Procedural Environment Generation for Tool-Use Agents

Michael Sullivan **Mareike Hartmann** **Alexander Koller**
Department of Language Science and Technology
Saarland Informatics Campus
Saarland University, Saarbrücken, Germany
{msullivan, mareikeh, koller}@coli.uni-saarland.de

Abstract

Although the power of LLM tool-use agents has ignited a flurry of recent research in this area, the curation of tool-use training data remains an open problem—especially for online RL training. Existing approaches to synthetic tool-use data generation tend to be non-interactive, and/or non-compositional. We introduce RandomWorld, a pipeline for the procedural generation of interactive tools and compositional tool-use data. We show that models tuned via SFT and RL on synthetic RandomWorld data improve on a range of tool-use benchmarks, and set the new SoTA for two metrics on the NESTFUL dataset. Further experiments show that downstream performance scales with the amount of RandomWorld-generated training data, opening up the possibility of further improvement through the use of entirely synthetic data.

1 Introduction

A substantial amount of current research has focused on equipping LLMs with the means to employ tools—external functions (e.g. APIs) that provide the LLM with enhanced knowledge and/or capabilities—with the goal of achieving AI assistants capable of executing complex sequences of tool calls to accomplish tasks such as information retrieval (e.g. Zheng et al., 2024; Li et al., 2025; Zheng et al., 2025), making online purchases (e.g. Yao et al., 2022; Cai et al., 2025), editing a user's local files (e.g. Trivedi et al., 2024), etc. In line with broader findings that LLMs post-trained via reinforcement learining (RL) exhibit superior generalization capabilities to supervised-fine-tuned (SFT) agents (Chu et al., 2025), recent work demonstrates that LLMs fine-tuned for tool use through online RL can better adapt to tools and tasks not seen during training (Qian et al., 2025; Feng et al., 2025).

To enable the agent to effectively employ tools across a wide range of tasks and domains, LLM

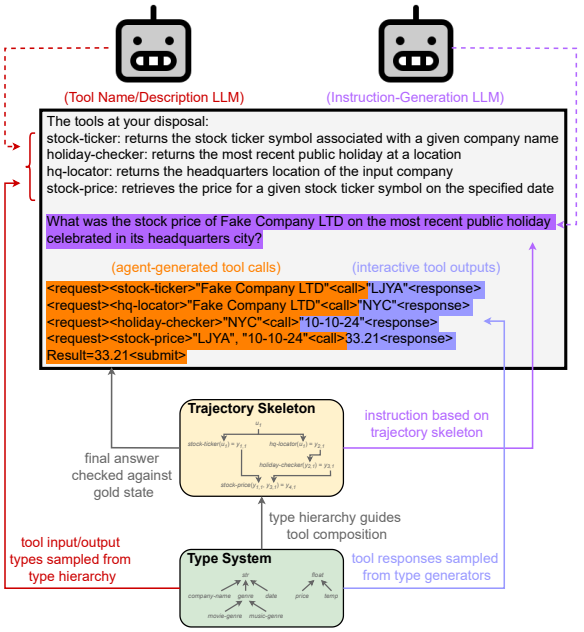

Figure 1: Example of an agent executing a (simple) non-linear, compositional task with interactive tools in a RandomWorld environment, with an illustration of the pipeline components that go into the generation of the task and environment.

tool-use training necessitates a sufficiently rich environment. However, dataset curation for online RL tool-use training remains a major challenge: it is not feasible to directly train tool use agents in the real-world (due to the obvious dangers of such an approach), and hand-crafting simulated APIs—and tasks using these APIs—is a time-consuming process that is not feasible at the scale required for (SFT or RL) LLM fine-tuning. As a result, many existing datasets are comprised of either small or non-interactive (i.e. not callable) tool inventories, and/or non-compositional tasks (see Section 2). While training on such datasets may be sufficient to conquer simpler benchmarks (e.g. Yan et al., 2024; Patil et al., 2024, etc.; see Section 2), they fail to satisfactorily improve model performance on benchmarks with more complex tasks: namely,

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 18544–18562
November 4-9, 2025

tasks necessitating the non-linear chaining of tool calls in an interactive setting (e.g. Zhuang et al., 2023; Basu et al., 2024; Trivedi et al., 2024, etc.; see Sections 2 and 4).

