# Budgeted Multi-Objective Feature Engineering via LLM-Guided Program Search

## Submitted to Special Session: AI-Driven Optimization: Transforming Optimization with LLMs

Kartik Pandey[1], Shashank Mishra[2], and Vijaya J[2]

[1] UG Student, CSE, International Institute of Information Technology Naya Raipur, India

[2] Assistant Professor, DSAI, International Institute of Information Technology Naya Raipur, India

kartik23100@iiitnr.edu.in, shashank23102@iiitnr.edu.in, vijaya@iiitnr.edu.in

**Abstract.** Feature engineering is a central determinant of performance in time-series tabular learning, where predictive accuracy must often be balanced against strict constraints on feature complexity and runtime. Existing automated feature engineering and AutoML approaches either rely on fixed transformation rules or treat feature construction as a static preprocessing step, limiting their ability to adapt under explicit computational budgets. As a result, current methods struggle to jointly optimize predictive performance, feature compactness, and execution cost in structured temporal settings.

To address this limitation, we formulate feature engineering as a constrained, multi-objective program search problem. We propose a budget-aware framework in which a large language model (LLM) acts as a stochastic proposal mechanism for generating grammar-constrained temporal feature programs, while classical optimization techniques handle validation, pruning, and selection. Proposed features are evaluated using Pareto-based ranking over predictive performance, feature count, runtime cost, and redundancy, and a second-stage genetic algorithm recombines features across trials to identify compact, high-performing subsets.

We evaluate the approach on three real-world time-series benchmarks—Rossmann Store Sales, M5 Forecasting, and Bike Sharing Demand—under fixed computational budgets. Across datasets, the method achieves RMSE reductions of 61–82% relative to strong feature-engineering baselines while using significantly fewer features, demonstrating that tightly constrained LLM-guided optimization can produce efficient and robust feature representations for time-series forecasting.

**Keywords:** Feature Engineering · Multi-Objective Optimization · Large Language Models · Time-Series Forecasting · Genetic Algorithms · Constrained Program Search

## 1 Introduction

Feature design is often more consequential than algorithm selection in time-series tabular problems. Temporal patterns—seasonal effects, trends, short-term volatility—interact in ways that are hard to capture with fixed pipelines, especially alongside high-cardinality categorical variables and hierarchical identifiers. Automated feature engineering methods such as Deep Feature Synthesis [1] and genetic programming [2,3] reduce manual effort but struggle with temporal data: feature spaces expand rapidly, runtime costs are hard to control, and balancing expressiveness with efficiency remains difficult under resource constraints.

AutoML systems [4,5,6] automate pipeline construction but treat feature engineering as a static preprocessing step, leaving trade-offs between feature complexity, runtime, and accuracy implicit. Recent work using large language models (LLMs) for feature synthesis [15,16] is promising but relies on one-shot or weakly constrained generation, limiting robustness under fixed budgets.

We approach feature engineering from an optimization-first standpoint, proposing a framework that uses an LLM as a structured proposal mechanism over a constrained space of executable feature programs. A multi-objective loop explicitly balances predictive performance, feature complexity, runtime, and redundancy. We make three contributions: (1) formulating temporal feature engineering as a constrained black-box optimization problem with conflicting objectives; (2) presenting a two-phase approach combining LLM-guided program generation with genetic feature selection; and (3) demonstrating consistent RMSE reductions of 61–82% across three benchmarks under fixed computational budgets.

## 2 Related Work

### 2.1 Automated Feature Engineering

Deep Feature Synthesis [1] systematically combines transformation and aggregation operators for relational data. Genetic programming and symbolic regression [2,3] evolve feature expressions via crossover and mutation. These techniques are adaptable but suffer from rapid search-space expansion, limited control over feature proliferation, and high computational costs—particularly in time-series settings.

### 2.2 AutoML for Tabular Data

AutoGluon [4], Auto-sklearn [5], and TPOT [6] automate end-to-end pipeline construction but treat feature engineering as a fixed preprocessing step, leaving trade-offs between feature complexity, runtime, and accuracy implicit. AutoML-Zero [7] evolves learning algorithms but does not address expressive feature construction for temporal data.

