2021. Revisiting deep learning models for tabular data. *Advances in neural information processing systems*, 34:18932–18943.

Sungwon Han, Jinsung Yoon, Sercan O Arik, and Tomas Pfister. 2024. Large language models can automatically engineer features for few-shot tabular learning. *arXiv preprint arXiv:2404.09491*.

Stefan Hegselmann, Alejandro Buendia, Hunter Lang, Monica Agrawal, Xiaoyi Jiang, and David Sontag. 2023. TabLLM: Few-shot classification of tabular data with large language models. In *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, pages 5549–5581. PMLR.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. 2022. Tabpfn: A transformer that solves small tabular classification problems in a second. *arXiv preprint arXiv:2207.01848*.

Noah Hollmann, Samuel Müller, and Frank Hutter. 2023. Large language models for automated data science: Introducing caafe for context-aware automated feature engineering. *Advances in Neural Information Processing Systems*, 36:44753–44775.

Jiwoo Hong, Noah Lee, and James Thorne. 2024. ORPO: Monolithic preference optimization without reference model. *arXiv preprint arXiv:2403.07691*.

Franziska Horn, Robert Pack, and Michael Rieger. 2019. The autofeat python library for automated feature engineering and selection. *arXiv preprint arXiv:1901.07329*.

Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. 2020. Tabtransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*.

James Max Kanter and Kalyan Veeramachaneni. 2015. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE.

Gilad Katz, Eui Chul Richard Shin, and Dawn Song. 2016. ExploreKit: Automatic feature generation and selection. In *Proceedings of the 2016 IEEE International Conference on Data Mining*. IEEE.

Udayan Khurana, Deepak Turaga, Horst Samulowitz, and Srinivasan Parthasarathy. 2016. Cognito: Automated feature engineering for supervised learning. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 1304–1307. IEEE.

Jeonghyun Ko, Gyeongyun Park, Donghoon Lee, and Kyunam Lee. 2025. FeRG-LLM: Feature engineering by reason generation large language models. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 4211–4228.

Jaris Küken, Lennart Purucker, and Frank Hutter. 2024. Large language models engineer too many simple features for tabular data. *arXiv preprint arXiv:2410.17787*.

Jaehyun Nam, Kyuyoung Kim, Seunghyuk Oh, Jihoon Tack, Jaehyung Kim, and Jinwoo Shin. 2024. Optimized feature generation for tabular data via llms with decision tree reasoning. *Advances in neural information processing systems*, 37:92352–92380.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Yu Pan, Zhongze Cai, Huaiyang Zhong, Guanting Chen, and Chonghuan Wang. 2026. What matters in data for dpo? *Advances in Neural Information Processing Systems*, 38:44689–44716.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Ravid Shwartz-Ziv and Amitai Armon. 2022. Tabular data: Deep learning is not all you need. *Information fusion*, 81:84–90.

Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C. Bayan Bruss, and Tom Goldstein. 2021. SAINT: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv preprint arXiv:2106.01342*.

Shusheng Xu, Wei Fu, Jiaxuan Gao, Wenjie Ye, Weilin Liu, Zhiyu Mei, Guangju Wang, Chao Yu, and Yi Wu. 2024. Is dpo superior to ppo for llm alignment? a comprehensive study. *arXiv preprint arXiv:2404.10719*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Tianping Zhang, Zheyu Zhang, Zhiyuan Fan, Haoyan Luo, Fengyuan Liu, Qian Liu, Wei Cao, and Jian Li. 2023. OpenFE: Automated feature generation with expert-level performance. In *Proceedings of the 40th International Conference on Machine Learning*.

A Dataset, Training, and Inference Details

We use public datasets, pretrained models, baseline implementations, and standard ML libraries according to their publicly available licenses or terms of use; licenses and terms may vary by source. All artifacts are used only for research evaluation in this work, and any derived artifacts, including preference pairs, generated feature-construction outputs, or aligned model checkpoints, are intended for research use rather than deployment in high-stakes settings without additional validation.

A.1 DPO Training Datasets

Table 4 lists the eight binary classification datasets used as preference-training contexts for both DPO stages. Stage 1 constructs proposal-level preference pairs from Critic scores, while Stage 2 resamples structured responses from the Stage 1 checkpoint and constructs execution-grounded preference pairs on the same dataset contexts. These datasets are disjoint from the evaluation benchmark.

