TASK SPECIALIZATION FINE-TUNING FOR CONTEXTUAL REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Contextual Reinforcement Learning (CRL) seeks to generalize classical RL by maximizing task coverage across a context space of related tasks. While prior works often train from scratch and rely on either multi-task learning for a single policy or strategically training multiple policies, we advocate for a unified alternative: pretraining a single policy with good initial performance, followed by fine-tuning multiple policies for task specialization. This new paradigm, however, introduces unique challenges, such as heterogeneous marginal returns and sample inefficiency. This raises a critical research question: given a pretrained policy and a constrained budget, *how much* fine-tuning should each task region receive to enable sample-efficient CRL? To this end, we propose *Task Specialization Fine-Tuning (TSFT)*, an online framework that predicts fine-tuning performance with a simple parametric model and exactly solves the resulting discrete budget allocation problem via integer linear programming. Extensive experiments across diverse decision domains, including combinatorial optimization, continuous control, and LLM fine-tuning, demonstrate that TSFT significantly outperforms baselines in task coverage and approaches oracle performance. Our work charts a new direction for model-based CRL, aligning with the modern pretrain-finetune era.

1 INTRODUCTION

Reinforcement learning (RL) has achieved remarkable success across various domains (Mnih et al., 2015; Silver et al., 2016; Fawzi et al., 2022; Mankowitz et al., 2023; Zheng et al., 2026), yet it remains fragile when applied to families of related tasks that differ only in a few key environment parameters (Benjamins et al., 2023; Degrave et al., 2022). Contextual RL (CRL) (Hallak et al., 2015; Modi et al., 2018; Benjamins et al., 2023) explicitly formalizes such families as Contextual Markov Decision Processes (CMDPs), where individual tasks are parameterized by vectors within a unified context space, with the objective of achieving broad coverage across that space.

Existing paradigms for solving CMDPs fall into three categories (see also Table 1): 1) *independent training* (Mnih et al., 2015; Rusu et al., 2016; Yu et al., 2020; Benjamins et al., 2023), which learns a separate policy for each task, straightforward yet expensive for covering high-dimensional context spaces; 2) *multi-task training* (Caruana, 1997; Wilson et al., 2007; Teh et al., 2017; Sodhani et al., 2021), which learns a single unified policy across all tasks but is constrained by limited model capacity and susceptible to negative transfer; 3) *multi-policy training* (Cho et al., 2024; Ivanov & Ben-Porat, 2024; Ge et al., 2025; Zhou et al., 2026; Cho et al., 2026), which trains multiple policies, each on a single task, and relies on their generalization to cover the unseen tasks. Despite their differences, these paradigms share a common assumption: each policy is trained from scratch until convergence. Such an approach stands at odds with the pretrain-finetune convention of modern machine learning (Bommasani et al., 2021; Achiam et al., 2023; Yang et al., 2023; Taiga et al., 2023; Sun et al., 2023), limiting the potential scalability of CRL in expansive, high-dimensional context space (Benjamins et al., 2023; Cho et al., 2024; Zhou et al., 2026).

In this paper, we advocate for a unified alternative: pretrain a *single* policy with good initial performance across the context space, then fine-tune the trained policy with *multiple* specialized variants under a constrained budget. A conceptual overview of the studied problem is illustrated in Fig. 1. This pretrain-finetune paradigm unifies multi-task and multi-policy training and holds potential advantages for three reasons. First, the pretrained policy may offer a good initialization across the

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.

Table 1: Comparison of CRL paradigms along three orthogonal axes.

Paradigm	# Policy	Training Method	Training Mode
Independent Training	Multiple $\mathcal{N}$	Single-task	From scratch
Multi-Task Training	Single	Multi-task	From scratch
Multi-Policy Training	Multiple $N \ll \mathcal{N}$	Single-task	From scratch
TSFT (Ours)	**Multiple** $N \ll \mathcal{N}$	**Multi-task**	**Pretrain + Fine-tune**

context space. Second, fine-tuning enables the specialization of multiple policies, mitigating negative transfer without the computational expense of training from scratch. Finally, this approach integrates CRL into the scalable frameworks characteristic of modern machine learning.

However, this paradigm introduces unique challenges: fine-tuning gains are *heterogeneous* across tasks, where some regions improve rapidly while others quickly saturate or even degrade. Consequently, a uniform allocation of fine-tuning compute is often sample-inefficient, prompting a key research question: *given a pretrained policy and a constrained budget, how much fine-tuning should each task region receive to enable sample-efficient CRL?* Notably, existing CRL literature has primarily investigated *where* to train, selecting source tasks to maximize coverage under the implicit assumption of a fixed per-policy budget. The orthogonal question of *how much* fine-tuning each region should receive remains largely open. To this end, this paper introduces a budget-aware perspective on CRL that complements existing source-task selection.

We introduce *Task Specialization Fine-Tuning* (TSFT), an online framework for model-based budget allocation in CRL. Specifically, TSFT leverages a simple yet effective parametric model to predict how task performance, and consequently the induced coverage set, evolves as additional budget is assigned. Based on these predictions, each planning step reduces to a maximum coverage problem (MCP) variant over allocation vectors, which can be solved exactly through integer linear programming (ILP). While ILP yields an optimal solution with respect to the model-based allocation, its theoretical optimality may be compromised by modeling errors. To mitigate this issue, we further embed ILP solving into an online framework with periodic model re-estimation. At each step, TSFT solves the current ILP, executes the derived policy for a limited horizon, collects new data, and updates the model accordingly. Empirical results demonstrate the promise of online budget allocation in CRL, enabling the coverage landscape to be substantially expanded under a constrained budget.

Contributions: 1) *Conceptually*, we formulate a budget-constrained fine-tuning problem in CRL, shifting the focus from *where* to train to *how much* fine-tuning to allocate under a pretrain-finetune paradigm; 2) *Methodologically*, we introduce TSFT, an online framework that optimizes model-based budget allocation via ILP, accompanied by a theoretical error analysis. 3) *Experimentally*, we validate TSFT across combinatorial optimization, continuous control, and LLM fine-tuning, with up to 2–3$\times$ improvements in task coverage over simple strategies, and 2$\times$ gains in sample efficiency over multi-task training baselines, while performing comparably to an oracle policy.

2 Related Work

Multi-task learning and CRL aim to generalize across related tasks through shared representations or task-conditioned policies (Caruana, 1997; Sodhani et al., 2021), but increasing task diversity can lead to negative transfer (Standley et al., 2020). Existing approaches mitigate such interference through mechanisms such as transfer learning (Teh et al., 2017) and policy composition (Sun et al., 2022). More recent multi-policy CRL methods (Cho et al., 2024; Zhou et al., 2026) further focus on selecting which source tasks to train so as to maximize transfer across the context space. These methods primarily address *where* to train and typically assume that each selected policy is trained under a fixed or converged budget. Our work follows this multi-policy paradigm to mitigate negative transfer, while addressing their lack of budget allocation mechanism. In parallel, RL fine-tuning studies how policies can be efficiently adapted through techniques such as meta-learning (Finn et al., 2017), and parameter-efficient adaptation (Schmied et al., 2023). These works mainly address how to obtain a useful initialization, stabilize one fine-tuning run, or adapt a single agent to a new task. In contrast, TSFT treats fine-tuning progress itself as a planning signal: it models heterogeneous marginal gains from specialization and coordinates compute across a portfolio of fine-tuned policies. Due to space constraints, we defer a more comprehensive discussion of related work to Appendix A.

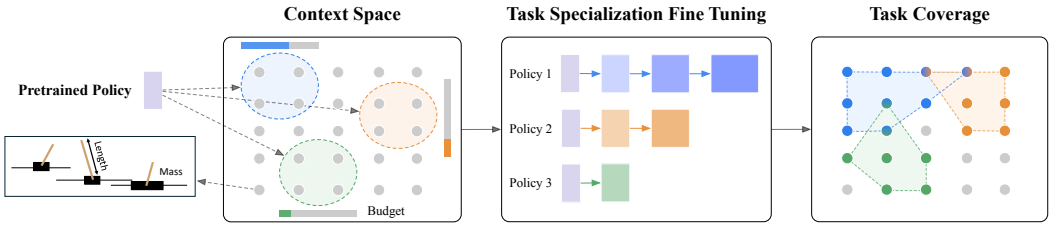


Figure 1: Conceptual Overview of Task Specialization Fine-Tuning for Contextual RL. We illustrate the framework using a two-dimensional CartPole context space, where a pretrained policy is fine-tuned on three source task sets under different budget allocations to maximize final task coverage.

3 Problem Statement

Contextual MDP. Let $M = (S, A, P, R, \rho)$ denote a standard MDP, where S is the state space, A the action space, P the transition dynamics, R the reward function, and ρ the initial state distribution. A contextual MDP, denoted by $\{M_x\}_{x\in\mathcal{X}}$, is a family of context-specific MDPs $M_x = (S, A, P_x, R_x, \rho_x)$ parameterized by a context vector x drawn from a finite and bounded context set $\mathcal{X}$, which can influence transition dynamics, reward function, and initial state distribution (Hallak et al., 2015; Modi et al., 2018; Benjamins et al., 2023). Hereafter, we simplify notation by using x to refer to a specific task.

Task Specialization Fine-Tuning. Given a pretrained policy with parameters θ, it can be fine-tuned on a source task set $X_{\mathcal{S}} \subset \mathcal{X}$ with k budget units to achieve task specialization ($\theta \to \theta^k_{X_{\mathcal{S}}}$) on a context subspace. The performance of the policy on a task $x \in \mathcal{X}$ is denoted by $J(\theta^k_{X_{\mathcal{S}}}, x)$. A task x is considered *covered* by the policy if its performance satisfies a predefined threshold, $J(\theta^k_{X_{\mathcal{S}}}, x) \le \epsilon$, assuming a minimization objective. The coverage set of the policy is defined as the union of all such covered tasks, $\mathcal{C}(\theta^k_{X_{\mathcal{S}}}) = \{x \in \mathcal{X} | J(\theta^k_{X_{\mathcal{S}}}, x) \le \epsilon\}$. This convention is without loss of generality: for a maximization objective, the inequality is simply reversed.

Prior studies typically assume full training to convergence, which can be sample inefficient. This paper instead considers a more practical budget-constrained setting and formulates task specialization as a budget allocation problem. Formally, given N distinct source task sets $\{X_{\mathcal{S}_1}, X_{\mathcal{S}_2}, \ldots, X_{\mathcal{S}_N}\}$ and a total budget of K units, the goal is to distribute this constrained budget to maximize the global coverage of the context space (i.e., the union of converage sets). Starting from a pretrained policy θ, we independently fine-tune it on each source task set $X_{\mathcal{S}_n}$, resulting in N specialized policies. The allocation objective is thus formulated as follows:

$$\begin{aligned} \max_{\mathcal{K}} \quad & G(\mathcal{K}), \\ \text{s.t.} \quad & \|\mathcal{K}\|_1 \le K, \end{aligned} \tag{1}$$

where $\mathcal{K} = [k_1, k_2, \ldots, k_N]^T \in \mathbb{Z}^N_{\ge 0}$ denotes the allocation vector, and k_n denotes the number of budget units allocated to fine-tuning the n-th policy on its source task set $X_{\mathcal{S}_n}$. Given that specialized policies may exhibit overlapping coverage across the context space, we define the *global coverage* as the measure of the union of individual coverage sets:

$$G(\mathcal{K}) = \sum_{x\in\mathcal{X}} \mathbb{I}\left(x \in \bigcup_{n=1}^{N} \mathcal{C}(\theta^{k_n}_{X_{\mathcal{S}_n}})\right) = \sum_{x\in\mathcal{X}} \mathbb{I}\left(\min_n J(\theta^{k_n}_{X_{\mathcal{S}_n}}, x) \le \epsilon\right). \tag{2}$$

In this paper, we define a *budget unit* as the number of data consumed over training epochs or gradient steps, depending on the specific domain. This formulation induces a *combinatorial* search space and leads to a *non-convex* optimization problem, whose solution strategy is discussed next.

4 Methodology

In this section, we formalize the problem as a model-based allocation problem and solve it exactly via ILP. Then, we introduce the overall framework TSFT, which embeds ILP solving into an online setting to enable effective task specialization in CRL.

4.1 PROBLEM FORMULATION

Given a pretrained policy, a policy class, an RL training algorithm, a set of source task distributions $\{X_{\mathcal{S}_1}, \ldots, X_{\mathcal{S}_N}\}$, and a total budget K, the goal is to decide *how much* budget should be assigned to each policy. Each allocation decision assigns one fixed budget unit to a policy, e.g., by running the given RL algorithm for a fixed number of samples or epochs on the corresponding source task set. Although the total budget induces a finite horizon of K allocation decisions, the final objective depends only on the resulting allocation vector, rather than on the order of decisions. We therefore directly optimize over feasible allocations subject to the budget constraint. An alternative MDP perspective is provided in Appendix C.1.

ILP Formulation. We first consider the model-free setting in which the coverage set $\mathcal{C}(\theta^k_{X_{\mathcal{S}_n}})$ is known for every policy n and allocation level k. Under this assumption, Eq. (1) can be reformulated exactly as an ILP. Specifically, we introduce a binary decision variable $z_{n,k}$ to indicate whether policy n is allocated k budget units, and a binary decision variable q_x to indicate whether task x is covered by at least one policy. The resulting ILP formulation is defined as follows:

$$\max_{\mathbf{z},\mathbf{q}} \quad \sum_{x\in\mathcal{X}} q_x, \tag{3}$$

$$\text{s.t.} \quad \sum_{k=0}^{K} z_{n,k} = 1, \qquad \forall n \in \{1,\ldots,N\}, \tag{4}$$

$$\sum_{n=1}^{N}\sum_{k=0}^{K} k\, z_{n,k} \le K, \tag{5}$$

$$q_x \le \sum_{n=1}^{N} \sum_{k:\, x\in\mathcal{C}(\theta^k_{X_{\mathcal{S}_n}})} z_{n,k}, \qquad \forall x \in \mathcal{X}, \tag{6}$$

$$z_{n,k} \in \{0,1\}, \quad q_x \in \{0,1\}. \tag{7}$$

Constraint (4) selects exactly one allocation level for each policy. Constraint (5) ensures that the total allocated budget does not exceed K. Constraint (6) enforces that q_x can be set to one only if task x is covered by at least one policy. The optimal solution to Eq. (1) is recovered as $\mathcal{K}^* = [k_1^*, k_2^*, \ldots, k_N^*]$, where $k_n^\star = \sum_{k=0}^{K} k z_{n,k}^\star$. Our formulation can be viewed as a multiple-choice budgeted variant of the maximum coverage problem (MCP), with alternative equivalent formulations provided in Appendix C.2. In specific, each policy is associated with $K+1$ candidate coverage sets, among which exactly one must be selected. The formulation contains $N(K+1)+|\mathcal{X}|$ decision variables and $N+|\mathcal{X}|+1$ constraints. The ILP solver (e.g., CBC) operates on the precomputed coverage sets and leverages mature branch-and-bound and cutting-plane methods for efficient optimization. When a time limit is imposed, the solver returns the best feasible incumbent along with its optimality status. However, in practice, the coverage set $\mathcal{C}(\theta^k_{X_{\mathcal{S}_n}})$ is not known a priori. We therefore introduce a surrogate performance model to approximate the induced coverage sets.

