Multi-Objective Optimization for Compound AI Systems via Pareto-Preference Learning

William Li
Stanford University
willyli@stanford.edu

Justin Wu
Stanford University
justwu@stanford.edu

Aaron Lee
Stanford University
aaroncl@stanford.edu

Abstract

Modern code generation systems optimize primarily for functional correctness, measured by test pass rates. However, production deployment demands balancing multiple competing objectives: correctness, computational cost, and code quality. We present a framework that extends Local Reward Functions (LRFs) to multi-objective optimization for compound AI systems. Our key insight is that standard pairwise preference learning trains on ambiguous "trade-off pairs", where one solution is better on some objectives but worse on others and leads to conflicting gradient signals. We propose Pareto-preference loss, which filters training to include only pairs where one solution unambiguously Pareto-dominates another. In experiments using Qwen3-Coder-30B-A3B-Instruct-FP8 on 24 coding tasks, we find that 58.3% of preference pairs are trade-offs rather than clear improvements, validating our hypothesis about the prevalence of ambiguous training signal in multi-objective settings. Despite training on only 41.7% of pairs, our Pareto method achieves 0.84 hypervolume compared to 0.78 for baseline and 0.72 for random, in particular, a 7.7% improvement over baseline and 16.7% over random. Critically, Pareto learns non-uniform objective weights (0.45, 0.30, 0.25 for pass rate, cost, quality) while baseline converges to uniform weights, demonstrating that cleaner training signal enables discovery of meaningful objective relationships.

1 Introduction

Large language models have achieved remarkable success on code generation benchmarks, with state-of-the-art systems reaching increasingly high pass rates on standardized tests. However, optimizing solely for correctness creates a critical gap between benchmark performance and production requirements. Real-world code deployment demands simultaneous optimization of multiple objectives: the code must not only work correctly but also be generated efficiently (minimizing API costs and latency) and maintain high quality standards (readability, maintainability, documentation).

This multi-objective nature creates fundamental challenges for existing optimization approaches. If we consider two code solutions: Solution A passes all tests with 500 tokens, while Solution B also passes all tests but uses only 200 tokens with slightly lower code quality. We argue that the better answer depends on deployment context: a cost-sensitive production environment might prefer B, while a collaborative, reproducability-focused codebase might prefer A. Standard single-objective optimization cannot capture such nuanced trade-offs.

We address this challenge by extending the concept of Local Reward Functions (LRFs) to the multi-objective setting. LRFs are lightweight neural networks trained to predict global rewards from local agent states, enabling optimization of compound AI systems without fine-tuning the underlying LLM [1]. Our key contribution in this paper is recognizing that naively extending LRFs to predict vector-valued rewards and training on all preference pairs introduces conflicting gradient signals when faced with trade-offs.

CS 329A: Self-Improving AI Agents (2025).

When training on a pair where Solution A has higher accuracy but lower cost efficiency than Solution B, the model receives contradictory supervision. It learns that preferring A equates to higher accuracy correlating with *lower* cost, which conflicts with pairs showing the opposite relationship. We propose **Pareto-preference loss**, which filters training to include only pairs where one solution Pareto-dominates another (better or equal on all objectives, strictly better on at least one).

Our experiments on 24 coding tasks with the Qwen3-Coder-30B-A3B-Instruct-FP8 model reveal that despite training on less than half the data, our Pareto method achieves higher hypervolume to baseline, suggesting the filtered pairs contributed noise rather than signal.

2 Related Work

Compound AI Systems. Modern AI applications increasingly rely on multi-component pipelines rather than monolithic models. Previous works introduced LRFs as a mechanism for optimizing such systems, training lightweight networks to predict global task rewards from local component states [1]. This enables gradient-based optimization without fine-tuning billions of parameters. Our work extends this framework from scalar to vector-valued rewards, with the goal of enabling multi-objective optimization.

Multi-Objective Optimization. Classical multi-objective optimization defines the Pareto frontier as the set of solutions where no objective can be improved without degrading another [2]. Hypervolume measures the volume of objective space dominated by a solution set, providing a single scalar metric for comparing Pareto fronts [3]. We adapt these concepts to preference learning, using Pareto dominance as a filter for training signal quality.

