AUTOMATING BENCHMARK DESIGN

Anonymous authors
Paper under double-blind review

ABSTRACT

The rapid progress and widespread deployment of LLMs and LLM-powered agents has outpaced our ability to evaluate them. Hand-crafted, static benchmarks are the primary tool for assessing model capabilities, but these quickly become saturated. In contrast, *dynamic benchmarks* evolve alongside the models they evaluate, but are expensive to create and continuously update. To address these challenges, we develop `BeTaL` (Benchmark Tuning with an LLM-in-the-loop), a framework that leverages environment design principles to ***automate the process of dynamic benchmark design***. `BeTaL` works by parametrizing key design choices in base benchmark templates and uses LLMs to reason through the parameter space to obtain target properties (such as difficulty and realism) in a cost-efficient manner. We use our approach to generate a new and challenging spatial reasoning benchmark and to develop new tasks for popular agentic tasks like τ-bench. We carry out extensive experiments on three datasets, at different benchmark target performance (difficulty) levels, and show that `BeTaL` achieves the lowest performance gap, as low as 0.4% and up to 5% in most settings; significantly improving over competing LLM and non-LLM based baselines. These experiments demonstrate that `BeTaL` opens the door to a new paradigm of self-adaptive, continually improving evaluation systems.

1 INTRODUCTION

New developments in LLMs, particularly in powering agents via advanced planning, reasoning, and tool-use capabilities Valmeekam et al. (2023; 2024); Ferrag et al. (2025), have outpaced current methods for evaluation. Static, human-curated benchmarks, such as GPQA Rein et al. (2024) or HLE Phan et al. (2025), remain popular, but are costly to develop and quickly become obsolete as models continue to improve. This is challenging for model developers, as increasingly saturated benchmarks make it impossible to differentiate between the performance of state-of-the-art models.

To address these challenges, researchers have turned to ***dynamic benchmarks*** that can be updated over time. These benchmarks avoid saturation via re-calibration or the introduction of new and harder data; this also limits the risk of *contamination*. For example, LiveBench White et al. (2024) periodically introduces new questions and harder tasks. However, these types of benchmarks still largely rely on *unscalable human authoring and manual updates*. Increasingly popular agentic tasks exacerbate this problem, as simulated environments must be carefully crafted; repeatedly designing and implementing new environments promises to be even more labor-intensive.

How can we build dynamic benchmarks for frontier LLMs without the expense and inefficiency of ongoing manual design and implementation? Unsupervised Environment Design (UED) methods Jiang et al. (2021a) work with environments that are built from abstract task templates with a set of configurable parameters. These parameters can be tuned to produce new and higher utility versions of the benchmark, thus enabling dynamic re-use. In practice, however, we find that the search space over such parameters is intractable for non-trivial environments. Naively sampling random configurations is inefficient, as many will be trivial or unsolvable.

We overcome these obstacles via a new approach, **Be**nchmark **T**uning with **L**LM-in-the-loop (`BeTaL`), that performs dynamic benchmark design. `BeTaL` leverages the capabilities of large reasoning models playing the role of designers. Central to our approach is the use of a powerful ***designer LLM*** tasked with reasoning over the space of possible parameter values, design choices, or tasks. The designer is prompted to consider the various parameters of an under-specified benchmark

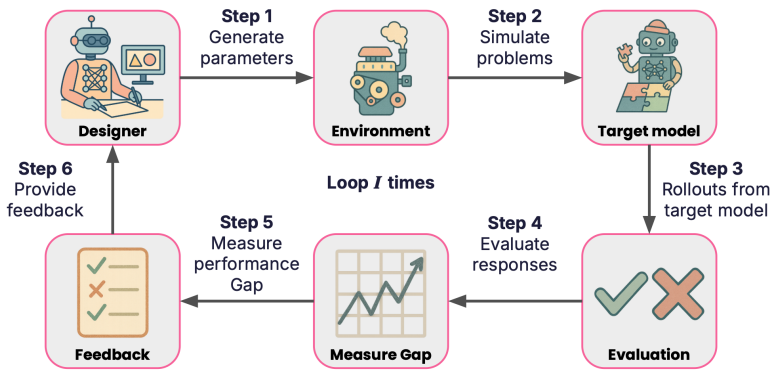

Figure 1: BeTaL automates the process of designing and adjusting ***dynamic benchmarks*** to meet target criteria.

or environment and to propose instances or values that expected to be high utility. This is set up as an interactive and iterative process: after the designer has specified an environment, a simulator creates tasks, and the model or agent being evaluated attempts the tasks, with results provided back to the designer. After each round, the designer must reason over choices and results and make ***changes to increase utility while maintaining the realism and integrity of the basic tasks***. This closed-loop multi-round strategy allows the benchmark to dynamically adjust over time to meet the objectives.

We hypothesize that the strong zero-shot or few-shot reasoning capabilities of frontier models enable the designer to understand the factors that influence usefulness (e.g., task difficulty) and design benchmarks that meet all desirable criteria (e.g., tasks that are just outside of a weaker model's current capabilities). This framework design reduces the burden of designing and continually updating benchmarks to meet the demands of ever improving models. In addition, it permits re-purposing of existing static benchmarks - breathing new life into datasets long considered outdated.

Our contributions are:

- **Dynamic benchmark generation and design**: We formulate the benchmark design process as an optimization problem, where the goal is to maximize utility or usefulness, and we introduce BeTaL as an agentic-driven design process for automatically producing and evolving benchmarks.
- **Efficient task synthesis**: We develop strategies to make BeTaL cost-effective and sample-efficient and compare our approach to baseline methods under a similar computation budget.
- **New benchmarks and empirical validation**: Using BeTaL we modify existing benchmarks to meet new requirements for dataset-level difficulty, and we introduce new benchmarks that focus on mathematical and spatial reasoning. Our empirical results show BeTaL consistently obtains benchmarks with any given target difficulty, achieving a performance gap of as low as 0.4% and up to 5% in several settings, a significant improvement over baselines.

2 Related Work

Automating benchmark design. Recent work automates task generation, verification, and evolution to reduce cost and improve controllability. BENCHMAKER (Yuan et al., 2025) and CHASE (Patel et al., 2025) leverage LLMs for systematic or compositional task building, while graph-based generators validate code tasks via self-consistency (Farchi et al., 2024). Other approaches evolve tasks through perturbation or probing (Wang et al., 2024), employ multi-agent coordination (Butt et al., 2024), or use co-evolutionary loops without seed data (Huang et al., 2025). However, existing methods rarely adapt benchmark difficulty in step with advancing models.

Environment design lineage. Automated benchmark synthesis parallels Unsupervised Environment Design (UED) in RL, where tasks must remain solvable yet challenging. Approaches such as PAIRED (Dennis et al., 2021), PLR (Jiang et al., 2021b), and ACCEL (Parker-Holder et al.,

2023) formalize task selection as an optimization or curation problem. LLM-driven variants like EnvGen (Zala et al., 2024) and LLM-POET (Aki et al., 2024) extend these ideas, emphasizing adaptive curricula that scale with capability.

Scaling environments and datasets. Complementary work builds looks to scale environments in a principled manner, either through synthetic data or manual annotation, to advance agentic intelligence, e.g., AgentScaler (Fang et al., 2025), APIGen (Liu et al., 2024), ToolACE (Liu et al., 2025), and ARE/Gaia2 (Andrews et al., 2025). These emphasize agentic capabilities, whereas our focus is capability-agnostic.

3 METHODOLOGY

We propose BeTaL , a novel framework that uses an LLM-in-the-loop to iteratively design dynamic benchmarks that meet desired objectives. We briefly describe some necessary preliminaries and continue on to explain our method in detail.

3.1 PRELIMINARIES

We describe the key components of our system. We start with an initial template or sketch for an environment. We then obtain a (i) a simulator, (ii) a target model to be evaluated, (iii) a set of target properties, and (iv) a designer model that will perform the design process.

Under-specified environment. The user begins with a high-level description of the task that they wish to realize/instantiate as an evaluation environment or benchmark. Consider the case of a spatial reasoning benchmark, where problems are based on queries about the position of objects on a grid world after applying some operations on the objects. Intuitively, the complexity of problems depends on several factors such as grid size, number of actions, types of operations, etc. While the exact details of the environment remain unspecified, we assume an environment can be characterized by a finite set of parameters $P = \{p_1, p_2, \ldots, p_k\}$, taking values from sets $V_1, V_2, \ldots, V_k$ respectively.

Problem/Task Generator. We also assume access to a simulator that can instantiate the environment for any given parameter configuration from $\mathcal{V} = V_1 \times V_2 \times \ldots \times V_k$. With a given instantiated environment, the simulator is used to generate sets of problems with ground truth. It is expected that the simulated problems adhere to the constraints specified by the parameter values. In this work, we focus on environments with verifiable or procedurally generated solutions, allowing us to assume that the generated ground truth is correct. This forms the dataset that will be used for evaluating models.

Target model. This component covers the model (off-the-shelf or proprietary) or system (e.g., multi-agent system) to be evaluated.

Target performance. Along with the target model, the user also specifies a target performance level $\rho \in \mathbb{R}$ and a distance measure d. The objective is to output a benchmark on which the target model's performance will be close to ρ. The exact definition of ρ is left to the user; for instance ρ could be accuracy, diversity, or an aggregate of multiple measures. In this work, we use target difficulty as our measure of performance of the generated benchmarks, as we seek to overcome the challenge of benchmark saturation.

Designer model. A sufficiently powerful model that can understand the under-specified environment description and the set of free parameters and constraints that influence the environment's complexity. We expect such a model to be able to reason about the design space defined by the high-level environment descriptions, parameters, and their domains and output specific values to the parameters that will result in an environment of given target complexity. We use recent state-of-the-art large reasoning models (LRMs) such as GPT-5, Grok 4, and Claude Opus 4.1.

3.2 BETAL : BENCHMARK TUNING WITH LLM-IN-THE-LOOP

Now that we are equipped with all the ingredients, we will describe BeTaL 's operation. This is shown in Alg. 1, and is explained in detail below.

Algorithm 1 Benchmark Tuning with LLM-in-the-loop (BeTaL)

Input: Under-specified Environment Description, Parameter Set P, Target Performance ρ, Target Model M_t, Designer Model M_d, Number of Iterations I.
Initialize $i^* \leftarrow 0$, $v_{i^*} \leftarrow \emptyset$, minimum gap $\hat{g}_{i^*} \leftarrow \infty$
for $i = 1$ to I **do**
    Prompt $\leftarrow$ Template with Environment description, P, ρ
    if $i > 1$ **then**
        Prompt $\leftarrow$ Prompt + Summary of previous iterations
    end if
    $v_i \leftarrow M_D(\texttt{Prompt})$ ▷ Get parameters from Designer Model
    $v_i \leftarrow \text{ProjectToDomain}(v_i, \mathcal{V})$
    $D_i \leftarrow \text{InstantiateSimulator}(v_i)$ ▷ Generate problems with simulator
    $\hat{\rho}_i \leftarrow \text{EvaluateModel}(M_T, D_i)$ ▷ Evaluate Target Model
    $\hat{g}_i \leftarrow |\hat{\rho}_i - \rho|$
    Update summary of previous iterations with v_i and $\hat{\rho}_i$ ▷ Step 4: Prepare feedback for next iteration
    if $\hat{g}_i < \hat{g}_{i^*}$ **then**
        $i^* \leftarrow i$
        $\hat{g}_{i^*} \leftarrow \hat{g}_i$
    end if
end for
Return: v_{i^*}

Step 1: Parameter Generation (LLM-Guided). In step one of BeTaL , the designer model, an LRM, is prompted to obtain a parameter configuration v_i. Since these values are generated by a language model, it is possible that they may be out of the domain $\mathcal{V}$. Verification is therefore necessary to ascertain that $v_i \in \mathcal{V}$, and, if not, this process is repeated until the generated v_i falls in $\mathcal{V}$. In the end, v_i is projected to $\mathcal{V}$ if it still out-of-domain.

