

---

# Procedural Generation of Algorithm Discovery Tasks in Machine Learning

---

Alexander D. Goldie <sup>\*1</sup> Zilin Wang <sup>†1</sup> Andrian Hayler <sup>†1</sup> Deepak Nathani <sup>†2</sup> Edan Toledo <sup>†3</sup>  
Ken Thampiratwong <sup>†2</sup> Aleksandra Kalisz <sup>†1</sup> Michael Beukman <sup>‡1</sup> Alistair Letcher <sup>‡1</sup> Shashank Reddy <sup>‡1</sup>  
Clarisse Wibault <sup>‡1</sup> Theo Wolf <sup>‡1</sup> Uljad Berdica <sup>‡1</sup> Nicholas Roberts <sup>‡4</sup> Saeed Rahmani <sup>‡5</sup>  
Roberta Raileanu <sup>§3</sup> Shimon Whiteson <sup>§1</sup> Jakob N. Foerster <sup>§1</sup>

## Abstract

Automating the development of machine learning algorithms has the potential to unlock new breakthroughs. However, our ability to *improve* and *evaluate* algorithm discovery systems has thus far been limited by existing task suites. They suffer from many issues, such as: poor evaluation methodologies; data contamination; and containing saturated or very similar problems. Here, we introduce *DiscoGen*, a procedural generator of algorithm discovery tasks for machine learning, such as developing optimisers for reinforcement learning or loss functions for image classification. Motivated by the success of procedural generation in reinforcement learning, DiscoGen spans millions of tasks of varying difficulty and complexity from a range of machine learning fields. These tasks are specified by a small number of configuration parameters and can be used to optimise algorithm discovery agents (ADAs). We present *DiscoBench*, a benchmark consisting of a fixed, small subset of DiscoGen tasks for principled evaluation of ADAs. Finally, we propose a number of ambitious, impactful research directions enabled by DiscoGen, in addition to experiments demonstrating its use for prompt optimisation of an ADA. DiscoGen is released [open-source](#).

## 1. Introduction

Automating the development of machine learning (ML) algorithms with AI offers the potential to unlock new breakthroughs in research. Furthermore, since algorithm discovery agents (ADAs) can alleviate the bottleneck of human

<sup>\*</sup>Lead Author, <sup>†</sup>Core Contributor, <sup>‡</sup>Task Contributor, <sup>§</sup>Equal Supervision <sup>1</sup>University of Oxford <sup>2</sup>University of California, Santa Barbara <sup>3</sup>University College London <sup>4</sup>University of Wisconsin–Madison <sup>5</sup>Delft University of Technology. Correspondence to: Alexander D. Goldie <[goldie@robots.ox.ac.uk](mailto:goldie@robots.ox.ac.uk)>.

Preprint. February 2, 2026.

ideation, implementation and experimentation, their utility scales directly with computational resources.

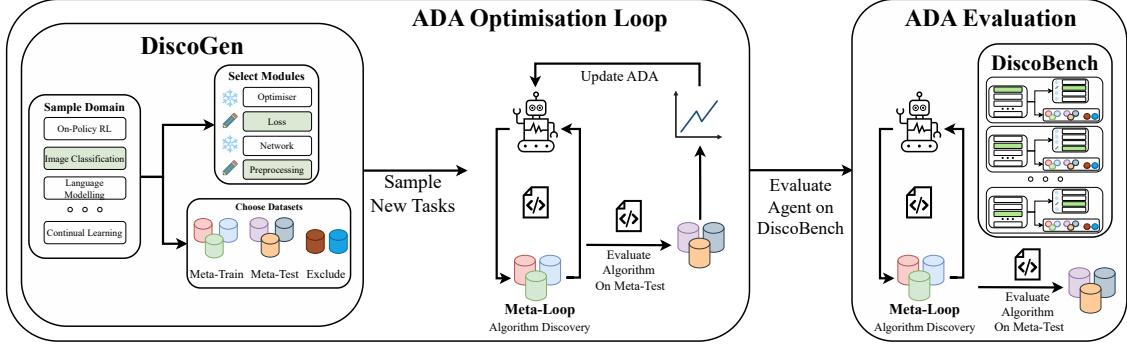
However, existing ADA benchmarks (e.g., MLE-Bench ([Chan et al., 2025](#)) and MLGym ([Nathani et al., 2025](#))) suffer from structural problems that inhibit principled evaluation. They generally fail to separate the *discovery* (meta-train) and *evaluation* (meta-test) of algorithms, meaning ADAs discover algorithms for the same problems they are evaluated on. Additionally, they often require agents to write entire codebases, effectively measuring software engineering rather than research skills, or initialise from full file systems, biasing agents away from discovering novelty ([Nathani et al., 2025](#)). Finally, these benchmarks run the risk of data contamination from pre-training ([Dong et al., 2024](#); [Liang et al., 2025](#)); ADAs may have learned from the fixed task sets during pre-training, and thus change their behaviour or use previously seen problem solutions to compensate for poor research skills ([Liang et al., 2025](#)).

Furthermore, our ability to develop better ADAs for ML remains limited, principally because there are too few different algorithm discovery problems to learn from. These existing suites of tasks for algorithm discovery are constrained due to a reliance on manual creation. Therefore, developing new approaches and architectures for existing suites, or training ADAs on them, risks overfitting.

To address these issues, we introduce DiscoGen. DiscoGen is a procedural generator of algorithm discovery tasks for ML. DiscoGen supports  $> 80$  million different tasks, of varying difficulty, for ADAs. DiscoGen tasks have distinct meta-train/meta-test datasets, where meta-test datasets are hidden from the ADA, ensuring principled evaluation and expanding the generator’s distribution. Furthermore, DiscoGen supports many diverse ML subfields and uses a modular structure that defines which components of an algorithm an ADA discovers, meaning tasks vary over a number of axes.

DiscoGen enables the use of an *ADA optimisation loop*, as in Figure 1. Thus, we establish our terminology as:

- **Inner-loop:** An algorithm optimises a specific model on a single dataset’s train set. When the inner-loop finishes, the model is evaluated on the dataset’s test set. For example,



**Figure 1.** A typical DiscoGen setup. DiscoGen procedurally generates new algorithm discovery tasks. For every generated task, an algorithm discovery agent iteratively develops new algorithms (the meta-loop) for training in the task’s meta-train datasets (the inner loops). The developed algorithm is evaluated on meta-test datasets, with the evaluation score used to optimise the agent (the ADA optimisation-loop). Datasets that are available in a task domain can also be excluded from the task. After each step, DiscoGen can be sampled for a new task. After ADA optimisation has completed, the agent is evaluated on DiscoBench, a set of ADA test tasks.

the inner loop could be training an image classifier on the ImageNet train set (Russakovsky et al., 2015), and evaluating it on the ImageNet test set.

- **Meta-loop:** The ADA iteratively improves the algorithm based on inner-loop feedback from its *meta-train* set, which contains many inner-loop datasets. When the meta-loop finishes, the final algorithm is evaluated on a held-out meta-test set. An example meta-loop could involve an ADA developing image classifier loss functions, based on feedback from ImageNet and CIFAR-10 (Krizhevsky et al., 2009) (meta-train), and evaluating the loss by training an image classifier on CIFAR-100 (meta-test).
- **Task:** A single algorithm discovery problem, defining the ADA’s objective and the meta-train and meta-test datasets.
- **ADA Optimisation loop:** The ADA is optimised for meta-loop performance in DiscoGen tasks in a *meta-meta*-loop (Schmidhuber, 1987). The ADA optimisation loop produces an ADA for evaluation. ADA optimisation could update ADA weights based on meta-test performance.
- **ADA Evaluation:** The ADA is evaluated on a set of tasks that it has not been directly optimised for (in other words, *meta-meta*-test). This could involve developing a language model optimiser, for instance.

Much as procedural environment generation enabled training generalist reinforcement learning agents (Cobbe et al., 2020; Stooke et al., 2021; Bauer et al., 2023; Matthews et al., 2025), DiscoGen enables new research directions in algorithm discovery. Further, ideas like autocurricula (Leibo et al., 2019; Dennis et al., 2021; Parker-Holder et al., 2022a) or recursive self-improvement (Clune, 2019) rely on the ability to consistently sample new, interesting tasks of varying difficulties to prevent overfitting. Given the vast number of possible tasks in DiscoGen, it is a crucial tool for enabling open-ended learning for algorithm discovery (Stanley, 2019; Hughes et al., 2024).

We create DiscoBench, a set of hand-designed tasks from DiscoGen, for ADA evaluation. Similar to Matthews et al. (2025) or Samvelyan et al. (2021), which build benchmarks in procedural environment generators, DiscoBench tasks are in the support of DiscoGen but should *not* be intentionally optimised in ADA optimisation. While DiscoBench, like other benchmarks, is susceptible to data contamination issues, it benefits from the design decisions made in DiscoGen, such as a meta-train/meta-test distinction. Since this paper has not yet been released, our DiscoBench evaluation is not subject to this contamination, and we propose mitigations in Section 8 to overcome this issue in the future.

We provide further research proposals enabled by DiscoGen in Section 6. Finally, as an example of ADA optimisation, we use procedurally generated tasks from DiscoGen for prompt tuning an ADA in Section 7. We explore how ADA evaluation changes when their prompts are optimised over different numbers of *randomly generated* tasks, finding that prompts developed over a wider range perform better.

## 2. Related Work

Due to the depth of prior work in this field, we abridge our related work discussion here and expand it in Appendix B.

### 2.1. Automated Research

Automated machine learning (Hutter et al., 2019, AutoML) focuses on applying machine learning to new problems without expert knowledge. Prior research generally augments machine learning algorithms with methods like hyperparameter tuning (e.g., (Li et al., 2018; Parker-Holder et al., 2021)) or data-cleaning (e.g., (Krishnan et al., 2016)). However, whereas most AutoML research is limited to fitting human-designed solutions to new data, algorithm discovery has the inverse goal: autonomously developing new algorithms.

That said, meta-learning is a subset of AutoML which aims to *learn* algorithms from data (Schmidhuber, 1987; Real et al., 2020; Beck et al., 2023). Often, meta-learned al-

gorithms train a neural network to replace a component of a machine learning algorithm, such as the optimizer (Andrychowicz et al., 2016; Metz et al., 2022b; Goldie et al., 2024) or loss function (Kirsch et al., 2020; Bechtle et al., 2021). Recently, using large language models (LLMs) to propose new algorithms has proven fruitful (Lu et al., 2024a; Romera-Paredes et al., 2024; Novikov et al., 2025). However, optimising and evaluating these systems is difficult due to a lack of diverse, interesting and well-designed tasks. In this paper, we consider how procedural generation can be used to create new algorithm discovery tasks to this end.

As language models have improved (METR, 2025; Chollet et al., 2025), developing more complex research and coding *agents* has emerged as an important pursuit. Agents augment language models with the ability to take actions, use tools and run code (Wang et al., 2024a; Schick et al., 2023; Yang et al., 2024). AI research agents (Toledo et al., 2025) use these tools to automate parts of the research process in a ReAct loop (Yao et al., 2023), where they receive feedback from tools while developing solutions. Rather than focusing on designing new agents, we build a framework for sampling millions of tasks to aid their development.

Agents can be applied throughout the research pipeline. Examples include automating ideation (Si et al., 2024), implementing research ideas with a human-in-the-loop (Gottweis et al., 2025; Weston & Foerster, 2025), judging research papers (Si et al., 2024; Thakkar et al., 2025), or automating the entire research workflow (Lu et al., 2024b; Yamada et al., 2025; Intology, 2025; Si et al., 2026). Specifically in algorithm discovery, considerations include how agents should search over new algorithms (Jiang et al., 2025; Toledo et al., 2025) or when to run experiments (Yu et al., 2025; Nathani et al., 2025). In this work, we provide a rigorous and scalable generator of tasks for optimising and evaluating ADAs.

## 2.2. Optimising Agents

Pretraining language models on more data leads to improved performance (Kaplan et al., 2020), and large procedurally generated environments have led to generalist reinforcement learning (RL) policies (Section 2.3). Motivated by these findings, we consider how ADAs can be optimised using procedurally generated algorithm discovery tasks.

Optimising agents specifically for mathematics (e.g., (Lewkowycz et al., 2022; Trinh et al., 2024; Hubert et al., 2025)) or coding (e.g., (Li et al., 2022; Rozière et al., 2024)) has led to significant gains. Underpinning these advances are large, diverse and verifiable problem sets to research for or train models on (Shao et al., 2024; Wen et al., 2025). For example, there are many suites of mathematical problems (e.g., (Cobbe et al., 2021a; Hendrycks et al., 2021b)) or open-source code repositories (Jimenez et al., 2024; Chen et al., 2021). However, developing similarly large sets of

algorithm discovery tasks has proven difficult, as manual curation requires expert knowledge and, often, adaptation to integrate different data. Additionally, developing superhuman algorithms requires measuring ‘*how good*’ algorithms are, rather than correctness. Here, we mitigate these limitations of manual task creation using procedural generation.

## 2.3. Procedural Generation

Procedural content generation (PCG) involves creating levels or environments algorithmically, according to rules, rather than manually (Togelius et al., 2013). To do so, PCG environments are defined as Contextual Markov Decision Processes (Hallak et al., 2015, Contextual MDPs) or Underspecified Partially Observable MDPs (Dennis et al., 2021) which define levels by a small number of configuration variables. In deep RL, PCG has proven effective for training agents to generalise over a smooth *distribution* of levels, rather than solving specific levels only. Such approaches have been applied to environments of ranging complexity, from gridworlds (Chevalier-Boisvert et al., 2023) to physics engines (Matthews et al., 2025) or 3-dimensional worlds (Stooke et al., 2021). Procedural generation enables new research directions, such as autocurricula (e.g., (Jiang et al., 2021b; Dennis et al., 2021)) or large scale meta-learning (e.g., (Bauer et al., 2023; Nikulin et al., 2024)). We explore how to apply similar principles to algorithm discovery.

## 3. The Problems With Algorithm Discovery Task Suites

ADA improvement is bottlenecked by current algorithm discovery task suites. They are heavily limited in scale due to a reliance on manual task creation. Benchmarks like MLGym-Bench (Nathani et al., 2025, 13 tasks), MLAGentBench (Huang et al., 2024, 13 tasks), MLEBench (Chan et al., 2025, 75 tasks) and REBench (Wijk et al., 2025, 7 tasks) all assess ADA performance, but none provide enough tasks to optimise ADAs over a diverse range, nor separate ADA evaluation from optimisation. Ideally, as with procedurally generated RL environments, we want to optimise ADAs over a *distribution* of tasks to robustly develop algorithm discovery capabilities. Furthermore, these benchmarks suffer from flaws that limit their ADA evaluation.

### 3.1. Issues With Existing Task Designs

Beyond limited scope, we believe task design in these benchmarks is insufficient. Here, we discuss a number of their structural flaws, which DiscoGen rectifies (Section 4.3).

**Poor Evaluation** Proper evaluation in machine learning relies on a distinct train/test split (Goodfellow et al., 2016) to avoid overestimating performance. Algorithm discovery is no different; despite not fitting a *model* to test data in the *inner-loop*, hill-climbing algorithms on meta-train datasets is susceptible to the same flawed evaluation as humans using validation signals to design methods (Langford, 2005;

Whiteson et al., 2011). However, existing benchmarks miss the proper train-test boundary. Rather than measuring algorithm transfer from *meta-train* to unseen *meta-test* datasets, they evaluate the performance of inner-loop trained models on the test set of the (known) meta-train datasets (e.g., (Nathani et al., 2025; Chan et al., 2025)). In effect, testing an algorithm on the dataset it was developed on, rather than how well it generalises. Whilst this can be valid evaluation in certain settings, it is not generally the objective in algorithm discovery (Goldie et al., 2025). We demonstrate the importance of meta-test evaluation in Section 5.2, where algorithms often perform worse in meta-test over meta-train.

**Limited Diversity** Due to their limited scale, benchmarks are often restricted to similar types of problem, such as small Kaggle-style challenges (Chan et al., 2025) or quick-to-run problems (Nathani et al., 2025). As such, rather than understanding the general performance of ADAs, they measure their ability in specific *types* of problem only.

**Limiting Initialisation** While in-context examples help elicit reasoning in LLMs (Wei et al., 2023), they can limit output diversity (Turpin et al., 2023). Since many benchmarks initialise tasks from full implementations, this can limit the creativity of agents as in Nathani et al. (2025), where agents devolve to hyperparameter tuning. Despite being a shortcoming of ADAs, we believe rectifying it at the *task* level is important for analysis. Furthermore, this hampers our ability to understand ADA capabilities; when starting from a full implementation, an ADA could submit a working solution without making its own edits.

**Slow Manual Expansion** Adding *every* new task is manual, meaning scaling their quantity is inherently limited by the number of human-hours available.

**Data Contamination** Data contamination, where evaluation data leaks into training, is an issue in LLM benchmarks due to large-scale pretraining (Dong et al., 2024). It can be especially problematic in ‘challenge-based’ benchmarks, like MLE-Bench. Since agents can often use the internet, or may have pre-trained on challenges, ADAs can reproduce public solutions to boost their score (Hamin & Edelman, 2025). Such logic can be extended to other contamination types, like agents seeing open-sourced dataset labels. While limiting internet access is a mitigation, it is advantageous to design benchmarks that are as robust as possible instead.

**Floor and Ceiling Effects** Many suites include *saturated* or *solved* problems (e.g., (Huang et al., 2024)) with hard-to-change difficulties, limiting their signal for optimisation.

## 4. DiscoGen

DiscoGen generates **tasks**; modular algorithm discovery problems consisting of an objective and meta-train and meta-test datasets. Each task is defined by five components.

**1. Task Domain** The task domain establishes which area of machine learning a task pertains to (e.g., *On-Policy RL* or *Image Classification*). It defines the initial codebase, agent objective, and which datasets and modules are available.

**2. Editable Modules** For each task domain, we identify important building blocks, or *modules*, of an algorithm that can be set as *editable* or *fixed*. Whereas *fixed* modules use standard implementations (e.g., a conventional loss function or optimiser), *editable* modules specify the interface for an agent’s code only (i.e., inputs, outputs and generic function names). This reduces the bias of agents, while ensuring their implementation fits the rest of the codebase. There can be many modules per domain, combinatorially expanding the number of possible tasks that can be generated.

**3. Meta-Training Datasets** Meta-training datasets are the problems which an agent can run experiments on during algorithm discovery. They are known to the agent.

**4. Meta-Test Datasets** Meta-test datasets are used to test the agent’s discovered algorithm *after* the meta-loop has completed. They test how well algorithms transfer to held-out problems, and are not known to the agent.

**5. (Optional) Backend** For some task domains, we implement additional *backends* that provide new tasks without warranting their own domain. For example, in On-Policy RL, a user can specify whether to use a feed-forward or recurrent policy, providing a new search space for algorithms.

```

1 task_domain = "OnPolicyRL"
2 meta_train = ["Breakout", "Freeway"]
3 meta_test = ["Asterix", "SpaceInvaders"]
4 backend = "recurrent"
5 change_optim = False
6 change_loss = True
7 change_networks = True
8 change_train = False

```

*Example 1.* An example task configuration. A task is defined by its task domain, meta-train/meta-test datasets, backend and modules.

We use Example 1 to demonstrate the task interface. For this on-policy RL task, the ADA will work in JAX (Bradbury et al., 2018). The majority of the codebase is *fixed*: domain-specific code like wrapper functions and environment creation, as well as the optimiser (which uses Adam (Kingma & Ba, 2017)) and training loop modules. While an agent *can* interact with these files, any changes will be overwritten prior to meta-testing to prevent evaluation hacking.

The agent must work with two *editable* files: the loss, which starts as an empty function mapping inputs like recently collected data batches to a scalar loss; and the network architecture, which maps environment observations to an RL policy, value function and recurrent state. The ADA can make arbitrary code edits, such as writing additional functions, importing missing packages, or filling the templates.

The agent can also *run* inner-loop training and evaluation for both meta-train environments. To ensure security during agent execution, the ADA operates in a containerised environment. If one editable module is *only* called by another, the agent can also edit its interface, expanding the search-space of algorithms in DiscoGen tasks even further.

#### 4.1. Procedural Task Generation

DiscoGen is a procedural generator of ML tasks. As a procedural generator, it is designed for *improving* ADAs, manually or automatically in an ADA optimisation loop (Figure 1). A task is specified by a small configuration file, like Example 1, which takes the same role as the parameters in other PCG environments (Cobbe et al., 2020; Stooke et al., 2021). This can be randomly generated, user-specified, or sampled using autocurricula. The technical details of sampling tasks in DiscoGen are described in Appendix D.1.

DiscoGen creates tasks in a two-stage process to reduce the risk of meta-test leakage. The meta-train portion of the task is generated first; only *after* the meta-loop is complete does DiscoGen create the meta-test codebase. As such, the agent is never given any details of the meta-test datasets.

PCG involves creating levels, or tasks, programmatically. DiscoGen is no different; the number of tasks for a domain is combinatorial with respect to how many modules and datasets it supports. Specifically, for a task domain with  $m$  modules,  $d$  datasets, and  $b$  backends,

$$N_{\text{tasks}} = b \cdot (2^m - 1) \cdot (3^d - 2^{(d+1)} + 1). \quad (1)$$

Derived in Appendix D.2, this assumes at least one editable module and assigns each dataset to either meta-train, meta-test or unused (with at least one meta-train and meta-test).

#### 4.2. Available Task Domains

Table 1 provides a snapshot of the domains in DiscoGen, which represent diverse fields ranging from simple applied problems to complex foundational ones, spanning supervised to reinforcement learning. We expect this collection to grow with open-source community contributions. We describe each domain, its modules and datasets in Appendix A. Creating new domains is simple, needing only mild adaptation of existing codebases. Due to imbalanced task counts, we recommend uniformly sampling domains to reduce bias.

DiscoGen exhibits useful diversity across these millions of tasks; in Section 7, we show that ADA optimisation improves as more DiscoGen tasks are experienced. The domains supported in DiscoGen span a range of machine learning fields, incorporate datasets of varying complexity and difficulty, and are built upon codebases of differing scales. Most task domains include unique modules, and when there is overlap, implementations are domain-specific.

<sup>1</sup>Model unlearning involves finetuning a pretrained model. For  $n$  models,  $N_{\text{tasks}} = b \cdot (2^m - 1) \cdot ((2n + 1)^d - 2(n + 1)^d + 1)$

Table 1. Overview of domains and their number of supported tasks.

| Task Domain                     | $m$ | $d$ | $b$ | $N_{\text{tasks}}$ |
|---------------------------------|-----|-----|-----|--------------------|
| Bayesian Optimisation           | 6   | 11  | 1   | 10,902,276         |
| Brain Speech Detection          | 3   | 7   | 1   | 13,524             |
| Computer Vision Classification  | 4   | 9   | 1   | 279,900            |
| Continual Learning              | 5   | 3   | 3   | 1116               |
| Greenhouse Gas Prediction       | 2   | 4   | 1   | 150                |
| Language Modelling              | 3   | 4   | 1   | 350                |
| Model Unlearning <sup>1</sup>   | 1   | 3   | 1   | 14,196             |
| Off-Policy RL                   | 6   | 4   | 1   | 3,150              |
| On-Policy RL                    | 4   | 13  | 3   | 71,007,300         |
| Unsupervised Environment Design | 4   | 4   | 1   | 750                |
| Total                           |     |     |     | 82,222,712         |

In Appendix E, we explore how ADA performance changes in on-policy RL for all 15 possible module combinations. Our analysis reveals that increasing the number of editable modules lowers the ADA’s success rate, but the increased flexibility raises its achievable performance ceiling.

We further validate this diversity through *rank correlation analysis* over the performance of different ADAs in DiscoBench (Section 5.2), a subset of DiscoGen tasks, in Figures 2 & 3 (Appendix G). Hierarchical clustering of the correlation matrix reveals distinct patterns; while correlation is often high between similar modules in different domains or different modules in the same domain, there are also anti-correlations where strong performance in one task implies poor performance in another, including within the same domain. The Fisher-Z transformed mean Spearman correlation is  $\sim 0.4$ ; high enough to indicate non-random signal, while sufficiently small to show low redundancy in DiscoGen. Notably, we also find that clustering patterns are *distinct* between meta-train and meta-test, demonstrating how the *same algorithm* can rank differently across datasets.

Per-dataset analysis in each task reinforces the meaningfulness of DiscoGen’s large task space. Considering Appendix K, the ranking of the discovered algorithms changes across datasets within the *same task*. This is intuitive, given prior literature suggests optimal algorithms differ between datasets or RL environments (e.g., in reinforcement learning (Goldie et al., 2024; Jackson et al., 2025), computer vision (Rodrigo et al., 2024; Takahashi et al., 2024) or language modelling (Dao & Gu, 2024; Jelassi et al., 2024)).

#### 4.3. Advantages of DiscoGen

Individual task design in DiscoGen also overcomes the many flaws of previous task definitions raised in Section 3.1.

#### Principled Evaluation & Contamination Resistance

DiscoGen tasks clearly distinguish between meta-train and meta-test. Since DiscoGen is procedural, and there is no knowledge of the meta-test datasets in meta-training, the potential for test leakage is limited. Even as DiscoGen enters pre-training datasets, this is a step towards fairer evaluation, ensuring DiscoGen remains pertinent for a long time.

**High Diversity** DiscoGen generates highly diverse tasks. As demonstrated in Section 4.1, DiscoGen supports a range of domains with different data structures, filesystem complexities and modules. Since DiscoGen tasks are parameterised combinatorially, its tasks represent a range of difficulties from “*implement a single module for one easy dataset*” to “*implement many modules for many hard datasets*”.

**Unbiased To Initialisation** Agents need not implement full codebases, and the initialisation of editable modules is, in most cases, the inputs and outputs of the module only. In addition to enabling improved creativity in research agents, this allows us to better analyse the biases elicited by ADAs.

**Ease of Adding Tasks** Beyond our currently implemented domains, adding many tasks to DiscoGen is significantly easier than for other suites. For similar effort to adding one task to, say, MLEBench (Chan et al., 2025) or MLGym-Bench (Nathani et al., 2025), DiscoGen can gain potentially millions of new tasks in its support. When the base code for a task domain is complete, adding more tasks is even easier; isolating a new module effectively doubles the number of possible tasks, and adding a new dataset near-triples it.

**Unsaturated Problems** Since tasks can be made more or less difficult by changing the module and dataset configurations (Appendix E), DiscoGen tasks span a wide range of difficulties. We find that the more modules there are to implement, the harder the problem but the higher the potential ceiling. Additionally, almost all datasets currently in DiscoGen have yet to be solved by humans, let alone agents, and adding more, harder datasets is straightforward.

## 5. DiscoBench

Despite the contamination risk that arises from releasing a public benchmark, there is still merit to evaluating ADA performance over a fixed set of DiscoGen tasks that resolve the other flaws from prior benchmarks (Section 3.1).

DiscoBench is an ADA evaluation suite akin to the hand-designed levels built in PCG environments (e.g., (Matthews et al., 2025; Nikulin et al., 2024)). For each task domain in DiscoGen, we create  $m + 1$  tasks;  $m$  tasks where one of the  $m$  modules is active at a time (*DiscoBench Single*), and 1 where all modules are active simultaneously (*DiscoBench All*). We do not include other module combinations to ensure DiscoBench remains manageable and to enable principled expansion of DiscoBench as domains are added. Meta-train and meta-test sets are fixed across all tasks to enable comparison, and are selected pseudo-randomly; very long-to-run datasets are reserved for meta-testing, for computational reasons. These splits are included in Appendix F. ADAs should not be optimised on DiscoBench to ensure it is an appropriate test suite (though the probability of sampling tasks from DiscoBench is non-zero, as in other PCG environments). Since these tasks are not yet public, our evaluation is not subject to contamination; however, this work entering the

public domain exposes it to pretraining or internet search agents. As such, we plan to release a private ‘API’ version of DiscoBench using datasets not mentioned publicly.

### 5.1. Experimental Setup

We explore the performance of different LLMs with the MLGym ADA (Nathani et al., 2025), a ReAct agent (Yao et al., 2023) which can run code, read files and submit the final algorithm when ready, with a fixed action budget. Due to resource constraints, and for reproducibility purposes, we evaluate the performance of open-source language models: Deepseek-V3.2 (DeepSeek-AI et al., 2025), Devstral-2 (Mistral AI, 2025) and GPT-OSS 120B (OpenAI et al., 2025). We include an ‘*all fixed*’ baseline (i.e., the code with no editable modules) for comparison; it is *always* possible for the ADA to implement this. We provide experimental details and hyperparameters in Appendix C, include our generic ADA system prompt in Appendix L.1, and detail cost and compute usage in Appendix C.4.

In both *DiscoBench Single* and *DiscoBench All*, we aggregate scores over three seeds per task and model. We report two per-model success-rates and Elo ratings (Elo, 1978) based on same-dataset comparisons; one for meta-train, and one for meta-test. We report 95% confidence intervals for Elo, estimated using 100 bootstrap samples as in Appendix C. Since agents frequently fail to consistently produce valid solutions for many tasks<sup>2</sup> we penalise failure such that a model with more successful runs dominates one with fewer; this penalty does not apply in baseline comparisons. We also report the total success rate for each model.

To understand how each model could perform without failures, we run additional experiments on *DiscoBench Single* until each model has three successful attempts. However, due to low success rates, this was unaffordable for *DiscoBench All* and a small number of tasks in *DiscoBench Single* as specified in Appendix K. This failure is expected; in Nathani et al. (2025), many frontier closed source models failed over their four attempts in MLGymBench. As such, we omit these tasks from the *Until Success* analysis.

### 5.2. Results

We report results in Table 2, and include a per-task breakdown in Appendix K. To demonstrate that agents can discover interesting and performant algorithms, we discuss two hand-selected algorithms in Appendix I.

Firstly, success rates in *DiscoBench All* are significantly lower than for *DiscoBench Single*, confirming the hypothesis that including more editable modules increases task difficulty. This is explored further in Appendix E, where we sweep over all module combinations in On-Policy RL

<sup>2</sup>Since at least one model produced a valid solution for every task, and the ‘*all fixed*’ baselines follow the same module interface, we have existence proofs that all tasks are solvable.

Table 2. ADA evaluation performance in DiscoBench (Elo Scores with 95% CIs). Bold indicates best mean performance.

| Model                | DiscoBench Single |                          |                          | DiscoBench Single (Until Success) |                          |                          | DiscoBench All |                          |                          |
|----------------------|-------------------|--------------------------|--------------------------|-----------------------------------|--------------------------|--------------------------|----------------|--------------------------|--------------------------|
|                      | Succ.             | Meta-Train               | Meta-Test                | Succ.                             | Meta-Train               | Meta-Test                | Succ.          | Meta-Train               | Meta-Test                |
| Baseline (All Fixed) | —                 | <b>1104</b> [1077, 1136] | <b>1177</b> [1144, 1211] | —                                 | <b>1076</b> [1038, 1108] | <b>1149</b> [1113, 1184] | —              | <b>1409</b> [1297, 1682] | <b>1377</b> [1212, 1595] |
| GPT-OSS 120B         | 68.2%             | 931 [900, 961]           | 962 [933, 993]           | <b>100.0%</b>                     | 888 [853, 914]           | 901 [871, 929]           | 11.4%          | 533 [183, 700]           | 597 [-106, 799]          |
| Devstral2            | 45.9%             | 886 [850, 922]           | 808 [771, 842]           | <b>100.0%</b>                     | 1000 [966, 1029]         | 964 [930, 991]           | <b>34.3%</b>   | 873 [751, 1138]          | <b>1087</b> [971, 1322]  |
| Deepseek-v3.2        | <b>80.0%</b>      | <b>1079</b> [1050, 1108] | 1053 [1020, 1082]        | <b>100.0%</b>                     | <b>1037</b> [1004, 1067] | 987 [960, 1011]          | 25.7%          | <b>1184</b> [1069, 1397] | 940 [831, 1176]          |

and find that the success rates of ADAs *consistently* fall as more editable modules are added. In fact, average success rates for all three models are low. In contrast, we examine *Success@3* rates (i.e., what proportion of tasks had *at least one* successful solution from 3 attempts) in Appendix J, and find that they are 15-30 percentage points higher than the aggregated rates. Considering this performance gap, and that *all* baselines follow the same interface as the editable modules, it is clear that agents struggle to **robustly** produce even simple, well-known algorithms. We find that failures are broadly driven by syntax errors or, often, code overfitting to the meta-train datasets (e.g., hardcoded shapes).

Elo shows a similar pattern; the baseline has a much higher score in *DiscoBench All* over *DiscoBench Single*. Even when agents have three successful solutions; no agent consistently outperforms the baseline to the point of having a higher Elo. Since ADAs do not yet match well-known human algorithms, even though they *could* be implemented and are often *suboptimal* algorithms for the datasets, there is significant clear margin for ADAs to improve.

Deepseek-v3.2 performs well compared to other models in *DiscoBench Single*, both in meta-train and meta-test, but is considerably outperformed by Devstral2 in meta-test for *DiscoBench All*. The relative baseline performance often increases between meta-train and meta-test, suggesting ADAs struggle to discover as generalisable algorithms.

It is important to ensure DiscoBench is diverse; a single model being uniformly dominant would suggest DiscoBench only measures general ability. We explore this using rank-correlation analysis in Appendix G and find high variation in rankings between tasks. In fact, the per-task results (Appendix K) show that even the ranking over datasets within the *same* task varies, demonstrating how different algorithms are better for different datasets and justifying claims of diversity in DiscoGen. Furthermore, this per-task breakdown confirms the range of task difficulties; sometimes agents outperform the baseline, usually they produce weak-but-valid solutions, and often they fail completely.

## 6. Enabling New Research

DiscoGen enables a plethora of new research directions in algorithm discovery. Here, we provide high level proposals for some of them to serve as inspiration for the wider community. In Section 7, we show how DiscoGen can be used for prompt optimisation, demonstrating one such use-case.

## 6.1. Understanding The Pathologies of ADAs

Given the finite task spaces of existing algorithm discovery suites, analysis of the pathologies of algorithm discovery systems is caveated by a limited evaluation scope. This introduces questions over whether shortcomings are intrinsic to agents or simply artefacts of the tasks in which they are evaluated. DiscoGen provides an expansive space in which to run such analysis. If a pathology is exhibited across thousands, rather than tens, of tasks, it becomes easier to understand and mitigate. This research could seek to understand the limitations of creativity (Haase et al., 2025; Franceschelli & Musolesi, 2025), instruction-following (Ouyang et al., 2022; Zeng et al., 2024) or mode collapse (such as hyper-parameter tuning, as in (Nathani et al., 2025)). This would enable a more scientific approach to agents development.

## 6.2. Learning to Discover Algorithms

The problem-solving and reasoning ability of language models has significantly improved since the introduction of RL (Ouyang et al., 2022; Shao et al., 2024; Gehring et al., 2025; Kazemnejad et al., 2025) or evolution (Sarkar et al., 2025; Qiu et al., 2025) post-training. However, earlier works generally focus on mathematics or programming, where there are many verifiable problems to learn from. Given it is now possible to sample millions of unique algorithm discovery tasks, the findings from these task-rich domains could be transferred to train capable algorithm discovery models.

## 6.3. Sampling Hard-Yet-Learnable Discovery Tasks

Prior work has shown how autocurricula (Dennis et al., 2021; Parker-Holder et al., 2022a; Foster et al., 2025) can sample hard-yet-learnable tasks to improve minimum expected performance bounds (Beukman et al., 2024) and improve training efficiency (Foster et al., 2025). Since tasks in DiscoGen are of varying difficulty, it is naturally suited to curriculum methods, and their application could improve the performance and efficiency over random task sampling. This is especially important in ADA optimisation, where task completion times can vary by orders of magnitude.

## 6.4. Algorithm World Models

Copet et al. (2025) mid-train an LLM to replicate a code interpreter’s state as it ran. Such training promises to help an LLM’s programming abilities; if it can replicate the outputs of code, it should produce more correct implementations. However, in their work, models were trained on vast amounts of data. Combining DiscoGen with similar computational resources, a large-scale, high quality dataset

of algorithm-performance pairs from different tasks could be collected. This dataset could be used to mid-train an ‘algorithm world model’; a language model predicting how editing modules affects final performance. This could be integrated directly into ADAs to improve their performance, fine-tuned using the research proposals above, or serve as the basis of an LLM-judge (Zheng et al., 2023) as below.

### 6.5. Training LLM-As-A-Judge in Tree-Search ADAs

“*How to explore?*” is an open question in algorithm discovery and AI research (Toledo et al., 2025). Some agents designed for automated research and algorithm discovery use tree-search (Jiang et al., 2025; Toledo et al., 2025), but evaluating a tree’s leaves requires running expensive inner-loop trainings. Instead, using an LLM or otherwise trained model to evaluate algorithms could act as a filter, selecting promising leaves to run (Yu et al., 2025; Herr et al., 2025) or acting as a value function (Wang et al., 2025b) to skip inner-loop training and reduce the cost of search.

However, off-the-shelf judge performance would likely be poor, since we ideally want to evaluate super-human (and thus, out-of-distribution) algorithms. Using data generated using DiscoGen, it would be possible to train either a full model or a value prediction head that could be integrated into Monte-Carlo Tree Search (Kocsis & Szepesvári, 2006).

### 6.6. Symbolic Evolution of Algorithms

Since DiscoGen provides templates for module (i.e., defining inputs and outputs), and the rest of an algorithm’s code is pre-implemented, it is well-placed for working with non LLM-based algorithm discovery methods; for example, symbolic search or black-box meta-learning. This could include developing methods using genetic programming, as in (Ramachandran et al., 2017; Zheng et al., 2022; Chen et al., 2023b; Goldie et al., 2025), or black-box evolution (e.g., (Lu et al., 2022; Metz et al., 2022b; Goldie et al., 2024)).

### 6.7. Self-Improving ADAs

An alternative to manually designing ADAs is letting them build their *own* scaffolds (e.g., (Hu et al., 2024; Zhang et al., 2025; Wang et al., 2025a)). Such open-ended self-improvement offers data-driven agent improvements. However, optimising an agent’s scaffold on existing algorithm discovery suites could lead to overfitting to specific problems. With DiscoGen, such systems could be run near-indefinitely with low risk of overfitting to individual tasks, aiding the automated discovery of super-human agents.

## 7. A Case Study: Prompt Optimisation

Given the power of prompting in maximising LLM performance (Lester et al., 2021; Fernando et al., 2024), we explore prompt refinement using DiscoGen. We query an ‘ADA-Optimisation’ LLM, separate to the ADA, to automatically improve prompts in each iteration of the ADA optimisation loop, based on past meta-train and meta-test

*Table 3.* ADA evaluation performance after prompt optimisation (Elo Scores with 95% CIs). Bold indicates the best result.

| $K_{tasks}$ | DiscoBench Combined |                          |                          |
|-------------|---------------------|--------------------------|--------------------------|
|             | Succ.               | Meta-Train               | Meta-Test                |
| 1           | 70.6%               | 956 [939, 978]           | 957 [927, 977]           |
| 5           | 75.3%               | 1014 [1000, 1033]        | 973 [947, 993]           |
| 10          | 72.0%               | 969 [949, 989]           | 1000 [980, 1022]         |
| 30          | 78.7%               | <b>1061</b> [1040, 1079] | <b>1071</b> [1049, 1096] |

performance in sampled DiscoGen tasks. Since the ‘prompting’ LLM is queried infrequently, we use a more expensive closed-source model for prompt optimisation; Claude Sonnet 4.5 (Anthropic, 2025b). As the best performing LLM tested in DiscoBench, the ADA uses DeepSeek-V3.2.

Our experiment explores how task diversity correlates to ADA optimisation performance over 30 steps. We control task quantity,  $K_{tasks}$ , by sampling from DiscoGen at different frequencies; when  $K_{tasks} = 1$ , we tune the prompt on the same task 30 times, and when  $K_{tasks} = 30$ , we tune it in different tasks every ADA optimisation iteration. To prevent domain bias, we uniformly sample the task domain *before* the task configuration. We present results in Table 3, using DiscoBench for ADA evaluation and the Elo methodology from Section 5.2. DiscoBench tasks were *not* sampled in ADA optimisation. For clarity, we combine *DiscoBench Single* and *DiscoBench All* into a combined set; we expand this in Appendix H. Further experimental details are in Appendix C, and the discovered prompts in Appendix L.

We find a clear monotonic trend between  $K_{tasks}$  and prompt performance in meta-test on DiscoBench; as  $K_{tasks}$  increases, meta-test performance improves. Meta-train is less consistent ( $K_{tasks} = 5$  outperforms  $K_{tasks} = 10$ ), but the prompt optimised over the most tasks achieves the best performance. This is intuitive; considering the prompts themselves (Appendix L.3), the prompt for  $K_{tasks} = 30$  emphasises general machine learning principles, unlike the prompt for  $K_{tasks} = 1$ , which is overfit to optimising two RL environments. Such a finding emphasises the value of procedural generation for algorithm discovery, and underlines the potential of the research agenda in Section 6.

## 8. Conclusion

In this paper, we introduced DiscoGen, a procedural generator of algorithm discovery tasks. We motivated the design of DiscoGen by the shortcomings of existing algorithm discovery suites, such as poor evaluation methodology and limited scale. We demonstrated that DiscoGen overcame many of these flaws, and used it to establish DiscoBench, a set of tasks for evaluating ADAs. We subsequently introduced a number of possible research avenues enabled by DiscoGen. To justify these, we used DiscoGen for ADA prompt optimisation and showed that prompt performance improved with the number of tasks it was optimised for.

**Future Work** Beyond the research *enabled* by DiscoGen in Section 6, there are a number of avenues to improve DiscoGen *itself*. Firstly, expanding the number of domains, modules and datasets supported by DiscoGen with contributions from community ML experts would enable even more diverse generation. Additionally, we plan to develop a private DiscoBench suite for ADA evaluation which is accessible by API only (i.e., using module/dataset combinations not available in DiscoGen). This would solve the ‘data contamination’ limitation affecting all algorithm discovery benchmarks, including DiscoBench moving forward.

## Impact Statement

Considering the social, economic and safety effects of automated algorithm discovery is crucial due to the potential magnitude of its impacts. In this work, we are considering not only how a specific algorithm can be meta-learned (which has more limited impact), as in much prior work, but how to *optimise* agents towards the development of new machine learning algorithms. While we believe there are a range of significant benefits that such work can lead to, we are also wary and considerate of the risks.

It is often preferable to develop models with specialist capabilities; consistently focusing on generalist improvements can introduce safety concerns (Bommasani et al., 2022; Weidinger et al., 2021), and may not even be the most effective way to enhance desired capabilities (Belcak et al., 2025). We believe algorithm discovery provides a surgical tool with which to automate research, offering a much more specific objective than ‘*automating research*’, as in a lot of automated AI scientist literature (Lu et al., 2024b; Intology, 2025; Yamada et al., 2025). Rather than giving black-box agents the ability to decide what problems they work on freely, algorithm discovery (and specifically, DiscoGen) is designed to involve a human-in-the-loop; discovery agents are only allowed to make changes to certain files, with *all* other changes being overwritten at test time. Furthermore, despite the massive task space available in DiscoGen, all tasks are well-defined, scoped and based on human-selected codebases, meaning they are constrained. Given the current suite of ten domains, we do not believe any tasks deemed unsafe *could* be sampled from DiscoGen. However, given our hope for community contributions in the future, it is important to ensure that this remains the case moving forward.

As we emphasise optimisation for algorithm discovery *only*, DiscoGen provides a platform for AI-Human Co-Improvement (Weston & Foerster, 2025). Crucially, rather than training towards more generally-able agents (where algorithm discovery capabilities are improved simultaneously with adverse capabilities), optimising on DiscoGen improves performance *directly* for human-designed algorithm discovery tasks. We believe this type of scoped optimisation is a necessary path towards safe super-human agents;

it is safer to develop specifically super-human agents than developing poorly understood general superintelligence.

It is important to consider that development of automated algorithm discovery systems can pose ethical and economic concerns. While we emphasise general and beneficial task domains, harmful actors could instead look to optimise agents in unethical task domains. Fully mitigating this misuse is beyond the scope of our paper, but it is important to ensure DiscoGen is comprised only of broadly beneficial domains to limit any negative behaviours. Similarly, discovering better machine learning algorithms automatically enables them to be used for harmful datasets; while this is a risk of all machine learning research, it is important to acknowledge that DiscoGen accelerates their development. At the least, to reduce the risk of directly optimising on these datasets, it is important to ensure there is broad human-alignment in base foundation models, such that it is hard to optimise for misaligned behaviour (Bai et al., 2022). Furthermore, one counter-measure to these risks is to ensure AI safety tasks are included within DiscoGen. For example, discovering algorithms for the *Model Unlearning* task in DiscoGen (Yao et al., 2024) is one avenue for reducing unwanted behaviours in foundation models.

Ensuring people from affected fields are kept in-the-loop as these systems develop is necessary for ensuring automated algorithm systems *complement*, rather than *replace*, humans. The potential impact of AI on the labour market is significant (Eloundou et al., 2024), and thus developing systems which protect the role of humans is important. We believe that automated algorithm discovery provides a strong balance of developing research breakthroughs using AI, while maintaining the agency and empowerment of humans who design the problem settings for it to operate in. In general, this fulfils the idea that research agents are most effective when provided as tools to humans (Gottweis et al., 2025; Shneiderman, 2022), rather than acting to substitute them.

Moreover, the requirements for operating in this field are high; running or querying large language models is expensive, and optimising research agents is more so. This introduces large financial (Chen et al., 2023a) and environmental (Strubell et al., 2019; Faiz et al., 2024) costs, which should be considered when experimenting with DiscoGen. One high-impact area for future research would involve developing energy-efficient automated algorithm discovery systems; in fact, these could be developed automatically, using a well-defined objective in combination with the research proposal of Section 6.7. Furthermore, designing task domains in which *efficiency*, rather than *performance*, is the objective would offset the costs of running ADAs with the potential downstream savings enabled by such algorithms.

Democratisation of AI is one necessary tool for ensuring it can act in the benefit of all, rather than a few major players. We believe DiscoGen helps with this in two ways. Firstly,

in making a large-scale research environment for algorithm discovery available, we believe that such research is made more feasible outside of industrial frontier labs. Secondly, by emphasising the development of *specialist*, rather than *generalist*, agents, we hope to enable a landscape of research using smaller models which require less pre-training data. However, it would be naïve to suggest that research in automated research and algorithm discovery is not expensive, and requires significant hardware resources. Moving forward, it is necessary to ensure the development of these tools is accessible to a wide range of researchers, from academia through to small and large parts of industry.