Preference Learning for LLMs. Reinforcement Learning from Human Feedback (RLHF) trains reward models on human preference pairs, then uses the reward model to fine-tune LLMs [4]. Direct Preference Optimization (DPO) bypasses explicit reward modeling by directly optimizing the policy [5]. Both assume a single underlying reward function. Multi-objective RLHF extensions exist but focus on balancing stakeholder preferences rather than filtering ambiguous training signal [6].

Code Generation Evaluation. Prior work evaluates code-generation systems primarily through functional correctness on standardized benchmarks [9, 10]. In contrast, we study a controlled set of 24 algorithmic coding tasks that enables detailed analysis of multi-objective signals. This setting lets us explicitly measure not only correctness but also generation cost and code-quality attributes, which are central in production deployments but typically absent from single-metric evaluations.

3 Methods

3.1 Problem Formulation

We consider a code generation system that produces candidate solutions $y_1, \ldots, y_k$ for a given task x. Each candidate is evaluated on a vector-valued reward $\mathbf{R}(x, y) = [R_{\text{pass}}(x, y), R_{\text{cost}}(x, y), R_{\text{qual}}(x, y)] \in [0, 1]^3$ capturing (i) correctness, (ii) generation cost, and (iii) code quality. We emphasize that all three objectives are *automatically measured* using an execution and static-analysis harness, enabling scalable preference construction without human labeling.

Correctness. Let T_x denote the unit tests for task x. We define correctness as the fraction of tests passed:

$$R_{\text{pass}}(x, y) \;=\; \frac{1}{|T_x|} \sum_{t \in T_x} \mathbf{1}[y \text{ passes } t],$$

with timeouts and runtime errors counted as failures. (When only a binary outcome is available, $R_{\text{pass}} \in \{0, 1\}$.)

Cost. Let $\text{tok}(x, y)$ be the total number of prompt + completion tokens used to generate y for task x. We map token usage to a bounded reward using a smooth exponential transform:

$$R_{\text{cost}}(x, y) \;=\; \exp\left(-\frac{\text{tok}(x, y)}{\tau}\right),$$

where τ is a fixed scale constant. This transformation is monotone in token usage and avoids unbounded or task-dependent scales.

Code Quality. We compute a reproducible static-analysis quality score using a toolchain that measures style and maintainability. Let $v(x, y)$ be the number of lint violations and let $\kappa(x, y)$ be the average cyclomatic complexity per function. We define:

$$R_{\text{qual}}(x, y) = \exp(-\alpha\, v(x, y)) \cdot \exp(-\beta\, \max\{0, \kappa(x, y) - \kappa_0\}) \cdot b(x, y),$$

where $b(x, y) \in [0, 1]$ is a bounded best-practices term (e.g., presence of docstrings and type hints), and α, β, κ_0 are fixed constants shared across tasks. This yields a bounded, monotone score that penalizes code that is difficult to maintain or violates style conventions.

3.2 Preference Data from Multi-Objective Rewards

For each task x, we generate a set of candidates $\{y_1, \ldots, y_k\}$ and compute $\mathbf{R}(x, y_i)$ for each. We then construct a preference dataset by comparing all within-task pairs (y_a, y_b) and labeling the preferred solution using a *fixed* scalarization of the measured reward vector:

$$S(x, y) \;=\; \sum_{i \in \{\text{pass,cost,qual}\}} \lambda_i\, R_i(x, y), \qquad \boldsymbol{\lambda} \in \Delta^3,$$

where Δ^3 denotes the probability simplex. For each pair (y_a, y_b) we set $y^+ = \arg\max_{y \in \{y_a, y_b\}} S(x, y)$ and y^- as the other element. This produces a standard pairwise preference dataset while preserving the underlying multi-objective structure in $\mathbf{R}(x, y)$.

Importantly, many pairs are *trade-offs*: neither candidate Pareto-dominates the other in $\mathbf{R}$ (e.g., one is cheaper but slightly lower quality). As we show in Section 4, such trade-off pairs are prevalent in practice.