Surrogate Performance Model. We model the evolution of each policy's task-wise performance and thereby implicitly estimate its coverage set. For each policy-task pair (n, x), we fit a parametric performance model of the form:

$$\mathcal{F}(y) = L \pm Be^{-dy}, \tag{8}$$

where L, $B \ge 0$, and $d > 0$ are learnable parameters, and y denotes the consumed budget. The sign of B controls the direction of the performance shift, capturing both improvement and degradation during fine-tuning. We fit $\mathcal{F}_n^x(y)$ via non-linear least squares (NLLS), which predicts the performance of policy n on task x after consuming a given amount of budget y. Empirically, this fitting process can be performed in parallel over thousands of tasks within seconds. We refer the reader to Appendix F.2 for a comprehensive study of alternative models.

The true optimal solution (i.e., Oracle) could, in principle, be obtained by training each policy to completion using the full budget K, and then solving Eq. (1) exactly in a model-free manner. However, in practice, when computing G via Eq. (2), $J(\theta^{k_n}_{X_{\mathcal{S}_n}}, x)$ is unknown during the planning phase. We therefore replace it with our model prediction $\mathcal{F}_n^x(\cdot)$, yielding a surrogate objective $\widehat{G}$ that

Algorithm 1 Task Specialization Fine-Tuning (TSFT)

Input: Context space $\mathcal{X}$, policy count N, total budget K, warmup budget W, execution budget E, source task sets $\{X_{\mathcal{S}_n}\}_{n=1}^{N}$;
Output: N trained policies;

1: **Initialize:** $\mathcal{K} = \{k_n\}_{n=1}^{N}$, $k_n = \frac{W}{N}$
2: $\Theta \leftarrow$ Train N policies on $\{X_{\mathcal{S}_n}\}_{n=1}^{N}$ following $\mathcal{K}$
3: $\mathcal{Z} \leftarrow$ Evaluate Θ on each task $x \in \mathcal{X}$ to construct dataset
4: Remaining budget $b \leftarrow K - W$
5: **while** $b > 0$ **do**
6: $\quad \forall n, x : \mathcal{F}_n^x \leftarrow$ Fit models using dataset $\mathcal{Z}$
7: $\quad \widehat{\mathcal{K}} \leftarrow$ Solve ILP with models $\{\mathcal{F}_n^x\}_{n,x}$
8: $\quad \Theta \leftarrow$ Distribute budget $\min(b, E)$ based on $\widehat{\mathcal{K}}$; Train $\mathcal{N}$ policies
9: $\quad \mathcal{Z}' \leftarrow$ Evaluate Θ on each task $x \in \mathcal{X}$
10: $\quad$ Update dataset $\mathcal{Z} \leftarrow \mathcal{Z} \cup \mathcal{Z}'$
11: $\quad$ Update remaining budget $b = b - \min(b, E)$
12: **end while**

we optimize within our framework. To optimize $\widehat{G}$ via ILP, we replace $\mathcal{C}(\theta^k_{X_{\mathcal{S}_n}})$ in Constraint (6) with the estimated coverage set $\widehat{\mathcal{C}}(\theta^k_{X_{\mathcal{S}_n}}) = \{x \in \mathcal{X} | \mathcal{F}_n^x(k) \leq \epsilon\}$. The resulting ILP can be solved exactly with respect to $\widehat{G}$, although its solution may differ from that of G because of modeling error.

4.2 Overall Framework

The above subsections detail how we solve the ILP to obtain the solution (i.e., allocation policy) at a single decision-making step (line 7). We now embed this procedure into our online framework TSFT for task specialization in contextual RL. The algorithmic workflow is presented in Alg. 1.

Concretely, the framework begins with a *warmup stage* (lines 1-4), in which the warmup budget W is uniformly allocated across N policies. Each policy n is then trained on its corresponding source task set $X_{\mathcal{S}_n}$ according to this allocation. During training, we save checkpoints for every budget unit. Subsequently, all checkpoints $\Theta = \{\theta^0_{X_{\mathcal{S}_n}}, \theta^1_{X_{\mathcal{S}_n}}, \ldots, \theta^{k_n}_{X_{\mathcal{S}_n}}\}_{n=1}^{N}$ are evaluated on each task $x \in \mathcal{X}$ to collect training data $\mathcal{Z}$. In practice, this evaluation can be performed online if checkpoint storage becomes prohibitive. This warmup phase ensures that all policies receive equal initial training and provides sufficient data to enable reliable model learning.

Next (lines 5-12), the framework iteratively allocates an *execution budget* E at each decision-making step until the remaining budget b is exhausted. In specific, at each step, for each policy-task pair (n, x), we first fit a model $\mathcal{F}_n^x$ to the available training data $\mathcal{Z}$. By calculating the surrogate objective with these models, we solve the model-based allocation via ILP to derive its solution $\widehat{\mathcal{K}}$. However, due to inherent modeling errors, we do not execute $\widehat{\mathcal{K}}$ to completion. Instead, we allocate the execution budget E proportionally according to $\widehat{\mathcal{K}}$, which may prevent over-committing to an imperfect model and ensure balanced exploration. We then train the policies under this allocation and collect new data through evaluation. This newly acquired data $\mathcal{Z}'$ is appended to the existing training dataset $\mathcal{Z}$, which may progressively reduce modeling error in subsequent iterations. Furthermore, if the ILP solver terminates early without allocating the entire execution budget, any unused budget is carried forward and reconsidered in subsequent TSFT iterations.

To analyze TSFT, we introduce two oracle allocation policies. The *Oracle* directly optimizes the true coverage objective over the entire budget and represents the best possible allocation in hindsight. The *Oracle-Warmup* follows the same warmup stage as TSFT but optimizes the remaining allocation using the true coverage objective. We then present the following error bound and decomposition.

Theorem 1 (Oracle Gap Decomposition)**.** *Let $G(\mathcal{K})$ denote the global coverage achieved by $\mathcal{K}$, $\mathcal{K}^*$ denote the Oracle allocation, $\mathcal{K}^*_{\mathrm{OW}}$ denote the Oracle-Warmup allocation, and $\mathcal{K}_{\mathrm{TSFT}}$ denote the allocation returned by TSFT. Under Assumptions 1 and 2, the optimality gap of TSFT satisfies*

$$G(\mathcal{K}^*) - G(\mathcal{K}_{\mathrm{TSFT}}) \leq G(\mathcal{K}^*) - G(\mathcal{K}^*_{\mathrm{OW}}) + 2\delta_m + \eta_{\mathrm{alg}}. \tag{9}$$

Theorem 1 shows that the gap between TSFT and the Oracle decomposes into three terms: the warmup error $G(\mathcal{K}^*) - G(\mathcal{K}^*_{\mathrm{OW}})$, the surrogate-model error $2\delta_m$, and the algorithmic planning error η_{alg}. If the post-warmup state lies on an optimal trajectory, the warmup error becomes zero. Moreover, this provides a conditional post-warmup guarantee with respect to a fixed surrogate objective. For a one-shot exact full-horizon planner that optimizes this fixed surrogate and executes the returned allocation exactly, $\eta_{\mathrm{alg}} = 0$. We refer to Appendix D for proofs and further discussion.

In summary, TSFT solves a model-based allocation via ILP at each decision-making step, while allowing for periodic model re-estimation to mitigate error accumulation. This enables adaptive and sample-efficient policy training, facilitating effective task specialization in CRL.

5 Experiment

In this section, we empirically validate our framework across continuous control, combinatorial optimization, and LLM fine-tuning. The source code will be publicly released upon publication.

Baseline. 1) *Oracle*: We first train each policy to completion (i.e., using the maximal budget K), and then solve the ILP defined in Eqs. (3)-(7). This serves as a standard oracle reference, representing an upper bound on achievable performance. 2) *Oracle-Warmup*: Similar to the Oracle, but isolates the warmup phase by solving the ILP after warmup. Specifically, it treats the post-warmup allocation $\mathcal{K}^0$ as the initial state and optimizes the true coverage objective over $\Omega_W = \{\mathcal{K} \in \mathbb{Z}^N_{\geq 0} : \mathcal{K} \succeq \mathcal{K}^0,\ \|\mathcal{K}\|_1 \leq K\}$. Therefore, Oracle-Warmup shares the same initial decision-making state as our proposed framework, serving as an additional oracle reference for validating model error. 3) *Pretrained*: The given pretrained policy without further adaptation. Depending on the decision domain, the pretrained policy may be trained outside the context space. 4) *MTL*: This approach fine-tunes the pretrained policy over the entire context space in a multi-task learning manner, using the same computation budget as the other methods. 5) *Random*: This approach randomly allocates the compute budget across the N policies. 6) *Uniform*: This approach evenly allocates the compute budget across the N policies. 7) *Adaptive*: At each decision-making step, this approach adaptively allocates the decision budget D according to a probability distribution derived from the observed improvement rate of each policy's coverage set. 8) *LinUCB* (Li et al., 2010; Chu et al., 2011): This approach formulates the compute allocation as a contextual multi-armed bandit problem. At each decision step, it utilizes the Linear Upper Confidence Bound (LinUCB) algorithm to select a policy for training. The context vector encodes the normalized training progress and coverage momentum, while the reward is defined as the monotonic improvement in the policy's coverage set to account for temporary performance dips. We refer to Appendix E.1 for additional details on the baselines.

We report average results in the main paper (see Appendix F for full results). During inference, we evaluate all N policies and select the best one for each task, following the convention in CRL.

5.1 Combinatorial Optimization

Environment and Context Space. We consider the capacitated vehicle routing problem (CVRP) and its variant with time window constraints (CVRPTW). 1) *CVRP*: We construct a 2D context space defined by customer node distribution and vehicle capacity. The distribution parameter ranges from [0.01, 0.25] with a step size of 0.01, while the vehicle capacity ranges from [10, 400] with a step size of 10. This results in a total of 1,000 tasks in the context space. 2) *CVRPTW*: We construct a 2D context space defined by time window tightness and vehicle capacity. The time window parameter ranges from [0.04, 1.00] with a step size of 0.04, while the vehicle capacity ranges from [10, 400] with a step size of 100. This also yields a total of 1,000 tasks in the context space. Each source task set corresponds to a square region in the context space, comprising 49 tasks centered around a reference task. More details are presented in Appendix E.3.

Setup. We adopt POMO (Kwon et al., 2020) as the policy network, consisting of 1.27M parameters, and train it using the REINFORCE algorithm (Williams, 1992), following the training configurations in (Kwon et al., 2020) (see Appendix E.2). In this domain, one budget unit corresponds to 100 training epochs, with each epoch processing 10,000 data samples. Starting from a policy pretrained over the entire context space for 5,000 epochs, we perform task specialization via multi-task fine-tuning on the corresponding source task sets. Performance is evaluated in terms of optimality gap with respect to HGS (Vidal, 2022). We set the performance threshold ϵ to 1.25% for CVRP and 3%

Table 2: Performance Comparison in Combinatorial Optimization and Continuous Control.

	CVRP			CVRPTW			CartPole			Ant		
N/K	3/100	3/150	4/150	3/100	3/150	4/150	3/50	3/100	4/100	3/50	3/100	4/100
Oracle	26.7%	29.3%	32.5%	49.4%	50.0%	50.0%	89.2%	89.7%	90.5%	100.0%	100.0%	100.0%
Oracle-Warmup	21.4%	24.5%	28.0%	46.6%	47.5%	47.5%	86.8%	87.5%	90.4%	100.0%	100.0%	100.0%
Pretrained	6.8%	6.8%	6.8%	40.9%	40.9%	40.9%	66.6%	66.6%	66.6%	31.1%	31.1%	31.1%
MTL	9.5%	11.0%	11.0%	44.8%	46.0%	46.0%	71.9%	74.5%	74.5%	31.1%	31.1%	31.1%
Random	10.1%	11.0%	12.9%	42.9%	43.2%	43.3%	75.4%	75.0%	82.4%	15.8%	42.5%	45.7%
Uniform	7.9%	15.4%	12.4%	42.8%	41.8%	42.0%	72.8%	65.4%	82.8%	25.7%	53.0%	64.7%
Adaptive	10.4%	16.1%	14.5%	43.2%	44.0%	44.0%	75.1%	76.9%	87.3%	54.1%	88.0%	91.9%
LinUCB	12.6%	20.5%	20.8%	42.8%	44.3%	44.3%	77.7%	82.6%	82.6%	64.3%	88.2%	76.9%
TSFT	**18.7%**	**23.0%**	**26.4%**	**45.8%**	**46.2%**	**46.9%**	**82.2%**	**84.0%**	**87.6%**	**88.4%**	**97.7%**	**99.5%**

for CVRPTW. We consider a range of budget configurations. For example, 3/100 denotes training $N = 3$ policies with a total budget of $K = 100$ units, as shown in Table 2. In our framework, the warmup budget is set to $W = 5 \times N$ units. The execution budget is initially set to $E = 5$ units, and is increased to 10 after a cumulative budget of $10 \times N$ has been consumed. This reflects increased confidence in the model as more data becomes available and reduces the frequency of ILP solving.

Result. We report the best coverage rate achieved by all approaches on CVRP and CVRPTW in Table 2, where the coverage rate is defined as the global coverage divided by the total number of tasks in the context space. TSFT achieves the best non-oracle performance across all six settings. On CVRP, TSFT brings substantial improvements over the strongest baseline, especially as the budget or number of policies increases, and closely approaches Oracle-Warmup. On CVRPTW, the gains are more moderate because several baselines, particularly MTL, already achieve strong coverage, possibly due to reduced task interference in the considered context space. Nevertheless, TSFT still consistently improves over all non-oracle baselines and remains close to Oracle-Warmup.

5.2 Continuous Control

Environment and Context Space. We evaluate on three continuous control suites: 1) *CartPole 3D* from CARL (Benjamins et al., 2023), where the context varies pole length, cart mass, and pole mass. Each axis is discretized into 10 values evenly spaced over $[0.1, 10]\times$ the CARL defaults, yielding a 10^3-task grid. 2) *Ant 2D* from CARL, where the context varies gravity and friction over $[0.2, 2.0]\times$ their defaults $(g_0, \mu_0) = (9.8, 1.0)$ on a 25×40-task grid. 3) *Meta-World MT50* (Yu et al., 2020), a benchmark of 50 robotic manipulation tasks. We refer to Appendix E.3 for more details.

Setup for CartPole and Ant. We use PPO (Schulman et al., 2017) as the base RL algorithm, with an MLP policy/value network of hidden sizes $[64, 64]$ for CartPole and $[256, 256]$ for Ant. The pretrained policy θ is obtained by multi-task PPO on the full grid for 5M (CartPole) and 1M (Ant) environment steps. One budget unit corresponds to 10,000 steps for CartPole and 40,000 steps for Ant. Source task sets correspond to N axis-aligned regions of the grid, each containing $3^3 = 27$ contexts for CartPole and $5\times4 = 20$ contexts for Ant. We set the performance threshold ϵ to 500 for CartPole and 6.9 for Ant. Note that larger values indicate better performance in continuous control. We consider budget settings with $N \in \{3, 4\}$ and $K \in \{50, 100\}$ for both environments. For TSFT, we set $E = 50$ for CartPole and $E = 10$ for Ant, while keeping the other hyperparameters the same as those used for combinatorial optimization. We report the average performance across seeds in the main paper, with per-seed results provided in Appendix F.