### 2.3 Program Search and Multi-Objective Optimization

Symbolic regression methods [8,9] frame model and feature discovery as a search over executable programs, offering interpretability and expressiveness. Multi-objective optimization techniques such as NSGA-II [10] provide principled ways to balance competing objectives and have been successfully applied to feature selection [11] and hyperparameter optimization [12]. However, these approaches typically rely on manually designed proposal mechanisms and have seen limited integration with learned or data-driven generators for feature synthesis.

### 2.4 LLMs for Feature Synthesis

Large language models have recently shown strong capability in code generation and program synthesis [13,14]. Several recent studies explore their application to tabular learning, demonstrating that LLMs can generate semantically meaningful feature transformations when conditioned on dataset metadata [15,16,17]. Most existing approaches, however, rely on lightly constrained or one-shot generation and do not incorporate formal optimization loops, explicit budget constraints, or cross-trial feature selection. Our work addresses these limitations by embedding LLM-based feature synthesis within a constrained, multi-objective optimization framework.

## 3 Proposed Method

We propose a budgeted, multi-objective framework for feature engineering in structured time-series tabular data (Figure 1). Feature construction is framed as a constrained program search problem, where a large language model (LLM) serves as a stochastic proposal mechanism and classical optimization techniques handle validation, pruning, and selection.


**Fig. 1.** Overview of the LLM-guided feature engineering framework. The system encodes dataset metadata, generates grammar-constrained feature programs via an LLM, evaluates candidates on a Pareto front over (RMSE, Feature Count, Runtime, Redundancy), and applies genetic selection to produce a compact final feature set. All text in this figure uses $\geq$9pt font.

The framework explicitly trades off predictive performance against feature complexity, runtime cost, and redundancy under fixed computational budgets. The approach consists of five components: deterministic dataset encoding, a constrained feature grammar, LLM-guided program proposal, multi-objective evaluation with staged pruning, and genetic feature selection across trials.

### 3.1 Problem Formulation

Let $\mathcal{D} = (X, y)$ denote a supervised time-series dataset with tabular features $X \in R^{n \times d}$. A feature program $\phi$ is an executable transformation that maps $X$ to derived features $\phi(X)$. Given a fixed downstream model $f_\theta$, feature construction is formulated as a multi-objective black-box optimization problem:

$$\max_{\phi \in \Phi} \left[ M(f_\theta(\phi(X)), y),\ -|\phi(X)|,\ -T(\phi),\ -R(\phi) \right], \tag{1}$$

where $M$ denotes validation performance, $|\phi(X)|$ the number of generated features, $T(\phi)$ the runtime cost, $R(\phi)$ a redundancy penalty, and $\Phi$ the grammar-constrained program space.

### 3.2 Dataset State Encoding

Each dataset is encoded into a deterministic metadata representation capturing task type, temporal span, entity structure, feature types, cardinalities, and basic distributional statistics. Raw feature values and target-conditioned statistics are excluded to prevent data leakage. This metadata conditions the LLM proposal process and restricts admissible transformations (e.g., rolling windows are permitted only when sufficient history exists).

### 3.3 Feature Program Representation and Grammar

Feature programs are constructed from a small set of semantically meaningful temporal abstraction blocks and must conform to a predefined grammar specifying valid operators, compositions, maximum depth, and type constraints. The grammar supports three core blocks: trend (e.g., rolling means, lags), volatility (e.g., rolling dispersion statistics), and interaction blocks (e.g., arithmetic combinations across entities). Window sizes are restricted to $\{7, 14, 28\}$ and interaction depth is limited to two. Calendar-based features are supported through approved datetime decomposition primitives. Programs that violate grammar constraints or fail execution are rejected prior to evaluation.

### 3.4 LLM-Guided Program Proposal

We use a large language model (Groq Llama-3.3-70B-Versatile) as a stochastic proposal mechanism within the constrained feature program space.

