TASK-DIRECTED REWARD SHAPING WITH WEIGHT-CONDITIONED POLICIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Reward shaping has been used successfully in large-scale reinforcement learning systems across a wide-range of domains. Yet poorly chosen reward weights can direct learning toward shaping signals at the expense of the task objective. Selecting these weights often requires manual iteration, repeated policy training, or bilevel optimization. We introduce Mix-Max Policy Search (MMPS), which takes a multi-objective reinforcement learning perspective to tune a fixed library of auxiliary rewards for a given task. MMPS trains a shared, weight-conditioned actor–critic across reward mixtures and uses its predicted primary-task value to select the mixture for each rollout. The selected mixture guides both on-policy data collection and the PPO update, sharing learning across weights while directing training toward behaviors predicted to yield high task return. Across nine Isaac-Gym tasks, MMPS achieves higher task performance than existing weight-tuning methods under matched environment-interaction budgets, with its largest gains on difficult manipulation tasks, achieving $2.26\times$ the score of policies trained with expert-tuned reward weights.

1 INTRODUCTION

Reward shaping helps train strong RL policies at scale, from Dota 2 to dexterous manipulation and legged locomotion (Berner et al., 2019; Andrychowicz et al., 2020; Rudin et al., 2022). It is especially useful when the task reward is sparse or delayed: distance and grasping rewards, for example, can guide a manipulation policy before it achieves its first success. But these signals can also draw the policy toward partial progress rather than task completion. We study environments with a fixed task reward and a supplied set of auxiliary reward components. Their weighted mixture guides training; the task reward alone determines performance.

Automated tuning poses two recurring challenges in existing methods outside of hyperparameter sweeps which require several training runs. Methods that evaluate reward candidates with separate learners divide interaction among them: Bayesian optimization trains a policy for each proposed weighting, ORSO allocates training among reward-specific learners, and population-based training maintains multiple agents even as it transfers parameters between them (Frazier, 2018; Zhang et al., 2025; Jaderberg et al., 2017). Methods that adapt a shaping rule within training instead must determine how a change in reward affects policy learning. BiPaRS approximates this dependence; MORSE shows how sparse task feedback can stall gradient-based reward search on poor mixtures and adds reward-space exploration and policy resets to escape them (Hu et al., 2020; Xie et al., 2025). This creates a clear opportunity: keep learning across reward mixtures in one policy with a simple, robust method to decide which mixture should generate the next rollout.

Multi-objective reinforcement learning (MORL) offers the machinery for sharing that learning: a policy conditioned on reward weights and a critic that estimates returns for the individual reward components (Roijers et al., 2013; Yang et al., 2019; Abels et al., 2019). In preference-conditioned RL, the weights usually specify the trade-off a policy should satisfy. Here, the objective is only to maximize the task reward. We use weights to control the behavior that collects training data and the reward mixture used to update it. The critic's primary-task value estimate then provides a common criterion for choosing among those behaviors.

We propose **Mix-Max Policy Search** (MMPS), a single-run method that combines a shared weight-conditioned PPO actor–critic with task-directed rollout selection. After an initial phase covering

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

different reward directions, MMPS samples candidate weights, including the task-only weight, and selects the one with the highest predicted primary-task value. It holds that weight fixed during the rollout and updates the shared policy using normalized component-wise advantages. Each rollout contributes to the same learner across mixtures. Weight selection requires neither a separate policy for each candidate nor differentiation through PPO updates, with no changes needed across all the tested environments.

We evaluate MMPS on nine modified IsaacGym continuous-control tasks spanning balance, locomotion, and dexterous manipulation, with a matched budget of 500 million environment steps per method. MMPS achieves an interquartile mean normalized terminal task score of 2.26, beating PPO using the original expert-tuned coefficients. The largest gains occur on sparse manipulation tasks, where choosing useful shaping signals is most consequential; differences are smaller on easier dense-reward tasks. These results show that sharing policy learning across mixtures and selecting rollouts by task value is an effective way to tune a supplied library of shaping rewards.

2 PRELIMINARIES

We consider a discounted decomposed-reward MDP, following the standard RL formalism (Sutton & Barto, 1998),

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \rho_0, \gamma, \mathbf{r}),$$

where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, P is the transition kernel, ρ_0 is the initial-state distribution, and $\gamma \in (0, 1)$ is the discount factor. The environment exposes a reward vector

$$\mathbf{r}(s, a) = \big(r^{(0)}(s, a), r^{(1)}(s, a), \ldots, r^{(m)}(s, a)\big)$$

, where $r^{(0)}$ is the task reward used for evaluation and $r^{(i)}$, $i \geq 1$, are auxiliary shaping rewards available during training (Ng et al., 1999). A shaping weight vector $\mathbf{w} \in \mathcal{W} \subseteq \mathbb{R}^{m+1}_{\geq 0}$ defines the scalar training reward

$$r_{\mathbf{w}}(s, a) = \mathbf{w}^\top \mathbf{r}(s, a) = \sum_{i=0}^{m} w_i r^{(i)}(s, a).$$

The difficulty is that optimizing $r_{\mathbf{w}}$ need not optimize the task reward: auxiliary terms may be mis-scaled, redundant, incomplete, or aligned with task progress only during part of training. This linear scalarization view matches the usual MORL setup, where reward vectors are combined by preference weights (Roijers et al., 2013; Hayes et al., 2022; Yang et al., 2019).

MMPS uses a weight-conditioned policy $\pi(a \mid s, \mathbf{w})$, following the broader idea of conditioning policies or value functions on goals, tasks, or reward preferences (Schaul et al., 2015; Barreto et al., 2017; Borsa et al., 2019). For each reward component, let

$$V_i^\pi(s, \mathbf{w}) = \mathbb{E}_{\pi(\cdot|\cdot,\mathbf{w})}\left[\sum_{t=0}^{\infty} \gamma^t r^{(i)}(s_t, a_t) \,\middle|\, s_0 = s\right], \qquad \mathbf{V}^\pi(s, \mathbf{w}) = \big(V_0^\pi(s, \mathbf{w}), \ldots, V_m^\pi(s, \mathbf{w})\big).$$

The task-value component $V_0^\pi(s, \mathbf{w})$ predicts task return for the policy conditioned on $\mathbf{w}$ and is used to choose the shaping weight that controls the next rollout. We defer additional formal objective details to Appendix A.

3 MULTI-OBJECTIVE RL-INSPIRED REWARD SHAPING

3.1 OVERVIEW

We introduce *Mix-Max Policy Search* (MMPS), which connects MORL-style weight-conditioned policies with online reward-shaping weight selection during training (Roijers et al., 2013; Hayes et al., 2022; Yang et al., 2019; Abels et al., 2019). Unlike standard MORL, where $\mathbf{w}$ is a test-time preference, here it is a training-time control variable. The evaluation objective is fixed to $r^{(0)}$. Rather than treating shaping weights as fixed hyperparameters and training a separate policy for each candidate weighting, we train a single weight-conditioned policy that represents a family of shaped behaviors:

$$\pi_\theta(a \mid s, \mathbf{w}).$$

Each weight vector $\mathbf{w}$ indexes a shaped behavior, so the shared policy represents an implicit ensemble. This conditioning follows the approach of universal value functions and successor-feature methods (Schaul et al., 2015; Barreto et al., 2017; Borsa et al., 2019). MMPS trains this family by selecting, at each rollout, the weight whose policy has the highest predicted task return. Reward weights thus act as rollout controls, directing data collection toward mixtures predicted to help the fixed task objective. Appendix A gives the formal weighted-objective view; the main method only needs the operational question of which rollout weight to use next.

3.2 Choosing rollout weights

At each PPO rollout, MMPS performs task-directed weight selection: it samples $K = 32$ candidate weights and adds the task-only weight $\mathbf{e}_0$ to the candidate set $\mathcal{C} = \{\mathbf{e}_0, \mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(K)}\}$. For the rollout's initial state s_0, it selects the weight with the highest predicted return under the fixed task reward:

$$\mathbf{w}^\star = \arg\max_{\mathbf{w}\in\mathcal{C}} V_{\phi,0}(s_0, \mathbf{w}),$$

The selector uses task value, not the scalarized value $\mathbf{w}^\top \mathbf{V}_\phi$, which favors weights that score well on their own shaped objectives. The task-only weight is available as a fallback, but need not be selected. Selection is greedy: critic error can lower actual task return, so MMPS offers no monotonic-improvement guarantee. It resamples candidates at every rollout, allowing nonlocal changes in the selected weight while continuing to train the same actor–critic. Appendix A analyzes the fixed-selector case.

3.3 Making reward weights comparable

Reward components can differ by orders of magnitude: a small coefficient on a large-scale auxiliary reward can dominate value learning and PPO updates, while a useful but low-magnitude task signal is washed out. Without scale control, weights compensate for reward units rather than encode relative emphasis.

MMPS therefore controls scale before using $\mathbf{w}$ as a training variable. Candidate weights are sampled uniformly from the positive ℓ_2 sphere, so $\mathbf{w}$ specifies a direction of emphasis rather than an absolute reward scale. The vector critic uses component-wise PopArt (adaptive value-target normalization that preserves predictions in the original scale), and the PPO actor update scalarizes per-component normalized advantages. Together, these choices make candidate mixtures comparable by predicted task usefulness and make shaped updates robust to differently scaled auxiliary terms.