Setup for Meta-World. We use MOORE (Hendawy et al., 2024) as the base multi-task RL algorithm. Specifically, we pretrain MOORE on the full MT50 benchmark for 50M steps to obtain the pretrained policy. The remaining 50M steps are used for task specialization, where each budget unit corresponds to 1M steps. To construct the source task sets, we extract the task-specific expert weights learned by pretrained MOORE and apply K-means clustering to partition the 50 tasks into 10 groups, each containing 5 tasks. We set the performance threshold to $\epsilon = 1.0$, corresponding to a success rate of 100% over five evaluation episodes. We consider the budget setting with $N = 10$ policies and $K = 50$ units. For TSFT, we set the warm-up budget to $W = 3 \times N$. The remaining hyperparameters are the same as those used for combinatorial optimization. To evaluate robustness, we additionally report results on a smaller setting with 5 random seeds in Appendix F.

Result. Table 2 reports global coverage on Cart-Pole and Ant. Gains over adaptive and bandit baselines are moderate, but TSFT consistently improves coverage and narrows the gap to Oracle-Warmup as the budget grows. MTL can be unstable here, e.g., on Ant, additional training does not necessarily yield higher task coverage. In contrast, Meta-World results (Table 3) show strong MTL performance, yet TSFT still achieves better coverage. Specifically, MOORE achieves an average success rate of 0.68 with 60% task coverage, whereas TSFT improves these to 0.72 and 74%, respectively. We also note that task grouping plays an important role, yet it is not deliberately optimized in this paper. Further improving the task grouping in Meta-World could yield even better performance and higher sample efficiency. Overall, these results show that model-based budget allocation is effective for continuous-control CRL, especially in higher-dimensional settings where naive multi-task fine-tuning is sample-inefficient.

Table 3: Results for Meta-World and LLM-FT.

	Meta-World (10/50)	LLM-FT (4/100)
Oracle	86.0%	77.8%
Oracle-Warmup	84.0%	77.8%
Pretrained	58.0%	0.0%
MOORE	60.0%	/
Random	71.2%	26.7%
Uniform	70.0%	22.2%
Adaptive	71.2%	28.9%
LinUCB	68.0%	33.3%
TSFT	**74.0%**	**55.6%**

5.3 LLM Fine-Tuning

Environment and Context Space. We evaluate TSFT on reinforcement fine-tuning (RFT) of LLMs. The discrete 1D context space consists of 9 reasoning tasks (benchmarks). The in-distribution (ID) set comprises the test splits of our four fine-tuning corpora: DAPO-17K (Yu et al., 2025), MATH-500 (Hendrycks et al., 2021; Lightman et al., 2024), GSM8K (Cobbe et al., 2021), and CodeContests+ (Wang et al., 2025). The out-of-distribution (OOD) set comprises five held-out benchmarks: AIME 2024/2025, Minerva Math (Lewkowycz et al., 2022), MBPP (Austin et al., 2021), and BigCodeBench (Zhuo et al., 2025).

Setup. We use Qwen3-4B-Base (Yang et al., 2025) as the pretrained policy and apply GRPO (Shao et al., 2024) with verifiable rule-based rewards. We construct $N = 4$ specialized policies, each fine-tuned on one of {DAPO-17K, MATH, GSM8K, CodeContests+}, under a total budget of $K = 100$, where one budget unit corresponds to a single GRPO update step. Following Eq. (1), each task is dispatched to all N policies and the best pass@1 (greedy decoding) is taken. A task is considered covered if pass@1 exceeds a per-benchmark threshold. For TSFT, we use the same hyperparameters as in the combinatorial optimization setting. Full training, decoding, and threshold details are presented in Appendices E.2 and E.3.

Result. Table 3 reports the global coverage on the LLM suite. TSFT consistently outperforms all non-oracle baselines and approach Oracle-Warmup, whereas heuristic allocation baselines recover only part of the specialization benefit. These results suggest that model-based budget allocation can scale to LLM RFT with billions of parameters.

5.4 Analysis

We further present detailed analyses using CVRP as a standard testbed. See Appendix F for additional empirical results and analyses (e.g., full results, performance model studies, sensitivity analyses, low-cost evaluation, computational cost, and visualizations).

Diverse Budget. We provide a detailed comparison of performance across varying budget settings for the case of $N = 3$ in Fig. 2. The results consistently demonstrate the superiority of TSFT in terms of global coverage across both low- and high-budget regimes. In contrast, LinUCB performs well only in high-budget settings, as it requires sufficient exploration to identify which bandit (policy) has the potential for improvement through trial-and-error, highlighting its relative sample inefficiency.

Computational Cost. TSFT introduces only modest overhead, as both surrogate performance modeling and ILP optimization are efficient. Thousands of surrogate models can be fitted in parallel within seconds, and an ILP with $N = 10$ and $K = 150$ can be solved to optimality in under one second. This cost is minor compared with policy training, which takes about one week for CVRP.

Model Error. For each policy-task pair (n, x), TSFT collects one data point per budget unit, which is used to fit the predictive model $\mathcal{F}_x^n$ at each decision-making step. We report the recall rate of the

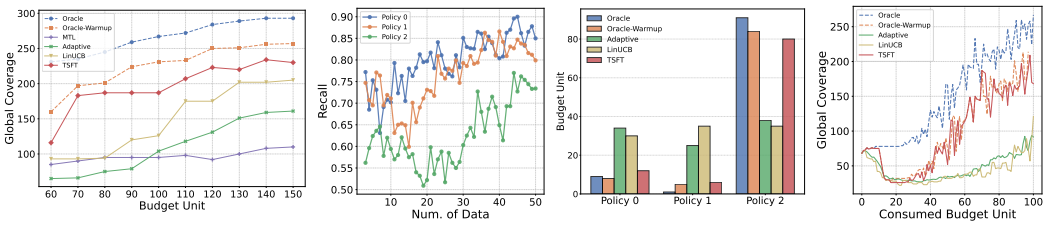


Figure 2: *From left to right:* Performance comparison under diverse budget settings; Average recall for predicting coverage sets over the future 50 budget units, given varying amounts of data for model fitting; An example run illustrating the allocation policies and the global coverage trajectories over the training process across different methods for the case with $N = 3$ and $K = 100$.

coverage set estimated by the fitted model as more data becomes available. The results in Fig. 2 demonstrate the effectiveness of our simple parametric model and justify the design of online model re-estimation with newly collected data. More detailed results are provided in Appendix F.2.

Sample Efficiency. We compare TSFT with the MTL baseline, which performs further training over the entire context space without task specialization. As shown in Fig. 8, MTL typically requires roughly $2\times$ more samples to achieve global coverage comparable to that of our method.

Allocation Policy. In Fig. 2, we present the allocation policies derived by each approach and the corresponding global coverage in the CVRP setting with $N = 3$ and $K = 100$. The coverage set may initially shrink due to the distribution shift between pretraining and fine-tuning.

Comparison with MTL. Since TSFT fine-tunes multiple policies during the specialization stage, we provide a comprehensive comparison with the MTL baseline on CVRP under matched training compute. Let Φ denote the number of parameters in the pretrained model, and let $\hat{K} = 200$ denote the total budget (pretraining + fine-tuning). We compare TSFT against two MTL settings: (1) MTL with a total compute budget of $\Phi \times \hat{K}$, corresponding to the standard setup; and (2) MTL with a matched compute budget of $5\Phi \times \frac{\hat{K}}{5}$, which uses a model with five times the parameters but one-fifth of the budget so that the overall number of parameter updates is comparable to TSFT. These two MTL baselines achieve only 11.0% and 7.8% task coverage, respectively, despite being trained on a larger set of source tasks (i.e., the entire context space) than TSFT (see Fig. 5).

6 Conclusion

This paper introduces Task Specialization Fine-Tuning (TSFT), a framework that intelligently guides task specialization in contextual reinforcement learning (CRL) by explicitly optimizing how much budget should be allocated to each task region under a constrained budget. Specifically, TSFT formulates the process as a model-based allocation problem using a simple parametric model, and solves it exactly via integer linear programming (ILP). We further embed the process into an online framework with periodic model re-estimation to mitigate modeling errors. Experiments on combinatorial optimization, continuous control, and large language model fine-tuning demonstrate that TSFT significantly improves task coverage, highlighting the promise of intelligent task specialization as a principled mechanism for sample-efficient policy training in complex CRL environments.

Limitations: First, the surrogate objective used for model-based allocation may be misaligned with the original objective. As a result, solutions that are optimal under the surrogate objective may not achieve strong performance under the original objective. Second, exhaustive evaluation can become computationally prohibitive as the context space grows or the number of specialized policies increases, potentially limiting the scalability of TSFT. Third, we assume a fixed task grouping and do not optimize the partitioning of the context space, which may affect the final coverage. Finally, the exponential performance model is not universally applicable and may be inadequate for domains with strongly non-monotonic fine-tuning dynamics. Developing better-aligned optimization objectives, more efficient evaluation strategies, joint optimization of task grouping and budget allocation, and more expressive performance models are promising directions for future work.

AI USE STATEMENT

In this work, we used generative AI tools to polish the writing of the paper. We did not use generative AI tools for other activities requiring disclosure, such as retrieval and discovery, drafting sections of the paper, or generating synthetic datasets. The remaining disclosure categories are not applicable to this work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Philip J Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, 2023.

Carolin Benjamins, Theresa Eimer, Frederik Schubert, Aditya Mohan, Sebastian Döhler, André Biedenkapp, Bodo Rosenhahn, Frank Hutter, and Marius Lindauer. Contextualize me–the case for context in reinforcement learning. *Transactions on Machine Learning Research*, 2023.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

Jung-Hoon Cho, Vindula Jayawardana, Sirui Li, and Cathy Wu. Model-based transfer learning for contextual reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 37, pp. 88279–88319, 2024.

Jung-Hoon Cho, Sirui Li, Jeongyun Kim, and Cathy Wu. Temporal transfer learning for traffic optimization with coarse-grained advisory autonomy. *IEEE Transactions on Robotics*, 2026.

Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. Contextual bandits with linear payoff functions. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 208–214. JMLR Workshop and Conference Proceedings, 2011.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

Adam N Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Management Science*, 68(1): 9–26, 2022.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2019.

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, 2017.

Carlos E Garcia, David M Prett, and Manfred Morari. Model predictive control: Theory and practice–a survey. *Automatica*, 25(3):335–348, 1989.

Luise Ge, Michael Lanier, Anindya Sarkar, Bengisu Guresti, Chongjie Zhang, and Yevgeniy Vorobeychik. Learning policy committees for effective personalization in MDPs with diverse tasks. In *International Conference on Machine Learning*, 2025.

Bram Grooten, Patrick MacAlpine, Kaushik Subramanian, Peter Stone, and Peter R Wurman. Out-of-distribution generalization with a sparc: Racing 100 unseen vehicles with a single policy. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026.

Assaf Hallak, Dotan Di Castro, and Shie Mannor. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015.

Ahmed Hendawy, Jan Peters, and Carlo D'Eramo. Multi-task reinforcement learning with mixture of orthogonal experts. In *International Conference on Learning Representations*, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *NeurIPS Datasets and Benchmarks Track*, 2021.

Dmitry Ivanov and Omer Ben-Porat. Personalized reinforcement learning with a budget of policies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

Samir Khuller, Anna Moss, and Joseph Seffi Naor. The budgeted maximum coverage problem. *Information processing letters*, 70(1):39–45, 1999.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. POMO: Policy optimization with multiple optima for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 21188–21198, 2020.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. In *Advances in Neural Information Processing Systems*, 2022.

Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pp. 661–670, 2010.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024.

Alexis Limozin, Eduard Durech, Torsten Hoefler, Imanol Schlag, and Valentina Pyatkin. Sft-then-rl outperforms mixed-policy methods for llm reasoning. *arXiv preprint arXiv:2604.23747*, 2026.

Jayanta Mandi, James Kotary, Senne Berden, Maxime Mulamba, Victor Bucarey, Tias Guns, and Ferdinando Fioretto. Decision-focused learning: Foundations, state of the art, benchmark and future opportunities. *Journal of Artificial Intelligence Research*, 80:1623–1701, 2024.

Daniel J Mankowitz, Andrea Michi, Anton Zhernov, Marco Gelmi, Marco Selvi, Cosmin Paduraru, Edouard Leurent, Shariq Iqbal, Jean-Baptiste Lespiau, Alex Ahern, et al. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618(7964):257–263, 2023.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Aditya Modi, Nan Jiang, Satinder Singh, and Ambuj Tewari. Markov decision processes with continuous side information. In *Algorithmic learning theory*, pp. 597–618. PMLR, 2018.

Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

Mitsuhiko Nakamoto, Simon Zhai, Anikait Singh, Max Sobol Mark, Yi Ma, Chelsea Finn, Aviral Kumar, and Sergey Levine. Cal-ql: Calibrated offline rl pre-training for efficient online fine-tuning. *Advances in Neural Information Processing Systems*, 36:62244–62269, 2023.

Michal Nauman, Marek Cygan, Carmelo Sferrazza, Aviral Kumar, and Pieter Abbeel. Bigger, regularized, categorical: High-capacity value functions are efficient multi-task learners. *arXiv preprint arXiv:2505.23150*, 2025.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International Conference on Machine Learning*, pp. 5331–5340. PMLR, 2019.

Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. In *International Conference on Learning Representations*, 2016.

Thomas Schmied, Markus Hofmarcher, Fabian Paischer, Razvan Pascanu, and Sepp Hochreiter. Learning to modulate pre-trained models in rl. *Advances in Neural Information Processing Systems*, 36:38231–38265, 2023.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Yu Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Prabhakant Sinha and Andris A Zoltners. The multiple-choice knapsack problem. *Operations Research*, 27(3):503–515, 1979.

Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. In *International Conference on Machine Learning*, 2021.

Trevor Standley, Amir Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning? In *International Conference on Machine Learning*, pp. 9120–9132. PMLR, 2020.

Lingfeng Sun, Haichao Zhang, Wei Xu, and Masayoshi Tomizuka. Paco: Parameter-compositional multi-task reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 21495–21507, 2022.

Yanchao Sun, Shuang Ma, Ratnesh Madaan, Rogerio Bonatti, Furong Huang, and Ashish Kapoor. SMART: Self-supervised multi-task pretraining with control transformers. In *International Conference on Learning Representations*, 2023.

Adrien Ali Taiga, Rishabh Agarwal, Jesse Farebrother, Aaron Courville, and Marc G Bellemare. Investigating multi-task pretraining and generalization in reinforcement learning. In *International Conference on Learning Representations*, 2023.

Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Thibaut Vidal. Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. *Computers & Operations Research*, 140:105643, 2022.

Zihan Wang et al. Codecontests+: High-quality test case generation for competitive programming. *arXiv preprint arXiv:2506.05817*, 2025.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *International Conference on Machine Learning*, pp. 1015–1022, 2007.

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

Sherry Yang, Ofir Nachum, Yilun Du, Jason Wei, Pieter Abbeel, and Dale Schuurmans. Foundation models for decision making: Problems, methods, and opportunities. *arXiv preprint arXiv:2303.04129*, 2023.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale. In *Advances in Neural Information Processing Systems*, 2025.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pp. 1094–1100. PMLR, 2020.

Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE transactions on knowledge and data engineering*, 34(12):5586–5609, 2021.

Mandi Zhao, Pieter Abbeel, and Stephen James. On the effectiveness of fine-tuning versus meta-reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 26519–26531, 2022.

Han Zheng, Yining Ma, Brandon Araki, Jingkai Chen, and Cathy Wu. Learning-guided prioritized planning for lifelong multi-agent path finding in warehouse automation. *Journal of Artificial Intelligence Research*, 85, 2026.

Tianyue Zhou, Jung-Hoon Cho, and Cathy Wu. Structure detection for contextual reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026.

Terry Yue Zhuo, Vu Minh Chien, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *International Conference on Learning Representations*, 2025.

A Related Work

Multi-Task Learning and Contextual RL. Multi-task learning improves generalization by jointly learning related tasks, but often suffers from negative transfer when tasks are insufficiently related (Caruana, 1997; Standley et al., 2020; Zhang & Yang, 2021). In reinforcement learning, this challenge is further exacerbated by diverse task dynamics, sparse feedback, and unstable optimization. Contextual RL can be viewed as a structured form of multi-task RL, where a family of related tasks is parameterized by a context variable that affects the environment dynamics, rewards, or initial-state distributions (Hallak et al., 2015; Modi et al., 2018; Benjamins et al., 2023). A common approach in both multi-task RL and CRL is to train a single policy that generalizes across tasks, often through shared task representations, context-conditioned policies, or task-conditioned policy and value heads (Sodhani et al., 2021; Yu et al., 2020; Nauman et al., 2025; Grooten et al., 2026). Although recent work shows that larger and better-regularized value functions can improve generalization across diverse tasks (Nauman et al., 2025), single-policy approaches remain limited by model capacity and may suffer from negative transfer as task diversity increases.

Another line of work addresses this through transfer learning (Teh et al., 2017) or policy composition (Sun et al., 2022), which alleviates task interference by allowing different components to specialize. Recent multi-policy CRL methods also address contextual generalization by strategically selecting source tasks and relying on zero-shot transfer to cover the remaining context space (Cho et al., 2024; Zhou et al., 2026; Cho et al., 2026). For example, MBTL models source task performance and transfer gaps to guide source task selection, while SD-MBTL further detects the underlying generalization structure of the CMDP and switches between suitable task selection strategies. These methods primarily address *where* to train and typically assume that each selected policy is trained under a fixed or converged budget. Our work follows this multi-policy paradigm to mitigate negative transfer, while addressing their lack of budget allocation mechanism.

RL Fine-Tuning. Fine-tuning has become a common way to reuse experience in RL. Early related work focused on learning an initialization or task representation that can adapt with a small amount of new data, such as MAML and PEARL (Finn et al., 2017; Rakelly et al., 2019). Another line learns reusable behaviors before downstream training, for example through unsupervised skill discovery (Eysenbach et al., 2019). More recent work studies pretraining followed by fine-tuning more directly: multi-task pretraining with task-specific fine-tuning has been shown to be a strong and simple alternative to meta-RL (Zhao et al., 2022), and multi-task pretraining on Atari variants has been shown to improve generalization to unseen variants even after substantial fine-tuning (Taiga et al., 2023). Self-supervised multi-task pretraining has also been explored for sequential decision-making models, improving downstream fine-tuning efficiency across seen and unseen control tasks (Sun et al., 2023). Offline-to-online RL methods use prior datasets to accelerate later online improvement (Nair et al., 2020; Ball et al., 2023; Nakamoto et al., 2023); and parameter-efficient methods such as L2M reduce forgetting when adapting pretrained decision-making models (Schmied et al., 2023). These works mainly address how to obtain a useful initialization, stabilize one fine-tuning run, or adapt a single agent to a new task. In contrast, TSFT treats fine-tuning progress itself as a planning signal: it models heterogeneous marginal gains from specialization and coordinates compute across a portfolio of fine-tuned policies.

B DISCUSSION

Relationship to Existing Literature. We provide a brief discussion to better contextualize the studied problem and the proposed framework. 1) At the formulation level, Eqs. (3)-(7) constitute a multiple-choice budgeted MCP (Khuller et al., 1999). Specifically, the $K + 1$ allocation levels of each policy form a group of candidate sets: selecting level k consumes k budget units and induces a corresponding coverage set. The objective is to select exactly one level for each policy so as to maximize the cardinality of the union of the selected coverage sets, subject to the total budget constraint. This structure resembles the multiple-choice knapsack problem (KP) (Sinha & Zoltners, 1979), as both problems involve selecting one item from each group under a shared budget. The key distinction is that KP assumes additive item values, whereas our coverage objective is non-additive: overlapping coverage sets create dependencies among policies. If the policies' coverage sets were disjoint, or, more generally, their contributions were additive, the formulation would reduce to a multiple-choice KP. 2) At the learning-optimization interface, TSFT follows the predict-then-optimize paradigm (Elmachtoub & Grigas, 2022): it first predicts task-wise performance using parametric models, converts these predictions into estimated coverage sets, and then optimizes the resulting surrogate problem. In contrast to decision-focused learning (Mandi et al., 2024), our performance models are trained with prediction-level losses rather than an end-to-end objective based on allocation quality or coverage regret. This distinction is particularly relevant for the thresholded coverage objective: prediction errors near the performance threshold ϵ, or on tasks that distinguish between competing allocations, can have a much larger impact than errors elsewhere. Decision-focused learning may therefore improve objective alignment, although differentiating through the discrete thresholding operation and the ILP remains challenging. 3) At the framework level, TSFT resembles the feedback structure of model predictive control (MPC) (Garcia et al., 1989): it estimates a surrogate model, optimizes an allocation, executes only a limited step, observes the resulting performance, and replans. In this view, the allocation vector plays the role of the planning state, while budget assignments serve as controls. Unlike conventional MPC, which typically controls a dynamical system with continuous states and actions, TSFT addresses a discrete allocation problem over policies and tasks. Accordingly, feedback is primarily used to correct uncertainty in the learned fine-tuning response, rather than to compensate for disturbances in physical dynamics.

Assumption. 1) *Quality of the pretrained policy:* TSFT assumes a pretrained policy that provides a meaningful foundation for adaptation. This assumption is consistent with the intended practical setting, where fine-tuning is used to adapt an already capable pretrained model, such as a foundation model in many domains. If the pretrained policy lacks capabilities relevant to the target contexts, adaptation becomes closer to learning each target task from scratch and may require substantially more data and computation than the fixed fine-tuning budget allows. Such scenarios therefore fall outside the intended scope of TSFT. Importantly, however, this assumption does not require the pretrained policy to exhibit uniformly strong performance across all target contexts. As shown in Table 2, TSFT can substantially expand limited target-context coverage (e.g., from 6.8% to 26.4% on CVRP), provided that the pretrained policy offers a sufficiently meaningful basis for adaptation. 2) *Efficient evaluation:* TSFT periodically evaluates policies across the context space to collect the data required for fitting the performance prediction models. Consequently, the framework assumes that these evaluations are sufficiently efficient so that they do not introduce prohibitive computational overhead during online optimization. This limitation can be mitigated by using cheaper evaluation strategies for model fitting, as demonstrated in Appendix F.5. More advanced methods, such as approximating the evaluation results through selective evaluation, constitute an interesting direction for future research. 3) *Structure of the context space:* By definition, a CMDP comprises a family of related, context-specific MDPs parameterized by a context vector drawn from a context space. We primarily consider well-structured context spaces, following conventions established in prior CRL work (Benjamins et al., 2023; Cho et al., 2024; Zhou et al., 2026). In contrast to these studies, we also consider a discrete context space in the LLM domain, where nine heterogeneous reasoning and coding benchmarks serve as contexts. In this setting, relationships among contexts are less explicit and less smoothly varying, allowing us to evaluate TSFT beyond conventional, strongly structured context spaces. When the context space lacks clear structure, methods that learn where to train through source-task grouping (e.g., based on task embeddings, or gradient-based task-affinity grouping methods) could complement our scope on how much to train and provide an orthogonal and important extension to TSFT in the future work.

C METHODOLOGY DETAIL

C.1 MDP INTERPRETATION

In this section, we present an alternative MDP interpretation of the studied problem. Under this view, the problem can be formulated as a model-based MDP and solved optimally via dynamic programming (DP). Although this approach can be incorporated into TSFT, it is less computationally efficient than the ILP-based formulation used in our main method, as demonstrated in Appendix F.4.

C.1.1 MDP FORMULATION

We frame the budget allocation in Eq. (1) as a sequential decision-making process over the fine-tuning dynamics. Given a pretrained policy, a policy class, an RL training algorithm, a set of source task distributions $\{X_{\mathcal{S}_1}, \ldots, X_{\mathcal{S}_N}\}$, and a total budget K, the goal is to decide *how much* budget should be assigned to each policy. Each decision step allocates one fixed budget unit to one policy, e.g., by running the given RL algorithm for a fixed number of samples or epochs on the corresponding source task set. Given the total budget K, the allocation induces a finite horizon of K decision steps. Although the final objective depends only on the resulting allocation vector rather than the action order, this sequential formulation provides a convenient way to search over feasible allocations under the budget constraint. Formally, we formulate this allocation process as a model-based MDP, specifically an undiscounted finite-horizon MDP with horizon K and discount factor $\gamma = 1$.

State Space: A state $s_t \in S$ at step t is defined by the allocation vector across policies, $s_t = (k_1^t, k_2^t, ..., k_N^t)$, where k_n^t denotes the cumulative number of budget units allocated to policy n up to step t. The state space is *discrete*, and the initial state corresponds to a zero allocation vector.

Action Space: An action $a_t \in [1, 2, ..., N, \emptyset]$ specifies which policy receives an additional budget unit at step t. Choosing $\emptyset$ corresponds to termination (i.e., early stopping), capturing cases where allocating further budget does not improve the final objective.

Transition Dynamics and Model: The state transition P is *deterministic* with respect to the allocation state. Specifically, selecting policy n at step t updates the allocation state by assigning one additional budget unit to that policy. For example, if the first policy is selected, the state transitions from $s_t = (k_1^t, k_2^t, ..., k_N^t)$ to $s_{t+1} = (k_1^t + 1, k_2^t, ..., k_N^t)$. The size of the budget unit determines the decision and transition granularity. This deterministic transition is defined over the abstract allocation state, not over the underlying stochastic training process. In practice, the parameter trajectory and realized performance of each policy after receiving additional training budget can be stochastic due to random initialization, data sampling, environment interaction, and the RL optimization procedure. We therefore use a model-based MDP: the allocation transition is deterministic, while the effect of training on task performance and coverage is estimated by a learned performance model.

Concretely, since the global coverage is not known a priori, we model the evolution of each policy's task-wise performance and thereby implicitly estimate its coverage set. For each policy-task pair (n, x), we fit a parametric performance model of the form:

$$\mathcal{F}(y) = L \pm Be^{-dy}, \tag{10}$$

where L, B, and d are learnable parameters, and y denotes the consumed training budget. The sign of B controls the direction of the performance shift, capturing both improvement and degradation during fine-tuning. We fit $\mathcal{F}_n^x(y)$ via non-linear least squares (NLLS), which predicts the performance of policy n on task x after consuming a given amount of budget y. Empirically, this fitting process can be performed in parallel over thousands of tasks within seconds.

Reward Function: The reward r_t is defined as the marginal improvement in global coverage. Given that specialized policies may exhibit overlapping coverage across the context space, we define the *global coverage* at state s_t as the measure of the union of individual coverage sets:

$$G(s_t) = \left| \bigcup_{n=1}^{N} \mathcal{C}(\theta^{k_n^t}_{X_{\mathcal{S}_n}}) \right|. \tag{11}$$

The reward at step t is then the marginal gain resulting from action a_t: $r_t = r(s_t, a_t) = G(s_{t+1}) - G(s_t)$. This formulation ensures that the reward is coupled across policies, naturally penalizing redundant optimization and incentivizing the expansion of the collective coverage boundary.

C.1.2 MDP SOLVING

The true optimal solution (i.e., Oracle) could, in principle, be obtained by training each policy to completion (using the full budget K) and solving Eq. (1) exactly via DP in a model-free MDP. However, in practice, $J(\theta^{k_n}_{X_{\mathcal{S}_n}}, x)$ is unknown during the planning phase. We therefore replace it with our model prediction $\mathcal{F}^x_n(\cdot)$, yielding a surrogate objective that we optimize within our framework. For clarity, we denote the resulting model-induced surrogate coverage objective by $\widehat{G}$. That is, $\widehat{G}(\cdot)$ is computed by replacing the unknown performance with the model prediction.

Based on the above model-based MDP formulation, we develop a discrete DP solver to derive its optimal allocation policy. To optimize the final objective, the solver applies *value iteration* on the surrogate objective, searching for the allocation trajectory that maximizes the global coverage upon exhaustion of the total budget K. Let $V^*(s_t)$ represent the optimal state value, defined as the maximum achievable global coverage from state s_t with remaining budget b_t. Our DP solver recursively computes this value following the Bellman optimality principle:

$$V^*(s_t) = \max\left(\widehat{G}(s_t), \max_{a_t} V^*(s_{t+1})\right), \tag{12}$$

where s_{t+1} is the deterministic next state induced by taking action a_t in state s_t. We omit a discount factor (i.e., $\gamma = 1$), as the objective is to maximize the final global coverage over a finite horizon. The optimal policy is obtained via backward induction over the state space. An implementation code snippet is provided in Appendix C.1.3. Our DP solver achieves global optimality and permutation invariance, as detailed in Remark 1. Note that this optimality is defined w.r.t. the model-based MDP under the surrogate objective, and does not necessarily carry over to the original objective in Eq. (1).

Remark 1 (DP Property)**.** *1) Global Optimality: The DP solver guarantees global optimality strictly for the full budget K. Intermediate allocations along the optimal trajectory do not necessarily constitute optimal solutions for smaller budget constraints. 2) Permutation Invariance: The final global coverage depends exclusively on the allocation vector $\mathcal{K}$. The specific sequence of actions taken to reach this distribution is commutative and irrelevant to the final objective.*

The computational complexity of DP solver is dominated by the size of the reachable state space $|S|$. Due to the combinatorial structure of the state representation and the total budget constraint K, the number of reachable states is mathematically upper bounded by the number of weak compositions of at most K units into N policies, i.e., $|S| \leq \binom{K+N}{N}$. For fixed N, this bound scales as $\mathcal{O}(K^N)$.

Remark 2 (Computational Complexity)**.** *The time and space complexities of the DP solver scale exponentially with the number of policies N, and polynomially with the total budget K.*

This curse of dimensionality motivates the exploration of more efficient strategies for solving the MDP. Inspired by the receding-horizon principle in model predictive control, we introduce *receding-horizon DP* (RH-DP), an h-step lookahead strategy that trades global optimality for improved tractability. RH-DP is characterized by a planning horizon h and an execution horizon e. Specifically, instead of solving the full-horizon DP over the entire remaining budget, RH-DP solves an h-step limited-horizon DP, executes the first $e \leq h$ allocation decisions, and then replans from the updated state. By bounding the complexity of each local planning phase to $\mathcal{O}(h^N)$, this approach reduces the overall computational burden while preserving adaptive replanning over the full budget.