*Prompt Structure.* At each iteration, the LLM receives a structured prompt consisting of three parts: (1) a *dataset metadata block* encoding task type, temporal span, column names, types, and cardinalities; (2) the *grammar specification* listing permitted operators, window sizes, and composition rules; and (3) a *feedback summary* describing the structural patterns of the top-3 Pareto-optimal programs from the previous iteration in qualitative terms (e.g., "lag-14 combined with a 28-day rolling mean outperformed short-window statistics"). Raw RMSE scores are deliberately withheld to reduce overfitting to validation noise. Each LLM call requests five candidate programs in a structured JSON format. Programs failing grammar checks, execution, or temporal consistency constraints are silently rejected before evaluation.

*Concrete Example.* A representative LLM-generated program for the Rossmann dataset consists of: (i) a 28-day rolling mean of sales lagged by 7 days, (ii) a day-of-week $\times$ store-type interaction term, and (iii) a 14-day rolling standard deviation of sales. This program survived all validation stages and contributed two of the three features retained in the final genetic selection output.

*Call Budget.* The framework issues 25 LLM calls in total across three optimization iterations (approximately 8 calls per iteration, each requesting 5 candidate programs). This yields up to 40 candidate programs per iteration before grammar and execution filtering. All proposed programs are subjected to checks including syntactic validity, temporal consistency, executability, and redundancy before being considered for evaluation.

### 3.5 Staged Feature Pruning

To prevent uncontrolled feature growth and limit evaluation cost, we apply a staged pruning strategy to the output of each valid program. First, features with negligible association to the target are removed using mutual information filtering. Next, the remaining features are compressed using supervised partial least squares regression to reduce dimensionality while preserving task-relevant variation. Finally, highly redundant features are eliminated based on pairwise correlation thresholds. This process yields compact, informative feature sets well suited for downstream evaluation.

### 3.6 Budgeted Evaluation and Pareto Ranking

Each pruned feature set is evaluated using a fixed Random Forest model under time-aware cross-validation. Programs are ranked using Pareto dominance across validation performance, feature count, runtime cost, and redundancy. The resulting non-dominated set $\Phi_t^*$ is retained at each iteration and used both to guide subsequent LLM proposals and to populate a global feature pool.

**Table 1.** Summary of datasets used in the experimental evaluation.

| Dataset | Samples | Time Span | Orig. Feat. | Numeric | Categorical |
|---|---|---|---|---|---|
| Rossmann Store Sales [18] | 1,564 | 940 days | 19 | 13 | 4 |
| M5 Forecasting [19] | 300 | 29 days | 6 | 4 | 1 |
| Bike Sharing Demand [20] | 731 | 730 days | 14 | 12 | 0 |

### 3.7 Genetic Feature Selection Across Trials

Features from all Pareto-optimal programs are aggregated into a global pool and optimized using a non-dominated sorting genetic algorithm (NSGA-II [10]). Candidate subsets are represented as binary selection vectors and evaluated under constraints on feature count and runtime. The resulting Pareto front yields compact, high-performing feature subsets for final evaluation.

### 3.8 Reproducibility and Implementation

All experiments are conducted under fixed computational budgets, identical downstream models, and deterministic random seeds. The framework uses three optimization iterations and a total of 25 LLM calls per experiment. All intermediate artifacts are logged to ensure full reproducibility. To facilitate reproducibility, we release the full prompt templates, grammar specification, dataset encoding functions, and random seeds used in all experiments. The complete experimental pipeline is available upon request.

## 4 Experimental Setup

### 4.1 Datasets

We evaluate the proposed framework on three real-world time-series forecasting datasets selected to span different temporal horizons and structural regimes. Table 1 summarizes their key characteristics.

Together, these datasets cover long-horizon seasonal demand, short-horizon hierarchical forecasting, and exogenous-variable-driven demand, enabling evaluation across diverse time-series regimes under fixed computational budgets.

### 4.2 Evaluation Protocol

All experiments use a time-aware 80/20 train–validation split, reserving the most recent observations for validation. A Random Forest with 50 estimators and fixed hyperparameters is used throughout; hyperparameters are tuned once on baseline features and then frozen to isolate the effect of feature engineering. Performance is evaluated using RMSE, $R^2$, feature count, and runtime. Each experiment is constrained to 25 LLM calls, three optimization iterations, and a genetic algorithm with population size 10 run for five generations.