Step 2: Environment Instantiation and Problem/Task Generation. A simulator is instantiated with the parameter configuration obtained in Step 1, which is then used to generate a small set of problems/tasks, with ground truth answers for evaluation, i.e. $D_i = \{(x_j, y_j)\}_{j=1}^{n_s}$.

Step 3: Performance Evaluation. The target model is evaluated on D_i to yield performance $\hat{\rho}_i$.

Step 4: Feedback and Iteration. The iteration details are summarized in natural language to the LRM, including v_i and ρ_i. This summary is included in the prompt as feedback to the LRM to produce the next round of parameters.

Step 5: Termination and Selection. In each iteration, we keep track of the observed performance gap $\hat{g}_i = |\hat{\rho}_i - \rho|$ and keep track of the iteration i^* that results in the smallest gap. After I iterations, the method exits and returns v_{i^*}.

4 Experimental Setup

In this section, we describe our setup for the experiments. First, we give high-level details of the benchmarking tasks, then discuss the baseline methods, our choices of designer and target models, evaluation metrics, and the protocol to run the experiments.

4.1 Benchmarking Tasks

We consider a range of tasks based on arithmetic, spatial reasoning, and airline customer service agents. Each of these settings has rich design space with several free parameters that govern the complexity of the benchmark. This makes them good candidates for evaluating our method. We discuss these tasks briefly here and defer the details to the Appendix.

Arithmetic Sequences. Given an input number x and an output number y, an agent must return the sequence of unary arithmetic operations that, when applied recursively to the intermediate results,

yield y; that is,

$$y = (o_N \circ o_{N-1} \circ \cdots \circ o_1)(x).$$

The benchmark space is constrained to simple operations of add, subtract, multiply, divide, square-root, and power(2). At inference time, the agent is given access to arithmetic operator tools to reason about the problem and come up with a sequence of operators.

Spatial Reasoning. We design a high-level description for a broad category of spatial reasoning tasks. In this setting, there is a 2D square grid (board) with some particles on it; the board and particles can rotate and move around. The questions are about the location and orientations of these objects after certain number of actions. Figure 6 shows an example of a 4x4 grid with two particles on it and the resulting states of the objects after applying some actions. This spatial reasoning environment can be made arbitrarily complex or simple with the choices of several parameters such as board size, types and number of of actions allowed, etc. Here, our goal is to come up with specific values to these parameters such that when we instantiate the environment with those values, it will result in a benchmark with target difficulty level.

τ-bench Airline. It is an interactive evaluation environment for customer service agents in simulated airline scenarios, where the agent must use tools to interact with a database and satisfy user requests (Yao et al., 2024). The reward is determined by checking the final database state against the database state following a series of golden actions. Using its setup, we construct a rule-based scenario generator that randomly samples action chains and corresponding user instructions. We parametrize the scenario generator both by parameters for the tools, such as number of passengers when booking a flight, and by parameters we discover through existing user instructions, such as whether the customer prioritizes the cheapest or the fastest flight.

For further details on these tasks and associated parameters, see Appendix A.1.

4.2 Baselines

We briefly discuss the baselines for evaluation. For details, please see the Appendix.

Random Sampling with Prioritized Parameter Replay (RS + PPR) It is inspired by Prioritized Level Replay (PLR) (Jiang et al., 2021b) in reinforcement learning literature. It works iteratively. In each iteration, it draws a sample $v_i \in \mathcal{V}$ randomly with probability p and with probability $1-p$ it draws a noisy sample from a buffer of "good" parameters it found in previous iterations. Similar to `BeTaL` v_i are evaluated to observe the performance gap $\hat{g}_i$ and if $\hat{g}_i \leq \Delta$, then v_i is added to the buffer. It also keeps track of the best parameters similar to `BeTaL` and returns them in the end.

LLM Prompting Strategies. We explore baselines where LLM can be prompted to obtain parameter values. We use variations of best-of-N (BoN) (**?**Beirami et al., 2025) with our notion of reward and choices of reward "oracles". Here, the reward for any given parameter values is defined as the negative of the observed performance gap. We use two empirical reward oracles, one based on an ML model trained offline with supervised learning and the other one based on simulation and evaluation with the target model as in `BeTaL` . We call the first variation **BoN-ML** and the second one **BoN-TM**. Chain of thought prompting (Wei et al., 2023) is used for both strategies.

4.3 Designer and Target Models

We test the latest reasoning models from three providers: OpenAI (GPT-5), Anthropic (Claude Opus 4.1), and xAI (Grok 4) as designer models. We use o4-mini as the target model in all the settings. We finally evaluate benchmarks developed by each method on three models: o4-mini, Gemini 2.5 Flash, and Claude 3.7 Sonnet. Whenever applicable, we configure the designer model with temperature 0.5 and a high reasoning budget (4096 tokens budget) for exploration, while the rollout and evaluation models use temperature 0.0 with a low reasoning budget (1024 tokens budget) for efficiency.

4.4 Metrics

Each benchmarking task can have its own notion of performance ρ (e.g., accuracy, pass@k, etc.). We assume this measure of performance is inversely proportional to the hardness of the task. We abstract out benchmark-specific measures and define the following metric to measure the effectiveness.

Table 1: Chain-of-thought (CoT) prompting does not consistently yield strong designer-model performance. While Opus-4.1 achieves competitive results on the arithmetic sequence and τ-Bench tasks, state-of-the-art LLMs often struggle to outperform a random sampling baseline. Reported values are $\bar{\hat{g}}(\%)$ with o4-mini as the target model, averaged over three independent runs and presented with 95% confidence intervals (CI).

Method	**Arith. Seq.**	**Spatial Reasoning**	**τ-Bench Airline**
Random Sampling	21.17±51.5	25.4±9.6	37.3±17.2
CoT Prompting (GPT-5)	28.33±25.8	45.3 ±26.3	23.6±16.1
CoT Prompting (Opus-4.1)	11.67±7.2	26.1±17.9	11.9±10.4
CoT Prompting (Grok-4)	20.83±3.6	39.1±25.5	31.9±13.3

Performance Gap. If a method is run with a given target performance level ρ, and say it results in a benchmark on which the target model has performance $\hat{\rho}$, then its performance gap is $\hat{g} = |\hat{\rho} - \rho|$.

4.5 Experiment Protocol

We run 10 iterations for iterative methods and sample 10 times for non-iterative ones. To evaluate a designer's ability to produce benchmarks with varying levels of complexity, we consider four target performance levels: Hard ($\rho^{\text{hard}} = 0.25$), Medium ($\rho^{\text{medium}} = 0.50$), Easy ($\rho^{\text{easy}} = 0.75$), and Trivial ($\rho^{\text{trivial}} = 0.90$). We report the average performance gap $\bar{\hat{g}}$ across these levels as the primary effectiveness metric. All experiments are repeated three times with different random seeds. The sizes of parameter-search rollouts and evaluation datasets are adjusted according to the requirements of each task. See Appendix A for additional details.

5 Results and Discussion

In this section, we present our main results and discussion. First, we study the effectiveness of simple chain-of-thought prompting against random sampling and then provide an in-depth discussion on `BeTaL` 's effectiveness in designing benchmarks for any given target difficulty.

C1: Chain-of-Thought prompting does not make LLMs efficient benchmark designers.

Despite the remarkable reasoning capacity and world knowledge of state-of-the-art LLMs, their ability to systematically design benchmarks remains unreliable. As shown in Table 1, LRMs given high reasoning budgets still exhibit high variance when tasked with producing benchmarks of varied complexity. With o4-mini as the target model, Opus-4.1 exceeds the random baseline only on Arithmetic Sequence and τ-Bench while failing on Spatial Reasoning. GPT-5 and GROK 4 underperform even further. These results demonstrate that chain-of-thought prompting alone does not endow LLMs with robust or generalizable benchmark design capabilities.

C2: `BeTaL` is more effective than the baselines in producing benchmarks with any target performance level.

Our hypothesis here is that while LLMs are highly capable models, a single round of prompting, even with a sufficiently large reasoning budget, may not be as effective as a procedure like `BeTaL` where it is prompted iteratively with feedback on its outputs from the previous rounds.

i) `BeTaL` versus other iterative methods. We compare `BeTaL` with ***RS+PPR*** to understand if LLMs are necessary for benchmark design, or iterative feedback alone can achieve the same performance as `BeTaL` . From our experiments, it is evident that both *Reasoning LLMs* and *iterative feedback* are necessary for an effective designer. Figure 2 shows that `BeTaL` learns to shrink the performance gap more strongly than RS+PPR over 10 iterations, with a wide margin (more than 20%) on both τ-Bench and Spatial Reasoning.

iii) Performance at target performance levels. We observe that `BeTaL` shows robustness, at all desired target levels, by consistently outperforming baselines.

Table 2: BeTaL consistently outperforms the iterative and Best-of-N baselines in both parameter search and evaluation phases across all three tasks and all three designer models. Reported numbers are $\bar{g}(\%)$ with o4-mini as the target model. For parameter search, we run either 10 samples or 10 iterations and report the best result for a fair comparison. All results are averaged over three independent runs and presented with 95% confidence intervals. See experimental details in A

Designer	Method	Arith. Seq.	Spatial Reasoning		τ-Bench Airline	
		Param Search	Param Search	Eval	Param Search	Eval
–	RS+PPR	15.8±2.43	6.6±7.3	8.4 ±6.3	18.3±21	21.3±10.6
GPT-5	BoN-TM	8.3±4.64	28.58±25.66	21.66±41.19	12.5±2.1	20.8±8.0
	BoN-ML	30.0±12.63	21.56±12.25	28.33±19.67	21.4±11.5	16.7±10.4
	BeTaL	**5.8 ±4.77**	**0.4±0.45**	**5.3±7.25**	**5.3 ± 3.2**	**13.2 ± 10.3**
Opus-4.1	BoN-TM	20.0±12.12	26.75±23.03	20.48±19.13	**3.6 ± 3.2**	10.0±12.4
	BoN-ML	31.7±6.20	20.35±12.05	26.93±41.31	11.7±7.5	9.7±7.6
	BeTaL	**12.5 ±4.42**	**3.8±5.3**	**7.3±6.5**	5.0±2.1	**7.7 ± 5.2**
GROK 4	BoN-TM	20.0±11.70	24.29±24.90	21.23±19.43	15.0±11.5	18.5±7.7
	BoN-ML	32.5±15.58	21.17±11.73	25.35±39.60	34.2±14.3	20.2±3.1
	BeTaL	**4.2 ±3.26**	**1.4±3.3**	**4.9±5.8**	**3.9 ± 3.2**	**10.3 ± 12.4**