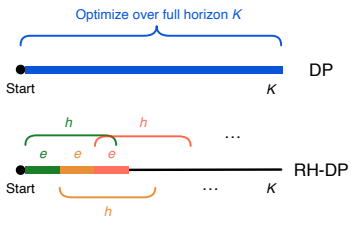


Figure 3: Illustration of RH-DP.

C.1.3 DYNAMIC PROGRAMMING

We provide a code snippet illustrating our DP implementation below.

```
# An example of DP implementation
@lru_cache(maxsize=None)
def dp_value_iter(self, state, steps_remain):
    current_reward = self.get_coverage(state)
```

```
    if steps_remain == 0:
        return current_reward, []

    possible_actions = []
    for id, s in enumerate(state):
        if s < self.max_epochs:
            new_state = list(state)
            new_state[id] += 1
            val, path = self.dp_value_iter(tuple(new_state), steps_remain
                                                    -1)
            possible_actions.append((val, id, path))

    # Optimization: Choose action with max future value
    best_future_val, best_action, best_path = max(possible_actions, key=
                                        lambda x: x[0])

    if best_future_val > current_reward:
        return best_future_val, [best_action] + best_path
    else:
        # No Action: early-stop as further training yields NO improvement
        return current_reward, []
```

C.2 Alternative Formulation

In addition to Eqs. (3)-(7), we present two alternative ILP formulations and a constraint-programming (CP) formulation. The original formulation uses $K+1$ binary variables $z_{n,k}$ to encode the allocation level of each policy. A natural way to potentially reduce the formulation size is to instead use a single integer variable for each policy. However, without additional structure (e.g., monotonicity), the coverage condition in Constraint (6) cannot be expressed linearly in these integer variables without introducing auxiliary variables. We first consider the case in which coverage is monotone with respect to the allocated budget: $\mathcal{C}(\theta^k_{X_{S_n}}) \subseteq \mathcal{C}(\theta^{k+1}_{X_{S_n}})$, $\forall n, k \in \{0, \dots, K-1\}$. Under this assumption, we define the minimum budget required for policy n to cover task x as:

$$T_{n,x} = \min \left\{ k \in \{0, \dots, K\} : x \in \mathcal{C}(\theta^k_{X_{S_n}}) \right\}, \tag{13}$$

where $T_{n,x} = +\infty$ if policy n cannot cover x within the available budget. In the model-based setting, $\mathcal{C}$ and $T_{n,x}$ are replaced by their surrogate counterparts induced by F^x_n. Under monotone coverage, this leads to the following two ILP formulations.

Threshold Formulation. We directly optimize the allocation variables $k_n \in \{0, \dots, K\}$ and introduce $c_{n,x} \in \{0,1\}$ to indicate whether policy n covers task x:

$$\max_{K,\mathbf{c},\mathbf{q}} \quad \sum_{x \in \mathcal{X}} q_x, \tag{14}$$

$$\text{s.t.} \quad \sum_{n=1}^{N} k_n \le K, \tag{15}$$

$$T_{n,x} c_{n,x} \le k_n \le (T_{n,x} - 1) + (K - T_{n,x} + 1) c_{n,x}, \qquad \forall (n,x) : T_{n,x} < +\infty, \tag{16}$$

$$q_x \le \sum_{n : T_{n,x} < +\infty} c_{n,x}, \qquad \forall x \in \mathcal{X}, \tag{17}$$

$$k_n \in \{0, \dots, K\}, \quad c_{n,x}, q_x \in \{0,1\}. \tag{18}$$

The two-sided constraint enforces $c_{n,x} = 1$ if and only if $k_n \ge T_{n,x}$. This formulation replaces the $N(K+1)$ binary variables $z_{n,k}$ with N integer variables. However, it introduces up to $N|\mathcal{X}|$ policy-task variables and is exact only when coverage is monotone.

Incremental Budget Formulation. Let $d_{n,k} = 1$ indicate that policy n receives at least k budget units, for $k \in \{1, \dots, K\}$, and define $d_{n,0} = 1$. The allocation is then $k_n = \sum_{k=1}^{K} d_{n,k}$, yielding:

$$\max_{\mathbf{d},\mathbf{q}} \quad \sum_{x \in \mathcal{X}} q_x, \tag{19}$$

$$\text{s.t.} \quad \sum_{n=1}^{N} \sum_{k=1}^{K} d_{n,k} \le K, \tag{20}$$

$$d_{n,k} \ge d_{n,k+1}, \qquad \forall n,\ k \in \{1, \dots, K-1\}, \tag{21}$$

$$q_x \le \sum_{n: T_{n,x} < +\infty} d_{n,T_{n,x}}, \qquad \forall x \in \mathcal{X}, \tag{22}$$

$$d_{n,k}, q_x \in \{0, 1\}. \tag{23}$$

This cumulative encoding eliminates policy-task variables $c_{n,x}$ and naturally represents training as accumulated budget. Its main drawbacks are the NK binary variables and ordering constraints it introduces, as well as the monotonicity requirement imposed by the threshold-based formulation.

Constraint Programming. In addition to ILP, we formulate the problem using the ELEMENT constraint supported by OR-Tools CP-SAT. For every policy n, allocation level k, and task x, define the constant $a^x_{n,k} = \mathbb{I}\Big(x \in \mathcal{C}(\theta^k_{X_{S_n}})\Big)$. We introduce an integer variable $k_n \in \{0, \dots, K\}$ for the budget allocated to policy n, and a binary variable $c_{n,x}$ indicating whether policy n covers task x under its selected allocation. The formulation is as follows:

$$\max_{\mathcal{K},\mathbf{c},\mathbf{q}} \quad \sum_{x \in \mathcal{X}} q_x, \tag{24}$$

$$\text{s.t.} \quad \sum_{n=1}^{N} k_n \le K, \tag{25}$$

$$c_{n,x} = \text{Element}\left(k_n; a^x_{n,0}, \dots, a^x_{n,K}\right), \qquad \forall n \in \{1, \dots, N\},\ x \in \mathcal{X}, \tag{26}$$

$$q_x \le \sum_{n=1}^{N} c_{n,x}, \qquad \forall x \in \mathcal{X}, \tag{27}$$

$$k_n \in \{0, \dots, K\}, \qquad c_{n,x}, q_x \in \{0, 1\}. \tag{28}$$

The ELEMENT constraint sets $c_{n,x}$ to the k_n-th entry of the coverage table $(a^x_{n,0}, \dots, a^x_{n,K})$. Unlike the threshold-based formulations, this formulation supports arbitrary non-monotone coverage sets. Compared with the original ILP, it replaces the $N(K+1)$ binary allocation variables with N integer variables, but introduces $N|\mathcal{X}|$ binary coverage variables and element constraints. It may therefore be advantageous when K is large relative to $|\mathcal{X}|$, but can become expensive for large context spaces, and its computational efficiency depends on CP-SAT's constraint propagation.

Overall, the threshold formulation can reduce the dependence on K when the context space is small or the policy-task coverage relation is sparse, while the incremental formulation can yield sparser coverage constraints under monotonicity. In practice, however, monotone coverage is difficult to guarantee. Consequently, the original ILP and the CP formulation are more general. Among them, the original ILP offers a direct representation that benefits from mature mixed-integer optimization techniques and, in our empirical evaluation, exhibits greater computational efficiency.

C.3 ALTERNATIVE MODEL

In addition to the parametric model, we further explore alternative modeling approaches.

Gaussian Process (GP). For each policy-task pair, we employ a GP model to implicitly represent a distribution over performance functions, providing a non-parametric alternative to the explicit exponential model. Specifically, for each policy-task pair (n, x), given the observed performance trajectory $\mathcal{Z}^x_n = \{(y_i, r_i)\}_{i=1}^m$, where y_i denotes the training step and r_i denotes the observed performance, we assume $r_i = f^x_n(y_i) + \epsilon_i,\ \epsilon_i \sim \mathcal{N}(0, \sigma^2_\epsilon)$. We place a GP prior over the latent function $f^x_n(\cdot) \sim \mathcal{GP}\big(0, k_\theta(\cdot, \cdot)\big)$, where k_θ is the covariance kernel. Given $\mathcal{Z}^x_n$, the GP yields a posterior

predictive distribution at any future training step y:

$$p\big(f_n^x(y) \mid \mathcal{Z}_n^x\big) = \mathcal{N}\big(\mu_n^x(y), \sigma_n^x(y)^2\big). \tag{29}$$

We use the posterior mean $\mu_n^x(y)$ as the predicted performance for policy n on task x at step y, thereby implicitly predicting the evolution of the corresponding coverage set.

Parametric Function with GP (PFGP). Although the parametric approach provides a smooth and interpretable estimate of the performance trajectory, fine-tuning, particularly in RL settings, often exhibits noise and instability. To account for this effect, we further combine the parametric model with a GP-based residual model. Specifically, the parametric function is used to model the base performance trend, while a GP is trained on the residuals to capture stochastic performance variations. For each policy-task pair (n, x), given the observed trajectory $\mathcal{Z}_n^x = \{(y_i, r_i)\}_{i=1}^m$, we first fit the parametric model $\mathcal{F}_n^x(y)$ and compute the residuals $\delta_i = r_i - \mathcal{F}_n^x(y_i)$. We then fit a GP to the residual data $\{(y_i, \delta_i)\}_{i=1}^m$, which gives a posterior mean $\mu_{\delta,n}^x(y)$ for the residual at any future training step y. The final prediction is obtained by adding this residual correction to the parametric trend: $\widehat{\mathcal{F}}_n^x(y) = \mathcal{F}_n^x(y) + \mu_{\delta,n}^x(y)$. In this way, PFGP preserves the smooth extrapolation structure of the parametric model, while using the GP residual component to capture local deviations and stochastic fluctuations around the fitted trend.

Other Parametric Forms. We further consider a power-law model, $\mathcal{F}(y) = L \pm B(y+1)^{-d}$ with $d > 0$; a logarithmic model, $\mathcal{F}(y) = L \pm B\log(1+y)$; and polynomial models of degree d, $\mathcal{F}(y) = \sum_{j=0}^{d} c_j y^j$, for $d \in \{2, 3, 4\}$ (quadratic, cubic, and quartic). We also evaluate a piecewise-linear model consisting of two independently fitted linear segments, with the breakpoint selected using the Bayesian information criterion (BIC). The power-law and logarithmic models impose smooth monotonic trends, whereas the polynomial and piecewise-linear models offer greater flexibility for modeling non-monotonic trajectories. A comprehensive discussion is provided in Appendix F.2.

D THEORETICAL ANALYSIS

We provide a theoretical analysis of the allocation optimality gap introduced by our TSFT framework. Recall that the objective is to maximize the global coverage under a total budget K:

$$G(\mathcal{K}) = \left| \bigcup_{n=1}^{N} \mathcal{C}\left(\theta^{k_n}_{\mathcal{X}_{S_n}}\right) \right|, \tag{30}$$

where $\mathcal{K} = (k_1, \ldots, k_N)$ denotes the allocation vector and k_n is the budget allocated to the n-th policy. Since the true task performance trajectory $J(\theta, x)$ is unknown during planning, TSFT relies on a predictive model $\mathcal{F}^x_n$ to construct a surrogate global coverage objective, denoted by $\widehat{G}(\mathcal{K})$.

Let $\mathcal{K}^0$ denote the allocation after the warmup stage. For example, when the warmup budget W is uniformly allocated across N policies and W is divisible by N, we have $\mathcal{K}^0 = (W/N, \ldots, W/N)$. The post-warmup feasible allocation set is defined as

$$\Omega_W = \left\{ \mathcal{K} \in \mathbb{Z}^N_{\geq 0} : \mathcal{K} \succeq \mathcal{K}^0,\ \|\mathcal{K}\|_1 \leq K \right\}, \tag{31}$$

where $\mathcal{K} \succeq \mathcal{K}^0$ denotes element-wise inequality. Both TSFT and Oracle-Warmup are constrained to optimize over this same post-warmup feasible allocation set.

Assumption 1 (Bounded Surrogate Coverage Error)**.** *There exists a constant $\delta_m \geq 0$ such that the surrogate coverage objective uniformly approximates the true coverage objective over the post-warmup feasible allocation set:*

$$\sup_{\mathcal{K} \in \Omega_W} \left| G(\mathcal{K}) - \widehat{G}(\mathcal{K}) \right| \leq \delta_m. \tag{32}$$

Assumption 2 (Surrogate Planning Accuracy)**.** *Let $\widehat{G}$ be the fixed surrogate objective used to evaluate allocations over Ω_W, and let $\mathcal{K}_{\mathrm{TSFT}} \in \Omega_W$ be the allocation returned by TSFT. We assume that the executed allocation is η_{alg}-optimal with respect to the surrogate objective:*

$$\widehat{G}(\mathcal{K}_{\mathrm{TSFT}}) \geq \max_{\mathcal{K} \in \Omega_W} \widehat{G}(\mathcal{K}) - \eta_{\mathrm{alg}}. \tag{33}$$

Theorem 2 (Performance Bound under Surrogate Model Error)**.** *Let $\mathcal{K}^*_{\mathrm{OW}} \in \arg\max_{\mathcal{K} \in \Omega_W} G(\mathcal{K})$ be the Oracle-Warmup allocation, and let $\mathcal{K}_{\mathrm{TSFT}}$ be the allocation returned by TSFT. Under Assumptions 1 and 2, we have*

$$G(\mathcal{K}^*_{\mathrm{OW}}) - G(\mathcal{K}_{\mathrm{TSFT}}) \leq 2\delta_m + \eta_{\mathrm{alg}}. \tag{34}$$

For a one-shot exact full-horizon planner that optimizes the same fixed surrogate objective and executes the resulting allocation exactly, $\eta_{\mathrm{alg}} = 0$, and the post-warmup gap is bounded by $2\delta_m$.

Proof. By adding and subtracting the surrogate objective, we obtain

$$\begin{aligned} G(\mathcal{K}^*_{\mathrm{OW}}) - G(\mathcal{K}_{\mathrm{TSFT}}) = &\left[G(\mathcal{K}^*_{\mathrm{OW}}) - \widehat{G}(\mathcal{K}^*_{\mathrm{OW}}) \right] + \left[\widehat{G}(\mathcal{K}^*_{\mathrm{OW}}) - \widehat{G}(\mathcal{K}_{\mathrm{TSFT}}) \right] \\ &+ \left[\widehat{G}(\mathcal{K}_{\mathrm{TSFT}}) - G(\mathcal{K}_{\mathrm{TSFT}}) \right]. \end{aligned} \tag{35}$$

By Assumption 1, the first and third terms are each upper bounded by δ_m. By Assumption 2,

$$\widehat{G}(\mathcal{K}^*_{\mathrm{OW}}) - \widehat{G}(\mathcal{K}_{\mathrm{TSFT}}) \leq \max_{\mathcal{K} \in \Omega_W} \widehat{G}(\mathcal{K}) - \widehat{G}(\mathcal{K}_{\mathrm{TSFT}}) \leq \eta_{\mathrm{alg}}. \tag{36}$$

Combining the three inequalities gives

$$G(\mathcal{K}^*_{\mathrm{OW}}) - G(\mathcal{K}_{\mathrm{TSFT}}) \leq 2\delta_m + \eta_{\mathrm{alg}}. \tag{37}$$

□

Remark on Coverage Error Assumption. Assumption 1 is stated directly at the level of the coverage objective. This is because G is a thresholded objective: small prediction errors in task performance can change the coverage indicator for tasks whose true performance lies close to the coverage threshold. Therefore, δ_m should be interpreted as the induced coverage-level error of the surrogate

model. In practice, online model re-estimation can reduce this error as more evaluation data are collected, although a monotonic decrease of δ_m is not guaranteed without additional assumptions.