Algorithm 1 Mix-Max Policy Search (MMPS)

1: Initialize a weight-conditioned actor $\pi_\theta(a \mid s, \mathbf{w})$ and vector critic $\mathbf{V}_\phi(s, \mathbf{w})$.
2: **for** each PPO iteration **do**
3: Sample candidate mixtures $\{\mathbf{w}^{(k)}\}_{k=1}^K$ and set $\mathcal{C} \leftarrow \{\mathbf{e}_0, \mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(K)}\}$.
4: Select rollout weights with the task-value head: $\mathbf{w}^\star \leftarrow \arg\max_{\mathbf{w}\in\mathcal{C}} V_{\phi,0}(s_0, \mathbf{w})$.
5: Collect on-policy rollouts using $\pi_\theta(\cdot \mid s, \mathbf{w}^\star)$.
6: Estimate component-wise returns and normalized advantages.
7: Update the vector critic and PPO actor using the selected mixtures.
8: **end for**

3.4 Mechanistic motivation: shaping weights as alignment controls

The task-value selector is a heuristic for choosing reward mixtures whose updates are likely to improve the fixed task objective. To motivate it, consider one update with the rollout selector held fixed. Theorems I.1 and J.1 in the Appendix establish local stability and improvement results, respectively.

Let g^μ_{mix} be the gradient of the selector-averaged shaped surrogate, g^μ_{task} the gradient of the induced task objective, and G_μ the Fisher matrix under the fixed rollout distribution. Under a local trust-region approximation, a shaped update improves the task objective to first order when

$$\left(g^\mu_{\text{task}}\right)^\top G_\mu^{-1} g^\mu_{\text{mix}} \geq 0.$$

This condition says that a useful reward mixture is not simply one with high shaped return; it is one whose update direction is aligned with task improvement. MMPS does not estimate this inner product directly. Instead, it uses $V_{\phi,0}(s, \mathbf{w})$ as a retrospective proxy: high predicted task value suggests that recent training under $\mathbf{w}$ has produced task-compatible behavior, so MMPS allocates the next rollout to that mixture.

4 Empirical Analyses

4.1 Experiment setup

We evaluate on nine modified IsaacGym (Makoviychuk et al., 2021) continuous-control benchmarks: `AllegroHand`, `Ant`, `Anymal`, `BallBalance`, `Cartpole`, `Humanoid`, `FrankaCabinet`, `ShadowHand`, and `Quadcopter`. The suite spans locomotion, balance, and dexterous-manipulation tasks, and each task exposes a default multi-term reward decomposition as $r_t = (r_t^{(0)}, \ldots, r_t^{(m)})$. Evaluation always uses the fixed task component $r^{(0)}$, while auxiliary components may be used for training. At evaluation, MMPS uses the same task-value selector at rollout start, holds the selected $\mathbf{w}$ fixed, and reports only $r^{(0)}$. We compare MMPS to Bayesian Optimization (BO), Population-Based Training (PBT), BiPaRS, and default-weight PPO using the same base PPO stack and a matched total environment-interaction budget. Appendix B gives the full baseline definitions, budget accounting, weight parameterizations, and evaluation protocol. The extra computation from online selection is small in this setting: environment stepping is the practical bottleneck, and in our profiling run candidate scoring and rollout inference account for roughly 4% of measured iteration time (Appendix Table 4).

4.2 Effectiveness: Can MMPS achieve higher task return?

We first test whether MMPS achieves higher task return. To compare environments with different reward scales, we normalize each method's terminal task return against random and default-weight PPO:

$$\mathrm{Score}(M) = \frac{R_M - R_{\mathrm{rand}}}{R_{\mathrm{default}} - R_{\mathrm{rand}}}. \tag{1}$$

Here, R_M is the terminal task return of method M; 0 corresponds to random-policy performance and 1 matches default-weight PPO. We report the interquartile mean of normalized scores and the relative win rate of MMPS against each baseline, following Agarwal et al. (2022). We use the completed runs available in the logged benchmark sweeps.

We estimate uncertainty in the aggregate normalized score by jointly bootstrapping terminal normalized scores across environments. Figure 1(a) reports the interquartile mean and a bootstrap 95% confidence interval (CI) for MMPS and the baselines. MMPS achieves higher aggregate normalized performance than the tuning baselines under the matched total interaction budget. Default-weight PPO has normalized score 1 by construction; the MMPS aggregate score is 2.26. These are normalized scores anchored by random-policy and default-weight PPO returns, not ratios of raw task returns. Figure 1(b) complements the aggregate score with pairwise win rates across environments. Figure 2 and Table 2 unpack this aggregate result into the per-task learning curves and terminal IQMs. Beyond improving the aggregate score, MMPS matches or exceeds the strongest tuning baseline on most tasks. Its largest margins occur on sparse manipulation tasks such as `FrankaCabinet`, `AllegroHand`, and `ShadowHand`, where auxiliary rewards are useful but difficult to weight. Table 1 gives the exact aggregate normalized IQM values used in Figure 1(a).

4.3 Mechanism: why does task-directed weight selection help?

The ablation compares two controls under the same interaction budget. *Random-candidate* keeps the shared weight-conditioned actor–critic but samples rollout weights uniformly. The *Split Networks* control retains task-directed weight selection but partitions weight space across independent actor–critics, each with separate parameters, optimizers, PPO updates, and PopArt statistics. These controls remove value-based rollout allocation and cross-weight sharing, respectively. Scores are normalized per environment: random policy = 0 and full MMPS = 1.

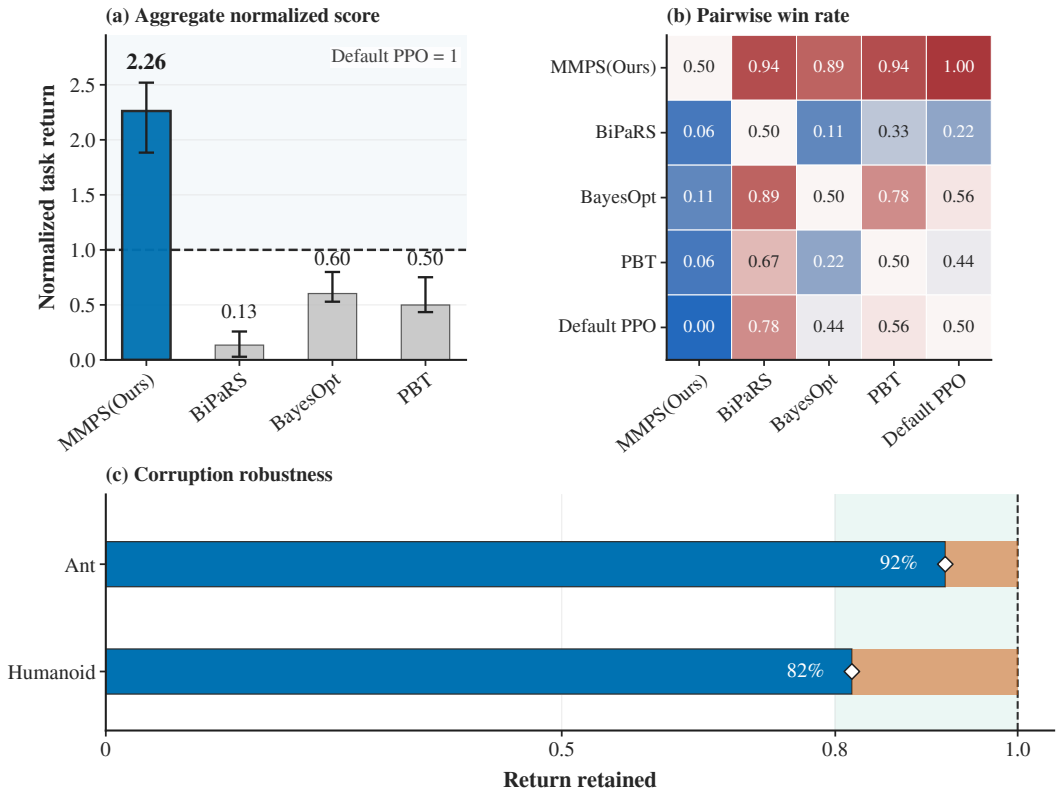

Figure 1: **Main empirical comparison across IsaacGym tasks.** **(a)** Aggregate terminal normalized score from Eq. 1, reported as the interquartile mean across environments with bootstrap 95% confidence intervals over seeds and tasks. **(b)** Pairwise win-rate matrix across nine environments, where each entry is the fraction of environments won by the row method against the column method (ties count as half a win); values above 0.5 favor the row method. **(c)** Robustness stress test under adversarially corrupted shaping terms on `Ant` and `Humanoid`, measured as corrupted-run task return divided by clean-run task return. Overall, MMPS achieves the strongest aggregate performance, wins most pairwise environment comparisons, and retains most task performance when auxiliary shaping terms are misleading.

Table 1: **Terminal normalized IQM scores on IsaacGym benchmarks.** Exact values for Figure 1(a). Each entry reports the interquartile mean (IQM) of terminal normalized scores from Eq. 1 and its bootstrap 95% confidence interval.