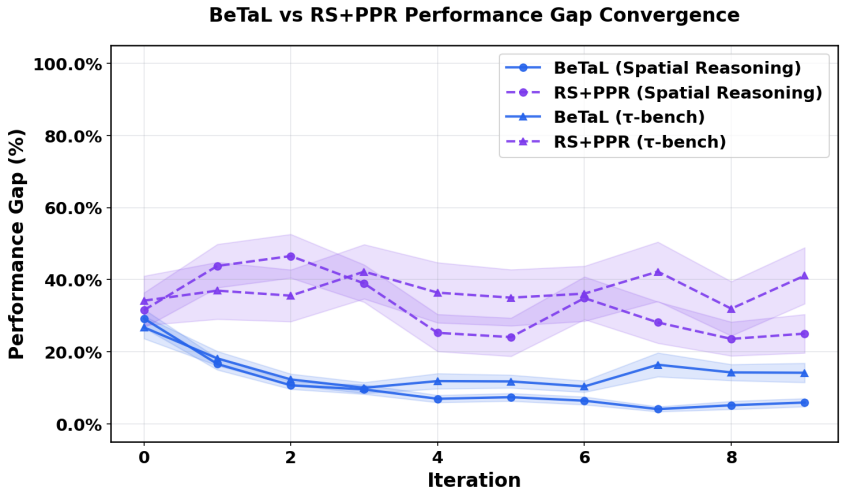

Figure 2: Convergence of iterative methods during parameter selection on Spatial Reasoning and τ-Bench benchmarks: BeTaL vs. RS+PPR. Performance gap of BeTaL shrinks faster compared to RS+PPR, within 10 iterations, indicating LLMs are more efficient than competing iterative methods at finding favorable environment parameters for benchmark creation.

ii) Performance comparison of designer models. While all three reasoning models outperform their respective baselines, we find that the choice of reasoning model may depend on the nature of the benchmark being developed. Comparing between the designers, Grok-4 and GPT-5 do well on the mathematical and logical reasoning domains of Arithmetic Sequences and Spatial Reasoning. On the other hand, Claude-Opus-4.1 excels on the real-world agentic benchmark of τ-Bench Airline.

Next, we study the performance across models, at each target performance level. We observe inherent difficulty levels in the underlying environment domains, that reflect in the designer's performance. For instance, τ-Bench and Spatial Reasoning, being challenging benchmarks, the performance gap is highest on the Trivial and Easy difficulty levels, for all models. Conversely, on the toy Arithmetic Sequences task, the highest gap is observed at hard and medium difficulty levels.

C3: Environment designed by BeTaL for one target model is transferable to other target models.

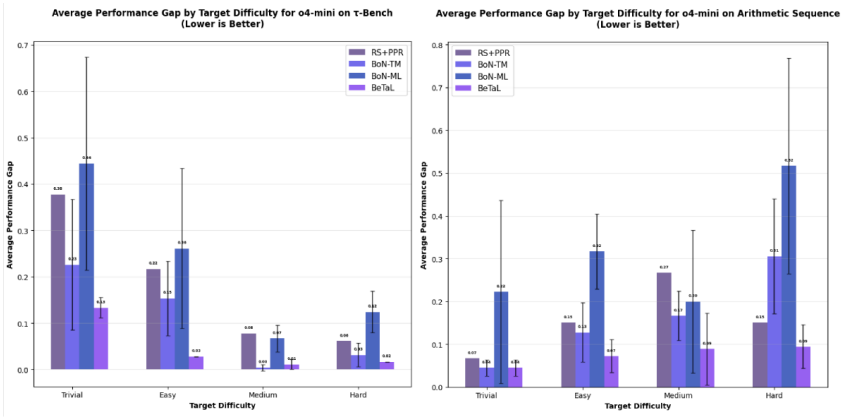

Figure 3: BeTaL performs robustly at different target performance levels, compared to baselines on τ-Bench and Arithmetic Sequences.

Our analysis demonstrates that environments designed by BeTaL exhibit robust transferability across different evaluation models. On τ-Bench, environments designed using o4-mini feedback achieve comparable performance when evaluated on Claude 3.7 Sonnet and Gemini 2.5 Flash (Figure **??**), with BeTaL consistently outperforming baseline approaches across all evaluation models.

We further validate this transferability on Arithmetic Sequences, where BeTaL environments designed with o4-mini feedback were successfully evaluated on both o4-mini and Gemini 2.5 Flash. The environments maintain their fundamental difficulty characteristics across models: At the hard 25% target, o4-mini achieves a performance gap of 9.7 ± 8.7 % while Gemini 2.5 Flash achieves that of 22.8 ± 17.4 %. Although absolute performance differs between models - with o4-mini showing higher accuracy (62.8%) compared to Gemini 2.5 Flash (43.0%) - the relative difficulty calibration transfers consistently across all target regret levels (25%, 50%, 75%, 90%).

This cross-model consistency across different benchmark domains: agentic planning in real-world tasks (τ-Bench) and mathematical reasoning (arithmetic sequences) domains provides strong evidence that BeTaL -designed environments test fundamental cognitive capabilities that generalize across different model architectures and families, rather than exploiting model-specific weaknesses.

C4. Are LLMs also able to generate better parameter space?

Despite LLMs' strong performance of generating arbitrarily complex benchmarks through BeTaL , the parameter spaces for the three tasks we experiment on are still manually designed by human. We further explore the next level of benchmark design autonomy by prompting Opus 4.1 to design the parameter space for τ-bench. We then manually implement feasible parameters into the scenario generator and experiment BeTaL on the AI generated parameter space. According to Figure 5, BeTaL iterated on AI design parameter space performs decently on generating Medium or Hard level benchmarks yet underperforms to human generated parameter space on Easy and Trivial level benchmarks.

6 Conclusion

We introduced **BeTaL** , an LLM-in-the-loop dynamic benchmark design framework. BeTaL is a method for *dynamic benchmark generation and design* that adaptively matches target performance levels, incorporates strategies for *efficient task synthesis* that improve cost-effectiveness compared to baselines, and can be used to create *new benchmarks and empirical validation*. We showed that iterative design with LLMs is consistently more effective than non-iterative or random baselines.

Limitations. BeTaL assumes access to parameterized and verifiable simulators, which may not always exist. Its effectiveness depends on the reasoning strength of the designer model and careful prompt construction. Moreover, our evaluation is limited to a small set of domains, leaving multimodal and more subjective tasks unexplored.

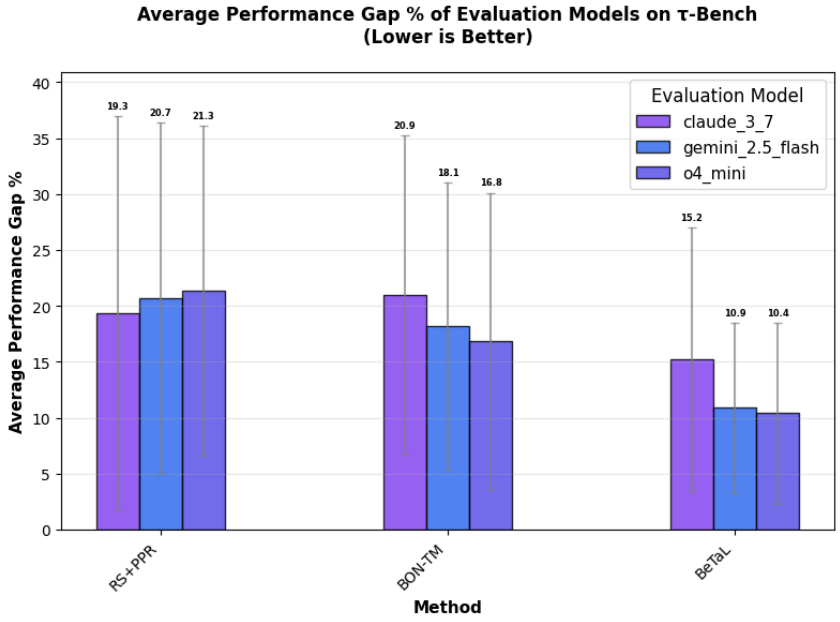

Figure 4: Performance of all evaluation models (o4-mini, Claude Sonnet 3.7, and Gemini-2.5-Flash on τ-Bench. (BeTaL) using feedback from o4-mini sees comparable performances on Claude 3.7 Sonnet and Gemini 2.5 Flash.

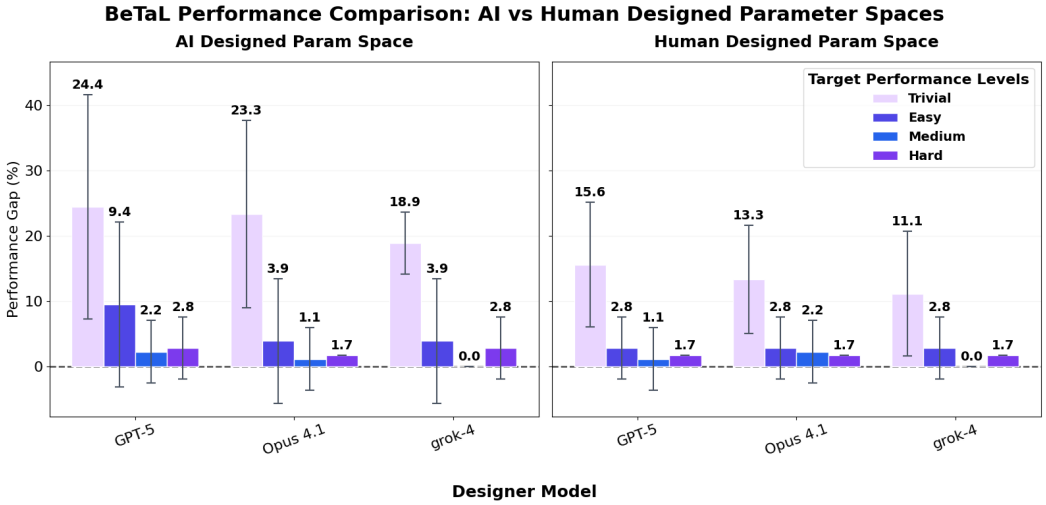

Figure 5: Performance of BeTaL on τ-bench parameter space generated by Opus 4.1 versus by human. BeTaL on AI generated parameter space is acceptably small performance gap for medium and hard benchmarks, yet still generally underperforms to that generated by human.

Future work. A natural direction is to use *environment scaling* more explicitly as a knob, enabling smooth transitions from simple to complex environments as models improve. Extending BeTaL to *automatically propose new parameters*, exploring *multi-agent or co-evolutionary design loops*, and incorporating *human-in-the-loop oversight* could further enhance adaptability and reliability. Ultimately, we envision *self-adaptive benchmarks* that evolve continuously with the systems they evaluate, ensuring robust and meaningful assessment as AI capabilities advance.

REFERENCES

Fuma Aki, Riku Ikeda, Takumi Saito, Ciaran Regan, and Mizuki Oka. Llm-poet: Evolving complex environments using large language models. *arXiv preprint arXiv:2406.04663*, 2024.

Pierre Andrews, Amine Benhalloum, Gerard Moreno-Torres Bertran, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Romain Froger, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Grégoire Mialon, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Thomas Scialom, Vladislav Vorotilov, Mengjue Wang, and Ian Yu. Are: Scaling up agent environments and evaluations. *arXiv preprint arXiv:2509.17158*, 2025.