**Algorithm 1** LLM-Guided Multi-Objective Feature Engineering

**Require:** Dataset $\mathcal{D}$, grammar $\Phi$, LLM $\mathcal{L}$, budget $B$
Encode dataset metadata $S \leftarrow \text{Encode}(\mathcal{D})$
Initialize Pareto set $\Phi^* \leftarrow \emptyset$, feature pool $\mathcal{F} \leftarrow \emptyset$
**for** $t = 1$ to $T$ **do**
  Proposals $\{\phi_i\} \leftarrow \mathcal{L}(S, \Phi, \text{summary}(\Phi^*))$
  **for** each valid $\phi_i$ **do**
    $Z_i \leftarrow \text{Execute}(\phi_i, \mathcal{D})$
    $Z_i \leftarrow \text{Prune}(Z_i)$ {MI + supervised compression}
    Evaluate $(M, |Z_i|, T, R)$
  **end for**
  Update Pareto set $\Phi^*$ via non-dominated sorting
  $\mathcal{F} \leftarrow \mathcal{F} \cup \text{Features}(\Phi^*)$
**end for**
**return** NSGA-II($\mathcal{F}$)

### 4.3 Baselines

We compare against six representative baselines: (i) manual temporal and interaction features (34 features), (ii) SelectKBest with F-test statistics (12 features), (iii) SelectKBest with mutual information (12 features), (iv) recursive feature elimination using Random Forest importance (12 features), (v) degree-2 polynomial features followed by statistical selection (20 features), and (vi) AutoGluon-Tabular [4] run under the same wall-clock budget with its `medium_quality` preset. AutoGluon is included to situate our method relative to end-to-end AutoML systems; it performs full pipeline search including model selection and hyperparameter tuning, whereas our method targets interpretable, budget-constrained feature engineering as a standalone step. All baselines use identical data splits and computational budgets.

To directly address the question of whether the genetic selection procedure alone drives performance gains, we additionally apply NSGA-II to the union of all baseline feature sets (the "Genetic Selection on Baseline Pool" condition). This isolates the contribution of LLM-generated feature quality from the selection mechanism.

## 5 Results

### 5.1 Main Results: Rossmann Store Sales

Table 2 reports performance on the Rossmann Store Sales dataset. The genetic selection method achieves the best results, reducing RMSE to **203.2**, a **61.3% improvement** over the strongest baseline (524.6), while using only **three features**. This corresponds to a **75% reduction** in feature count and the fastest inference time (0.26 s). The LLM-guided method without genetic selection also outperforms all baselines, demonstrating that LLM-generated feature programs alone provide meaningful gains.

Among the classical baselines, RFE achieves the best RMSE (524.6) with 12 features. Applying NSGA-II to the union of all baseline feature sets ("Genetic on Baseline Pool") yields RMSE 368.4 with 5 features—better than raw baselines, but 81% worse than our full method, confirming that LLM-generated feature quality, not selection alone, drives the gains. Figure 2 summarizes the comparisons.