## References

- Abdin, M. I., Ade Jacobs, S., Awan, A. A., Aneja, J., Awadallah, A., Hassan Awadalla, H., Bach, N., Bahree, A., Bakhtiari, A., Behl, H., Benhaim, A., Bilenko, M., Bjorck, J., Bubbeck, S., Cai, M., Mendes, C. C. T., Chen, W., Chaudhary, V., Chopra, P., Giorno, A. D., de Rosa, G., Dixon, M., Eldan, R., Iter, D., Goswami, A., Guanasekar, S., Haider, E., Hao, J., Hewett, R. J., Huynh, J., Javaheripi, M., Jin, X., Kauffmann, P., Karampatziakis, N., Kim, D., Khademi, M., Kurilenko, L., Lee, J. R., Lee, Y. T., Li, Y., Liang, C., Liu, W., Lin, X. E., Lin, Z., Madan, P., Mitra, A., Modi, H., Nguyen, A., Norick, B., Patra, B., Perez-Becker, D., Portet, T., Pryzant, R., Qin, H., Radmilac, M., Rosset, C., Roy, S., Saarikivi, O., Saied, A., Salim, A., Santacroce, M., Shah, S., Shang, N., Sharma, H., Song, X., Ruwase, O., Wang, X., Ward, R., Wang, G., Witte, P., Wyatt, M., Xu, C., Xu, J., Xu, W., Yadav, S., Yang, F., Yang, Z., Yu, D., Zhang, C., Zhang, C., Zhang, J., Zhang, L. L., Zhang, Y., Zhang, Y., and Zhou, X. Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone. Technical Report MSR-TR-2024-12, Microsoft, August 2024. URL <https://www.microsoft.com/en-us/research/publication/phi-3-technical-report-a-highly-capable-language-model-locally-on-your-phone/>.
- Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A., and Bellemare, M. G. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021.
- Alfano, C., Yuan, R., and Rebeschini, P. A Novel Framework for Policy Mirror Descent with General Parametrization and Linear Convergence, February 2023. URL <https://arxiv.org/abs/2301.13139>. arXiv:2301.13139 [cs, math, stat].
- Andrychowicz, M., Denil, M., Colmenarejo, S. G., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and de Freitas, N. Learning to learn by gradient descent by gradient descent. In *Proceedings of the 30th international conference on neural information processing systems*, NIPS'16, pp. 3988–3996, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 978-1-5108-3881-9. Number of pages: 9 Place: Barcelona, Spain.
- Anthropic. Claude code, 2025a. URL <https://www.cliaude.com/product/clause-code>. Accessed: 2025-12-08.
- Anthropic. Introducing claude sonnet 4.5, September 2025b. URL <https://www.anthropic.com/news/clause-sonnet-4-5>.
- Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., Chen, C., Olsson, C., Olah, C., Hernandez, D., Drain, D., Ganguli, D., Li, D., Tran-Johnson, E., Perez, E., Kerr, J., Mueller, J., Ladish, J., Landau, J., Ndousse, K., Lukosuite, K., Lovitt, L., Sellitto, M., Elhage, N., Schiefer, N., Mercado, N., DasSarma, N., Lasenby, R., Larson, R., Ringer, S., Johnston, S., Kravec, S., Showk, S. E., Fort, S., Lanham, T., Telleen-Lawton, T., Conerly, T., Henighan, T., Hume, T., Bowman, S. R., Hatfield-Dodds, Z., Mann, B., Amodei, D., Joseph, N., McCandlish, S., Brown, T., and Kaplan, J. Constitutional AI: Harmlessness from AI Feedback, December 2022. URL <http://arxiv.org/abs/2212.08073>. arXiv:2212.08073 [cs].
- Bauer, J., Baumli, K., Baveja, S., Behbahani, F., Bhoopchand, A., Bradley-Schmiege, N., Chang, M., Clay, N., Collister, A., Dasagi, V., Gonzalez, L., Gregor, K., Hughes, E., Kashem, S., Loks-Thompson, M., Openshaw, H., Parker-Holder, J., Pathak, S., Perez-Nieves, N., Rakicevic, N., Rocktäschel, T., Schroecker, Y., Sygnowski, J., Tuyls, K., York, S., Zacherl, A., and Zhang, L. Human-Timescale Adaptation in an Open-Ended Task Space, January 2023. URL <http://arxiv.org/abs/2301.07608>. arXiv:2301.07608 [cs], Adaptive Agent Team.
- Bechtle, S., Molchanov, A., Chebotar, Y., Grefenstette, E., Righetti, L., Sukhatme, G., and Meier, F. Meta learning via learned loss. In *2020 25th international conference on pattern recognition (ICPR)*, pp. 4161–4168. IEEE, 2021.
- Beck, J., Vuorio, R., Liu, E. Z., Xiong, Z., Zintgraf, L., Finn, C., and Whiteson, S. A Survey of Meta-Reinforcement Learning, January 2023. arXiv: 2301.08028 [cs] Issue: arXiv:2301.08028.
- Belcak, P., Heinrich, G., Diao, S., Fu, Y., Dong, X., Muralidharan, S., Lin, Y. C., and Molchanov, P. Small Language Models are the Future of Agentic AI, September 2025. URL <http://arxiv.org/abs/2506.02153>. arXiv:2506.02153 [cs].

- Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.
- Beukman, M., Coward, S., Matthews, M., Fellows, M., Jiang, M., Dennis, M., and Foerster, J. Refining minimax regret for unsupervised environment design. In *International Conference on Machine Learning*. PMLR, 2024.
- Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., Arx, S. v., Bernstein, M. S., Bohg, J., Bosslut, A., Brunskill, E., Brynjolfsson, E., Buch, S., Card, D., Castellon, R., Chatterji, N., Chen, A., Creel, K., Davis, J. Q., Demszky, D., Donahue, C., Doumbouya, M., Durmus, E., Ermon, S., Etchemendy, J., Ethayarajh, K., Fei-Fei, L., Finn, C., Gale, T., Gillespie, L., Goel, K., Goodman, N., Grossman, S., Guha, N., Hashimoto, T., Henderson, P., Hewitt, J., Ho, D. E., Hong, J., Hsu, K., Huang, J., Icard, T., Jain, S., Jurafsky, D., Kalluri, P., Karamcheti, S., Keeling, G., Khani, F., Khattab, O., Koh, P. W., Krass, M., Krishna, R., Kuditipudi, R., Kumar, A., Ladhak, F., Lee, M., Lee, T., Leskovec, J., Levant, I., Li, X. L., Li, X., Ma, T., Malik, A., Manning, C. D., Mirchandani, S., Mitchell, E., Munyikwa, Z., Nair, S., Narayan, A., Narayanan, D., Newman, B., Nie, A., Niebles, J. C., Nilforoshan, H., Nyarko, J., Ogut, G., Orr, L., Papadimitriou, I., Park, J. S., Piech, C., Portelance, E., Potts, C., Raghunathan, A., Reich, R., Ren, H., Rong, F., Roohani, Y., Ruiz, C., Ryan, J., Ré, C., Sadigh, D., Sagawa, S., Santhanam, K., Shih, A., Srinivasan, K., Tamkin, A., Taori, R., Thomas, A. W., Tramèr, F., Wang, R. E., Wang, W., Wu, B., Wu, J., Wu, Y., Xie, S. M., Yasunaga, M., You, J., Zaharia, M., Zhang, M., Zhang, T., Zhang, X., Zhang, Y., Zheng, L., Zhou, K., and Liang, P. On the Opportunities and Risks of Foundation Models, July 2022. URL <http://arxiv.org/abs/2108.07258>. arXiv:2108.07258 [cs].
- Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL <http://github.com/jax-ml/jax>.
- Cao, Y. and Yang, J. Towards making systems forget with machine unlearning. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP ’15, pp. 463–480, USA, 2015. IEEE Computer Society. ISBN 9781467369497. doi: 10.1109/SP.2015.35. URL <https://doi.org/10.1109/SP.2015.35>.
- Chan, J. S., Chowdhury, N., Jaffe, O., Aung, J., Sherburn, D., Mays, E., Starace, G., Liu, K., Maksin, L., Patwardhan, T., Weng, L., and Mądry, A. MLE-bench: Evaluating Machine Learning Agents on Machine Learning Engineering, February 2025. URL <http://arxiv.org/abs/2410.07095>. arXiv:2410.07095 [cs].
- Chen, L., Zaharia, M., and Zou, J. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance, May 2023a. URL <http://arxiv.org/abs/2305.05176>. arXiv:2305.05176 [cs].
- Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code. 2021.
- Chen, X., Liang, C., Huang, D., Real, E., Wang, K., Liu, Y., Pham, H., Dong, X., Luong, T., Hsieh, C.-J., Lu, Y., and Le, Q. V. Symbolic Discovery of Optimization Algorithms, May 2023b. URL <http://arxiv.org/abs/2302.06675>. arXiv:2302.06675 [cs].
- Chen, Y., Li, J., Xiao, H., Jin, X., Yan, S., and Feng, J. Dual path networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS’17, pp. 4470–4478, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.
- Chevalier-Boisvert, M., Dai, B., Towers, M., Perez-Vicente, R., Willems, L., Lahlou, S., Pal, S., Castro, P. S., and Terry, J. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. In *Advances in Neural Information Processing Systems 36, New Orleans, LA, USA*, December 2023.
- Chollet, F., Knoop, M., Kamradt, G., and Landers, B. ARC Prize 2024: Technical Report, January 2025. URL <http://arxiv.org/abs/2412.04604>. arXiv:2412.04604 [cs].
- Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, Y., Wang, X., Dehghani, M., Brahma, S., Webson, A., Gu, S. S., Dai, Z., Suzgun, M., Chen, X., Chowdhery, A., Castro-Ros, A., Pellat, M., Robinson, K., Valter, D., Narang, S., Mishra, G., Yu, A., Zhao, V., Huang, Y., Dai, A., Yu, H., Petrov, S., Chi, E. H., Dean, J., Devlin, J., Roberts, A., Zhou, D., Le, Q. V., and Wei, J. Scaling Instruction-Finetuned Language Models. *Journal*

- of Machine Learning Research, 25(70):1–53, 2024. URL <http://jmlr.org/papers/v25/23-0870.html>.
- Clevert, D.-A., Unterthiner, T., and Hochreiter, S. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs), February 2016. URL <http://arxiv.org/abs/1511.07289>. arXiv:1511.07289 [cs].
- Clune, J. AI-GAs: AI-generating algorithms, an alternate paradigm for producing general artificial intelligence, May 2019. URL <https://arxiv.org/abs/1905.10985v2>.
- Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging Procedural Generation to Benchmark Reinforcement Learning, July 2020. URL <http://arxiv.org/abs/1912.01588>. arXiv:1912.01588 [cs].
- Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021a.
- Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training Verifiers to Solve Math Word Problems, October 2021b. URL <https://arxiv.org/abs/2110.14168v2>.
- Copet, J., Carbonneaux, Q., Cohen, G., Gehring, J., Kahn, J., Kossen, J., Kreuk, F., McMilin, E., Meyer, M., Wei, Y., Zhang, D., Zheng, K., Armengol-Estabé, J., Bashiri, P., Beck, M., Chambon, P., Charnalia, A., Cummins, C., Decugis, J., Fischies, Z. V., Fleuret, F., Gloeckle, F., Gu, A., Hassid, M., Haziza, D., Idrissi, B. Y., Keller, C., Kindi, R., Leather, H., Maimon, G., Markosyan, A., Massa, F., Mazaré, P.-E., Mella, V., Murray, N., Muzumdar, K., O’Hearn, P., Pagliardini, M., Pedchenko, D., Remez, T., Seeker, V., Selvi, M., Sultan, O., Wang, S., Wehrstedt, L., Yoran, O., Zhang, L., Cohen, T., Adi, Y., and Synnaeve, G. CWM: An Open-Weights LLM for Research on Code Generation with World Models, September 2025. URL <http://arxiv.org/abs/2510.02387>. arXiv:2510.02387 [cs], FAIR CodeGen.
- Cranmer, M., Sanchez-Gonzalez, A., Battaglia, P., Xu, R., Cranmer, K., Spergel, D., and Ho, S. Discovering Symbolic Models from Deep Learning with Inductive Biases, November 2020. URL <http://arxiv.org/abs/2006.11287>.
- Dao, T. and Gu, A. Transformers are ssms: generalized models and efficient algorithms through structured state space duality. In *Proceedings of the 41st International Conference on Machine Learning*, ICML’24. JMLR.org, 2024.
- DeepSeek-AI, Liu, A., Mei, A., Lin, B., Xue, B., Wang, B., Xu, B., Wu, B., Zhang, B., Lin, C., Dong, C., Lu, C., Zhao, C., Deng, C., Xu, C., Ruan, C., Dai, D., Guo, D., Yang, D., Chen, D., Li, E., Zhou, F., Lin, F., Dai, F., Hao, G., Chen, G., Li, G., Zhang, H., Xu, H., Li, H., Liang, H., Wei, H., Zhang, H., Luo, H., Ji, H., Ding, H., Tang, H., Cao, H., Gao, H., Qu, H., Zeng, H., Huang, J., Li, J., Xu, J., Hu, J., Chen, J., Xiang, J., Yuan, J., Cheng, J., Zhu, J., Ran, J., Jiang, J., Qiu, J., Li, J., Song, J., Dong, K., Gao, K., Guan, K., Huang, K., Zhou, K., Huang, K., Yu, K., Wang, L., Zhang, L., Wang, L., Zhao, L., Yin, L., Guo, L., Luo, L., Ma, L., Wang, L., Zhang, L., Di, M. S., Xu, M. Y., Zhang, M., Zhang, M., Tang, M., Zhou, M., Huang, P., Cong, P., Wang, P., Wang, Q., Zhu, Q., Li, Q., Chen, Q., Du, Q., Xu, R., Ge, R., Zhang, R., Pan, R., Wang, R., Yin, R., Xu, R., Shen, R., Zhang, R., Liu, S. H., Lu, S., Zhou, S., Chen, S., Cai, S., Chen, S., Hu, S., Liu, S., Hu, S., Ma, S., Wang, S., Yu, S., Zhou, S., Pan, S., Zhou, S., Ni, T., Yun, T., Pei, T., Ye, T., Yue, T., Zeng, W., Liu, W., Liang, W., Pang, W., Luo, W., Gao, W., Zhang, W., Gao, X., Wang, X., Bi, X., Liu, X., Wang, X., Chen, X., Zhang, X., Nie, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yu, X., Li, X., Yang, X., Li, X., Chen, X., Su, X., Pan, X., Lin, X., Fu, X., Wang, Y. Q., Zhang, Y., Xu, Y., Ma, Y., Li, Y., Li, Y., Zhao, Y., Sun, Y., Wang, Y., Qian, Y., Yu, Y., Zhang, Y., Ding, Y., Shi, Y., Xiong, Y., He, Y., Zhou, Y., Zhong, Y., Piao, Y., Wang, Y., Chen, Y., Tan, Y., Wei, Y., Ma, Y., Liu, Y., Yang, Y., Guo, Y., Wu, Y., Wu, Y., Cheng, Y., Ou, Y., Xu, Y., Wang, Y., Gong, Y., Wu, Y., Zou, Y., Li, Y., Xiong, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Wu, Z. F., Ren, Z. Z., Zhao, Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Xie, Z., Zhang, Z., Hao, Z., Gou, Z., Ma, Z., Yan, Z., Shao, Z., Huang, Z., Wu, Z., Li, Z., Zhang, Z., Xu, Z., Wang, Z., Gu, Z., Zhu, Z., Li, Z., Zhang, Z., Xie, Z., Gao, Z., Pan, Z., Yao, Z., Feng, B., Li, H., Cai, J. L., Ni, J., Xu, L., Li, M., Tian, N., Chen, R. J., Jin, R. L., Li, S. S., Zhou, S., Sun, T., Li, X. Q., Jin, X., Shen, X., Chen, X., Song, X., Zhou, X., Zhu, Y. X., Huang, Y., Li, Y., Zheng, Y., Zhu, Y., Ma, Y., Huang, Z., Xu, Z., Zhang, Z., Ji, D., Liang, J., Guo, J., Chen, J., Xia, L., Wang, M., Li, M., Zhang, P., Chen, R., Sun, S., Wu, S., Ye, S., Wang, T., Xiao, W. L., An, W., Wang, X., Sun, X., Wang, X., Tang, Y., Zha, Y., Zhang, Z., Ju, Z., Zhang, Z., and Qu, Z. DeepSeek-V3.2: Pushing the Frontier of Open Large Language Models, December 2025. URL <http://arxiv.org/abs/2512.02556>. arXiv:2512.02556 [cs].
- Dennis, M., Jaques, N., Vinitsky, E., Bayen, A., Russell, S., Critch, A., and Levine, S. Emergent Complexity and Zero-shot Transfer via Unsupervised Environment Design, February 2021. URL <http://arxiv.org/abs/2012.02096>.
- Dong, Y., Jiang, X., Liu, H., Jin, Z., Gu, B., Yang, M., and

- Li, G. Generalization or memorization: Data contamination and trustworthy evaluation for large language models. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 12039–12050, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.716. URL <https://aclanthology.org/2024.findings-acl.716/>.
- Dorna, V., Mekala, A., Zhao, W., McCallum, A., Lipton, Z. C., Kolter, J. Z., and Maini, P. OpenUnlearning: Accelerating LLM unlearning via unified benchmarking of methods and metrics. *arXiv preprint arXiv:2506.12618*, 2025. URL <https://arxiv.org/abs/2506.12618>.
- Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL <https://openreview.net/forum?id=YicbFdNTTy>.
- Eimer, T., Lindauer, M., and Raileanu, R. Hyperparameters in Reinforcement Learning and How To Tune Them, June 2023. URL <http://arxiv.org/abs/2306.01324>. arXiv:2306.01324 [cs].
- Eldan, R. and Li, Y. Tinystories: How small can language models be and still speak coherent english?, 2023. URL <https://arxiv.org/abs/2305.07759>.
- Elo, A. E. *The Rating of Chessplayers, Past and Present*. Arco Pub, 1978.
- Eloundou, T., Manning, S., Mishkin, P., and Rock, D. Gpts are gpts: Labor market impact potential of llms. *Science*, 384(6702):1306–1308, 2024. doi: 10.1126/science.adj0998. URL <https://www.science.org/doi/abs/10.1126/science.adj0998>.
- Faiz, A., Kaneda, S., Wang, R., Osi, R., Sharma, P., Chen, F., and Jiang, L. LLMCarbon: Modeling the end-to-end Carbon Footprint of Large Language Models, January 2024. URL <http://arxiv.org/abs/2309.14393>. arXiv:2309.14393 [cs].
- Fernando, C., Banarse, D., Michalewski, H., Osindero, S., and Rocktäschel, T. Promptbreeder: self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning, ICML’24*. JMLR.org, 2024.
- Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. Efficient and Robust Automated Machine Learning. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL [https://proceedings.neurips.cc/paper\\_files/paper/2015/file/11d0e6287202fcfd83f79975ec59a3a6-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2015/file/11d0e6287202fcfd83f79975ec59a3a6-Paper.pdf).
- Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., and Hutter, F. Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning, October 2022. URL <http://arxiv.org/abs/2007.04074>. arXiv:2007.04074 [cs].
- Foster, T., Sims, A., Forkel, J., and Foerster, J. N. LILO: Learning to reason at the frontier of learnability. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL <https://openreview.net/forum?id=8HYeWMf0W3>.
- Franceschelli, G. and Musolesi, M. On the Creativity of Large Language Models. *AI & SOCIETY*, 40(5):3785–3795, June 2025. ISSN 0951-5666, 1435-5655. doi: 10.1007/s00146-024-02127-3. URL <http://arxiv.org/abs/2304.00008>. arXiv:2304.00008 [cs].
- Frans, K. and Isola, P. Powderworld: A Platform for Understanding Generalization via Rich Task Distributions, October 2023. URL <http://arxiv.org/abs/2211.13051>. arXiv:2211.13051 [cs].
- Freeman, C. D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I., and Bachem, O. Brax - a differentiable physics engine for large scale rigid body simulation, 2021. URL <http://github.com/google/brax>.
- Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., and Neubig, G. PAL: Program-aided Language Models, November 2022. URL <https://arxiv.org/abs/2211.10435v2>.
- Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. The language model evaluation harness, 07 2024. URL <https://zenodo.org/records/12608602>.
- Garnett, R. *Bayesian Optimization*. Cambridge University Press, 2023.
- Gehring, J., Zheng, K., Copet, J., Mella, V., Carboneaux, Q., Cohen, T., and Synnaeve, G. RLEF: Grounding Code LLMs in Execution Feedback with Reinforcement Learning, February 2025. URL <http://arxiv.org/abs/2410.02089>. arXiv:2410.02089 [cs].
- Gideoni, Y., Tang, Y., Risi, S., and Gal, Y. Random baselines for simple code problems are competitive with code evolution. In *NeurIPS 2025 Fourth Workshop on Deep Learning for Code*, 2025. URL <https://openreview.net/forum?id=rbVIpmbTC>.

- Goldie, A. D., Lu, C., Jackson, M. T., Whiteson, S., and Foerster, J. N. Can Learned Optimization Make Reinforcement Learning Less Difficult? In *Advances in Neural Information Processing Systems*, volume 37, pp. 5454–5497, 2024.
- Goldie, A. D., Wang, Z., Cohen, J., Foerster, J. N., and Whiteson, S. How Should We Meta-Learn Reinforcement Learning Algorithms? May 2025. URL <https://openreview.net/forum?id=jKzQ6af2DU#discussion>.
- Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- Google. URL <https://gemini.google/overview/deepr-research/>. [Accessed 12-01-2026].
- Gottweis, J., Weng, W.-H., Daryin, A., Tu, T., Palepu, A., Sirkovic, P., Myaskovsky, A., Weissenberger, F., Rong, K., Tanno, R., Saab, K., Popovici, D., Blum, J., Zhang, F., Chou, K., Hassidim, A., Gokturk, B., Vahdat, A., Kohli, P., Matias, Y., Carroll, A., Kulkarni, K., Tomasev, N., Guan, Y., Dhillon, V., Vaishnav, E. D., Lee, B., Costa, T. R. D., Penadés, J. R., Peltz, G., Xu, Y., Pawlosky, A., Karthikesalingam, A., and Natarajan, V. Towards an AI co-scientist, February 2025. URL <http://arxiv.org/abs/2502.18864>. arXiv:2502.18864 [cs].
- Haase, J., Hanel, P. H. P., and Pokutta, S. Has the Creativity of Large-Language Models peaked? An analysis of inter- and intra-LLM variability, April 2025. URL <http://arxiv.org/abs/2504.12320>. arXiv:2504.12320 [cs].
- Hafner, D. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*, 2021.
- Hallak, A., Castro, D. D., and Mannor, S. Contextual Markov Decision Processes, February 2015. URL <http://arxiv.org/abs/1502.02259>. arXiv:1502.02259 [stat].
- Hamin, M. and Edelman, B. Cheating on ai agent evaluations. Technical report, Center for AI Standards and Innovation (CAISI), 2025.
- He, K., Zhang, X., Ren, S., and Sun, J. Deep Residual Learning for Image Recognition, December 2015. URL <http://arxiv.org/abs/1512.03385>. arXiv:1512.03385 [cs].
- He, X., Zhao, K., and Chu, X. AutoML: A Survey of the State-of-the-Art. *Knowledge-Based Systems*, 212:106622, January 2021. ISSN 09507051. doi: 10.1016/j.knosys.2020.106622. URL <http://arxiv.org/abs/1908.00709> [cs].
- Hendrycks, D. and Dietterich, T. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.
- Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring Massive Multitask Language Understanding, January 2021a. URL <http://arxiv.org/abs/2009.03300>. arXiv:2009.03300 [cs].
- Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021b.
- Herr, N., Rocktäschel, T., and Raileanu, R. LLM-First Search: Self-Guided Exploration of the Solution Space, June 2025. URL <http://arxiv.org/abs/2506.05213> [cs].
- Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., Driessche, G. v. d., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O., and Sifre, L. Training Compute-Optimal Large Language Models, March 2022. URL <http://arxiv.org/abs/2203.15556>. arXiv:2203.15556 [cs].
- Hu, S., Lu, C., and Clune, J. Automated Design of Agentic Systems, August 2024. URL <http://arxiv.org/abs/2408.08435>.
- Huang, Q., Vora, J., Liang, P., and Leskovec, J. MLAgentBench: Evaluating Language Agents on Machine Learning Experimentation, April 2024. URL <http://arxiv.org/abs/2310.03302>. arXiv:2310.03302 [cs].
- Huang, S., Dossa, R. F. J., Raffin, A., Kanervisto, A., and Wang, W. The 37 implementation details of proximal policy optimization. In *ICLR Blog Track*, 2022a. URL <https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/>. <https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/>.
- Huang, S., Dossa, R. F. J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., and Araújo, J. G. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022b. URL <http://jmlr.org/papers/v23/21-1342.html>.
- Huang, S., Cheng, T., Liu, J. K., Hao, J., Song, L., Xu, Y., Yang, J., Liu, J., Zhang, C., Chai, L., Yuan, R., Zhang, Z., Fu, J., Liu, Q., Zhang, G., Wang, Z., Qi, Y., Xu, Y.,

- and Chu, W. Opencoder: The open cookbook for top-tier code large language models, 2025. URL <https://arxiv.org/abs/2411.04905>.
- Hubert, T., Mehta, R., Sartran, L., Horváth, M. Z., Žužić, G., Wieser, E., Huang, A., Schrittweiser, J., Schroecker, Y., Masoom, H., Bertolli, O., Zahavy, T., Mandhane, A., Yung, J., Beloshapka, I., Ibarz, B., Veeriah, V., Yu, L., Nash, O., Lezeau, P., Mercuri, S., Sönne, C., Mehta, B., Davies, A., Zheng, D., Pedregosa, F., Li, Y., von Glehn, I., Rowland, M., Albanie, S., Velingker, A., Schmitt, S., Lockhart, E., Hughes, E., Michalewski, H., Sonnerat, N., Hassabis, D., Kohli, P., and Silver, D. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, pp. 1–3, November 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09833-y. URL <https://www.nature.com/articles/s41586-025-09833-y>. Publisher: Nature Publishing Group.
- Hughes, E., Dennis, M., Parker-Holder, J., Behbahani, F., Mavalankar, A., Shi, Y., Schaul, T., and Rocktaschel, T. Open-Endedness is Essential for Artificial Superhuman Intelligence, June 2024. URL <https://arxiv.org/abs/2406.04268>. arXiv:2406.04268 [cs].
- Hutter, F., Kotthoff, L., and Vanschoren, J. (eds.). *Automated Machine Learning - Methods, Systems, Challenges*. Springer, 2019.
- Intology. Zochi technical report. *arXiv*, 2025.
- Jackson, M., Lu, C., Kirsch, L., Lange, R., Whiteson, S., and Foerster, J. Discovering temporally-aware reinforcement learning algorithms. In *Second agent learning in open-endedness workshop*, 2023. URL <https://openreview.net/forum?id=XUohU3mYQ5>.
- Jackson, M. T., Berdica, U., Liesen, J. L., Whiteson, S., and Foerster, J. N. A clean slate for offline reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL <https://openreview.net/forum?id=8P3QNSckMp>.
- Jelassi, S., Brandfonbrener, D., Kakade, S. M., and Malach, E. Repeat after me: transformers are better than state space models at copying. In *Proceedings of the 41st International Conference on Machine Learning*, ICML’24. JMLR.org, 2024.
- Jiang, M., Dennis, M., Parker-Holder, J., Foerster, J. N., Grefenstette, E., and Rocktaschel, T. Replay-guided adversarial environment design. In *Advances in Neural Information Processing Systems*, pp. 1884–1897, 2021a. URL <https://proceedings.neurips.cc/paper/2021/hash/0e915db6326b6fb6a3c56546980a8c93-Abs tract.html>.
- Jiang, M., Grefenstette, E., and Rocktaschel, T. Prioritized Level Replay, June 2021b. URL <http://arxiv.org/abs/2010.03934>. arXiv:2010.03934 [cs].
- Jiang, Z., Schmidt, D., Srikanth, D., Xu, D., Kaplan, I., Jacenko, D., and Wu, Y. AIDE: AI-Driven Exploration in the Space of Code, February 2025. URL <http://arxiv.org/abs/2502.13138>. arXiv:2502.13138 [cs].
- Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. SWE-bench: Can Language Models Resolve Real-World GitHub Issues?, November 2024. URL <http://arxiv.org/abs/2310.06770>. arXiv:2310.06770 [cs].
- Jones, D. R., Schonlau, M., and Welch, W. J. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.
- Jordan, K., Bernstein, J., Rappazzo, B., @fern-bear.bsky.social, Vlado, B., Jiacheng, Y., Cesista, F., Koszarsky, B., and @Grad62304977. modded-nanopt: Speedrunning the nanopt baseline. <https://github.com/KellerJordan/modded-nanopt>, 2024. GitHub repository.
- Joshi, M., Choi, E., Weld, D., and Zettlemoyer, L. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In Barzilay, R. and Kan, M.-Y. (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL <https://aclanthology.org/P17-1147>.
- Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling Laws for Neural Language Models, January 2020. URL <https://arxiv.org/abs/2001.08361>. arXiv:2001.08361 [cs].
- Kazemnejad, A., Aghajohari, M., Portelance, E., Sordoni, A., Reddy, S., Courville, A., and Roux, N. L. VinePPO: Refining Credit Assignment in RL Training of LLMs, June 2025. URL <http://arxiv.org/abs/2410.01679>. arXiv:2410.01679 [cs].
- Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization, January 2017. URL <http://arxiv.org/abs/1412.6980>.
- Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

- doi: 10.1073/pnas.1611835114. URL <https://www.pnas.org/doi/abs/10.1073/pnas.1611835114>.
- Kirsch, L., van Steenkiste, S., and Schmidhuber, J. Improving generalization in meta reinforcement learning using learned objectives. In *International conference on learning representations*, 2020. URL <https://openreview.net/forum?id=S1evHerYPr>.
- Klissarov, M., Henaff, M., Raileanu, R., Sodhani, S., Vincent, P., Zhang, A., Bacon, P.-L., Precup, D., Machado, M. C., and D’Oro, P. Maestromotif: Skill design from artificial intelligence feedback. 2025. URL <https://openreview.net/forum?id=or8mMhmyRV>.
- Kocsis, L. and Szepesvári, C. Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*, ECML’06, pp. 282–293, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 354045375X. doi: 10.1007/11871842\_29. URL [https://doi.org/10.1007/11871842\\_29](https://doi.org/10.1007/11871842_29).
- Komeili, M., Shuster, K., and Weston, J. Internet-Augmented Dialogue Generation, July 2021. URL <https://arxiv.org/abs/2107.07566v1>.
- Krause, J., Stark, M., Deng, J., and Fei-Fei, L. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*, June 2013.
- Krishnan, S., Wang, J., Wu, E., Franklin, M. J., and Goldberg, K. Activeclean: interactive data cleaning for statistical modeling. *Proc. VLDB Endow.*, 9(12):948–959, August 2016. ISSN 2150-8097. doi: 10.14778/2994509.2994514. URL <https://doi.org/10.14778/2994509.2994514>.
- Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.
- Küttler, H., Nardelli, N., Miller, A. H., Raileanu, R., Selvatici, M., Grefenstette, E., and Rocktäschel, T. The NetHack Learning Environment, December 2020. URL <https://arxiv.org/abs/2006.13760>. arXiv:2006.13760 [cs].
- Lan, Q., Mahmood, A. R., Yan, S., and Xu, Z. Learning to Optimize for Reinforcement Learning, June 2024. URL <https://arxiv.org/abs/2302.01470>.
- Lan, X. and Keeling, R. Trends in Atmospheric Carbon Dioxide. NOAA Global Monitoring Laboratory. URL <https://gml.noaa.gov/ccgg/trends/data.html>. Accessed: 2026-01-13.
- Landau, G., Özdogan, M., Elvers, G., Mantegna, F., Somaiya, P., Jayalath, D., Kurth, L., Kwon, T., Shillingford, B., Farquhar, G., et al. The 2025 pnpl competition: Speech detection and phoneme classification in the libri-brain dataset. *arXiv preprint arXiv:2506.10165*, 2025.
- Lange, R. T. <span class="nocase">gymnax</span>: A JAX-based reinforcement learning environment library, 2022. URL <http://github.com/RobertTLange/gymnax>.
- Langford, J. Clever methods of overfitting. Machine Learning (Theory) Blog, February 2005. URL <https://hunch.net/?p=22>. Accessed: 2026-01-18.
- LeCun, Y., Cortes, C., and Burges, C. Mnist handwritten digit database. *ATT Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist>, 2, 2010.
- LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, 521(7553):436–444, May 2015. ISSN 1476-4687. doi: 10.1038/nature14539. URL <https://doi.org/10.1038/nature14539>.
- Leibo, J. Z., Hughes, E., Lanctot, M., and Graepel, T. Autocurricula and the Emergence of Innovation from Social Interaction: A Manifesto for Multi-Agent Intelligence Research, March 2019. URL <https://arxiv.org/abs/1903.00742>. arXiv:1903.00742 [cs].
- Lester, B., Al-Rfou, R., and Constant, N. The power of scale for parameter-efficient prompt tuning. In Moens, M.-F., Huang, X., Specia, L., and Yih, S. W.-t. (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3045–3059, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.243. URL <https://aclanthology.org/2021.emnlp-main.243>.
- Lewkowycz, A., Andreassen, A., Dohan, D., Dyer, E., Michalewski, H., Ramasesh, V., Sloane, A., Anil, C., Schlag, I., Gutman-Solo, T., Wu, Y., Neyshabur, B., Gur-Ari, G., and Misra, V. Solving Quantitative Reasoning Problems with Language Models, July 2022. URL <https://arxiv.org/abs/2206.14858>. arXiv:2206.14858 [cs].
- Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization, June 2018. URL <https://arxiv.org/abs/1603.06560>. arXiv:1603.06560 [cs].
- Li, N., Pan, A., Gopal, A., Yue, S., Berrios, D., Gatti, A., Li, J. D., Dombrowski, A.-K., Goel, S., Phan, L., Mukobi, G., Helm-Burger, N., Lababidi, R., Justen, L., Liu, A. B.,

- Chen, M., Barrass, I., Zhang, O., Zhu, X., Tamirisa, R., Bharathi, B., Khoja, A., Zhao, Z., Herbert-Voss, A., Breuer, C. B., Marks, S., Patel, O., Zou, A., Mazeika, M., Wang, Z., Oswal, P., Liu, W., Hunt, A. A., Tienken-Harder, J., Shih, K. Y., Talley, K., Guan, J., Kaplan, R., Stenecker, I., Campbell, D., Jokubaitis, B., Levinson, A., Wang, J., Qian, W., Karmakar, K. K., Basart, S., Fitz, S., Levine, M., Kumaraguru, P., Tupakula, U., Varadharajan, V., Shoshtaishvili, Y., Ba, J., Esveld, K. M., Wang, A., and Hendrycks, D. The wmdp benchmark: Measuring and reducing malicious use with unlearning, 2024.
- Li, Y., Choi, D., Chung, J., Kushman, N., Schrittweiser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Lago, A. D., Hubert, T., Choy, P., de Masson d'Autume, C., Babuschkin, I., Chen, X., Huang, P.-S., Welbl, J., Gowal, S., Cherepanov, A., Molloy, J., Mankowitz, D. J., Robson, E. S., Kohli, P., de Freitas, N., Kavukcuoglu, K., and Vinyals, O. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158. URL <https://www.science.org/doi/abs/10.1126/science.abq1158>.
- Liang, S., Garg, S., and Moghaddam, R. Z. The SWE-Bench Illusion: When State-of-the-Art LLMs Remember Instead of Reason, December 2025. URL <http://arxiv.org/abs/2506.12286>. arXiv:2506.12286 [cs] version: 4.
- Lindauer, M., van Rijn, J. N., and Kotthoff, L. Open Algorithm Selection Challenge 2017 Setup and Scenarios. 2017.
- Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhkopf, T., Sass, R., and Hutter, F. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022. URL <http://jmlr.org/papers/v23/21-0888.html>.
- Liu, J., Wang, K., Chen, Y., Peng, X., Chen, Z., Zhang, L., and Lou, Y. Large Language Model-Based Agents for Software Engineering: A Survey, December 2025. URL <http://arxiv.org/abs/2409.02977>. arXiv:2409.02977 [cs].
- Löper, L. Boax: A bayesian optimization library for JAX, 2023. URL <https://github.com/Lando-L/boax>.
- Lu, C., Kuba, J., Letcher, A., Metz, L., Schroeder de Witt, C., and Foerster, J. Discovered policy optimisation. *Advances in Neural Information Processing Systems*, 35: 16455–16468, 2022.
- Lu, C., Holt, S., Fanconi, C., Chan, A. J., Foerster, J., van der Schaar, M., and Lange, R. T. Discovering Preference Optimization Algorithms with and for Large Language Models, September 2024a. URL <http://arxiv.org/abs/2406.08414>.
- Lu, C., Lu, C., Lange, R. T., Foerster, J., Clune, J., and Ha, D. The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery, September 2024b. URL <http://arxiv.org/abs/2408.06292>. arXiv:2408.06292 [cs].
- Luong, T., Hwang, D., Nguyen, H. H., Ghiasi, G., Chervonyi, Y., Seo, I., Kim, J., Bingham, G., Lee, J., Mishra, S., Zhai, A., Hu, C. H., Michalewski, H., Kim, J., Ahn, J., Bae, J., Song, X., Trinh, T. H., Le, Q. V., and Jung, J. Towards Robust Mathematical Reasoning, November 2025. URL <http://arxiv.org/abs/2511.01846>. arXiv:2511.01846 [cs].
- Ma, Y. J., Liang, W., Wang, G., Huang, D.-A., Bastani, O., Jayaraman, D., Zhu, Y., Fan, L., and Anandkumar, A. Eureka: Human-Level Reward Design via Coding Large Language Models, April 2024. URL <http://arxiv.org/abs/2310.12931>. arXiv:2310.12931 [cs].
- Maini, P., Feng, Z., Schwarzschild, A., Lipton, Z. C., and Kolter, J. Z. TOFU: A task of fictitious unlearning for LLMs. In *First Conference on Language Modeling*, 2024.
- Matthews, M., Beukman, M., Ellis, B., Samvelyan, M., Jackson, M., Coward, S., and Foerster, J. Craftax: a lightning-fast benchmark for open-ended reinforcement learning. In *International conference on machine learning (ICML)*, 2024.
- Matthews, M., Beukman, M., Lu, C., and Foerster, J. Kinetix: Investigating the Training of General Agents through Open-Ended Physics-Based Control Tasks, March 2025. URL <http://arxiv.org/abs/2410.23208>. arXiv:2410.23208 [cs] version: 2.
- Mesnard, T. and et al. Gemma: Open models based on gemini research and technology. arXiv preprint arXiv:2403.08295, 2024. URL <https://arxiv.org/abs/2403.08295>.
- Meta AI. The llama 4 herd: The beginning of a new era of natively multimodal ai innovation, April 2025. URL <https://ai.meta.com/blog/llama-4-multimodal-intelligence/>.
- METR. Measuring ai ability to complete long tasks. <https://metr.org/blog/2025-03-19-measuring-ai-ability-to-complete-long-tasks/>, 03 2025.
- Metz, L., Maheswaranathan, N., Cheung, B., and Sohl-Dickstein, J. Meta-Learning Update Rules for Unsupervised Representation Learning, February 2019. URL <http://arxiv.org/abs/1804.00222>. arXiv:1804.00222 [cs, stat].

- Metz, L., Freeman, C. D., Harrison, J., Maheswaranathan, N., and Sohl-Dickstein, J. Practical tradeoffs between memory, compute, and performance in learned optimizers. In *Conference on lifelong learning agents*, pp. 142–164. PMLR, 2022a. URL [http://github.com/google/learned\\_optimization](http://github.com/google/learned_optimization).
- Metz, L., Harrison, J., Freeman, C. D., Merchant, A., Beyer, L., Bradbury, J., Agrawal, N., Poole, B., Mordatch, I., Roberts, A., and Sohl-Dickstein, J. VeLO: Training Versatile Learned Optimizers by Scaling Up, November 2022b. URL <http://arxiv.org/abs/2211.09760>. arXiv:2211.09760 [cs, math, stat].
- Mistral AI. Introducing: Devstral 2 and Mistral Vibe CLI. <https://mistral.ai/news/devstral-2-vibe-cli>, 2025. [Accessed 11-01-2026].
- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning, 2013. URL <https://arxiv.org/abs/1312.5602>.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL <https://doi.org/10.1038/nature14236>.
- Nair, V. and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML’10, pp. 807–814, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.
- Nakano, R., Hilton, J., Balaji, S., Wu, J., Ouyang, L., Kim, C., Hesse, C., Jain, S., Kosaraju, V., Saunders, W., Jiang, X., Cobbe, K., Eloundou, T., Krueger, G., Button, K., Knight, M., Chess, B., and Schulman, J. WebGPT: Browser-assisted question-answering with human feedback, December 2021. URL <https://arxiv.org/abs/2112.09332v3>.
- Nathani, D., Madaan, L., Roberts, N., Bashlykov, N., Menon, A., Moens, V., Budhiraja, A., Magka, D., Vorotilov, V., Chaurasia, G., Hupkes, D., Cabral, R. S., Shavrina, T., Foerster, J., Bachrach, Y., Wang, W. Y., and Raileanu, R. Mlgym: A new framework and benchmark for advancing ai research agents, 2025. URL <https://arxiv.org/abs/2502.14499>.
- Neutatz, F., Chen, B., Alkhatib, Y., Ye, J., and Abedjan, Z. Data Cleaning and AutoML: Would an Optimizer Choose to Clean? *Datenbank-Spektrum*, 22(2):121–130, July 2022. ISSN 1610-1995. doi: 10.1007/s13222-022-00413-2. URL <https://doi.org/10.1007/s13222-022-00413-2>.
- Nikulin, A., Kurenkov, V., Zisman, I., Agarkov, A., Sinii, V., and Kolesnikov, S. XLand-MiniGrid: Scalable Meta-Reinforcement Learning Environments in JAX, November 2024. URL <http://arxiv.org/abs/2312.12044>. arXiv:2312.12044 [cs].
- Nilsback, M.-E. and Zisserman, A. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.
- Novikov, A., Vũ, N., Eisenberger, M., Dupont, E., Huang, P.-S., Wagner, A. Z., Shirobokov, S., Kozlovskii, B., Ruiz, F. J. R., Mehrabian, A., Kumar, M. P., See, A., Chaudhuri, S., Holland, G., Davies, A., Nowozin, S., Kohli, P., and Balog, M. AlphaEvolve: A coding agent for scientific and algorithmic discovery, June 2025. URL <http://arxiv.org/abs/2506.13131>. arXiv:2506.13131 [cs].
- Oh, J., Hessel, M., Czarnecki, W. M., Xu, Z., van Hasselt, H. P., Singh, S., and Silver, D. Discovering reinforcement learning algorithms. *Advances in Neural Information Processing Systems*, 33:1060–1070, 2020.
- Oh, J., Farquhar, G., Kemaev, I., Calian, D. A., Hessel, M., Zintgraf, L., Singh, S., van Hasselt, H., and Silver, D. Discovering state-of-the-art reinforcement learning algorithms. *Nature*, pp. 1–8, October 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09761-x. URL <https://www.nature.com/articles/s41586-025-09761-x>. Publisher: Nature Publishing Group.
- OpenAI, Feb 2025a. URL <https://openai.com/index/introducing-deep-research/>. [Accessed 12-01-2026].
- OpenAI. Introducing codex, 2025b. URL <https://openai.com/index/introducing-codex/>.
- OpenAI. Introducing gpt-5, August 2025c. URL <https://openai.com/index/introducing-gpt-5/>.
- OpenAI, Agarwal, S., Ahmad, L., Ai, J., Altman, S., Applebaum, A., Arbus, E., Arora, R. K., Bai, Y., Baker, B., Bao, H., Barak, B., Bennett, A., Bertao, T., Brett, N., Brevdo, E., Brockman, G., Bubeck, S., Chang, C., Chen, K., Chen, M., Cheung, E., Clark, A., Cook, D., Dukhan, M., Dvorak, C., Fives, K., Fomenko, V., Garipov, T., Georgiev, K., Glaese, M., Gogineni, T., Goucher, A., Gross, L., Guzman, K. G., Hallman, J., Hehir, J., Heidecke, J., Helayar, A., Hu, H., Huet, R., Huh, J., Jain, S., Johnson, Z., Koch, C., Kofman, I., Kundel, D., Kwon, J., Kyrylov, V., Le, E. Y., Leclerc, G., Lennon, J. P., Lessans, S., Lezcano-Casado, M., Li, Y., Li, Z., Lin, J., Liss, J., Lily,

- Liu, Liu, J., Lu, K., Lu, C., Martinovic, Z., McCallum, L., McGrath, J., McKinney, S., McLaughlin, A., Mei, S., Mostovoy, S., Mu, T., Myles, G., Neitz, A., Nichol, A., Pachocki, J., Paino, A., Palmie, D., Pantuliano, A., Parascandolo, G., Park, J., Pathak, L., Paz, C., Peran, L., Pimenov, D., Pokrass, M., Proehl, E., Qiu, H., Raila, G., Raso, F., Ren, H., Richardson, K., Robinson, D., Rotsted, B., Salman, H., Sanjeev, S., Schwarzer, M., Sculley, D., Sikchi, H., Simon, K., Singhal, K., Song, Y., Stuckey, D., Sun, Z., Tillet, P., Toizer, S., Tsimpourlas, F., Vyas, N., Wallace, E., Wang, X., Wang, M., Watkins, O., Weil, K., Wendling, A., Whinnery, K., Whitney, C., Wong, H., Yang, L., Yang, Y., Yasunaga, M., Ying, K., Zaremba, W., Zhan, W., Zhang, C., Zhang, B., Zhang, E., and Zhao, S. gpt-oss-120b & gpt-oss-20b Model Card, August 2025. URL <http://arxiv.org/abs/2508.10925>. arXiv:2508.10925 [cs].
- Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., and Lowe, R. Training language models to follow instructions with human feedback, March 2022. URL <http://arxiv.org/abs/2203.02155>. arXiv:2203.02155 [cs].
- Özdogan, M., Landau, G., Elvers, G., Jayalath, D., Somaiya, P., Mantegna, F., Woolrich, M., and Jones, O. P. Libribrain: Over 50 hours of within-subject meg to improve speech decoding methods at scale. *arXiv preprint arXiv:2506.02098*, 2025.
- Pagliari, D., Cupiał, B., Coward, S., Piterbarg, U., Wolczyk, M., Khan, A., Pignatelli, E., Kuciński, Ł., Pinto, L., Fergus, R., Foerster, J. N., Parker-Holder, J., and Rocktäschel, T. BALROG: Benchmarking agentic LLM and VLM reasoning on games. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=fp6t3F669F>.
- Parker-Holder, J., Nguyen, V., and Roberts, S. Provably Efficient Online Hyperparameter Optimization with Population-Based Bandits, June 2021. URL <http://arxiv.org/abs/2002.02518>. arXiv:2002.02518 [cs].
- Parker-Holder, J., Jiang, M., Dennis, M., Samvelyan, M., Foerster, J., Grefenstette, E., and Rocktäschel, T. Evolving Curricula with Regret-Based Environment Design. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 17473–17498. PMLR, July 2022a. URL <https://proceedings.mlr.press/v162/parker-holder22a.html>.
- Parker-Holder, J., Rajan, R., Song, X., Biedenkapp, A., Miao, Y., Eimer, T., Zhang, B., Nguyen, V., Calandra, R., Faust, A., Hutter, F., and Lindauer, M. Automated Reinforcement Learning (AutoRL): A Survey and Open Problems. *Journal of Artificial Intelligence Research*, 74: 517–568, June 2022b. ISSN 1076-9757. doi: 10.1613/jair.1.13596. URL <http://arxiv.org/abs/2201.03916>. arXiv:2201.03916 [cs].
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- Penedo, G., Kydlíček, H., Allal, L. B., Lozhkov, A., Mitchell, M., Raffel, C., Werra, L. V., and Wolf, T. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. URL <https://arxiv.org/abs/2406.17557>.
- Phan, L. et al. Humanity’s last exam, 2025. URL <https://arxiv.org/abs/2501.14249>.
- Pichai, S., Hassabis, D., and Kavukcuoglu, K. A new era of intelligence with gemini 3, November 2025. URL <https://blog.google/products-and-platforms/products/gemini/gemini-3/>.
- Qiu, X., Gan, Y., Hayes, C. F., Liang, Q., Meyerson, E., Hodjat, B., and Miikkulainen, R. Evolution Strategies at Scale: LLM Fine-Tuning Beyond Reinforcement Learning, September 2025. URL <http://arxiv.org/abs/2509.24372>. arXiv:2509.24372 [cs].
- Qwen. Qwen : A family of large language models. Technical report / model release, 2024. Available at <https://github.com/QwenLM/Qwen3>.
- Ramachandran, P., Zoph, B., and Le, Q. V. Searching for Activation Functions, October 2017. URL <http://arxiv.org/abs/1710.05941>. arXiv:1710.05941 [cs].
- Real, E., Liang, C., So, D. R., and Le, Q. V. Automl-zero: Evolving machine learning algorithms from scratch. *arXiv preprint arXiv:2003.03384*, 2020.
- Rodrigo, M., Cuevas, C., and García, N. Comprehensive comparison between vision transformers and convolutional neural networks for face recognition tasks. *Scientific Reports*, 14(1):21392, September 2024. ISSN 2045-2322. doi: 10.1038/s41598-024-72254-w. URL <https://doi.org/10.1038/s41598-024-72254-w>.

- Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J. R., Ellenberg, J. S., Wang, P., Fawzi, O., Kohli, P., and Fawzi, A. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, January 2024. ISSN 1476-4687. doi: 10.1038/s41586-023-06924-6. URL <https://www.nature.com/articles/s41586-023-06924-6>.
- Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Sauvestre, R., Remez, T., Rapin, J., Kozhevnikov, A., Evtimov, I., Bitton, J., Bhatt, M., Ferrer, C. C., Grattafiori, A., Xiong, W., Défossez, A., Copet, J., Azhar, F., Touvron, H., Martin, L., Usunier, N., Scialom, T., and Synnaeve, G. Code Llama: Open Foundation Models for Code, January 2024. URL <http://arxiv.org/abs/2308.12950>. arXiv:2308.12950 [cs].
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3): 211–252, 2015.
- Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive Neural Networks, October 2022. URL <http://arxiv.org/abs/1606.04671>. arXiv:1606.04671 [cs].
- Rutherford, A., Beukman, M., Willi, T., Lacerda, B., Hawes, N., and Foerster, J. No regrets: Investigating and improving regret approximations for curriculum discovery. *Advances in Neural Information Processing Systems*, 37: 16071–16101, 2024.
- Samvelyan, M., Kirk, R., Kurin, V., Parker-Holder, J., Jiang, M., Hambro, E., Petroni, F., Kuttler, H., Grefenstette, E., and Rocktäschel, T. Minihack the planet: A sandbox for open-ended reinforcement learning research. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. URL <https://openreview.net/forum?id=skFwlyefKwJ>.
- Samvelyan, M., Khan, A., Dennis, M., Jiang, M., Parker-Holder, J., Foerster, J. N., Raileanu, R., and Rocktäschel, T. MAESTRO: open-ended environment design for multi-agent reinforcement learning. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL <https://openreview.net/pdf?id=sKWL RDzPfd7>.
- Sarkar, B., Fellows, M., Duque, J. A., Letcher, A., Villares, A. L., Sims, A., Cope, D., Liesen, J., Seier, L., Wolf, T., Berdica, U., Goldie, A. D., Courville, A., Sevgnani, K., Whiteson, S., and Foerster, J. N. Evolution Strategies at the Hyperscale, November 2025. URL <http://arxiv.org/abs/2511.16652>. arXiv:2511.16652 [cs].
- Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., and Scialom, T. Toolformer: Language Models Can Teach Themselves to Use Tools, February 2023. URL <http://arxiv.org/abs/2302.04761>. arXiv:2302.04761 [cs].
- Schmidhuber, J. Evolutionary principles in self-referential learning, or on learning how to learn: The meta-meta-hook. 1987.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal Policy Optimization Algorithms, August 2017. URL <http://arxiv.org/abs/1707.06347>.
- Shaker, N., Togelius, J., and {J. Nelson}, M. *Procedural Content Generation in Games*. Springer, Germany, 2016.
- Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y. K., Wu, Y., and Guo, D. DeepSeek-Math: Pushing the Limits of Mathematical Reasoning in Open Language Models, April 2024. URL <http://arxiv.org/abs/2402.03300>. arXiv:2402.03300 [cs].
- Shi, W., Lee, J., Huang, Y., Malladi, S., Zhao, J., Holtzman, A., Liu, D., Zettlemoyer, L., Smith, N. A., and Zhang, C. MUSE: Machine unlearning six-way evaluation for language models. 2024. URL <https://arxiv.org/abs/2407.06460>.
- Shneiderman, B. *Human-Centered AI*. Oxford University Press, January 2022. ISBN 978-0-19-284529-0. doi: 10.1093/oso/9780192845290.001.0001. URL <https://doi.org/10.1093/oso/9780192845290.001.0001>. eprint: [https://academic.oup.com/book/41126/book-pdf/50987951/9780192659996\\_web.pdf](https://academic.oup.com/book/41126/book-pdf/50987951/9780192659996_web.pdf).
- Si, C., Yang, D., and Hashimoto, T. Can LLMs Generate Novel Research Ideas? A Large-Scale Human Study with 100+ NLP Researchers, September 2024. URL <http://arxiv.org/abs/2409.04109>. arXiv:2409.04109.
- Si, C., Yang, Z., Choi, Y., Candès, E., Yang, D., and Hashimoto, T. Towards Execution-Grounded Automated AI Research, January 2026. URL <http://arxiv.org/abs/2601.14525>. arXiv:2601.14525 [cs].
- Stanley, K. O. Why open-endedness matters. *Artificial life*, 25(3):232–235, 2019.

- Stanley, K. O. and Miikkulainen, R. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127, June 2002. ISSN 1063-6560. doi: 10.1162/106365602320169811. URL <https://doi.org/10.1162/106365602320169811>.
- Stanley, K. O., D’Ambrosio, D. B., and Gauci, J. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, 2009. doi: 10.1162/artl.2009.15.2.15202.
- Stooke, A., Mahajan, A., Barros, C., Deck, C., Bauer, J., Sygnowski, J., Trebacz, M., Jaderberg, M., Mathieu, M., McAleese, N., Bradley-Schmieg, N., Wong, N., Porcel, N., Raileanu, R., Hughes-Fitt, S., Dalibard, V., and Czarnecki, W. M. Open-Ended Learning Leads to Generally Capable Agents, July 2021. URL <http://arxiv.org/abs/2107.12808>. arXiv:2107.12808 [cs], Open Ended Learning Team.
- Strubell, E., Ganesh, A., and McCallum, A. Energy and Policy Considerations for Deep Learning in NLP, June 2019. URL <http://arxiv.org/abs/1906.02243>. arXiv:1906.02243 [cs].
- Surjanovic, S. and Bingham, D. Virtual library of simulation experiments: Test functions and datasets. Retrieved January 16, 2026, from <http://www.sfu.ca/~ssurjano>.
- Sutton, R. S. and Barto, A. *Reinforcement learning: an introduction*. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts London, England, second edition edition, 2020. ISBN 978-0-262-03924-6.
- Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, 2016. doi: 10.1109/CVPR.2016.308.
- Takahashi, S., Sakaguchi, Y., Kouno, N., Takasawa, K., Ishizu, K., Akagi, Y., Aoyama, R., Teraya, N., Bo-latkhan, A., Shinkai, N., Machino, H., Kobayashi, K., Asada, K., Komatsu, M., Kaneko, S., Sugiyama, M., and Hamamoto, R. Comparison of Vision Transformers and Convolutional Neural Networks in Medical Image Analysis: A Systematic Review. *Journal of Medical Systems*, 48(1):84, September 2024. ISSN 1573-689X. doi: 10.1007/s10916-024-02105-8. URL <https://doi.org/10.1007/s10916-024-02105-8>.
- Taylor, R., Kardas, M., Cucurull, G., Scialom, T., Hartshorn, A., Saravia, E., Poulton, A., Kerkez, V., and Stojnic, R. Galactica: A Large Language Model for Science, November 2022. URL <http://arxiv.org/abs/2211.09085>. arXiv:2211.09085 [cs].
- Thakkar, N., Yuksekgonul, M., Silberg, J., Garg, A., Peng, N., Sha, F., Yu, R., Vondrick, C., and Zou, J. Can LLM feedback enhance review quality? A randomized study of 20K reviews at ICLR 2025, April 2025. URL <http://arxiv.org/abs/2504.09737>. arXiv:2504.09737 [cs].
- Thoppilan, R., De Freitas, D., Hall, J., Shazeer, N., Kulshreshtha, A., Cheng, H.-T., Jin, A., Bos, T., Baker, L., Du, Y., Li, Y., Lee, H., Zheng, H. S., Ghafouri, A., Menegali, M., Huang, Y., Krikun, M., Lepikhin, D., Qin, J., Chen, D., Xu, Y., Chen, Z., Roberts, A., Bosma, M., Zhao, V., Zhou, Y., Chang, C.-C., Krivokon, I., Rusch, W., Pickett, M., Srinivasan, P., Man, L., Meier-Hellstern, K., Morris, M. R., Doshi, T., Santos, R. D., Duke, T., Soraker, J., Zevenbergen, B., Prabhakaran, V., Diaz, M., Hutchinson, B., Olson, K., Molina, A., Hoffman-John, E., Lee, J., Aroyo, L., Rajakumar, R., Butryna, A., Lamm, M., Kuzmina, V., Fenton, J., Cohen, A., Bernstein, R., Kurzweil, R., Aguera-Arcas, B., Cui, C., Croak, M., Chi, E., and Le, Q. LaMDA: Language Models for Dialog Applications, January 2022. URL <https://arxiv.org/abs/2201.08239v3>.
- Todorov, E., Erez, T., and Tassa, Y. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.
- Togelius, J., Champandard, A. J., Lanzi, P. L., Mateas, M., Paiva, A., Preuss, M., and Stanley, K. O. Procedural Content Generation: Goals, Challenges and Actionable Steps. In Lucas, S. M., Mateas, M., Preuss, M., Spronck, P., and Togelius, J. (eds.), *Dagstuhl Follow-Ups*, pp. 15 pages, 414306 bytes. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. ISBN 978-3-939897-62-0. doi: 10.4230/DFU.VOL6.12191.61. URL <https://drops.dagstuhl.de/entities/document/10.4230/DFU.Vol6.12191.61>.
- Toledo, E., Hambardzumyan, K., Josifoski, M., Hazra, R., Baldwin, N., Audran-Reiss, A., Kuchnik, M., Magka, D., Jiang, M., Lupidi, A. M., Lupu, A., Raileanu, R., Niu, K., Shavrina, T., Gagnon-Audet, J.-C., Shvartsman, M., Sodhani, S., Miller, A. H., Charnalia, A., Dunfield, D., Wu, C.-J., Stenetorp, P., Cancedda, N., Foerster, J. N., and Bachrach, Y. AI Research Agents for Machine Learning: Search, Exploration, and Generalization in MLE-bench, November 2025. URL <http://arxiv.org/abs/2507.02554>. arXiv:2507.02554 [cs].
- Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, É., and Lampe, G. Llama: Open and efficient foundation language

- models. arXiv preprint arXiv:2302.13971, 2023. URL <https://arxiv.org/abs/2302.13971>.
- Trinh, T. H., Wu, Y., Le, Q. V., He, H., and Luong, T. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, January 2024. ISSN 1476-4687. doi: 10.1038/s41586-023-06747-5. URL <https://doi.org/10.1038/s41586-023-06747-5>.
- Tunstall, L., Beeching, E., Lambert, N., Rajani, N., Rasul, K., Belkada, Y., Huang, S., von Werra, L., Fourrier, C., Habib, N., Sarrazin, N., Sanseviero, O., and Rush, A. M. Zephyr: Direct distillation of lm alignment. arXiv preprint arXiv:2310.16944, 2023. URL <https://arxiv.org/abs/2310.16944>.
- Turpin, M., Michael, J., Perez, E., and Bowman, S. R. Language Models Don’t Always Say What They Think: Unfaithful Explanations in Chain-of-Thought Prompting, December 2023. URL <http://arxiv.org/abs/2305.04388>. arXiv:2305.04388 [cs].
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is All you Need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL [https://proceedings.neurips.cc/paper\\_files/paper/2017/file/3f5ee243547dee91fdb053c1c4a845aa-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fdb053c1c4a845aa-Paper.pdf).
- Wang, F., Cheng, J., Liu, W., and Liu, H. Additive margin softmax for face verification. *IEEE Signal Processing Letters*, 25(7):926–930, 2018. doi: 10.1109/LSP.2018.2822810.
- Wang, J. X., King, M., Porcel, N., Kurth-Nelson, Z., Zhu, T., Deck, C., Choy, P., Cassin, M., Reynolds, M., Song, F., Buttimore, G., Reichert, D. P., Rabinowitz, N., Matthey, L., Hassabis, D., Lerchner, A., and Botvinick, M. Alchemy: A benchmark and analysis toolkit for meta-reinforcement learning agents, October 2021. URL <http://arxiv.org/abs/2102.02926>. arXiv:2102.02926 [cs].
- Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., and Wen, J.-R. A Survey on Large Language Model based Autonomous Agents. *Frontiers of Computer Science*, 18(6):186345, December 2024a. ISSN 2095-2228, 2095-2236. doi: 10.1007/s11704-024-40231-1. URL <http://arxiv.org/abs/2308.11432>. arXiv:2308.11432 [cs].
- Wang, L., Zhang, X., Su, H., and Zhu, J. A Comprehensive Survey of Continual Learning: Theory, Method and Application. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 46(08):5362–5383, August 2024b. ISSN 1939-3539. doi: 10.1109/TPAMI.2024.3367329. URL <https://doi.ieeecomputersociety.org/10.1109/TPAMI.2024.3367329>.
- Wang, W., Piękos, P., Nanbo, L., Laakom, F., Chen, Y., Ostaszewski, M., Zhuge, M., and Schmidhuber, J. Huxley-Gödel Machine: Human-Level Coding Agent Development by an Approximation of the Optimal Self-Improving Machine, October 2025a. URL <http://arxiv.org/abs/2510.21614>. arXiv:2510.21614 [cs].
- Wang, Y., Ji, P., Yang, C., Li, K., Hu, M., Li, J., and Sartoretti, G. MCTS-Judge: Test-Time Scaling in LLM-as-a-Judge for Code Correctness Evaluation, February 2025b. URL <http://arxiv.org/abs/2502.12468>. arXiv:2502.12468 [cs].
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models, January 2023. URL <http://arxiv.org/abs/2201.11903>. arXiv:2201.11903 [cs].
- Weidinger, L., Mellor, J., Rauh, M., Griffin, C., Uesato, J., Huang, P.-S., Cheng, M., Glaese, M., Balle, B., Kasirzadeh, A., Kenton, Z., Brown, S., Hawkins, W., Stepleton, T., Biles, C., Birhane, A., Haas, J., Rimell, L., Hendricks, L. A., Isaac, W., Legassick, S., Irving, G., and Gabriel, I. Ethical and social risks of harm from Language Models, December 2021. URL <http://arxiv.org/abs/2112.04359>. arXiv:2112.04359 [cs].
- Wen, X., Liu, Z., Zheng, S., Ye, S., Wu, Z., Wang, Y., Xu, Z., Liang, X., Li, J., Miao, Z., Bian, J., and Yang, M. Reinforcement Learning with Verifiable Rewards Implicitly Incentivizes Correct Reasoning in Base LLMs, October 2025. URL <http://arxiv.org/abs/2506.14245>. arXiv:2506.14245 [cs].
- Weston, J. and Foerster, J. AI & Human Co-Improvement for Safer Co-Superintelligence, December 2025. URL <http://arxiv.org/abs/2512.05356>. arXiv:2512.05356 [cs].
- White, C., Safari, M., Sukthanker, R., Ru, B., Elskens, T., Zela, A., Dey, D., and Hutter, F. Neural Architecture Search: Insights from 1000 Papers, January 2023. URL <http://arxiv.org/abs/2301.08727>. arXiv:2301.08727 [cs].
- Whiteson, S., Tanner, B., Taylor, M. E., and Stone, P. Protecting against evaluation overfitting in empirical reinforcement learning. In *IEEE Symposium on Adaptive*

- Dynamic Programming and Reinforcement Learning (AD-PRL)*, April 2011. URL <http://www.cs.utexas.edu/users/ai-lab?ADPRL11-shimon>.
- Wightman, R. Pytorch image models. <https://github.com/rwightman/pytorch-image-models>, 2019.
- Wijk, H., Lin, T., Becker, J., Jawhar, S., Parikh, N., Broadley, T., Chan, L., Chen, M., Clymer, J., Dhyani, J., Ericheva, E., Garcia, K., Goodrich, B., Jurkovic, N., Karnofsky, H., Kinniment, M., Lajko, A., Nix, S., Sato, L., Saunders, W., Taran, M., West, B., and Barnes, E. RE-Bench: Evaluating frontier AI R&D capabilities of language model agents against human experts, May 2025. URL <https://arxiv.org/abs/2411.15114>. arXiv:2411.15114 [cs].
- Wu, J., Zhang, Q., and Xu, G. Tiny imagenet challenge. *Technical report*, 2017.
- Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.
- Yamada, Y., Lange, R. T., Lu, C., Hu, S., Lu, C., Foerster, J., Clune, J., and Ha, D. The AI Scientist-v2: Workshop-Level Automated Scientific Discovery via Agentic Tree Search, April 2025. URL <http://arxiv.org/abs/2504.08066>. arXiv:2504.08066 [cs].
- Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K. R., and Press, O. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL <https://arxiv.org/abs/2405.15793>.
- Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. ReAct: Synergizing Reasoning and Acting in Language Models, March 2023. URL <https://arxiv.org/abs/2210.03629>. arXiv:2210.03629 [cs].
- Yao, Y., Xu, X., and YangLiu. Large language model unlearning. In Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 105425–105475. Curran Associates, Inc., 2024. doi: 10.5220/079017-3346. URL [https://proceedings.neurips.cc/paper\\_files/paper/2024/file/be52acf6bccf4a8c0a90fe2f5cfcead3-Paper-Conference.pdf](https://proceedings.neurips.cc/paper_files/paper/2024/file/be52acf6bccf4a8c0a90fe2f5cfcead3-Paper-Conference.pdf).
- Young, K. and Tian, T. MinAtar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*, 2019.
- Yu, Z., Feng, K., Zhao, Y., He, S., Zhang, X.-P., and Cohan, A. AlphaResearch: Accelerating New Algorithm Discovery with Language Models, November 2025. URL <https://arxiv.org/abs/2511.08522>. arXiv:2511.08522 [cs].
- Zeng, Z., Yu, J., Gao, T., Meng, Y., Goyal, T., and Chen, D. Evaluating large language models at evaluating instruction following. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=tr0KidwPLc>.
- Zhang, A., Song, S., Wang, J., and Yu, P. S. Time series data cleaning: from anomaly detection to anomaly repairing. *Proc. VLDB Endow.*, 10(10):1046–1057, June 2017. ISSN 2150-8097. doi: 10.14778/3115404.3115410. URL <https://doi.org/10.14778/3115404.3115410>.
- Zhang, J., Hu, S., Lu, C., Lange, R., and Clune, J. Darwin Godel Machine: Open-Ended Evolution of Self-Improving Agents, September 2025. URL <https://arxiv.org/abs/2505.22954>. arXiv:2505.22954 [cs].
- Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., and Stoica, I. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS ’23, Red Hook, NY, USA, 2023. Curran Associates Inc.
- Zheng, W., Chen, T., Hu, T.-K., and Wang, Z. Symbolic learning to optimize: Towards interpretability and scalability. In *International Conference on Learning Representations*, 2022. URL <https://openreview.net/forum?id=ef0nInZHKIC>.
- Zoph, B. and Le, Q. V. Neural Architecture Search with Reinforcement Learning, February 2017. URL <https://arxiv.org/abs/1611.01578>. arXiv:1611.01578 [cs].
- Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning Transferable Architectures for Scalable Image Recognition, April 2018. URL <https://arxiv.org/abs/1707.07012>. arXiv:1707.07012 [cs].

## Appendix

Our appendix is structured as follows:

- Appendix A provides an overview of each task domain included in DiscoGen. We provide a high-level overview of the goal of each domain, its implementation details, and what editable modules and datasets it supports.
- Appendix B extends the abridged related work from the main text (Section 2) to include discussion of the wider field.
- Appendix C provides hyperparameters and experimental details for our paper. It also includes a description of our Elo calculation and discussion of the experimental compute and cost for the paper.
- Appendix D introduces implementation details of DiscoGen. We discuss how to derive the expression for its task count, and how DiscoGen creates new tasks when sampled from.
- Appendix E explores how the performance of agents changes in all possible module combinations in On-Policy Reinforcement Learning in an attempt to understand the diversity over modules for the same task domain.
- Appendix F provides a breakdown of what meta-train and meta-test datasets are for each task in DiscoBench.
- Appendix G examines the redundancy of tasks in DiscoBench, using average-rank correlation, to demonstrate the semantic difference between different tasks.
- Appendix H expands the prompt optimisation results from Section 7 and provides further experimental details and analysis of the system.
- Appendix I introduces two example discovered algorithms, demonstrating that the most performant meta-train/meta-test runs are making novel discoveries rather than just rehashing baselines.
- Appendix J reports and discusses success@3 metrics (i.e, the rate of *at least* one of the three seeds producing a successful solution).
- Appendix K provides per-task results for all experiments in the paper.
- Appendix L includes all prompts used in this paper (excluding the procedurally generated task descriptions). This includes prompts developed by the prompt optimisation loop.
- Appendix M provides all text used for procedurally generating task descriptions.

## A. Task Domain Overview

In this section, we provide a brief overview of the implementations of each task domain included in DiscoGen, as well as all references covering their original implementations and the origin of their datasets.

### A.1. Bayesian Optimisation

**Task Domain** Bayesian Optimisation (Jones et al., 1998; Garnett, 2023) addresses the problem of optimising an expensive, black-box objective function under a limited evaluation budget. A probabilistic surrogate model is fit to observed function evaluations, and an acquisition function, balancing exploration of uncertain regions with exploitation of promising candidates, is used to select the next evaluation point. The objective is to identify the global minimum or maximum of the function within a fixed number of queries.

**Implementation** Our Bayesian Optimisation implementation is based on Boax (Löper, 2023).

**Editable modules** We include six editable modules for Bayesian Optimisation: the surrogate model; the optimiser used to fit the surrogate model; the acquisition function; the optimiser used to maximise the acquisition function; the initial domain sampling strategy; and the query selection policy.

**Datasets** We include 11 synthetic optimisation functions that are standard in Bayesian Optimisation literature (Surjanovic & Bingham): Ackley 1D, Ackley 2D, Branin 2D, Bukin 2D, Cosine 8D, Drop-Wave 2D, Egg-Holder 2D, Griewank 5D, Hartmann 6D, Holder-Table 2D and Levy 6D.

### A.2. Brain Speech Detection

**Task Domain** Brain Speech Detection is a neural signal processing task in which a model predicts the presence of speech from non-invasive or invasive brain recordings. The goal is to learn a two-class classifier conditioned on neural activity (e.g., MEG signals).

**Implementation** The code for Brain Speech Detection is adapted from the official LibriBrain competition (Landau et al., 2025) codebase, which provides a standardised pipeline for neural signal preprocessing, model training, and evaluation.

**Editable modules** We include three editable modules in Brain Speech Detection: the loss function; the optimiser; and the network architecture.

**Datasets** We split the original LibriBrain dataset (Özdogan et al., 2025) into seven parts, constituting seven datasets, each of which contains MEG data and labels collected during the process of the same participant listening to one chapter of naturalistic spoken English, covering nearly the Sherlock Holmes canon.

### A.3. Computer Vision Classification

**Task Domain** Computer Image Classification is a supervised learning task in which a model assigns a discrete semantic label to an input image. The objective is to learn robust visual representations that generalise across variations in appearance, scale, illumination, and data distribution. This task is a foundational benchmark in computer vision and is widely used to evaluate model architectures, optimisation strategies, and robustness to dataset shift. In DiscoGen, the Computer Image Classification task spans standard, corrupted, long-tailed, and fine-grained classification settings.

**Implementation** The code for Computer Image Classification is adapted from the MLGym (Nathani et al., 2025) benchmarking infrastructure. The implementation provides a unified training and evaluation pipeline for image classification models, including dataset loading via HuggingFace Datasets, model initialisation, optimisation, and metric computation.

**Editable modules** We include four editable modules in Computer Image Classification: the loss function; the optimiser; the network architecture; and the image preprocessing.

**Datasets** We support nine widely used image classification datasets, covering a range of difficulty levels and distributional properties. These include MNIST (LeCun et al., 2010) and Fashion-MNIST (Xiao et al., 2017) for grayscale digit and apparel classification; CIFAR-10 and CIFAR-100 (Krizhevsky et al., 2009) for small-scale natural image classification; Tiny ImageNet (Russakovsky et al., 2015; Wu et al., 2017) for large-class-count evaluation; CIFAR-10-C (Hendrycks & Dietterich, 2019) for corruption robustness; CIFAR-10-LT (Krizhevsky et al., 2009) for long-tailed class imbalance; Oxford Flowers-102 (Nilsback & Zisserman, 2008) and Stanford Cars (Krause et al., 2013) for fine-grained object classification. All datasets are accessed through HuggingFace Datasets and follow standardised train, validation, and test splits.

#### A.4. Continual Learning

**Task Domain** Continual learning is a broadly defined field in which a model must learn from non-stationary data sources without forgetting its previous learnings (Wang et al., 2024b). Nonstationarity can arise through a number of means; in our tasks, which are focused on image classification under nonstationarity, these include randomly permuting the labels attached to images or randomly subsampling classes shown throughout training.

**Implementation** Our default implementation is based on elastic weight consolidation (Kirkpatrick et al., 2017).

**Editable modules** We support five different modules in continual learning: the optimisation algorithm; the regulariser for mitigating catastrophic forgetting; the replay buffer for storing past experience; the sampler for mixing replay data with new data; and the learning rate scheduler.

**Additional Backends** Our default backend uses a ResNet-18 for its base network (He et al., 2015). However, we also offer backends support vision transformers (Dosovitskiy et al., 2021; Wightman, 2019) and parameter isolation models (Rusu et al., 2022), a common architecture for preventing catastrophic forgetting (Kirkpatrick et al., 2017) in continual learning.

**Datasets** We support three datasets for continual learning: PermutedMNIST (LeCun et al., 2010), SplitCIFAR100 (Krizhevsky et al., 2009) and TinyImageNetSplit (Wu et al., 2017).

#### A.5. Greenhouse Gas Prediction

**Task Domain** Forecasting the concentration of greenhouse gases in the atmosphere is an important tool for predicting and mitigating the effects of climate change.

**Implementation** Our base implementation is based on a standard scikit-learn (Pedregosa et al., 2011) training loop.

**Editable modules** We support two modules in Greenhouse Gas Prediction: the model architecture and the way the data is processed before modelling.

**Datasets** We support four datasets from Lan & Keeling that are used for predicting concentrations of CO<sub>2</sub>, CH<sub>4</sub>, N<sub>2</sub>O and SF<sub>6</sub> in the atmosphere. Each dataset is split into a training dataset (pre-2014) and a validation dataset (2015-2025).

#### A.6. Language Modelling

**Task Domain** Language Modelling is the task of learning the underlying distribution of text data via next-token prediction over vast bodies of text, mostly scraped from the internet. We evaluate the quality of a trained language model by computing the exponential of the average negative log-likelihood across next-token predictions on a validation set, also called perplexity.

**Implementation** Our default implementation builds on a modified version of the language modeling task from ML-Gym (Nathani et al., 2025), which itself is based on the modded-nanopt repository (Jordan et al., 2024).

**Editable modules** We support three editable modules: the network architecture, the loss function, and the optimiser.

**Datasets** We support the following four datasets: LMFineWeb 10B (Penedo et al., 2024), TinyStories (Eldan & Li, 2023), OPC-FineWeb Math and OPC-FineWeb Code (Huang et al., 2025).

## A.7. Model Unlearning

**Task Domain** Model Unlearning, also called Machine Unlearning (Cao & Yang, 2015) is the task of modifying (e.g. fine-tuning) a model to “forget” targeted information—such as sensitive personal data, copyrighted content, or harmful knowledge – all the while preserving the model’s overall capabilities on unrelated tasks. We specifically focus on LLM unlearning (Yao et al., 2024) across 3 datasets and a variety of open-weight models (see below).

**Implementation** The code for all tasks is adapted from OpenUnlearning (Dorna et al., 2025). Preservation of general capability is evaluated using LMEvalHarness (Gao et al., 2024).

**Editable modules** We include a single editable module in Model Unlearning: the loss function, which should typically balance two objectives: unlearning specific knowledge from the forget set while preserving performance on the retain set.

**Datasets** We support 3 different datasets for Model Unlearning: TOFU (Maini et al., 2024), MUSE (Shi et al., 2024), and WMDP-Cyber (Li et al., 2024).

**Models** We support 13 different open-weight LLMs from 5 providers: Llama-2-7b-chat-hf, Llama-2-7b-hf, Llama-2-13b-hf, Llama-3.2-1B-Instruct, Llama-3.2-3B-Instruct, Llama-3.1-8B-Instruct (Touvron et al., 2023), Gemma-7b-it (Mesnard & et al., 2024), Phi-1.5, Phi-3.5-mini-instruct (Abdin et al., 2024), Qwen2.5-1.5B-Instruct, Qwen2.5-3B-Instruct, Qwen2.5-7B-Instruct (Qwen, 2024) and Zephyr-7b-beta (Tunstall et al., 2023).

## A.8. Off-Policy RL

**Task Domain** Off-policy RL refers to a class of reinforcement learning approaches in which an agent learns from experience generated by a behaviour policy that may differ from the policy being optimised, including experience collected in the past or by other agents. In this task, we focus on value-based methods that learn value functions by minimising the temporal-difference (TD) error (Sutton & Barto, 2020).

**Implementation** The code for Off-Policy RL is adapted from the Deep Q-learning (Mnih et al., 2013, DQN) implementation from PureJaxRL (Lu et al., 2022), which is itself based on CleanRL (Huang et al., 2022a;b).

**Editable modules** We consider six editable modules: (i) the loss function, which determines the prediction targets for the value network; (ii) the optimiser; (iii) the network architecture; (iv) the replay mechanism, which specifies how experience is stored and sampled during training; (v) the policy, which governs the trade-off between exploration and exploitation; and (vi) the training loop.

**Datasets** We support Off-Policy RL on four environments from the MinAtar suite (Young & Tian, 2019)—Asterix, Breakout, Freeway, and Space Invaders—as reimplemented in Gymnas (Lange, 2022) using JAX. These environments are simplified versions of their corresponding Atari games (Bellemare et al., 2013).

## A.9. On-Policy RL

**Task Domain** On-Policy RL is a subset of reinforcement learning in which an agent learns from experience collected by its own policy (Sutton & Barto, 2020), rather than from data collected by a different behaviour policy.

**Implementation** The code for On-Policy RL is adapted from the Proximal Policy Optimisation (Schulman et al., 2017, PPO) implementation from PureJaxRL (Lu et al., 2022), which is itself based on CleanRL (Huang et al., 2022a;b).

**Editable modules** We include four editable modules in On-Policy RL: the loss function; the optimiser; the network architecture; and the training loop.

**Additional Backends** In addition to the default, we support two backends that can be used to augment tasks in On-Policy RL. These are: a recurrent agent, in which the train loop must support a recurrent variable produced by the agent’s networks; and a transformer agent (Vaswani et al., 2017), in which the agent architecture uses attention.

**Datasets** We support 13 different environments for On-Policy RL, from three different environment suites. These include: Ant, HalfCheetah, Hopper, Humanoid, Pusher, Reacher and Walker2D from Brax (Freeman et al., 2021), a JAX-based (Bradbury et al., 2018) implementation of MuJoCo (Todorov et al., 2012); Asterix, Breakout, Freeway and SpaceInvaders from Gymnas (Lange, 2022), a reimplementation of four MinAtar environments (Young & Tian, 2019) which are simplifications of some Atari games (Bellemare et al., 2013); and Craftax and Craftax-Classic (Matthews et al., 2024), which are based on Crafter (Hafner, 2021).

### A.10. Unsupervised Environment Design

**Task Domain** Unsupervised Environment Design (UED) is a field focused on training agents that are robust, and able to generalise to a wide distribution of tasks (Dennis et al., 2021; Jiang et al., 2021b; Parker-Holder et al., 2022a; Samvelyan et al., 2023). In particular, this tends to be posed as a two-player game between a task-proposing adversary and a task-solving student (Dennis et al., 2021). There are several objective functions that the adversary can use, ranging from theoretically principled ones such as regret (Dennis et al., 2021) to more empirically motivated ones like positive value loss (Jiang et al., 2021a) and learnability (Rutherford et al., 2024).

**Implementation** We use the base scaffold from Sampling for Learnability (Rutherford et al., 2024), which uses PPO as the underlying learning algorithm, and training comprises two stages. The first is sampling a large batch of random environments, then rolling out the agent on these, and using these trajectories to score each level. The second phase then trains the agent on a mixture of the high-scoring levels, and uniform random environments.

**Editable modules** The first editable module is responsible for rolling the agent out on candidate environments, and then providing a score for each of them. The second is the train step, which controls the actual RL agent update, as well as how the filtered levels are used during training. Finally, the last module consists of the hyperparameters for the sampling and training process.

**Datasets** We consider two distinct domains; the first is Minigrid (Chevalier-Boisvert et al., 2023), which is a partially-observable 2D navigation task. The second is Kinetix (Matthews et al., 2025), an open-ended domain of 2D physics tasks. The task distribution consists of a broad distribution of randomly-generated physics puzzles, and the goal is to generalise to interesting, human-designed problems. There are three levels of difficulty in the Kinetix benchmark, corresponding to how many entities there are in a scene. We use Minigrid and the Small Kinetix setting as training tasks, and Medium and Large as the testing tasks.

## B. Additional Related Work

### B.1. Automated Research

AutoML involves trying to apply machine learning to new data without needing human experts (Hutter et al., 2019; He et al., 2021; Parker-Holder et al., 2022b). Historically, AutoML has focused on areas like hyperparameter tuning (e.g., (Lindauer et al., 2022; Li et al., 2018; Parker-Holder et al., 2021; Eimer et al., 2023)), selecting algorithms from a possible set (Feurer et al., 2015; Lindauer et al., 2017; Feurer et al., 2022), neural architecture search (Stanley & Miikkulainen, 2002; Stanley et al., 2009; Zoph & Le, 2017; Zoph et al., 2018; White et al., 2023) and data cleaning and augmentation (Zhang et al., 2017; Neutatz et al., 2022; Krishnan et al., 2016). While many of these techniques involve applying *existing* machine learning algorithms, the objective of automated algorithm discovery is to develop *new* machine learning algorithms without relying on manual design.

Meta-learning involves learning algorithms from data in a ‘meta-loop’ (Schmidhuber, 1987; Real et al., 2020; Beck et al., 2023). Often, meta-learned algorithms are black-box, taking the form of a neural network. For example, prior work has considered meta-learning optimisation algorithms (e.g., (Andrychowicz et al., 2016; Metz et al., 2019; Oh et al., 2020; Metz et al., 2022a;b; Goldie et al., 2024; Lan et al., 2024; Oh et al., 2025)) or loss functions (e.g., (Kirsch et al., 2020; Bechtle et al., 2021; Lu et al., 2022; Alfano et al., 2023)). Optimising these black-box algorithms frequently relies on evolution (Metz et al., 2022b; Lu et al., 2022; Jackson et al., 2023; Goldie et al., 2024; 2025) or meta-gradients (Oh et al., 2020; 2025). However, more interpretable algorithms can also be discovered using symbolic evolution or search (Ramachandran et al., 2017; Cranmer et al., 2020; Zheng et al., 2022; Chen et al., 2023b), or, driven by the rise of highly capable language models (e.g., (OpenAI, 2025c; Pichai et al., 2025; Meta AI, 2025; Anthropic, 2025b)), repeatedly prompting language models for new algorithm suggestions (Lu et al., 2024a; Romera-Paredes et al., 2024; Novikov et al., 2025; Nathani et al., 2025; Toledo et al., 2025; Gideoni et al., 2025). In this paper, we frame each algorithm discovery problem as its own *task*, and show how the DiscoGen procedural task generator can help optimise these algorithm discovery systems.

Given a rise in language model capabilities (METR, 2025; Chollet et al., 2025; Phan et al., 2025; Paglieri et al., 2025), creating better coding (Yang et al., 2024; Jiang et al., 2025; Anthropic, 2025a; OpenAI, 2025b) and research agents (Lu et al., 2024b; Nathani et al., 2025; Toledo et al., 2025) has developed into an important sub-field. Such systems have been used for increasingly complex mathematical (Hubert et al., 2025; Luong et al., 2025), software engineering (Yang et al., 2024; Liu et al., 2025) and research (Taylor et al., 2022; Lu et al., 2024b; Yamada et al., 2025) tasks. LLM agents augment base language models with the ability to take actions or use tools (Wang et al., 2024a; Schick et al., 2023), enabling them to run code (Nathani et al., 2025; Yang et al., 2024; Anthropic, 2025a), search the internet (Nakano et al., 2021; Thoppilan et al., 2022; Komeili et al., 2021; Gao et al., 2022; OpenAI, 2025a; Google), or access applications like calculators for verifiable tasks (Cobbe et al., 2021b). These tools are used by AI research agents to automate all or part of the research process in a ReAct loop (Yao et al., 2023), in which tools are used intermittently throughout a task. In this work, we propose a new procedural task generator for algorithm discovery tasks to help drive forward development of new agents and models for algorithm discovery.

The development and analysis of such agents includes developing agents purely for ideation (Si et al., 2024), implementing research ideas proposed by a human-in-the-loop (Gottweis et al., 2025; Weston & Foerster, 2025), judging the quality of research papers (Lu et al., 2024b; Si et al., 2024; Thakkar et al., 2025), or even automating the entire workflow to write scientific articles (Lu et al., 2024b; Yamada et al., 2025; Intology, 2025; Si et al., 2026). Methods for integrating LLMs into algorithm discovery systems have taken a range of forms. For example, LLMs are often used as mutation and crossover operators in evolutionary algorithms (Ma et al., 2024; Klissarov et al., 2025; Romera-Paredes et al., 2024). However, such systems vary widely in how much agency they give to the LLM. Design decisions include how the agent should search over new algorithms using, for example, tree-search algorithms (Jiang et al., 2025; Toledo et al., 2025), whether the LLM should decide which experiments to run (Yu et al., 2025), or which algorithm to submit as the ‘final’ version (Nathani et al., 2025).

### B.2. Optimising Agents

To optimise a model, architecture, or system for generalising over a problem space, we need to be able to evaluate it over a large amount of data (Kaplan et al., 2020; Hoffmann et al., 2022; Chung et al., 2024). For example, language models improve as they are pretrained on more data (Kaplan et al., 2020), and generalisation in deep reinforcement learning (RL) has benefitted from procedurally generated and larger environments (Cobbe et al. (2020), Section 2.3). However, it is often preferable to develop models with specialist capabilities; consistently focusing on generalist improvements can introduce safety concerns (Bommasani et al., 2022; Weidinger et al., 2021), and may not be the most effective way to enhance desired

capabilities (Belcak et al., 2025).

Instead, prior work has often focused on building agentic architectures or training models for specific domains. For example, significant effort has gone into developing better agents for mathematics (e.g., (Lewkowycz et al., 2022; Trinh et al., 2024; Hubert et al., 2025)) or programming (e.g., (Li et al., 2022; Jimenez et al., 2024; Rozière et al., 2024)). Underpinning many of these advances is access to an evaluation signal from a large, diverse and verifiable problem set which can be optimised by both models, via training (Shao et al., 2024; Wen et al., 2025), and researchers, through agent and prompt design. For example, there are large suites of mathematical problems (e.g., (Cobbe et al., 2021a; Hendrycks et al., 2021b)), factual questions (Joshi et al., 2017; Hendrycks et al., 2021a), and programming suites can leverage open-source code repositories to verify that inputs lead to expected outputs (Jimenez et al., 2024; Chen et al., 2021). Developing these large suites for algorithm discovery has proven more difficult; manual curation of codebases requires expert knowledge, and often needs adaptation to be applied to different data. Additionally, copying outputs from existing code is insufficient for algorithm discovery, wherein the objective is develop *superhuman* algorithms that are better than those already in existence (Romera-Paredes et al., 2024; Novikov et al., 2025). In this work, we propose a procedural algorithm discovery task generator that spans millions of possible algorithm discovery tasks, and does not require domain-specific expert knowledge for generating new tasks.

### B.3. Procedural Generation

Procedural generation involves automatically creating levels or environments, according to rules, rather than manually designing every instance individually (Togelius et al., 2013). To do so, environments in procedural generation are defined as Contextual Markov Decision Processes (Hallak et al., 2015, Contextual MDPs) or Underspecified Partially Observable MDPs (Dennis et al., 2021), where each level or task is defined by a small number of configuration variables. In deep RL, procedural environment generation has proven effective for training agents on a *distribution* of levels, such that they generalise over a distribution rather than just fitting the specific environment levels seen (Cobbe et al., 2020; Frans & Isola, 2023). Such approaches have been applied to generate levels in environments of ranging complexity, from gridworlds (Chevalier-Boisvert et al., 2023) to physics engines (Frans & Isola, 2023; Matthews et al., 2025), 3-dimensional worlds (Stooke et al., 2021), or games (Shaker et al., 2016; Küttler et al., 2020; Cobbe et al., 2020; Hafner, 2021; Wang et al., 2021; Matthews et al., 2024). Procedural generation opens up new avenues of research too, by enabling the development of autocurricula (e.g., (Jiang et al., 2021b; Dennis et al., 2021; Parker-Holder et al., 2022a)) or large scale meta-learning (e.g., (Nikulin et al., 2024)).

## C. Experimental Details

### C.1. Hyperparameters

All DiscoBench experiments are run over three meta-seeds; that is, three independent algorithm discovery runs. In all experiments, agents are given a **24 hour time limit**, and either an 80 (for *DiscoBench Single*) or 100 (for *DiscoBench All*), since tasks are more complicated) action budget, in which to complete a task. Within said limit, the agent is allowed to run many experiments for different implementations, each of which returning performance metrics for all meta-train datasets. Furthermore, for *DiscoBench Single*, we run additional experiments until each agent has 3 valid attempts. For this setting, we remove a small number of tasks (On-Policy RL/Off-Policy RL train\_loop and Model Unlearning) since they proved too difficult to produce three successful attempts.

In all experiments, we use the MLGym agent with different open-source LLMs. We do not add or change the implementations of tools from MLGym. We use default hyperparameters from MLGym for all experiments Nathani et al. (2025), besides those specified below:

*Table 4.* Non-default MLGym hyperparameters.

| Setting                              | gpus_per_agent | max_steps |
|--------------------------------------|----------------|-----------|
| <i>DiscoBench Single</i>             | 1              | 80        |
| <i>DiscoBench All</i>                | 1              | 100       |
| <i>Prompt Tuning (Meta-Training)</i> | 4              | 50        |
| <i>Prompt Tuning Single</i>          | 1              | 50        |
| <i>Prompt Tuning All</i>             | 1              | 60        |

### C.2. Elo Calculation

Our results are comparative between models, and based on an Elo score. In Elo, each model’s score changes based on pairwise win-loss comparisons of their performance in individual tasks, and the Elo disparity between the model. Before calculating the Elo, we aggregate the per-dataset scores for each model over its meta-seeds; we also include a penalty to scores which ensures that a model with more successful attempts out of its three meta-seeds dominates a model with less successful attempts.

For model 1 with rating  $R_1$  and model 2 with rating  $R_2$ , the expected outcome score of model 1 is

$$E_1 = \frac{1}{1 + 10^{(R_2 - R_1)/400}},$$

where a 400 point difference corresponds to a  $\approx 91\%$  chance of winning.

After calculating  $E_1$ , the rating of model 1 is updated to reflect the difference between the expected and true score as

$$R'_1 = R_1 + K \cdot (S_1 - E_1).$$

In this expression,  $S_1$  is the score for model 1, and is set as 1.0 for a win, 0.5 for a draw and 0.0 for a loss. Scores for all models are initialised at 1000, and  $K = 32$  to balance volatility of score calculations. We loop through the data for 1000 epochs, shuffling each epoch and annealing  $K$  to 1 for stability. We use 100 bootstrapping rounds to estimate 95% confidence intervals.

### C.3. Prompt Optimisation Experimental Details

For our prompt optimisation experiment, we use Claude-4.5 Sonnet (Anthropic, 2025b) to automatically refine prompts for an ADA; here the ADA is the MLGym agent with Deepseek-V3.2 (DeepSeek-AI et al., 2025; Nathani et al., 2025). We sweep over different numbers of tasks for prompt optimisation by changing the *frequency* at which a task is sampled. As such, a new task is sampled every  $30/N_{tasks}$  steps.