Ahmad Beirami, Alekh Agarwal, Jonathan Berant, Alexander Nicholas D'Amour, Jacob Eisenstein, Chirag Nagpal, and Ananda Theertha Suresh. Theoretical guarantees on the best-of-n alignment policy. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=u3U8qzFV7w.

Natasha Butt, Varun Chandrasekaran, Neel Joshi, Besmira Nushi, and Vidhisha Balachandran. Benchagents: Automated benchmark creation with agent interaction. *arXiv preprint arXiv:2410.22584*, 2024.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *arXiv preprint arXiv:2012.02096*, 2021.

Runnan Fang, Shihao Cai, Baixuan Li, Jialong Wu, Guangyu Li, Wenbiao Yin, Xinyu Wang, Xiaobin Wang, Liangcai Su, Zhen Zhang, Shibin Wu, Zhengwei Tao, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. Towards general agentic intelligence via environment scaling. *arXiv preprint arXiv:2509.13311*, 2025.

Eitan Farchi, Shmulik Froimovich, Rami Katan, and Orna Raz. Automatic generation of benchmarks and reliable llm judgment for code tasks. *arXiv preprint arXiv:2410.21071*, 2024.

Mohamed Amine Ferrag, Norbert Tihanyi, and Merouane Debbah. From llm reasoning to autonomous ai agents: A comprehensive review. *arXiv preprint arXiv:2504.19678*, 2025.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-zero: Self-evolving reasoning llm from zero data. *arXiv preprint arXiv:2508.05004*, 2025.

Minqi Jiang, Michael Dennis, Jack Parker-Holder, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Replay-guided adversarial environment design. *Advances in Neural Information Processing Systems*, 34:1884–1897, 2021a.

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. *arXiv preprint arXiv:2010.03934*, 2021b.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. Toolace: Winning the points of llm function calling. *arXiv preprint arXiv:2409.00920*, 2025.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh Murthy, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. *arXiv preprint arXiv:2406.18518*, 2024.

Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. *arXiv preprint arXiv:2203.01302*, 2023.

Arkil Patel, Siva Reddy, and Dzmitry Bahdanau. How to get your llm to generate challenging problems for evaluation. *arXiv preprint arXiv:2502.14678*, 2025.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *Advances in Neural Information Processing Systems*, 36:38975–38987, 2023.

Karthik Valmeekam, Kaya Stechly, and Subbarao Kambhampati. Llms still can't plan; can lrms? a preliminary evaluation of openai's o1 on planbench. *arXiv preprint arXiv:2409.13373*, 2024.

Siyuan Wang, Zhuohan Long, Zhihao Fan, Zhongyu Wei, and Xuanjing Huang. Benchmark self-evolving: A multi-agent framework for dynamic llm evaluation. *arXiv preprint arXiv:2402.11443*, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Siddartha Naidu, et al. Livebench: A challenging, contamination-free llm benchmark. *arXiv preprint arXiv:2406.19314*, 4, 2024.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045.

Peiwen Yuan, Shaoxiong Feng, Yiwei Li, Xinglin Wang, Yueqi Zhang, Jiayi Shi, Chuyi Tan, Boyuan Pan, Yao Hu, and Kan Li. Llm-powered benchmark factory: Reliable, generic, and efficient. *arXiv preprint arXiv:2502.01683*, 2025.

Abhay Zala, Jaemin Cho, Han Lin, Jaehong Yoon, and Mohit Bansal. Envgen: Generating and adapting environments via llms for training embodied agents. *arXiv preprint arXiv:2403.12014*, 2024.

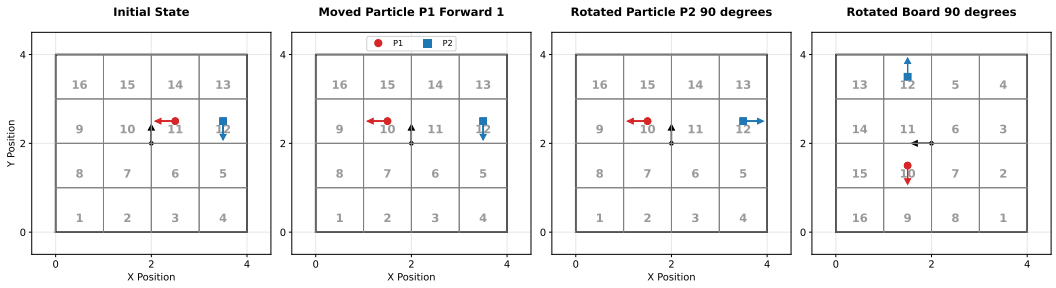

Figure 6: Illustration of objects and actions in spatial reasoning tasks. Here the board is 4x4 and initially oriented towards north (black arrow). There are two particles $P1$ and $P2$ oriented towards west and south respectively. The first action moved the particle forward by one step, second action rotated the particle by 90 degrees and the last action shows rotation of the board by 90 degrees. The board rotations are w.r.t. to its center and when a board rotates or moves the particles on it also rotoate and move along with it.

A Additional Experiments and Details

A.1 Details of benchmarking tasks

Spatial Reasoning. The descriptions of parameters and actions are provided in the prompt (Appendix B) for the designer model. Figure 6 illustrates an example of a sample from the spatial reasoning environment. On such samples, we ask 4 types of queries. i) Absolute location (x,y) co-ordinates of the particle or the board. The board's location is defined as the location of its center. ii) The tile number on which a specific particle is located. iii) The orientation of a given particle. It could be north, east, west, or south. iv) Relative location of a particle or board with respect to another particle or board. When an LLM is prompted with such problems, we instruct it to produce structured outputs along with its reasoning traces. The structured output is verified easily with the groundtruth computed programmatically.

Human Designed Tau Bench Airline. The parameter descriptions and expected behaviors are specified in the designer prompt (Appendix B). Each sample corresponds to an airline itinerary planning scenario parameterized by a small set of discrete controls. The parameter space includes numerical factors such as `num_actions` (1–6), `num_passengers` (1–3), and `num_baggages` (0–3), as well as categorical attributes like `booking_strategy` ("cheapest"/"earliest_arrival"), `is_direct`, `is_round_trip`, `cabin` ("economy"/"business"), and `insurance` ("yes"/"no"). These parameters jointly control itinerary complexity: increasing action count, passengers, or bags expands the combinatorial search space, while enabling multiple strategies, connecting flights, or round-trip requirements adds additional reasoning constraints. When prompted with such parameterized tasks, the LLM designer is instructed to output both a *thought process* describing how the configuration achieves the target failure rate and the final parameter values in structured JSON. This structured output can be programmatically validated against the student model's measured failure rate.

Opus 4.1 Designed Tau Bench Airline. The designer model receives a target failure rate ρ_{fail} and is asked to generate task parameters that achieve $1 - \text{pass@1} \approx \rho_{\text{fail}}$. The parameter space extends beyond structural complexity (e.g., `num_actions` $\in [1, 6]$, `num_passengers` $\in [1, 3]$) to include behavioral and informational dimensions. Categorical controls specify booking preferences (`booking_strategy`: "cheapest"/"earliest_arrival"), routing options (`is_direct`, `is_round_trip`), cabin composition (`cabin_mix`: economy, business, or mixed), and environment conditions such as `information_completeness` (whether all data is provided upfront), `cooperation_level` (helpful/demanding/uncooperative agents), `information_pattern` (upfront, gradual, reactive revelation of details), and `preference_clarity` (explicit vs. implicit preferences). Together, these parameters modulate combinatorial difficulty, reasoning burden, and dialogue complexity, allowing fine-grained control of task hardness to steer the student model's empirical failure rate toward ρ_{fail}. Structured outputs include both the parameter configuration and a *thought process* explaining why it should achieve the desired difficulty level.

A.2 Detailed Baselines

BoN-ML and BoN-TM. These are inspired from the best-of-n alignment technique Beirami et al. (2025). The key idea here is to generate n configurations $v_1, v_2, \ ldots, v_n$ from LLM in parallel and then select the ones that yield the highest "reward". Here, our notion of reward is based on the proximity to the target performance level ρ.

More precisely, a reward oracle in our setting is a function $r : \mathcal{V} \mapsto \mathbb{R}$ that predicts the performance gap for any $v \in \mathcal{V}$. Here, we realize such a reward oracle in two ways. First, based on an offline trained classical ML model, and second, based on online estimation by drawing samples and evaluating them. The steps to estimate performance in the second approach are the same as steps 2 and 3 of the `BeTaL` . The variation of BoN using this oracle is referred to as BoN-TM, and the other one is called BoN-ML. The ML model for BoN-ML is trained in two steps: 1) draw a set of random configurations $\tilde{v}_1, \tilde{v}_2, \ldots \tilde{v}_s$. For each of these configurations, obtain simulated datasets $\tilde{D}_i$ and $\tilde{\rho}_i$ by evaluating the target model M_t on $\tilde{D}_i$. 2) Train multiple classical supervised learning models on $\{\tilde{v}_i, \tilde{\rho}_i\}_{i=1}^{s}$ and pick the best one with cross-validation. In the end, we expect to get a good predictor $\hat{f} : \mathcal{V} \mapsto \mathbb{R}$ that can predict performance for any given $v \in \mathcal{V}$, and define the reward function $r(v) = -|\rho - \hat{f}(v)|$.

BoN-ML Model Training and Selection. As part of the BoN-ML experiments, we trained and compared models to predict regret efficiently. Across all three domains, we explored over 800 different parameter configurations and architectures. Given the relatively small datasets (100 samples per domain, with feature counts ranging from 13 to 74), we applied 5-fold cross-validation to obtain reliable performance estimates.

All features were derived directly from the environment parameters, ensuring the predictors remained lightweight and domain-specific. Models were selected based on the highest cross-validation R^2 score, and the best candidates were saved for deployment. Performance was domain-specific: small neural networks performed best for Arithmetic Sequences, Random Forests excelled in Spatial Reasoning, and gradient boosting worked best for τ-Bench. This process yielded fast, domain-tailored predictors to guide BoN-ML parameter selection effectively.

A.3 Details of LLM Models

LLM Versions GPT-5: undisclosed - the latest GPT-5 version as of Sep 25, 2025 Opus 4.1: claude-opus-4-1-20250805 Grok 4: grok-4-0709 o4-mini: o4-mini-2025-04-16 claude3.7: claude-3-7-sonnet-20250219 gemini-2.5-flash: gemini-2.5-flash

LLM Inference Parameters The default temperature for designer models is 0.5 and for target models is 0.0. However, claude-opus-4-1-20250805 and claude-3-7-sonnet-20250219 are only available with a temperature of 1.

The default reasoning budget for designer models is 4096 tokens and for target models is 1024. However, grok-4-0709 do not support configurable reasoning budget.