Next, we decompose the gap between TSFT and the full Oracle. Assume $\|\mathcal{K}^0\|_1 \leq K$, so that $\Omega_W \subseteq \Omega_0$. Let the full feasible allocation set be

$$\Omega_0 = \left\{\mathcal{K} \in \mathbb{Z}_{\geq 0}^N : \|\mathcal{K}\|_1 \leq K\right\}. \tag{38}$$

The Oracle optimizes over Ω_0, whereas Oracle-Warmup and TSFT optimize over the restricted post-warmup feasible set $\Omega_W \subseteq \Omega_0$. Let $\mathcal{K}^* \in \arg\max_{\mathcal{K}\in\Omega_0} G(\mathcal{K})$ be the Oracle allocation.

Theorem 3 (Oracle Gap Decomposition)**.** *Define the warmup error as*

$$\mathcal{E}_{\text{warmup}} = G(\mathcal{K}^*) - G(\mathcal{K}^*_{\text{OW}}). \tag{39}$$

Then the total optimality gap of TSFT with respect to the full Oracle satisfies

$$G(\mathcal{K}^*) - G(\mathcal{K}_{\text{TSFT}}) \leq \mathcal{E}_{\text{warmup}} + 2\delta_m + \eta_{\text{alg}}. \tag{40}$$

Proof. By adding and subtracting the performance of Oracle-Warmup, we obtain the exact decomposition

$$G(\mathcal{K}^*) - G(\mathcal{K}_{\text{TSFT}}) = \underbrace{\left[G(\mathcal{K}^*) - G(\mathcal{K}^*_{\text{OW}})\right]}_{\text{warmup error}} + \underbrace{\left[G(\mathcal{K}^*_{\text{OW}}) - G(\mathcal{K}_{\text{TSFT}})\right]}_{\text{post-warmup planning error}}. \tag{41}$$

The first term is $\mathcal{E}_{\text{warmup}}$ by definition. The second term is bounded by Theorem 2. Therefore,

$$G(\mathcal{K}^*) - G(\mathcal{K}_{\text{TSFT}}) \leq \mathcal{E}_{\text{warmup}} + 2\delta_m + \eta_{\text{alg}}. \tag{42}$$

□

Interpretation. Theorem 3 decomposes the total gap to the full Oracle into two sources. The first term, $\mathcal{E}_{\text{warmup}}$, captures the loss induced by committing the initial warmup budget before model-based planning begins. Note that this term can be zero if the post-warmup state lies on an optimal trajectory (i.e., $\mathcal{K}^0 \preceq \mathcal{K}^*$). We further ablate its effect on global coverage when it introduces non-zero error (see Fig. 9 for the warmup sensitivity analysis). The second term, bounded by $2\delta_m + \eta_{\text{alg}}$, captures the post-warmup loss caused by surrogate model error and algorithmic approximation. Thus, TSFT approaches Oracle-Warmup when the surrogate coverage error is small and the planning procedure is accurate. It approaches the full Oracle when, in addition, the warmup allocation does not substantially restrict the optimal final allocation.

Effect of Surrogate Error. Surrogate prediction error does not necessarily induce allocation error. If the surrogate objective preserves the ranking of high-quality feasible allocations, TSFT can still recover a strong post-warmup allocation even when $\widehat{G} \neq G$. Conversely, small prediction errors can affect the selected allocation when multiple feasible allocations have similar true coverage or when many tasks lie close to the threshold ϵ.

E EXPERIMENT DETAIL

We conduct experiments on a machine equipped with NVIDIA GeForce RTX 4090 GPUs and an AMD Ryzen Threadripper PRO 7975WX CPU for combinatorial optimization, on a machine with NVIDIA V100 GPUs and an Intel Xeon E5-2670 CPU for continuous control, and on a cluster with 16 NVIDIA H200 GPUs for LLM fine-tuning.

E.1 BASELINE

Here, we detail the baseline implementations of Adaptive and LinUCB. 1) *Adaptive:* We maintain a weight w_n for each policy n, initialized uniformly. At each decision-making step, the weights are converted into allocation probabilities using a temperature-scaled softmax, and the execution budget E is allocated across policies by sampling from the resulting multinomial distribution. After training, we evaluate the updated checkpoints and update each weight based on the coverage improvement per allocated budget unit. In this way, Adaptive prioritizes policies that have recently expanded their coverage set more efficiently. We set the temperature to 0.05 in all experiments. 2) *LinUCB:* We formulate the budget allocation as a contextual multi-armed bandit problem (Li et al., 2010; Chu et al., 2011), where each policy corresponds to an arm. For each policy n, we maintain a ridge-regression estimator with a design matrix A_n and a response vector b_n. At each decision-making step, LinUCB constructs a 5-dimensional context vector z_n for each policy, which includes a bias term, normalized training progress, its squared value, the current coverage ratio, and recent coverage momentum. The estimated parameter for policy n is given by $\hat{\delta}_n = A_n^{-1} b_n$. LinUCB then computes the upper-confidence score

$$s_n = \hat{\delta}_n^\top z_n + \alpha \sqrt{z_n^\top A_n^{-1} z_n}, \tag{43}$$

where $\alpha = 1.0$ controls the exploration strength. The policy n^* with the largest score is selected and allocated one budget unit. After training, we evaluate the updated checkpoint and compute the reward as the monotonic coverage improvement over the historical best coverage of the selected policy, normalized by the allocated budget. This reward design avoids penalizing temporary performance drops during fine-tuning. Finally, the statistics A_{n^*} and b_{n^*} of the selected policy n^* are updated using the observed context and reward.

E.2 POLICY TRAINING

Combinatorial Optimization. We adopt POMO (Kwon et al., 2020) as the policy network, a strong attention-based neural solver for routing problems. We largely follow the training setup in (Kwon et al., 2020). Specifically, we use the Adam optimizer with a learning rate of 1×10^{-4}, a weight decay of 1×10^{-6}, and a batch size of 64. The policy is pretrained for $5{,}000$ epochs over the entire context space, with each epoch containing $10{,}000$ training instances. Both the problem size and the POMO size are set to 100.

Continuous Control. For CartPole and Ant experiments, we used PPO (Schulman et al., 2017) implemented by Stable Baseline3 (Raffin et al., 2021) and used the default hyperparameters, including a learning rate of 3×10^{-4}, `n_steps`$= 2048$, batch size 64, discount factor 0.99, GAE parameter 0.95, clipping parameter 0.2, entropy coefficient 0, and a value function loss coefficient 0.5. The policy and value networks share an MLP backbone whose hidden sizes are environment-specific: $[64, 64]$ for CartPole 3D and $[256, 256]$ for Ant 2D (the latter to accommodate Ant's 111-dimensional observation). The pretraining run targets 5M environment steps over the CartPole grid and 1M over the Ant grid. Fine-tuning on each source region uses the same algorithm and hyperparameters across all N regions; only the source task subset and the budget allocation $\mathcal{K}$ vary across runs. For Meta-World experiments, we used MOORE (Hendawy et al., 2024) as the base multi-task RL algorithm. We kept all MOORE hyperparameters identical to those used in the original implementation. We evaluate on the Meta-World MT50 benchmark, which contains 50 manipulation tasks. The model is first pretrained for 50M environment steps on all 50 tasks, and then fine-tuned using a total budget of 50M environment steps. During fine-tuning, the same algorithm and hyperparameter configuration are used across all task groups; only the task grouping and the budget allocation $\mathcal{K}$ vary across runs.

LLM Fine-Tuning. We perform reinforcement fine-tuning on Qwen3-4B-Base using GRPO (Shao et al., 2024), implemented in VeRL (Sheng et al., 2025) with a vLLM-based rollout backend (Kwon et al., 2023). All four specialized policies (DAPO-17K, MATH, GSM8K, CodeContests+) share the same training configuration, only the training corpus changes across runs. The full hyperparameter list is given in Table 4. After every GRPO update, the checkpoint is evaluated on each benchmark in the context space (capped at 100 problems per benchmark to keep online evaluation tractable).

Table 4: GRPO hyperparameters used for LLM fine-tuning.

Hyperparameter	**Value**
Parameter precision	BF16 (mixed precision)
Group size (rollouts per prompt)	8
KL loss coefficient	0 (KL loss disabled)
KL penalty in reward	Disabled
Advantage normalization by group std	Disabled
Entropy coefficient	0
Optimizer	AdamW
Learning rate	2×10^{-6}
Train batch size	128
PPO mini-batch size	64
PPO micro-batch size per GPU	16
Max prompt length	2,048 tokens
Max response length	8,192 tokens

E.3 Context Space

Here, we detail the data generation, as well as the settings of the context space and source task sets.

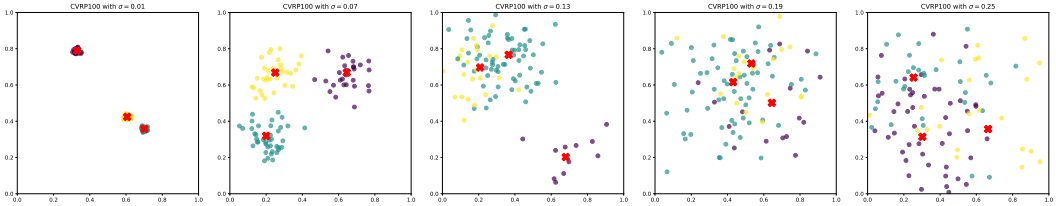


Figure 4: Examples of node distributions obtained by varying the covariance σ of a GMM.

Combinatorial Optimization. For CVRP, we follow the common settings in the literature (Kool et al., 2019). The demand of each customer node is randomly sampled from a discrete uniform distribution over $\{1, 2, \ldots, 9\}$. Before being fed into the network, each demand is normalized by the vehicle capacity. The context space is defined by two dimensions: vehicle capacity and node distribution. The vehicle capacity ranges from 10 to 400 with a step size of 10. We control the node distribution through the covariance of a Gaussian mixture model (GMM). Specifically, node locations are generated from a GMM with three clusters, where the cluster centers are sampled uniformly from $[0.2, 0.8]^2$. Each cluster uses a diagonal covariance matrix $\Sigma = \mathrm{diag}(\sigma^2, \sigma^2)$, and we vary σ to control the spatial dispersion of customer nodes along the two dimensions. Nodes sampled outside the unit square $[0, 1]^2$ are rejected and resampled. A visualization example is provided in Fig. 4. We construct five source task sets, each containing 49 tasks centered around a reference task (see Fig. 5). The reference tasks for $X_{S_1}, X_{S_2}, X_{S_3}, X_{S_4}, X_{S_5}$ are $(0.07, 100)$, $(0.07, 300)$, $(0.13, 200)$, $(0.19, 100)$, and $(0.19, 300)$, respectively. We use $\{X_{S_1}, X_{S_3}, X_{S_5}\}$ and $\{X_{S_1}, X_{S_2}, X_{S_4}, X_{S_5}\}$ as the source task sets for the $N = 3$ and $N = 4$ cases in Table 2, respectively.

For CVRPTW, we extend CVRP by introducing time-window constraints. Specifically, each customer node v_i is associated with a time window $[f_i, g_i]$ and a service time m_i. A vehicle must start serving customer v_i within the interval $[f_i, g_i]$. If the vehicle arrives earlier than f_i, it must wait until f_i before service can begin. All vehicles must return to the depot v_0 no later than g_0.

We set the depot time window to $[f_0, g_0] = [0, 4]$ and assign zero service time to the depot. The time window for each customer node v_i is then generated as follows: 1) sample the time-window center $\gamma_i \sim U(f_0 + d_{0i}, g_0 - d_{i0} - m_i)$, where $d_{0i} = d_{i0}$ denotes the distance, or equivalently the travel time, between v_0 and v_i; 2) sample the time-window half-width w_i uniformly at random from $[m_i/2, g_0/3]$; and 3) set the customer time window as $[f_i, g_i] = [\max(f_0, \gamma_i - w_i), \min(g_0, \gamma_i + w_i)]$. The context space is defined along two dimensions: vehicle capacity and time window (TW) tightness. Vehicle capacity ranges from 10 to 400 with a step size of 10. We control time window tightness by varying the service time m_i. Specifically, m_i is varied from 0.04 to 1.00 with a step size of 0.04. Larger service times implicitly correspond to tighter time windows. Similar to CVRP, we construct five source task sets, each containing 49 tasks centered around a reference task (see Fig. 5). The reference tasks for $X_{S_1}, X_{S_2}, X_{S_3}, X_{S_4}, X_{S_5}$ are $(0.28, 100)$, $(0.28, 300)$, $(0.52, 200)$, $(0.76, 100)$, and $(0.76, 300)$, respectively. We use $\{X_{S_1}, X_{S_3}, X_{S_5}\}$ and $\{X_{S_1}, X_{S_2}, X_{S_4}, X_{S_5}\}$ as the source task sets for the $N = 3$ and $N = 4$ cases in Table 2, respectively.

Continuous Control. For CartPole, we construct a 3-dimensional context space along (pole length, cart mass, pole mass). Each axis is discretized into 10 values evenly spaced over $[0.1, 10]\times$ the CARL defaults $(\ell_0, m_{c,0}, m_{p,0}) = (0.5, 1.0, 0.1)$, giving the explicit grids $\{0.05, 0.6, 1.15, 1.7, \ldots, 5.0\}$ for pole length, $\{0.1, 1.2, 2.3, \ldots, 10.0\}$ for cart mass, and $\{0.01, 0.12, 0.23, \ldots, 1.0\}$ for pole mass. Source task sets correspond to $N = 5$ axis-aligned $3^3 = 27$-context regions: four extremal corners and a geometric center. These regions are mutually disjoint and together cover $5 \times 27 = 135$ of the $1{,}000$ context grid points; the remaining 865 contexts serve as held-out targets that the fine-tuned policies must generalize to.

For Ant, we construct a 2-dimensional context space along (gravity, friction). The two axes are discretized into 25 values for gravity ($\{1.96, 2.69, \ldots, 19.6\}$ m/s^2, i.e. $[0.2, 2.0] \times g_0$) and 40 values for friction ($\{0.2, 0.246, \ldots, 2.0\}$, i.e. $[0.2, 2.0] \times \mu_0$), giving a $25{\times}40 = 1{,}000$-context grid. Source task sets correspond to $N = 5$ axis-aligned $5{\times}4 = 20$-context rectangles: four corners and a center. These five regions span 100 of the $1{,}000$ grid points; the remaining 900 contexts are held-out targets. The square and source regions are visualized in Fig. 5.

Meta-World MT50 contains 50 robotic manipulation tasks, each corresponding to a distinct task context. In MOORE, each task is initially represented by a one-hot encoding. We instead use the (K)-dimensional task-specific expert weights from the pretrained policy as the task context, since they provide a learned continuous representation of how MOORE combines experts for each task. We project the K-dimensional expert weights of the 50 tasks into a 2D space using PCA, and annotate each point with its task description, as shown in Fig. 5.