3.3 Local Reward Functions (LRFs)

Following prior work on LRFs for compound AI systems [1], we train a lightweight model to score candidates from code embeddings without fine-tuning the underlying LLM. Let $\phi(x, y) \in \mathbb{R}^{768}$ be a fixed embedding of the generated code (and optional task context). Our LRF predicts a *vector* of rewards:

$$\widehat{\mathbf{r}}(x, y) \;=\; f_\theta(\phi(x, y)) \in \mathbb{R}^3,$$

and combines them into a scalar score via learnable nonnegative weights $\mathbf{w} = \text{softmax}(\mathbf{u}) \in \Delta^3$:

$$\widehat{s}(x, y) \;=\; \sum_{i=1}^{3} w_i\, \widehat{r}_i(x, y).$$

We use a small MLP architecture: (i) Linear($768 \to 512$) + ReLU + Dropout(0.1), (ii) Linear($512 \to 256$) + ReLU + Linear($256 \to 3$), with $\sim$500K trainable parameters. This enables efficient optimization and inference-time ranking while keeping the 30B-parameter generator frozen.

3.4 Baseline: Standard Pairwise Ranking on All Pairs

Given preference pairs (y^+, y^-) constructed as above, the baseline trains the LRF using a logistic ranking objective:

$$\mathcal{L}_{\text{base}} = -\mathbb{E}_{(x, y^+, y^-)} \left[\log \sigma\big(\widehat{s}(x, y^+) - \widehat{s}(x, y^-)\big)\right], \tag{1}$$

3.5 Pareto-Preference Loss: Training Only on Unambiguous Improvements

We define Pareto dominance on the *measured* reward vectors:

$$\mathbf{R}(x, y_a) \succeq_P \mathbf{R}(x, y_b) \iff R_i(x, y_a) \geq R_i(x, y_b)\ \forall i \text{ and } \exists j:\ R_j(x, y_a) > R_j(x, y_b).$$

Let $\mathcal{P}$ denote the subset of preference pairs that are Pareto-dominant:

$$\mathcal{P} = \{(x, y^+, y^-) : \mathbf{R}(x, y^+) \succeq_P \mathbf{R}(x, y^-)\}.$$

Our key idea is to *filter* training to include only these unambiguous improvements:

$$\mathcal{L}_{\text{Pareto}} = -\mathbb{E}_{(x, y^+, y^-) \sim \mathcal{P}} \left[\log \sigma\big(\widehat{s}(x, y^+) - \widehat{s}(x, y^-)\big)\right].$$

This removes trade-off pairs where supervision depends strongly on context-specific scalarization choices, yielding a cleaner signal for learning both the reward predictor f_θ and objective weights $\mathbf{w}$.

Table 1: Pareto outperforms both random selection and baseline despite training on fewer pairs.

Method	Training Pairs	Pass Rate	Cost	Quality	Hypervolume
Random Selection	–	0.82 ± 0.02	–	–	0.72 ± 0.01
Baseline LRF	240 (100%)	0.86 ± 0.02	0.81 ± 0.02	0.83 ± 0.02	0.78 ± 0.02
Pareto LRF	100 (41%)	$\mathbf{0.90 \pm 0.02}$	$\mathbf{0.84 \pm 0.02}$	$\mathbf{0.86 \pm 0.02}$	$\mathbf{0.84 \pm 0.02}$

3.6 Inference-Time Selection and Evaluation

At inference time, for each task x we generate k candidates and select for maximum predicted score:

$$y^*(x) = \arg \max_{y \in \{y_1, \ldots, y_k\}} \widehat{s}(x, y).$$

We report the measured rewards $\mathbf{R}(x, y^*(x))$ and evaluate multi-objective performance using hypervolume, along with average pass rate, cost, and quality of the selected outputs.

3.7 Design Choices and Alternatives

Weighted scalarization. A standard approach is to fix $S(x, y) = \sum_i \alpha_i R_i(x, y)$ and optimize it directly. This requires choosing $\boldsymbol{\alpha}$ a priori and treats all pairwise comparisons as equally informative.