A.2 Evaluation Dataset Details

We evaluate on public tabular datasets spanning classification and regression tasks, drawn from the UCI Machine Learning Repository, Kaggle competitions, OpenML, TabZilla, and public research data. Table 5 lists the 24 datasets used in the main evaluation tables, including row counts, raw feature counts, task type, and the one-line natural-language task description passed to the LLM.

A.3 Training and Inference Configuration

This section provides the training and evaluation configurations used for MSDPO. Table 6 summarizes the two DPO stages, including initialization, reference policies, preference-pair counts, optimization settings, and hardware. Table 7 summarizes the evaluation-time generation and candidate-selection protocol. These details are intended to support reproducibility while keeping the main experimental section concise. Preference-data candi-

Table 4: DPO preference-training datasets used in both stages. All datasets are binary classification tasks and are disjoint from the downstream evaluation datasets.

Dataset	#Rows	#Feat	#Cls	Task
bankruptcy	25,595	64	2	Does the company go bankrupt?
garment	1,197	14	2	Does the team have high productivity?
iranian-churn	3,150	13	2	Does the customer churn at the end of 12 months?
taiwan-credit	30,000	23	2	Will the client default on next month's payment?
transfusion	748	4	2	Does the person donate blood in March 2007?
vehicle-coupon	12,684	24	2	Does the person accept the coupon?
purchasing-intention	12,330	17	2	Does the session end with a purchase?
news-popularity	18,834	60	2	Does the article have $\geq$1,400 shares?

Table 5: List of the 24 public datasets used in the main evaluation tables. "#Rows" is the total row count in the CSV; "#Feat" is the number of raw input columns with the target column excluded; "#Cls" is the number of classes for classification or `reg` for regression. The task text is the one-line natural-language task description passed to the LLM as {TASK}.

Dataset	#Rows	#Feat	#Cls	Task text given to LLM
`ad-click`	10,000	6	2	Predict whether a user clicks an advertisement.
`adult`	48,842	14	2	Does this person earn more than $50,000 per year?
`balance-scale`	625	4	3	Predict whether the balance scale tilts left, right, or is balanced.
`bank`	45,211	16	2	Predict if the client will subscribe to a term deposit.
`banknote`	1,372	4	2	Is this banknote genuine or forged?
`breast-w`	699	9	2	Classify breast tissue as benign or malignant.
`car`	1,728	6	4	Evaluate car acceptability across four levels.
`cdc`	253,680	21	2	Predict whether a person has diabetes or prediabetes.
`cmc`	1,473	9	3	Predict contraceptive method used across three classes.
`communities`	1,994	102	3	Predict violent crime rate category per community.
`credit-g`	1,000	20	2	Classify customer credit risk as good or bad.
`cultivar`	320	10	2	Predict soybean cultivar identity.
`customer`	7,043	19	2	Predict telecom customer churn.
`heart`	918	11	2	Does the coronary angiography of this patient show a heart disease? Yes or no?
`junglechess`	44,819	6	3	Predict the outcome of a jungle chess position.
`myocardial`	686	91	2	Predict complications of myocardial infarction.
`pc1`	1,109	21	2	Classify software modules as defect-prone or not.
`titanic`	891	11	2	Predict survival of Titanic passengers.
`vehicle`	846	18	4	Classify vehicle type from silhouette features.
`water`	3,276	9	2	Predict water potability.
`diamonds`	53,940	9	reg	Predict diamond price from physical attributes.
`housing`	20,640	9	reg	Predict median house value in California districts.
`insurance`	1,338	6	reg	Predict individual medical insurance cost.
`student_score`	6,607	19	reg	Predict student exam performance.

dates in both DPO stages are sampled at temperature 1.0 to encourage diversity, whereas evaluation-time generation uses temperature 0.6 as reported in Table 7.

For Claude Sonnet 4.6, we use the same dataset prompt and candidate-selection protocol.

A.4 Training Algorithm

Algorithm 1 summarizes the sequential construction of proposal-level and execution-grounded preference pairs and the corresponding DPO updates in the two training stages.

A.5 Baseline Details

We compare against a no-feature-engineering XGBoost reference and seven automated feature-engineering baselines. All baselines are evaluated under the same fixed 80/20 train/test split over five seeds (42–46). When a method generates multiple candidate feature sets or functions, we select the candidate using validation performance on the training split and report the final performance on the held-out test split.