On the other hand, those datasets that do contain large, interactive tool inventories with compositional tasks are carefully hand-crafted (e.g. Trivedi et al., 2024), and so contain a very limited number of tasks: while valuable as benchmarks, these datasets are not suitable for large-scale RL training.

In this paper, we introduce *RandomWorld* (Section 3), a pipeline for the procedural generation of tools and tasks, that is capable of generating a virtually unlimited amount of training data for online RL (or SFT) training of tool-use agents. As illustrated in Figure 1, RandomWorld employs a fine-grained type hierarchy to generate non-linear DAGs (trajectory skeletons) of tool calls from the signatures of fully-interactive (i.e. callable) synthetic tools (see Section 3). These trajectory skeletons parameterize the construction of environments: data structures consisting of tools, a goal state (value to return), and an instruction. This procedural generation pipeline results in environments that have:

i. *Depth* (of the tool inventory): a large toolset across a diverse assortment of domains, to facilitate the agent's ability to generalize to unseen tools.

ii. (Non-linear) *Compositionality*: chainable tools—and objectives necessitating non-linear tool-chaining—to emulate complex, real-world tasks (see e.g. Figure 1).

iii. *Interactivity*: intermediate tool outputs that are visible to the agent, allowing the inspection of outputs and (if necessary) correction of the tool-call sequence—as is possible in many real-world settings.

Models fine-tuned with RandomWorld—through either RL or SFT—exhibit increased performance on various tool-use benchmarks (Section 4), and set the new SoTA on two NESTFUL (Basu et al., 2024) metrics. Further experiments show that downstream performance scales with the size of the tool inventory and number of tasks in the training set (Section 5), indicating that further training with RandomWorld can further improve performance, without the need for costly human annotation.

We release all code for the RandomWorld pipeline and the experiments conducted in this paper on GitHub[1].

2 Related Work

Non-Interactive Tool-Use Datasets. Although many existing tool use datasets have large API inventories and complex tasks that require non-linear tool chaining, they do not include interactive tools (see Table 1). For example, while APIGen (Liu et al., 2024b) employs real-world, callable APIs during dataset generation, this pipeline only creates non-interactive SFT (and similar, e.g. DPO; Rafailov et al., 2023) training data: namely, queries/instructions annotated with solution paths. Similarly, the ToolBench (Qin et al., 2024) dataset contains an interactive test set, but a non-interactive train set—while it may be theoretically possible to adapt ToolBench to online RL training, the latency accompanying this dataset's use of real-world APIs would impede such an approach.

In the ToolACE (Liu et al., 2024a) and APIBank (Li et al., 2023) training sets, both the tasks/solution paths *and* tools are synthesized, as in RandomWorld. Unlike RandomWorld, the tools in ToolACE and APIBank are not callable: an LLM simulates outputs during dataset creation. This again results in SFT-only training sets.

The UltraTool dataset (Huang et al., 2024) contains LLM-synthesized tasks and solution paths that require the complex, non-linear compositional use of over 2,000 tools. However, these tools are not functional, and so are not interactive.

While APIBench (Patil et al., 2024) has a deep tool inventory—over 700 real-world APIs—it is not interactive or compositional: the LLM-synthesized tasks in this dataset require only one API call.

The non-interactivity of the tools in these datasets impairs evaluation, which further hinders their employment for online RL training, as evaluation and reward are inextricably linked. When tools are not interactive, scores must be assigned through either exact match of the tool call sequence—which ignores the possibility of a task having more than one solution path—or LLM-as-a-judge evaluation.

Interactive Tool-Use Datasets and Benchmarks. On the other hand, the WebShop (Yao et al., 2022) dataset is compositional and fully interactive. However, the WebShop tool inventory is shallow (eight actions/tools), which limits the knowledge obtained from training on the dataset to this narrow domain.