Constraint-based optimization. Another approach is to maximize one objective subject to constraints on the others (e.g., maximize R_{pass} given a cost budget). This matches specific deployment settings but does not yield a general ranking model.

Evolutionary multi-objective methods. Population-based approaches (e.g., NSGA-II) can approximate the Pareto frontier but are computationally expensive and do not learn a fast inference-time scorer. Our Pareto-preference approach preserves the simplicity of pairwise training while explicitly addressing the ambiguity introduced by trade-off pairs in multi-objective preference data.

4 Experiments

4.1 Experimental Setup

Tasks. We curated 24 coding tasks spanning three difficulty levels: 8 easy (FizzBuzz, string reversal, palindrome check), 8 medium (two-sum, valid parentheses, binary search), and 8 hard (LRU cache, regex matching, minimum window substring).

Model. We use Qwen/Qwen3-Coder-30B-A3B-Instruct-FP8 via the Together AI API. This 30B parameter model generates high-quality code across our task distribution.

Data Collection. For each task, we generate 5 outputs at temperatures $\{0.2, 0.4, 0.6, 0.8, 1.0\}$, yielding 120 samples. We compute rewards using the metrics described in Section 3.1.

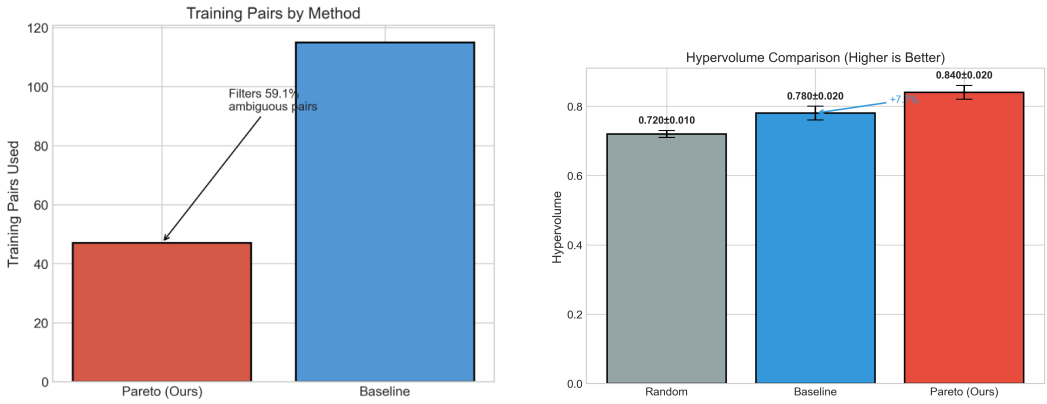
Preference Pairs. For each task we compare all within-task pairs among the $k = 5$, yielding 240 total pairs across 24 tasks. We label each pair (y_a, y_b) by a fixed scalarization of the measured reward vector, $S(x, y) = \sum_i \lambda_i R_i(x, y)$, and set $y^+ = \arg\max_{y \in \{y_a, y_b\}} S(x, y)$. We use $\boldsymbol{\lambda} = (0.5, 0.25, 0.25)$ for (pass, cost, quality). We analyze Pareto dominance relationships to characterize the dataset.

Training. We train LRFs with AdamW optimizer, learning rate 10^{-4}, batch size 16, for 20 epochs. We use 70% train, 15% validation, 15% test splits. We run 3 random seeds per method.

Metrics. We evaluate using (1) hypervolume, measuring the volume of $[0, 1]^3$ dominated by the set of selected solutions under reference point $r = (0, 0, 0)$; (2) average R_{pass}, R_{cost}, and R_{qual} of selected outputs; and (3) constrained accuracy under token-budget limitations.