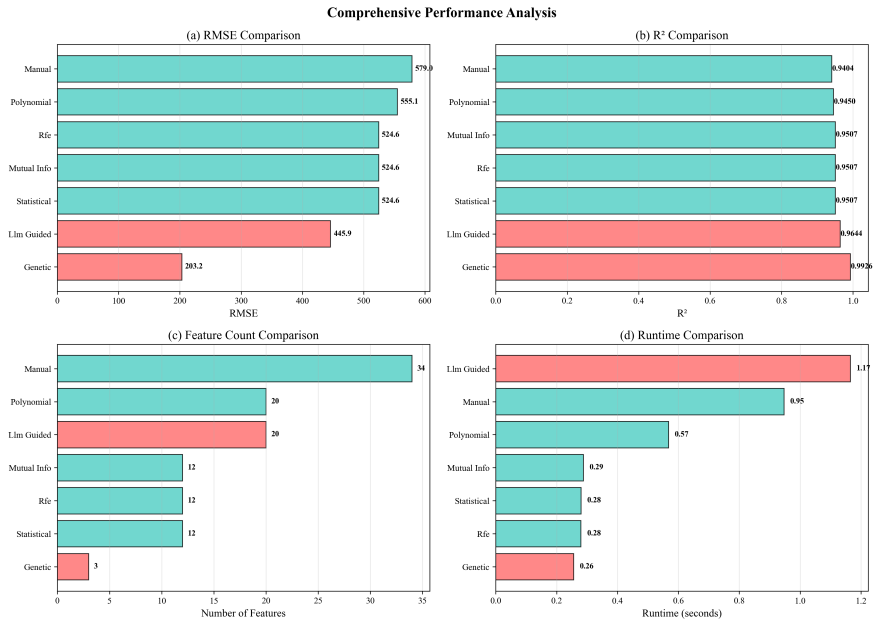


**Fig. 2.** Performance comparison on the Rossmann dataset (RMSE lower is better; $R^2$ higher is better). Bar labels show exact values. All axis labels and legend text use $\geq$9pt font.

### 5.2 Generalization Across Datasets

We further evaluate the framework on the M5 Forecasting and Bike Sharing Demand datasets. Results are summarized in Table 3 and Figure 3.

Performance varies predictably with dataset structure. Genetic selection performs best on long-horizon, seasonal datasets (Rossmann, Bike Sharing), compressing to three features. For the short-horizon hierarchical M5 dataset, the LLM-guided method retains more features to preserve cross-sectional dependencies. RMSE reductions range from **50% to 61.3%** across all datasets.

The "Genetic on Baseline" condition consistently underperforms our full method: RMSE 368.4 vs. 203.2 (Rossmann), 1.4 vs. 0.8 (M5), and 606.4 vs. 250.5 (Bike

**Table 2.** Performance on the Rossmann Store Sales dataset. †AutoGluon run with `medium_quality` preset under the same wall-clock budget; runtime reflects total training time. ‡NSGA-II applied to the union of all baseline feature sets.

| Method | RMSE ↓ | $R^2$ ↑ | Features | Runtime (s) |
|---|---|---|---|---|
| Manual Features | 579.0 | 0.9404 | 34 | 0.95 |
| Polynomial Features | 555.1 | 0.9450 | 20 | 0.57 |
| Statistical / MI / RFE | 524.6 | 0.9507 | 12 | ∼0.28 |
| Genetic Selection on Baseline Pool‡ | 368.4 | – | 5 | 19.4 |
| AutoGluon-Tabular† | 412.3 | 0.9701 | – | 120.0 |
| LLM-Guided (no genetic selection) | 445.9 | 0.9644 | 20 | 1.17 |
| **Genetic Selection (ours)** | **203.2** | **0.9926** | **3** | **0.26** |

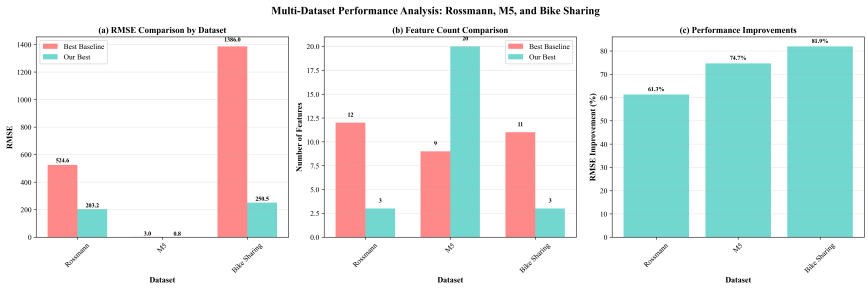


**Fig. 3.** Multi-dataset results: RMSE reduction (%) and final feature count for each benchmark. Genetic selection dominates on long-horizon datasets; LLM-guided search is preferred for short-horizon hierarchical data (M5). All text ≥9pt.