[1] https://github.com/coli-saar/randomworld

Dataset	Tool Inventory			Tasks	
	Interactive	Deep	Synthetic	Compositional	Synthetic
APIGen	X	✓	X	✓	✓
ToolBench	X	✓	X	✓	✓
UltraTool	X	✓	X	✓	✓
ToolAce	X	✓	✓	✓	✓
APIBank	X	✓	✓	✓	✓
APIBench	X	X	X	X	✓
WebShop	✓	X	X	✓	X
BFCL V3	✓	X	X	✓*	✓+
AppWorld	✓	✓	X	✓	X
RandomWorld	✓	✓	✓	✓	✓

Table 1: Comparison between RandomWorld and existing tool-use datasets and benchmarks. (*compositional but linear-only; +checked by human annotators)

Similarly, the Berkeley Function-Calling Leaderboard (BFCL V3; Yan et al., 2024) contains fully interactive APIs. As in RandomWorld, BFCL V3 task synthesis begins with the tool call sequence, rather than instruction. Although the BFCL V3 generation procedure results in compositional tasks, it does not result in *non-linear* tasks.

AppWorld (Trivedi et al., 2024) is a simulated world of realistic, carefully hand-crafted users, apps, and APIs. This benchmark contains a deep tool inventory of over 450 fully interactive APIs, which are employed in highly complex and non-linearly compositional tasks. But, as they are hand-crafted, the number of tasks in AppWorld is severely limited (750): this benchmark is therefore not suitable for online RL (or SFT) training.

Other Synthetic Training Data Pipelines. However, synthetic data-generation pipelines show promise in other areas. For example, the AGENT-GEN pipeline Hu et al. (2024) demonstrates that LLMs trained on synthetic data can markedly improve on PDDL (McDermott et al., 1998) planning tasks—and even achieve SoTA results on multiple benchmarks. Similarly, Davidson et al. (2025) find that LLMs trained on synthetic data generated from the SuperGLUE dataset (Wang et al., 2019) improve in performance over baseline models. These authors additionally show that a sufficiently advanced synthetic-data pipeline is capable of constructing examples that are more complex than their real-world counterparts.

Matthews et al. (2024) show that the advantages of synthetic data are not limited to SFT: pretraining an RL agent in randomly-generated physics environments can improve performance on actual downstream tasks. In some cases, this pretrained agent can outperform task-specific models.

3 RandomWorld

As discussed in Section 1, RandomWorld leverages a fine-grained type system (see Section 3.1) to achieve the procedural generation of interactive tools, environments, and non-linear, compositional tasks: instruction/goal state pairs that necessitate non-linear sequences of API calls. Potential input/output types are sampled and passed to an LLM to create tool names and descriptions (see Section 3.2). The signatures of these generated tools (and, optionally, hand-crafted tools) are used to generate trajectory skeletons, which parameterize environment and instruction generation (see Section 3.3).

These generated tools and environments are then easily interfaced with existing RL and SFT pipelines (see Section 3.4).

3.1 Type System

To create the type hierarchy T, we constructed 73 base types: fine-grained subtypes of strings (e.g. *month-name*, *movie-title*, *address*), integers (e.g. *age*, *year*, *spotify-id*), and floats (e.g. *hotel-rating*, *temperature*, *price*). To faciltate the task generation procedure (see Section 3.3), we impose further subtype constraints within this set of custom types: for example, *actor-name* is a subtype of *person-name* (which in turn is a subtype of *string*). Our full type hierarchy is given in Figure 3 in the Appendix.

For each of these base types, we craft a description (used for automated tool generation in Section 3.2), a *generator*, and a *recognizer*. Type generators create new instances of that type, and are

used to produce automatically-generated tool outputs (see Section 3.2): for example, the generator for *month-name* simply samples one of the twelve month names, while the generator for *price* samples a float between 1 and 5000, rounded to two decimal points. Type recognizers are boolean-valued functions that check if an object belongs to the type in question, and are used for type-checking agent inputs when it is interacting with the environment (i.e. using the tools). Generators and recognizers for super-types are inherited from their subtypes.