Classical automated feature engineering. DFS (Kanter and Veeramachaneni, 2015) applies a fixed library of transformation and aggregation primitives and selects generated features using

Table 6: Training configuration for MSDPO.

Item	Stage-1 Proposal-DPO	Stage-2 Function-DPO
Backbone / initialization	Qwen3-8B	Stage-1 checkpoint
Critic model	Qwen3-32B	–
Reference policy	Frozen Qwen3-8B	Frozen Stage-1 checkpoint
Preference pairs	1,378	2,757
DPO β	0.5	0.5
Per-device batch size	2	2
Gradient accumulation	4	4
Effective batch size	8 preference pairs / step	8 preference pairs / step
Learning rate	1×10^{-6}	1×10^{-6}
Optimizer	Adam, $\epsilon = 10^{-4}$	Adam, $\epsilon = 10^{-4}$
Weight decay	0.0	0.0
Hardware	4×A100 80GB	4×A100 80GB

Table 7: Evaluation-time generation and model-selection protocol.

Item	Setting
Candidates per dataset	20
Sampling temperature	0.6
Top-p	0.95
Top-k	20
Thinking mode	Enabled for Qwen3-based methods
Demonstration rows	4
Candidate selection	5-fold CV on the 80% training split
Downstream learner	XGBoost with library-default hyperparameters

univariate statistics. AutoFeat (Horn et al., 2019) constructs nonlinear symbolic feature combinations from operators such as addition, subtraction, multiplication, division, logarithms, and square roots, followed by feature selection. OpenFE (Zhang et al., 2023) performs a coarse-to-fine candidate-feature search using a downstream gradient-boosted model as an evaluation oracle. These methods operate on tabular schemas and values, but do not use natural-language task descriptions.

LLM-based automated feature engineering. CAAFE (Hollmann et al., 2023) prompts an LLM to generate Python feature transformations and uses validation feedback to guide feature construction. OCTree (Nam et al., 2024) uses decision-tree reasoning to guide iterative feature-rule improvement. LLM-FE (Abhyankar et al., 2025) formulates feature generation as program search over an operator-based transformation space with execution feedback. FeatLLM (Han et al., 2024) prompts the LLM to generate class-indicative rules or features paired with a simple downstream classifier. For fair comparison, LLM-based baselines use the same Qwen3-8B backbone when applicable. MSDPO, CAAFE, LLM-FE, and FeatLLM use four in-context demonstration rows sampled from the corresponding dataset.

A.6 Extended Related Work

Automated feature engineering. Automated feature engineering reduces manual feature design by searching over candidate transformations. Classical methods such as Deep Feature Synthesis, AutoFeat, and OpenFE construct relational, symbolic, or model-guided feature candidates (Kanter and Veeramachaneni, 2015; Khurana et al., 2016; Katz et al., 2016; Horn et al., 2019; Zhang et al., 2023). These methods are effective when useful transformations fall within their operator libraries, but they primarily rely on column types and values rather than natural-language task context or domain semantics. MSDPO instead uses task descriptions, feature metadata, and example rows to generate executable feature-construction functions for unseen datasets.

LLM-based feature engineering. Recent work uses language models to exploit semantic information in tabular tasks, including few-shot tabular prediction and automated feature engineering. TabLLM serializes tabular rows into natural-language strings for few-shot classification (Hegselmann et al., 2023). CAAFE iteratively generates semantically meaningful Python features from

Algorithm 1 Multi-Stage DPO for Feature Engineering (MSDPO)

Require: Initial policy π_0; training contexts $\mathcal{P}$; Critic ϕ; downstream evaluator $\mathcal{E}$.

Stage 1: Proposal-DPO

for $c \in \mathcal{P}$ **do**

  Sample structured responses $z_i = (S_i, G_i) \sim \pi_0(\cdot \mid c)$.

  Score each proposal span S_i with the Critic ϕ.

  Construct preference pairs (z^+, z^-) using $s_{\text{crit}}(S^+ \mid c) - s_{\text{crit}}(S^- \mid c) > \delta_{\text{feat}}$.

end for

Train π_1 from π_0 using DPO on Stage-1 pairs.