Method	IQM	95% CI
MMPS(Ours)	2.262	[1.884, 2.520]
BiPaRS	0.134	[0.028, 0.258]
BayesOpt	0.603	[0.529, 0.799]
PBT	0.499	[0.434, 0.751]

On the three dexterous-manipulation tasks, both controls fall well below full MMPS. This is consistent with value-based allocation directing rollouts toward mixtures predicted to improve the task and a shared learner transferring experience across mixtures. On easier tasks, some controls approach MMPS; the evidence therefore supports a regime-specific benefit, not universal superiority of either component.

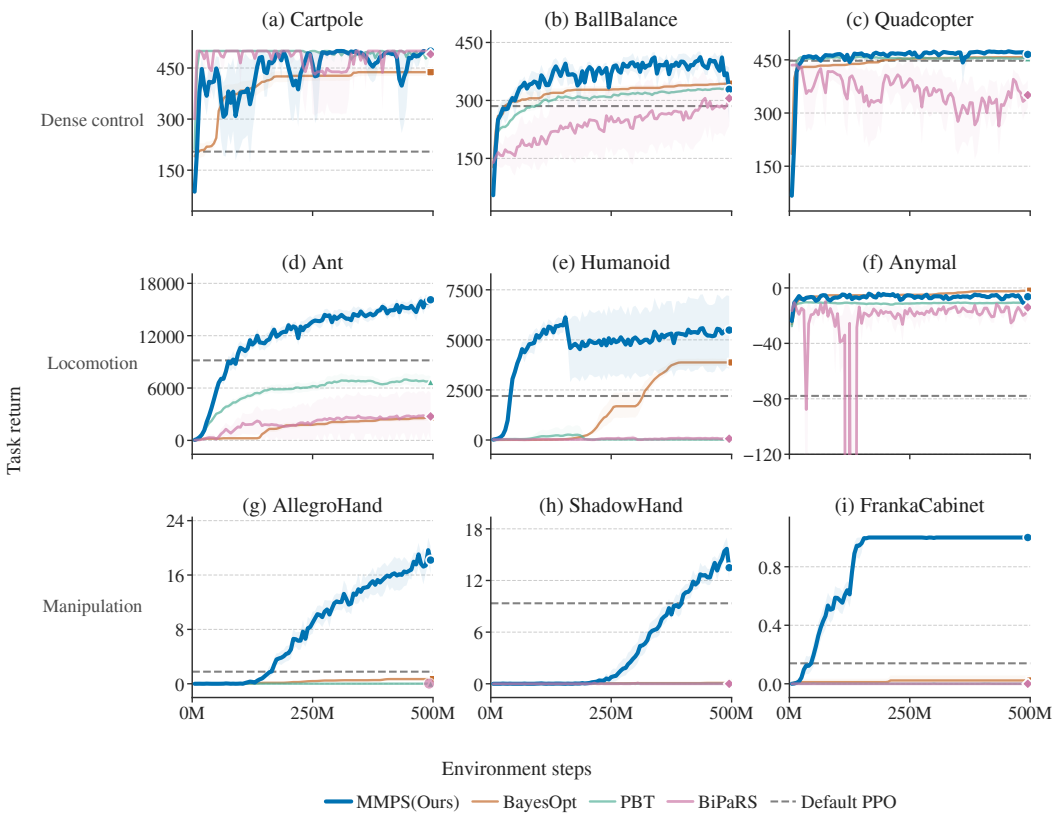

Figure 2: **Performance benchmarks across IsaacGym tasks.** Task return (component $r^{(0)}$) versus environment steps (mean $\pm$ 95% CI over seeds) for each benchmark. Rows group environments by task family and difficulty: dense-control tasks, locomotion tasks, and hard manipulation tasks. Colored end markers identify the final plotted checkpoint for each method (circle: MMPS; square: BayesOpt; triangle: PBT; diamond: BiPaRS). Overall, MMPS matches or exceeds the best tuning baselines on most tasks. The "Default-weight PPO" reference denotes PPO trained with the expert-tuned IsaacGym reward weights.

4.4 Robustness: Can MMPS ignore harmful shaping terms?

We test whether MMPS can avoid auxiliary rewards when they are actively misleading. In this stress test, we keep the task reward $r^{(0)}$ unchanged and negate each auxiliary component, $r^{(i)} \leftarrow -r^{(i)}$ for $i \geq 1$. Shaping terms that originally encouraged progress or stability therefore become incentives for the opposite behavior. A method that blindly optimizes the shaped scalarization should be pulled away from the task objective.

We run this experiment on `Ant` and `Humanoid`, where the task reward is relatively dense. These domains make the test interpretable because task-only learning remains a viable fallback: if the auxiliary rewards are harmful, a robust method should shift emphasis back toward $r^{(0)}$.

MMPS largely preserves task performance under this corruption. The task-value selector assigns lower value to mixtures dominated by the negated auxiliaries and shifts weight mass toward the task-only direction $\mathbf{e}_0$. Thus, MMPS can use its selection curriculum not only to exploit helpful shaping, but also to isolate and ignore harmful shaping when the task reward provides a reliable learning signal. Figure 1(c) reports the clean-to-corrupted performance ratio, quantifying how much task performance is retained.

Table 2: **Per-environment final task reward from logged IsaacGym runs.** For each environment and method, we report the interquartile mean (IQM) of final task reward and its bootstrap 95% confidence interval. Bold entries denote the highest IQM for that environment, with ties bolded together.

Environment	MMPS(Ours)	BiPaRS	BayesOpt	PBT
Easy / dense control				
Cartpole	**500.00** **[497.80, 500.00]**	**500.00** **[456.91, 500.00]**	438.50 [421.12, 454.77]	**500.00** **[500.00, 500.00]**
BallBalance	**345.98** **[160.42, 378.71]**	278.18 [127.72, 403.19]	343.43 [338.98, 346.25]	330.10 [326.56, 336.20]
Quadcopter	**470.25** **[448.85, 472.09]**	348.50 [143.41, 414.15]	458.27 [456.35, 461.09]	453.61 [452.74, 457.01]
Locomotion / medium				
Ant	**16481.88** **[15964.71, 16983.91]**	961.00 [0.75, 5415.80]	2788.11 [2215.78, 3177.48]	6527.56 [5859.46, 7855.44]
Humanoid	**6998.88** **[2354.27, 8018.97]**	48.08 [27.86, 110.40]	3915.74 [3711.13, 4048.75]	23.15 [13.30, 35.37]
Anymal	-7.52 [-8.42, -5.66]	-11.26 [-41.73, -3.81]	**-1.81** **[-2.08, -1.69]**	-10.69 [-11.84, -9.40]
Hard manipulation				
AllegroHand	**17.66** **[15.73, 20.91]**	0.02 [0.01, 0.03]	0.75 [0.53, 1.25]	0.01 [0.01, 0.02]
ShadowHand	**11.95** **[11.12, 15.13]**	0.00 [0.00, 0.00]	0.12 [0.12, 0.12]	0.01 [0.01, 0.02]
FrankaCabinet	**1.00** **[1.00, 1.00]**	0.00 [0.00, 0.00]	0.04 [0.00, 0.08]	0.00 [0.00, 0.00]

Table 3: **Mechanism ablations.** Normalized task-return scores (random policy = 0, full MMPS = 1; higher is better). Random-candidate selection removes predicted-value ranking while retaining the shared conditioned learner. The Split Networks control retains task-directed selection but removes cross-weight sharing.

Environment	Random-candidate	Split networks
Easy / dense control		
Cartpole	1.000	1.000
BallBalance	0.704	0.961
Quadcopter	0.901	0.939
Locomotion		
Ant	0.227	0.221
Humanoid	0.156	0.634
Anymal	0.089	0.646
Dexterous manipulation		
AllegroHand	0.032	0.131
ShadowHand	0.108	0.076
FrankaCabinet	0.403	0.472

4.5 What weights does the selector choose?

In `Cartpole` and `BallBalance`, selected weights retain substantial mass on shaping components throughout training. In `FrankaCabinet`, they place more mass on shaping early and shift toward the task-only direction as training progresses. Figure 3 shows these trajectories.

These trajectories are descriptive; they do not establish that shaping caused the performance differences or became harmful later.

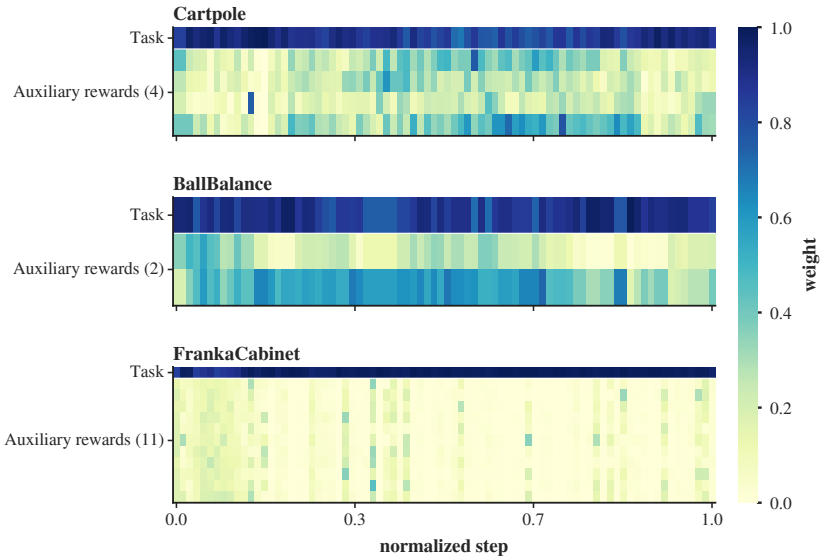

Figure 3: **Selected reward mixtures over training.** Each heatmap shows selected weights over normalized training progress; darker colors indicate larger weights.