LLM Fine-Tuning. We conduct experiments on a discrete context space comprising nine benchmarks across math reasoning, code generation, and general reasoning: DAPO-17K, MATH-500, GSM8K, CodeContests+, AIME 2024, AIME 2025, Minerva Math, MBPP, and BigCodeBench. We use benchmark-specific performance thresholds ϵ, set to [0.75, 0.10, 0.10, 0.27, 0.36, 0.30, 0.70, 0.20, 0.90] for these benchmarks, respectively.

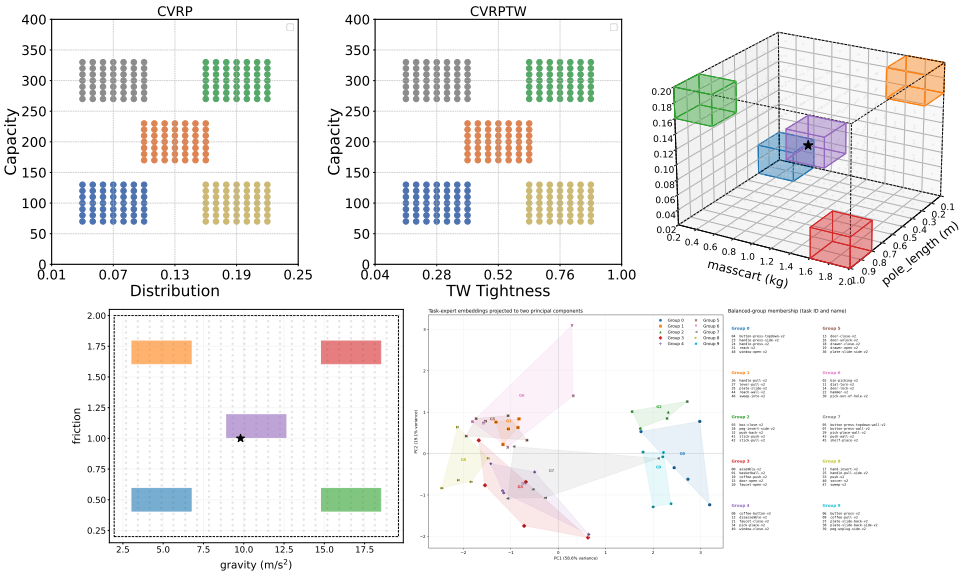


Figure 5: Visualization of the context spaces and corresponding source task sets for CVRP, CVRPTW, CartPole, Ant, and Meta-World, respectively. The task embeddings in Meta-World are the K-dimensional expert weights associated with each task from the pretrained model. We use PCA to reduce them to two dimensions for visualization.

F Additional Result

F.1 Full Result

We conduct experiments with 5 seeds for CartPole, and 3 seeds for Ant, with the full results reported in Tables 5 and 6. For combinatorial optimization and LLM fine-tuning, we follow the common convention of reporting results from a single seed (Kool et al., 2019; Kwon et al., 2020; Yu et al., 2025; Limozin et al., 2026), due to their relatively stable performance or high computational cost. For Meta-World, we use a single seed in the main experiments and additionally evaluate robustness by running 5 random seeds on a smaller setting with $N = 5$ and $K = 25$. We summarize the results in Fig. 6, where error bars denote the standard deviation across seeds. We observe that MTL can be unstable when trained over a large context space, unless equipped with sufficient model capacity and state-of-the-art MTL algorithms. For example, in Ant, allocating additional budgets to MTL does not necessarily improve task coverage. In contrast, TSFT generally performs robustly across these diverse settings. Moreover, we note that the large discrepancy between 93.4% and 0.0% coverage across Ant trials is primarily an artifact of the hard coverage threshold ($\epsilon = 6.9$). Specifically, the pretrained policies from the three seeds achieve mean returns of 6.917, 6.825, and 6.858, respectively. Therefore, 0% coverage does not indicate a failure to learn an effective policy, but rather that the policy narrowly misses the predefined threshold.

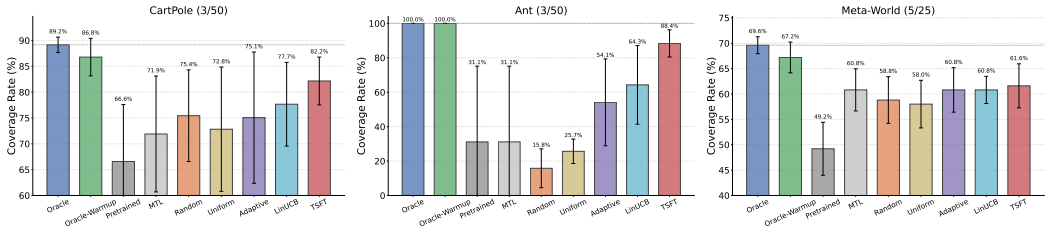


Figure 6: Coverage rate comparison on continuous control. Bars report mean coverage, and error bars denote standard deviation.

Table 5: Full Results for CartPole.

	Trial 1			**Trial 2**			**Trial 3**			**Trial 4**			**Trial 5**		
	3/50	3/100	4/100	3/50	3/100	4/100	3/50	3/100	4/100	3/50	3/100	4/100	3/50	3/100	4/100
Oracle	86.5%	86.8%	90.0%	89.7%	91.2%	91.5%	89.9%	90.5%	90.6%	90.0%	90.0%	90.2%	89.7%	90.0%	90.1%
Oracle-Warmup	82.1%	82.4%	89.7%	83.7%	85.5%	91.5%	89.9%	90.5%	90.6%	88.5%	89.0%	90.0%	89.7%	89.9%	90.0%
Pretrained	53.5%	53.5%	53.5%	83.1%	83.1%	83.1%	68.2%	68.2%	68.2%	60.5%	60.5%	60.5%	67.6%	67.6%	67.6%
MTL	63.7%	73.0%	73.0%	83.1%	83.1%	83.1%	84.6%	88.4%	88.4%	60.5%	60.5%	60.5%	67.6%	67.6%	67.6%
Random	60.2%	65.7%	79.5%	78.4%	77.5%	**90.2%**	82.0%	87.6%	77.7%	75.6%	64.8%	76.9%	81.0%	79.4%	87.5%
Uniform	55.9%	62.7%	80.4%	78.3%	63.9%	78.6%	**86.4%**	87.4%	**89.6%**	65.6%	40.0%	76.4%	77.9%	73.1%	**88.9%**
Adaptive	52.9%	59.8%	83.7%	78.4%	78.4%	89.0%	81.4%	83.5%	88.5%	77.8%	77.8%	86.6%	84.8%	84.8%	88.5%
LinUCB	68.4%	70.3%	70.3%	**88.8%**	**89.2%**	89.2%	74.0%	81.6%	81.6%	74.2%	**87.8%**	**87.8%**	82.9%	84.2%	84.2%
TSFT	**74.3%**	**74.5%**	**86.2%**	83.3%	83.3%	89.7%	82.2%	**89.8%**	85.3%	**85.3%**	85.3%	87.7%	**85.7%**	**87.0%**	**88.9%**

Table 6: Full Results for Ant and Meta-World.

	Ant Trial 1			**Ant Trial 2**			**Ant Trial 3**			**Meta-World 5/25**				
	3/50	3/100	4/100	3/50	3/100	4/100	3/50	3/100	4/100	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
Oracle	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	68.0%	72.0%	70.0%	70.0%	68.0%
Oracle-Warmup	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	62.0%	70.0%	68.0%	68.0%	68.0%
Pretrained	93.4%	93.4%	93.4%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	48.0%	48.0%	52.0%	56.0%	42.0%
MTL	**93.4%**	93.4%	93.4%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	**62.0%**	60.0%	**64.0%**	64.0%	54.0%
Random	21.6%	33.1%	60.2%	0.0%	74.2%	73.4%	25.8%	20.3%	3.5%	54.0%	58.0%	60.0%	**66.0%**	56.0%
Uniform	21.1%	28.0%	51.3%	20.2%	**100.0%**	84.2%	35.7%	30.9%	58.7%	50.0%	58.0%	60.0%	60.0%	**62.0%**
Adaptive	33.1%	86.9%	94.6%	89.6%	89.6%	**100.0%**	39.7%	87.4%	81.2%	54.0%	62.0%	60.0%	**66.0%**	**62.0%**
LinUCB	**93.4%**	**96.7%**	96.7%	61.8%	71.9%	68.7%	37.6%	96.0%	65.4%	58.0%	58.0%	62.0%	64.0%	**62.0%**
TSFT (DP)	77.5%	93.0%	**98.5%**	**91.6%**	**100.0%**	**100.0%**	**96.0%**	**100.0%**	**100.0%**	58.0%	**68.0%**	60.0%	64.0%	58.0%

F.2 A Comprehensive Study of Performance Model

TSFT employs an exponential model to predict performance, which implicitly assumes that fine-tuning trajectories are approximately monotonic. While RL fine-tuning is stochastic and no surrogate model can perfectly predict the entire training trajectory, **effective budget allocation in TSFT does not require highly accurate performance prediction**. Instead, it is often sufficient to capture the coarse trend of the learning curve. This observation motivates our monotonicity assumption and the use of a simple parametric model that emphasizes the global trajectory rather than short-term fluctuations. Below, we provide empirical evidence and discussion supporting this design choice.

First, *the monotonic model remains effective even when the actual training trajectories are noisy and locally non-monotonic*. Across evaluated domains, the observed RL learning curves exhibit substantial fluctuations. We quantify this phenomenon by uniformly sampling either 5 or 10 points from each training trajectory and measuring its monotonicity (Table 7). With 10 sampled points, nearly all trajectories exhibit local non-monotonic behavior. With only 5 sampled points, the measured monotonicity increases because coarse sampling filters out high-frequency noise. Despite these local deviations, TSFT consistently achieves strong improvements, indicating that a simple monotonic model is sufficient to capture the broader trend required for effective budget allocation.

Table 7: Approximate Monotonicity Rate.

	CVRP	CVRPTW	CartPole	Ant
5 Points	30.2%	30.7%	28.2%	3.3%
10 Points	1.9%	0.8%	8.3%	0.0%

Second, *monotonicity provides a simple and robust inductive bias that reduces overfitting under limited observations*. We compare three monotonic models (Exponential, Power-Law, and Logarithmic) with four non-monotonic models (Quadratic, Cubic, Quartic, and Piecewise) on the relatively monotonic CVRP domain and the more challenging, non-monotonic Ant domain. We evaluate each model in terms of fitting error on the observed data, prediction error on future training points, and the resulting coverage achieved by TSFT (Table 8). Overall, the monotonic models perform consistently well across both domains. While some non-monotonic models remain competitive on Ant, they perform noticeably worse on CVRP. Moreover, although the more flexible models often achieve lower in-distribution (ID) fitting error, they generally incur substantially larger out-of-distribution (OOD) prediction error, suggesting that they overfit the observed noise and produce unstable extrapolations.

Table 8: Comparison of different parametric models. We report the TSFT coverage and the average in-distribution (ID) and out-of-distribution (OOD) mean squared errors (MSEs).

CVRP 3/100					**Ant 3/50**				
Type	**Model**	**Coverage**	**Avg. ID MSE**	**Avg. OOD MSE**	**Type**	**Model**	**Coverage**	**Avg. ID MSE**	**Avg. OOD MSE**
Monotonic	Exponential	18.7%	0.027682	0.103628	Monotonic	Exponential	88.4%	0.000694	0.001309
	Power-Law	19.7%	0.065287	0.192049		Power-Law	88.4%	0.000702	0.001048
	Logarithmic	19.0%	0.063002	0.179515		Logarithmic	90.7%	0.000760	0.001076
Non-monotonic	Quadratic	8.1%	0.027072	0.565207	Non-monotonic	Quadratic	85.2%	0.000625	0.012584
	Cubic	7.1%	0.015860	0.585428		Cubic	89.7%	0.000628	1.969833
	Quartic	7.4%	0.015324	11.138788		Quartic	81.1%	0.000475	11.286251
	Piecewise	15.3%	0.015488	0.147162		Piecewise	70.5%	0.000514	0.010115

Third, *explicitly modeling high-frequency fluctuations provides limited practical benefit*. Beyond purely parametric models, we also evaluate a hybrid parametric Gaussian process model (PFGP), in which the parametric component captures the global trend, while a non-parametric Gaussian process models residual fluctuations, such as sudden performance jumps. We conduct experiments on the 3/100 CVRP setting by fitting models to data collected from policies trained on the source task set X_{S_1}. As shown in Fig. 7, our parametric model consistently achieves higher recall than GP across future budget units, indicating that the simple exponential form provides more reliable extrapolation of fine-tuning trajectories. In contrast, the GP baseline exhibits unstable long-horizon predictions and large uncertainty when extrapolating beyond the observed budget range, which leads to more false negatives in the predicted coverage set. PFGP improves over GP by using the parametric function as a global trend and modeling only the residual variation, and its recall is often close to our parametric model. However, it does not provide a clear advantage over the parametric model, while incurring a significantly higher computational cost due to GP fitting. Therefore, we do not adopt PFGP in our main experiments. The bottom panels further confirm our observation: our

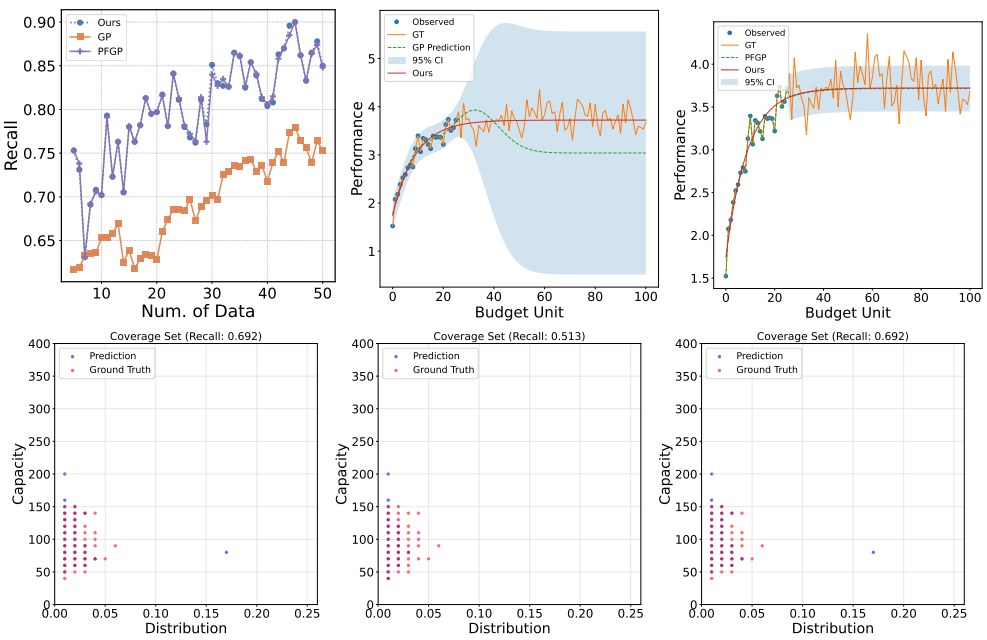


Figure 7: *Top left panel:* Comparison of recall rates of coverage sets. *Top right panels:* Comparison of fitted models on a single task. *Bottom panels:* Predicted coverage sets from three modeling approaches (i.e., Ours, GP, and PFGP) after 50 additional budget units. **Blue**, **red**, and **dark red** points denote false positives (FP), false negatives (FN), and true positives (TP), respectively.

model recovers a substantially larger portion of the ground-truth coverage set than GP, whereas PFGP achieves comparable recall without delivering a meaningful improvement.