Stage 2: Function-DPO

for $c \in \mathcal{P}$ **do**

  Sample structured responses $z_i = (S_i, G_i) \sim \pi_1(\cdot \mid c)$.

  Execute each G_i and evaluate validation performance $\mathcal{M}(G_i)$.

  Construct preference pairs (z^+, z^-) from execution validity and validation performance.

end for

Train π_2 from reference policy π_1 using DPO with loss masked to the function-code span.

return π_2

dataset descriptions (Hollmann et al., 2023); LLM-FE formulates feature engineering as iterative program search guided by data-driven feedback (Abhyankar et al., 2025); OCTree uses decision-tree reasoning to provide feedback for feature-rule improvement (Nam et al., 2024); and FeatLLM generates class-indicative rules for few-shot tabular learning (Han et al., 2024). These methods show that LLMs can use semantic cues for feature construction, but they largely rely on inference-time prompting, search, feedback, or filtering. MSDPO differs by internalizing these preferences into the generator through two DPO stages, targeting both proposal-level feature quality and executable implementation quality.

Preference alignment. Preference-based alignment adapts language models toward desired behavior from chosen–rejected pairs. RLHF optimizes against learned reward models, often with policy-gradient methods such as PPO (Ouyang et al., 2022; Schulman et al., 2017), while DPO directly optimizes from preference pairs without an explicit reward model (Rafailov et al., 2023; Ethayarajh et al., 2024; Hong et al., 2024). MSDPO applies preference alignment to automated feature engineering, constructing preferences from feature quality, execution validity, and downstream validation performance.

A.7 Critic Consistency

To assess repeated-score consistency of the Stage-1 critic, we compare Qwen3-32B, the critic used by MSDPO, with Qwen3-8B under identical prompts and decoding settings. We evaluate seven preference-training datasets: `garment`, `iranian-churn`, `taiwan-credit`, `transfusion`, `bankruptcy`, `news-popularity`, and `vehicle-coupon`.

For each dataset, both critics independently score the same 10 shared valid candidates three times. For each candidate, we compute the standard deviation across the three scores and report the median across the 10 candidates. Aggregated over the 70 candidates, Qwen3-32B yields a median score standard deviation of 0.047, compared with 0.062 for Qwen3-8B, and shows lower variability on six of the seven datasets. These results indicate that Qwen3-32B is more consistent under our evaluation protocol, while also confirming that critic scoring remains stochastic and may still be sensitive to model choice.

B Generation Prompt Template

Listing 1 shows the canonical generation prompt template fully instantiated for the `heart` dataset. The same template is used for generation across datasets, with only `{TASK}`, `{FEATURES}`, and `{EXAMPLES}` replaced by dataset-specific content. The placeholders are replaced per dataset as follows: `{TASK}` is a one-line natural-language question describing the prediction target; `{FEATURES}` lists every input column with its one-line description and type, including the category set for categorical columns; and `{EXAMPLES}` contains four demonstration rows, each serialized as a sentence of the form "*column* is *value*, . . . , the result is *label*." There is no Markdown table or CSV; the model receives plain natural-language sentences. Listing 1 shows the fully instantiated prompt for the `heart` dataset. At inference, we decode with temperature 0.6, top-p $= 0.95$, top-k $= 20$, following Qwen3 recommended sampling.

C Critic-LLM Prompt Template

Listing 2 shows the exact Critic prompt (`templates/critic_template.txt`) used for Stage-1 preference construction. The Critic LLM (Qwen3-32B at temperature 0.6, thinking mode enabled) outputs free-form reasoning followed by a single numeric score in $[0, 1]$ on the final line.

D Frontier LLM Comparison with Claude Sonnet 4.6

Table 8 compares MSDPO with two strong feature-generation prompting baselines on the 20 held-out classification datasets: the unaligned Qwen3-8B backbone and Claude Sonnet 4.6. Qwen3-8B uses the same backbone, generation prompt, four demonstration rows, and thinking-mode inference protocol as MSDPO, differing only in the absence of the two DPO alignment stages. MSDPO achieves the best mean rank, 1.65, compared with 2.20 for Claude Sonnet 4.6 and 1.85 for Qwen3-8B. In terms of sole-best outcomes, MSDPO wins 8 datasets, Qwen3-8B wins 5, and Claude Sonnet 4.6 wins 3, with the remaining datasets tied. Although Sonnet remains stronger on several datasets, these results show that task-specific preference alignment can achieve a better mean rank than Claude Sonnet 4.6 under this controlled, budget-matched feature-engineering harness.