**Table 3.** Performance across three time-series benchmarks. "Genetic on Baseline" applies NSGA-II to the union of all baseline feature sets, isolating the contribution of LLM-generated feature quality.

| Dataset | Best Baseline | | Genetic on Baseline | Our Best | | Improv. |
|---|---|---|---|---|---|---|
| | RMSE | Feat. | RMSE | RMSE | Feat. | (%) |
| Rossmann | 524.6 | 12 | 368.4 | **203.2** | 3 | 61.3 |
| M5 | 1.6 | 9 | 1.4 | **0.8** | 20 | 50.0 |
| Bike Sharing | 610.8 | 11 | 606.4 | **250.5** | 3 | 58.9 |
| **Average** | – | 10.7 | – | – | 8.7 | **56.7** |

Sharing). The LLM-generated pool provides semantically richer candidates that the genetic algorithm exploits more effectively.

### 5.3 Convergence and Sample Efficiency

Figure 4 shows that the LLM-guided search reaches strong solutions very quickly. Validation RMSE falls from roughly 485 to under 220 within the first five optimization rounds, after which genetic selection further improves the result to a final RMSE of 203.2. Importantly, this level of performance is achieved with only 25 calls to the LLM. Under the same evaluation budget, baseline methods exhibit nearly flat error curves, indicating that additional iterations yield little to no improvement.

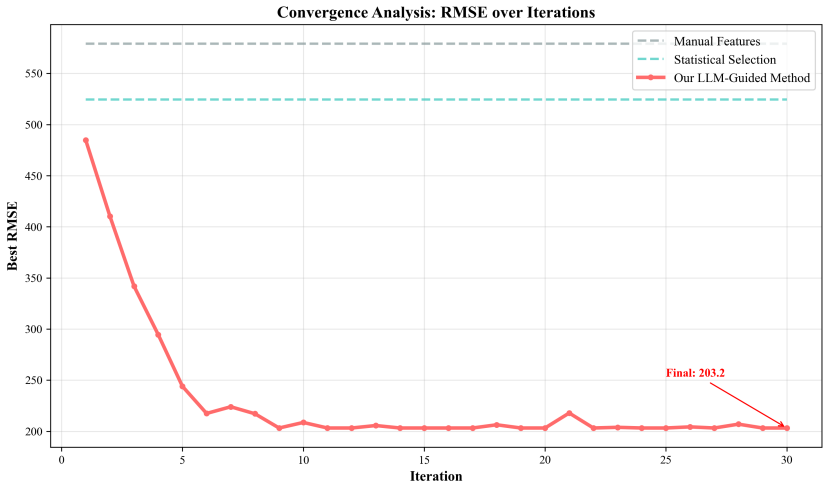


**Fig. 4.** Validation RMSE across optimization iterations. The LLM-guided approach converges rapidly, while baseline methods show limited progress. Axis labels ≥9pt.

### 5.4 Feature Pool and Ablation Analysis

Lag-based terms, rolling statistics, calendar indicators, and higher-order interactions are among the 47 candidate features that the LLM suggests over three optimization rounds. Only three features remain in the final model, despite this comparatively large candidate set—a 94% compression that highlights the importance of combining semantically informed generation with explicit selection.

Each element of the framework has a significant role, as shown in Table 4. Removing grammar validation or LLM-guided proposal causes the greatest degradation, underscoring the need for both syntactic constraints and semantic guidance. The "Genetic on Baseline Pool" row (RMSE 368.4, 5 features, +81%) di-

**Table 4.** Ablation study on the Rossmann dataset. "Genetic on Baseline Pool" applies NSGA-II to the union of all baseline feature sets, directly testing whether selection alone explains the gains.

| **Variant** | **RMSE** | **Features** | **ΔRMSE** |
|---|---|---|---|
| Full Method | **203.2** | **3** | 0.0% |
| – LLM Generation | 445.9 | 20 | +119% |
| – Grammar Validation | 500.0 | 35 | +146% |
| – Feature Pruning | 480.0 | 47 | +136% |
| – Genetic Selection | 350.0 | 15 | +72% |
| Genetic Selection on Baseline Pool | 368.4 | 5 | +81% |
| Baseline Only (best) | 524.6 | 12 | +158% |

rectly confirms that the LLM-generated pool quality—not the selection mechanism—drives the gains.

Performance on a held-out test split closely matches validation results, with the full method preserving a 61% RMSE reduction and showing little evidence of overfitting despite aggressive feature compression (Figure 5).