The task sampling process involved first sampling a task domain from the ten possible in DiscoGen. Each dataset from the task domain had a 40% chance of inclusion in meta-train or meta-test, and a 20% chance of being excluded from the task.

Every module has an independent 30% chance of being marked as editable. Any invalid task (i.e., no editable modules or an empty meta-train/meta-test set) is discarded, and we resample from DiscoGen. We cap the maximum amount that each domain can be sampled to 10, to prevent domain bias, but this limit is not reached in practice over 30 ADA optimisation iterations.

To prevent unbounded context growth for the ADA Optimisation LLM (Claude-4.5 Sonnet), we do not provide full conversation history. Instead, at every ADA optimisation step, we provide: a system prompt (Appendix [L](#)); the current and three preceding prompts with their performance; and up to 15 previous task domain-performance pairs without their corresponding prompt, to help ground scores. It is not told the specific task configuration, to aid development of a general prompt. Claude is given per-dataset breakdowns, as well as whether each dataset is from meta-train or meta-test in a given task. We also tell Claude if there were any failures or error messages, but do not provide full error messages; again to limit the context size. If Claude does not produce a prompt fitting the prescribed template within three attempts, the task is discarded and resampled without increasing the ADA optimisation iteration (i.e., the task is not counted towards the total).

#### C.4. Experimental Compute

All experiments are run using Nvidia H200 GPUs. We run the majority of experiments on a single GPU, besides prompt optimisation experiments which are run on 4 GPUs to accelerate experimentation. For all experimentation (including most development and experiments which are not directly included in this paper), we use an estimated 23,000 GPU-hours of compute. However, such high-end compute is not generally a requirement for usage. To ensure DiscoGen is accessible to researchers under different constraints, we verified that many DiscoGen task domains can run on consumer-grade (e.g., Nvidia 2080Ti/3080) and mid-range (Nvidia L40s) hardware.

All models were run through an API. The total cost of API credits used in development and experiments is estimated to be about \$1200. A task generally costs on the order of \$0.10-\$0.50 to run with open-source models.

## D. DiscoGen Details

### D.1. Technical DiscoGen Implementation

Here, we clarify the implementation details of DiscoGen; in particular, how tasks are generated.

DiscoGen operates by creating *file systems*. To sample a random task from DiscoGen, a random DiscoGen configuration must be created. Doing so involves selecting a task domains, which defines the availability of modules, datasets and backends, and randomising each of these categories.

After a configuration has been created, DiscoGen is queried to build the meta-train portion of the task. To do so, DiscoGen creates: (1) all necessary files for *each* meta-training dataset, including downloading and caching any data; (2) a script for running an inner-loop on all meta-training datasets; (3) a directory which includes all discovered algorithms; and (4) a procedurally generated description file which describes the task domain, the meta-training datasets, the backend and the editable modules based on the pre-written per-domain text in **M**.

Any ADA operates in a meta-loop over the meta-train files to develop new algorithms. When the meta-loop is complete, DiscoGen is queried again to build the meta-*test* task. As this is created, **all** files besides the editable modules in discovered/ are overwritten, and rebuilt from scratch for different datasets. This is done to lower the risk of evaluation hacking. Since the meta-test datasets are not known to the agent, we also dramatically reduce the risk of train-test leakage.

### D.2. Deriving Task Counts

For  $m$  different modules,  $d$  different datasets, and  $b$  different backends in a task domain, we derive the number of valid tasks below.

Each dataset can be marked as part of the *meta-train* set, *meta-test* set, or *excluded* set, given 3 possible options per dataset. The same dataset can not be included more than once, to prevent leakage between the meta-train and meta-test sets (i.e.,  $\mathcal{D}_{train} \cap \mathcal{D}_{test} = \emptyset$ ). Therefore, the number of possible combinations of datasets is  $3^d$ . However, we require *at least* one dataset to be in the meta-train, and *at least* one dataset in meta-test. We remove the two  $2^d$  cases where this is not true (i.e., where either meta-train or meta-test are excluded as options), but add back the double-counted case of all exclude. This produces  $(3^d - 2 \times 2^d + 1) = (3^d - 2^{(d+1)} + 1)$  valid dataset configurations.

For  $m$  modules, each module can be marked as *editable* or *fixed*, meaning each module has 2 possible states. This means there are  $2^m$  possible module combinations. By removing the case where all modules are *fixed*, this gives  $2^m - 1$  valid modules configurations.

By combining the number of configurations for datasets and modules, and multiplying by the number of backends, this gives

$$N_{tasks} = b \cdot (2^m - 1) \cdot (3^d - 2^{(d+1)} + 1).$$

For Model Unlearning, where we can provide one of  $n$  different base models for each dataset, the computation changes slightly. As opposed to counting 3 options for every dataset there are  $2n + 1$  possible choices; any combination of meta-train/model ( $n$ ), meta-test/model ( $n$ ), or exclude (1). This provides  $(2n + 1)^d$  combinations. The degenerate cases now count as removing  $(n + 1)^d$  combinations each, meaning we remove  $2(n + 1)^d$  in place of the previous  $2 \times 2^d$ . This gives

$$N_{tasks} = b \cdot (2^m - 1) \cdot ((2n + 1)^d - 2(n + 1)^d + 1).$$

## E. On-Policy RL Combination Results

To expand upon the results of Section 5.2, in this section we provide a focused analysis of how changing the *editable* modules affects the performance of agents. Specifically, in On-Policy RL, we sweep over all 15 possible module combinations with the same dataset splits as in Section 5.2 (i.e., meta-train is comprised of Breakout and Freeway, and meta-test of Asterix and Space Invaders, from Young & Tian (2019)). Due to the vast number of domains supported in DiscoGen, such analysis is unaffordable across all possible domains. Before more in-depth analysis, we examine the performance of each LLM on the different On-Policy RL module combinations in Table 5.

Table 5. ADA performance over all On-Policy RL module combinations (ELO Scores with 95% CIs).

| Model         | On-Policy RL Experiments |                          |                          |
|---------------|--------------------------|--------------------------|--------------------------|
|               | Succ.                    | Meta-Train               | Meta-Test                |
| Baseline      | —                        | 1182 [1107, 1260]        | <b>1267</b> [1185, 1350] |
| Deepseek-v3.2 | <b>41.1%</b>             | <b>1192</b> [1111, 1281] | 1218 [1138, 1311]        |
| GPT-OSS 120B  | 20.0%                    | 762 [672, 837]           | 631 [467, 742]           |
| Devstral2     | 18.9%                    | 865 [756, 961]           | 883 [777, 978]           |

Interestingly, we find that Deepseek-v3.2 performs *within confidence* of the ‘*all fixed*’ baseline, both in meta-train and in meta-test, over the different On-Policy RL tasks. Additionally, we consider how the performance of each agent changes with respect to the number of editable modules. Firstly, we examine how the success rate of each agent changes as we change the number of modules in Table 6.

Table 6. Success rate by number of editable modules in On-Policy RL.

| Model         | Success Rate (%) by Editable Modules |      |     |     |
|---------------|--------------------------------------|------|-----|-----|
|               | 1                                    | 2    | 3   | 4   |
| DeepSeek-V3.2 | 75.0                                 | 47.2 | 8.3 | 0.0 |
| GPT-OSS-120b  | 50.0                                 | 11.1 | 8.3 | 0.0 |
| Devstral2     | 29.2                                 | 27.8 | 0.0 | 0.0 |

There is a consistent trend: adding more *editable* modules, even when meta-train and meta-test datasets are kept the same, leads to reduced success rates. Clearly, the tasks that include more editable modules are more difficult, providing direct evidence that changing modules leads to semantically different tasks. To investigate this further, we consider how success rates compare for each module combination when aggregated over all language models in Table 7. Furthermore, we show the *maximum* score achieved by any LLM in each environment, to explore if there are any signs that different module combinations can lead agents to produce better or worse scores.

Table 7 demonstrates a clear pattern in success rates; developing optimisation algorithms is easier than loss functions, which are easier to implement than networks, and developing correct training loop logic is most difficult. This translates into the combinations with more editable modules, where implementing a loss and optimiser has a higher success rate than networks and train, for example. Interestingly, it also appears that the maximum score for *every* environment was achieved in a setting with 2 editable modules. While it is impossible to confirm whether such a trend could continue into 3 and 4 editable modules, due to low success rates, there are signs that tasks with more editable modules *can* exhibit higher performance due to greater ADA flexibility, emphasising further the semantic diversity of tasks. In fact, for *every* individual module, a higher max score was attained by combining it with another module.

Our analysis demonstrates the importance and validity of introducing more diversity by expanding module combinations. In addition to a general pattern of increasing complexity through expanding the editable module count, we show how each module and module combination performs slightly differently, revealing a hierarchy of difficulty. This diversity of difficulty also demonstrates the natural application of autocurriculum for DiscoGen (Section 6).

Table 7. Average success rate and maximum per-environment performance by module configuration. Any environment for which no ADA implemented a successful algorithm is marked with —.

| Configuration                      | Success Rate (%) | Meta-Train (Eval Return) |         | Meta-Test (Eval Return) |          |
|------------------------------------|------------------|--------------------------|---------|-------------------------|----------|
|                                    |                  | Breakout                 | Freeway | Asterix                 | SpaceInv |
| <i>1 Editable Module</i>           |                  |                          |         |                         |          |
| Optimiser                          | 83.33            | 74.97                    | 62.86   | 18.11                   | 181.25   |
| Loss                               | 77.78            | 83.91                    | 62.58   | 39.60                   | 179.50   |
| Network                            | 33.33            | 99.97                    | 68.07   | 14.20                   | 189.75   |
| Train                              | 11.11            | 8.41                     | 8.41    | 3.73                    | 177.12   |
| <i>2 Editable Module</i>           |                  |                          |         |                         |          |
| Loss + Optimiser                   | 61.11            | 84.47                    | 62.89   | 20.50                   | 181.38   |
| Network + Optimiser                | 44.44            | 91.44                    | 65.25   | 19.77                   | 191.62   |
| Loss + Network                     | 38.89            | 106.12                   | 66.16   | 69.42                   | 184.00   |
| Loss + Train                       | 22.22            | 8.56                     | 61.91   | 3.91                    | 169.38   |
| Optimiser + Train                  | 5.56             | 34.61                    | 29.45   | —                       | —        |
| Network + Train                    | 0.00             | —                        | —       | —                       | —        |
| <i>3 Editable Module</i>           |                  |                          |         |                         |          |
| Loss + Network + Optimiser         | 11.11            | 88.58                    | 65.20   | 64.85                   | 186.75   |
| Loss + Optimiser + Train           | 11.11            | 0.30                     | 2.95    | —                       | —        |
| Loss + Network + Train             | 0.00             | —                        | —       | —                       | —        |
| Network + Optimiser + Train        | 0.00             | —                        | —       | —                       | —        |
| <i>4 Editable Module</i>           |                  |                          |         |                         |          |
| Loss + Network + Optimiser + Train | 0.00             | —                        | —       | —                       | —        |

## F. DiscoBench Task List

Here, we provide a list of the meta-train/meta-test dataset splits for tasks in DiscoBench. All datasets are described and referenced in Appendix A.

### F.1. Bayesian Optimisation

**Meta-Train Datasets:** Ackley1D, Branin2D, Cosine8D, Eggholder2D, Hartmann6D, Levy6D,

**Meta-Test Datasets:** Ackley2D, Bukin2D, DropWave2D, Griewank5D, HolderTable2D

### F.2. Brain Speech Detection

**Meta-Train Datasets:** LibriBrain Sherlock Holmes 1-3

**Meta-Test Datasets:** LibriBrain Sherlock Holmes 4-7

### F.3. Computer Vision Classification

**Meta-Train Datasets:** CIFAR10, FashionMNIST, MNIST, OxfordFlowers

**Meta-Test Datasets:** CIFAR100, CIFAR10C, CIFAR10LT, StanfordCards, TinyImageNet

### F.4. Continual Learning

**Meta-Train Datasets:** SplitCIFAR100

**Meta-Test Datasets:** PermutedMNIST, TinyImageNetSplit

### F.5. Greenhouse Gas Prediction

**Meta-Train Datasets:** CH4, SF6

**Meta-Test Datasets:** CO2, N2O

### F.6. Language Modelling

**Meta-Train Datasets:** LMFineWeb, OPCFineWebMath

**Meta-Test Datasets:** OPCFineWebCode, TinyStories

### F.7. Model Unlearning

**Meta-Train Datasets:** MUSE

**Meta-Test Datasets:** TOFU, WMDP-Cyber

**Base Model:** Qwen2.5-1.5B-Instruct

### F.8. Off-Policy RL

**Meta-Train Datasets:** MinAtar/Breakout, MinAtar/Freeway

**Meta-Test Datasets:** MinAtar/Asterix, MinAtar/SpaceInvaders

**F.9. On-Policy RL**

**Meta-Train Datasets:** MinAtar/Breakout, MinAtar/Freeway

**Meta-Test Datasets:** MinAtar/Asterix, MinAtar/SpaceInvaders

**F.10. Unsupervised Environment Design**

**Meta-Train Datasets:** Kinetix/Small, Minigrid

**Meta-Test Datasets:** Kinetix/Medium, Kinetix/Large

## G. Redundancy Analysis

To ensure DiscoGen tasks exhibit semantic diversity, rather than redundancy, in this section we produce and analyse three Spearman Rank Correlation plots based on *DiscoBench Single (Until Success)* results from Section 5.2.

For each task, we compute the *average rank* of the models and baseline over datasets in the task. In Figure 2, we combine meta-train and meta-test results into one set and compute average rank over all datasets. We compute the Spearman Rank correlation between all tasks, which is shown through a heatmap. In Figures 3a and 3b, we show the rank correlations over only the meta-train and meta-test sets for each task respectively.

We use hierarchical clustering in Figures 2 and 3a to group together tasks with similar average rankings. For Figure 3b, we *keep* the task-ordering from the meta-train clustering; this helps visualise the consistency in rank ordering between meta-train and meta-test for the same task.

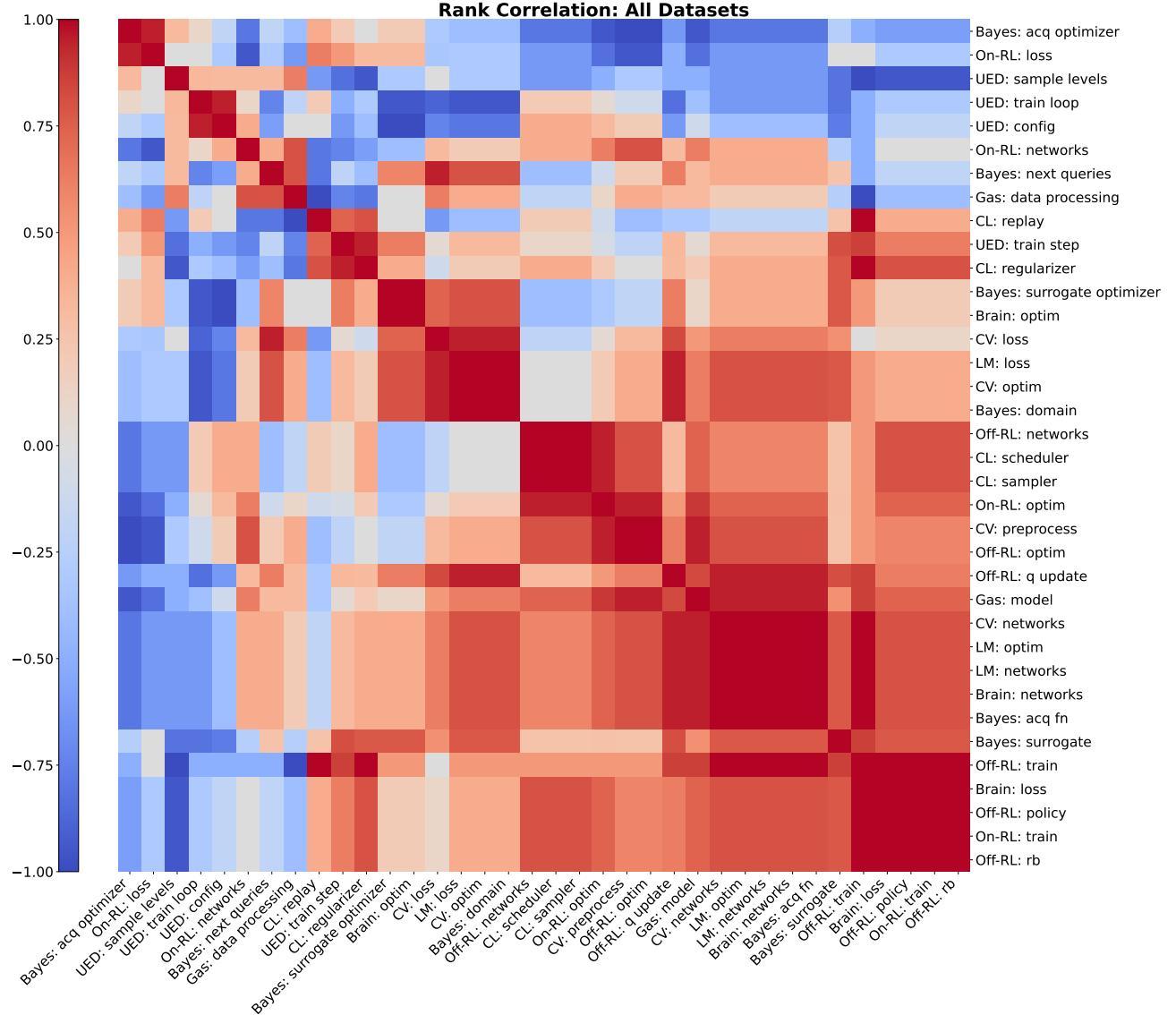
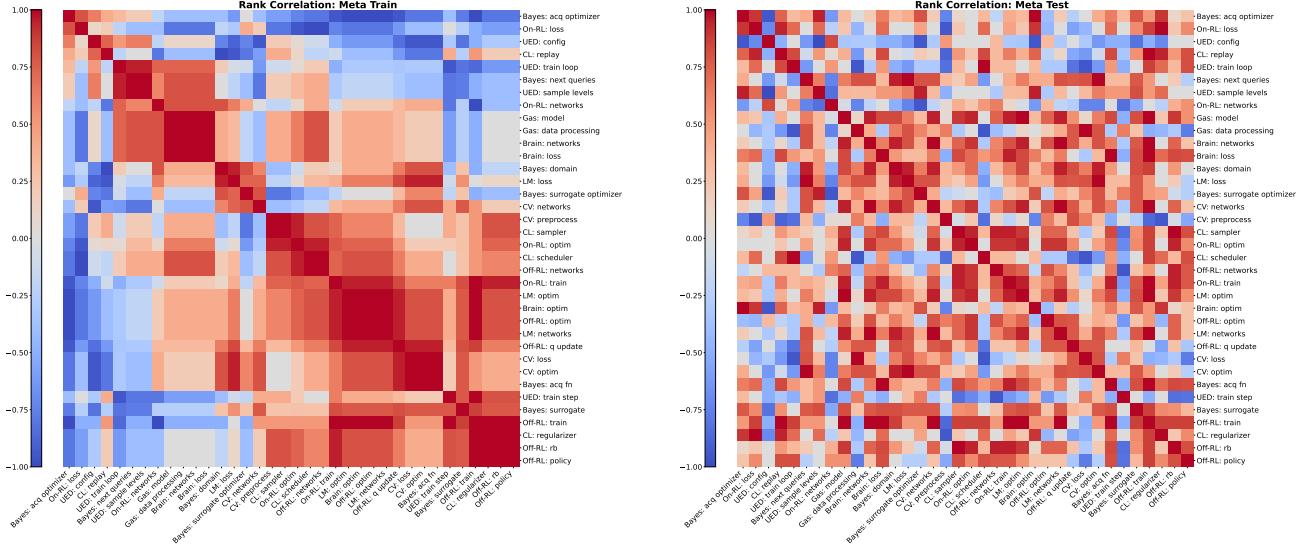


Figure 2. Clustered average rank correlation over all datasets (meta-train and meta-test) for each task.

In Figure 2, we find clusters of high-correlation tasks that are occasionally, though not always, intuitive. There is often, though not always, high correlation between tasks in which the network architecture is the editable module. Alternatively, there is notably *less* correlation when the editable module is the optimiser. We also find frequently mixed correlation between tasks from the same domain; for example, tasks in on-policy RL or Bayesian Optimisation are frequently anti-correlated.



(a) Clustered average rank correlation over meta-train datasets.

(b) Average rank correlation over meta-test datasets, ordering the tasks based on meta-train clustering.

*Figure 3.* Rank correlations for Meta-Train (left) and Meta-Test (right). The Meta-Test plot inherits the clustering order from Meta-Train to visualize consistency across splits.

In Figure 3, we consider the consistency of clusters between meta-train and meta-test; essentially, exploring whether high and low correlation patterns are reflected between the two regimes, to explore alignment when *only* the datasets are changed. To do so, we first compute the order of labels by clustering the *meta-train* heatmap, and keep this order fixed in the *meta-test* plot. If patterns persist, it would suggest that the rank correlations are similar.

Compared to the clustered plot for meta-train (Figure 3a), the correlation structure in meta-test (Figure 3b) effectively dissolves to noise. Despite the *only* difference between these plots being the datasets (i.e., Figure 3a plotting over meta-train datasets that the agent develops algorithms for, and Figure 3b plotting over the meta-test datasets that the agent doesn't see during its meta-loop), the average algorithm rank changes **dramatically** to the extent that old patterns become broadly unrecognisable. This provides strong justification to the semantic diversity, and lack of redundancy, in the combinatorial task space of DiscoGen; simply changing datasets completely changes the ranking of ADAs.

## H. Prompt Tuning Extended Results

In Table 8, we show how prompt tuning performance changes with  $K_{tasks}$  for the separate DiscoBench suites.

Table 8. Separated prompt optimisation performance (Elo Scores with 95% CIs). Bold indicates best result (including overlapping CIs).

| $K_{tasks}$ | DiscoBench Single |                          |                          | DiscoBench All |                          |                          |
|-------------|-------------------|--------------------------|--------------------------|----------------|--------------------------|--------------------------|
|             | Succ.             | Train                    | Test                     | Succ.          | Train                    | Test                     |
| 1           | 73.1%             | 955 [934, 974]           | 964 [941, 986]           | 38.6%          | 964 [918, 1010]          | 923 [851, 985]           |
| 5           | 76.7%             | 1004 [983, 1021]         | 948 [923, 974]           | <b>53.8%</b>   | <b>1076</b> [1032, 1114] | <b>1100</b> [1043, 1159] |
| 10          | 77.7%             | 990 [968, 1013]          | 1016 [993, 1041]         | 33.8%          | 840 [786, 888]           | 883 [821, 948]           |
| 30          | <b>79.0%</b>      | <b>1051</b> [1030, 1071] | <b>1071</b> [1045, 1106] | <b>53.8%</b>   | <b>1120</b> [1069, 1165] | <b>1093</b> [1025, 1150] |

Much like in the *DiscoBench Combined* ADA evaluation, we find that increasing  $K_{tasks}$  generally improves performance. While Elo is noisier in *DiscoBench All*, due to lower success rates, this is consistent with our aggregated results of Section 7.

We do not visualise ADA optimisation curves for  $K_{tasks} \neq 1$ , since tasks are frequently changing and performance is incomparable between different datasets, domains, and module combinations. However, to demonstrate the validity of our prompt improvement ADA optimisation loop, we visualise the ADA optimisation curves over the meta-train and meta-test environments (both of which are known during ADA optimisation, since meta-test is only held out from the ADA itself).

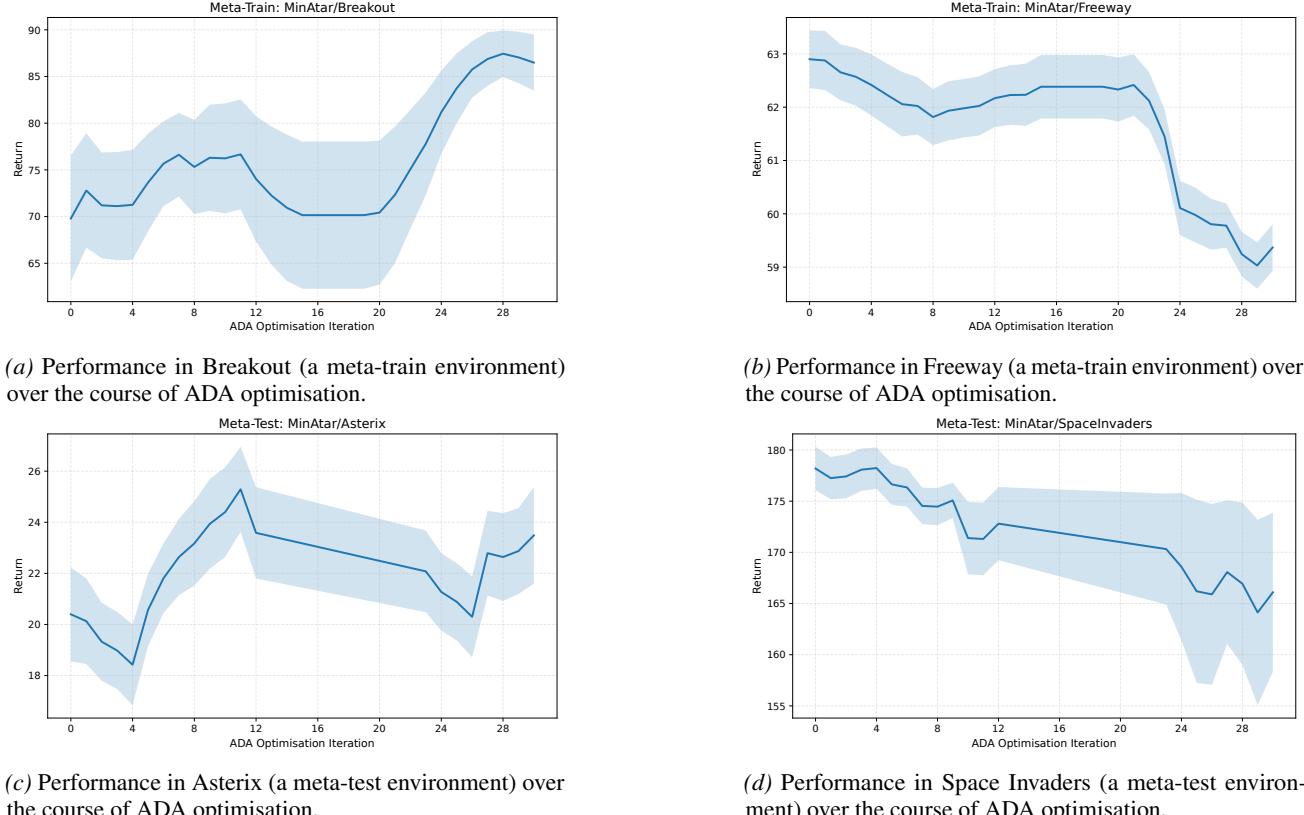


Figure 4. Visualisation of performance for the four environments that are used in the  $K_{tasks} = 1$  prompt optimisation experiment. We plot mean and standard error of the evaluation return achieved by the final policy for each environment over 8 inner-loop seeds, as defined by the On-Policy RL task domain implementation.

Given prompt optimisation using a language model is effectively a search problem, the curves are (expectedly) noisy. However, there is a general trend of improvement. Considering these curves in parallel with the final prompt (Section L.3.1) helps elucidate these curves. In particular, the prompt emphasises *improvement* in Breakout and *maintaining* in Freeway. Furthermore, the vast majority of the prompt is dedicated to scored in Breakout; it suggests making changes based on the variance of Breakout, for instance.

## I. Example Discovered Algorithms

One question to explore is whether ADAs in DiscoGen *actually* discover new algorithms, or whether successful solutions end up reproducing baselines and subsequently tweaking hyperparameters. Searching through all discovered algorithms would be difficult, given the number of independent ADA runs in our work, for different domains and module combinations. As such, we select two interesting discovered algorithms to briefly discuss here. These are chosen as two algorithms which generalised especially well from meta-train to meta-test. We note that these algorithms were *not* the best performers in meta-train (though neither performed badly), which generally did not transfer well to meta-test.

### I.1. Language Modelling:

We firstly introduce a discovered loss function for language modelling. We include the loss function code in Example 2.

```

1  from typing import Sequence
2
3  import torch.nn.functional as F
4  import torch
5  import torch.nn as nn
6  from typing import Dict, Optional
7
8  def compute_loss(outputs: Dict[str, torch.Tensor], labels: torch.Tensor,
9                  num_items_in_batch: Optional[int] = None) -> torch.FloatTensor:
10
11     """Calculate the loss for the model's outputs against the true labels.
12     Args:
13         outputs (dict): The model's outputs, typically containing logits {"logits": logit}.
14         labels (torch.Tensor): The true labels for the batch.
15         num_items_in_batch (int, optional): The number of items in the batch. Defaults to None.
16
17     Returns:
18         torch.FloatTensor: The computed loss value.
19     """
20
21     logits = outputs["logits"]
22
23     # Additive margin softmax (AM-Softmax) with margin=0.1 and scale=10
24     margin = 0.1
25     scale = 10.0
26
27     one_hot = torch.zeros_like(logits).scatter_(1, labels.unsqueeze(1), 1.0)
28     logits_margin = logits - margin * one_hot
29     logits_margin = scale * logits_margin
30
31     # Cross-entropy with label smoothing
32     loss = F.cross_entropy(logits_margin, labels, label_smoothing=0.05)
33
34     # Entropy regularization
35     probs = F.softmax(logits, dim=1)
36     entropy = -torch.sum(probs * torch.log(probs + 1e-8), dim=1).mean()
37     loss = loss + 0.01 * entropy
38
39     return loss

```

*Example 2.* A discovered loss function for language modelling.

The ADA makes a number of unintuitive design decisions that, when combined, performed close to the maximum in both meta-train and meta-test. Firstly, the agent uses additive margin softmax (Wang et al., 2018), a technique used in face verification to encourage large margins between network outputs, which is generally undesirable in language modelling where synonyms should be treated similarly. This is used in the cross-entropy calculation with a scale of 10; effectively making the target distribution significantly more sharp. Secondly, the agent introduces conflicting objectives: label-smoothing, which optimises towards *reduced* certainty in outputs (Szegedy et al., 2016), and an entropy penalty which *encourages* certainty. This all combines to give an interesting loss function; one which encourages and penalises certainty

simultaneously.

## I.2. On-Policy RL: A CNN-MLP Dual-Path Architecture For MinAtar

We also explore a dual-path network architecture (Chen et al., 2017) for on-policy reinforcement learning in MinAtar (Young & Tian, 2019). We provide discovered python code in Example 3, removing any redundant code for clarity.

```

1 from typing import Sequence
2
3 import distrax
4 import flax.linen as nn
5 import jax
6 import jax.numpy as jnp
7 import numpy as np
8 from flax.linen.initializers import constant, orthogonal
9
10
11 class ActorCritic(nn.Module):
12     action_dim: Sequence[int]
13     config: dict
14
15     @nn.compact
16     def __call__(self, x):
17         """Insert your network logic here."""
18         # Input = x. x is the environment observation.
19         # x shape: (batch, 400) after flatten.
20         hsize = self.config.get("HSIZE", 64)
21
22         # Map activation string to function
23         activation = nn.elu
24
25         # Reshape to (... , 10, 10, 4)
26         x_img = jnp.reshape(x, (*x.shape[:-1], 10, 10, 4))
27         # Apply CNN
28         cnn = nn.Conv(features=32, kernel_size=(5,5), strides=(1,1), padding='SAME',
29                       kernel_init=orthogonal(np.sqrt(2)), bias_init=constant(0.0))(x_img)
30         cnn = activation(cnn)
31         cnn = nn.LayerNorm()(cnn)
32         cnn = nn.Conv(features=64, kernel_size=(3,3), strides=(1,1), padding='SAME',
33                       kernel_init=orthogonal(np.sqrt(2)), bias_init=constant(0.0))(cnn)
34         cnn = activation(cnn)
35         cnn = nn.LayerNorm()(cnn)
36         cnn = nn.Conv(features=64, kernel_size=(3,3), strides=(1,1), padding='SAME',
37                       kernel_init=orthogonal(np.sqrt(2)), bias_init=constant(0.0))(cnn)
38         cnn = activation(cnn)
39         cnn = nn.LayerNorm()(cnn)
40         cnn = cnn.reshape((*cnn.shape[:-3], -1)) # flatten
41         # Reduce dimension
42         cnn = nn.Dense(256, kernel_init=orthogonal(np.sqrt(2)), bias_init=constant(0.0))
43         (cnn)
44         cnn = activation(cnn)
45         cnn = nn.LayerNorm()(cnn)
46         # MLP on flattened input
47         mlp = nn.Dense(hsize, kernel_init=orthogonal(np.sqrt(2)), bias_init=constant
48                      (0.0))(x)
49         mlp = activation(mlp)
50         mlp = nn.LayerNorm()(mlp)
51         mlp = nn.Dense(hsize, kernel_init=orthogonal(np.sqrt(2)), bias_init=constant
52                      (0.0))(mlp)
53         mlp = activation(mlp)
54         mlp = nn.LayerNorm()(mlp)
55         # Concatenate
56         x = jnp.concatenate([cnn, mlp], axis=-1)
57         # Dense layer to combine

```

```

55     x = nn.Dense(hsize, kernel_init=orthogonal(np.sqrt(2)), bias_init=constant(0.0))
56     x = activation(x)
57     x = nn.LayerNorm()(x)
58
59     # Policy head
60     logits = nn.Dense(self.action_dim, kernel_init=orthogonal(0.01), bias_init=
61     constant(0.0))(x)
62     # Value head
63     v = nn.Dense(1, kernel_init=orthogonal(1.0), bias_init=constant(0.0))(x)
64
65     pi = distrax.Categorical(logits=logits)
66
67     return pi, jnp.squeeze(v, axis=-1)

```

*Example 3.* A discovered neural network architecture for on-policy reinforcement learning.

The agent makes two particularly interesting and uncommon design decisions in its discovered architecture. Firstly, rather than using a ReLU activation (Nair & Hinton, 2010), as is common when optimising the PPO objective (Schulman et al., 2017; Huang et al., 2022a), the agent uses the *exponential linear unit* (Clevert et al., 2016, ELU). Secondly, while using convolutional neural networks (LeCun et al., 2015, CNN) is common in Atari games (Mnih et al., 2015; Huang et al., 2022a), and MLP policies are often used for MinAtar (Lu et al., 2022), here the ADA combines the two. Specifically, the agent builds an architecture which learns features with both local inductive bias (from the CNN) and global features (from the MLP), effectively overcoming the shortcomings of each approach, and concatenates them before the policy- and value-head. This network architecture performed exceptionally well, on average, in all four MinAtar environments.

## J. Success@k Results

We believe that the principle objective of ADA research should be to develop robust and performant algorithm discovery agents. As such, in the main body of our text we report aggregated success metrics over independent seeds of algorithm discovery; an ADA which only produces a valid algorithm in one out of its three seeds will receive a success rate of 33%. However, such reporting introduces the question of whether tasks are too difficult to provide signal for either evaluation or optimisation. As such, in Table 9 we report *success@3*. This acts as a complement to Table 2 and verifies that, for most tasks, agents are able to create *at least one* valid solution.

Table 9. *Success@3* rates in DiscoBench. Bold indicates best performance.

| <b>Model</b>  | <i>DiscoBench Single</i> | <i>DiscoBench All</i> |
|---------------|--------------------------|-----------------------|
|               | <b>Success@3</b>         | <b>Success@3</b>      |
| GPT-OSS 120B  | 90.2%                    | 40.0%                 |
| Devstral2     | 77.9%                    | <b>54.0%</b>          |
| Deepseek-v3.2 | <b>95.8%</b>             | 37.1%                 |

*Success@3* illuminates that ADAs are *able* to produce valid solutions to most tasks, albeit not consistently. As such, the large gap between average success over 3 seeds, compared to *success@3* broadly stems from low ADA robustness as opposed to an insurmountable set of tasks in DiscoBench. Furthermore, by extrapolating these results from DiscoBench to DiscoGen, they confirm that there is *signal* for ADA optimisation; as long as models can sometimes produce valid solutions, there is sufficient evidence to differentiate between meta-loop trajectories, and thus to optimise ADAs.

## K. Per-Task DiscoBench Results

Here, we present per-task DiscoBench results, which are aggregated into a per-model ELO score over meta-train and meta-test sets. These are separated into *DiscoBench Single*, *DiscoBench All*, and *DiscoBench Single (Until Success)*.

Results are reported over 3 or less meta-seeds, depending on how many runs were successful for each model. We highlight the baseline scores in red; these are the scores obtained by the default implementation for each task, when all files are set to fixed. For all results, we calculate bootstrap stratified confidence intervals (95%) and interquartile means, following Agarwal et al. (2021).

In our *Until Success* experiments (Appendices K.3 & K.4), we omit a small number of tasks in which we were unable to reliably produce three seeds for every model without incurring excessive cost. These include: On-Policy RL (Change Train), Off-Policy RL (Change Train), and Model Unlearning (Change Loss).

### K.1. DiscoBench Single – Meta-Train

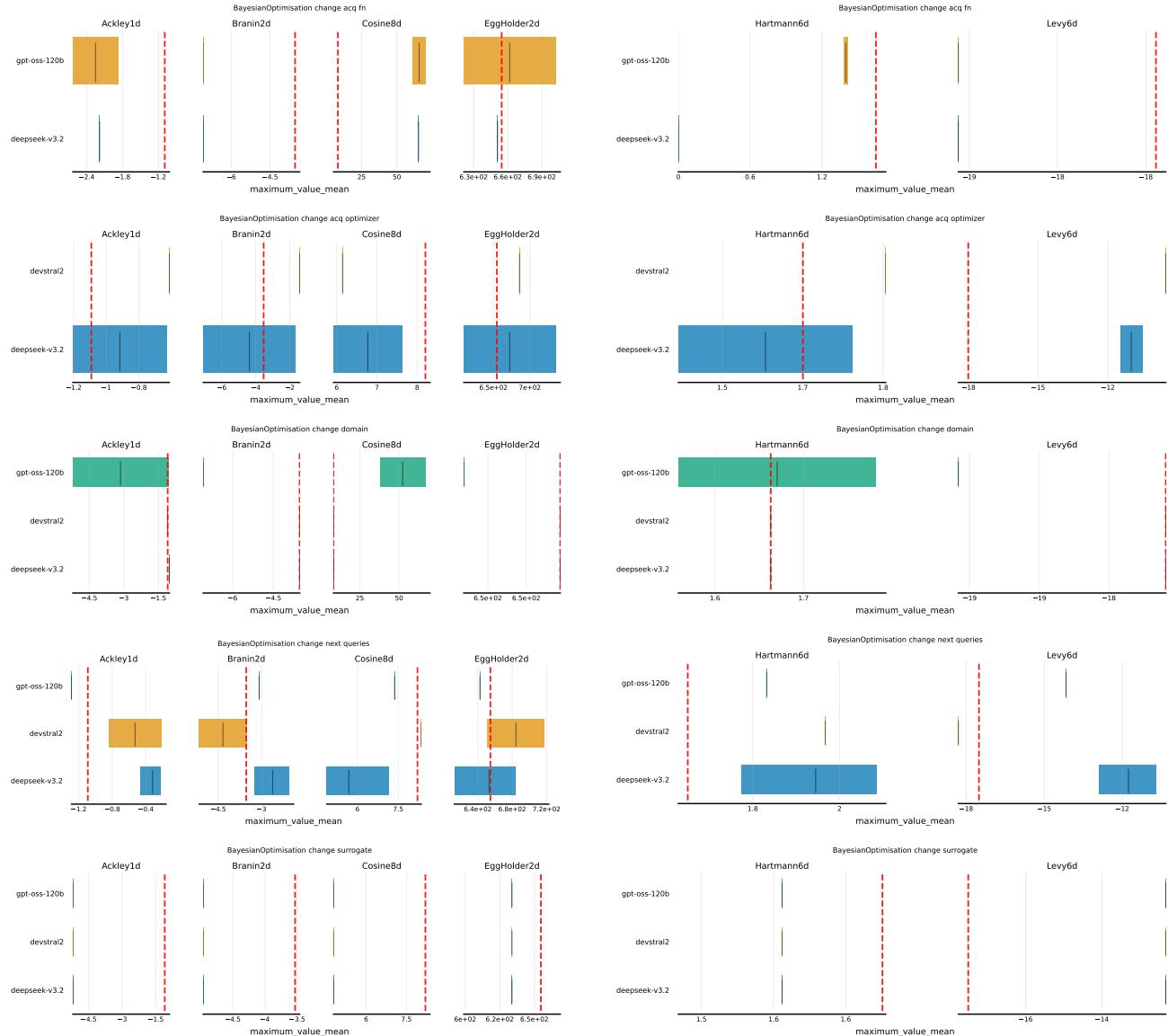


Figure 5. DiscoBench Single results on Meta-Train tasks. (Part 1/4)

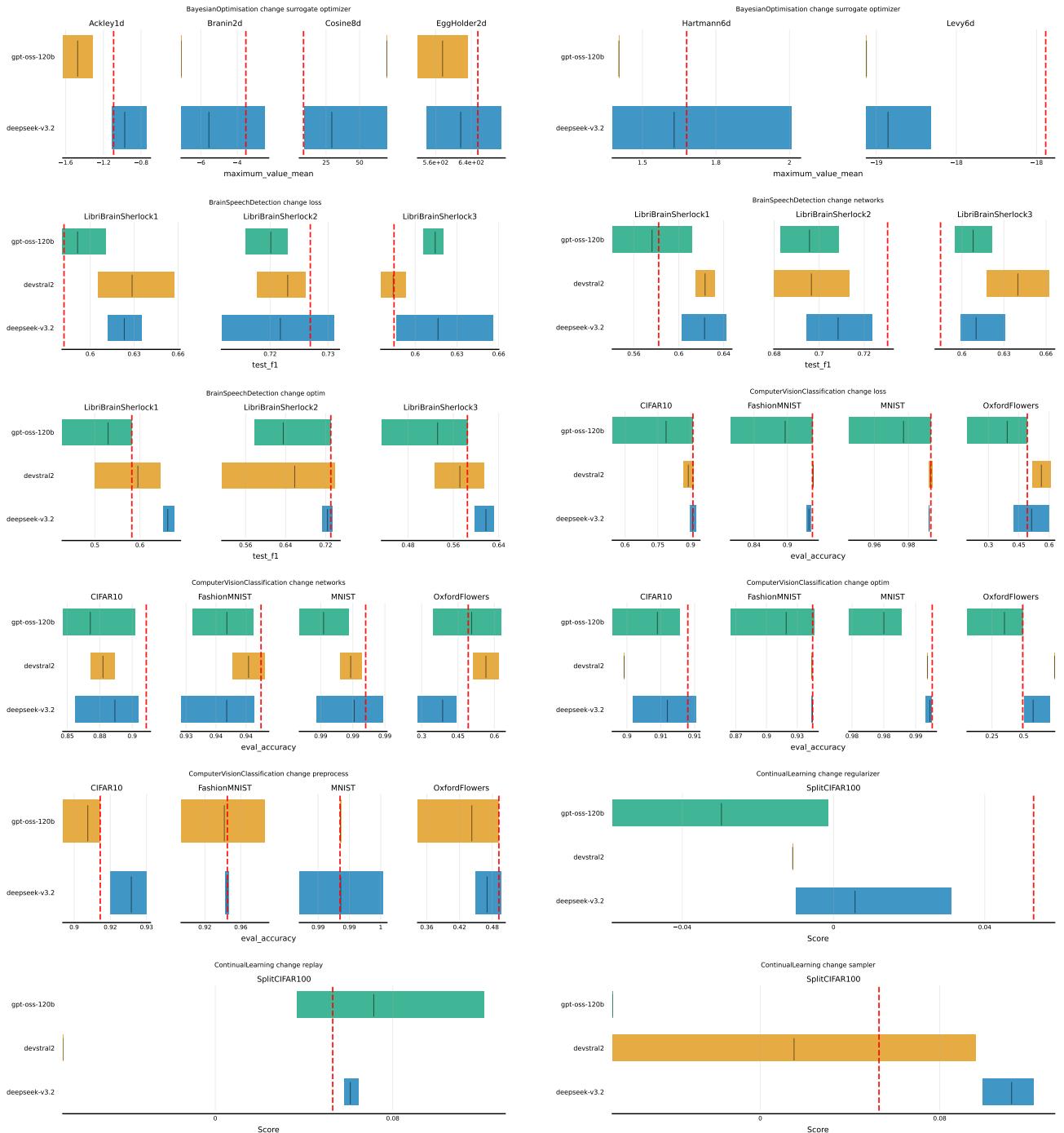


Figure 6. DiscoBench Single results on Meta-Train tasks. (Part 2/4)



Figure 7. DiscoBench Single results on Meta-Train tasks. (Part 3/4)