E Per-Dataset Ablation Results

Table 9 provides the full per-dataset classification ablation under the same configuration. We compare four variants: Base, the unaligned Qwen3-8B generator; S1-only, using only Critic-based Proposal-DPO; S2-only, using only Function-DPO; and the full MSDPO pipeline. Values are AUC; higher is better. Best per row is bolded.

F Per-Dataset Downstream-Learner Results

Table 10 reports the per-dataset values behind Table 3, for both downstream learners at seed 42. Within each learner block, candidate selection uses the same learner that performs the final evaluation, so the two blocks differ only in the downstream model applied to an otherwise identical protocol. Table 3 recomputes mean ranks on the common 19-dataset subset, whereas the aggregate rows in Table 10 follow each method's valid-dataset denominator.

The `titanic` N/A under AutoGluon. The cross-validation-selected MSDPO candidate for this dataset one-hot encodes `Embarked` with `pandas.get_dummies` after replacing missing values by the literal string `'Missing'`. Our harness applies each generated `modify_features` function independently to the train and the test split, and the two missing `Embarked` values in this dataset both fall in the training portion. The function therefore emits an `Embarked_Missing` column on the train split but not on the test split; the two feature matrices have different schemas and the downstream fit fails. The transformation itself is well formed, and the same candidate is scored successfully under XGBoost, where the column mismatch does not arise. The split-dependent encoding therefore creates a schema mismatch under AutoGluon, and we record the candidate as N/A under the shared execution rule.

G Case Study: Heart Disease

We include a case study on the `heart` dataset to illustrate the type of feature transformations produced by the aligned model. In the main five-seed evaluation, MSDPO achieves the best AUC on this dataset, 0.9253, compared with 0.9247 for the strongest baseline in Table 1. The per-dataset ablation also shows all aligned variants outperform Base on this dataset, with the full model performing best: Base obtains 0.9229, S1-only obtains 0.9267, S2-only obtains 0.9344, and the full MSDPO model obtains 0.9376.

Listing 3 shows the CV-selected MSDPO candidate generated by MSDPO. The candidate introduces 14 new features. These transformations fall into four interpretable groups. First, the model converts categorical clinical signals into numerical or binary encodings, including `abnormal_resting_ecg`, `encoded_st_slope`, `encoded_chest_pain`, and `encoded_exercise_angina`. Second, it constructs nonlinear risk terms such as `age_squared` and `oldpeak_squared`. Third, it creates ratio-style physiological features, including `cholesterol_restingbp_ratio`, `maxhr_age_ratio`, and `maxhr_minus_220_plus_age`. Finally, it adds interaction features such as `oldpeak_st_slope_product`, `exercise_angina_oldpeak_interaction`, and `abnormal_ecg_exercise_interaction`.

Table 8: Frontier-model comparison under the seed-42, 80/20 split, thinking-mode setting. We report ROC-AUC; higher is better. Qwen3-8B uses the same backbone and inference prompt as MSDPO but without DPO alignment. Best per row is bolded, best non-tied runner-up underlined. Ties use the min-rank convention; "Wins / ties" counts sole-best and tied-best datasets.

Dataset	Qwen3-8B	Claude Sonnet 4.6	MSDPO
ad-click	0.7770	0.7613	**0.7772**
adult	**0.9295**	0.9283	0.9280
balance-scale	**1.0000**	**1.0000**	**1.0000**
bank	0.9286	0.9270	**0.9307**
banknote	**1.0000**	**1.0000**	0.9996
breast-w	0.9932	0.9943	**0.9948**
car	**1.0000**	0.9559	**1.0000**
cdc	**0.8194**	0.8186	0.8187
cmc	**0.7075**	0.7003	0.7031
communities	0.8705	0.8742	**0.8751**
credit-g	0.7507	**0.7629**	0.7526
cultivar	0.8408	0.7676	**0.8545**
customer	**0.8245**	0.8241	0.8231
heart	0.9285	0.9230	**0.9286**
junglechess	0.9983	**0.9998**	0.9993
myocardial	**0.6892**	0.6232	0.6476
pc1	0.8285	**0.8411**	0.8201
titanic	**0.8465**	0.8304	**0.8465**
vehicle	0.9238	0.9352	**0.9420**
water	0.6194	0.6489	**0.6554**
Mean rank	1.85	2.20	**1.65**
Wins / ties	5 / 4	3 / 2	**8 / 3**

Overall, the generated function illustrates the intended behavior of MSDPO: it uses schema and task semantics to produce executable, interpretable, and prediction-relevant feature transformations.