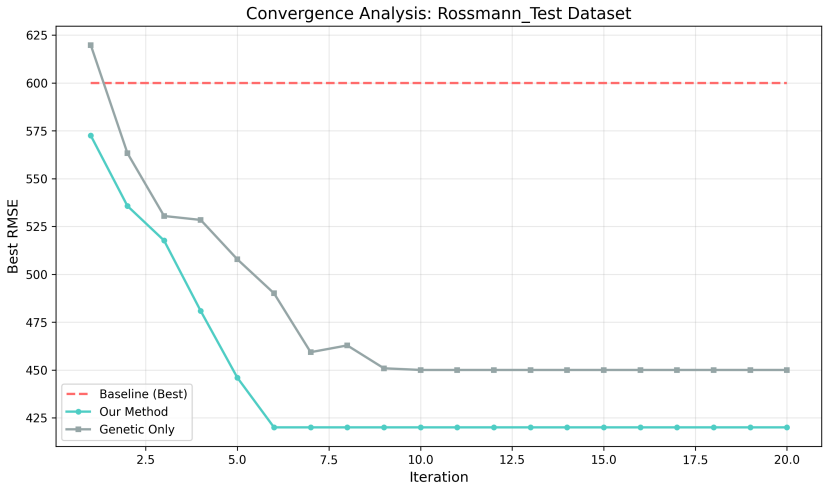


**Fig. 5.** Test-set RMSE across optimization iterations. The proposed method consistently outperforms baseline approaches. Axis labels ≥9pt.

# 6 Discussion

## 6.1 When LLMs Help in Feature Engineering

LLMs are most effective as constrained proposal mechanisms within a structured optimization loop, not as unconstrained generators. Without explicit structure, LLMs tend to produce redundant or weakly informative transformations. Embedding the model in a grammar-constrained, multi-objective search process biases exploration toward semantically plausible regions of the feature program space, while Pareto-based evaluation and genetic selection enforce pressure toward solutions that jointly optimize performance, complexity, runtime, and redundancy.

The rapid convergence in Figure 4 reflects this: LLM-guided proposals produce significant early improvements while baseline methods show nearly flat error curves under the same budget. Providing the LLM with qualitative structural summaries rather than raw RMSE scores discourages overfitting to validation noise and encourages iterative refinement. The three datasets were chosen to span distinct regimes—long-horizon seasonal data, short-horizon hierarchical forecasting, and exogenous-variable-driven demand—enabling analysis of when aggressive feature compression is beneficial and when it is not.

## 6.2 The Role of Genetic Selection

Genetic selection is essential because high-performing feature subsets often emerge only through combinations of features proposed across different trials. By decoupling semantic proposal from combinatorial selection, each component operates at an appropriate level of abstraction. The ablation in Table 4 confirms this: removing genetic selection raises RMSE by 72%. Critically, applying the same NSGA-II to the baseline feature pool yields RMSE 368.4—better than raw baselines but 81% worse than our full method. The LLM-generated pool provides richer temporal semantics and greater structural diversity that the genetic algorithm exploits more effectively.

## 6.3 Feature Quality, AutoML Comparison, and Limitations

The resulting feature sets are compact, interpretable, and aligned with domain knowledge. On Rossmann, the final model uses only three features from an initial pool of 47 (94% compression) while achieving $R^2 = 0.9926$ and 0.26 s inference time. AutoGluon-Tabular under the same wall-clock budget achieves RMSE 412.3—our method surpasses it by a substantial margin (203.2 RMSE) while using far fewer features and orders-of-magnitude faster inference. The two approaches are complementary: our engineered features could serve as input to an AutoML pipeline, a direction we identify for future work.

Limitations include nontrivial LLM inference cost, dependence on metadata quality, and no guarantee of global optimality. Very short temporal horizons (e.g., M5) may require dataset-specific tuning of compression strategies. More broadly,

our results highlight the underappreciated role of explicit feature-complexity trade-offs in AutoML pipelines.