5 Related Work

Reward shaping is a standard tool for supplying intermediate learning signals alongside a task objective. OpenAI Five combined its win objective with team-shared signals for intermediate events in Dota 2 (Berner et al., 2019); dexterous-hand systems used progress and subgoal signals for in-hand reorientation and Rubik's Cube manipulation (Andrychowicz et al., 2020; OpenAI et al., 2019); and ANYmal locomotion used a hand-designed mixture of command-tracking and stability-related terms (Rudin et al., 2022). Across these applications, reward components provide guidance toward a task, while their relative weights determine how that guidance shapes learning. Choosing those weights remains a practical challenge.

One approach is to treat the weights as hyperparameters and search over them outside the policy learner. Bayesian optimization proposes candidate weights, trains a policy for each, and uses the resulting task returns to guide subsequent proposals (Frazier, 2018). Population-based training (PBT) runs multiple agents concurrently and periodically copies parameters from stronger agents while perturbing their hyperparameters; when reward weights are included among those hyperparameters, PBT can discover time-varying weight schedules (Jaderberg et al., 2017). Both approaches use task performance to guide the search, but spend interaction evaluating candidate policies or maintaining a population.

Other methods adapt auxiliary rewards within the learning process. ASR evaluates policy-gradient directions associated with multiple auxiliary rewards and selects weights to guide each update, allowing the chosen signals to change over training (Fu et al., 2019). BiPaRS instead learns a state- and action-dependent shaping multiplier through a bilevel objective: the policy learns under the shaped reward, while an outer update adjusts the multiplier to improve task return (Hu et al., 2020). MORSE builds on bilevel weight optimization and adds exploration in reward-weight space, including resets to candidate mixtures when current optimization stalls (Xie et al., 2025). ORSO takes a candidate-selection approach: it treats a finite set of shaping rewards as online model-selection arms and allocates training interaction among their associated policy learners based on task performance (Zhang et al., 2025). These methods bring reward adaptation closer to training, while differing in whether they adjust weights for the current learner, optimize a shaping rule, or allocate interaction among separate candidates.

MORL offers a complementary idea: represent reward components as a vector and use scalarization weights to define different learning objectives. Prior work has used MORL to search for reward-component weights, including for balancing objectives in dialogue systems (Ultes et al., 2017). Other methods condition a policy on weights so one network can represent behaviors for multi-

ple reward trade-offs (Friedman & Fontaine, 2018; Abels et al., 2019; Reymond et al., 2022; Alegre et al., 2025). Much of this work uses the weight vector to express which trade-off a user wants the deployed policy to satisfy. MMPS uses the same general representation for a different training decision: its primary task objective is fixed, and the weight vector indexes mixtures that may help the policy learn that task.

MMPS combines these ideas differently. ASR selects weights from update directions, BiPaRS and MORSE optimize shaping weights through bilevel objectives, and ORSO allocates interaction among reward-specific learners. MMPS instead uses established weight conditioning to share one actor–critic across mixtures, while its fixed-task value head chooses the next rollout mixture. That mixture supplies the PPO signal, so data from different shaped behaviors update the shared learner while rollout budget favors behaviors predicted to perform well on the task. This avoids isolated candidate training and differentiation through PPO updates, but depends on accurate task-value estimates and a useful supplied reward decomposition.

6 Discussion

Where MMPS helps. MMPS gains most where the choice of shaping weights matters most. At a matched budget of 500 million environment steps, its normalized terminal-score IQM is 2.26, compared with 1.00 for default-weight PPO and 0.60 for Bayesian optimization. The gains are largest on sparse dexterous manipulation tasks and smaller on dense-control tasks, where the task reward already provides a useful learning signal. The ablations sharpen this picture: across the three manipulation tasks, random weight selection scores 0.03–0.40 and Split Networks scores 0.08–0.47, with full MMPS normalized to 1.00 on each task. Strong performance in this group depends on both task-directed rollout selection and a shared learner across weights. Because Split Networks also changes data coverage and optimization, it establishes the value of the shared-learner design.

Robustness and cost. With sign-flipped auxiliary rewards, MMPS retains 92% of its clean task return on Ant and 82% on Humanoid. The task-only candidate gives the selector a way to avoid harmful auxiliaries when the task reward supplies enough signal; the experiment demonstrates tolerance to this corruption without isolating the candidate's contribution. In a short profiling run, candidate scoring and rollout inference together account for about 4% of iteration time.

Scope and limitations. MMPS trades separate candidate learners and outer optimization through PPO for one conditioned learner selected by predicted task value. This avoids isolated searches but makes rollout allocation sensitive to critic error, especially before warmup covers the weight space. MMPS also assumes a supplied reward decomposition and nonnegative linear mixtures, which cannot represent arbitrary shaping rules.

AI use statement

Large language models (LLMs) supported literature review; manuscript editing and organization; and plot and table formatting. They were not used to generate synthetic data, develop the theoretical model or implementation, formulate mathematical claims or proofs, propose hypotheses, translate text, clean or reformat datasets, or conduct qualitative or thematic analysis. The authors reviewed the AI-assisted prose, citations, code, and rendered artifacts and take responsibility for the manuscript.

Reproducibility statement

Sections 3–4 and Appendices A and C describe the objective, method, implementation, and PPO update. Appendix B documents task metrics, reward components, baselines, budgets, and evaluation; Table 2 reports per-environment IQMs and confidence intervals. Appendices D–H detail compute, controls, ablations, architecture, and advantage estimation; Appendices I–J give the theoretical assumptions and derivations.

REFERENCES

Axel Abels, Diederik Roijers, Tom Lenaerts, Ann Nowé, and Denis Steckelmacher. Dynamic weights in multi-objective deep reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 11–20. PMLR, 2019.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice, 2022. URL https://arxiv.org/abs/2108.13264.

Lucas Alegre, Agon Serifi, Ruben Grandia, David Müller, Espen Knoop, and Moritz Bächer. AMOR: Adaptive character control through multi-objective reinforcement learning. In *ACM SIGGRAPH Conference Proceedings*, 2025.

Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Andre Barreto, Will Dabney, Remi Munos, Jonathan J. Hunt, Tom Schaul, Hado P. van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. In *Advances in Neural Information Processing Systems*, 2017.

Christopher Berner et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Diana Borsa, André Barreto, John Quan, Daniel J. Mankowitz, Hado van Hasselt, Rémi Munos, David Silver, and Tom Schaul. Universal successor features approximators. In *International Conference on Learning Representations*, 2019.

CleanRL Developers. Cleanrl license, 2026. URL https://github.com/vwxyzjn/cleanrl/blob/master/LICENSE. MIT License; accessed September 2026.

Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

Eli Friedman and Fred Fontaine. Generalizing across multi-objective reward functions in deep reinforcement learning. *arXiv preprint arXiv:1809.06364*, 2018.

Zhao-Yang Fu, De-Chuan Zhan, Xin-Chun Li, and Yi-Xing Lu. Automatic successive reinforcement learning with multiple auxiliary rewards. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pp. 2336–2342, 2019. doi: 10.24963/ijcai.2019/324.

Conor F. Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M. Zintgraf, Richard Dazeley, Fredrik Heintz, Enda Howley, Athirai A. Irissappane, Patrick Mannion, Ann Nowé, Gabriel Ramos, Marcello Restelli, Peter Vamplew, and Diederik M. Roijers. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems*, 36(1), April 2022. ISSN 1573-7454. doi: 10.1007/s10458-022-09552-y. URL http://dx.doi.org/10.1007/s10458-022-09552-y.

Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado Van Hasselt. Multi-task deep reinforcement learning with PopArt. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3796–3803, 2019.

Yujing Hu, Weixun Wang, Hangtian Jia, Yixiang Wang, Yingfeng Chen, Jianye Hao, Feng Wu, and Changjie Fan. Learning to utilize shaping rewards: A new approach of reward shaping. In *Advances in Neural Information Processing Systems*, volume 33, pp. 15931–15941, 2020.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G. M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks, 2017. URL https://arxiv.org/abs/1711.09846.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning. In *Advances in Neural Information Processing Systems*, volume 34, 2021. Datasets and Benchmarks Track.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pp. 278–287, 1999.

NVIDIA. Isaac gym download archive, 2026. URL https://developer.nvidia.com/isaac-gym/download. Includes the NVIDIA Isaac Gym License Agreement and terms of use; accessed September 2026.

OpenAI, Ilge Akkaya, Marcin Andrychowicz, et al. Solving rubik's cube with a robot hand, 2019.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Mathieu Reymond, Eugenio Bargiacchi, and Ann Nowe. Pareto conditioned networks. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, 2022.

Diederik M. Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013.

Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pp. 91–100. PMLR, 2022.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pp. 1312–1320. PMLR, 2015.

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

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1998. ISBN 0-262-19398-1. URL http://www.cs.ualberta.ca/%7Esutton/book/ebook/the-book.html.