Table 9: Per-dataset ablation across the four model variants on the 20 classification datasets used in the main evaluation. Values are AUC under the seed-42, 80/20 split, thinking-mode evaluation convention. Best per row is bolded.

Dataset	Base	S1-only	S2-only	MSDPO
ad-click	0.7625	0.7488	0.7504	**0.7678**
adult	0.9291	0.9297	0.9290	**0.9309**
balance-scale	**1.0000**	**1.0000**	**1.0000**	**1.0000**
bank	0.9295	**0.9319**	0.9308	0.9306
banknote	**1.0000**	0.9999	**1.0000**	**1.0000**
breast-w	0.9939	0.9925	0.9943	**0.9984**
car	**1.0000**	**1.0000**	**1.0000**	**1.0000**
cdc	0.8194	0.8199	0.8194	**0.8258**
cmc	0.7114	0.7161	0.7062	**0.7448**
communities	0.8728	0.8714	**0.8789**	0.8526
credit-g	0.7774	0.7606	0.7413	**0.8054**
cultivar	0.8750	0.8691	0.8584	**0.8968**
customer	0.8205	0.8167	0.8153	**0.8352**
heart	0.9229	0.9267	0.9344	**0.9376**
junglechess	0.9983	0.9981	0.9991	**0.9993**
myocardial	**0.6976**	0.6379	0.6515	0.6795
pc1	0.8456	0.8195	0.8462	**0.8763**
titanic	0.8506	0.8532	0.8573	**0.8730**
vehicle	0.9425	**0.9441**	0.9356	0.9348
water	0.6414	0.6297	0.6616	**0.6700**
Mean rank	2.45	2.80	2.50	**1.45**

	XGBoost			AutoGluon		
Dataset	Raw	Base FE	MSDPO FE	Raw	Base FE	MSDPO FE
ad-click	0.7527	0.7770	**0.7772**	0.7937	**0.7950**	0.7947
adult	**0.9297**	0.9295	0.9280	**0.9316**	**0.9316**	0.9315
balance-scale	0.8950	**1.0000**	**1.0000**	0.9917	**1.0000**	**1.0000**
bank	0.9267	0.9286	**0.9307**	0.9371	0.9379	**0.9383**
banknote	**1.0000**	**1.0000**	0.9996	**1.0000**	**1.0000**	**1.0000**
breast-w	0.9862	0.9932	**0.9948**	**0.9930**	0.9896	0.9918
car	0.9999	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.9998
cdc	**0.8194**	**0.8194**	0.8187	**0.8257**	0.8256	0.8250
cmc	0.6963	**0.7075**	0.7031	0.7632	0.7613	**0.7638**
communities	0.8665	0.8705	**0.8751**	**0.8932**	0.8906	0.8886
credit-g	**0.7577**	0.7507	0.7526	0.7780	0.7943	**0.8125**
cultivar	0.7246	0.8408	**0.8545**	**0.8096**	0.7080	0.7773
customer	0.8179	**0.8245**	0.8231	0.8448	0.8457	**0.8467**
heart	0.9230	0.9285	**0.9286**	0.9249	0.9230	**0.9281**
junglechess	0.9754	0.9983	**0.9993**	0.9801	0.9997	**0.9999**
myocardial	0.6759	**0.6892**	0.6476	**0.7338**	0.7021	0.6877
pc1	**0.8346**	0.8285	0.8201	**0.8742**	0.8578	0.8414
titanic	0.8318	**0.8465**	**0.8465**	**0.8567**	0.8490	N/A
vehicle	0.9382	0.9238	**0.9420**	0.9561	0.9475	**0.9593**
water	0.6088	0.6194	**0.6554**	0.6542	**0.6609**	0.6584
Mean rank ↓	2.40	1.70	**1.65**	1.85	1.95	**1.84**
Mean AUC	0.8480	0.8638	**0.8648**	**0.8771**	0.8710	0.8760
Rank-1	5/20	8/20	**12/20**	**10/20**	6/20	9/19

Table 10: Per-dataset classification results under two downstream learners at seed 42, the detail behind Table 3. Values are ROC-AUC; higher is better. Within each learner block the best value in a row is bolded and the best non-tied runner-up is underlined. Mean ranks follow the min-rank convention over the four-decimal values shown. N/A marks a feature source whose selected candidate could not be applied to the test split; under AutoGluon this occurs for MSDPO on titanic, so its AutoGluon aggregates are computed over 19 datasets while the other two columns cover all 20.