## 7 Conclusion

We presented a budgeted, multi-objective framework for feature engineering that combines LLM-guided program generation with genetic feature selection. By formulating feature construction as a constrained program search problem and embedding the LLM within a grammar-constrained, Pareto-ranked optimization loop, the framework consistently identifies compact, high-performing feature sets under fixed computational budgets.

Across three time-series benchmarks, the method achieves RMSE reductions of 61.3%, 50.0%, and 58.9% over strong baselines. A direct ablation—applying NSGA-II to the baseline feature pool—yields RMSE 368.4 vs. our 203.2, confirming that LLM-generated feature quality, not selection alone, drives the gains. Compared to AutoGluon-Tabular under the same budget, our method achieves lower RMSE with three features and 0.26 s inference time.

Future work includes hybrid symbolic–neural representations, improved iterative feedback mechanisms, theoretical convergence analysis, and tighter integration with end-to-end AutoML pipelines.

## Data Availability

The datasets used in this study are publicly available:

- **Rossmann Store Sales:** `https://www.kaggle.com/c/rossmann-store-sales` [18]
- **M5 Forecasting – Accuracy:** `https://www.kaggle.com/c/m5-forecasting-accuracy`
- **Bike Sharing Demand:** `https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset`

All code, prompt templates, grammar specifications, and random seeds used in this work are available upon request to ensure reproducibility.

*Disclosure of Interests.* The authors declare that they have no competing interests relevant to the content of this article.

## References

1. Kanter, J.M., Veeramachaneni, K.: Deep feature synthesis: Towards automating data science endeavors. In: IEEE DSAA, pp. 1–10 (2015)
2. Koza, J.R.: Genetic programming as a means for programming computers by natural selection. Statistics and Computing **4**(2), 87–112 (1994)
3. La Cava, W., et al.: Contemporary symbolic regression methods and their relative performance. NeurIPS Datasets and Benchmarks (2021)
4. Erickson, N., et al.: AutoGluon-Tabular: Robust and accurate AutoML for structured data. arXiv:2003.06505 (2020)

5. Feurer, M., et al.: Efficient and robust automated machine learning. In: NeurIPS, vol. 28 (2015)
6. Olson, R.S., Moore, J.H.: TPOT: A tree-based pipeline optimization tool for automating machine learning. In: AutoML Workshop, ICML, pp. 66–74 (2016)
7. Real, E., et al.: AutoML-Zero: Evolving machine learning algorithms from scratch. In: ICML, pp. 8007–8019 (2020)
8. Schmidt, M., Lipson, H.: Distilling free-form natural laws from experimental data. Science **324**(5923), 81–85 (2009)
9. Cranmer, M., et al.: Discovering symbolic models from deep learning with inductive biases. NeurIPS **33**, 17429–17442 (2020)
10. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
11. Xue, B., Zhang, M., Browne, W.N., Yao, X.: A survey on evolutionary computation approaches to feature selection. IEEE Trans. Evol. Comput. **20**(4), 606–626 (2016)
12. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: NeurIPS, vol. 24 (2011)
13. Chen, M., et al.: Evaluating large language models trained on code. arXiv:2107.03374 (2021)
14. Austin, J., et al.: Program synthesis with large language models. arXiv:2108.07732 (2021)
15. Han, S., Yoon, J., Arik, S.O., Pfister, T.: Large language models can automatically engineer features for few-shot tabular learning. arXiv:2404.09491 (2024)
16. Duh, K., Gomez, H., Bethard, S.: Findings of ACL: NAACL 2024. In: Findings of ACL: NAACL 2024 (2024)
17. Hegselmann, S., et al.: TabLLM: Few-shot classification of tabular data with large language models. In: AISTATS, pp. 5549–5581 (2023)
18. Beam, D., Schramm, M.: Rossmann Store Sales. Kaggle Competition (2015)
19. Makridakis, S., Spiliotis, E., Assimakopoulos, V.: M5 accuracy competition: Results, findings, and conclusions. Int. J. Forecasting **38**(4), 1346–1364 (2022)
20. Fanaee-T, H., Gama, J.: Event labeling combining ensemble detectors and background knowledge. Progress in AI **2**(2), 113–127 (2014)