Mikhail Terekhov and Caglar Gulcehre. In search for architectures and loss functions in multi-objective reinforcement learning, 2024. URL https://arxiv.org/abs/2407.16807.

Stefan Ultes, Pawel Budzianowski, Inigo Casanueva, Nikola Mrksic, Lina M. Rojas-Barahona, Pei-Hao Su, Tsung-Hsien Wen, Milica Gasic, and Steve Young. Reward-balancing for statistical spoken dialogue systems using multi-objective reinforcement learning. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pp. 65–70. Association for Computational Linguistics, 2017. doi: 10.18653/v1/W17-5509.

Yuqing Xie, Jiayu Chen, Wenhao Tang, Ya Zhang, Chao Yu, and Yu Wang. Automatic reward shaping from multi-objective human heuristics, 2025. URL https://arxiv.org/abs/2512.15120. Second Workshop on Aligning RL Experimentalists and Theorists (ARLET 2025).

Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. In *Advances in Neural Information Processing Systems*, 2019.

Chen Bo Calvin Zhang, Zhang-Wei Hong, Aldo Pacchiano, and Pulkit Agrawal. Orso: Accelerating reward design via online reward selection and policy optimization. In *International Conference on Learning Representations*, 2025.

A Formal weighted-objective view

For completeness, we spell out the formal objective underlying the rollout selection view used in the main text. Consider a discounted MDP

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \rho_0, \gamma, \mathbf{r}),$$

where the environment exposes reward components $\mathbf{r}(s,a) = (r^{(0)}(s,a), \ldots, r^{(m)}(s,a))$. The fixed evaluation objective is task return,

$$J_{\text{task}}(\pi) = \mathbb{E}_{s_0 \sim \rho_0,\, a_t \sim \pi(\cdot|s_t)} \left[\sum_{t=0}^{\infty} \gamma^t r^{(0)}(s_t, a_t) \right]. \tag{2}$$

A weight vector $\mathbf{w}$ defines the scalarized training reward $r_{\mathbf{w}}(s,a) = \mathbf{w}^\top \mathbf{r}(s,a)$ and the corresponding fixed-weight shaped objective

$$J^{\rho}_{\mathbf{w}}(\pi_{\mathbf{w}}) = \mathbb{E}_{s \sim \rho,\, a_t \sim \pi_{\mathbf{w}}(\cdot|s_t)} \left[\sum_{t=0}^{\infty} \gamma^t r_{\mathbf{w}}(s_t, a_t) \,\middle|\, s_0 = s \right], \tag{3}$$

where $\pi_{\mathbf{w}}(\cdot \mid s) = \pi(\cdot \mid s, \mathbf{w})$.

MMPS allows the rollout weight to depend on the rollout-start state. Let ρ be the rollout-start distribution and $p(\mathbf{w} \mid s)$ be the selector distribution over weights. The resulting weighted shaping objective is

$$J_{\rho,p}(\pi) = \mathbb{E}_{s \sim \rho,\, \mathbf{w} \sim p(\cdot|s)} \left[J^{\delta_s}_{\mathbf{w}}(\pi_{\mathbf{w}}) \right], \tag{4}$$

where δ_s denotes a point mass at the rollout-start state s. This is the formal version of selecting rollout weights online and then collecting on-policy data under $\pi_{\mathbf{w}}$.

Claim A.1 (Implicit ensemble view of the weighted shaping objective)**.** *The same objective can be viewed as training an implicit ensemble of shaped policies. The selector $p(\mathbf{w} \mid s)$ induces a marginal distribution over weights and, for each weight with $p(\mathbf{w}) > 0$, a corresponding restart distribution:*

$$p(\mathbf{w}) = \int \rho(s) p(\mathbf{w} \mid s)\, ds, \qquad \rho_{\mathbf{w}}(s) = \frac{\rho(s) p(\mathbf{w} \mid s)}{p(\mathbf{w})}.$$

Therefore,

$$J_{\rho,p}(\pi) = \mathbb{E}_{\mathbf{w} \sim p(\mathbf{w})} \left[J^{\rho_{\mathbf{w}}}_{\mathbf{w}}(\pi_{\mathbf{w}}) \right], \qquad J^{\rho_{\mathbf{w}}}_{\mathbf{w}}(\pi_{\mathbf{w}}) = \mathbb{E}_{s \sim \rho_{\mathbf{w}},\, a_t \sim \pi_{\mathbf{w}}(\cdot|s_t)} \left[\sum_{t=0}^{\infty} \gamma^t r_{\mathbf{w}}(s_t, a_t) \,\middle|\, s_0 = s \right]. \tag{5}$$

B Experimental Details

Task metrics and reward components. For `ShadowHand` and `AllegroHand`, the first reward component indicates goal reset or success. Their auxiliary components encode distance to the goal, orientation alignment, action penalty, success bonus, fall penalty, and timeout penalty. We use the modified IsaacGym reward functions in `shadow_hand_mod.py` and `allegro_hand_mod.py`. For `FrankaCabinet`, the task component is the binary drawer-open success indicator (drawer joint position greater than 0.39). Its auxiliary components encode gripper-to-handle distance and orientation, finger placement, drawer-opening progress, finger distance, action and style penalties, and staged opening bonuses; the modified function is in `franka_cabinet_mod.py`. These metrics separate hand-task success from the dense progress and regularization signals used for shaping. For locomotion and balance environments, we use task metrics such as progress or distance-to-goal that reflect the high-level task objective. For training, each environment returns the default IsaacGym reward terms as a vector $r_t = (r_t^{(0)}, \ldots, r_t^{(m)})$; $r^{(0)}$ is the task component used for evaluation, and $r^{(i)}$ for $i \geq 1$ are auxiliary shaping terms.

Baselines. Bayesian Optimization (BO) (Frazier, 2018) treats the reward-weight vector $\mathbf{w}$ as a static hyperparameter and selects $\mathbf{w}$ based on final task return after training a PPO agent. Population-Based Training (PBT) (Jaderberg et al., 2017) trains a population of PPO agents, each with its own weight vector, and periodically exploits and mutates weights based on task performance. BiPaRS (Hu et al., 2020) is included as a single-run bi-level reward-shaping baseline that adapts the use of shaping rewards during training. Default-weight PPO uses the expert-tuned IsaacGym reward weights. These are the original coefficients supplied by the IsaacGym task authors. We do not treat language-model reward-design systems as direct baselines because they solve a different problem–generating or rewriting reward functions–whereas all methods in this benchmark receive the same fixed reward-component library and may only select how those components are weighted. We first ran pilot sweeps for the baseline methods on all environments, with up to 50 million environment steps per candidate configuration. The sweeps tuned baseline hyperparameters and reward normalization; for each baseline, we selected the configuration with the highest aggregate normalized score from Eq. 1.

Interaction budget. All methods use the same CleanRL-style PPO implementation. MMPS and default-weight PPO use the same fixed PPO profile, while BO and PBT use the profiles selected by the pilot sweeps. No method is retuned per environment. Each tuning procedure receives a total budget of $B = 500$ million environment steps. For BO, each trial trains for $B_{\text{trial}} = 393{,}216$ environment steps, and we cap the number of trials N such that

$$N\, B_{\text{trial}} \le B. \tag{6}$$

The selected BO profile uses positive-ℓ_2-sphere weights, globally normalized scalarized rewards, a PopArt critic, 512 environments, rollout horizon 16, learning rate 10^{-3}, PPO clip coefficient 0.30, target KL 0.10, 2 minibatches, and 2 update epochs. For PBT, we use a population of $P = 16$ agents, perform truncation-based exploit/explore every 8 PPO iterations, and account for the total interaction budget across the full population:

$$\sum_{i=1}^{P} B_i \le B. \tag{7}$$

The selected PBT profile uses positive-ℓ_2-sphere weights, raw scalarized rewards, no value normalization, 2048 total environments, rollout horizon 16, learning rate 3×10^{-4}, PPO clip coefficient 0.20, entropy coefficient 0, value-loss coefficient 1, 2 minibatches, and 2 update epochs. Learning curves are plotted against total environment steps consumed by the full tuning procedure.

Weight parameterization. MMPS selects nonnegative reward weights $w \in \mathbb{R}^{m+1}_{\ge 0}$. Because the actor update scalarizes per-component normalized advantages (Eq. 11–12), the main degree of freedom is the direction of $\mathbf{w}$, so we parameterize MMPS weights on the positive ℓ_2 sphere:

$$\mathcal{W} = \left\{ \mathbf{w} \in \mathbb{R}^{m+1}_{\ge 0} : \ \|\mathbf{w}\|_2 = 1 \right\}. \tag{8}$$

Both baselines search over log-space parameters, which are exponentiated and normalized to the positive ℓ_2 sphere in the selected profiles above. BO evaluates each resulting vector as a trial; PBT assigns vectors to population members and mutates them during training.

Evaluation protocol. All reported performance uses terminal task return under the matched interaction budget. For MMPS, the reported rollout policy is the learned conditioned actor queried with the online selector used during data collection: at the start of each rollout, the critic selects $\mathbf{w}$ from the candidate set using Section 3.2, and the actor executes $\pi(\cdot \mid s, \mathbf{w})$ with that $\mathbf{w}$ held fixed for the rollout. The return plotted for MMPS is always the task component $r^{(0)}$ accumulated during these selected-weight rollouts. Aggregate statistics use environments with completed data and finite normalization constants for each comparison.