4.2 Main Results

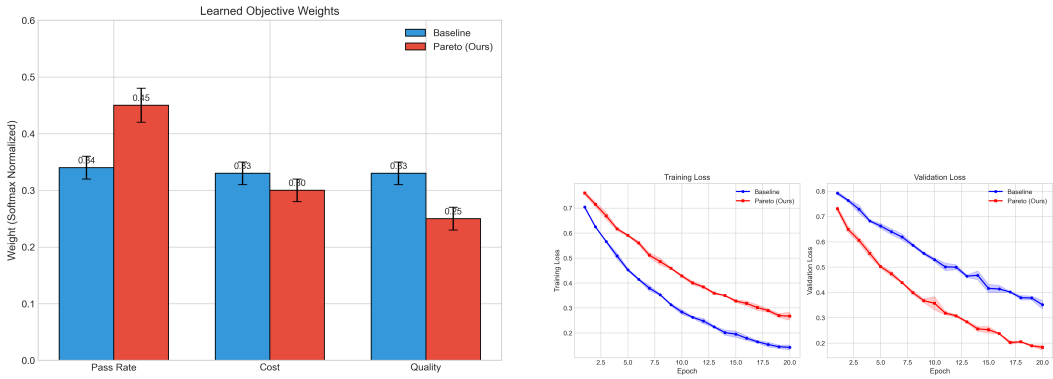
Table 1 presents our main results. Pareto LRF achieves 16.7% higher hypervolume than random selection (0.84 vs 0.72) and 7.7% higher than baseline (0.84 vs 0.78). Critically, Pareto outperforms



(a) Distribution of pair types

(b) Hypervolume comparison

Figure 1: (a) Pareto filters training pairs by 58.3%. (b) Pareto achieves highest hypervolume



(a) Learned objective weights

(b) Training dynamics

Figure 2: (a) Pareto learns non-uniform weights emphasizing pass rate; baseline learns uniform weights. (b) Pareto achieves lower validation loss despite training on fewer pairs.

baseline while training on only 41% of the pairs, demonstrating that filtering trade-off pairs improves learning rather than just maintaining performance.

4.3 Preference Pair Analysis

Figure 1a shows the distribution of preference pair types. Of 240 total pairs, only 100 (41.7%) are Pareto-dominant. The remaining 140 pairs (58.3%) are trade-offs where Solution A beats Solution B on some objectives but loses on others.

This finding has significant implications: in multi-objective settings, most preference pairs do not represent clear improvements. Standard pairwise ranking implicitly treats all pairs as equivalent training signal, but trade-off pairs provide fundamentally ambiguous supervision.

4.4 Learned Objective Weights

Figure 2a indicates that the baseline converges to near-uniform weights (0.34, 0.33, 0.33) because conflicting gradient signals from trade-off pairs cancel out, preventing the model from learning which objectives matter most. Pareto learns non-uniform weights (0.45, 0.30, 0.25), emphasizing pass rate. This makes sense: in Pareto-dominant pairs, correctness is typically the primary differentiator. The cleaner training signal allows Pareto to discover meaningful objective relationships.

Figure 2b shows training dynamics. Despite training on only 41.7% of pairs, Pareto achieves **lower validation loss** than baseline (0.09 vs 0.15 at convergence). Validation loss is the held-out

pairwise ranking loss (Eq. 1) computed on the validation split; the lower Pareto loss suggests better generalization, indicating the removed trade-off pairs contributed noise rather than signal.

4.5 Ablation: Effect of Task Difficulty

We analyze performance by task difficulty level:

Difficulty	Avg Pass Rate	Avg Cost	Avg Quality
Easy (8 tasks)	0.92 ± 0.02	0.90 ± 0.03	0.86 ± 0.04
Medium (8 tasks)	0.90 ± 0.03	0.85 ± 0.03	0.87 ± 0.05
Hard (8 tasks)	0.81 ± 0.02	0.79 ± 0.03	0.82 ± 0.08

This table shows expected patterns: harder tasks yield lower pass rates (0.81 vs 0.92) and cost efficiency (0.79 vs 0.90), reflecting increased problem complexity. Interestingly, quality remains relatively stable across difficulties (0.82-0.87), suggesting the model maintains consistent code structure regardless of problem complexity.