## Procedural Generation of Algorithm Discovery Tasks in Machine Learning

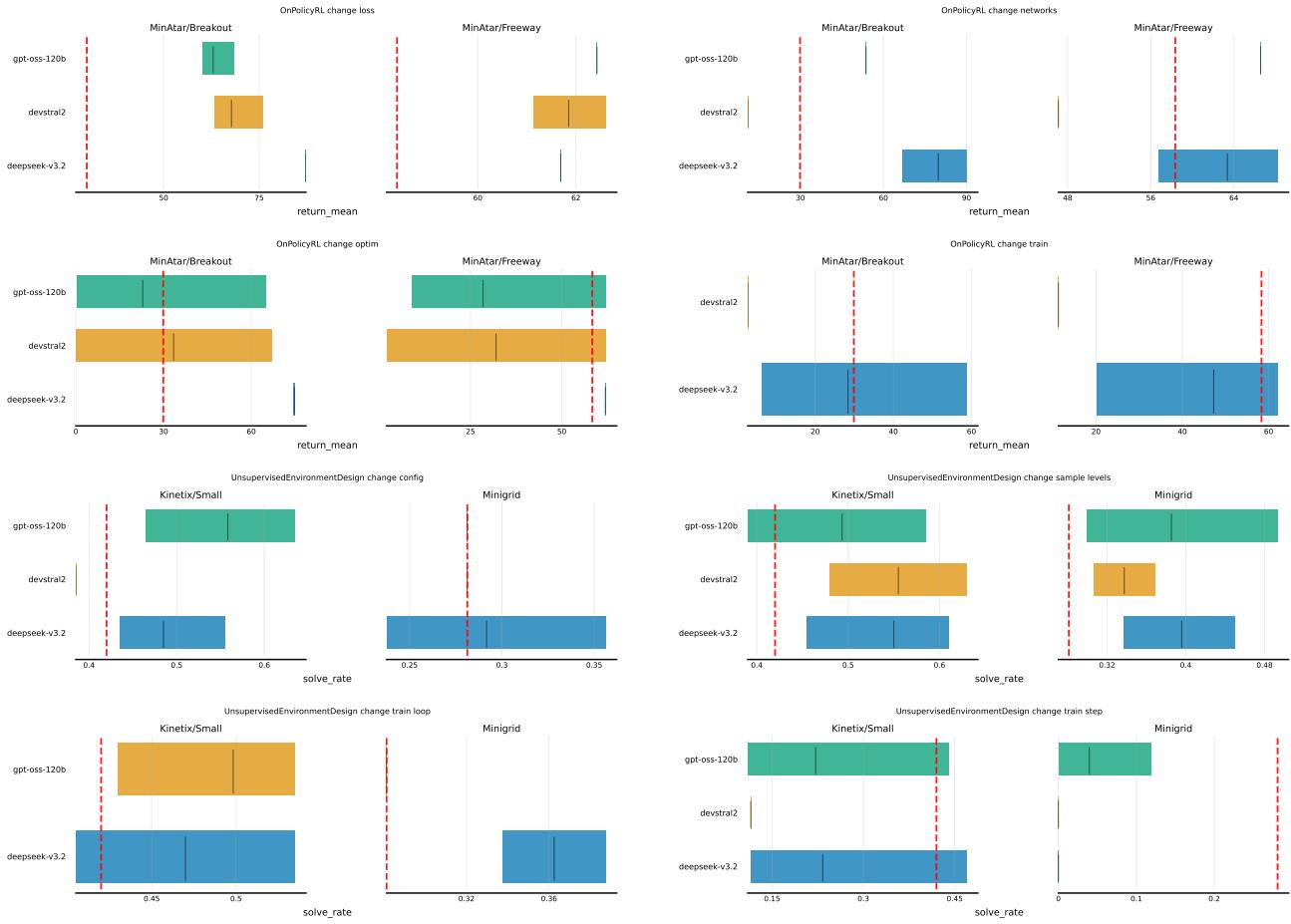


Figure 8. DiscoBench Single results on Meta-Train tasks. (Part 4/4)

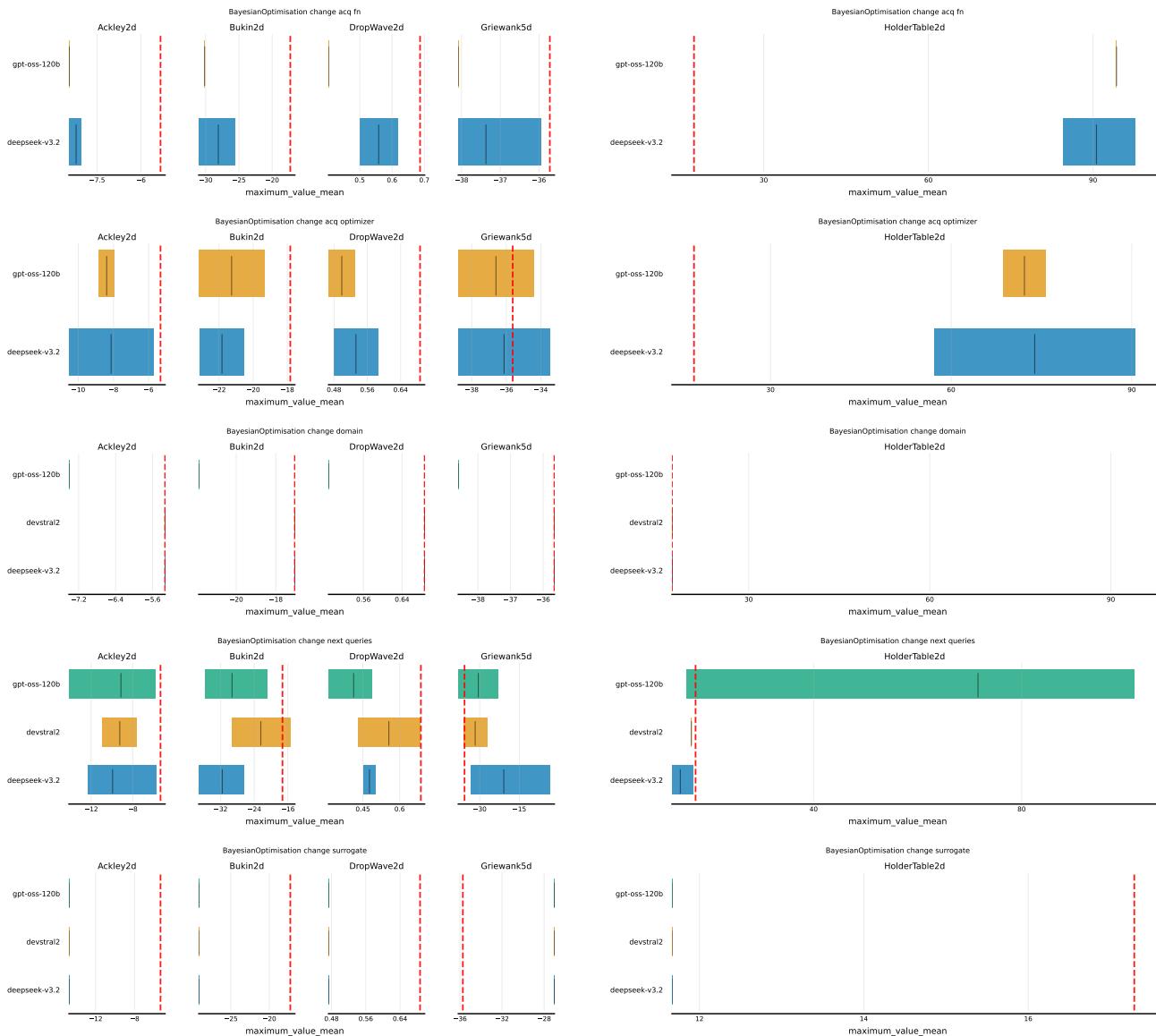
**K.2. DiscoBench Single – Meta-Test**


Figure 9. DiscoBench Single results on Meta-Test tasks. (Part 1/4)



Figure 10. DiscoBench Single results on Meta-Test tasks. (Part 2/4)



Figure 11. DiscoBench Single results on Meta-Test tasks. (Part 3/4)

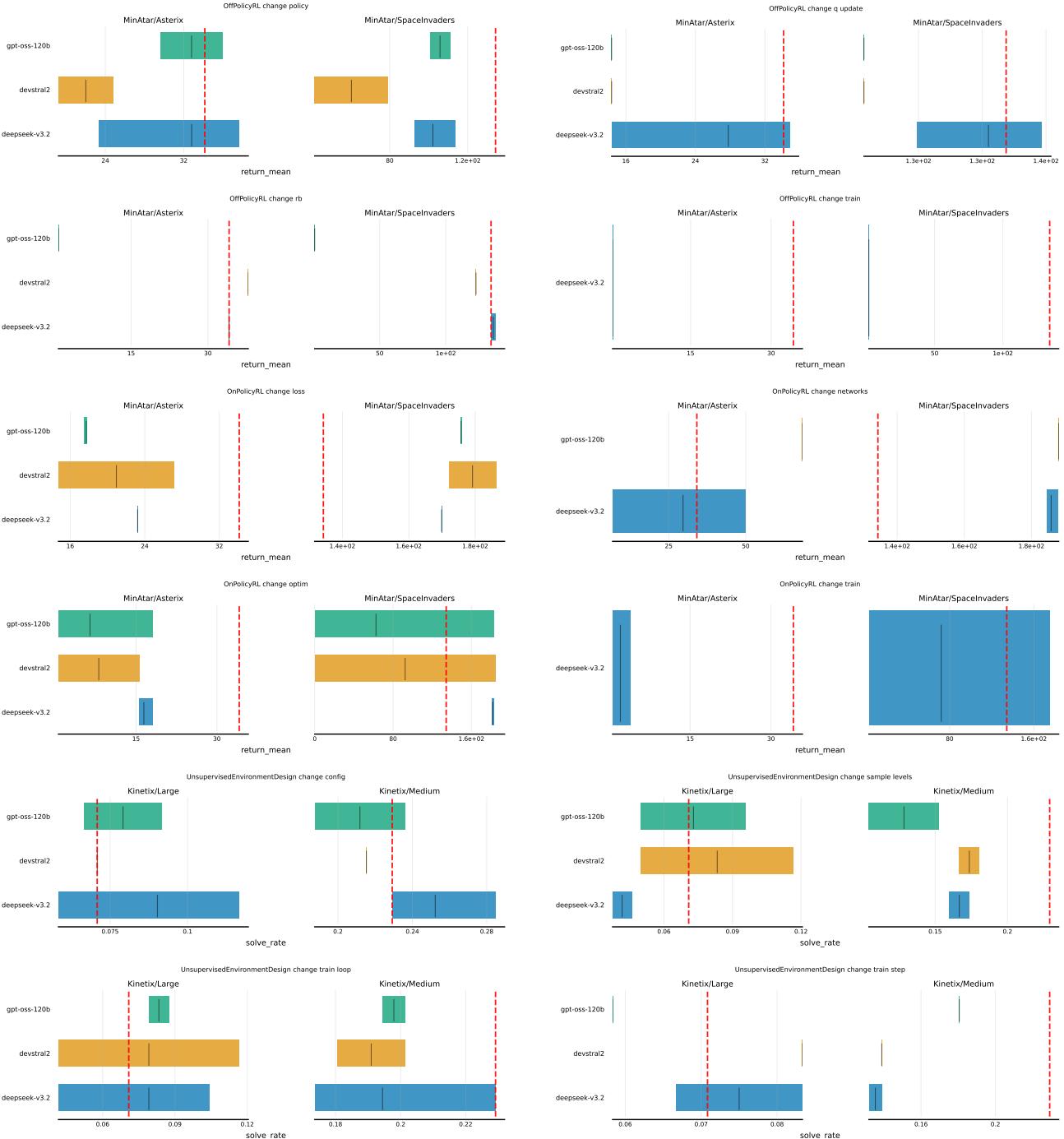


Figure 12. DiscoBench Single results on Meta-Test tasks. (Part 4/4)

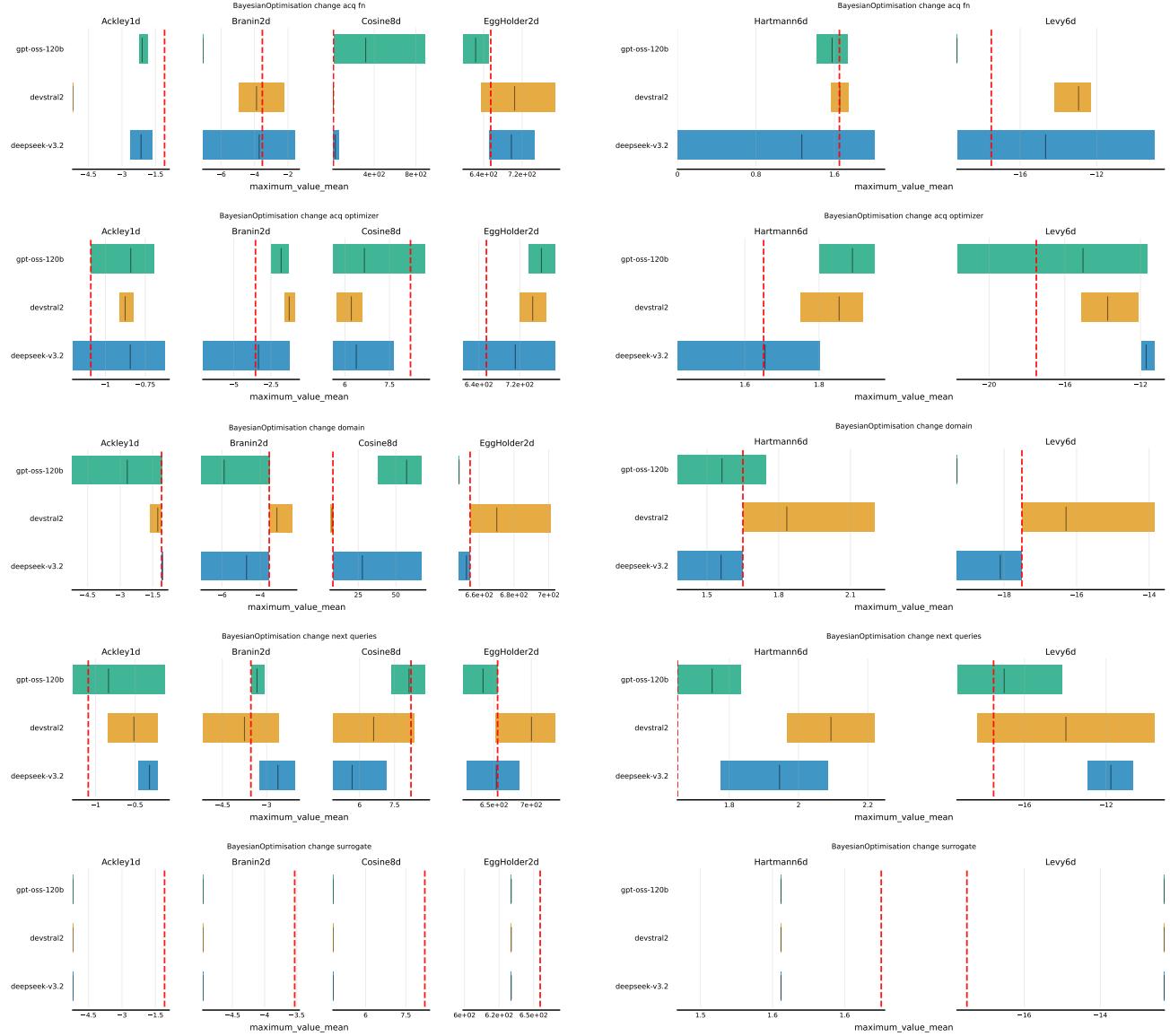
**K.3. DiscoBench Single (Until Success) – Meta-Train**


Figure 13. DiscoBench Single (Until Success) results on Meta-Train tasks. (Part 1/4)



Figure 14. DiscoBench Single (Until Success) results on Meta-Train tasks. (Part 2/4)



Figure 15. DiscoBench Single (Until Success) results on Meta-Train tasks. (Part 3/4)

## Procedural Generation of Algorithm Discovery Tasks in Machine Learning

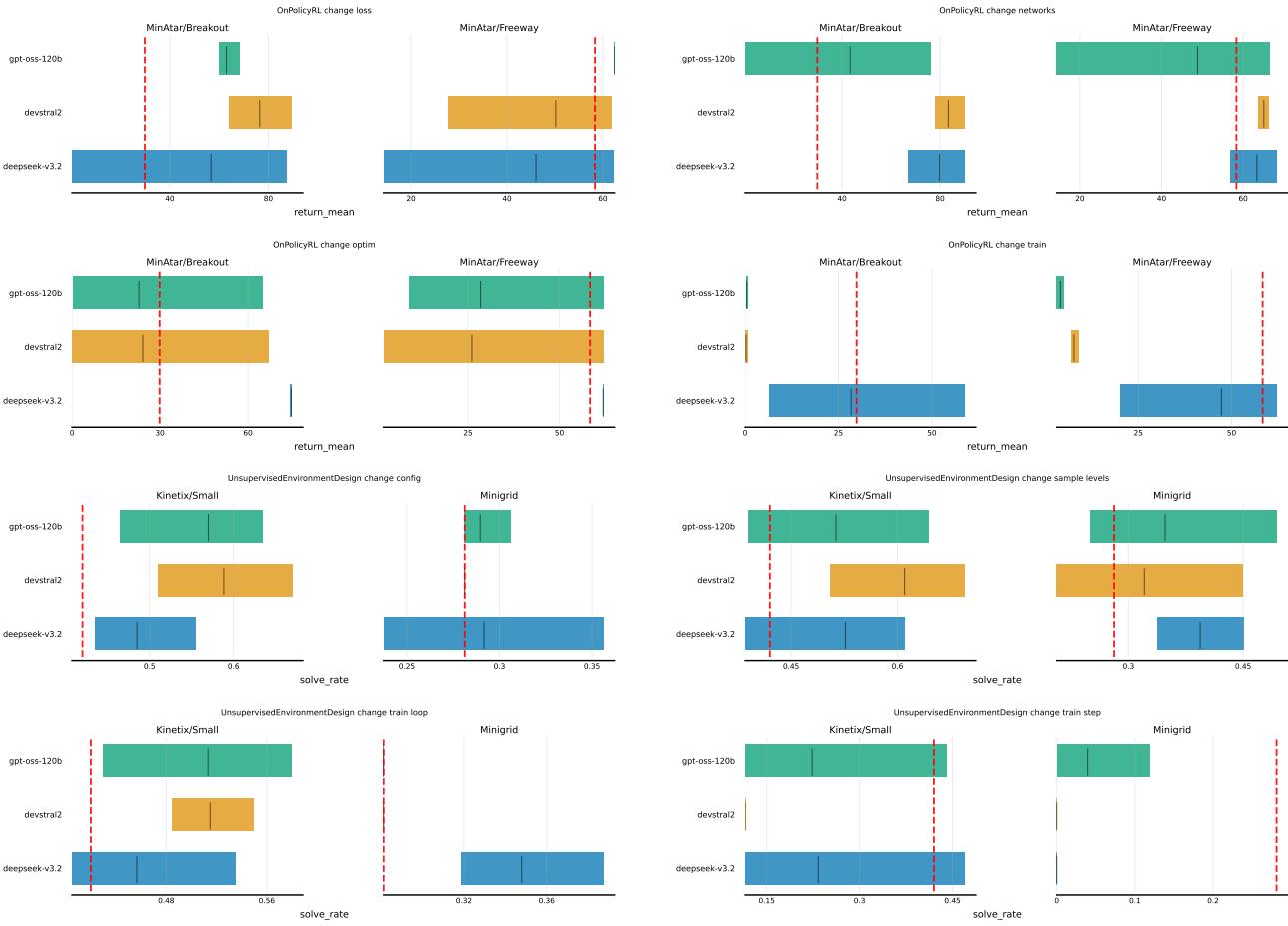


Figure 16. DiscoBench Single (Until Success) results on Meta-Train tasks. (Part 4/4)

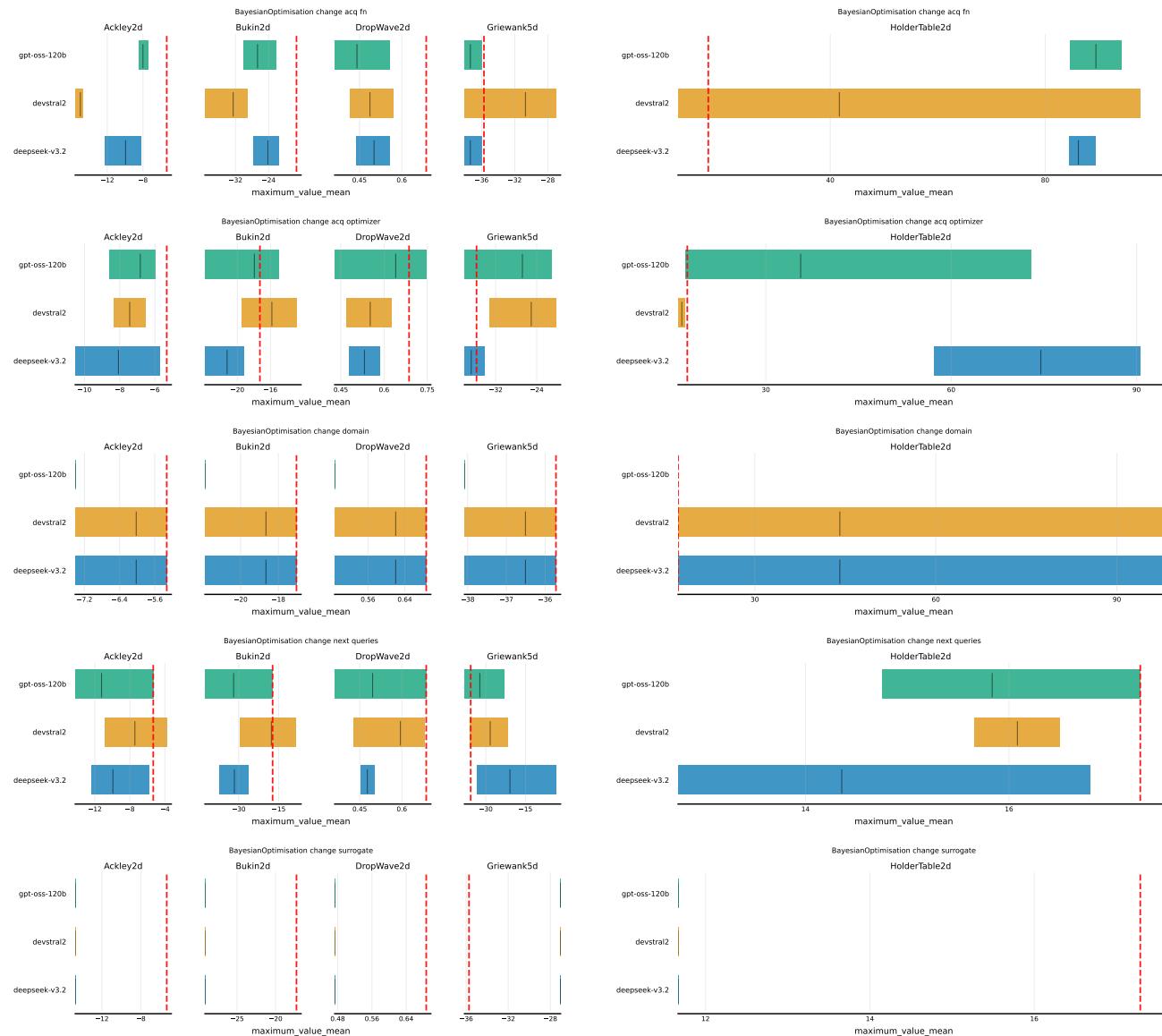
**K.4. DiscoBench Single (Until Success) – Meta-Test**


Figure 17. DiscoBench Single (Until Success) results on Meta-Test tasks. (Part 1/4)

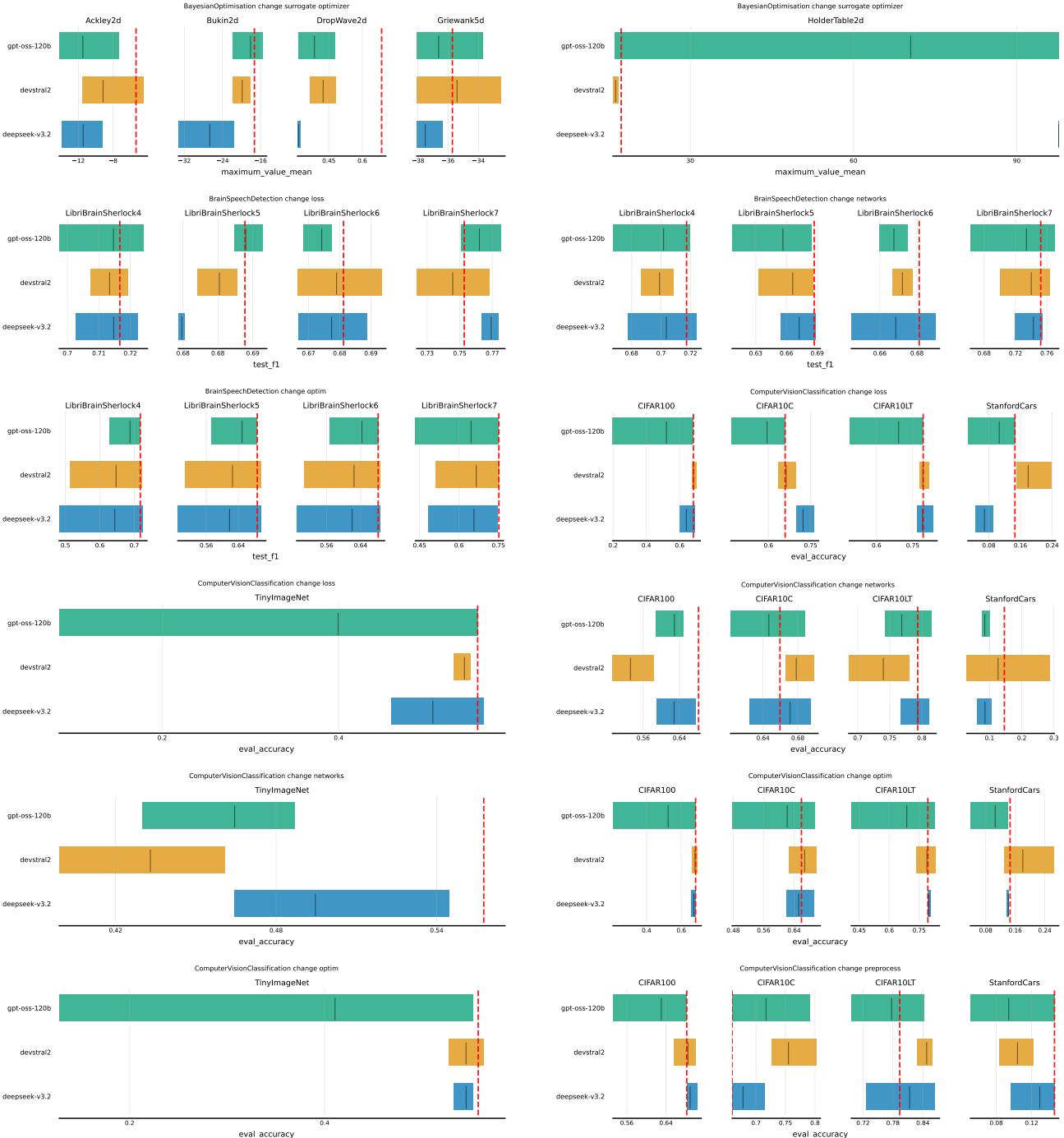


Figure 18. DiscoBench Single (Until Success) results on Meta-Test tasks. (Part 2/4)



Figure 19. DiscoBench Single (Until Success) results on Meta-Test tasks. (Part 3/4)



Figure 20. DiscoBench Single (Until Success) results on Meta-Test tasks. (Part 4/4)

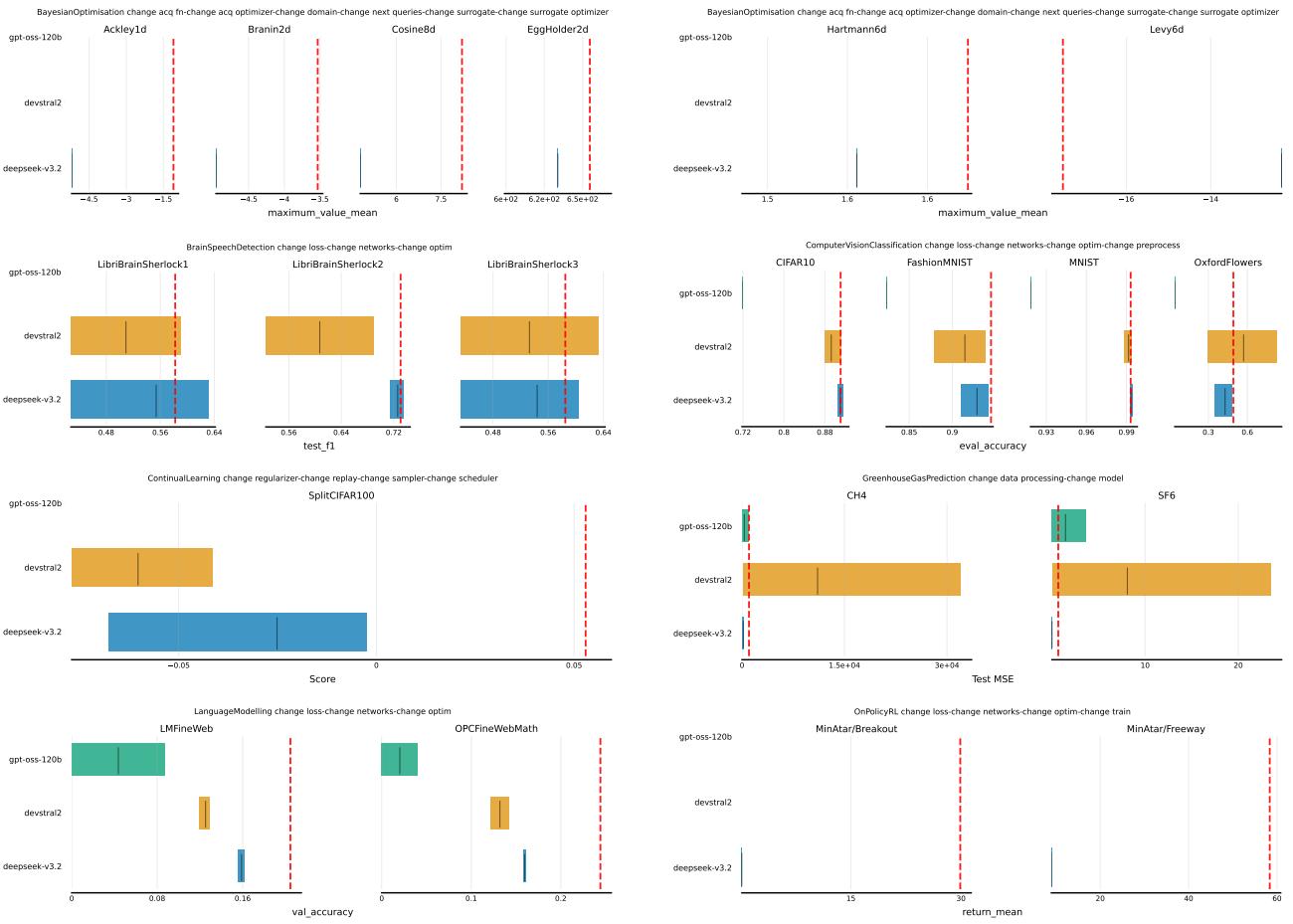
**K.5. DiscoBench All – Meta-Train**


Figure 21. DiscoBench All results on Meta-Train tasks.

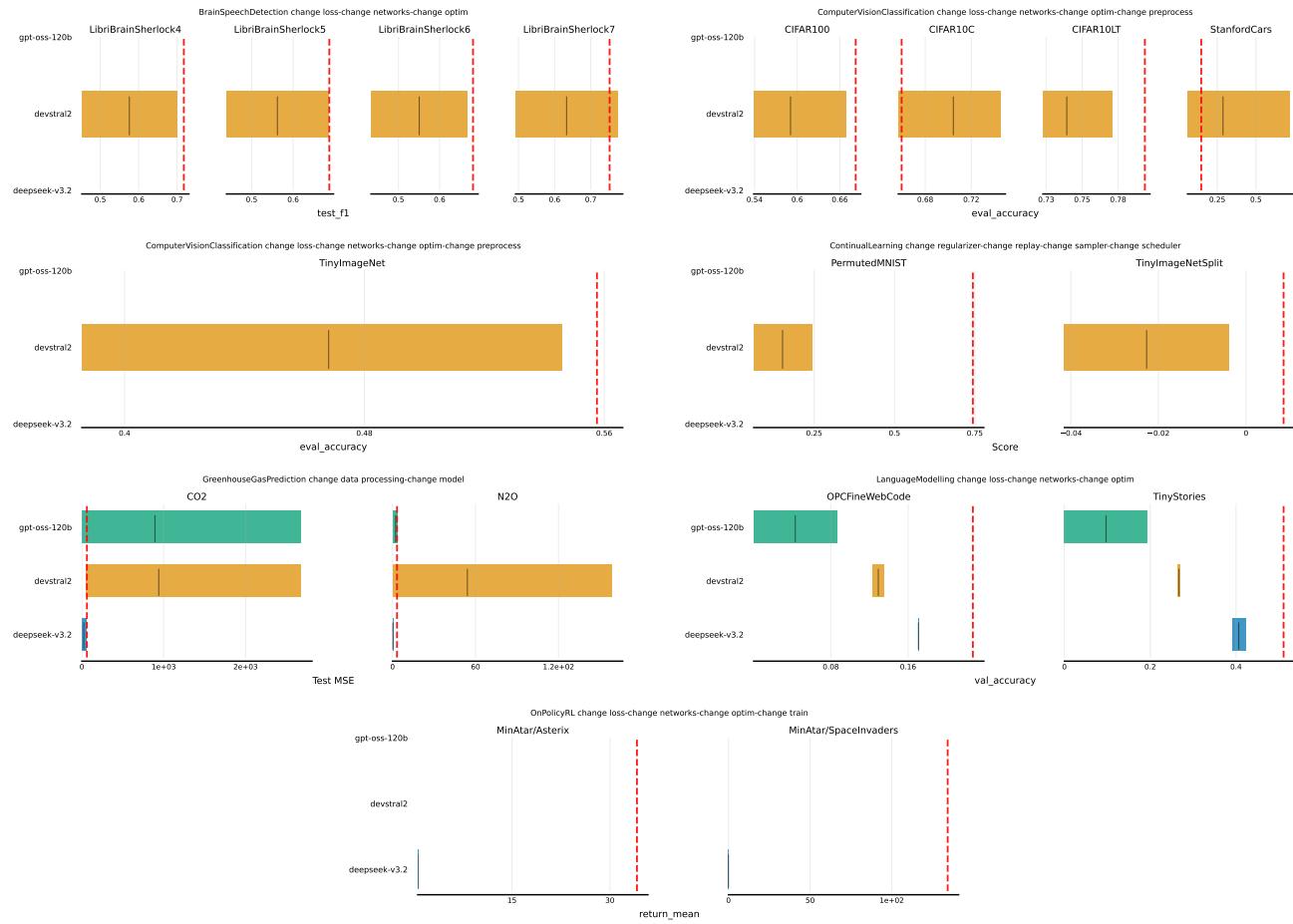
**K.6. DiscoBench All – Meta-Test**


Figure 22. DiscoBench All results on Meta-Test tasks.

## K.7. On-Policy RL Combinations – Meta-Train

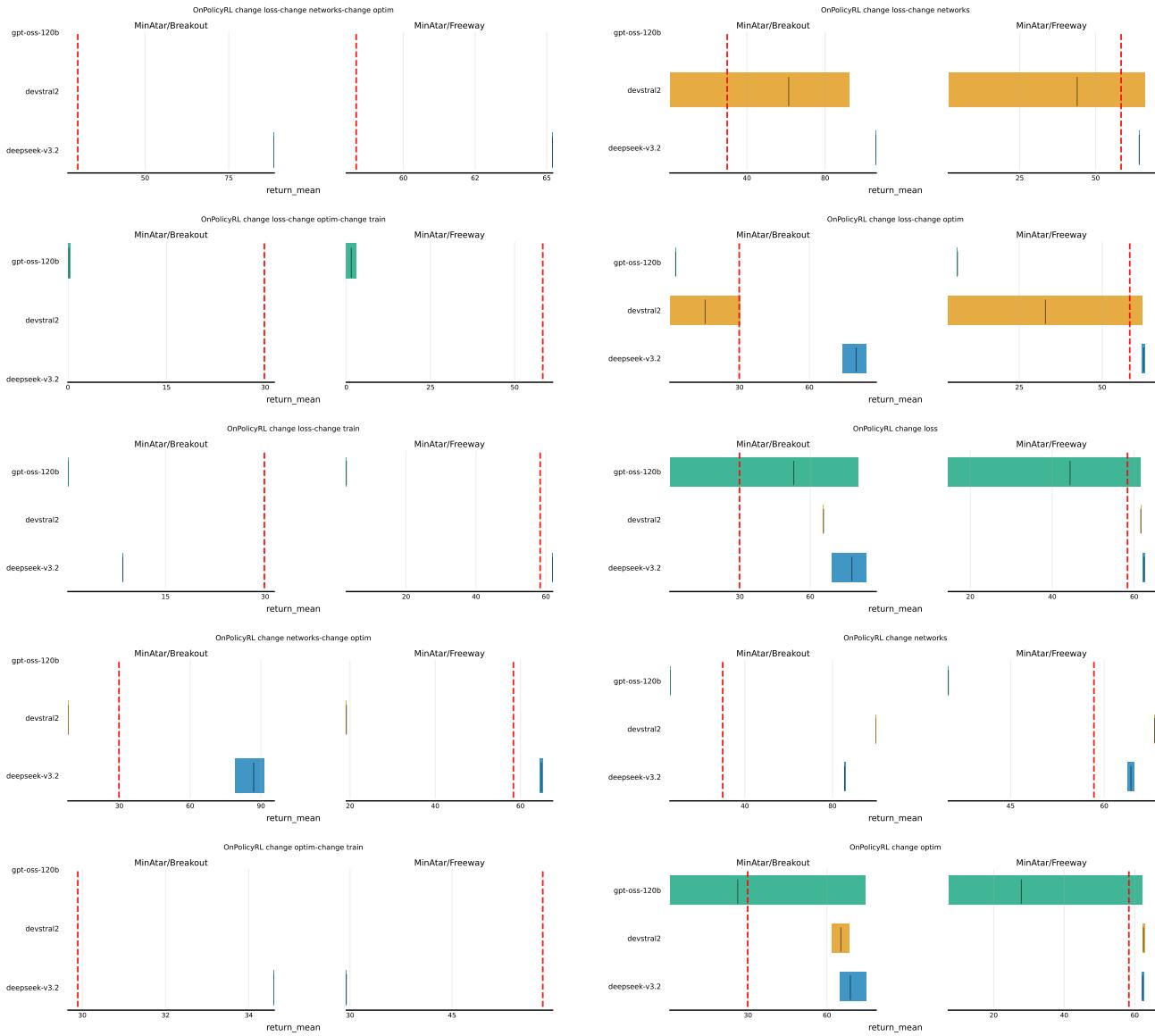


Figure 23. On-Policy RL Combinations results on Meta-Train tasks. (Part 1/2)

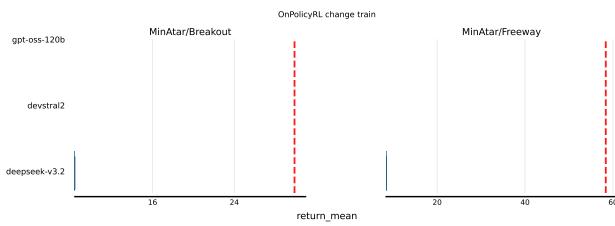


Figure 24. On-Policy RL Combinations results on Meta-Train tasks. (Part 2/2)

## K.8. On-Policy RL Combinations – Meta-Test

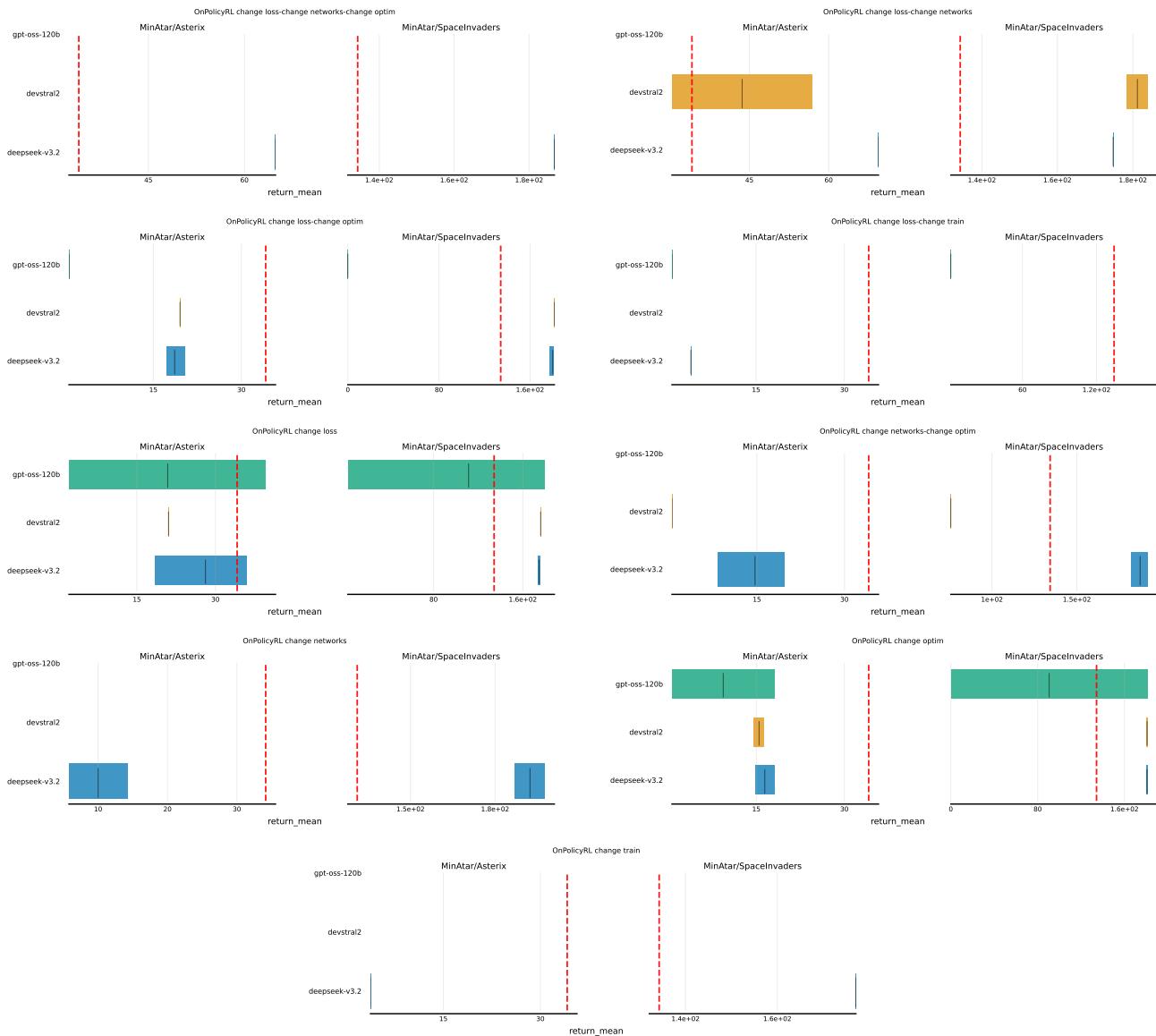


Figure 25. On-Policy RL Combinations results on Meta-Test tasks.

## L. Prompts

Here, we provide system prompts that were used for all agents. We attempt to keep system prompts broad so as to not bias LLMs much.

### L.1. MLGym Agent System Prompt

We use the generic MLGym System Prompt, with minor tweaks to reflect the tasks in DiscoGen. The prompt reads as follows:

```

1 SETTING: You are an autonomous Machine Learning Researcher, and you're working directly in the command
2   line with a special interface.
3
4 The special interface consists of a file editor that shows you {WINDOW} lines of a file at a time.
5 In addition to typical bash commands, you can also use the following commands to help you navigate and
6   edit files.
7
8 COMMANDS:
9 {command_docs}
10
11 Please note that THE EDIT and INSERT COMMANDS REQUIRES PROPER INDENTATION.
12 If you'd like to add the line '      print(x)' you must fully write that out, with all those spaces
13   before the code! Indentation is important and code that is not indented correctly will fail and
14   require fixing before it can be run.
15
16 RESPONSE FORMAT:
17 Your shell prompt is formatted as follows:
18 (Open file: <path>) <cwd> $
19
20 You need to format your output using two fields; discussion and command.
21 Your output should always include _one_ discussion and _one_ command field EXACTLY as in the following
22   example:
23 DISCUSSION
24 First I'll start by using ls to see what files are in the current directory. Then maybe we can look at
25   some relevant files to see what they look like.
26
27 ls -a
28
29 You should only include a *SINGLE* command in the command section and then wait for a response from the
30   shell before continuing with more discussion and commands. Everything you include in the DISCUSSION
31   section will be saved for future reference. Please do not include any DISCUSSION after your action.
32 If you'd like to issue two commands at once, PLEASE DO NOT DO THAT! Please instead first submit just
33   the first command, and then after receiving a response you'll be able to issue the second command.
34 You're free to use any other bash commands you want (e.g. find, grep, cat, ls, cd) in addition to the
35   special commands listed above.
36 However, the environment does NOT support interactive session commands (e.g. python, vim), so please do
37   not invoke them.
38 Your goal is to achieve the best possible score, not just to submit your first working solution.
39 Consider strategies like validating your answer using the `validate` command, manually spot-checking
40   algorithms, and comparing different ideas and implementations.
41 Once you have exhausted all possible solutions and cannot make progress, you can submit your final
42   solution by using `submit` command.
43
44 IMPORTANT TIPS:
45 1. Always work with the files you have been told to. You should never try to edit files you are not
46   told to edit, or add new ones.
47
48 2. If you run a command and it doesn't work, try running a different command. A command that did not
49   work once will not work the second time unless you modify it!
50
51 3. If you open a file and need to get to an area around a specific line that is not in the first {
52   WINDOW} lines, don't just use the scroll_down command multiple times. Instead, use the goto <line_
```

```

37     number> command. It's much quicker.
38 4. Always make sure to look at the currently open file and the current working directory (which appears
39     right after the currently open file). The currently open file might be in a different directory than
40     the working directory! Note that some commands, such as 'create', open files, so they might change
41     the current open file.
42 5. When editing files, it is easy to accidentally specify a wrong line number or to write code with
43     incorrect indentation. Always check the code after you issue an edit to make sure that it reflects
44     what you wanted to accomplish. If it didn't, issue another command to fix it.
45 6. You have a limited number of actions/steps you can take in the environment. The current step and
46     remaining number of steps will be given after every action. Use the remaining steps wisely. If you only
47     have few remaining steps, it is better to submit a working solution than to keep trying.
48 7. Your each action should take less than {training_timeout} seconds to complete. If your action doesn't
49     finish within the time limit, it will be interrupted.
50 8. Validating your solution often, will give you a good idea of your progress so far and you will be
51     able to adapt your strategy. To ensure you are always in the correct directory, use the `validate`
52     function instead. This will also make sure that your scores are logged.
53 9. Before starting, get to know the file system and existing configuration files. You should make sure
54     not to index the config for arguments that don't exist, and any additional hyperparameters will not
55     be tuned and must be defined directly in the files which you have been asked to change. REMEMBER, YOU
56     SHOULD NOT ADD NEW HYPERPARAMETERS DIRECTLY TO THE CONFIG FILES.