Existing assets and licenses. The benchmark environments use Isaac Gym (Makoviychuk et al., 2021), which is distributed by NVIDIA under the NVIDIA Isaac Gym License Agreement and terms of use (NVIDIA, 2026). Our PPO training stack follows CleanRL (Huang et al., 2022), which is released under the MIT License (CleanRL Developers, 2026). We cite the original papers for Bayesian optimization, population-based training, and BiPaRS, but implement these baselines ourselves; we redistribute no external baseline code or datasets.

C MMPS IMPLEMENTATION DETAILS

C.1 COMPONENT-WISE VALUES

For each reward component $i = 0, \ldots, m$, define its value under the weight-conditioned policy as

$$V_i^{\pi}(s, \mathbf{w}) = \mathbb{E}_{\pi_{\mathbf{w}}} \left[\sum_{t=0}^{\infty} \gamma^t r^{(i)}(s_t, a_t) \,\middle|\, s_0 = s \right]. \tag{9}$$

The corresponding vector value is

$$\mathbf{V}^{\pi}(s, \mathbf{w}) = \big(V_0^{\pi}(s, \mathbf{w}), \ldots, V_m^{\pi}(s, \mathbf{w})\big). \tag{10}$$

The first component, $V_0^{\pi}(s, \mathbf{w})$, estimates the task return of the policy induced by $\mathbf{w}$. MMPS uses this task-value component to choose which shaping weights should control data collection.

C.2 WEIGHT-CONDITIONED ACTOR–CRITIC

MMPS trains a policy conditioned on both the state and shaping weight:

$$\pi_\theta(a \mid s, \mathbf{w}),$$

where $\mathbf{w} \in \mathcal{W} \subseteq \mathbb{R}^{m+1}$ is a feasible reward-weight vector. This policy represents a family of behaviors indexed by the training mixture. Unlike outer-loop reward tuning, which trains a separate policy for each candidate $\mathbf{w}$, MMPS amortizes learning across mixtures through shared parameters.

The critic predicts component-wise values:

$$\mathbf{V}_\phi(s, \mathbf{w}) = \big(V_{\phi,0}(s, \mathbf{w}), \ldots, V_{\phi,m}(s, \mathbf{w})\big),$$

where each head estimates the discounted return of one reward component under the policy conditioned on $\mathbf{w}$:

$$V_{\phi,i}(s, \mathbf{w}) \approx \mathbb{E}_{\pi_\theta(\cdot|\cdot,\mathbf{w})} \left[\sum_{t=0}^{\infty} \gamma^t r^{(i)}(s_t, a_t) \,\middle|\, s_0 = s \right].$$

The task head $V_{\phi,0}$ estimates return under the evaluation reward. The remaining heads estimate auxiliary-component returns and are used to compute component-wise advantages.

Given trajectories collected under rollout weights $\mathbf{w}_t$, the critic is trained with component-wise bootstrapped targets. For notation, a one-step target is

$$\mathbf{y}_t = \mathbf{r}_t + \gamma \mathbf{V}_\phi(s_{t+1}, \mathbf{w}_t).$$

We use the corresponding TD(λ) return for each component in implementation; Appendix H details this estimate. The critic loss is

$$\mathcal{L}_V(\phi) = \mathbb{E}_t \left[\|\mathbf{V}_\phi(s_t, \mathbf{w}_t) - \mathbf{y}_t\|_2^2 \right].$$

We condition both actor and critic on $\mathbf{w}$ using feature-wise modulation (Perez et al., 2018). The role of conditioning is to share representations across related shaping weights while allowing the policy $\pi_\theta(\cdot \mid s, \mathbf{w})$ to specialize as the training mixture changes. The weight vector is therefore a training context, not a test-time preference that defines the evaluation objective through its scalarized return.

C.3 COMPONENT-WISE VALUE NORMALIZATION

The vector critic must jointly learn heads for reward components whose returns can differ substantially in scale. If trained directly on raw targets, large-magnitude auxiliary components can dominate the critic loss, while sparse or low-magnitude components can be learned slowly. This is particularly important in MMPS because the task-value head is used for rollout selection, and all heads are used to construct component-wise advantages.

MMPS therefore normalizes value targets independently for each reward component. For component i, we maintain running statistics μ_i and σ_i of the corresponding value target and train a normalized prediction $\widehat{V}_{\phi,i}$. The unnormalized value is recovered as

$$V_{\phi,i}(s, \mathbf{w}) = \sigma_i \widehat{V}_{\phi,i}(s, \mathbf{w}) + \mu_i.$$

The normalized target is

$$\widehat{y}_{t,i} = \frac{y_{t,i} - \mu_i}{\sigma_i + \epsilon},$$

and the critic is optimized on normalized targets component-wise.

Our implementation uses PopArt for this normalization (Hessel et al., 2019). When the running statistics change, PopArt rescales the final value layer so that the unnormalized predictions $V_{\phi,i}(s, \mathbf{w})$ are preserved. This lets the critic train on normalized targets while still producing unnormalized values for return prediction, advantage computation, and task-value rollout selection.

C.4 Scale-controlled multi-component PPO update

The actor is updated with PPO (Schulman et al., 2017) using trajectories collected under selected rollout weights. For each transition, MMPS computes component-wise advantages

$$\mathbf{A}_t = \big(A_{t,0}, \ldots, A_{t,m}\big),$$

where $A_{t,i}$ is computed from the corresponding value head $V_{\phi,i}$.

A direct scalarization $\mathbf{w}_t^\top \mathbf{A}_t$ is sensitive to reward scale. If one reward component has much larger advantage variance than the others, it can dominate the actor update even when its assigned weight is small. In that case, the selected weight vector no longer has a stable interpretation as relative emphasis across components.

MMPS instead normalizes advantages independently for each reward component within each PPO minibatch:

$$\widetilde{A}_{t,i} = \frac{A_{t,i} - \mu_i^A}{\sigma_i^A + \epsilon}, \tag{11}$$

where μ_i^A and σ_i^A are the minibatch mean and standard deviation of component i. The scalar advantage used in the PPO objective is then

$$A_t^{\mathbf{w}} = \mathbf{w}_t^\top \widetilde{\mathbf{A}}_t. \tag{12}$$

The policy is optimized with the standard clipped PPO surrogate:

$$\mathcal{L}_\pi(\theta) = \mathbb{E}_t\Big[\min\big(\rho_t(\theta) A_t^{\mathbf{w}}, \\ \mathrm{clip}\big(\rho_t(\theta), 1 - \epsilon_{\mathrm{clip}}, 1 + \epsilon_{\mathrm{clip}}\big) A_t^{\mathbf{w}}\big)\Big], \tag{13}$$

where

$$\rho_t(\theta) = \frac{\pi_\theta(a_t \mid s_t, \mathbf{w}_t)}{\pi_{\theta_{\mathrm{old}}}(a_t \mid s_t, \mathbf{w}_t)}.$$

Thus, MMPS does not apply the rollout weight as an unnormalized scalar reward coefficient in the actor update. The weight vector scalarizes normalized component-wise learning signals, representing relative emphasis among policy-gradient directions associated with the reward components.

C.5 Task-value rollout allocation

MMPS chooses the rollout weight $\mathbf{w}$ using predicted task value. Uniformly sampling shaping weights can spend interaction budget on mixtures that induce behavior unrelated to the task objective. Selecting weights by scalarized shaped value,

$$\mathbf{w}^\top \mathbf{V}_\phi(s, \mathbf{w}),$$

would instead favor mixtures that are rewarding under their own training signal, which need not correspond to high task return.

At the start of each rollout, the method samples K candidate shaping weights

$$\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(K)} \sim \rho_{\mathcal{W}},$$

Table 4: **MMPS runtime profile on AntMod.** Wall-clock breakdown from a short benchmark-scale profiling run.

Component	Total time (s)	Share (%)	ms / iter
Weight selection	0.084	1.6	16.80
Rollout inference	0.127	2.4	25.46
Env step	1.409	26.6	281.79
GAE	0.016	0.3	3.22
PopArt	0.005	0.1	1.06
PPO update	0.330	6.2	65.97
Logging	2.708	51.2	541.67
Other	0.609	11.5	121.78

from a candidate distribution over feasible weights. In our experiments, $\rho_{\mathcal{W}}$ is uniform over the positive ℓ_2 sphere. MMPS also includes the task-only mixture

$$\mathbf{e}_0 = (1, 0, \ldots, 0)$$

in the candidate set. Given the rollout initial state s_0, the selected weight is

$$\mathbf{w}^\star = \arg \max_{\mathbf{w} \in \{\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(K)}, \mathbf{e}_0\}} V_{\phi,0}(s_0, \mathbf{w}).$$

The selected mixture $\mathbf{w}^\star$ is held fixed for the rollout. The resulting data are then used to update the shared conditioned actor–critic.

This selection rule allocates rollout budget according to the critic's estimate of the fixed evaluation objective. Auxiliary rewards can influence the actor update through the selected mixture, but the criterion for selecting that mixture is predicted task return. The task-only candidate provides a direct fallback when auxiliary shaping is not predicted to improve task-directed behavior.