A.4 Additional Results

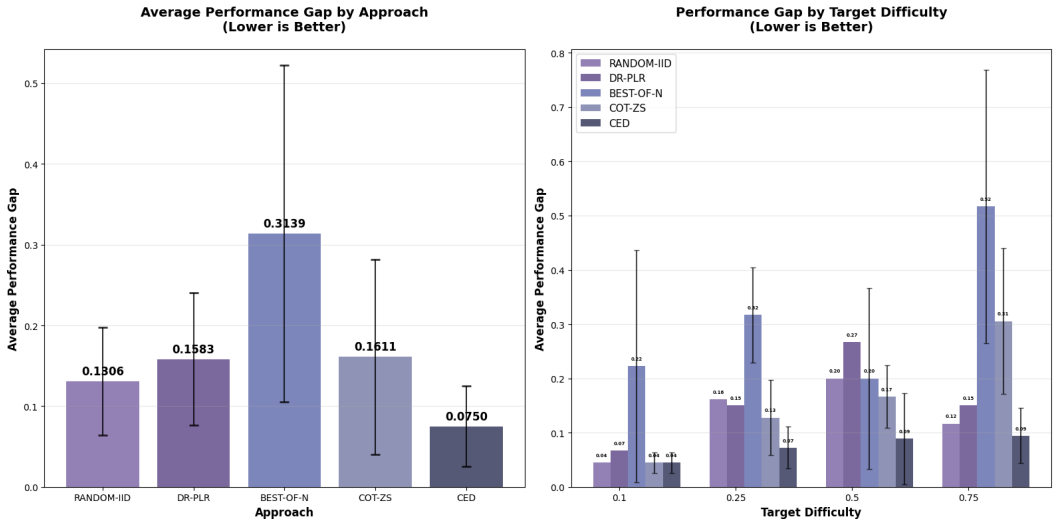

Figure 7: Performance of all methods on Arithmetic Sequences during Parameter Learning Phase, (BeTaL) has the lowest performance gap among competing methods, across target difficulty levels.

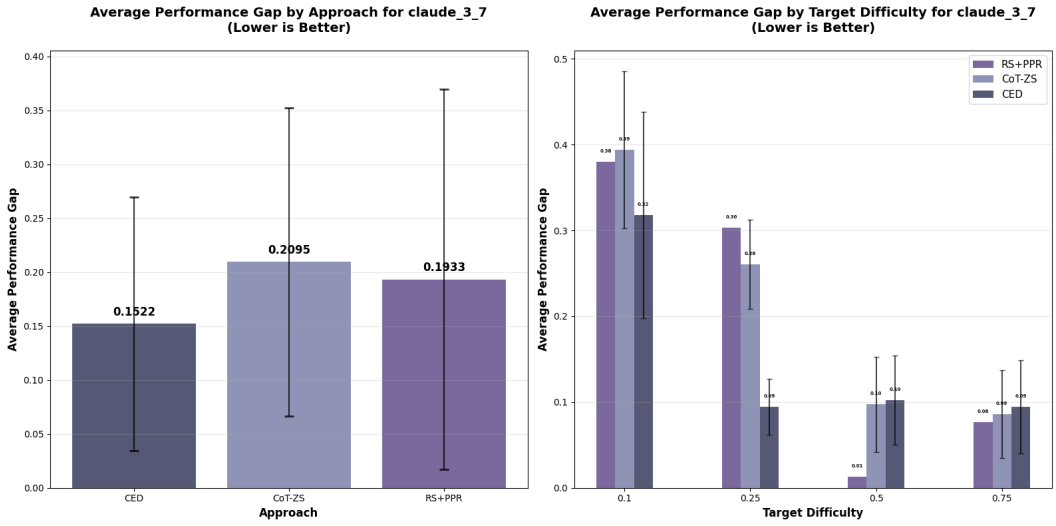

Figure 8: Performance of all methods on τ-bench on Claude 3.7 Sonnet evaluation, (BeTaL) has the lowest performance gap among competing methods, across target difficulty levels.

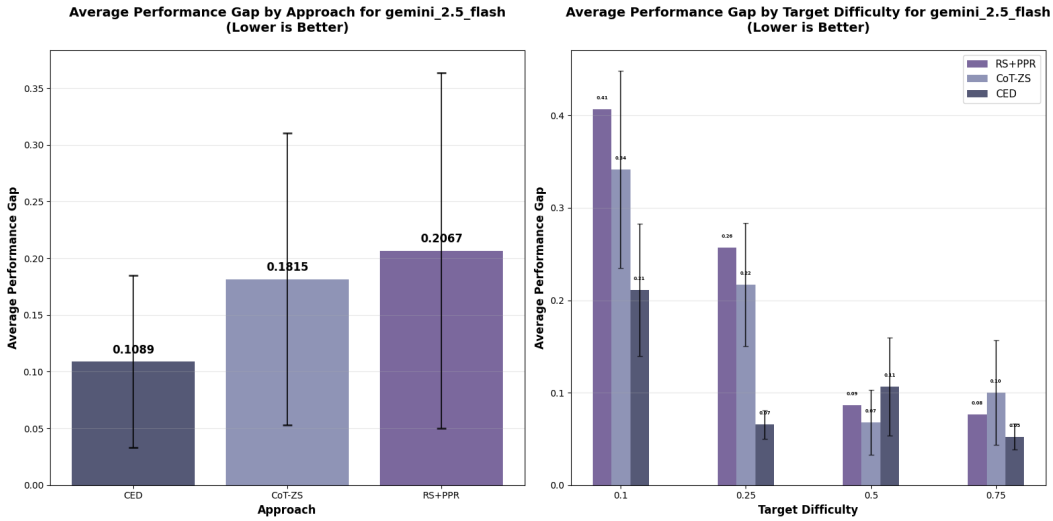

Figure 9: Performance of all methods on τ-bench on Gemini 2.5 Flash evaluation, (BeTaL) has the lowest performance gap among competing methods, across target difficulty levels.

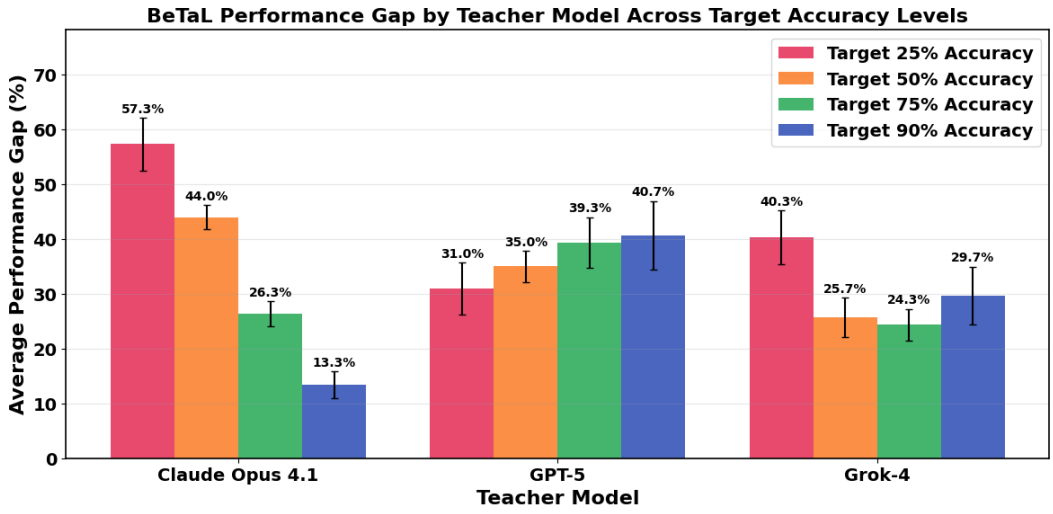

Figure 10: Average performance gap with respect to teacher model for Arithmetic Sequences.

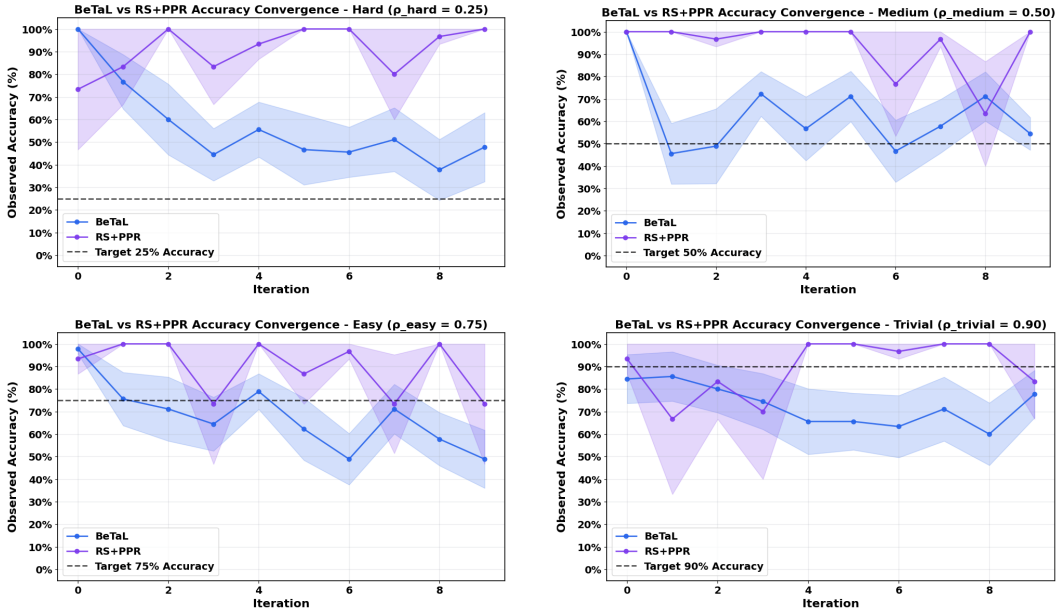

Figure 11: `BeTaL` vs. DR-PLR convergence on Arithmetic Sequences across target accuracies.