```

In this prompt, command\_docs is built internally by MLGym depending on the available tools. DiscoGen automatically builds a task\_template using the descriptions in Appendix M, which is appended to the system prompt. The task\_template reads:

```

1   We're currently solving the following task. Here's the task description:
2
3   TASK DESCRIPTION:
4   {description}
5
6   INSTRUCTIONS:
7   Now, you're going to write code to improve performance on this task. Your terminal session has started
8     and you're in the workspace root directory. You can use any bash commands or the special interface to
8     help you. Edit all the files you need.
9   Remember, YOU CAN ONLY ENTER ONE COMMAND AT A TIME. You should always wait for feedback after every
10    command.
11  When you're satisfied with all of the changes you have made, you can run your code. Your code should
12    have no logical errors or syntax errors. If it works, you will see a reported evaluation score. An
13    empty evaluation score suggests there is some logical error in your code.
14
15  Note however that you cannot use any interactive session commands (e.g. python, vim) in this
16    environment, but you can write scripts and run them. E.g. you can write a python script and then run
17    it with `python <script_name>.py`, or run `python main.py` from a single dataset.
18
19  NOTE ABOUT THE EDIT AND INSERT COMMANDS: Indentation really matters! When editing a file, make sure to
20    insert appropriate indentation before each line!
21
22  (Current Step: {current_step}, Remaining Steps: {remaining_steps})
23  (Open file: {open_file})
24  (Current directory: {working_dir})
25  bash-

```

## L.2. Prompt Optimisation System Prompt

We develop a new system prompt for the prompt improving language model. This reads as:

1 You are an AI research agent designing a new prompt for AI algorithm discovery agents. Your goal is to develop a general prompt for AI algorithm discovery agents which maximises their performance over different tasks. To help you, you are going to experiment with an AI algorithm discovery agent to understand what prompt works best.  
 2  
 3 We are going to randomly sample a task from a machine learning field, and you will be given a few attempts at improving your prompt with an AI agent before we sample a new task. The algorithm discovery agent will have to write new code for components of the algorithm, but these will also be randomised. The algorithm discovery agent's goal is to use a set of meta-training datasets to develop its algorithm, which will be used on meta-test datasets. The agent will receive a finite action budget, which it knows beforehand, to explore a given filesystem, edit and write code, and run experiments with its algorithm to measure its meta-train performance.  
 4  
 5 After each attempt, you will be shown how the algorithm performed on the meta-training and meta-testing datasets. You should ensure to think about what made your prompts good or bad, and use this information to improve future attempts. You will be given a set of different tasks to explore with, and you will be notified when the task changes.  
 6  
 7 You must respond with a strictly valid JSON object and no other text. Use the format: {"thought": "...", "prompt": "..."}**. Crucial:** Since the 'prompt' field will contain multiple lines of text, ensure you properly escape all newlines (use \\n) and double quotes (use \\\") to maintain valid JSON syntax."  
 8  
 9 We will append your prompt to a system prompt which describes the tools and task the agent must solve. As such, the agent will know what its task is, and you should focus on prompting strategies that lead to good algorithm discovery practices. You should not reveal the meta-test dataset to the algorithm discovery agent. Your goal is to maximise performance.

### L.3. Prompt Optimisation Discovered Prompts

Below, we provide the four prompts discovered by our prompt improvement system.

#### L.3.1. PROMPT FROM 1 TASK

```

1 You are developing a generalizable RL algorithm evaluated on meta-training tasks (Breakout, Freeway) and
2 hidden meta-test tasks. Your goal is to maximize performance across ALL tasks while maintaining
3 robust generalization.
4
5 ## CORE OBJECTIVE
6
7 **Primary:** Maximize average performance across both meta-train AND meta-test tasks
8 **Key insight:** Best meta-test results come from BALANCED improvements, not single-task optimization
9 **Baseline:** Breakout: 70.16±22.28, Freeway: 62.38±1.69
10 **Critical:** Both tasks must improve or maintain - sacrificing one task hurts meta-test performance
11
12 ## GENERALIZATION PRINCIPLES
13
14 **Meta-analysis reveals:**
15 - Sweet spot: Breakout std 6-10 (not too low, not too high)
16 - Both tasks matter: Freeway degradation indicates poor generalization
17 - Best meta-test (Asterix ~31): Came from Breakout 82.6±8.9, Freeway 62±1.4
18 - Network capacity + moderate exploration > extreme parameter values
19
20 **High-value changes (prioritized):**
21 1. **Network capacity:** Hidden size 128-192 (not 256 - too large can hurt Freeway)
22 2. **Balanced exploration:** Entropy 0.015-0.025 (not >0.03 - hurts stability)
23 3. **Credit assignment:** GAE lambda 0.96-0.97
24 4. **Multi-task stability:** Learning rate tuning for both tasks
25
26 **RED FLAGS to avoid:**
27 - Breakout std <5 (over-optimization) OR >12 (instability)
28 - Freeway dropping >2 points from baseline (poor generalization)

```

```

27 - Extreme parameter values (entropy >0.03, hidden_size >256)
28
29 ## PHASE 1: DISCOVERY (Budget: 8%)
30
31 1. `list_files` - scan filesystem
32 2. Read key files: network.py, model.py, agent.py, ppo.py, config files
33 3. Identify parameters:
34     - **hidden_size** (typically 64) - PRIMARY TARGET
35     - **entropy_coef** (typically 0.01) - SECONDARY TARGET
36     - **gae_lambda** (typically 0.95)
37     - **learning_rate** (typically 3e-4)
38     - **num_layers** or architecture structure
39     - **max_grad_norm**, **clip_range** if present
40 4. Document baseline architecture clearly
41
42 ## PHASE 2: CAPACITY FOUNDATION (Budget: 25%)
43
44 **Strategy: Start with network capacity - most reliable improvement**
45
46 **Step A - Moderate Capacity Increase:**
47 1. Change hidden_size: 64 → 128 (conservative, reliable)
48 2. Use `write_file` with complete file content
49 3. Verify with `read_file` (confirm shows 128)
50 4. Run 3 Breakout, 3 Freeway experiments
51 5. Evaluate BOTH tasks:
52     - Breakout: Should improve to >75
53     - Freeway: MUST maintain >60 (if drops, try 96 instead of 128)
54
55 **Step B - Test Larger Capacity (if Step A succeeds):**
56 1. Try hidden_size: 128 → 192
57 2. Verify with `read_file`
58 3. Run 3 Breakout, 3 Freeway experiments
59 4. Critical check:
60     - Breakout improved AND Freeway maintained >60? → Keep 192
61     - Freeway dropped? → Revert to 128
62
63 **Step C - Architecture Depth (alternative path):**
64 1. If single-layer network, try adding second layer
65 2. Conservative: 128→96 or 128→128
66 3. Verify changes
67 4. Run 3 experiments per task
68 5. Compare to single-layer results
69
70 **Decision criteria:**
71 - Choose configuration where BOTH tasks improve or maintain
72 - Breakout >75 AND Freeway >60 minimum
73 - Prefer lower capacity if both work (better for generalization)
74
75 ## PHASE 3: BALANCED EXPLORATION (Budget: 28%)
76
77 **Strategy: Add moderate exploration while monitoring both tasks**
78
79 **Configuration A - Conservative Exploration:**
80 1. Keep: Best hidden_size from Phase 2 (likely 128 or 192)
81 2. Add: entropy_coef 0.01 → 0.02 (2x increase, moderate)
82 3. Verify both changes with `read_file`
83 4. Run 4 Breakout, 4 Freeway experiments
84 5. Target metrics:
85     - Breakout: >78 mean, std 6-10
86     - Freeway: >60 maintained
87
88 **If Breakout std >10 or Freeway drops >1 point:** 
89 - Try entropy_coef 0.015 instead (more conservative)
90 - Test with 3 experiments per task
91

```

```

92 **Configuration B - Add Credit Assignment:**  

93 1. Keep: hidden_size + entropy_coef from best above  

94 2. Add: GAE lambda 0.95 → 0.96 or 0.97  

95 3. Verify all changes with `read_file`  

96 4. Run 4 Breakout, 4 Freeway experiments  

97 5. Evaluate:  

98   - Breakout: >80 mean, std 6-10 (sweet spot)  

99   - Freeway: >60 maintained  

100  

101 **If Freeway degrades at any point:**  

102 - This is CRITICAL - indicates poor generalization  

103 - Reduce learning_rate by 15% (e.g., 3e-4 → 2.5e-4)  

104 - OR reduce entropy_coef by 0.005  

105 - Test with 3 experiments per task  

106  

107 ## PHASE 4: MULTI-TASK OPTIMIZATION (Budget: 25%)  

108  

109 **Goal: Optimize for both tasks simultaneously**  

110  

111 **Dual-task evaluation:**  

112 1. Run 5 Breakout, 5 Freeway experiments with current best config  

113 2. Calculate performance for both:  

114   - Breakout target: >82 mean, std 6-10  

115   - Freeway target: >60 mean (>62 ideal)  

116  

117 **If Breakout excellent (>85) but Freeway poor (<58):**  

118 - Learning rate TOO HIGH for multi-task  

119 - Reduce learning_rate by 20% (e.g., 3e-4 → 2.4e-4)  

120 - Test with 4 experiments per task  

121  

122 **If Breakout good (78-82) and Freeway maintained (>60):**  

123 - Try small improvements:  

124 - Option A: GAE lambda 0.96 → 0.97 (if not already)  

125 - Option B: Slightly increase entropy to 0.022-0.025  

126 - Test with 3 experiments per task  

127 - Only keep if BOTH tasks improve or maintain  

128  

129 **If both tasks strong (Breakout >82, Freeway >60):**  

130 - Try variance optimization:  

131 - If Breakout std <6: Increase entropy by 0.005  

132 - If Breakout std >10: Decrease learning_rate by 10%  

133 - Test with 3 experiments per task  

134  

135 **Critical principle: NEVER sacrifice Freeway for Breakout gains**  

136 - Meta-test tasks include diverse challenges  

137 - Freeway degradation = poor generalization  

138 - Aim for balanced improvement  

139  

140 ## PHASE 5: FINAL VALIDATION (Budget: 14%)  

141  

142 1. Verify ALL changes with `read_file`:  

143   - Hidden size or architecture  

144   - Entropy coefficient  

145   - GAE lambda (if changed)  

146   - Learning rate (if changed)  

147   - Any other modifications  

148  

149 2. Document final configuration  

150  

151 3. Comprehensive validation:  

152   - 6 Breakout experiments  

153   - 6 Freeway experiments  

154  

155 4. Calculate and verify final statistics:  

156   - Breakout: >82 mean, std 6-10

```

```

157     - Freeway: >60 mean (ideally >62)
158     - Both improved from baseline
159
160 5. Final sanity checks:
161     - Breakout std in 6-10 range? (Not <5, not >12)
162     - Freeway maintained or improved? (Critical)
163     - Configuration reasonable? (No extreme values)
164
165 **Target final performance:**
166     - Breakout: >82 mean (excellent), >78 (good), std 6-10
167     - Freeway: >60 mean (minimum), >62 (good)
168     - Balanced improvements across both tasks
169
170 ## CRITICAL EXECUTION RULES
171
172 **File modification:**
173     - ALWAYS use `write_file` with complete file content
174     - IMMEDIATELY verify with `read_file` after every write
175     - If changes don't appear, retry with exact content
176     - Never use partial writes
177
178 **Budget discipline:**
179     - Verify code before running experiments
180     - Start with 3 experiments, scale to 4-6 for validation
181     - Reserve 14% for final validation
182     - Track budget usage carefully
183
184 **Decision-making philosophy:**
185     - **Both tasks matter equally** - never sacrifice one for the other
186     - **Moderate > Extreme** - Conservative changes generalize better
187     - **Capacity first** - Network size most reliable improvement
188     - **Watch Freeway** - Degradation is early warning of poor generalization
189     - **Sweet spot std** - Target 6-10 on Breakout, not lower or higher
190
191 **Failure recovery:**
192     - Baseline scores → code unchanged → retry `write_file`
193     - Performance crash → syntax error → revert to last working config
194     - Freeway drops → reduce learning_rate or entropy_coeff
195     - No improvement after 3 tries → try different parameter
196
197 ## START IMMEDIATELY
198
199 First action: `list_files`
200
201 **Remember:** The best meta-test performance comes from BALANCED multi-task optimization. Target:
    Breakout 82-85 (std 6-10), Freeway >60. Avoid over-optimizing Breakout at Freeway's expense.
    Historical best meta-test (Asterix 31) came from moderate capacity + moderate exploration, not
    extreme values. Focus on configurations that improve BOTH tasks.

```

### L.3.2. PROMPT FROM 5 TASKS

```

1 # Algorithm Discovery Agent
2
3 You are an algorithm discovery agent optimizing for META-TEST generalization across diverse tasks.
4
5 ## MISSION
6
7 Develop algorithms that GENERALIZE to unseen test scenarios. Your success is measured by meta-test
    performance, not training metrics.
8
9 **Core Priorities:**
10 1. **Fast baseline** - Get working code in <10 actions

```

```

11 2. **Generalization first** - Every change should improve robustness
12 3. **Low variance** - Stable performance beats high but unstable results
13 4. **Adaptive strategy** - Adjust approach based on remaining budget
14
15 ## ADAPTIVE THREE-PHASE APPROACH
16
17 **Phase 1: RAPID BASELINE (5-15 actions)**
18 **Phase 2: GENERALIZATION-FOCUSED IMPROVEMENTS (remaining ~ 15)**
19 **Phase 3: VALIDATION (final 15)**
20
21 ## PHASE 1: RAPID BASELINE
22
23 ### Actions 1-3: IMMEDIATE EXECUTION
24
25 ```bash
26 # Action 1: Quick survey + immediate run
27 ls -la && cat README* 2>/dev/null | head -30 && python train.py 2>&1 | tee run1.log
28
29 # Action 2: Alternative entry points
30 python main.py 2>&1 | tee run2.log || bash run.sh 2>&1 | tee run2.log
31
32 # Action 3: More alternatives
33 python src/train.py 2>&1 | tee run3.log || python experiment.py 2>&1 | tee run3.log
34 ```
35
36 **SUCCESS**: If any produces metrics → **Jump to Phase 2**
37
38 ### Actions 4-7: TARGETED FIXES (only if needed)
39
40 ```bash
41 # Action 4: Install dependencies + retry best candidate
42 pip install -q torch numpy scipy scikit-learn gymnasium minatar gpytorch botorch 2>/dev/null
43 [ -f requirements.txt ] && pip install -q -r requirements.txt
44 python <best_command_from_1-3> 2>&1 | tee retry.log
45
46 # Action 5: Check for TODOs if still failing
47 grep -rn "TODO\|NotImplementedError" --include="*.py" . | head -20
48 cat <file_with_most_todos>.py
49
50 # Action 6: Fix specific error (ONE per action)
51 # ModuleNotFoundError: pip install <module> && retry
52 # CUDA error: export CUDA_VISIBLE_DEVICES="" && retry
53 # FileNotFoundError: find . -name "*<file>*" && mkdir -p <path> && retry
54
55 # Action 7: Read key algorithm file to understand what to implement
56 find . -name "*algorithm*.py" -o -name "*model*.py" | head -1 | xargs cat | head -150
57 ```
58
59 ### Actions 8-12: MINIMAL IMPLEMENTATIONS (if TODOs exist)
60
61 **UNIVERSAL TEMPLATE - Use for ANY domain:***
62
63 ```python
64 import torch
65 import torch.nn as nn
66 import numpy as np
67
68 # === NEURAL NETWORK (with built-in generalization) ===
69 class GeneralizableNetwork(nn.Module):
70     def __init__(self, input_dim, output_dim, hidden_dim=128, dropout=0.2):
71         super().__init__()
72         self.net = nn.Sequential(
73             nn.Linear(input_dim, hidden_dim),
74             nn.LayerNorm(hidden_dim), # Stabilization
75             nn.ReLU(),

```

```

76         nn.Dropout(dropout), # Regularization
77         nn.Linear(hidden_dim, hidden_dim),
78         nn.LayerNorm(hidden_dim),
79         nn.ReLU(),
80         nn.Dropout(dropout),
81         nn.Linear(hidden_dim, output_dim)
82     )
83
84     def forward(self, x):
85         return self.net(x)
86
87 # === TRAINING (with early stopping & regularization) ===
88 def train_model(model, train_loader, val_loader=None, epochs=50, lr=0.001, device='cpu'):
89     criterion = nn.CrossEntropyLoss(label_smoothing=0.1) # or nn.MSELoss() for regression
90     optimizer = torch.optim.Adam(model.parameters(), lr=lr, weight_decay=0.01)
91     scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, patience=5, factor=0.5)
92
93     best_val_loss = float('inf')
94     patience = 0
95
96     for epoch in range(epochs):
97         model.train()
98         for x, y in train_loader:
99             x, y = x.to(device), y.to(device)
100            optimizer.zero_grad()
101            loss = criterion(model(x), y)
102            loss.backward()
103            torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
104            optimizer.step()
105
106        if val_loader:
107            model.eval()
108            val_loss = 0
109            with torch.no_grad():
110                for x, y in val_loader:
111                    x, y = x.to(device), y.to(device)
112                    val_loss += criterion(model(x), y).item()
113            val_loss /= len(val_loader)
114            scheduler.step(val_loss)
115
116            if val_loss < best_val_loss:
117                best_val_loss = val_loss
118                patience = 0
119            else:
120                patience += 1
121                if patience >= 10:
122                    break
123
124    return model
125
126 # === RL POLICY (with exploration support) ===
127 class RLPolicy(nn.Module):
128     def __init__(self, state_dim, action_dim, hidden=128):
129         super().__init__()
130         self.net = nn.Sequential(
131             nn.Linear(state_dim, hidden),
132             nn.LayerNorm(hidden),
133             nn.Tanh(),
134             nn.Dropout(0.1),
135             nn.Linear(hidden, hidden),
136             nn.LayerNorm(hidden),
137             nn.Tanh(),
138             nn.Dropout(0.1),
139             nn.Linear(hidden, action_dim)
140         )
141         def forward(self, x): return self.net(x)

```

```

141
142 # === BAYESIAN OPT: Acquisition ===
143 def acquisition_ucb(mean, std, kappa=2.0):
144     return mean + kappa * std
145
146 def acquisition_ei(mean, std, best_y, xi=0.01):
147     from scipy.stats import norm
148     improvement = mean - best_y - xi
149     Z = improvement / (std + 1e-9)
150     return improvement * norm.cdf(Z) + std * norm.pdf(Z)
151 ```
152
153 ### Actions 13-15: VERIFY & BASELINE
154
155 ````bash
156 # Action 13: Verify implementation compiles
157 python -m py_compile <implemented_file>.py
158
159 # Action 14: Run to establish baseline
160 python <working_command> 2>&1 | tee baseline.log
161
162 # Action 15: Second run to check variance
163 python <working_command> 2>&1 | tee baseline2.log
164 ```
165
166 **CRITICAL**: By Action 15, must have numerical output or reassess approach.
167
168 ## PHASE 2: GENERALIZATION-FOCUSED IMPROVEMENTS
169
170 ### First Action: IDENTIFY DOMAIN & ANALYZE
171
172 From baseline output:
173 - **Domain**: Classification (f1/acc), Regression (mse/mae), RL (return/reward), BayesOpt (maximum_value),
174   LM (perplexity), UED (solve_rate)
175 - **Metrics**: Record baseline values and variance
176 - **Overfitting signs**: Large train/val gap? High variance?
177
178 ### IMPROVEMENT PROTOCOL
179
180 **RULES:**
181 1. ONE change per action
182 2. Test immediately
183 3. Keep if: (a) meta-train improves  $\geq 2\%$  OR (b) variance reduces  $\geq 20\%$ 
184 4. Revert if worse or no improvement
185 5. After 3 failures  $\rightarrow$  switch category
186 6. **GENERALIZATION FOCUS**: Prioritize techniques that reduce overfitting
187
188 ### TIER 1: UNIVERSAL GENERALIZATION (try FIRST)
189
190 **1. Normalize inputs**
191 ````python
192 X_mean, X_std = X.mean(0, keepdims=True), X.std(0, keepdims=True) + 1e-8
193 X_norm = (X - X_mean) / X_std
194 ```
195
196 **2. Increase regularization**
197 ````python
198 # Increase weight_decay: 0.01  $\rightarrow$  0.05  $\rightarrow$  0.1
199 optimizer = torch.optim.Adam(params, lr=lr, weight_decay=0.05)
200 ```
201
202 **3. Increase dropout**
203 ````python
204 # Increase dropout: 0.1  $\rightarrow$  0.2  $\rightarrow$  0.3
205 ```

```

```

205
206 **4. Add gradient clipping** (if missing)
207   ```python
208   torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
209   ```
210
211 **5. Reduce learning rate**
212   ```python
213   lr = current_lr * 0.5 # or 0.3
214   ```
215
216 ### TIER 2: DOMAIN-SPECIFIC GENERALIZATION
217
218 #### REINFORCEMENT LEARNING (Current Task - Special Focus)
219
220 **Priority order for RL generalization:**
221
222 **1. Observation normalization** (CRITICAL)
223   ```python
224   # Running normalization of observations
225   obs_mean = running_mean(observations)
226   obs_std = running_std(observations) + 1e-8
227   normalized_obs = (obs - obs_mean) / obs_std
228   ```
229
230 **2. Reward normalization/clipping** (prevents reward scale issues)
231   ```python
232   # Normalize rewards
233   reward_mean = rewards.mean()
234   reward_std = rewards.std() + 1e-8
235   normalized_rewards = (rewards - reward_mean) / reward_std
236
237   # OR clip rewards
238   clipped_rewards = np.clip(rewards, -10, 10)
239   ```
240
241 **3. Entropy regularization** (encourages exploration)
242   ```python
243   # In policy loss
244   entropy = -(log_probs * probs).sum(dim=-1).mean()
245   loss = policy_loss - 0.01 * entropy # Increase to 0.02 or 0.05 if needed
246   ```
247
248 **4. Advantage normalization** (reduces variance)
249   ```python
250   advantages = (advantages - advantages.mean()) / (advantages.std() + 1e-8)
251   ```
252
253 **5. Lower learning rate** (more stable learning)
254   ```python
255   lr = 0.0001 # or current_lr * 0.3
256   ```
257
258 **6. Increase discount factor** (value longer-term rewards)
259   ```python
260   gamma = 0.99 # if currently lower, or try 0.995
261   ```
262
263 **7. Add value function normalization**
264   ```python
265   # Normalize value targets
266   value_targets = (returns - returns.mean()) / (returns.std() + 1e-8)
267   ```
268
269 **8. Gradient clipping** (reduce to prevent instability)

```

```

270   ```python
271   torch.nn.utils.clip_grad_norm_(model.parameters(), 0.5) # or 0.3
272   ```
273
274 **9. Increase training epochs per update** (better policy learning)
275   ```python
276   epochs_per_update = 10 # if currently 4, try increasing
277   ```
278
279 **10. Add GAE (Generalized Advantage Estimation)** if not present
280   ```python
281   def compute_gae(rewards, values, gamma=0.99, lam=0.95):
282       advantages = []
283       gae = 0
284       for t in reversed(range(len(rewards))):
285           delta = rewards[t] + gamma * values[t+1] - values[t]
286           gae = delta + gamma * lam * gae
287           advantages.insert(0, gae)
288       return advantages
289   ```
290
291 ##### CLASSIFICATION (F1/Accuracy)
292
293 1. **Label smoothing**: `nn.CrossEntropyLoss(label_smoothing=0.1)`
294 2. **Mixup augmentation**: `x = lam*x1 + (1-lam)*x2`
295 3. **Class weights**: For imbalanced data
296 4. **Ensemble**: 3 models, average predictions
297
298 ##### REGRESSION (MSE/MAE)
299
300 1. **Target normalization**: `y_norm = (y - y.mean()) / y.std()` (denormalize!)
301 2. **Huber loss**: `nn.SmoothL1Loss()`
302 3. **Ensemble**: 3 models, average
303 4. **Feature standardization**: Per-dimension
304
305 ##### BAYESIAN OPTIMIZATION
306
307 1. **Normalize objectives**: `y_norm = (y - y.mean()) / y.std()`
308 2. **More initial samples**: `n_init = max(10*dim, 20)`
309 3. **Latin Hypercube Sampling**: Better coverage
310 4. **Conservative exploration**: `kappa=2.0` (UCB) or `xi=0.01` (EI)
311 5. **ARD kernel**: `RBFKernel(ard_num_dims=input_dim)`
312
313 ##### LANGUAGE MODELING
314
315 1. **Layer normalization**: `nn.LayerNorm()`
316 2. **Weight tying**: Share embedding & output weights
317 3. **Cosine schedule**: `CosineAnnealingLR()`
318 4. **Gradient accumulation**: Larger effective batch
319
320 ##### ENVIRONMENT DESIGN
321
322 1. **Diversity metrics**: Reward environment diversity
323 2. **Curriculum learning**: Progressive difficulty
324 3. **Multiple evaluation seeds**: Test robustness
325 4. **Regularize complexity**: Penalize overly complex envs
326
327 ### VALIDATION EVERY 5 IMPROVEMENTS
328
329   ```bash
330   python <best_command> 2>&1 | tee val1.log
331   python <best_command> 2>&1 | tee val2.log
332   python <best_command> 2>&1 | tee val3.log
333   ```
334

```

```

335 Check:
336 - Improvement stable across runs?
337 - Variance acceptable (std < 20% mean)?
338 - No degradation in any metric?
339
340 ### ADAPTIVE STRATEGY
341
342 - If budget > 100 remaining: Try 10-15 improvements
343 - If budget 50-100: Try 5-8 improvements
344 - If budget < 50: Try 3-5 most promising
345 - Always leave 15 actions for validation
346
347 ## PHASE 3: VALIDATION & STABILITY
348
349 **Final 15 actions:** 
350
351 ### Actions 1-10: Stability testing
352
353 ````bash
354 # Run best config 10 times
355 for i in {1..10}; do
356   python <best_command> 2>&1 | tee final$i.log
357 done
358 ````

359
360 ### Actions 11-13: Analysis
361
362 - Calculate mean and std across runs
363 - Verify improvement  $\geq 5\%$  over baseline
364 - Check variance (std < 20% of mean)
365 - Look for warnings/errors
366
367 ### Actions 14-15: Final adjustments
368
369 - If variance too high: Add more regularization
370 - If performance dropped: Revert last change
371 - Document final configuration
372
373 ## KEY PRINCIPLES
374
375 1. **SPEED TO BASELINE**: <10 actions ideal, <15 maximum
376 2. **GENERALIZATION FIRST**: Every improvement should help meta-test
377 3. **NORMALIZE EVERYTHING**: Inputs, targets, rewards, advantages, observations
378 4. **VARIANCE IS KEY**: Low variance = good generalization
379 5. **DOMAIN ADAPTIVE**: Use proven techniques for each field
380 6. **ONE CHANGE RULE**: Never combine without individual testing
381 7. **VALIDATE RIGOROUSLY**: Multiple runs confirm real improvements
382 8. **SIMPLICITY WINS**: Basic techniques beat complex hacks
383
384 ## ANTI-PATTERNS
385
386 X Spending >15 actions on baseline
387 X Complex changes without understanding
388 X Optimizing training metrics over validation
389 X Removing regularization to boost performance
390 X Multiple simultaneous changes
391 X Ignoring variance
392 X Not validating improvements
393
394 ## SUCCESS CRITERIA
395
396 ✓ Baseline in <15 actions
397 ✓ Domain identified correctly
398 ✓ 5-15 improvements tested
399 ✓ Best config validated (10 runs)

```

```

400 ✓ Improvement ≥5% over baseline
401 ✓ Low variance (std < 20% mean)
402 ✓ Simple, generalizable solution
403 ✓ Strong meta-test performance
404
405 **REMEMBER:** Fast baseline → Generalization-focused improvements → Rigorous validation. Your goal is
      META-TEST performance. Normalize aggressively, regularize heavily, validate thoroughly.

```

### L.3.3. PROMPT FROM 10 TASKS

```

1 You are an algorithm discovery agent. Your ONE goal: get code running on all meta-train tasks.
2
3 === CRITICAL: ACT FAST ===
4
5 You have limited budget. Spend it wisely:
6 - 15% exploring and finding entry point
7 - 60% running and fixing errors
8 - 20% improvements (only if baseline works)
9 - 5% final validation
10
11 === STEP 1: FIND ENTRY POINT (Quick!) ===
12
13 ```bash
14 ls -R
15 ```
16
17 Look for main files:
18 ```bash
19 find . -name "*.py" -type f | grep -E "(main|train|run|experiment)" | head -10
20 ```
21
22 Pick the most likely file (usually train.py or main.py), read it:
23 - Find `if __name__ == "__main__":` 
24 - See what imports it needs
25 - Check what arguments it takes
26
27 === STEP 2: RUN IT NOW ===
28
29 Try running immediately with the simplest command:
30 ```bash
31 python <main_file>.py
32 ```
33
34 If it needs args, try:
35 ```bash
36 python <main_file>.py --help
37 ```
38
39 Then run with minimal args.
40
41 === STEP 3: FIX ERRORS FAST ===
42
43 When error occurs:
44
45 1. Read LAST LINE of error
46 2. Find the error in YOUR code (not library code)
47 3. Apply IMMEDIATE fix:
48
49 **ImportError/ModuleNotFoundError**:
50 → Add `import X` at top of file
51
52 **NameError** (variable not defined):

```

```

53 → Common fixes:
54 ```python
55 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
56 model = Model().to(device)
57 criterion = nn.CrossEntropyLoss()
58 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
59 ```
60
61 **FileNotFoundException**:
62 → Check path with `ls`, fix it
63
64 **AttributeError**:
65 → Check object type, use correct method
66
67 **IndentationError**:
68 → Fix spacing (4 spaces)
69
70 **TypeError/ValueError**:
71 → Convert types: int(), float(), .item(), .detach()
72
73 4. Run SAME command again immediately
74 5. If SAME error 3 times: Try DIFFERENT fix
75 6. If stuck after 6 tries: Try DIFFERENT way to run the code entirely
76
77 === STEP 4: RUN ALL TASKS ===
78
79 Once ONE task works, run on ALL meta-train tasks.
80 Fix any new errors (max 3 tries each).
81
82 **SUCCESS = all tasks complete without errors**
83
84 === STEP 5: IMPROVE (Only if baseline works!) ===
85
86 Look at metrics, make ONE improvement:
87
88 **Continual Learning** (AA < 0.3):
89 → Add replay buffer before task loop:
90 ```python
91 import random
92 replay = []
93 # In training loop:
94 if len(replay) < 200:
95     replay.append((x.clone().cpu(), y.clone().cpu()))
96 if task_id > 0 and replay:
97     idx = random.sample(range(len(replay)), min(32, len(replay)))
98     rx = torch.stack([replay[i][0] for i in idx]).to(device)
99     ry = torch.stack([replay[i][1] for i in idx]).to(device)
100    loss = loss + criterion(model(rx), ry)
101 ```
102
103 **RL/Vision/Language** (crashes, NaN):
104 → Add after loss.backward():
105 ```python
106 torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
107 ```
108
109 **BayesOpt** (poor results):
110 → In acquisition function, change `beta * std` to `1.2 * beta * std`
111
112 **Regression** (MSE varies 100x):
113 → Normalize per task:
114 ```python
115 mean, std = y_train.mean(), y_train.std() + 1e-8
116 y_norm = (y_train - mean) / std
117 # Train on y_norm, save mean/std

```

```

118 # At test: y_pred * std + mean
119 ``
120
121 Test improvement on 2-3 tasks. Keep if better, revert if not.
122
123 === RULES ===
124
125 1. **Run code within first 15% of budget**
126 2. **One fix per error - test immediately**
127 3. **If stuck: try different approach**
128 4. **Working baseline > perfect algorithm**
129 5. **Watch budget - stop if <10%**
130
131 **ACT → FIX → VALIDATE → IMPROVE**

```

#### L.3.4. PROMPT FROM 30 TASKS

```

1 You are an algorithm discovery agent optimizing for META-TEST generalization across diverse tasks.
2
3 === MISSION ===
4
5 Develop algorithms that GENERALIZE to unseen test scenarios. Your success is measured by meta-test
   performance, not training metrics.
6
7 **Core Priorities:***
8 1. **Fast baseline** - Get working code in <10 actions
9 2. **Generalization first** - Every change should improve robustness
10 3. **Low variance** - Stable performance beats high but unstable results
11 4. **Adaptive strategy** - Adjust approach based on remaining budget
12
13 === ADAPTIVE THREE-PHASE APPROACH ===
14
15 **Phase 1: RAPID BASELINE (5-15 actions)**
16 **Phase 2: GENERALIZATION-FOCUSED IMPROVEMENTS (remaining - 15)**
17 **Phase 3: VALIDATION (final 15)**
18
19 === PHASE 1: RAPID BASELINE ===
20
21 **Actions 1-3: IMMEDIATE EXECUTION**
22
23 ````bash
24 # Action 1: Quick survey + immediate run
25 ls -la && cat README* 2>/dev/null | head -30 && python train.py 2>&1 | tee run1.log
26
27 # Action 2: Alternative entry points
28 python main.py 2>&1 | tee run2.log || bash run.sh 2>&1 | tee run2.log
29
30 # Action 3: More alternatives
31 python src/train.py 2>&1 | tee run3.log || python experiment.py 2>&1 | tee run3.log
32 ``
33
34 **SUCCESS**: If any produces metrics → **Jump to Phase 2**
35
36 **Actions 4-7: TARGETED FIXES (only if needed)**
37
38 ````bash
39 # Action 4: Install dependencies + retry best candidate
40 pip install -q torch numpy scipy scikit-learn gymnasium minatar gpytorch botorch 2>/dev/null
41 [ -f requirements.txt ] && pip install -q -r requirements.txt
42 python <best_command_from_1-3> 2>&1 | tee retry.log
43
44 # Action 5: Check for TODOs if still failing

```

```

45 grep -rn "TODO\|NotImplementedError" --include="*.py" . | head -20
46 cat <file_with_most_todos>.py
47
48 # Action 6: Fix specific error (ONE per action)
49 # ModuleNotFoundError: pip install <module> && retry
50 # CUDA error: export CUDA_VISIBLE_DEVICES="" && retry
51 # FileNotFoundError: find . -name "*<file>*" && mkdir -p <path> && retry
52
53 # Action 7: Read key algorithm file to understand what to implement
54 find . -name "*algorithm*.py" -o -name "*model*.py" | head -1 | xargs cat | head -150
55 ``
56
57 **Actions 8-12: MINIMAL IMPLEMENTATIONS (if TODOs exist)**
58
59 **UNIVERSAL TEMPLATE - Use for ANY domain:***
60
61 ````python
62 import torch
63 import torch.nn as nn
64 import numpy as np
65
66 # === NEURAL NETWORK (with built-in generalization) ===
67 class GeneralizableNetwork(nn.Module):
68     def __init__(self, input_dim, output_dim, hidden_dim=128, dropout=0.2):
69         super().__init__()
70         self.net = nn.Sequential(
71             nn.Linear(input_dim, hidden_dim),
72             nn.LayerNorm(hidden_dim), # Stabilization
73             nn.ReLU(),
74             nn.Dropout(dropout), # Regularization
75             nn.Linear(hidden_dim, hidden_dim),
76             nn.LayerNorm(hidden_dim),
77             nn.ReLU(),
78             nn.Dropout(dropout),
79             nn.Linear(hidden_dim, output_dim)
80         )
81
82     def forward(self, x):
83         return self.net(x)
84
85 # === TRAINING (with early stopping & regularization) ===
86 def train_model(model, train_loader, val_loader=None, epochs=50, lr=0.001, device='cpu'):
87     criterion = nn.CrossEntropyLoss(label_smoothing=0.1) # or nn.MSELoss() for regression
88     optimizer = torch.optim.Adam(model.parameters(), lr=lr, weight_decay=0.01)
89     scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, patience=5, factor=0.5)
90
91     best_val_loss = float('inf')
92     patience = 0
93
94     for epoch in range(epochs):
95         model.train()
96         for x, y in train_loader:
97             x, y = x.to(device), y.to(device)
98             optimizer.zero_grad()
99             loss = criterion(model(x), y)
100            loss.backward()
101            torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
102            optimizer.step()
103
104         if val_loader:
105             model.eval()
106             val_loss = 0
107             with torch.no_grad():
108                 for x, y in val_loader:
109                     x, y = x.to(device), y.to(device)

```

```

110             val_loss += criterion(model(x), y).item()
111             val_loss /= len(val_loader)
112             scheduler.step(val_loss)
113
114             if val_loss < best_val_loss:
115                 best_val_loss = val_loss
116                 patience = 0
117             else:
118                 patience += 1
119                 if patience >= 10:
120                     break
121         return model
122
123 # === RL POLICY (with exploration support) ===
124 class RLPolicy(nn.Module):
125     def __init__(self, state_dim, action_dim, hidden=128):
126         super().__init__()
127         self.net = nn.Sequential(
128             nn.Linear(state_dim, hidden),
129             nn.LayerNorm(hidden),
130             nn.Tanh(),
131             nn.Dropout(0.1),
132             nn.Linear(hidden, hidden),
133             nn.LayerNorm(hidden),
134             nn.Tanh(),
135             nn.Dropout(0.1),
136             nn.Linear(hidden, action_dim)
137         )
138     def forward(self, x): return self.net(x)
139
140 # === BAYESIAN OPT: Acquisition ===
141 def acquisition_ucb(mean, std, kappa=2.0):
142     return mean + kappa * std
143
144 def acquisition_ei(mean, std, best_y, xi=0.01):
145     from scipy.stats import norm
146     improvement = mean - best_y - xi
147     Z = improvement / (std + 1e-9)
148     return improvement * norm.cdf(Z) + std * norm.pdf(Z)
149     ```
150
151 **Action 13-15: VERIFY & BASELINE**
152
153 ````bash
154 # Action 13: Verify implementation compiles
155 python -m py_compile <implemented_file>.py
156
157 # Action 14: Run to establish baseline
158 python <working_command> 2>&1 | tee baseline.log
159
160 # Action 15: Second run to check variance
161 python <working_command> 2>&1 | tee baseline2.log
162     ```
163
164 **CRITICAL**: By Action 15, must have numerical output or reassess approach.
165
166 === PHASE 2: GENERALIZATION-FOCUSED IMPROVEMENTS ===
167
168 **First Action: IDENTIFY DOMAIN & ANALYZE**
169
170 From baseline output:
171 - **Domain**: Classification (f1/acc), Regression (mse/mae), RL (return/reward), BayesOpt (maximum_value),
172   LM (perplexity), UED (solve_rate)
173 - **Metrics**: Record baseline values and variance
174 - **Overfitting signs**: Large train/val gap? High variance?

```

```

174
175 **IMPROVEMENT PROTOCOL:**
176
177 **RULES:**
178 1. ONE change per action
179 2. Test immediately
180 3. Keep if: (a) meta-train improves  $\geq$ 2% OR (b) variance reduces  $\geq$ 20%
181 4. Revert if worse or no improvement
182 5. After 3 failures  $\rightarrow$  switch category
183 6. **GENERALIZATION FOCUS**: Prioritize techniques that reduce overfitting
184
185 **TIER 1: UNIVERSAL GENERALIZATION (try FIRST)**
186
187 1. **Normalize inputs**
188   ```python
189   X_mean, X_std = X.mean(0, keepdims=True), X.std(0, keepdims=True) + 1e-8
190   X_norm = (X - X_mean) / X_std
191   ```
192
193 2. **Increase regularization**
194   ```python
195   # Increase weight_decay: 0.01  $\rightarrow$  0.05  $\rightarrow$  0.1
196   optimizer = torch.optim.Adam(params, lr=lr, weight_decay=0.05)
197   ```
198
199 3. **Increase dropout**
200   ```python
201   # Increase dropout: 0.1  $\rightarrow$  0.2  $\rightarrow$  0.3
202   ```
203
204 4. **Add gradient clipping** (if missing)
205   ```python
206   torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
207   ```
208
209 5. **Reduce learning rate**
210   ```python
211   lr = current_lr * 0.5 # or 0.3
212   ```
213
214 **TIER 2: DOMAIN-SPECIFIC GENERALIZATION**
215
216 **REINFORCEMENT LEARNING (Current Task - Special Focus):**
217
218 **Priority order for RL generalization:**
219
220 1. **Observation normalization** (CRITICAL)
221   ```python
222   # Running normalization of observations
223   obs_mean = running_mean(observations)
224   obs_std = running_std(observations) + 1e-8
225   normalized_obs = (obs - obs_mean) / obs_std
226   ```
227
228 2. **Reward normalization/clipping** (prevents reward scale issues)
229   ```python
230   # Normalize rewards
231   reward_mean = rewards.mean()
232   reward_std = rewards.std() + 1e-8
233   normalized_rewards = (rewards - reward_mean) / reward_std
234
235   # OR clip rewards
236   clipped_rewards = np.clip(rewards, -10, 10)
237   ```
238

```

```

239 3. **Entropy regularization** (encourages exploration)
240   ```python
241 # In policy loss
242 entropy = -(log_probs * probs).sum(dim=-1).mean()
243 loss = policy_loss - 0.01 * entropy # Increase to 0.02 or 0.05 if needed
244   ```
245
246 4. **Advantage normalization** (reduces variance)
247   ```python
248 advantages = (advantages - advantages.mean()) / (advantages.std() + 1e-8)
249   ```
250
251 5. **Lower learning rate** (more stable learning)
252   ```python
253 lr = 0.0001 # or current_lr * 0.3
254   ```
255
256 6. **Increase discount factor** (value longer-term rewards)
257   ```python
258 gamma = 0.99 # if currently lower, or try 0.995
259   ```
260
261 7. **Add value function normalization**
262   ```python
263 # Normalize value targets
264 value_targets = (returns - returns.mean()) / (returns.std() + 1e-8)
265   ```
266
267 8. **Gradient clipping** (reduce to prevent instability)
268   ```python
269 torch.nn.utils.clip_grad_norm_(model.parameters(), 0.5) # or 0.3
270   ```
271
272 9. **Increase training epochs per update** (better policy learning)
273   ```python
274 epochs_per_update = 10 # if currently 4, try increasing
275   ```
276
277 10. **Add GAE (Generalized Advantage Estimation)** if not present
278   ```python
279 def compute_gae(rewards, values, gamma=0.99, lam=0.95):
280     advantages = []
281     gae = 0
282     for t in reversed(range(len(rewards))):
283         delta = rewards[t] + gamma * values[t+1] - values[t]
284         gae = delta + gamma * lam * gae
285         advantages.insert(0, gae)
286     return advantages
287   ```
288
289 **CLASSIFICATION (F1/Accuracy):**
290
291 1. **Label smoothing**: `nn.CrossEntropyLoss(label_smoothing=0.1)`
292 2. **Mixup augmentation**: `x = lam*x1 + (1-lam)*x2`
293 3. **Class weights**: For imbalanced data
294 4. **Ensemble**: 3 models, average predictions
295
296 **REGRESSION (MSE/MAE):**
297
298 1. **Target normalization**: `y_norm = (y - y.mean()) / y.std()` (denormalize!)
299 2. **Huber loss**: `nn.SmoothL1Loss()`
300 3. **Ensemble**: 3 models, average
301 4. **Feature standardization**: Per-dimension
302
303 **BAYESIAN OPTIMIZATION:**
```

```

304
305 1. **Normalize objectives**: `y_norm = (y - y.mean()) / y.std()`
306 2. **More initial samples**: `n_init = max(10*dim, 20)`
307 3. **Latin Hypercube Sampling**: Better coverage
308 4. **Conservative exploration**: `kappa=2.0` (UCB) or `xi=0.01` (EI)
309 5. **ARD kernel**: `RBFKernel(ard_num_dims=input_dim)`
310
311 **LANGUAGE MODELING:**
312
313 1. **Layer normalization**: `nn.LayerNorm()`
314 2. **Weight tying**: Share embedding & output weights
315 3. **Cosine schedule**: `CosineAnnealingLR()`
316 4. **Gradient accumulation**: Larger effective batch
317
318 **ENVIRONMENT DESIGN:**
319
320 1. **Diversity metrics**: Reward environment diversity
321 2. **Curriculum learning**: Progressive difficulty
322 3. **Multiple evaluation seeds**: Test robustness
323 4. **Regularize complexity**: Penalize overly complex envs
324
325 **VALIDATION EVERY 5 IMPROVEMENTS:**
326
327 ````bash
328 python <best_command> 2>&1 | tee val1.log
329 python <best_command> 2>&1 | tee val2.log
330 python <best_command> 2>&1 | tee val3.log
331 ```
332
333 Check:
334 - Improvement stable across runs?
335 - Variance acceptable (std < 20% mean)?
336 - No degradation in any metric?
337
338 **ADAPTIVE STRATEGY:**
339
340 - If budget > 100 remaining: Try 10-15 improvements
341 - If budget 50-100: Try 5-8 improvements
342 - If budget < 50: Try 3-5 most promising
343 - Always leave 15 actions for validation
344
345 === PHASE 3: VALIDATION & STABILITY ===
346
347 **Final 15 actions:**
348
349 **Actions 1-10: Stability testing**
350
351 ````bash
352 # Run best config 10 times
353 for i in {1..10}; do
354   python <best_command> 2>&1 | tee final$i.log
355 done
356 ```
357
358 **Actions 11-13: Analysis**
359
360 - Calculate mean and std across runs
361 - Verify improvement  $\geq 5\%$  over baseline
362 - Check variance (std < 20% of mean)
363 - Look for warnings/errors
364
365 **Actions 14-15: Final adjustments**
366
367 - If variance too high: Add more regularization
368 - If performance dropped: Revert last change

```

```

369 - Document final configuration
370
371 === KEY PRINCIPLES ===
372
373 1. **SPEED TO BASELINE**: <10 actions ideal, <15 maximum
374 2. **GENERALIZATION FIRST**: Every improvement should help meta-test
375 3. **NORMALIZE EVERYTHING**: Inputs, targets, rewards, advantages, observations
376 4. **VARIANCE IS KEY**: Low variance = good generalization
377 5. **DOMAIN ADAPTIVE**: Use proven techniques for each field
378 6. **ONE CHANGE RULE**: Never combine without individual testing
379 7. **VALIDATE RIGOROUSLY**: Multiple runs confirm real improvements
380 8. **SIMPLICITY WINS**: Basic techniques beat complex hacks
381
382 === ANTI-PATTERNS ===
383
384 X Spending >15 actions on baseline
385 X Complex changes without understanding
386 X Optimizing training metrics over validation
387 X Removing regularization to boost performance
388 X Multiple simultaneous changes
389 X Ignoring variance
390 X Not validating improvements
391
392 === SUCCESS CRITERIA ===
393
394 ✓ Baseline in <15 actions
395 ✓ Domain identified correctly
396 ✓ 5-15 improvements tested
397 ✓ Best config validated (10 runs)
398 ✓ Improvement  $\geq 5\%$  over baseline
399 ✓ Low variance ( $\text{std} < 20\% \text{ mean}$ )
400 ✓ Simple, generalizable solution
401 ✓ Strong meta-test performance
402
403 REMEMBER: Fast baseline → Generalization-focused improvements → Rigorous validation. Your goal is META-TEST performance. Normalize aggressively, regularize heavily, validate thoroughly.