Before the task-value head is reliable across shaping weights, a short coverage phase distributes rollouts across reward components. For rollouts assigned to component i, the selector uses $V_{\phi,i}(s_0, \mathbf{w})$ instead of $V_{\phi,0}(s_0, \mathbf{w})$. This phase seeds experience across reward directions; afterward, all rollouts use task-value selection.

D Compute profile

MMPS and default-weight PPO share one fixed CleanRL-style PPO profile (Huang et al., 2022). BO and PBT use their globally selected pilot-sweep profiles (Appendix B); no method is retuned per environment. The shared MMPS/default PPO profile uses Adam, two-hidden-layer Tanh MLPs with width 256, learning rate 2.6×10^{-3}, 16,384 parallel environments, rollout horizon 16, discount $\gamma = 0.99$, GAE parameter $\lambda = 0.95$, 4 minibatches, 4 update epochs, PPO clip coefficient 0.20, entropy coefficient 0, value-loss coefficient 2, target KL 0.10, and reward scaler 1. MMPS uses $K = 32$ sampled candidates plus the task-only candidate, FiLM conditioning with a 128-dimensional context, PopArt with update rate 0.01, per-component normalized advantages, and warm-start coverage for the first 25% of training.

D.1 Computational resources and feasibility

MMPS increases the size of a single training run because it conditions the actor and critic on reward weights and predicts multiple value heads. Across the IsaacGym tasks, the FiLM-conditioned MMPS agent has approximately 0.81M–1.12M trainable parameters, depending on the number of reward heads; the corresponding default PPO baselines have approximately 0.13M–0.25M parameters. Table 4 reports a short AntMod profiling run at benchmark scale with 32768 environments, five measured iterations, one H100 GPU, eight CPUs, and 64GB RAM. Weight selection and rollout inference account for roughly 4% of measured iteration time; environment stepping, PPO updates, and logging dominate the wall-clock profile.

Table 5: **Task-only vector-critic control.** Normalized performance relative to MMPS. This control keeps the multi-headed critic but removes shaping-based actor updates.

Environment	MMPS(Ours)	Task-only vector critic
AllegroHand	1.000	0.319
Ant	1.000	0.650
Anymal	1.000	0.586
BallBalance	1.000	0.452
Cartpole	1.000	0.897
FrankaCabinet	1.000	0.000
Humanoid	1.000	**1.272**
Quadcopter	1.000	0.370
ShadowHand	1.000	0.242

Table 6: **MMPS component ablations.** Scores are normalized with MMPS set to 1.0 per environment and the random-policy baseline set to 0.0. Values above 1.0 indicate an ablation outperforming full MMPS on that environment.

Environment	MMPS(Ours)	No PopArt	No adv. norm	No warm start
AllegroHand	1.000	0.321	0.000	0.536
Ant	1.000	0.760	1.024	**1.349**
Anymal	1.000	0.878	0.960	0.996
BallBalance	1.000	**1.508**	1.337	1.037
Cartpole	1.000	**1.155**	0.685	1.000
FrankaCabinet	1.000	1.000	0.000	0.000
Humanoid	1.000	0.635	0.332	**1.203**
ShadowHand	1.000	0.000	0.000	0.312

E Representation and selection controls

To isolate the effect of the dynamic shaping curriculum from that of the decomposed vector critic, we run a task-only control. It keeps the multi-headed critic trained on all reward components but trains the actor only with the task reward $r^{(0)}$, preserving vector-value representation learning while removing shaping-based actor updates. Table 5 reports normalized scores with MMPS set to 1.0 per environment and the random policy set to 0.0. MMPS outperforms this control on eight of nine environments, with the largest gaps on sparse or difficult manipulation tasks; Humanoid is the exception, indicating that the shaping curriculum can be unnecessary or detrimental when the task reward is already dense enough.

F Component ablations

Table 6 ablates PopArt, per-component advantage normalization, and the warm-start phase. These components are not uniformly beneficial: on easier dense-reward tasks, removing one can match or exceed the full method because the task signal is already informative and extra normalization or coverage can add variance. On sparse or poorly scaled tasks such as AllegroHand, FrankaCabinet, and ShadowHand, the full method is more reliable and ablations often collapse. We therefore view these components as robustness mechanisms for settings where reward shaping is most needed, not as universal improvements.

G Architecture ablation

Table 7 compares the default FiLM conditioning architecture against alternatives for multi-objective or context-conditioned networks, including concatenation-based, merge-based, multi-body, and hy-

Table 7: **IsaacGym architecture ablation for weight conditioning.** Scores are normalized with FiLM set to 1.0 per environment and the random-policy baseline set to 0.0.

Environment	FiLM	Concat	Merge	Multi-body	Hyper
AllegroHand	**1.000**	0.865	0.075	0.139	0.016
Ant	**1.000**	0.581	0.590	0.395	0.245
Anymal	**1.000**	0.970	0.959	0.919	0.983
BallBalance	**1.000**	0.957	0.671	0.852	0.817
Cartpole	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
FrankaCabinet	**1.000**	0.350	0.020	0.000	0.520
Humanoid	1.000	1.155	**1.849**	0.115	0.277
Quadcopter	**1.000**	**1.000**	0.981	0.989	0.993
ShadowHand	**1.000**	0.009	0.008	0.005	0.010

pernetwork conditioning (Terekhov & Gulcehre, 2024). Each environment-level entry reports a normalized score using the random-policy baseline as the lower anchor and FiLM as the reference, so the FiLM column is 1.0 by construction. Overall, FiLM is the most reliable default across tasks; several alternatives are competitive on dense/easy tasks, but they underperform on the sparse manipulation tasks where representation sharing and stable conditioning matter most.

H TD(λ) ADVANTAGE ESTIMATES

For each component $i \in \{0, \dots, m\}$, consider the TD(0) advantage $\delta_t^{(i)}$ computed using the corresponding value head:

$$\delta_t^{(i)} = r_t^{(i)} + \gamma(1 - d_t)\, V_{\phi,i}(s_{t+1}, \mathbf{w}_t) - V_{\phi,i}(s_t, \mathbf{w}_t) \tag{14}$$

We then construct the per-component TD(λ) advantage estimate and critic target as

$$\hat{A}_t^{(i)} = \delta_t^{(i)} + \gamma\lambda(1 - d_t)\hat{A}_{t+1}^{(i)}, \qquad y_{t,i}^{(\lambda)} = \hat{A}_t^{(i)} + V_{\phi,i}(s_t, \mathbf{w}_t), \tag{15}$$

where $\mathbf{w}_t$ is the stored weight that generated transition t, and d_t indicates termination, which stops bootstrapping.

I STABILITY OF GREEDY WEIGHT RESELECTION

At the start of each rollout, MMPS samples K candidate weight vectors and selects the one with the highest predicted task value under the current weight-conditioned critic. The results in this appendix are deliberately limited. They do not establish convergence, optimality, monotonic improvement, or soundness of the full nonstationary MMPS algorithm with a learned selector. Instead, they isolate two local mechanisms: when a sampled greedy selector is stable under value perturbations, and what a policy step optimizes when the selector is treated as fixed. The theorem below formalizes the first mechanism.

Fix a state $s \in \mathcal{S}$. Let $\rho_{\mathcal{W}}$ denote the candidate-weight sampling distribution over $\mathcal{W}$, and sample

$$\mathbf{w}_1, \dots, \mathbf{w}_K \overset{\text{iid}}{\sim} \rho_{\mathcal{W}}.$$

Define the greedy selector

$$I_\phi(s) := \arg\max_{j \in \{1,\dots,K\}} V_{\phi,0}(s, \mathbf{w}_j),$$

where ties are broken by any fixed deterministic rule. For any measurable subset $A \subseteq \mathcal{W}$, define the induced distribution over selected weights by

$$\mu_\phi(A \mid s) := \mathbb{P}\big(\mathbf{w}_{I_\phi(s)} \in A\big)\,.$$

Theorem I.1 (Stability of greedy weight reselection)**.** *Fix $s \in \mathcal{S}$, and couple the selectors at critic parameters ϕ and ϕ' using the same sampled candidates $\mathbf{w}_{1:K}$. Define the task-value perturbation*

$$\varepsilon_s := \sup_{\mathbf{w} \in \mathcal{W}} \left| V_{\phi',0}(s, \mathbf{w}) - V_{\phi,0}(s, \mathbf{w}) \right|,$$

and the empirical top-two margin under ϕ,

$$\Delta_\phi^{(K)}(s; \mathbf{w}_{1:K}) := \max_{j \in \{1,\dots,K\}} V_{\phi,0}(s, \mathbf{w}_j) - \max_{j \neq I_\phi(s)} V_{\phi,0}(s, \mathbf{w}_j).$$

Then

$$\left\| \mu_{\phi'}(\cdot \mid s) - \mu_\phi(\cdot \mid s) \right\|_{\mathrm{TV}} \leq \mathbb{P}\Big(\Delta_\phi^{(K)}(s; \mathbf{w}_{1:K}) \leq 2\varepsilon_s \Big).$$

In particular, if

$$\Delta_\phi^{(K)}(s; \mathbf{w}_{1:K}) > 2\varepsilon_s,$$

then the maximizing candidate does not change, i.e.

$$I_{\phi'}(s) = I_\phi(s).$$

Proof. Couple both selectors using the same candidate set $\mathbf{w}_{1:K}$. By the coupling characterization of total variation distance,

$$\left\| \mu_{\phi'}(\cdot \mid s) - \mu_\phi(\cdot \mid s) \right\|_{\mathrm{TV}} \leq \mathbb{P}(I_{\phi'}(s) \neq I_\phi(s)).$$

It therefore suffices to bound the probability that the maximizing index changes.