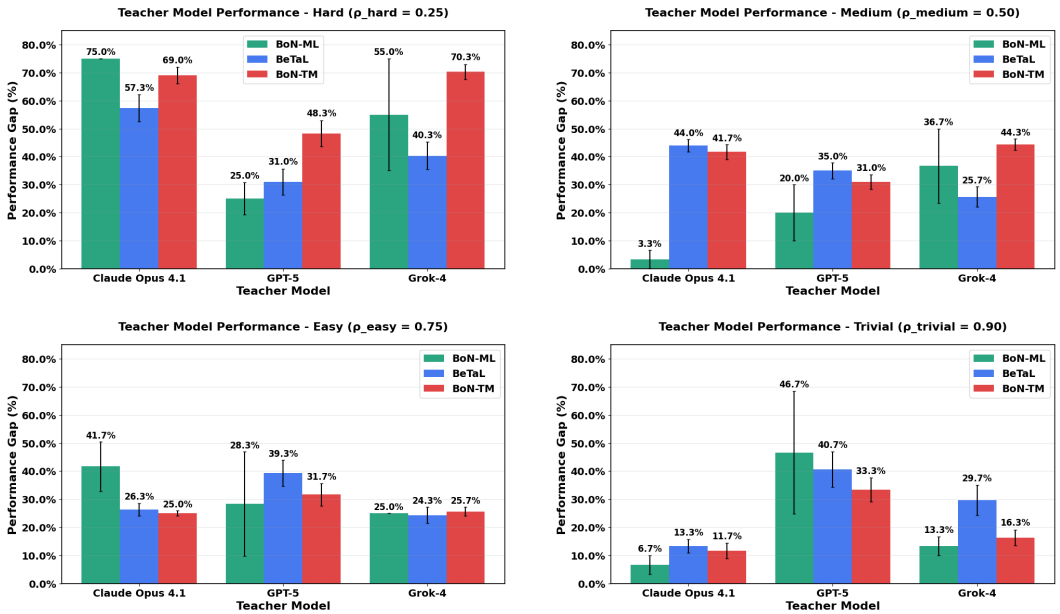

Figure 12: Comprehensive Teacher Performance for Arithmetic Sequences

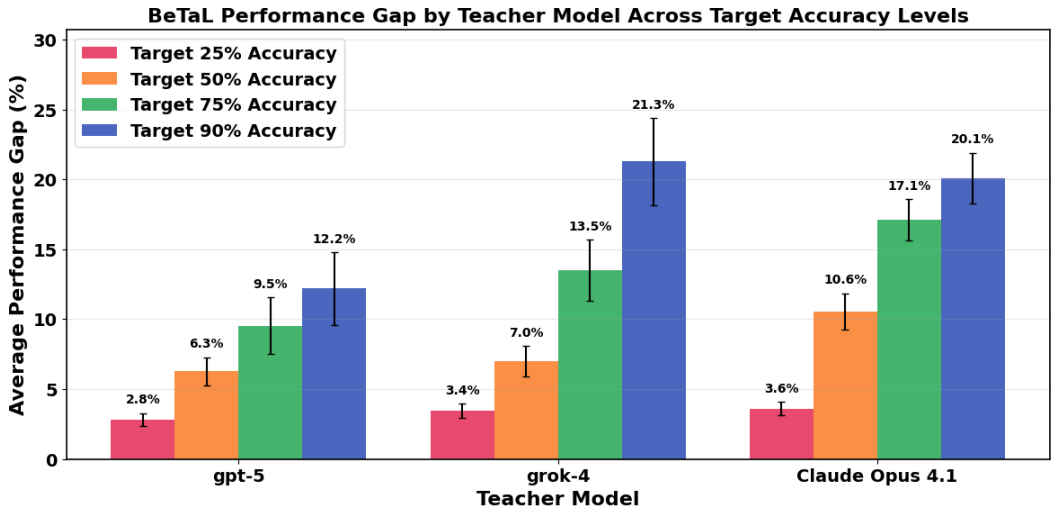

Figure 13: Average performance gap with respect to teacher model for Spatial Reasoning.

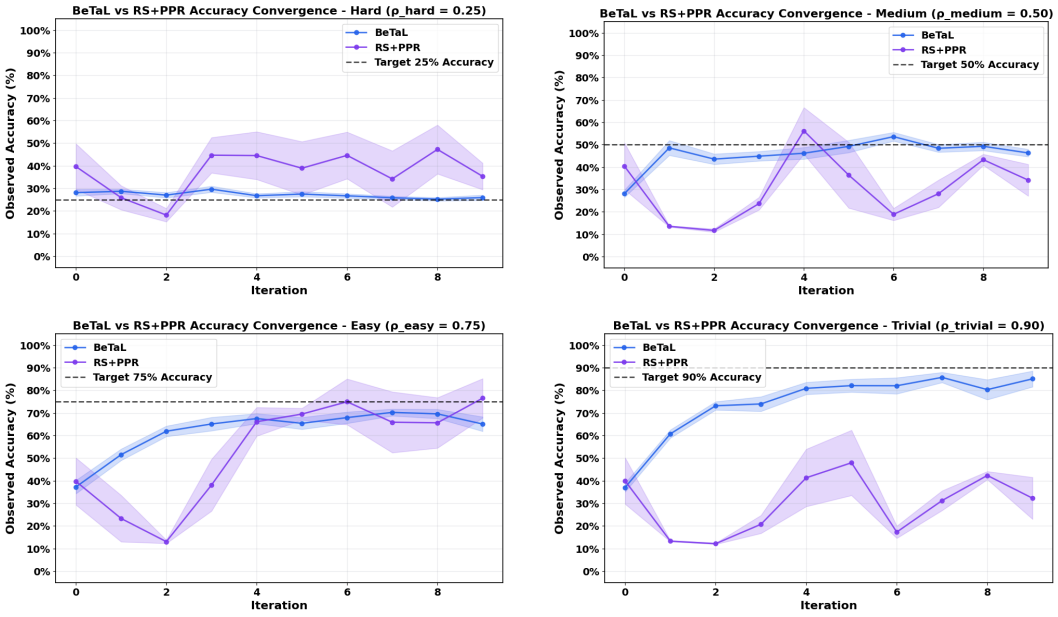

Figure 14: `BeTaL` vs. DR-PLR convergence on Spatial Reasoning across target accuracies.

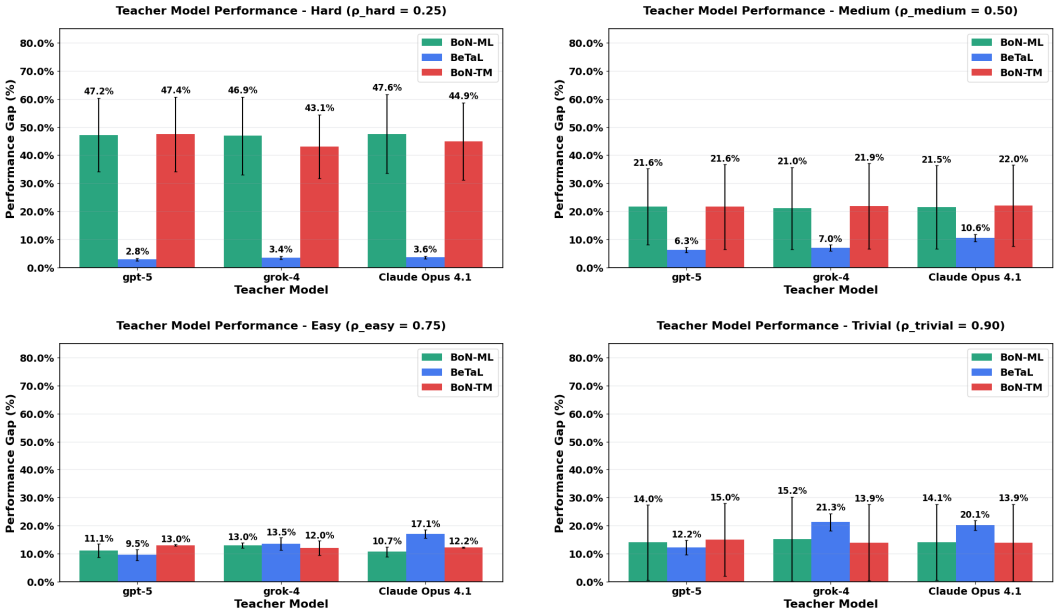

Figure 15: Comprehensive Teacher Performance for Spatial Reasoning

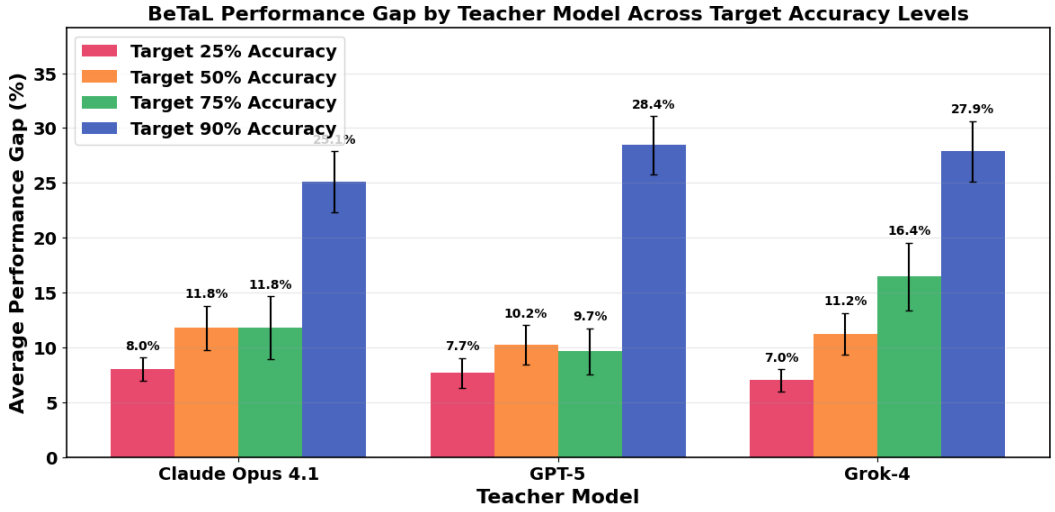

Figure 16: Average performance gap with respect to teacher model for τ-bench.

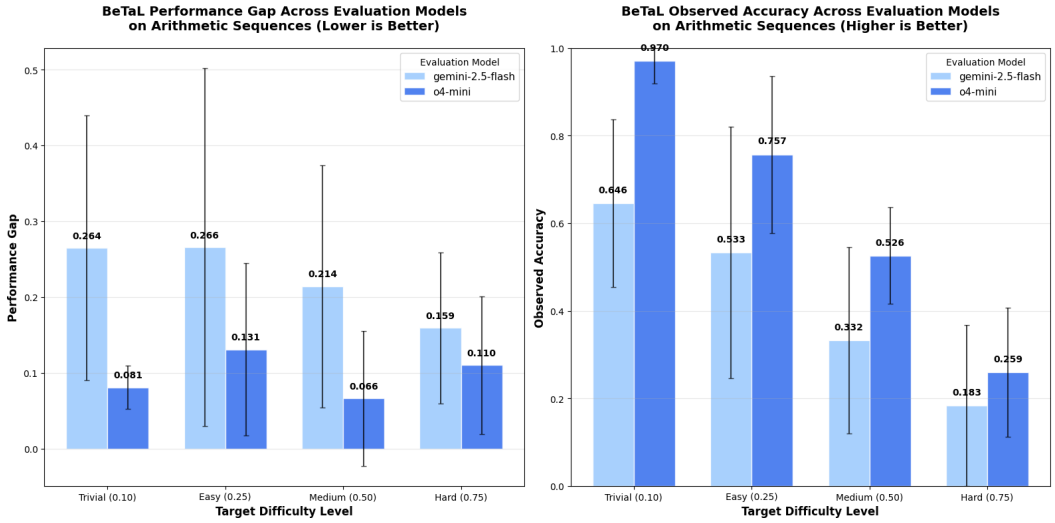

Figure 17: Performance of different models on evaluation datasets on Arithmetic Sequences

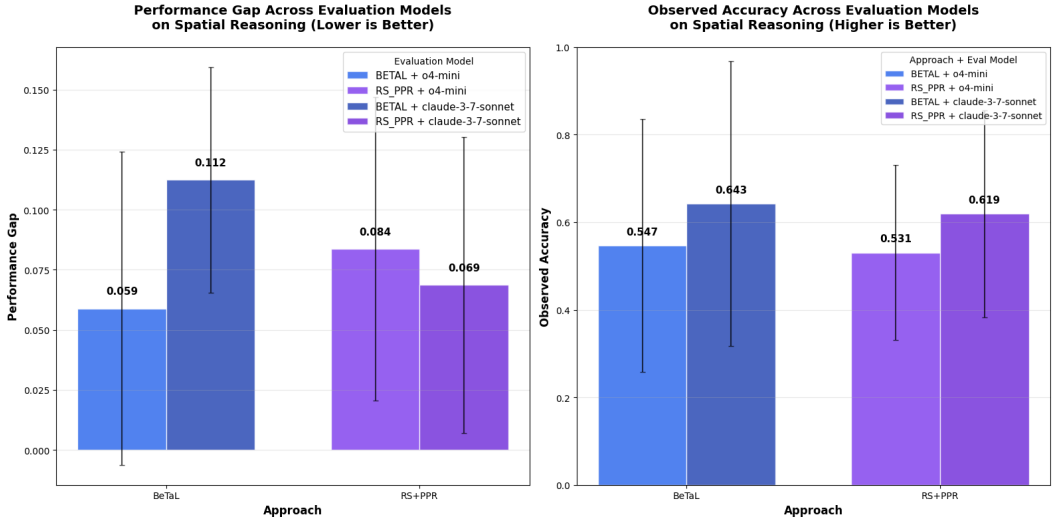

Figure 18: Performance of different models on evaluation datasets on Spatial Reasoning.