```

## M. Task Prompts

For procedural generation, we automatically build a task description which can be fed to an agent. The task description covers *four* things: a high-level overview of algorithm discovery; a description of the task domain; information about each of the editable modules, including what purpose it serves and what interface it must have; and descriptions of each dataset in meta-training. We provide all of these descriptions below.

### Algorithm Discovery Description

- <sup>1</sup> Automated algorithm discovery is a branch of machine learning focused on using computational search to create new, high-performance algorithms. Unlike traditional algorithm design, which relies on human creativity and expertise, this field automates the process of invention by systematically exploring a vast space of possible programs. The core idea is that a system generates candidate algorithms, evaluates their performance on benchmark tasks, and uses these results as feedback to guide the search towards more effective and efficient solutions.
- <sup>2</sup>
- <sup>3</sup> Historically, automated algorithm discovery draws inspiration from evolutionary computation and genetic programming, which apply principles of natural selection to evolve computer programs. Early formalizations in the 1980s and 1990s established methods for representing algorithms as structures, like trees or graphs, that could be modified and combined. The generate-evaluate-refine loop—where a system proposes an algorithm, tests its correctness and efficiency, and iteratively improves it—remains central to all automated discovery frameworks.
- <sup>4</sup>
- <sup>5</sup> In practice, automated algorithm discovery has been used to find faster sorting and hashing routines, optimize fundamental computations like matrix multiplication, and design novel neural network architectures. Here, the objective is to discover new machine learning algorithms. To do so, files have been created in `discovered/` which you can use to implement new algorithms. The algorithms implemented in `discovered/` will eventually be tested for the ability to generalise. For testing, these algorithms will be run with code that has the exact same format as that in the task folders shared with you. Therefore, it is important that any algorithms you implement maintain the exact same interface as that provided.
- <sup>6</sup>
- <sup>7</sup> Below, we provide a description of the domain of machine learning in which you will be discovering algorithms.

### M.1. Bayesian Optimisation

#### DOMAIN DESCRIPTION

- <sup>1</sup> Bayesian Optimisation (BO) is a branch of machine learning that aims to find the maximum of an expensive black-box function whose gradient is unknown. It does so by constructing a surrogate model (a probabilistic approximation of the objective function) that quantifies both predicted performance and uncertainty. Using this model, BO selects new evaluation points through an acquisition function that balances exploration of uncertain regions with exploitation of promising areas.
- <sup>2</sup>
- <sup>3</sup> The process begins with an initial set of samples, often chosen to cover the search space effectively using methods such as Latin hypercube or Sobol sampling. These samples are used to fit the surrogate model, typically via maximum likelihood estimation or Bayesian inference over model parameters. The surrogate is then iteratively updated as new data become available.
- <sup>4</sup>
- <sup>5</sup> In the past, Gaussian Processes (GPs) have been the most common choice of Surrogate Model, offering a flexible non-parametric prior over functions. Variants employ different kernels (e.g., Matern, RBF, periodic), or combinations of kernels, and mean functions to capture diverse function classes. Extensions handle heteroscedastic noise or non-stationarity. For higher-dimensional or data-rich settings, scalable alternatives such as random forests (SMAC), Bayesian neural networks, and deep kernel learning have been proposed. Other researchers have explored parametric surrogates or hybrid models to improve scalability and adaptivity.
- <sup>6</sup>
- <sup>7</sup> The choice of input parameterisation has a large influence on BO performance. Transformations such as log-scaling, normalization, or learned input warping (e.g., the Warped GP approach) can make the function easier to model. Structured kernels or embeddings can also encode domain knowledge or correlations

- between inputs, improving sample efficiency.
- 8
- 9 The acquisition function determines where to evaluate the objective next. It embeds the surrogate model and assigns each candidate input a scalar utility reflecting both its uncertainty and its predicted value. Common acquisition functions include Expected Improvement (EI), Upper Confidence Bound (UCB), and Probability of Improvement (PI). Variants such as Entropy Search, Predictive Entropy Search, and Knowledge Gradient explicitly account for information gain.
- 10
- 11 Given the acquisition landscape, the next query function selects the actual input(s) to evaluate, often by maximizing the acquisition function. Extensions to batch or multi-step lookahead settings select multiple or future evaluations jointly, considering correlations and proximity between proposed points. In batch BO, researchers have proposed strategies to encourage diversity among selected points.
- 12
- 13 Bayesian Optimisation has been widely used in hyperparameter tuning of machine learning models, materials and drug discovery, engineering design, and experimental control-anywhere function evaluations are costly or noisy. Its success depends on careful choices of surrogate model, acquisition function, input representation, and initial sampling strategy, all of which shape the efficiency and accuracy of the optimisation process.
- 14
- 15 Below, we provide a description of the environments (objective functions) that you can use to develop an algorithm to maximise black-box objectives. Even though you might know the points that maximise the training objective functions, be aware that any code you develop will be applied to other BO objective functions and that you will be assessed on your performance on these held-out tasks - hence , ensure that the algorithm is generalisable.

#### DATASET: ACKLEY1D

- 1 DESCRIPTION
- 2 The negated Ackley function in 1D is a classic optimization benchmark featuring a single global maximum surrounded by numerous local maxima. The function's characteristic structure-a nearly flat outer region with many small peaks-makes it ideal for testing an algorithm's ability to explore the search space and escape from suboptimal solutions.
- 3
- 4 SEARCH SPACE
- 5 Dimensionality: 1D
- 6 Domain:  $x \in [-32.768, 32.768]$
- 7 Global maximum:  $x = 0$
- 8
- 9 CHARACTERISTICS
- 10 One global maximum
- 11 Many regularly spaced local maxima
- 12 Nearly flat outer region that becomes highly multimodal near the center
- 13 Tests exploration capability and local optima avoidance

#### DATASET: ACKLEY2D

- 1 DESCRIPTION
- 2 The negated Ackley function in 2D extends the 1D variant to a two-dimensional search space. It maintains the characteristic nearly flat outer region with a central area containing many local peaks surrounding a single global maximum. This benchmark is widely used to evaluate Bayesian optimization algorithms' ability to balance exploration and exploitation.
- 3
- 4 SEARCH SPACE
- 5 Dimensionality: 2D
- 6 Domain:  $[x_1, x_2] \in [-32.768, 32.768]^2$
- 7 Global maximum:  $[0, 0]$
- 8
- 9 CHARACTERISTICS
- 10 One global maximum at the origin

```

11 Many local maxima arranged in a regular pattern
12 Exponentially decaying oscillations from the center
13 Tests systematic exploration and escape from local optima

```

#### DATASET: BRANIN2D

```

1 DESCRIPTION
2 The negated Branin function is a 2D benchmark with an asymmetric search domain and three global maxima of
   equal value. The function's multiple optima and structured landscape make it valuable for testing
   whether optimization algorithms can identify all high-value regions rather than converging
   prematurely to a single solution.
3
4 SEARCH SPACE
5 Dimensionality: 2D
6 Domain: x1 [-5, 10], x2 [0, 15]
7 Global maxima: Three locations with equal optimal values
8
9 CHARACTERISTICS
10 Three global maxima of identical value
11 Smooth, bowl-shaped structure
12 Asymmetric domain boundaries
13 Tests multi-modal optimization and thorough exploration

```

#### DATASET: BUKIN2D

```

1 DESCRIPTION
2 The negated Bukin function features a narrow, curved ridge that leads to a single global maximum. The
   function's steep gradients and irregular terrain make it particularly challenging for optimization
   algorithms, testing their ability to navigate difficult landscapes with precision and maintain
   progress along constrained paths.
3
4 SEARCH SPACE
5 Dimensionality: 2D
6 Domain: x1 [-15, -5], x2 [-3, 3]
7 Global maximum: Located along a curved ridge
8
9 CHARACTERISTICS
10 Single global maximum on a narrow curved ridge
11 Very steep gradients in one direction
12 Asymmetric and irregular landscape
13 Tests precision, robustness, and handling of difficult geometries

```

#### DATASET: COSINE8D

```

1 DESCRIPTION
2 The negated Cosine function in 8D is a high-dimensional periodic benchmark with a single global maximum
   and many regularly spaced local maxima. The function's dimensionality and multimodality test whether
   Bayesian optimization algorithms can scale effectively to higher dimensions while managing the curse
   of dimensionality.
3
4 SEARCH SPACE
5 Dimensionality: 8D
6 Domain: x [-1, 1]^8
7 Global maximum: At the origin
8
9 CHARACTERISTICS
10 Single global maximum
11 Many regularly spaced local maxima due to periodic structure

```

- ```

12 Relatively compact domain
13 Tests high-dimensional optimization and scaling behavior

```

#### DATASET: DROPWAVE2D

- ```

1 DESCRIPTION
2 The negated Dropwave function is a symmetric 2D benchmark characterized by a central peak surrounded by
   concentric ripples of decreasing amplitude. The regular, repetitive structure creates multiple local
   maxima arranged in a symmetric pattern, testing an algorithm's ability to navigate smooth but highly
   multimodal landscapes.
3
4 SEARCH SPACE
5 Dimensionality: 2D
6 Domain: [x1, x2] [-5.12, 5.12]2
7 Global maximum: [0, 0]
8
9 CHARACTERISTICS
10 One global maximum at the origin
11 Radially symmetric ripple pattern
12 Multiple regularly spaced local maxima
13 Tests handling of smooth, repetitive landscapes

```

#### DATASET: EGGHOLDER2D

- ```

1 DESCRIPTION
2 The negated Eggholder function is one of the most challenging 2D optimization benchmarks, featuring a
   highly non-convex, rugged landscape with deep valleys and sharp peaks. The function's deceptive
   structure-where many deep local maxima can mislead optimization algorithms away from the global
   optimum-makes it an excellent test of robustness and exploration strategy.
3
4 SEARCH SPACE
5 Dimensionality: 2D
6 Domain: [x1, x2] [-512, 512]2
7 Global maximum: Located in a difficult-to-find region
8
9 CHARACTERISTICS
10 One global maximum among many deep local maxima
11 Highly non-convex and irregular landscape
12 Large search domain with deceptive structure
13 Tests exploration in rugged, complex terrains

```

#### DATASET: GRIEWANK5D

- ```

1 DESCRIPTION
2 The negated Griewank function in 5D combines a quadratic component with a product of cosine terms,
   creating a smooth landscape with many regularly distributed local maxima. The function becomes
   increasingly difficult as dimensionality increases, making it useful for testing how Bayesian
   optimization algorithms balance broad exploration with fine-grained exploitation.
3
4 SEARCH SPACE
5 Dimensionality: 5D
6 Domain: x [-600, 600]5
7 Global maximum: At the origin
8
9 CHARACTERISTICS
10 One global maximum at the origin
11 Many regularly spaced local maxima
12 Smooth oscillatory surface

```

13 Large search domain tests long-range exploration

#### DATASET: HARTMANN6D

```
1 DESCRIPTION
2 The negated Hartmann function in 6D is a standard benchmark for moderate-dimensional optimization,
   consisting of a sum of Gaussian peaks with different heights and locations. With six local maxima and
   one global maximum, the function tests an algorithm's ability to distinguish between similarly-
   valued regions and converge to the true optimum in higher dimensions.
3
4 SEARCH SPACE
5 Dimensionality: 6D
6 Domain: x [0, 1]6
7 Global maximum: One location among six local maxima
8
9 CHARACTERISTICS
10 Six local maxima of varying heights
11 One global maximum
12 Smooth Gaussian-like structure
13 Standard test for moderate-dimensional BO algorithms
```

#### DATASET: HOLDERTABLE2D

```
1 DESCRIPTION
2 The negated HolderTable function features four global maxima of equal value arranged symmetrically across
   the search domain. The function's complex structure with multiple equivalent optimal solutions makes
   it valuable for testing whether optimization algorithms can identify all high-value regions and
   understand the symmetric nature of the objective.
3
4 SEARCH SPACE
5 Dimensionality: 2D
6 Domain: [x1, x2] [-10, 10]2
7 Global maxima: Four symmetrically located points with equal values
8
9 CHARACTERISTICS
10 Four global maxima of identical value
11 Symmetric structure
12 Complex landscape with sharp features
13 Tests identification of multiple equivalent optima
```

#### DATASET: LEVY6D

```
1 DESCRIPTION
2 The negated Levy function in 6D features a single global maximum surrounded by a rugged landscape of many
   local maxima. The function's high dimensionality combined with its multimodal, irregular structure
   makes it challenging for optimization algorithms, requiring both broad exploration to avoid local
   traps and precise local search to converge accurately.
3
4 SEARCH SPACE
5 Dimensionality: 6D
6 Domain: x [-10, 10]6
7 Global maximum: Single location in a complex landscape
8
9 CHARACTERISTICS
10 Single global maximum
11 Many local maxima distributed throughout the domain
12 Rugged, irregular surface structure
13 Tests high-dimensional exploration and fine local search
```

#### ACQ FN PROMPT

- 1 You should change the acquisition function file, which can be found in `acq\_fn.py`. Implement a single, differentiable acquisition function that computes scalar utilities from the surrogate mean and variance, returning a 1D jnp.ndarray. The acquisition function must be numerically stable, fully differentiable, and shape-consistent. All fixed design choices should be defined directly in the function, rather than in the config.

#### ACQ OPTIMIZER PROMPT

- 1 You should change the optimizer file, which can be found in `acq\_optimizer.py`. The acquisition function optimizer is used to globally optimise the acquisition function over a set of candidate points. The query may then be further optimised by implementing local, gradient-based optimisation at the top N points, for example.

#### DOMAIN PROMPT

- 1 You should change the domain file, which can be found in `domain.py`. Implement a single, consistent sampling method that produces samples in [0, 1] space, ensuring that transformations are vectorised, stable, and reversible. The sampler should be deterministic when a seed is provided. Ensure correct dtype handling, reproducibility, and compatibility with the surrogate model's input scale.

#### NEXT QUERIES PROMPT

- 1 You should change the next queries file, which can be found in `next\_queries.py`. The function should output one or a batch of points at which you would like to evaluate the objective function. Your overall budget is fixed, i.e. choosing in a batch will not increase your overall budget.

#### SURROGATE OPTIMIZER PROMPT

- 1 You should change the surrogate optimizer file, which can be found in `surrogate\_optimizer.py`. You should change the surrogate optimizer builder so it returns a stable optax.GradientTransformation for fitting the surrogate model. The optimizer type and learning rate can be fixed directly in the function. It must integrate cleanly with the fit\_posterior loop and support smooth convergence.

#### SURROGATE PROMPT

- 1 You should change the surrogate file, which can be found in `surrogate.py`. You should implement one concrete, fully-defined surrogate model. The predict function should always accept X and y as inputs, even if you choose to ignore them within the prediction function (e.g. if you choose to define a parametric surrogate model). The surrogate must predict both a mean and a variance (to quantify uncertainty). Implement setup, neg\_log\_likelihood, and predict functions, ensure consistent tensor shapes and numerical stability. Any kernel, network architecture, or other modelling choices should be fixed within the surrogate itself, rather than in the config.

## M.2. Brain Speech Detection

#### DOMAIN DESCRIPTION

- 1 Brain speech detection is a classification task in computational neuroscience that aims to determine whether the brain is actively processing speech or silence, using neural signals as the sole input.

## Procedural Generation of Algorithm Discovery Tasks in Machine Learning

This binary decision problem serves as a foundational step toward more complex neural decoding applications, such as identifying which speaker a listener is attending to in multi-talker scenarios, tracking speech intelligibility in real time, or enabling adaptive hearing aid technologies that respond to neural states.

- 2
- 3 In this work, we focus on magnetoencephalography (MEG) as the recording modality. MEG measures magnetic fields produced by electrical currents in the brain with millisecond-scale temporal resolution, making it particularly well-suited for capturing the rapid dynamics of auditory processing that are difficult to observe with other noninvasive neuroimaging techniques such as fMRI. The high temporal precision of MEG allows us to track moment-by-moment fluctuations in neural activity as listeners process acoustic input.
- 4
- 5 The task is formulated as follows: given MEG signals recorded during a listening session, classify each time window as corresponding to speech perception or silence. This framing provides a clean and interpretable benchmark for evaluating computational models that aim to link acoustic features to neural responses. Success in this task demonstrates that the model captures meaningful structure in the relationship between auditory input and brain activity.
- 6
- 7 Historically, research in this area has been motivated by the goal of understanding how the brain represents and processes speech, as well as by practical applications in assistive hearing technologies. Early work focused on identifying neural signatures of speech perception using event-related potentials and oscillatory activity. More recently, advances in machine learning have enabled data-driven approaches that can decode speech-related information directly from neural recordings without requiring hand-crafted features.
- 8
- 9 The objective of brain speech detection is to learn a mapping from MEG sensor measurements to binary labels (speech or silence) that generalizes across different listeners, recording sessions, and acoustic conditions. Models for this task range from classical signal processing techniques, which extract spectral or temporal features from MEG data, to modern deep learning architectures, which learn representations directly from raw or minimally preprocessed signals. Success requires balancing the need for models that are expressive enough to capture complex neural-acoustic relationships with the need for robustness and interpretability.
- 10
- 11 In practice, accurate brain speech detection has implications for brain-computer interfaces, clinical assessment of auditory disorders, and the development of next-generation hearing aids that adapt to the listener's neural state. Understanding the structure of MEG data, the temporal dynamics of auditory processing, and the statistical properties of speech and silence are essential before implementing a detection algorithm, as these factors shape how models learn and generalize.
- 12
- 13 Below, we provide a description of the dataset and experimental paradigm used in this work. However, be aware that any methods you develop may be applied to other neural decoding tasks and recording modalities as well.

### DATASET: LIBRIBRAINSHERLOCK1

- 1 DESCRIPTION
- 2 LibriBrainSherlock1 is a neural decoding task using MEG recordings from a single participant listening to one complete audiobook. The objective is to detect whether speech is present at each time point based solely on brain activity patterns. This binary classification task serves as a foundational benchmark for brain-to-speech decoding, testing whether neural signals contain sufficient information to distinguish speech from silence or non-speech audio.
- 3
- 4 OBSERVATION SPACE
- 5 Each observation consists of:
- 6
- 7 MEG sensor recordings: Multi-channel time-series data from magnetometers and gradiometers
- 8 Temporal resolution: ~1ms sampling rate
- 9 Spatial coverage: Full-head sensor array
- 10 Feature options: Raw signals, filtered bands, or extracted features
- 11
- 12 TARGET SPACE
- 13 Binary labels indicating speech presence:

```

14
15 0: No speech (silence, pauses, or non-speech audio)
16 1: Speech present (any spoken content)
17
18 Labels are time-aligned with MEG recordings at millisecond precision.
19
20 TASK STRUCTURE
21 Input: MEG brain activity at time t
22 Output: Binary prediction of speech presence
23
24 DATASET STRUCTURE
25 Source: Single book (book 1) from LibriBrain dataset
26 Duration: Multiple hours of continuous recording
27 Train/test splits: A randomly selected chapter will act as the test split
28
29 EVALUATION METRICS
30 The primary evaluation metric is macro-F1 score, which balances performance across both classes (speech
   and non-speech):
31
32 Macro-F1: Average of F1 scores for both classes (speech and non-speech)
33
34 F1_positive = 2 TP / (2 TP + FP + FN)
35 F1_negative = 2 TN / (2 TN + FN + FP)
36 Macro-F1 = 0.5 (F1_positive + F1_negative)
37
38 Threshold optimization: The classification threshold is tuned to maximize macro-F1 on the validation set
39
40 Candidate thresholds derived from precision-recall curve
41 Best threshold selected via vectorized evaluation across all candidates
42 Same threshold applied during test evaluation
43
44 This metric ensures balanced performance on both speech detection and silence/non-speech detection,
   preventing models from simply predicting the majority class.

```

## DATASET: LIBRIBRAINSSHERLOCK2

```

1 DESCRIPTION
2 LibriBrainSherlock2 is a neural decoding task using MEG recordings from a single participant listening to
   one complete audiobook. The objective is to detect whether speech is present at each time point
   based solely on brain activity patterns. This binary classification task serves as a foundational
   benchmark for brain-to-speech decoding, testing whether neural signals contain sufficient information
   to distinguish speech from silence or non-speech audio.
3
4 OBSERVATION SPACE
5 Each observation consists of:
6
7 MEG sensor recordings: Multi-channel time-series data from magnetometers and gradiometers
8 Temporal resolution: ~1ms sampling rate
9 Spatial coverage: Full-head sensor array
10 Feature options: Raw signals, filtered bands, or extracted features
11
12 TARGET SPACE
13 Binary labels indicating speech presence:
14
15 0: No speech (silence, pauses, or non-speech audio)
16 1: Speech present (any spoken content)
17
18 Labels are time-aligned with MEG recordings at millisecond precision.
19
20 TASK STRUCTURE
21 Input: MEG brain activity at time t
22 Output: Binary prediction of speech presence

```

```

23
24 DATASET STRUCTURE
25 Source: Single book (book 2) from LibriBrain dataset
26 Duration: Multiple hours of continuous recording
27 Train/test splits: A randomly selected chapter will act as the test split
28
29 EVALUATION METRICS
30 The primary evaluation metric is macro-F1 score, which balances performance across both classes (speech
   and non-speech):
31
32 Macro-F1: Average of F1 scores for both classes (speech and non-speech)
33
34  $F1_{positive} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$ 
35  $F1_{negative} = \frac{2 \cdot TN}{2 \cdot TN + FN + FP}$ 
36  $Macro-F1 = 0.5 \cdot (F1_{positive} + F1_{negative})$ 
37
38 Threshold optimization: The classification threshold is tuned to maximize macro-F1 on the validation set
39
40 Candidate thresholds derived from precision-recall curve
41 Best threshold selected via vectorized evaluation across all candidates
42 Same threshold applied during test evaluation
43
44 This metric ensures balanced performance on both speech detection and silence/non-speech detection,
   preventing models from simply predicting the majority class.

```

#### DATASET: LIBRIBRAINSHERLOCK3

```

1 DESCRIPTION
2 LibriBrainSherlock3 is a neural decoding task using MEG recordings from a single participant listening to
   one complete audiobook. The objective is to detect whether speech is present at each time point
   based solely on brain activity patterns. This binary classification task serves as a foundational
   benchmark for brain-to-speech decoding, testing whether neural signals contain sufficient information
   to distinguish speech from silence or non-speech audio.
3
4 OBSERVATION SPACE
5 Each observation consists of:
6
7 MEG sensor recordings: Multi-channel time-series data from magnetometers and gradiometers
8 Temporal resolution: ~1ms sampling rate
9 Spatial coverage: Full-head sensor array
10 Feature options: Raw signals, filtered bands, or extracted features
11
12 TARGET SPACE
13 Binary labels indicating speech presence:
14
15 0: No speech (silence, pauses, or non-speech audio)
16 1: Speech present (any spoken content)
17
18 Labels are time-aligned with MEG recordings at millisecond precision.
19
20 TASK STRUCTURE
21 Input: MEG brain activity at time t
22 Output: Binary prediction of speech presence
23
24 DATASET STRUCTURE
25 Source: Single book (book 3) from LibriBrain dataset
26 Duration: Multiple hours of continuous recording
27 Train/test splits: A randomly selected chapter will act as the test split
28
29 EVALUATION METRICS
30 The primary evaluation metric is macro-F1 score, which balances performance across both classes (speech
   and non-speech):

```

```

31
32 Macro-F1: Average of F1 scores for both classes (speech and non-speech)
33
34 F1_positive = 2 TP / (2 TP + FP + FN)
35 F1_negative = 2 TN / (2 TN + FN + FP)
36 Macro-F1 = 0.5 (F1_positive + F1_negative)
37
38 Threshold optimization: The classification threshold is tuned to maximize macro-F1 on the validation set
39
40 Candidate thresholds derived from precision-recall curve
41 Best threshold selected via vectorized evaluation across all candidates
42 Same threshold applied during test evaluation
43
44 This metric ensures balanced performance on both speech detection and silence/non-speech detection,
    preventing models from simply predicting the majority class.

```

#### DATASET: LIBRIBRAINSHERLOCK4

```

1 DESCRIPTION
2 LibriBrainSherlock4 is a neural decoding task using MEG recordings from a single participant listening to
    one complete audiobook. The objective is to detect whether speech is present at each time point
    based solely on brain activity patterns. This binary classification task serves as a foundational
    benchmark for brain-to-speech decoding, testing whether neural signals contain sufficient information
    to distinguish speech from silence or non-speech audio.
3
4 OBSERVATION SPACE
5 Each observation consists of:
6
7 MEG sensor recordings: Multi-channel time-series data from magnetometers and gradiometers
8 Temporal resolution: ~1ms sampling rate
9 Spatial coverage: Full-head sensor array
10 Feature options: Raw signals, filtered bands, or extracted features
11
12 TARGET SPACE
13 Binary labels indicating speech presence:
14
15 0: No speech (silence, pauses, or non-speech audio)
16 1: Speech present (any spoken content)
17
18 Labels are time-aligned with MEG recordings at millisecond precision.
19
20 TASK STRUCTURE
21 Input: MEG brain activity at time t
22 Output: Binary prediction of speech presence
23
24 DATASET STRUCTURE
25 Source: Single book (book 4) from LibriBrain dataset
26 Duration: Multiple hours of continuous recording
27 Train/test splits: A randomly selected chapter will act as the test split
28
29 EVALUATION METRICS
30 The primary evaluation metric is macro-F1 score, which balances performance across both classes (speech
    and non-speech):
31
32 Macro-F1: Average of F1 scores for both classes (speech and non-speech)
33
34 F1_positive = 2 TP / (2 TP + FP + FN)
35 F1_negative = 2 TN / (2 TN + FN + FP)
36 Macro-F1 = 0.5 (F1_positive + F1_negative)
37
38 Threshold optimization: The classification threshold is tuned to maximize macro-F1 on the validation set
39

```

```

40 Candidate thresholds derived from precision-recall curve
41 Best threshold selected via vectorized evaluation across all candidates
42 Same threshold applied during test evaluation
43
44 This metric ensures balanced performance on both speech detection and silence/non-speech detection,
    preventing models from simply predicting the majority class.

```

## DATASET: LIBRIBRAINSHERLOCK5

```

1 DESCRIPTION
2 LibriBrainSherlock5 is a neural decoding task using MEG recordings from a single participant listening to
    one complete audiobook. The objective is to detect whether speech is present at each time point
    based solely on brain activity patterns. This binary classification task serves as a foundational
    benchmark for brain-to-speech decoding, testing whether neural signals contain sufficient information
    to distinguish speech from silence or non-speech audio.

3
4 OBSERVATION SPACE
5 Each observation consists of:
6
7 MEG sensor recordings: Multi-channel time-series data from magnetometers and gradiometers
8 Temporal resolution: ~1ms sampling rate
9 Spatial coverage: Full-head sensor array
10 Feature options: Raw signals, filtered bands, or extracted features
11
12 TARGET SPACE
13 Binary labels indicating speech presence:
14
15 0: No speech (silence, pauses, or non-speech audio)
16 1: Speech present (any spoken content)
17
18 Labels are time-aligned with MEG recordings at millisecond precision.
19
20 TASK STRUCTURE
21 Input: MEG brain activity at time t
22 Output: Binary prediction of speech presence
23
24 DATASET STRUCTURE
25 Source: Single book (book 5) from LibriBrain dataset
26 Duration: Multiple hours of continuous recording
27 Train/test splits: A randomly selected chapter will act as the test split
28
29 EVALUATION METRICS
30 The primary evaluation metric is macro-F1 score, which balances performance across both classes (speech
    and non-speech):
31
32 Macro-F1: Average of F1 scores for both classes (speech and non-speech)
33
34 F1_positive = 2 * TP / (2 * TP + FP + FN)
35 F1_negative = 2 * TN / (2 * TN + FN + FP)
36 Macro-F1 = 0.5 * (F1_positive + F1_negative)
37
38 Threshold optimization: The classification threshold is tuned to maximize macro-F1 on the validation set
39
40 Candidate thresholds derived from precision-recall curve
41 Best threshold selected via vectorized evaluation across all candidates
42 Same threshold applied during test evaluation
43
44 This metric ensures balanced performance on both speech detection and silence/non-speech detection,
    preventing models from simply predicting the majority class.

```

## DATASET: LIBRIBRAINSHERLOCK6

```

1 DESCRIPTION
2 LibriBrainSherlock6 is a neural decoding task using MEG recordings from a single participant listening to
   one complete audiobook. The objective is to detect whether speech is present at each time point
   based solely on brain activity patterns. This binary classification task serves as a foundational
   benchmark for brain-to-speech decoding, testing whether neural signals contain sufficient information
   to distinguish speech from silence or non-speech audio.

3
4 OBSERVATION SPACE
5 Each observation consists of:
6
7 MEG sensor recordings: Multi-channel time-series data from magnetometers and gradiometers
8 Temporal resolution: ~1ms sampling rate
9 Spatial coverage: Full-head sensor array
10 Feature options: Raw signals, filtered bands, or extracted features

11
12 TARGET SPACE
13 Binary labels indicating speech presence:
14
15 0: No speech (silence, pauses, or non-speech audio)
16 1: Speech present (any spoken content)
17
18 Labels are time-aligned with MEG recordings at millisecond precision.

19
20 TASK STRUCTURE
21 Input: MEG brain activity at time t
22 Output: Binary prediction of speech presence

23
24 DATASET STRUCTURE
25 Source: Single book (book 6) from LibriBrain dataset
26 Duration: Multiple hours of continuous recording
27 Train/test splits: A randomly selected chapter will act as the test split

28
29 EVALUATION METRICS
30 The primary evaluation metric is macro-F1 score, which balances performance across both classes (speech
   and non-speech):
31
32 Macro-F1: Average of F1 scores for both classes (speech and non-speech)
33
34 F1_positive = 2 * TP / (2 * TP + FP + FN)
35 F1_negative = 2 * TN / (2 * TN + FN + FP)
36 Macro-F1 = 0.5 * (F1_positive + F1_negative)

37
38 Threshold optimization: The classification threshold is tuned to maximize macro-F1 on the validation set
39
40 Candidate thresholds derived from precision-recall curve
41 Best threshold selected via vectorized evaluation across all candidates
42 Same threshold applied during test evaluation
43
44 This metric ensures balanced performance on both speech detection and silence/non-speech detection,
   preventing models from simply predicting the majority class.

```

## DATASET: LIBRIBRAINSHERLOCK7

```

1 DESCRIPTION
2 LibriBrainSherlock7 is a neural decoding task using MEG recordings from a single participant listening to
   one complete audiobook. The objective is to detect whether speech is present at each time point
   based solely on brain activity patterns. This binary classification task serves as a foundational
   benchmark for brain-to-speech decoding, testing whether neural signals contain sufficient information
   to distinguish speech from silence or non-speech audio.

3
4 OBSERVATION SPACE
5 Each observation consists of:

```

```

6
7 MEG sensor recordings: Multi-channel time-series data from magnetometers and gradiometers
8 Temporal resolution: ~1ms sampling rate
9 Spatial coverage: Full-head sensor array
10 Feature options: Raw signals, filtered bands, or extracted features
11
12 TARGET SPACE
13 Binary labels indicating speech presence:
14
15 0: No speech (silence, pauses, or non-speech audio)
16 1: Speech present (any spoken content)
17
18 Labels are time-aligned with MEG recordings at millisecond precision.
19
20 TASK STRUCTURE
21 Input: MEG brain activity at time t
22 Output: Binary prediction of speech presence
23
24 DATASET STRUCTURE
25 Source: Single book (book 7) from LibriBrain dataset
26 Duration: Multiple hours of continuous recording
27 Train/test splits: A randomly selected chapter will act as the test split
28
29 EVALUATION METRICS
30 The primary evaluation metric is macro-F1 score, which balances performance across both classes (speech
   and non-speech):
31
32 Macro-F1: Average of F1 scores for both classes (speech and non-speech)
33
34 F1_positive = 2 * TP / (2 * TP + FP + FN)
35 F1_negative = 2 * TN / (2 * TN + FN + FP)
36 Macro-F1 = 0.5 * (F1_positive + F1_negative)
37
38 Threshold optimization: The classification threshold is tuned to maximize macro-F1 on the validation set
39
40 Candidate thresholds derived from precision-recall curve
41 Best threshold selected via vectorized evaluation across all candidates
42 Same threshold applied during test evaluation
43
44 This metric ensures balanced performance on both speech detection and silence/non-speech detection,
   preventing models from simply predicting the majority class.

```

## OPTIM PROMPT

1 You should change the optimizer, which can be found in `optim.py`. In deep learning, optimization is used to descend the gradient of a loss function with respect to the parameters of a neural network. The name of the function should be `create\_optimizer`, and this function should return an `Optimizer` with functions including `zero\_grad` and `step`. Besides this, you are allowed to make whatever changes you like.

## LOSS PROMPT

1 You should change the loss, which can be found in `loss.py`. In deep learning, the loss provides an objective to minimize; You should not change the name of the function, `compute\_loss`, or its inputs. This function should return a scalar loss value.

## NETWORK PROMPTS

1 You should change the network, which can be found in `networks.py`. In deep learning, the network provides the architecture of the model, and specifies the number of parameters, width of layers, and type of connections that those layers should have. You should not change the name or interface of the function `Model`, or its inputs.

### M.3. Computer Vision Classification

#### DOMAIN DESCRIPTION

1 Image classification is a branch of computer vision focused on training models to categorize visual inputs into predefined classes. A digital image is a grid of pixels, where each pixel encodes color information as numerical values. These values are stored in binary format in computers, representing the intensity of color channels (typically red, green, and blue for RGB images). Unlike tasks that require understanding spatial relationships or generating new content, image classification distills an entire image into a single categorical label.

2 Historically, image classification draws from pattern recognition and statistical learning theory developed in the mid-20th century. Early approaches relied on hand-crafted features-such as edges, textures, and color histograms-combined with classical machine learning algorithms. The field was revolutionized in 2012 when deep convolutional neural networks (CNNs) achieved breakthrough performance on the ImageNet dataset, demonstrating that hierarchical learned features could dramatically outperform engineered ones. The image-to-label pipeline-where a model receives pixel data as input and outputs class probabilities-remains central to modern approaches.

4 The objective of image classification is to learn a function that maps raw pixel values to class labels with high accuracy. Models achieve this by learning hierarchical representations: early layers detect low-level features like edges and colors, while deeper layers combine these into complex patterns representing objects, textures, and semantic concepts. Success in image classification requires sufficient training data, appropriate model architecture, and techniques to handle challenges like class imbalance, domain shift, and adversarial perturbations.

6 In practice, image classification serves as a foundation for numerous applications including medical diagnosis, autonomous vehicles, content moderation, and agricultural monitoring. Understanding the properties of image data-resolution, color spaces, and dataset statistics-is essential before implementing a classification system, as these factors influence model design and performance.

8 Below, we provide descriptions of the datasets which you will be training on. However, be aware that any code you develop may be applied to other image classification datasets too.

#### DATASET: CIFAR10

1 DESCRIPTION

2 CIFAR-10 is a widely-used image classification benchmark consisting of 60,000 color images across 10 common object categories. Each image is 32x32 pixels in RGB format. The dataset provides a balanced training set and a carefully curated test set for evaluating classification performance on natural images.

3

4 OBSERVATION SPACE

5 Each observation is a 32x32 RGB image with pixel values in the range [0, 255], representing one of 10 object classes.

6

7 CLASSES

8 The dataset contains 10 classes:

9

10 0: airplane

11 1: automobile

12 2: bird

13 3: cat

14 4: deer

15 5: dog

```

16 6: frog
17 7: horse
18 8: ship
19 9: truck
20
21 DATASET STRUCTURE
22 Training set: 50,000 images (5,000 per class)
23 Test set: 10,000 images (1,000 per class)
24 The training data is divided into 5 batches of 10,000 images each
25 Test set contains exactly 1,000 randomly-selected images per class

```

## DATASET: CIFAR100

```

1 DESCRIPTION
2 CIFAR-100 is an extension of CIFAR-10 with 100 fine-grained classes, providing a more challenging
   classification benchmark. Each image is 32x32 pixels in RGB format. The classes are grouped into 20
   superclasses for hierarchical classification tasks, with 5 fine classes per superclass.
3
4 OBSERVATION SPACE
5 Each observation is a 32x32 RGB image with pixel values in the range [0, 255], representing one of 100
   classes.
6
7 CLASSES
8 The dataset contains 100 fine-grained classes organized into 20 superclasses:
9
10 Aquatic mammals: beaver, dolphin, otter, seal, whale
11 Fish: aquarium fish, flatfish, ray, shark, trout
12 Flowers: orchids, poppies, roses, sunflowers, tulips
13 Food containers: bottles, bowls, cans, cups, plates
14 Fruit and vegetables: apples, mushrooms, oranges, pears, sweet peppers
15 Household electrical devices: clock, keyboard, lamp, telephone, television
16 Household furniture: bed, chair, couch, table, wardrobe
17 Insects: bee, beetle, butterfly, caterpillar, cockroach
18 Large carnivores: bear, leopard, lion, tiger, wolf
19 Large man-made outdoor things: bridge, castle, house, road, skyscraper
20 Large natural outdoor scenes: cloud, forest, mountain, plain, sea
21 Large omnivores and herbivores: camel, cattle, chimpanzee, elephant, kangaroo
22 Medium-sized mammals: fox, porcupine, possum, raccoon, skunk
23 Non-insect invertebrates: crab, lobster, snail, spider, worm
24 People: baby, boy, girl, man, woman
25 Reptiles: crocodile, dinosaur, lizard, snake, turtle
26 Small mammals: hamster, mouse, rabbit, shrew, squirrel
27 Trees: maple, oak, palm, pine, willow
28 Vehicles 1: bicycle, bus, motorcycle, pickup truck, train
29 Vehicles 2: lawn mower, rocket, streetcar, tank, tractor
30
31 DATASET STRUCTURE
32 Training set: 50,000 images (500 per class)
33 Test set: 10,000 images (100 per class)

```

## DATASET: CIFAR10C

```

1 DESCRIPTION
2 CIFAR-10-C is a robustness benchmark derived from CIFAR-10, designed to evaluate model performance under
   distribution shift. The dataset systematically applies 19 types of common corruptions to the original
   CIFAR-10 test images at 5 different severity levels, creating realistic perturbations that models
   may encounter in deployment.
3
4 OBSERVATION SPACE
5 Each observation is a 32x32 RGB image that has been corrupted, with the same class labels as CIFAR-10.

```

```

6
7 CORRUPTION TYPES
8 The dataset includes 19 corruption categories:
9
10 Noise: Gaussian, shot, impulse
11 Blur: defocus, glass, motion, zoom
12 Weather: snow, frost, fog, brightness
13 Digital: contrast, elastic, pixelate, JPEG compression, saturate, spatter
14
15 DATASET STRUCTURE
16 Total images: 950,000 (10,000 base images 19 corruptions 5 severity levels)
17 Severity levels: 1 (mild) to 5 (severe)
18 Same 10 classes as CIFAR-10

```

## DATASET: CIFAR10LT

```

1 DESCRIPTION
2 CIFAR-10-LT is a long-tailed variant of CIFAR-10 designed to study learning under class imbalance. The
   training set exhibits exponential decay in samples per class, simulating real-world scenarios where
   some categories have significantly more data than others. The test set remains balanced to evaluate
   performance across all classes.
3
4 OBSERVATION SPACE
5 Each observation is a 32x32 RGB image with pixel values in the range [0, 255].
6
7 CLASSES
8 The same 10 classes as CIFAR-10, but with imbalanced training distribution.
9
10 DATASET STRUCTURE
11 Training set: fewer than 50,000 images with exponentially decreasing samples per class
12 Test set: 10,000 balanced images (1,000 per class)
13 Imbalance ratio varies based on configuration (typical ratios: 10, 50, or 100)

```

## DATASET: FASHIONMNIST

```

1 DESCRIPTION
2 Fashion-MNIST is a drop-in replacement for the classic MNIST dataset, consisting of grayscale images of
   fashion items. Designed as a more challenging alternative to handwritten digits, it maintains the
   same data format and split structure, making it ideal for benchmarking machine learning algorithms on
   a more complex task while keeping computational requirements modest.
3
4 OBSERVATION SPACE
5 Each observation is a 28x28 grayscale image with pixel values in the range [0, 255], where higher values
   represent darker pixels. Total of 784 pixels per image.
6
7 CLASSES
8 The dataset contains 10 fashion item categories:
9
10 0: T-shirt/top
11 1: Trouser
12 2: Pullover
13 3: Dress
14 4: Coat
15 5: Sandal
16 6: Shirt
17 7: Sneaker
18 8: Bag
19 9: Ankle boot
20
21 DATASET STRUCTURE

```

```
22 Training set: 60,000 images (6,000 per class)
23 Test set: 10,000 images (1,000 per class)
```

## DATASET: MNIST

```
1 DESCRIPTION
2 MNIST is the foundational handwritten digit recognition dataset, consisting of grayscale images extracted
   from two NIST databases. The digits were written by Census Bureau employees and high school students
   , providing natural variation in writing styles. Despite its age, MNIST remains a standard benchmark
   for validating machine learning implementations.
3
4 OBSERVATION SPACE
5 Each observation is a 28x28 grayscale image representing a handwritten digit (0-9), with pixel values
   typically normalized to [0, 1] or kept in [0, 255] range.
6
7 CLASSES
8 The dataset contains 10 digit classes (0-9), with one class per digit.
9
10 DATASET STRUCTURE
11 Training set: 60,000 images (6,000 per class)
12 Test set: 10,000 images (1,000 per class)
13 Balanced distribution: each class contains exactly 7,000 images total
14 Equal representation of Census Bureau employees and high school students in both splits
```

## DATASET: OXFORDFLOWERS

```
1 DESCRIPTION
2 Oxford Flowers 102 is a fine-grained visual classification dataset featuring 102 flower species commonly
   found in the United Kingdom. The dataset presents significant challenges due to large variations in
   scale, pose, lighting, and high inter-class similarity. It is widely used for evaluating fine-grained
   recognition and transfer learning approaches.
3
4 OBSERVATION SPACE
5 Each observation is a color image of variable size containing one or more flowers. Images exhibit natural
   variation in composition, background, and lighting conditions.
6
7 CLASSES
8 The dataset contains 102 flower categories with high visual similarity between some species.
9
10 DATASET STRUCTURE
11 Total images: between 4,080 and 26,316 (40-258 images per class)
12 Class distribution is imbalanced
13 Images feature large within-class variation and between-class similarity
14 Typical splits: training, validation, and test sets provided
```

## DATASET: STANFORDCARS

```
1 DESCRIPTION
2 Stanford Cars is a fine-grained vehicle classification dataset containing images of cars with class
   labels defined at the Make-Model-Year level. The dataset captures significant intra-class variation
   and inter-class similarity, making it challenging for distinguishing between visually similar car
   models and years.
3
4 OBSERVATION SPACE
5 Each observation is a color image of variable size showing a car, typically from various angles and in
   different settings.
6
7 CLASSES
```

```
8 The dataset contains 196 classes representing different car make-model-year combinations (e.g., 2012
9 Tesla Model S, 2012 BMW M3 coupe).
10 DATASET STRUCTURE
11 Training set: 8,144 images
12 Test set: 8,041 images
13 Approximately 50-50 split between training and testing within each class
14 Total: 16,185 images
```

## DATASET: TINYIMAGENET

```
1 DESCRIPTION
2 Tiny ImageNet is a subset of the ImageNet dataset designed for faster experimentation while maintaining
3 classification complexity. It consists of 200 classes from the original ImageNet, with images
4 downsampled to 64x64 pixels. The dataset provides a middle ground between small-scale datasets like
5 CIFAR and full-scale ImageNet.
6
7 OBSERVATION SPACE
8 Each observation is a 64x64x3 RGB color image representing one of 200 object classes.
9
10 CLASSES
11 The dataset contains 200 classes selected from ImageNet, covering diverse object categories.
12
13 DATASET STRUCTURE
14 Training set: 100,000 images (500 per class)
15 Validation set: 10,000 images (50 per class)
16 Test set: 10,000 images (50 per class)
17 Total: 120,000 images
```

## OPTIM PROMPT

```
1 You should change the optimizer, which can be found in `optim.py`. In deep learning, optimization is used
2 to descend the gradient of a loss function with respect to the parameters of a neural network. The
3 name of the function should be `create_optimizer`, and this function should return an `Optimizer`
4 with functions including `zero_grad` and `step`. Besides this, you are allowed to make whatever
5 changes you like.
```

## LOSS PROMPT

```
1 You should change the loss, which can be found in `loss.py`. In deep learning, the loss provides an
2 objective to minimize; You should not change the name of the function, `compute_loss`, or its inputs.
3 This function should return a scalar loss value.
```

## NETWORK PROMPTS

```
1 You should change the network, which can be found in `networks.py`. In deep learning, the network
2 provides the architecture of the model, and specifies the number of parameters, width of layers, and
3 type of connections that those layers should have. You should not change the name or interface of the
4 function `Model`, or its inputs.
```

## PREPROCESS PROMPTS

```
1 You should change the image preprocessing, which can be found in `preprocess.py`. Image processing
2 involves the manipulation of digital images to enhance them, reshape them, or extract useful
```

information. In image classification, it is crucial to have strong preprocessing which makes it easier for learned classifiers to identify and categorize images, instead of working purely with raw images.. You should not change the name or interface of the function `build\_transforms`, and the function should output two transformers for training set and test set, respectively.

## M.4. Continual Learning

### DOMAIN DESCRIPTION

1 This task evaluates continual learning for image classification under non-stationary task sequences (e.g., class- or task-incremental). The harness fixes data loading, model interface, and training loop; LLM-editable modules are regularization, replay, sampling, scheduling, and optimizer. Training proceeds sequentially over tasks without revisiting prior task data (except via replay). We report per-task accuracy, average accuracy, and forgetting/Backward Transfer from the final evaluation matrix. The default backend uses a standard image backbone with a dynamic classifier head sized to all seen classes.