Listing 1: The canonical generation prompt template, shown fully instantiated for the heart dataset.

```
You are an expert feature engineer in the field of the given dataset.

Given the prediction task and the dataset's feature list & examples below, generate as many
   diverse, creative, and high-quality new feature ideas as possible based on your domain
   knowledge. Each feature should be computationally distinct from the others. For each feature,
   describe the transformation in a concise, one-sentence docstring. Then implement all features
   in Python as a single function that is correct and runs without errors. Remove any irrelevant,
   redundant, or less informative original features to enhance overall performance. You should
   strictly follow the format:
<features_start>
new feature 1: <feature_name>: <one-sentence description>
new feature 2: <feature_name>: <one-sentence description>
...
<features_end>
<function_start>
def modify_features(df):
    # ... your code here ...
    return df
<function_end>

Task:
Does the coronary angiography of this patient show a heart disease? Yes or no?

Features:
- Age: age of the patient (numerical variable)
- Sex: sex of the patient (categorical variable with categories [F, M])
- ChestPainType: chest pain type (categorical variable with categories [ASY, ATA, NAP, TA])
- RestingBP: resting blood pressure [mm Hg] (numerical variable)
- Cholesterol: serum cholesterol [mm/dl] (numerical variable)
- FastingBS: fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise] (numerical variable)
- RestingECG: resting electrocardiogram results (categorical variable with categories [LVH,
   Normal, ST])
- MaxHR: maximum heart rate achieved (numerical variable)
- ExerciseAngina: exercise-induced angina (categorical variable with categories [N, Y])
- Oldpeak: oldpeak = ST [Numeric value measured in depression] (numerical variable)
- ST_Slope: the slope of the peak exercise ST segment (categorical variable with categories
   [Down, Flat, Up])

Examples:
Age is 63, Sex is F, ChestPainType is ATA, RestingBP is 140, Cholesterol is 195, FastingBS is 0,
   RestingECG is Normal, MaxHR is 179, ExerciseAngina is N, Oldpeak is 0.0, ST_Slope is Up, the
   result is no.
Age is 53, Sex is M, ChestPainType is NAP, RestingBP is 145, Cholesterol is 518, FastingBS is 0,
   RestingECG is Normal, MaxHR is 130, ExerciseAngina is N, Oldpeak is 0.0, ST_Slope is Flat, the
   result is yes.
Age is 65, Sex is M, ChestPainType is ASY, RestingBP is 160, Cholesterol is 0, FastingBS is 1,
   RestingECG is ST, MaxHR is 122, ExerciseAngina is N, Oldpeak is 1.2, ST_Slope is Flat, the
   result is yes.
Age is 56, Sex is M, ChestPainType is ASY, RestingBP is 130, Cholesterol is 0, FastingBS is 0,
   RestingECG is LVH, MaxHR is 122, ExerciseAngina is Y, Oldpeak is 1.0, ST_Slope is Flat, the
   result is yes.
```

Listing 2: The Critic prompt used to score Stage 1 feature proposals.

```
You are a discerning critic of feature-engineering ideas.

Task:
{TASK}

Features:
{FEATURES}

Examples:
{EXAMPLES}

Proposed new features:
{NEW_FEATURES}

Rating:
Rate the overall quality and diversity of the proposed new features
on a scale from 0.0 (worst) to 1.0 (best).
On the last line, output only the final numeric score between 0.0
and 1.0. If there are no new features, output 0.0.
```