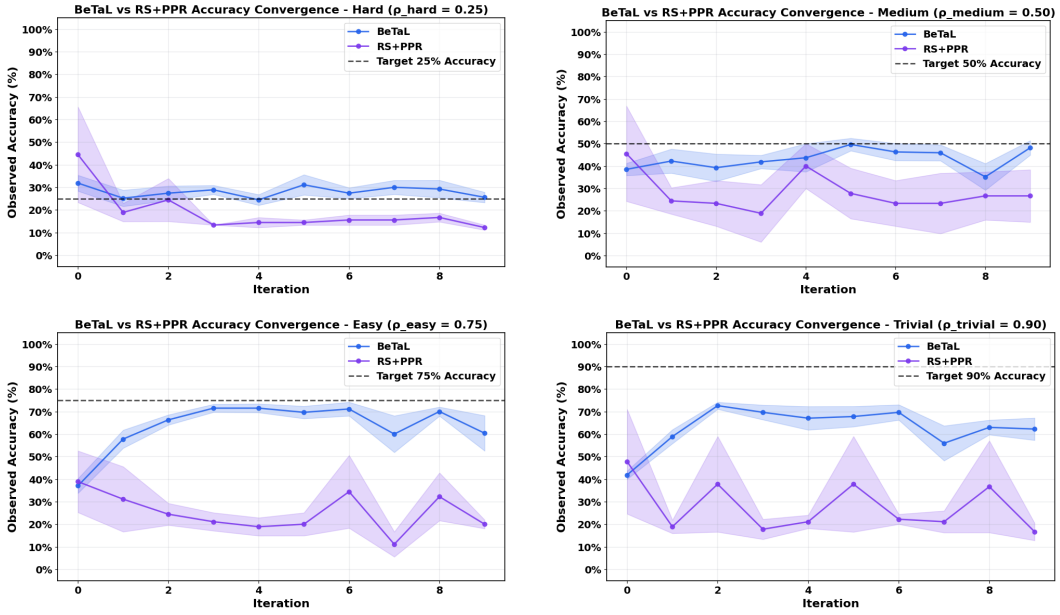

Figure 19: `BeTaL` vs. DR-PLR convergence on τ-bench across target accuracies.

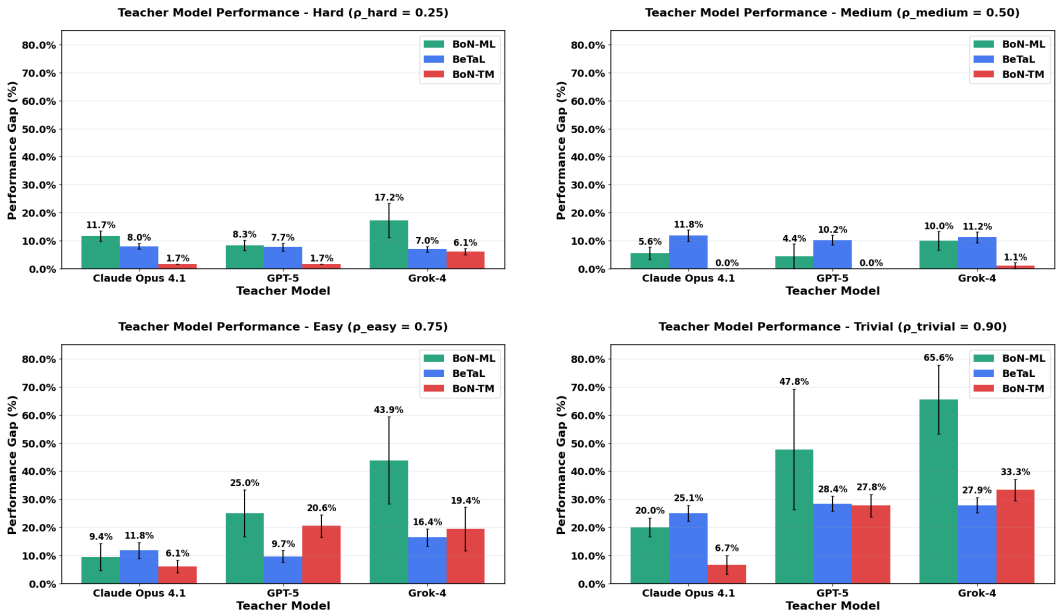

Figure 20: Comprehensive Teacher Performance for τ-bench

B PROMPTS

LLM Designer Prompt for Arithmetic Sequence

The math problem is to apply a sequence of operators on a number to produce a final answer. The sequence of operators are applied recursively on intermediate results, i.e., `num = operator(num)` for each operator in the sequence. The operators only take in one number as input.
You should target the given model regret at {`target_regret`}, so that the parameters can generate a math problem for the model at the desired regret level. A high regret indicates a challenging environment (1 for unsolvable), while a low regret indicates an easy environment (0 for easy).
Here is the feedback from the previous iterations, which you can use to generate new parameters: {`feedback`}
First, reason about the feedback from previous iterations. Specifically note what parameters/aspects made previous environments challenging or trivial.
Then, given a list of common math operators {`operators`}, your task is to generate values for the given parameters:

1. `feedback_summary`: your summary of the feedback from the previous iterations.
2. `thought_process`: your thought process for generating the parameters.
3. `max_range_of_nums`: the upper bound of range the input number can take on, i.e. (1, `max_range_of_nums`). Pick a number between 5 and 50.
4. `N`: the length of the sequence of operators to apply on a number (between 5 and 10)
5. `K`: The maximum number of times an operator can be repeated in the sequence (between 1 and 5)
6. `type_of_nums`: the type of numbers in the input (`int` or `float`)
7. `operator_sequence`: elect 3 operators from the list above, to generate a sequence of operators of length `N` to apply on a number, where each operator can be repeated at most `K` times.

Output format (JSON):

```
{
    "feedback_summary": str,
    "thought_process": str,
    "max_range_of_nums": int,
    "N": int,
    "K": int,
    "type_of_nums": str,
    "operator_sequence": list[str]
}
```

LLM Designer Prompt for the Spatial Reasoning Environment

You are an expert in designing spatial reasoning environments. The environment is a 2D grid world. It consists of a square board and a two particles on the board. The board's dimensions can be from 5 to 100. The board is divided into tiles of size 1x1. The particles are at the center of the tiles.

Each object (board and particles) in the environment has an orientation and a location. The orientation is the direction in which the object is facing, which can be one of the following: NORTH, EAST, SOUTH, WEST. The location of particle is given by the 2D coordinates of

the center of the tile on which the particle is located. The orientation and location of particle are initialized randomly. The location of the board is the 2D coordinates of the center of the board. The orientation of the board is the orientation of its center. It is always initialized to NORTH.

The environments complexity can be controlled by the following parameters:

- The board size determined by the width parameter.
- The board can either allow particles to wrap around the edges or not. It is determined by the wrap_around parameter. If it is true, then the particles can wrap around the edges of the board. If it is false, then the particles cannot wrap around the edges of the board.
- The movements allowed for the objects (board and particles). Each object can have a subset of the following movements: LEFT, RIGHT, FORWARD, BACKWARD.
- The rotations allowed for the objects (board and particles). Each object can have a subset of the following rotations: 0, 90, 180, 270, 360. If the rotation is 0, then the object is not rotated. If the rotation is 90, then the object is rotated 90 degrees counter-clockwise. If the rotation is 180, then the object is rotated 180 degrees counter-clockwise. If the rotation is 270, then the object is rotated 270 degrees counter-clockwise. If the rotation is 360, then the object is rotated 360 degrees counter-clockwise.

You are given a list of parameters for a board and a list of parameters for a particle. You are also given a list of parameters for actions that can be performed on the board and the particle. You need to design a spatial reasoning environment that is sufficiently challenging and an average language model can achieve a target accuracy of <accuracy>.

Response format - JSON schema You must get the final answer and convert it to the following JSON data structure. Follow the schema exactly.

Key: thought_process
Type: String,
Description: Your thought process when designing the environment.

Key: ‘wrap_around‘
Type: Boolean,
Description: Whether the board can wrap around the edges.

Key: ‘width‘
Type: Integer,
Description: The width of the board.

Key: ‘board_moves‘
Type: Boolean,
Description: Whether the board can move.

Key: ‘board_allowed_moves‘
Type: List of Strings,
Description: The movements allowed for the board, must be subset of: LEFT, RIGHT, FORWARD, BACKWARD.

Key: ‘board_rotates‘
Type: Boolean,
Description: Whether the board can rotate.

Key: ‘board_allowed_rotations‘
Type: List of Integers,
Description: The rotations allowed for the board, must be subset of: 0, 90, 180, 270, 360.

Key: ‘particle_moves‘
Type: Boolean,
Description: Whether the particle can move.

Key: ‘particle_allowed_moves‘
Type: List of Strings,

Description: The movements allowed for the particle, must be subset of: LEFT, RIGHT, FORWARD, BACKWARD.

Key: ‘particle_rotates‘
Type: Boolean,
Description: Whether the particle can rotate.

Key: ‘particle_allowed_rotations‘
Type: List of Integers,
Description: The rotations allowed for the particle, must be subset of: 0, 90, 180, 270, 360.

Key: ‘number_of_board_rotation_actions‘
Type: Integer,
Description: The number of times the board can be rotated if board_rotates is true.

Key: ‘number_of_particle_rotation_actions‘
Type: Integer,
Description: The number of times the particles can be rotated if particle_rotates is true.

Key: ‘number_of_board_movement_actions‘
Type: Integer,
Description: The number of times the board can be moved if board_moves is true.

Key: ‘number_of_particle_movement_actions‘
Type: Integer,
Description: The number of times the particles can be moved if particle_moves is true.

LLM Designer Prompt for Tau Bench Airline Environment

You are an expert in designing airline-booking tasks for language-model agents.

Your goal is to propose *task parameters* that drive a student model to a **target failure rate of 0.75**. Here, the failure rate is defined as $1-\text{pass@1}$ for the student model, i.e., the probability that the student fails to solve the task on the first attempt. You are **directly rewarded** for minimizing the absolute distance to the 0.75 failure rate, so choose parameters that make the task challenging enough to approach this target as closely as possible.

The task setting is an airline-shopping environment where an agent must construct an itinerary subject to constraints (e.g., number of actions, bags, cabin class, routing).

Controllable parameters and intended complexity effects:

- `num_actions (1--6)`: Upper bound on primitive planning/interaction steps. *Start simple* with fewer actions; increase to raise difficulty.
- `num_passengers (1--3)`: More passengers typically increases combinatorial constraints and price/timing trade-offs.
- `num_baggages (0--3)`: More bags interact with fare rules and cabin choices; higher values generally increase difficulty.
- `booking_strategy`: Subset of {“cheapest”, “earliest_arrival”}. Multiple strategies introduce objective trade-offs.
- `is_direct`: Boolean. Allowing `false` admits connections and routing search complexity.
- `is_round_trip`: Boolean. Round-trips add coupling between outbound/return constraints.
- `cabin`: Subset of {“economy”, “business”}. More options broaden fare/rule search space.
- `insurance`: One of {“yes”, “no”}. Insurance interacts with cost-focused strategies and can add goal ambiguity.