Based on the above evidence, TSFT adopts a simple parametric model to capture the global trend of fine-tuning. Nevertheless, we acknowledge that the exponential model is not universally applicable. When the underlying fine-tuning trajectory exhibits strongly non-monotonic global behavior, the model may produce suboptimal budget allocations. Promising directions for future work include uncertainty-aware exploration strategies and more expressive surrogate models (e.g., neural networks) with stronger extrapolation capabilities.

F.3 Sensitivity Analysis

We tune several key hyperparameters, including the performance threshold ϵ, the number of policies N, the warmup budget W, and the execution budget E. The tuning procedure is based on a grid search over predefined ranges selected according to computational feasibility and empirical robustness.

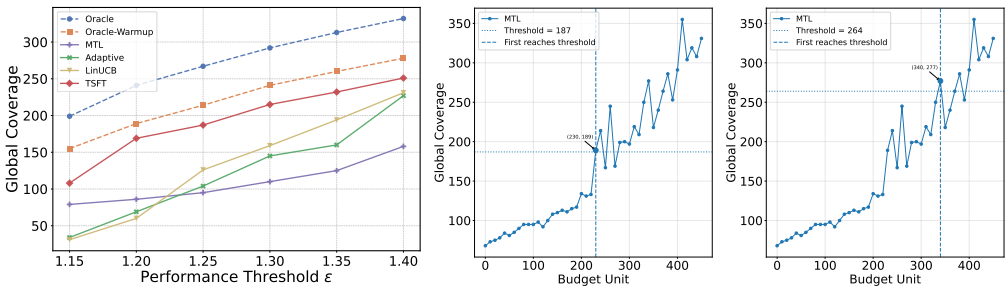


Figure 8: *Left panel:* Sensitivity test of performance threshold ϵ. *Right panels:* Sample efficiency comparison between TSFT and MTL on CVRP. We use the best TSFT results reported in Table 2 as thresholds, namely 187 and 264 for budgets of 100 and 150 units, respectively.

Performance Threshold. We conduct a sensitivity analysis on the performance threshold ϵ in the 3/100 CVRP setting, varying ϵ from 1.15% to 1.4% with increments of 0.05%. As shown in Fig. 8, the global coverage of all methods increases as ϵ becomes larger, since a looser threshold allows more tasks to be counted as covered. TSFT achieves competitive and stable performance over a wide range of ϵ, consistently outperforming standard baselines.

Number of Policies. We conduct a scalability test on the CVRP setting with a fixed budget of $K = 150$, while varying the number of policies from $N \in \{3, 4, 5, 6, 7\}$. The results in Fig. 9 show that TSFT scales substantially better than the baseline methods as the number of policies increases. Most baselines exhibit a clear performance drop as N becomes larger, suggesting that they struggle to effectively allocate the fixed budget across an increasing number of policies. Although LinUCB is competitive when N is small, its performance deteriorates sharply for larger N. Overall, these results demonstrate that TSFT is more effective at managing the increased allocation complexity induced by a larger policy portfolio. At the same time, they also reveal a fundamental trade-off: when the total budget is fixed, an excessively large N eventually reduces performance because each policy receives a smaller specialization budget (i.e., budget dilution). Consequently, TSFT would be expected to achieve higher coverage if the per-policy specialization budget were held constant.

Warmup Budget. We conduct a sensitivity analysis on the warmup budget W in the 3/100 and 3/150 CVRP settings, varying the warmup budget per policy from 3 to 15. As shown in Fig. 9, TSFT generally achieves stronger performance with relatively small warmup budgets, whereas an excessively large warmup budget may reduce the budget available for subsequent adaptive specialization, potentially leading to degraded global coverage. Note that the warmup budget introduces suboptimality only when it allocates more budget units to some policy than required by the optimal allocation under the true objective.

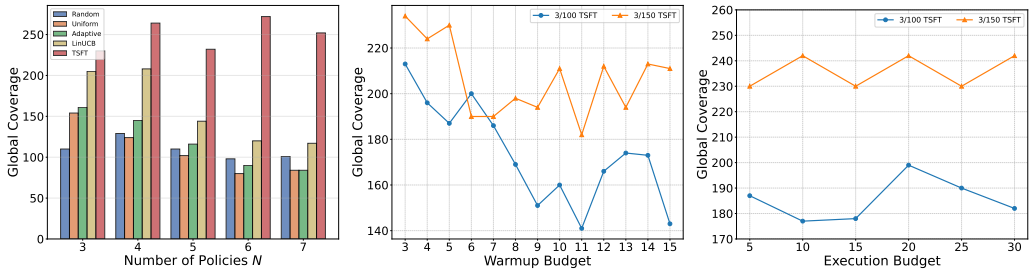


Figure 9: *Left panel:* Performance comparison under different number of policies. *Middle panel:* Sensitivity test of warmup budget. *Right panel:* Sensitivity test of execution budget.

Execution Budget. We conduct a sensitivity analysis on the execution budget E in the 3/100 and 3/150 CVRP settings, varying the execution budget unit from 5 to 30. As shown in Fig. 9, TSFT remains relatively stable across different execution budget settings, indicating that the method is not overly sensitive once a reasonable execution budget is used. This also suggests that TSFT may further robustly benefit from adaptive budget allocation without requiring extensive tuning of E.

Source Task Set. A substantial body of prior CRL research focuses on where to train (e.g., how to select representative source tasks). Our work addresses a complementary question of how much to fine-tune each given region. Nevertheless, We conduct additional experiments in the 3/100 CVRP setting using different source-task configurations to investigate TSFT's sensitivity to the choice of source-task set. Specifically, we define the five candidate regions illustrated in Fig. 5 and evaluate different selections of three regions. As shown in Table 9, TSFT remains effective across the tested configurations and consis-

Table 9: Sensitivity Test of Source Task Set.

CVRP 3/100	Adaptive	LinUCB	TSFT (DP)
$(X_{S_1}, X_{S_3}, X_{S_5})$	10.4%	12.6%	**18.7%**
$(X_{S_1}, X_{S_2}, X_{S_3})$	7.1%	8.1%	**11.4%**
$(X_{S_2}, X_{S_3}, X_{S_4})$	6.4%	6.1%	**12.3%**
$(X_{S_3}, X_{S_4}, X_{S_5})$	9.0%	5.7%	**16.9%**
$(X_{S_1}, X_{S_2}, X_{S_5})$	10.5%	11.7%	**21.8%**
$(X_{S_1}, X_{S_4}, X_{S_5})$	7.4%	7.3%	**18.4%**
$(X_{S_2}, X_{S_3}, X_{S_5})$	6.3%	9.4%	**17.0%**
$(X_{S_2}, X_{S_4}, X_{S_5})$	6.9%	9.8%	**20.1%**

tently identifies high-quality budget allocations, demonstrating that our TSFT is largely robust to different choices of source tasks.

F.4 Computational Cost

We provide a detailed comparison of the computational costs of the ILP- and DP-based methods in Table 10. The exact DP solver scales poorly with the number of policies N, quickly becoming impractical as N increases. Although RH-DP substantially reduces both runtime and memory consumption, it remains computationally prohibitive for large-scale planning over larger policy sets. In contrast, ILP solves the problem exactly while remaining computationally efficient. Unlike DP, which explicitly enumerates and caches nearly all reachable allocation states, ILP provides a compact formulation that enables the solver to exploit LP relaxations and branch-and-bound pruning to eliminate large portions of the search space. Note that the slight non-monotonicity in ILP runtime arises because branch-and-bound complexity depends on instance-specific structure and pruning effectiveness rather than solely on N and K. Consequently, ILP is often faster in practice, though both formulations remain exponential in the worst case.

Table 10: Computational cost comparison of TSFT under different numbers of policies N and total budgets K. Each entry reports runtime and peak memory usage per decision-making step.

Setting	Method	Runtime / Memory					
Varying the number of policies N with fixed budget $K = 100$							
$K = 100$		$N = 2$	$N = 3$	$N = 4$	$N = 5$	$N = 6$	$N = 7$
	TSFT (DP)	0.04s / 2.5MB	2.1s / 47.8MB	47.0s / 771.8MB	642.3s / 9476.2MB	–	–
	TSFT (RH-DP)	0.02s / 1.0MB	0.4s / 8.4MB	3.3s / 65.2MB	25.2s / 406.4MB	136.8s / 2186.9MB	717.2s / 10418.9MB
	TSFT (ILP)	0.08s / 2.2MB	0.2s / 6.1MB	0.5s / 8.7MB	0.5s / 9.0MB	0.5s / 10.3MB	0.4s / 11.0MB
Varying the total budget K with fixed number of policies $N = 3$							
$N = 3$		$K = 50$	$K = 100$	$K = 150$	$K = 200$	$K = 250$	$K = 300$
	TSFT (DP)	0.11s / 2.8MB	2.1s / 47.8MB	11.1s / 228.3MB	34.3s / 673.1MB	78.7s / 1580.2MB	165.1s / 3197.5MB
	TSFT (RH-DP)	0.08s / 2.3MB	0.4s / 8.4MB	0.6s / 13.7MB	0.9s / 20.0MB	1.2s / 25.0MB	1.5s / 32.6MB
	TSFT (ILP)	0.09s / 2.2MB	0.2s / 6.1MB	0.2s / 11.4MB	0.5s / 11.4MB	0.2s / 11.4MB	0.2s / 11.4MB

F.5 Cheap Evaluation

TSFT periodically evaluates policies across the context space. In our experiments, the total evaluation time is on the order of hours, which is substantially smaller than the days required for policy training. Nevertheless, exhaustive evaluation may be impractical in domains with expensive evaluations. To mitigate this, we may reduce the online evaluation cost required for model fitting by using fewer validation samples or evaluating less frequently with a larger execution interval. As shown in Table 11, using one-tenth as many samples for online evaluation yields comparable coverage to the original setting. Final global coverage is still calculated using the full dataset. More advanced methods, such as approximating the evaluation results through selective evaluation, constitute an interesting direction for future research.

Table 11: Results for Cheap Evaluation.

TSFT (DP)	CVRP 3/100	CVRP 3/150
Original	18.7%	23.0%
Reduced Evaluation (10x)	18.9%	22.3%

F.6 Visualization

We visualize the performance heatmap of each policy to illustrate task specialization. Specifically, we consider the final policies in the 5/150 CVRP setting, where specialization is guided by TSFT. Since TSFT allocates different budgets across policies, some remain at early stages of fine-tuning, whereas others (e.g., the last policy) progress to later stages.

As shown in Fig. 10, each policy achieves low optimality gaps primarily around the region associated with its corresponding source task set, while its performance generally degrades when moving farther away from that region. This indicates that TSFT encourages different policies to specialize in distinct subregions of the context space rather than forcing a single policy to perform uniformly well across all tasks. More importantly, the specialized regions are complementary across policies, suggesting that the final policy portfolio can provide broader global coverage through task-wise policy selection. This effect is further illustrated in the bottom row of Fig. 10. Although the pretrained and MTL policies achieve reasonably good average performance across the context space, their global coverage remains limited because they lack sufficient specialization.

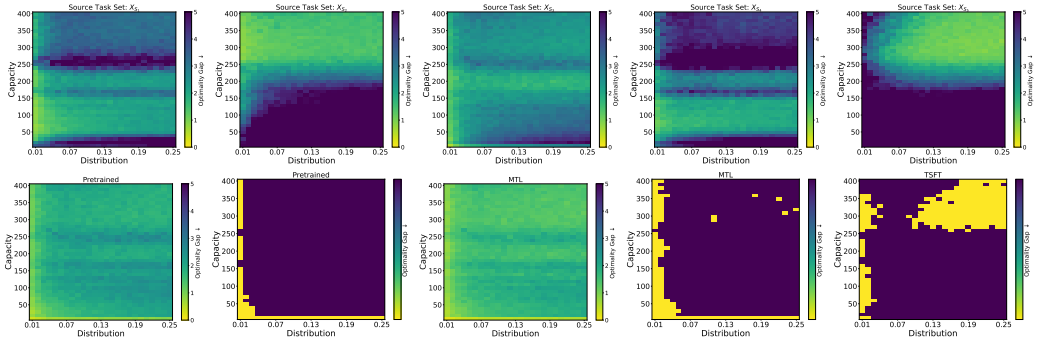


Figure 10: *Top row:* Performance heatmaps of policies selected by TSFT. *Bottom row:* Performance heatmaps and coverage sets (yellow regions) of the pretrained policy and multi-task learning (MTL) policy under the same budget as TSFT, together with the final coverage set achieved by TSFT.

G BROADER IMPACT

This work studies task specialization fine-tuning for contextual reinforcement learning. One positive societal impact is that it contributes directly to *greener* AI practices: by improving sample efficiency, such methods may reduce the computational and energy costs required to adapt RL agents across diverse task conditions. This can make learning-based decision systems more accessible to researchers and practitioners with limited resources. In application domains such as logistics, transportation, robotics, and resource management, improved contextual generalization may also support more robust and efficient automated decision-making in changing environments. However, negative societal impacts may arise if these methods are deployed in high-stakes settings without sufficient validation, since improved sample efficiency alone does not guarantee safety, fairness, or reliability across all contexts. In addition, more efficient adaptation could accelerate the deployment of RL systems in domains where automation may affect labor, privacy, or human oversight. Therefore, practical deployment should be accompanied by careful evaluation, domain-specific safeguards, and appropriate human supervision.

H LICENSE

The licenses and usage of the existing assets are listed in Table 12. Our source code and datasets will be publicly released under the MIT License upon publication.

Table 12: Licenses and usage for existing assets.

Type	Asset	License	Usage
Code	POMO (Kwon et al., 2020)	MIT License	Policy network (CVRP/CVRPTW)
	Stable Baselines3 (Raffin et al., 2021)	MIT License	PPO algorithm (CartPole/Ant)
	MOORE (Hendawy et al., 2024)	MIT License	Meta-World multi-task RL
	VeRL	Apache 2.0	LLM RFT framework
Model	Qwen3-4B-Base (Yang et al., 2025)	Apache 2.0	LLM pretrained policy
Datasets	DAPO-17K (Yu et al., 2025)	Apache 2.0	LLM training / evaluation
	MATH (Hendrycks et al., 2021)	MIT License	LLM training / evaluation (MATH-500)
	GSM8K (Cobbe et al., 2021)	MIT License	LLM training / evaluation
	CodeContests+ (Wang et al., 2025)	Apache 2.0	LLM training / evaluation
Benchmark	CARL (Benjamins et al., 2023)	Apache 2.0	Continuous control CRL benchmark
	Meta-World (Yu et al., 2020)	MIT License	Multi-task robotic manipulation benchmark
	AIME 2024	Public	LLM evaluation
	AIME 2025	Public	LLM evaluation
	Minerva Math (Lewkowycz et al., 2022)	MIT License	LLM evaluation
	MBPP (Austin et al., 2021)	CC-BY-4.0	LLM evaluation
	BigCodeBench (Zhuo et al., 2025)	Apache 2.0	LLM evaluation