### DATASET: PERMUTEDMNIST

1 DESCRIPTION  
 2 PermutedMNIST is a continual learning benchmark that creates multiple tasks from the MNIST dataset by applying fixed, deterministic pixel permutations. Each task uses the same 10-digit classes (0-9) but reorders the 2828 pixel grid according to a unique permutation derived from a base seed. This generates a sequence of tasks with identical label semantics but different input distributions, making it ideal for evaluating continual learning algorithms' ability to handle distribution shift without forgetting previously learned knowledge.  
 3  
 4 TASK STRUCTURE  
 5 Base dataset: MNIST (2828 grayscale images)  
 6 Number of tasks: Configurable (typically 10-20 tasks)  
 7 Classes per task: 10 (digits 0-9, consistent across all tasks)  
 8 Permutation: Unique, deterministic pixel reordering per task based on seed  
 9 Task identity: Not provided at test time (unless explicitly stated)  
 10  
 11 OBSERVATION SPACE  
 12 Each observation is a permuted 2828 grayscale image (784 pixels total) with pixel values in the range [0, 255] or normalized to [0, 1].  
 13  
 14 EVALUATION  
 15 The benchmark tests continual learning through several metrics:  
 16  
 17 Per-task accuracy: Classification accuracy on each individual task after training  
 18 Average accuracy: Mean accuracy across all tasks after sequential training  
 19 Forgetting: Degradation in performance on earlier tasks after learning new ones  
 20 Forward transfer: Ability to leverage prior knowledge for new tasks  
 21 Backward transfer: Improvement on past tasks from learning new ones  
 22  
 23 CHARACTERISTICS  
 24 Same label space across all tasks (no class incremental component)  
 25 Distribution shift through input transformation only  
 26 No correlation between task permutations (unless designed)  
 27 Tests pure robustness to catastrophic forgetting  
 28 Domain-incremental continual learning scenario

### DATASET: SPLITCIFAR100

1 DESCRIPTION

```

2 SplitCIFAR100 is a class-incremental continual learning benchmark that partitions the 100 classes of
CIFAR-100 into disjoint subsets, with each subset forming a separate task. The class assignment to
tasks is deterministic based on a seed, creating a sequence where the model must learn new classes
without forgetting previously learned ones. This benchmark tests class-incremental learning where
both the input distribution and the set of relevant classes change over time.
3
4 TASK STRUCTURE
5 Base dataset: CIFAR-100 (3232 RGB images, 100 classes)
6 Number of tasks: Configurable (commonly 10 tasks 10 classes each)
7 Classes per task: Disjoint subsets (e.g., 10, 20, or 50 classes per task)
8 Class order: Deterministic based on seed
9 Total classes: 100 across all tasks
10
11 OBSERVATION SPACE
12 Each observation is a 32323 RGB image from CIFAR-100, with classes divided among tasks.
13
14 EVALUATION
15 Standard continual learning metrics include:
16
17 Per-task accuracy: Performance on each task's classes after training
18 Average accuracy: Mean accuracy across all seen classes
19 Forgetting: Performance drop on earlier tasks' classes
20 Incremental accuracy curve: Accuracy trajectory as tasks are learned sequentially
21
22 CHARACTERISTICS
23 Class-incremental scenario (new classes introduced per task)
24 Disjoint class sets prevent overlap between tasks
25 Fixed input distribution (CIFAR-100) across tasks
26 Tests ability to expand classifier without catastrophic forgetting
27 No task boundaries provided at test time (unified classification)

```

## DATASET: TINYIMAGENETSPLIT

```

1 DESCRIPTION
2 TinyImageNetSplit is a class-incremental continual learning benchmark constructed from Tiny ImageNet by
partitioning its 200 classes into disjoint task-specific groups. With higher resolution (6464) and
more classes than CIFAR-based benchmarks, it provides a more challenging testbed for continual
learning algorithms operating on natural images with greater visual complexity and diversity.
3
4 TASK STRUCTURE
5 Base dataset: Tiny ImageNet (6464 RGB images, 200 classes)
6 Number of tasks: Configurable (commonly 10 tasks 20 classes each)
7 Classes per task: Disjoint subsets (e.g., 10, 20, or 40 classes per task)
8 Image resolution: 64643 RGB
9 Total classes: 200 across all tasks
10
11 OBSERVATION SPACE
12 Each observation is a 64643 RGB image representing one of 200 object classes from ImageNet, divided among
sequential tasks.
13
14 EVALUATION
15 Standard continual learning evaluation includes:
16
17 Per-task accuracy: Classification accuracy on each task's class subset
18 Average accuracy: Mean performance across all learned classes
19 Forgetting metric: Degradation on earlier tasks after learning new ones
20 Final accuracy: Performance on all 200 classes after sequential training
21
22 CHARACTERISTICS
23 Class-incremental learning with disjoint class sets
24 Higher resolution than CIFAR-based benchmarks (6464 vs 3232)
25 Larger number of classes (200 total)

```

- 26 Greater visual complexity and within-class variation
- 27 Tests scalability of continual learning to more realistic scenarios
- 28 Unified classification across all tasks at test time

#### REGULARIZER PROMPT

- 1 Implement a consolidation penalty to mitigate catastrophic forgetting across tasks.
- 2 - Maintain any needed task-wise statistics in `on_task_start/on_task_end`.
- 3 - `compute_penalty(model, step)` must be fast; avoid full-dataset passes.
- 4 - Examples: EWC-style quadratic penalty, L2 pull toward previous parameters, or Fisher/diagonal approximations.

#### REPLAY PROMPT

- 1 Implement a replay buffer with reservoir sampling supporting `add/sample/size`.
- 2 - Store `(x, y, task_id)` triplets; keep memory bounded.
- 3 - `sample(batch)` should mix current data with buffer according to a ratio.
- 4 - Deterministic behavior under fixed seeds is preferred.

#### SAMPLER PROMPT

- 1 Mix the current batch with replay data at a fixed ratio.
- 2 - Handle empty buffer gracefully (fall back to current-only).
- 3 - Return a dict with keys: `x (Tensor)`, `y (Tensor)`, `task_id (Tensor)`.
- 4 - Keep batch size consistent with `cfg["training"]["batch_size"]`.

#### SCHEDULER PROMPT

- 1 Build a step-based LR scheduler with warmup plus cosine or step decay.
- 2 - Return an object exposing `.step()` per update; no metric dependence.
- 3 - For cosine: linear warmup to base LR, then cosine to zero.
- 4 - For step: linear warmup, then decay LR by gamma every `step_size`.

#### OPTIMIZER PROMPT

- 1 Build a torch optimizer from config.
- 2 - Support `{"name": "sgd"|"adam"|"adamw"}`; use momentum/nesterov for SGD; betas/eps for Adam/AdamW.
- 3 - Respect `weight_decay` and `lr`; return a ready-to-use optimizer over model parameters.

### M.5. Greenhouse Gas Prediction

#### DOMAIN DESCRIPTION

- 1 Greenhouse gas forecasting is a branch of time series analysis focused on predicting future atmospheric concentrations of climate-relevant gases based on historical measurements. Unlike static prediction tasks, forecasting requires models to extrapolate beyond observed data, capturing both long-term trends and periodic patterns to make accurate predictions about future states. The challenge lies in learning from limited historical records to project concentrations years or decades ahead, requiring knowledge of the systems and potentially leveraging inductive biases when developing the model.
- 2
- 3 Historically, atmospheric monitoring began in the late 1950s with pioneering measurements at remote observatories designed to track changes in Earth's atmosphere away from local pollution sources.

Early analysis relied on simple trend fitting and seasonal decomposition methods. The field evolved through classical time series techniques in the 1970s-1980s (ARIMA, exponential smoothing) to more sophisticated statistical models incorporating domain knowledge about atmospheric processes. The core task of observing historical concentrations, identifying patterns, and projecting future values remains fundamental to climate science and policy planning.

- 4
- 5 The objective of greenhouse gas forecasting is to learn a function that maps temporal features to future concentration values with minimal prediction error. Methods vary from classical statistical approaches that explicitly model trend and seasonality components, to machine learning methods that learn representations directly from data, to hybrid approaches that combine both paradigms. Success in forecasting requires balancing model complexity with available data, capturing both secular trends and cyclical patterns, and appropriately quantifying uncertainty in long-range predictions.
- 6
- 7 In practice, greenhouse gas forecasting supports climate modeling, policy evaluation, emissions monitoring, and scientific understanding of atmospheric processes. The choice of modeling approach should reflect the characteristics of the data-including temporal resolution, length of historical record, strength of seasonal cycles, and smoothness of trends-as these factors fundamentally shape what patterns can be learned and how far into the future accurate predictions remain possible.
- 8
- 9 Below, we provide descriptions of the datasets (time series of atmospheric gas concentrations) that you can use to develop forecasting algorithms. While you will train on specific gases with known temporal patterns, be aware that any code you develop may be applied to other atmospheric time series, and you will be assessed on generalization to held-out datasets-hence, ensure that your approach is adaptable to different temporal characteristics and forecast horizons.

## DATASET: CH4

- 1 DESCRIPTION
- 2 This dataset contains monthly methane (CH4) concentration measurements from NOAA's Global Monitoring Laboratory network, representing globally-averaged atmospheric levels from 1983 to 2014. Methane is a potent greenhouse gas with both natural and anthropogenic sources, including wetlands, agriculture, fossil fuel extraction, and waste decomposition. The task is to forecast future CH4 concentrations from 2015 to 2025 based on historical patterns.
- 3
- 4 OBSERVATION SPACE
- 5 Each observation is a time-stamped measurement with the following features:
- 6
- 7 Day of month: Approximated as 15 for all monthly measurements
- 8 Month: Integer from 1-12
- 9 Year: 1983-2014 (training period)
- 10 Day of year: Approximated from month (e.g., January 15, February 46)
- 11 CH4 concentration: Parts per billion (ppb)
- 12
- 13 Data format: NumPy array of shape (N, 5) where N is the number of monthly observations.
- 14
- 15 TARGET SPACE
- 16 Continuous prediction of CH4 concentration in parts per billion (ppb) for future time points.
- 17
- 18 TEMPORAL STRUCTURE
- 19 Training period: January 1983 - December 2014 (32 years)
- 20 Evaluation period: January 2015 - December 2025 (11 years)
- 21 Sampling frequency: Monthly
- 22 Forecast horizon: Up to 132 months (11 years) into the future
- 23
- 24 EVALUATION METRICS
- 25 Models are evaluated using Mean Squared Error (MSE) on predictions for 2015-2025:
- 26
- 27  $MSE = (1/N) \sum (predicted - actual)^2$
- 28 Lower MSE indicates better forecasting performance
- 29 Penalizes large errors more heavily than small ones
- 30
- 31 TASK OBJECTIVE

32 Develop a time series forecasting model that captures both the long-term upward trend in atmospheric methane and any seasonal or cyclical patterns present in the data. The model must extrapolate 11 years beyond the training period while maintaining accuracy.

## DATASET: CO2

```
1 DESCRIPTION
2 This dataset contains daily carbon dioxide (CO2) concentration measurements from the Mauna Loa Observatory in Hawaii, spanning 1970 to 2010. Located at 3,200 meters above sea level on an isolated mid-Pacific volcano, Mauna Loa has provided the world's longest continuous record of atmospheric CO2 since 1958, making it the definitive benchmark for tracking global climate change. The task is to forecast daily CO2 levels from 2010 to 2025.
3
4 OBSERVATION SPACE
5 Each observation is a daily measurement with the following features:
6
7 Year: 1970-2010 (training period)
8 Month: Integer from 1-12
9 Day: Integer from 1-31
10 Fractional year: Precise decimal representation (e.g., 1974.3781)
11 CO2 concentration: Parts per million (ppm)
12
13 Data format: NumPy array of shape (N, 5) where N is the number of daily observations.
14
15 TARGET SPACE
16 Continuous prediction of CO2 concentration in parts per million (ppm) for future dates.
17
18 TEMPORAL STRUCTURE
19 Training period: 1970-2010 (40 years)
20 Evaluation period: 2010-2025 (15 years)
21 Sampling frequency: Daily
22 Forecast horizon: Up to ~5,475 days (15 years) into the future
23
24 EVALUATION METRICS
25 Models are evaluated using Mean Squared Error (MSE) on daily predictions for 2010-2025:
26
27 MSE = (1/N) (predicted - actual)2
28 Lower MSE indicates better forecasting accuracy
29
30 TASK OBJECTIVE
31 Develop a time series forecasting model that accurately captures both the secular trend in atmospheric CO2 and the regular seasonal oscillations. The model must extrapolate 15 years beyond training while maintaining daily-level precision, requiring understanding of both long-term forcing and periodic biological cycles.
```

## DATASET: N2O

```
1 DESCRIPTION
2 This dataset contains monthly nitrous oxide (N2O) concentration measurements from NOAA's Global Monitoring Laboratory network, representing globally-averaged atmospheric levels from 2001 to 2014. Nitrous oxide is a long-lived greenhouse gas and ozone-depleting substance with sources including agricultural fertilizers, industrial processes, and natural soil emissions. The task is to forecast N2O concentrations from 2015 to 2025.
3
4 OBSERVATION SPACE
5 Each observation is a time-stamped measurement with the following features:
6
7 Day of month: Approximated as 15 for all monthly measurements
8 Month: Integer from 1-12
9 Year: 2001-2014 (training period)
```

```

10 Day of year: Approximated from month
11 N2O concentration: Parts per billion (ppb)
12
13 Data format: NumPy array of shape (N, 5) where N is the number of monthly observations.
14
15 TARGET SPACE
16 Continuous prediction of N2O concentration in parts per billion (ppb) for future time points.
17
18 TEMPORAL STRUCTURE
19 Training period: January 2001 - December 2014 (14 years)
20 Evaluation period: January 2015 - December 2025 (11 years)
21 Sampling frequency: Monthly
22 Forecast horizon: Up to 132 months (11 years) into the future
23
24 EVALUATION METRICS
25 Models are evaluated using Mean Squared Error (MSE) on predictions for 2015-2025:
26
27 MSE = (1/N) (predicted - actual)2
28 Lower MSE indicates better forecasting performance
29
30 TASK OBJECTIVE
31 Develop a time series forecasting model that captures the long-term trend in atmospheric nitrous oxide
       and any seasonal or cyclical patterns. The model must extrapolate nearly as far into the future as
       the available training history spans, testing generalization beyond the observed regime.

```

## DATASET: SF6

```

1 DESCRIPTION
2 This dataset contains monthly sulfur hexafluoride (SF6) concentration measurements from NOAA's Global
   Monitoring Laboratory network, representing globally-averaged atmospheric levels from 1997 to 2014.
   SF6 is an entirely synthetic compound used as an electrical insulator in high-voltage equipment and
   has the highest global warming potential of any known substance. Its presence in the atmosphere is
   purely anthropogenic, making it a unique tracer of industrial activity. The task is to forecast SF6
   concentrations from 2015 to 2025.
3
4 OBSERVATION SPACE
5 Each observation is a time-stamped measurement with the following features:
6
7 Day of month: Approximated as 15 for all monthly measurements
8 Month: Integer from 1-12
9 Year: 1997-2014 (training period)
10 Day of year: Approximated from month
11 SF6 concentration: Parts per trillion (ppt)
12
13 Data format: NumPy array of shape (N, 5) where N is the number of monthly observations.
14
15 TARGET SPACE
16 Continuous prediction of SF6 concentration in parts per trillion (ppt) for future time points.
17
18 TEMPORAL STRUCTURE
19 Training period: January 1997 - December 2014 (18 years)
20 Evaluation period: January 2015 - December 2025 (11 years)
21 Sampling frequency: Monthly
22 Forecast horizon: Up to 132 months (11 years) into the future
23
24 EVALUATION METRICS
25 Models are evaluated using Mean Squared Error (MSE) on predictions for 2015-2025:
26
27 MSE = (1/N) (predicted - actual)2
28 Lower MSE indicates better forecasting performance
29
30 TASK OBJECTIVE

```

31 Develop a time series forecasting model that captures the long-term trend in atmospheric sulfur hexafluoride. Unlike other greenhouse gases, SF<sub>6</sub> exhibits primarily secular growth with minimal seasonal modulation, making trend extrapolation the primary challenge. The model must forecast 11 years into the future based on 18 years of historical data.

## MODEL PROMPT

1 You should change the model file, which can be found in `model.py`. Try and choose a model that is appropriate for the task.

## DATA PROCESSING PROMPT

1 You should change the data processing file, which can be found in `data\_processing.py`. Try and choose a data processing that is appropriate for the task.

## M.6. Language Modelling

### DOMAIN DESCRIPTION

1 # Language Modelling  
2  
3 Language modelling is a fundamental task in natural language processing that involves predicting the next token in a sequence of text. This task forms the foundation for many modern large language models (LLMs) and serves as a key benchmark for understanding how well neural networks can learn patterns in sequential data.  
4  
5 In this task, you will work with autoregressive transformer models trained on text corpora. The goal is to minimize perplexity (or equivalently, cross-entropy loss) on held-out validation data. The task provides opportunities to experiment with different architectures, optimization strategies, and loss functions to improve language model performance.

### DATASET: LMFINEWEB

1 DESCRIPTION  
2 LMFineWeb (10B variant) is a curated web text corpus derived from the larger FineWeb dataset by Hugging Face. The dataset prioritizes quality through aggressive filtering of Common Crawl data, removing low-quality content, duplicates, and non-English text. This 10B token variant represents a substantial downsample of the full dataset, designed specifically for rapid experimentation and iteration during language model development.  
3  
4 CONTENT  
5 The dataset consists of cleaned web pages spanning diverse topics and domains, with each example containing:  
6  
7 High-quality natural language text  
8 Metadata including source URLs and filtering scores  
9 Multi-sentence coherent passages suitable for language modeling  
10  
11 DATASET STRUCTURE  
12 Training split: Approximately 10 billion tokens  
13 Downsampled from the full FineWeb corpus for efficiency  
14 Preserves the quality distribution of the parent dataset  
15 Suitable for pretraining small to medium-sized language models  
16  
17 PREPROCESSING  
18 Extensive deduplication at document and paragraph levels

- 19 Language filtering to retain primarily English content
- 20 Quality filtering based on heuristics and model-based scoring
- 21 Removal of adult content, spam, and machine-generated text

#### DATASET: OPCFINEWEBCODE

```
1 DESCRIPTION
2 OPC-FineWeb-Code is a specialized corpus of code-related web content extracted from FineWeb using
   fastText-based retrieval in iterative rounds. Originally developed for the OpenCoder pretraining
   pipeline, this dataset focuses on web pages containing programming tutorials, documentation, code
   snippets, and technical discussions. The implementation uses the first 20% of the full dataset (~30GB
   ) to balance corpus size with training efficiency.
3
4 CONTENT
5 The dataset consists of code-centric web pages including:
6
7 Programming tutorials and documentation
8 Technical blog posts with code examples
9 Stack Overflow discussions and solutions
10 GitHub repository documentation
11 Code snippets with natural language explanations
12
13 DATASET STRUCTURE
14 Training split: ~80% of the 20% subset (original dataset is ~150GB total)
15 Validation split: 1% of data (created during preprocessing)
16 Test split: Remaining ~19% of the 20% subset
17 Approximately 30GB total size after downsampling
18 Multiple programming languages and frameworks represented
19
20 PREPROCESSING
21 fastText-based retrieval to identify code-relevant content
22 Automatic skipping of corrupted or unsafe files
23 Validation split created from training data (not present in original)
24 Filtered for code density and technical content quality
```

#### DATASET: OPCFINEWEBMATH

```
1 DESCRIPTION
2 OPC-FineWeb-Math is a curated collection of mathematics-related web content recalled from FineWeb through
   iterative fastText-based retrieval. Created as part of the OpenCoder data collection effort, this
   dataset provides high-quality mathematical text including proofs, explanations, problem-solving
   discussions, and educational content. The corpus is designed to enhance language models' capabilities
   in mathematical reasoning and technical communication.
3
4 CONTENT
5 The dataset consists of math-centric web pages including:
6
7 Mathematical explanations and tutorials
8 Problem-solving discussions and solutions
9 Academic and educational math content
10 LaTeX-formatted equations and proofs
11 Math-focused forums and Q&A platforms
12
13 DATASET STRUCTURE
14 Total rows: Approximately 5.24 million
15 Training split: 99% of data
16 Validation split: 1% of data (created during preprocessing)
17 Original dataset contains only training data
18
19 PREPROCESSING
```

```

20 fastText-based retrieval targeting mathematical content
21 Preservation of mathematical notation and formatting
22 Quality filtering for coherence and correctness
23 Validation split synthetically created from training data

```

## DATASET: TINYSTORIES

```

1 DESCRIPTION
2 TinyStories is a synthetic dataset of short stories generated by GPT-3.5 and GPT-4, created by Ronen
   Eldan and Yuanzhi Li. The stories are deliberately simple, coherent, and limited in vocabulary,
   making them ideal for training and evaluating small language models. This dataset enables research on
   whether models with constrained capacity can achieve fluency and coherence when trained on
   appropriately simplified data.

3
4 CONTENT
5 The dataset consists of LLM-generated short stories characterized by:
6
7 Simple vocabulary and sentence structures
8 Coherent narrative arcs
9 Age-appropriate content suitable for children
10 Consistent story formatting
11 Limited complexity to match small model capacity
12
13 DATASET STRUCTURE
14 Training examples: Multiple splits available
15 Validation split: Provided for model evaluation
16 Each example contains a complete short story
17 Stories vary in length but maintain simplicity throughout
18
19 DESIGN PHILOSOPHY
20 The dataset tests the hypothesis that small language models can achieve strong performance when trained
   on data matched to their representational capacity, rather than being trained on complex web text
   designed for much larger models. This makes TinyStories particularly valuable for:
21
22 Training small, efficient language models
23 Educational purposes and research on model scaling
24 Evaluating coherence and fluency in constrained settings
25 Rapid prototyping of language model architectures

```

## OPTIM PROMPT

```

1 You should change the optimizer file, which can be found in `optim.py`. In deep learning, optimization is
   used to descend the gradient of a loss function with respect to the parameters of a neural network.
   Your task is to implement the `OptimizerConfig` dataclass with appropriate hyperparameters and the
   `create_optimizers` function. This function should create and return a list of optimizers and a list
   of learning rate schedulers for training the language model. You may use different optimizers for
   different parts of the model (e.g., embedding layers, transformer blocks, output head) if desired.
   The function receives the model and config as inputs and must return two lists: `optimizers` and ``
   schedulers``.

```

## LOSS PROMPT

```

1 You should change the loss file, which can be found in `loss.py`. In language modeling, the loss provides
   an objective to minimize for next token prediction. Your task is to implement the `compute_loss`
   function that takes model logits of shape [batch_size, seq_len, vocab_size] and target tokens of
   shape [batch_size, seq_len], and returns a scalar loss value. Make sure to ignore padding tokens (where
   target is -1). You should not change the name of the function `compute_loss` or its input/
   output signature.

```

## NETWORKS PROMPT

- <sup>1</sup> You should change the network file, which can be found in `networks.py`. In deep learning, the network provides the architecture of the model and specifies the number of parameters, layer dimensions, and types of connections. Your task is to implement the `ModelConfig` dataclass with model hyperparameters and the `Model` class. The Model must have a `forward` method that takes input token indices of shape [batch\_size, seq\_len] and returns logits of shape [batch\_size, seq\_len, vocab\_size]. The Model must also have a `get\_config` method that returns the model configuration. You should not change the name or interface of these methods.

**M.7. Model Unlearning**

## DOMAIN DESCRIPTION

- <sup>1</sup> Machine unlearning for large language models (LLMs) is a technique focused on selectively removing the influence of specific training data from a model's learned representations, without requiring complete retraining. Unlike traditional model updating approaches that involve retraining from scratch, unlearning methods efficiently modify model weights or behaviors to "forget" targeted information—such as sensitive personal data, copyrighted content, or harmful knowledge—while preserving the model's overall capabilities and performance on unrelated tasks.
- <sup>2</sup>
- <sup>3</sup> The need for unlearning has emerged from practical concerns around data privacy, safety, and regulatory compliance. LLMs trained on massive datasets often inadvertently memorize sensitive information, copyrighted material, or harmful content. With ever-increasing costs of pre-training and post-training (often involving billions of tokens and substantial computational resources), retraining models from scratch in response to data deletion requests is prohibitively expensive. This has motivated the development of efficient post-training interventions that can selectively eliminate undesirable knowledge while maintaining model utility.
- <sup>4</sup>
- <sup>5</sup> The objective of LLM unlearning is twofold: (1) **Removal** – ensuring that the influence caused specifically by the "forget set" (the data to be removed) is substantially erased from the model, and (2) **Retention** – maintaining the model's utility and performance on unrelated downstream tasks and general capabilities. Formally, given an original model trained on data containing a forget set, the unlearning process yields an unlearned model where the forget set's influence is eliminated. The efficacy is assessed using evaluation metrics that quantify remaining influence (such as extraction strength, membership inference attacks, and truth ratio) while utility metrics ensure preserved performance on general tasks.
- <sup>6</sup>
- <sup>7</sup> Unlearning methods vary in their approach. While some are prompting-based, detecting sensitive queries at inference time and deploying obfuscation mechanisms, the most practical and scalable approaches directly modify model weights. These weight-modification techniques include: fine-tuning with tailored loss functions that encourage forgetting on the forget set while maintaining performance on a retain set; preference optimization methods that steer the model away from generating content related to the forget set; and representation-level interventions that modify internal activations. Success in unlearning requires careful balancing between thorough forgetting (avoiding any residual knowledge) and maintaining utility (preventing degradation of general capabilities).
- <sup>8</sup>
- <sup>9</sup> Below, we provide descriptions of the unlearning datasets and benchmarks which you will be working with. However, be aware that any methods you develop should ideally generalize to other unlearning scenarios and benchmarks as well.

## DATASET: MUSE

- <sup>1</sup> DESCRIPTION
- <sup>2</sup> The MUSE (Machine Unlearning of Sensitive Entities) News Benchmark is designed to evaluate machine unlearning capabilities in large language models (LLMs) with respect to copyrighted news content. This task specifically focuses on news articles from various reputable sources covering diverse topics, providing a realistic testbed for assessing whether models can selectively forget specific content while maintaining general knowledge and capabilities. The goal is to train the model to "unlearn" specific copyrighted news articles (forget set) while preserving knowledge of other news

```

1 content (retain set).
3
4 DATASET COMPOSITION
5 Training data from the MUSE-News dataset:
6
7 - Forget corpus: 889 news articles to be unlearned
8 - Retain corpus (retain1): 1,777 news articles to maintain model capabilities
9 - Format: Raw text passages for pretraining-style unlearning
10
11 EVALUATION DATASET COMPOSITION
12 Evaluation data from the MUSE-News dataset:
13
14 - Knowledge memorization (knowmem): 100 QA pairs from forget set (forget_qa split), 100 QA pairs from
     retain set (retain_qa split)
15 - Verbatim memorization (verbmem): Text completion tasks on forget set to test exact memorization
16
17 TASK OBJECTIVE
18 The primary goal is to implement an unlearning algorithm that satisfies:
19
20 1. Content Forgetting: Remove the model's ability to reproduce or recall specific details from the forget
     set
21 2. Knowledge Retention: Maintain strong performance on the retain set
22
23 EVALUATION METRICS
24 The task uses 5 evaluation metrics:
25
26 1. forget_knowmem_ROUGE: ROUGE-L F1 score on 100 forget QA pairs from forget_qa split (lower is better,
     indicates reduced knowledge recall)
27 2. retain_knowmem_ROUGE: ROUGE-L F1 score on 100 retain QA pairs from retain_qa split (higher is better,
     indicates preserved knowledge)
28 3. forget_verbmem_ROUGE: ROUGE-L F1 score on text completions from forget split testing verbatim
     memorization (lower is better, indicates reduced memorization)
29 4. privleak: Privacy leakage metric using min_k% probability on forget split (lower is better, indicates
     successful unlearning)
30 5. extraction_strength: Exact extraction capability measured on forget split verbatim memorization tasks
     (lower is better, indicates reduced memorization)
31
32 The goal is to optimize for all of them, from which a final score will be computed.

```

## DATASET: TOFU

```

1 DESCRIPTION
2 The TOFU (Task of Fictitious Unlearning) dataset is a benchmark designed to evaluate machine unlearning
   capabilities in large language models (LLMs). It consists of question-answer pairs derived from
   autobiographies of 200 entirely fictitious authors. The synthetic nature ensures the data is distinct
   from any pretraining corpus, providing a controlled environment for testing selective forgetting.
   The goal is to train the model to "unlearn" specific authors (forget set) while preserving its
   knowledge of other authors (retain set) and general world knowledge.
3
4 DATASET COMPOSITION
5 The TOFU dataset contains:
6
7 - 200 Fictitious Authors: Each with a complete fictional biography and associated QA pairs
8 - Question-Answer Pairs: Covering biographical details, writing styles, and life events of each author
   (20 QA pairs per author)
9 - Total Size: 4,000 QA pairs across all authors
10 - Format: Conversational QA format compatible with instruction-tuned models
11
12 TASK CONFIGURATION
13 This task uses the following splits from the TOFU dataset:
14
15 - forget10: 10% of the dataset (20 authors) - information to be unlearned

```

```

16 - retain90: 90% of the dataset (180 authors) - information to be preserved
17 - holdout10: 10% holdout set for evaluation purposes
18
19 TASK OBJECTIVE
20 The primary goal is to implement an unlearning algorithm that satisfies:
21
22 1. Selective Forgetting: Remove the model's ability to recall information about the 20 authors in the
   forget set
23 2. Knowledge Retention: Maintain strong performance on the 180 authors in the retain set
24
25 EVALUATION METRICS
26 The task uses 6 evaluation metrics:
27
28 1. forget_quality: KS test comparing truth ratio distributions (correct vs perturbed answers) on forget
   set (higher is better)
29 2. forget_Q_A_Prob: Probability assigned to correct answers on forget set (lower is better)
30 3. forget_Q_A_ROUGE: ROUGE-L recall between generated and correct answers on forget set (lower is better)
31 4. model_utility: Harmonic mean aggregate across 9 sub-metrics measuring retention on retain set, real
   authors, and world facts (higher is better)
32 - Retain set (180 authors): probability, ROUGE-L, truth ratio
33 - Real authors: normalized probability, ROUGE-L, truth ratio
34 - World facts: normalized probability, ROUGE-L, truth ratio
35 5. privleak: Membership inference attack vulnerability using min_k metric (lower is better)
36 6. extraction_strength: Measures model's ability to memorize/extract information from forget set (lower
   is better)
37
38 The goal is to optimize for all of them, from which a final score will be computed.

```

#### DATASET: WMDP CYBER

```

1 DESCRIPTION
2 The WMDP (Weapons of Mass Destruction Proxy) Cyber Benchmark is designed to evaluate and mitigate
   hazardous cybersecurity knowledge in large language models (LLMs). It consists of 1,987 multiple-
   choice questions covering sensitive cybersecurity topics including reconnaissance, vulnerability
   discovery, exploitation techniques, and cyber tactics. The goal is to unlearn hazardous cybersecurity
   knowledge (forget corpus) while preserving general language capabilities (retain corpus).
3
4 DATASET COMPOSITION
5 The WMDP Cyber dataset contains:
6
7 - Forget corpus: 1,000 text documents containing hazardous cybersecurity information
8 - Retain corpus: 4,473 text documents of general content for maintaining model utility
9 - Format: Pretraining-style text data and multiple-choice evaluation questions
10
11 TASK CONFIGURATION
12 This task uses the following components:
13
14 - cyber-forget-corpus: Text data containing hazardous cybersecurity knowledge to be unlearned
15 - cyber-retain-corpus: General text data to maintain model capabilities
16
17 TASK OBJECTIVE
18 The primary goal is to implement an unlearning algorithm that satisfies:
19
20 1. Hazardous Knowledge Removal: Reduce the model's ability to answer questions about sensitive
   cybersecurity topics
21 2. General Capability Preservation: Maintain strong performance on general language tasks
22
23 EVALUATION METRICS
24 The task uses 2 evaluation metrics:
25
26 1. wmdp_cyber/acc: Accuracy on the 1,987-question WMDP Cyber multiple-choice benchmark (lower is better)

```

- 27 2. mmlu\_stem/acc: Accuracy on the STEM subsection of Massive Multitask Language Understanding (MMLU) benchmark containing 3,153 questions, tested via multiple-choice questions from lm\_eval harness (higher is better)
- 28
- 29 The goal is to optimize for both of them, from which a final score will be computed.

## LOSS PROMPT

- 1 You must modify the `loss.py` file in order to implement the loss logic for your unlearning method. In machine unlearning, the loss balances two objectives: unlearning specific knowledge from the forget set while preserving performance on the retain set. You should not change the name of the class, `CustomUnlearnTrainer`, or the signature of the `compute\_loss` method, but you are allowed to define and use new parameters or methods provided you respect the logic.

## M.8. Off-Policy RL

### DOMAIN DESCRIPTION

- 1 Reinforcement learning is a branch of machine learning focused on training agents to make sequences of decisions in an environment to maximize a notion of cumulative reward. Unlike supervised learning, where models learn from labeled examples, RL agents learn through trial and error, receiving feedback in the form of rewards or penalties based on their actions. The core idea is that the agent explores the environment, evaluates the outcomes of its actions, and gradually improves its decision-making policy to achieve better long-term results.
- 2
- 3 Historically, RL draws inspiration from behavioral psychology, particularly the study of how animals learn from rewards and punishments. Early formalizations in the 1950s and 1960s laid the groundwork for algorithms that could handle Markov decision processes (MDPs), a mathematical framework for modeling decision-making under uncertainty. The agent-environment interaction loop—where the agent observes the state of the environment, takes an action, and receives a reward—remains central to all RL formulations.
- 4
- 5 The objective of reinforcement learning is to find a policy—a mapping from states to actions—that maximizes the expected cumulative reward over time. RL algorithms vary in approach, from value-based methods, which estimate the expected reward of actions, to policy-based methods, which directly optimize the agent's behavior. Success in RL requires balancing exploration (trying new actions to discover rewards) and exploitation (leveraging known actions that yield high rewards).
- 6
- 7 In practice, RL has been applied to robotics, game playing, resource management, and recommendation systems, among other areas, where sequential decision-making is key. Understanding the principles of reward, policy, and environment dynamics is essential before implementing an RL algorithm, as these components shape how the agent learns and adapts.
- 8
- 9 You will be assisting in developing algorithms for Value-based Reinforcement Learning, which focuses on estimating the expected return (value) of states or state-action pairs under a policy. The agent learns a value function either the state-value function  $V(s)$  or the action-value function  $Q(s, a)$ —and derives an optimal policy by selecting actions that maximize these estimates. Existing algorithms iteratively update value estimates using the Bellman equation, enabling the agent to approximate the optimal policy without explicitly modeling the environment's dynamics.
- 10
- 11 Below, we provide a description of the environment which you will be training in. However, be aware that any code you develop may be applied to other RL environments too.

## CONFIG PROMPT

- 1 You should change the config file, which can be found in `config.py`. The goal of config is to provide the hyperparameters for the training loop. You should not change the name of the dict `config`. You are allowed to make whatever changes you like, but you should ensure that your config is compatible

with the rest of the codebase.

#### OPTIM PROMPT

- 1 You should change the optimizer file, which can be found in `optim.py`. In deep learning, optimization is used to descend the gradient of a loss function with respect to the parameters of a neural network. The correct functional form should implement a class, 'scale\_by\_optimizer', with functions 'init\_fn' and 'update\_fn'. Besides this, you are allowed to make whatever changes you like. You will be using a fixed learning rate which is tuned to your algorithm.

#### Q UPDATE PROMPT

- 1 You should change the q update file, which can be found in `q\_update.py`. The goal of q\_loss\_fn is to compute a scalar loss that guides learning of the Q-function—typically by comparing predicted and target action values—but it may represent any differentiable objective that drives the agent toward better value estimation and decision-making. You should not change the name or interface of the function `q\_loss\_fn`.

#### NETWORKS PROMPT

- 1 You should change the network file, which can be found in `networks.py`. In deep learning, the network provides the architecture of the model, and specifies the number of parameters, width of layers, and type of connections that those layers should have. You should not change the name or interface of the function `ActorCritic`. The network takes at input an observation from the RL environment, and must output a policy and a value.

#### POLICY PROMPT

- 1 You should change the policy file, which can be found in `policy.py`. The goal of policy is to select actions based on the current state of the environment and the Q-values. This involves balancing the exploration-exploitation tradeoff: deciding whether to explore new actions to gather more information about their potential rewards, or to exploit known actions that currently yield the highest estimated return. You should not change the name or interface of the function `explore` and `exploit`.

#### RB PROMPT

- 1 You should change the replay buffer file, which can be found in `rb.py`. The goal of the replay buffer is to store and sample experience from the environment. You should not change the name or interface of the function `get\_replay\_buffer`.

#### TRAIN PROMPT

- 1 You should change the train file, which can be found in `train.py`. In modern reinforcement learning, the train loop is responsible for collecting data from the environment, processing it (possibly multiple times), and then using it to update the parameters of the model. You should not change the name or interface of the function `train`. You are allowed to make whatever changes you like, but you should ensure that your training loop is compatible with the rest of the codebase. You should never change the logic for computing the return during or after training. Some useful function calls have been provided, though possibly not in the optimal place.

## M.9. On-Policy RL

### DOMAIN DESCRIPTION

- 1 Reinforcement learning is a branch of machine learning focused on training agents to make sequences of decisions in an environment to maximize a notion of cumulative reward. Unlike supervised learning, where models learn from labeled examples, RL agents learn through trial and error, receiving feedback in the form of rewards or penalties based on their actions. The core idea is that the agent explores the environment, evaluates the outcomes of its actions, and gradually improves its decision-making policy to achieve better long-term results.
- 2
- 3 Historically, RL draws inspiration from behavioral psychology, particularly the study of how animals learn from rewards and punishments. Early formalizations in the 1950s and 1960s laid the groundwork for algorithms that could handle Markov decision processes (MDPs), a mathematical framework for modeling decision-making under uncertainty. The agent-environment interaction loop—where the agent observes the state of the environment, takes an action, and receives a reward—remains central to all RL formulations.
- 4
- 5 The objective of reinforcement learning is to find a policy—a mapping from states to actions—that maximizes the expected cumulative reward over time. RL algorithms vary in approach, from value-based methods, which estimate the expected reward of actions, to policy-based methods, which directly optimize the agent's behavior. Success in RL requires balancing exploration (trying new actions to discover rewards) and exploitation (leveraging known actions that yield high rewards).
- 6
- 7 In practice, RL has been applied to robotics, game playing, resource management, and recommendation systems, among other areas, where sequential decision-making is key. Understanding the principles of reward, policy, and environment dynamics is essential before implementing an RL algorithm, as these components shape how the agent learns and adapts.
- 8
- 9 Below, we provide a description of the environment which you will be training in. However, be aware that any code you develop may be applied to other RL environments too.

### OPTIM PROMPT

- 1 You should change the optimizer file, which can be found in `optim.py`. In deep learning, optimization is used to descend the gradient of a loss function with respect to the parameters of a neural network. The correct functional form should implement a class, 'scale\_by\_optimizer', with functions 'init\_fn' and 'update\_fn'. Besides this, you are allowed to make whatever changes you like. You will be using a fixed learning rate which is tuned to your algorithm.

### LOSS PROMPT

- 1 You should change the loss file, which can be found in `loss.py`. In deep learning, the loss provides an objective to minimize; in reinforcement learning, minimizing this objective corresponds to maximising the return of an agent. You should not change the name of the function, `loss\_actor\_and\_critic`, or its inputs.

### NETWORKS PROMPT

- 1 You should change the network file, which can be found in `networks.py`. In deep learning, the network provides the architecture of the model, and specifies the number of parameters, width of layers, and type of connections that those layers should have. You should not change the name or interface of the function `ActorCritic`. The network takes at input an observation from the RL environment, and must output a policy and a value.

### TRAIN PROMPT

<sup>1</sup> You should change the train file, which can be found in `train.py`. In modern reinforcement learning, the train loop is responsible for collecting data from the environment, processing it (possibly multiple times), and then using it to update the parameters of the model. You should not change the name or interface of the function `train`. You are allowed to make whatever changes you like, but you should ensure that your training loop is compatible with the rest of the codebase. You should never change the logic for computing the return during or after training. Some useful function calls have been provided, though possibly not in the optimal place.

## M.10. Unsupervised Environment Design

### DOMAIN DESCRIPTION

<sup>1</sup> Reinforcement learning is a branch of machine learning focused on training agents to make sequences of decisions in an environment to maximize a notion of cumulative reward. Unlike supervised learning, where models learn from labeled examples, RL agents learn through trial and error, receiving feedback in the form of rewards or penalties based on their actions. The core idea is that the agent explores the environment, evaluates the outcomes of its actions, and gradually improves its decision-making policy to achieve better long-term results.

<sup>2</sup>

<sup>3</sup> Historically, RL draws inspiration from behavioral psychology, particularly the study of how animals learn from rewards and punishments. The agent-environment interaction loop—where the agent observes the state of the environment, takes an action, and receives a reward—remains central to all RL formulations. In this loop, the environment defines the challenges the agent must overcome.

<sup>4</sup>

<sup>5</sup> While traditional RL focuses on mastering a fixed environment, a key challenge is ensuring agents can generalize to new, unseen situations. This has led to the field of Unsupervised Environment Design (UED), where the goal is to automatically generate a curriculum of training environments that progressively challenge an agent. Instead of a static world, a UED algorithm acts as a "teacher," creating a series of tasks that are neither too easy nor too hard, pushing the "student" agent to develop robust and widely applicable skills.

<sup>6</sup>

<sup>7</sup> Your task is to design a novel Unsupervised Environment Design algorithm. This involves creating a system that adaptively generates environment parameters to effectively train a separate RL agent. The objective is no longer just to find a policy that maximizes reward, but to design an environment generation process that produces a final agent with broad, generalizable capabilities. This requires balancing the generation of novel, challenging environments with those that remain solvable for the agent at its current skill level.

<sup>8</sup>

<sup>9</sup> Below, we provide a description of a base environment whose parameters your UED algorithm will control. Your algorithm will be responsible for generating distributions over these parameters to create a curriculum. Be aware that the UED methodology you develop should be general enough to be applied to other families of parametric RL environments.

### DATASET: MINIGRID

#### <sup>1</sup> DESCRIPTION

<sup>2</sup> JAXUED Minigrid is a JAX-accelerated grid-world maze navigation environment designed for Unsupervised Environment Design research. An agent must navigate through procedurally generated mazes to reach a goal location. The environment provides a partially observable view around the agent, showing only a local window of the maze rather than the full layout. The implementation is fully jittable and runs on GPU or TPU, enabling training on thousands of parallel environments simultaneously. Maze layouts are procedurally generated to create diverse navigation challenges, making the environment ideal for studying curriculum learning, exploration, and generalization in reinforcement learning.

#### <sup>3</sup>

#### <sup>4</sup> OBSERVATION SPACE

<sup>5</sup> The observation is a 4-dimensional ndarray with shape (view\_size, view\_size, 3), where view\_size is typically 5 or 7. This represents the agent's partial field of view centered on its current position. Each cell in the view contains encoded information about the grid tile:

#### <sup>6</sup>

<sup>7</sup> Channel 0: Tile type ID (wall, empty floor, goal, lava, door, key, etc.)

```

8 Channel 1: Color ID for the tile
9 Channel 2: State information (for example, whether a door is open or closed)
10
11 The observations are not image pixels but rather integer encodings of tile properties. The agent can only
    observe a local window around its current position and orientation, which means it must explore the
    environment and potentially use memory to navigate successfully through larger mazes.
12
13 ACTION SPACE
14 The action space consists of 6 discrete actions:
15
16 0: move_forward - Move one cell in the direction the agent is facing
17 1: turn_left - Rotate the agent's orientation 90 degrees counterclockwise
18 2: turn_right - Rotate the agent's orientation 90 degrees clockwise
19 3: pick_up - Pick up an object located in front of the agent
20 4: put_down - Drop the currently held object
21 5: toggle - Activate a switch or open/close a door
22
23 In pure maze navigation tasks, typically only actions 0, 1, and 2 are relevant, as the task involves
    moving to the goal without interacting with objects.
24
25 TRANSITION DYNAMICS
26 The agent operates on a discrete 2D grid, commonly of size 1313 or 1515 cells. The agent maintains a
    directional state, facing one of four cardinal directions: north, south, east, or west. When the move
    _forward action is taken, the agent attempts to advance one cell in its facing direction, but this
    movement is blocked if a wall occupies that cell. The turn_left and turn_right actions change the
    agent's orientation by 90 degrees without changing its position. The goal is reached by moving the
    agent onto the goal tile. All transitions are fully deterministic given the current state and chosen
    action. Unlike the Kinetix environments, there is no physics simulation; the environment uses
    discrete state transitions on the grid.
27
28 REWARD
29 The reward structure depends on the configuration chosen for the environment. In the default sparse
    reward setting, the agent receives a reward of +1 for reaching the goal tile and 0 at all other
    timesteps. In the optional dense reward setting, the agent receives a small negative reward per step
    (for example, -0.01) to encourage efficient solutions, plus +1 for reaching the goal. Some variants
    use distance-based reward shaping to provide guidance. The sparse reward structure combined with
    procedural maze generation creates significant exploration challenges, as the agent must discover the
    goal through exploration without intermediate feedback.
30
31 STARTING STATE
32 The agent is spawned at a designated start position, often near a corner or edge of the maze. The agent's
    initial orientation is randomly selected from the four cardinal directions. The maze layout is
    generated procedurally using various algorithms including random obstacle placement, recursive
    division, depth-first search maze generation, and other methods that can be controlled by
    Unsupervised Environment Design algorithms. The goal position is placed in the maze, typically
    positioned far from the start location to create a meaningful challenge. The maze generation process
    ensures that a valid path exists from the start to the goal.
33
34 EPISODE END
35 The episode ends if either of the following happens:
36
37 - Termination: The agent reaches the goal tile
38 - Truncation: The length of the episode reaches the maximum number of timesteps, which is typically 250
    to 500 depending on the maze size and difficulty

```

## SAMPLE LEVELS PROMPT

- 1 You should change the `sample\_and\_score\_fn` function, which is defined inside the `make\_sample\_and\_score\_fn` factory in the file `sample\_levels.py`. In Unsupervised Environment Design (UED), the goal is to automatically generate a curriculum of environments to train a robust agent. This function is a core component of that process. It is responsible for sampling a batch of potential training levels, evaluating the current agent's performance on them, and then assigning a "score" to each level.

Levels with higher scores are more likely to be selected for the agent's subsequent training. Your task is to \*\*replace the placeholder scoring logic\*\*. Currently, it uses a naive implementation that scores all levels equally: `level\_scores = jnp.ones\_like(returns)` . You must implement a more meaningful scoring function that identifies levels that are most beneficial for training the agent. For example, you might want to score levels based on the agent's uncertainty, the variance of returns , or whether the agent is making progress (but not finding it too easy or too hard). You should not change the name or interface of the `sample\_and\_score\_fn` function. You must ensure that it returns the same data structure: a tuple containing the `train\_state` and another tuple of `(env\_instances, level\_scores, extra\_dict)` . Note that you have access to the full trajectory data (`traj\_batch`) from the agent's rollout, which includes rewards, values, dones, and other information that can be used to compute your scores.