Listing 3: CV-selected MSDPO output for the `heart` dataset. The listing shows the complete generated feature set and the corresponding executable `modify_features` function.

```
<features_start>
new feature 1: abnormal_resting_ecg: Binary indicator for abnormal resting ECG (LVH or ST).
new feature 2: encoded_st_slope: Numerical encoding of ST_Slope (Down=0, Flat=1, Up=2).
new feature 3: encoded_chest_pain: Numerical encoding of ChestPainType (ASY=0, ATA=1, NAP=2, TA=3).
new feature 4: chest_pain_sex_interaction: Interaction between encoded ChestPainType and sex (encoded as 0 for F, 1 for M).
new feature 5: oldpeak_st_slope_product: Product of Oldpeak and encoded ST_Slope to capture interaction between ST depression
   and slope.
new feature 6: age_squared: Square of Age to capture non-linear age effects.
new feature 7: exercise_angina_oldpeak_interaction: Interaction between encoded ExerciseAngina and Oldpeak.
new feature 8: cholesterol_restingbp_ratio: Ratio of Cholesterol to RestingBP to assess lipid-pressure relationship.
new feature 9: fastingbs_cholesterol_product: Product of FastingBS and Cholesterol to identify combined hyperglycemic and
   hyperlipidemic states.
new feature 10: maxhr_age_ratio: Ratio of MaxHR to Age to evaluate heart rate efficiency relative to age.
new feature 11: maxhr_minus_220_plus_age: Deviation of MaxHR from the expected maximum heart rate based on age (220 - Age).
new feature 12: oldpeak_squared: Square of Oldpeak to capture non-linear ST depression effects.
new feature 13: restingbp_cholesterol_product: Product of RestingBP and Cholesterol to assess combined cardiovascular risk
   factors.
new feature 14: abnormal_ecg_exercise_interaction: Interaction between abnormal_resting_ecg and encoded ExerciseAngina to
   capture combined cardiac abnormalities.
<features_end>

<function_start>
def modify_features(df):
    # Compute abnormal_resting_ecg and drop original feature
    df['abnormal_resting_ecg'] = df['RestingECG'].apply(
        lambda x: 1 if x in ['LVH', 'ST'] else 0
    )
    df = df.drop(columns=['RestingECG'])
    # Encode ST_Slope
    st_slope_map = {'Down': 0, 'Flat': 1, 'Up': 2}
    df['encoded_st_slope'] = df['ST_Slope'].map(st_slope_map)
    # Encode ChestPainType
    chest_pain_map = {'ASY': 0, 'ATA': 1, 'NAP': 2, 'TA': 3}
    df['encoded_chest_pain'] = df['ChestPainType'].map(chest_pain_map)
    # ChestPainType-sex interaction
    sex_encoded = df['Sex'].map({'F': 0, 'M': 1})
    df['chest_pain_sex_interaction'] = (
        df['encoded_chest_pain'] * sex_encoded
    )
    # Oldpeak * encoded_st_slope
    df['oldpeak_st_slope_product'] = (
        df['Oldpeak'] * df['encoded_st_slope']
    )
    # Age squared
    df['age_squared'] = df['Age'] ** 2
    # Encode ExerciseAngina
    exercise_map = {'N': 0, 'Y': 1}
    df['encoded_exercise_angina'] = df['ExerciseAngina'].map(exercise_map)
    # ExerciseAngina-Oldpeak interaction
    df['exercise_angina_oldpeak_interaction'] = (
        df['encoded_exercise_angina'] * df['Oldpeak']
    )
    # Cholesterol/RestingBP
    df['cholesterol_restingbp_ratio'] = (
        df['Cholesterol'] / df['RestingBP']
    )
    # FastingBS*Cholesterol
    df['fastingbs_cholesterol_product'] = (
        df['FastingBS'] * df['Cholesterol']
    )
    # MaxHR/Age
    df['maxhr_age_ratio'] = df['MaxHR'] / df['Age']
    # MaxHR - (220 - Age)
    df['maxhr_minus_220_plus_age'] = (
        df['MaxHR'] - (220 - df['Age'])
    )
    # Oldpeak squared
    df['oldpeak_squared'] = df['Oldpeak'] ** 2
    # RestingBP*Cholesterol
    df['restingbp_cholesterol_product'] = (
        df['RestingBP'] * df['Cholesterol']
    )
    # AbnormalECG-ExerciseAngina interaction
    df['abnormal_ecg_exercise_interaction'] = (
        df['abnormal_resting_ecg'] * df['encoded_exercise_angina']
    )
    return df
<function_end>
```