Tune these parameters to steer the student model's $1 - \text{pass@1}$ toward 0.75.

Response format — JSON schema
You must get the final answer and convert it to the following JSON data structure. Follow the schema *exactly*.

Key: `thought_process`
Type: String
Description: Concise reasoning explaining how the chosen parameters are expected to yield a failure rate near 0.75; reference how each parameter affects difficulty.

Key: `num_actions`
Type: Integer (range: 1–6)
Description: Maximum number of allowed actions/steps.

Key: `num_passengers`
Type: Integer (range: 1–3)
Description: Number of travelers to book.

Key: `num_baggages`
Type: Integer (range: 0–3)
Description: Total checked bags across passengers.

Key: `booking_strategy`
Type: List of Strings (subset of: {"cheapest", "earliest_arrival"})
Description: Allowed objective(s) for the student; may include one or both.

Key: `is_direct`
Type: Boolean
Description: If `true`, only nonstop itineraries are valid; if `false`, connections are allowed.

Key: `is_round_trip`
Type: Boolean
Description: Whether the itinerary must include return travel.

Key: `cabin`
Type: List of Strings (subset of: {"economy", "business"})
Description: Allowed cabin classes.

Key: `insurance`
Type: String (one of: "yes", "no")
Description: Whether trip insurance is part of the task constraints.

Example of a Question in the Spatial Reasoning Setting

Following is the description of the spatial reasoning environment. Go through it carefully and then answer the question in the requested format.

Environment

Setup
All locations are pairs of real numbers (x, y). North corresponds to increasing y, and South corresponds to decreasing y. East corresponds to increasing x, and West corresponds to decreasing x. Orientation is a direction, and can be one of the following: North, East, South, or West. Orientation is also measured in degrees, and can be one of the following: 0, 90, 180, 270. Where 0 means East, 90 means North, 180 means West, and 270 means South.

A board's rotation is defined as the rotation of the board around its center. When a board rotates, the orientation of the board changes, and the tiles and particles on the board also rotate along with it. A particle's rotation changes the orientation of the particle, but does not

change the location of the particle. As a general rule, any entity's rotation can change the orientation of the entity, but does not change the location of the entity.

A board's location is defined as the location of its center. A board's movement changes the location of the board, and the tiles and particles on the board also move along with it. For example, if a board moves forward 1 unit, the center of the board and the tiles and particles on the board all move 1 unit along the orientation of the board. A particle's movement changes the location of the particle For example, if a particle moves forward 1 unit, the location of the particle changes by 1 unit along the orientation of the particle.

If the movement of particles results in the particle moving beyond the boundary of the board, then the particle will either wrap around the boundary of the board or remain at the current tile. It depends on the board's wrap around settings, which are described in the description of the board. As a general rule, any entity's movement can change the location of the entity, but does not change the orientation of the entity. The orientation of an entity can be thought of as the direction in which the entity is facing. This determines the meaning of forward, backward, left, right, etc., for the entity.

Entities

The environment contains the following entities:

Board B1

Setup A board is 12.0 units wide and 12.0 units tall, and contains 2 particle(s). It is centered at (0.0, 0.0). Its orientation is defined as the center's orientation, which is NORTH. Initially, the board is oriented NORTH.
The board has four sides: SIDE-1, SIDE-2, SIDE-3, SIDE-4 The side from the south west corner to south east corner is the bottom side of the board. It is called SIDE-1 The side from the south east corner to north east corner is the right side of the board. It is called SIDE-2 The side from the north east corner to north west corner is the top side of the board. It is called SIDE-3 The side from the north west corner to south west corner is the left side of the board. It is called SIDE-4

Boundaries

In the event the particle move results in the particle moving beyond the boundary of the board, the resulting location is decided as follows:

When a particle is on a tile, it means its location is the tile's centroid. The SIDE-1 of the board can be crossed when approaching from the SIDE-3, and the particle(s) will move to the opposite tile on the SIDE-3. The SIDE-2 of the board can be crossed when approaching from the SIDE-4, and the particle(s) will move to the opposite tile on the SIDE-4. The SIDE-3 of the board can be crossed when approaching from the SIDE-1, and the particle(s) will move to the opposite tile on the SIDE-1. The SIDE-4 of the board can be crossed when approaching from the SIDE-2, and the particle(s) will move to the opposite tile on the SIDE-2.

Tiles on the board

The board is divided into square tiles of size 1 units by 1 units. Tiles are numbered from 1 to (width * height), starting from the bottom left corner in a zigzag pattern. Going from left to right, then right to left, and so on. For example, for a 3x3 board, the tiles are numbered as follows: 9 8 7 6 5 4 1 2 3

Allowed moves

The following moves are allowed for the board: FORWARD - board moves forward 1 unit. BACKWARD - board moves backwards 1 unit. Orientation remains the same. LEFT - board sidesteps 1 unit to the left. Orientation remains the same. RIGHT - board sidesteps 1 unit to the right. Orientation remains the same.

Allowed rotations

The following rotations are allowed for the board: 90 - board rotates 90 degrees. 180 - board rotates 180 degrees. 270 - board rotates 270 degrees.

Particle P1

Initial State

It is located at (3.5, 3.5), and is facing WEST (180 degrees). It is on tile 111. It is on board B1.

Allowed moves

The following moves are allowed for this particle: FORWARD - particle moves forward 1 unit. BACKWARD - particle moves backwards 1 unit. Orientation remains the same. LEFT - particle sidesteps 1 unit to the left. Orientation remains the same. RIGHT - particle sidesteps 1 unit to the right. Orientation remains the same.

Allowed rotations

The following rotations are allowed for this particle: 90 - particle rotates 90 degrees. 180 - particle rotates 180 degrees. 270 - particle rotates 270 degrees.

Particle P2

Initial State

It is located at (-0.5, 5.5), and is facing SOUTH (270 degrees). It is on tile 139. It is on board B1.

Allowed moves

The following moves are allowed for this particle: FORWARD - particle moves forward 1 unit. BACKWARD - particle moves backwards 1 unit. Orientation remains the same. LEFT - particle sidesteps 1 unit to the left. Orientation remains the same. RIGHT - particle sidesteps 1 unit to the right. Orientation remains the same.

Allowed rotations

The following rotations are allowed for this particle: 90 - particle rotates 90 degrees. 180 - particle rotates 180 degrees. 270 - particle rotates 270 degrees.

Actions

The actions are the following: First, board B1 is rotated by 270 degrees. Then, particle P2 is rotated by 270 degrees. Then, particle P1 is rotated by 270 degrees. Then, particle P1 is rotated by 90 degrees. Finally, move particle P2 BACKWARD by 1 units.

Question

What is the location of board B1 after all the actions?

Response format - JSON schema You must get the final answer and convert it to the following JSON data structure. Follow the schema exactly.

Key: ‘board_B1_x‘
Type: Float,
Description: The x-coordinate of board B1 after all the actions.

Key: ‘board_B1_y‘
Type: Float,
Description: The y-coordinate of board B1 after all the actions.

Example of a Task in the Tau-Bench Airline Setting

Following is the description of the airline environment. Go through it carefully and then answer the question in the requested format.

Environment

Setup
The environment simulates a commercial airline booking system. Airports are identified by IATA codes (e.g., SEA, EWR). Dates are formatted `YYYY-MM-DD`. Times are `HH:MM:SS` in local (EST) for scheduling metadata. Cabins include `basic_economy`, `economy`, and `business`. Bookings may be `one_way` or `round_trip`. Payment instruments include `certificate`, `gift_card`, and `credit_card`. Baggage may be free or non-free depending on fare rules (not shown here). Insurance is optional.

Capabilities
Agents may:

- Search flights (nonstop or onestop) between an origin and a destination on a specified date.
- Book reservations with specified flight legs, cabin, passengers, baggages, insurance, and payment methods (in priority order).
- Request issuance of a travel certificate with a specified ID and amount.

Entities

User U1
User identifier: `mohamed_li_7869`.
The user's birthday is present in the profile and should not be requested during the interaction.

Passenger(s)
A single passenger is provided and known to the user:

- `first_name`: Yusuf, `last_name`: Thomas, `dob`: 1966-05-11

Payment Instruments (available to U1)

- `gift_card_3525913`: amount 27
- `gift_card_5876000`: amount 176
- `gift_card_7716568`: amount 237
- `credit_card_1922786`: amount 139

Preferred payment order: **certificate** → **gift card** → **credit card**.

Demands

Demand 1: Flight Search
Search for an **onestop** flight from `LGA` to `DTW` on `2024-05-25`.

Demand 2: Booking
Book a **one-stop**, **one-way** itinerary from `SEA` to `EWR` on `2024-05-30` in **business** cabin for **1 passenger** with **1 total baggage**. Choose the **cheapest** eligible option. Include **insurance**. Use payments in the order: certificate(s) first, then gift card(s), then credit card(s).
Candidate flights presented (for selection during booking):

- Leg 1:
  - `flight_number`: HAT117, `origin`: SEA, `destination`: DFW
  - `scheduled_departure_time_est`: 10:00:00, `scheduled_arrival_time_est`: 14:00:00
  - `status`: available, `date`: 2024-05-30
  - Seats available: basic_economy 5, economy 0, business 1
  - Prices: basic_economy 62, economy 119, business 263
- Leg 2:
  - `flight_number`: HAT063, `origin`: DFW, `destination`: EWR

- scheduled_departure_time_est: 18:00:00, scheduled_arrival_time_est: 21:30:00
- status: available, date: 2024-05-30
- Seats available: basic_economy 11, economy 15, business 9
- Prices: basic_economy 80, economy 137, business 286

Demand 3: Certificate Issuance
Request a certificate with:

- certificate_id: certificate_4314319
- amount: 170

Actions
The intended agent actions, in order, are as follows:

1. **search_onestop_flight** with {origin: PHX, destination: DFW, date: 2024-05-18}.
2. **book_reservation** with the provided passenger, baggage, cabin, flight legs (SEA→DFW, then DFW→EWR on 2024-05-30), one-way, business, cheapest, insurance yes, and payment methods listed above in the stated priority order.
3. **send_certificate** with {certificate_id: certificate_4314319, amount: 170}.

Note. Although Demand 1 specifies LGA→DTW (2024-05-25) search, the sample action shows PHX→DFW (2024-05-18). The agent must honor the stated Demands when resolving inconsistencies (prefer Demands).

Question
Produce the exact JSON payload(s) for the three API calls in the correct order that satisfy all Demands above (use LGA→DTW for the search as specified by Demand 1; for booking, choose the cheapest eligible business one-stop SEA→EWR itinerary from the two legs provided; include insurance; and apply payment instruments in the order certificate → gift card(s) → credit card(s)).

Response format - JSON schema
Return a single JSON object with the following keys:

Key: action_sequence
Type: Array of Objects
Description: The ordered list of actions. Each object must have:

- name (string; one of search_onestop_flight, book_reservation, send_certificate)
- kwargs (object; the exact arguments for the call)

Key: notes
Type: String
Description: Brief justification for flight choice, insurance inclusion, baggage count, and the payment breakdown order.