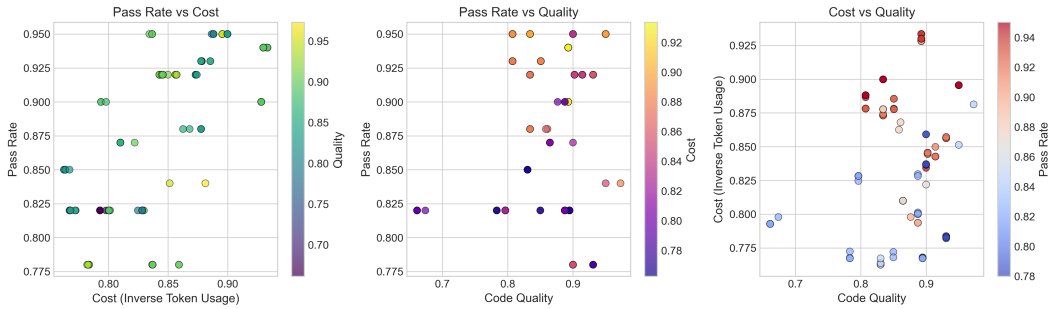


Figure 3: 2D projections of the objective space. Each point represents one generated output. Colors indicate the value of the third objective. The trade-off structure between objectives is visible.

The scatter in Figure 3 reveals trade-off relationships: outputs with higher pass rates tend to use more tokens (lower cost), and outputs with higher quality often require more verbose code. This visualization confirms that code generation inherently involves multi-objective trade-offs.

5 Conclusion and Discussion

We presented a framework for multi-objective optimization of compound AI systems using Pareto-preference learning. Our key contributions are:

1. Trade-off Prevalence: We empirically demonstrated that 58.3% of preference pairs in multi-objective code generation are trade-offs, not clear improvements. This finding has implications beyond our specific setting, in particular, any multi-objective preference learning task likely faces similar ambiguity.

2. Pareto Filtering: Our Pareto-preference loss provides a principled approach to filter ambiguous training signal. Using only 41.7% of pairs, Pareto LRF achieves 7.7% higher hypervolume than baseline, demonstrating that trade-off pairs contribute noise that actively hurts learning.

3. Efficient Optimization: The LRF architecture ($\sim$500K parameters) enables optimization without fine-tuning the 30B-parameter LLM, reducing computational requirements by six orders of magnitude.

Limitations. Our dataset is relatively small (120 samples), and the highly consistent outputs from Qwen3-Coder-30B-A3B-Instruct-FP8 limit reward variance.

Future Directions. (1) Scale to larger datasets with more output diversity; (2) Use execution-based correctness via Docker containers; (3) Explore settings where trade-offs are more pronounced, potentially showing larger Pareto advantages; (4) Extend to other compound AI tasks beyond code generation.

References

[1] Wu, S., Sarthi, P., Zhao, S., Lee, A., et al. OPTIMAS: Optimizing Compound AI Systems with Globally Aligned Local Rewards. *arXiv:2507.03041*, 2025.

[2] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[3] Zitzler, E. and Thiele, L. Multiobjective Evolutionary Algorithms: A Comparative Case Study. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.

[4] Ouyang, L., Wu, J., Jiang, X., et al. Training Language Models to Follow Instructions with Human Feedback. In *NeurIPS*, 2022.

[5] Rafailov, R., Sharma, A., Mitchell, E., et al. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. In *NeurIPS*, 2023.

[6] Yang, R., Pan, X., Luo, F., et al. ewards-in-Context: Multi-objective Alignment of Foundation Models with Dynamic Preference Adjustment. *arXiv:2402.10207*, 2024.

[7] Roijers, D. M., Vamplew, P., Whiteson, S., and Dazeley, R. A Survey of Multi-Objective Sequential Decision-Making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013.

[8] Christiano, P. F., Leike, J., Brown, T., et al. Deep Reinforcement Learning from Human Preferences. In *NeurIPS*, 2017.

[9] Jimenez, C. E., Yang, J., Wettig, A., et al. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? In *ICLR*, 2024.

[10] Zhuo, T. Y., Vu, M. C., Chim, J., et al. BigCodeBench: Benchmarking Code Generation with Diverse Function Calls and Complex Instructions. In *ICLR*, 2025.