Let $i^\star := I_\phi(s)$ be the winner under ϕ, and let $j^\star$ be the second-best index under ϕ. If the winner changes under ϕ', then some competitor must overtake $i^\star$. Since every task-value score changes by at most ε_s,

$$V_{\phi',0}(s, \mathbf{w}_{i^\star}) \geq V_{\phi,0}(s, \mathbf{w}_{i^\star}) - \varepsilon_s, \qquad V_{\phi',0}(s, \mathbf{w}_j) \leq V_{\phi,0}(s, \mathbf{w}_j) + \varepsilon_s \quad \forall j.$$

Hence a change of winner can occur only if

$$V_{\phi,0}(s, \mathbf{w}_{i^\star}) - \varepsilon_s \leq V_{\phi,0}(s, \mathbf{w}_{j^\star}) + \varepsilon_s,$$

that is,

$$\Delta_\phi^{(K)}(s; \mathbf{w}_{1:K}) \leq 2\varepsilon_s.$$

Therefore

$$\mathbb{P}(I_{\phi'}(s) \neq I_\phi(s)) \leq \mathbb{P}\Big(\Delta_\phi^{(K)}(s; \mathbf{w}_{1:K}) \leq 2\varepsilon_s \Big),$$

which proves the claim. □

Corollary I.2 (Anti-concentration form)**.** *Assume that for a fixed state s there exists $C_s > 0$ such that for all sufficiently small $\tau > 0$,*

$$\mathbb{P}\Big(\Delta_\phi^{(K)}(s; \mathbf{w}_{1:K}) \leq \tau \Big) \leq C_s \tau.$$

Then

$$\left\| \mu_{\phi'}(\cdot \mid s) - \mu_\phi(\cdot \mid s) \right\|_{\mathrm{TV}} \leq 2C_s\varepsilon_s.$$

Proof. Apply Theorem I.1 with $\tau = 2\varepsilon_s$. □

Remark I.3. Theorem I.1 isolates the extra distribution shift introduced by online greedy weight selection in MMPS. The selector changes only when the best and second-best sampled candidates are close under the task-value head $V_{\phi,0}$. Thus the induced distribution over conditioning weights remains stable whenever the task-value predictions change little over an update and the sampled winner margin is nontrivial.

J TRUST-REGION IMPROVEMENT WITH A FROZEN SELECTOR

The previous theorem controls the extra distribution shift created when the greedy selector changes across updates. We now consider one trust-region step with the selector frozen. In this case, MMPS reduces to a standard trust-region update on the composite policy obtained by averaging over the fixed state-dependent selector.

Fix a selector $\mu(\cdot \mid s)$, treated as constant during one policy update, and define the induced composite policy

$$\bar{\pi}_\theta^\mu(a \mid s) := \int_{\mathcal{W}} \pi_\theta(a \mid s, \mathbf{w})\, \mu(d\mathbf{w} \mid s).$$

Let

$$J_{\text{task}}^\mu(\theta) := \mathbb{E}_{\bar{\pi}_\theta^\mu}\left[\sum_{t\geq 0} \gamma^t \mathbf{r}^{(0)}(s_t, a_t, s_{t+1})\right]$$

denote the task-return objective under the frozen selector. Define the selector-averaged PPO surrogate

$$L_{\text{mix}}^\mu(\theta'; \theta) := \mathbb{E}_{\substack{s\sim d_\theta^\mu,\\ \mathbf{w}\sim\mu(\cdot|s),\\ a\sim\pi_\theta(\cdot|s,\mathbf{w})}}\left[\frac{\pi_{\theta'}(a \mid s, \mathbf{w})}{\pi_\theta(a \mid s, \mathbf{w})} \hat{A}_\theta(s, a, \mathbf{w})\right],$$

where d_θ^μ is the discounted state distribution induced by $\bar{\pi}_\theta^\mu$, and $\hat{A}_\theta(s, a, \mathbf{w})$ is the scalarized advantage used by MMPS for the conditioning weight $\mathbf{w}$.

Let

$$g_{\text{mix}}^\mu := \nabla_{\theta'} L_{\text{mix}}^\mu(\theta'; \theta)\big|_{\theta'=\theta}, \qquad g_{\text{task}}^\mu := \nabla_\theta J_{\text{task}}^\mu(\theta),$$

and let G_μ be the Fisher information matrix of the composite policy family $\bar{\pi}_\theta^\mu$.

Theorem J.1 (Trust-region improvement with a frozen selector)**.** *Assume the selector μ is frozen during one update, G_μ is positive definite, $g_{\text{mix}}^\mu \neq 0$, and J_{task}^μ admits a second-order Taylor remainder in a neighborhood of θ. Consider the trust-region problem*

$$\max_{\Delta\theta} \quad (g_{\text{mix}}^\mu)^\top \Delta\theta \qquad \textit{subject to} \qquad \frac{1}{2}\Delta\theta^\top G_\mu \Delta\theta \leq \delta.$$

Let $\Delta\theta^\star$ denote its optimizer. Then:

1. *The optimizer is the natural-gradient direction*

$$\Delta\theta^\star = \sqrt{\frac{2\delta}{(g_{\text{mix}}^\mu)^\top G_\mu^{-1} g_{\text{mix}}^\mu}}\, G_\mu^{-1} g_{\text{mix}}^\mu.$$

2. *If the Fisher-geometry alignment condition*

$$(g_{\text{task}}^\mu)^\top G_\mu^{-1} g_{\text{mix}}^\mu \geq 0$$

holds, then the first-order term in the task-objective expansion is nonnegative:

$$J_{\text{task}}^\mu(\theta + \Delta\theta^\star) - J_{\text{task}}^\mu(\theta) = (g_{\text{task}}^\mu)^\top \Delta\theta^\star + O(\|\Delta\theta^\star\|^2),$$
$$(g_{\text{task}}^\mu)^\top \Delta\theta^\star \geq 0.$$

In particular, if

$$(g_{\text{task}}^\mu)^\top G_\mu^{-1} g_{\text{mix}}^\mu > 0,$$

then for sufficiently small δ,

$$J_{\text{task}}^\mu(\theta + \Delta\theta^\star) \geq J_{\text{task}}^\mu(\theta).$$

3. *Moreover, $\Delta\theta^\star$ maximizes the first-order improvement in the selector-averaged surrogate among all updates with the same Fisher radius:*

$$(g_{\text{mix}}^\mu)^\top \Delta\theta^\star = \max_{\frac{1}{2}\Delta\theta^\top G_\mu \Delta\theta \leq \delta} (g_{\text{mix}}^\mu)^\top \Delta\theta.$$

Proof. The constrained optimization problem is linear in $\Delta\theta$ with a quadratic constraint. Its Lagrangian is

$$\mathcal{L}(\Delta\theta, \lambda) = (g^{\mu}_{\text{mix}})^{\top}\Delta\theta - \lambda\left(\frac{1}{2}\Delta\theta^{\top}G_{\mu}\Delta\theta - \delta\right).$$

Stationarity gives

$$g^{\mu}_{\text{mix}} - \lambda G_{\mu}\Delta\theta = 0 \quad \Longrightarrow \quad \Delta\theta = \lambda^{-1}G_{\mu}^{-1}g^{\mu}_{\text{mix}}.$$

Substituting into the active constraint yields

$$\lambda = \sqrt{\frac{(g^{\mu}_{\text{mix}})^{\top}G_{\mu}^{-1}g^{\mu}_{\text{mix}}}{2\delta}},$$

which proves the formula for $\Delta\theta^{\star}$.

Next, Taylor expand the frozen-selector task objective around θ:

$$J^{\mu}_{\text{task}}(\theta + \Delta\theta^{\star}) - J^{\mu}_{\text{task}}(\theta) = (g^{\mu}_{\text{task}})^{\top}\Delta\theta^{\star} + O(\|\Delta\theta^{\star}\|^2).$$

Using the closed form for $\Delta\theta^{\star}$,

$$(g^{\mu}_{\text{task}})^{\top}\Delta\theta^{\star} = \sqrt{\frac{2\delta}{(g^{\mu}_{\text{mix}})^{\top}G_{\mu}^{-1}g^{\mu}_{\text{mix}}}}\,(g^{\mu}_{\text{task}})^{\top}G_{\mu}^{-1}g^{\mu}_{\text{mix}}.$$

Therefore nonnegative Fisher alignment implies nonnegative first-order task improvement, and strict alignment implies improvement for sufficiently small trust-region radius.

Finally, by construction $\Delta\theta^{\star}$ solves the trust-region problem, so it is exactly the maximizer of first-order improvement in the selector-averaged surrogate over the Fisher ball. □

Remark J.2. Theorem J.1 does not by itself prove monotonic improvement for the full MMPS update: MMPS changes the selector μ online and learns its value estimates from nonstationary data. The theorem isolates the frozen-selector policy step, where task improvement depends on the alignment between the shaped surrogate and task objective in Fisher geometry. The full algorithm must also contend with distribution shift from the changing selector.