We implemented three type constructors, which allow for the generation of a theoretically unlimited number of types: *list*: $T \to T$, which takes a type t and returns the type *list*(t) of lists of objects of type t; *dict*: $T \times T \to T$, which takes types t, u and returns the type *dict*(t, u) of dictionaries mapping objects of type t to objects of type u; and *union*: $T \times T \to T$, which takes types t, u and returns the type *union*(t, u) of objects of type t or u. Subtype relations between—and type recognizers/generators for—constructed types are automatically inferred from their constituent type(s).

A detailed description of the RandomWorld type system is located in Appendix A.

3.2 Tool Creation

To automatically generate a tool—i.e. an LLM-callable function—in RandomWorld, we sample input types $X_1, \ldots, X_n$ and output types $Y_1, \ldots, Y_m$ (see Section 3.1). We then use an LLM to generate a possible name and description for a tool $f\colon X_1 \times \cdots \times X_n \to Y_1 \times \cdots \times Y_m$, and prompt the LLM to score the plausibility and realism of the generated description on a scale from 1-5: tools with a score less than 4 are discarded.

When passed inputs $x_1, \ldots, x_n$, the tool returns $y_1, \ldots, y_m$, where each y_i is sampled from the type generator for Y_i. While the agent is interacting with the environment, input/output pairs $((x_1, \ldots, x_n), (y_1, \ldots, y_m))$ are temporarily stored, ensuring that—from the perspective of the agent—the tool always returns the same output for a given input (see Section 3.3.2). If the tool is passed an input of an invalid type—as determined by tool input type's recognizer (see Section 3.1)—the tool returns an error message.

In comparison to a tool generation procedure in which an LLM directly codes the tools, our type-guided procedure guarantees the behavior of the tools: each tool always returns values of its annotated type. This consistency facilitates the environment and task generation process of Section 3.3. Examples of tools generated by RandomWorld are given in Table 6 in the Appendix.

Note that a tool in RandomWorld is simply a function annotated with input/output types (see Section 3.1). This permits the use of hand-crafted tools in place of—or alongside—synthetic tools. For example, our experiments (Section 4) included six hand-crafted calculator tools: *add*, *subtract*, *multiply*, *divide*, *max*, and *min*.

RandomWorld additionally implements dependently-typed tools: tools whose output type is a function of their input. For example, *add*: $(t \leq \textit{float}) \times (t \leq \textit{float}) \to t$ will (for example) take the prices of two items and return their total price (see Figure 2). This permits richer and more controlled environment generation (see Section 3.3). However, due to the added complexity that they introduce, we only experiment with hand-crafted dependently-typed tools.

3.3 Environment and Instruction Generation

In comparison to most synthetic task generation pipelines, in which an LLM first synthesizes a query that is then used to generate a sequence of API calls (e.g. Liu et al., 2024b; Qin et al., 2024; Liu et al., 2024a), RandomWorld tasks are synthesized by first generating a sequence of API calls through a type-guided sampling procedure (see Section 3.3.1). These API calls are then used to populate the environment (i.e. generate tool input/output values; see Section 3.3.2) and generate a corresponding instruction via LLM (see Section 3.3.3).

See Appendix B for examples of RandomWorld-generated instructions and tool call sequences.

3.3.1 Trajectory Skeletons

To automatically construct a task—i.e. an instruction/goal state pair—in RandomWorld, we first generate a *trajectory skeleton*: a sequence of tool calls $f_1, \ldots, f_n$, along with annotations indicating the output(s) of the tool(s) $f_j, \ldots, f_k$ that f_i $(i > j, k)$ takes as input (see Figure 2). The fine-grained type system defined in Section 3.1 permits lazy evaluation: we do not compute tool outputs until the trajectory skeleton is constructed. This is essential to RandomWorld's type-checked function sampling procedure, which necessitates back-and-forth construction and trimming of trajectory skeletons until convergence.

A trajectory skeleton is constructed by first sampling *user input* type(s) $Y_{0,1}, \ldots, Y_{0,m}$, correspond-

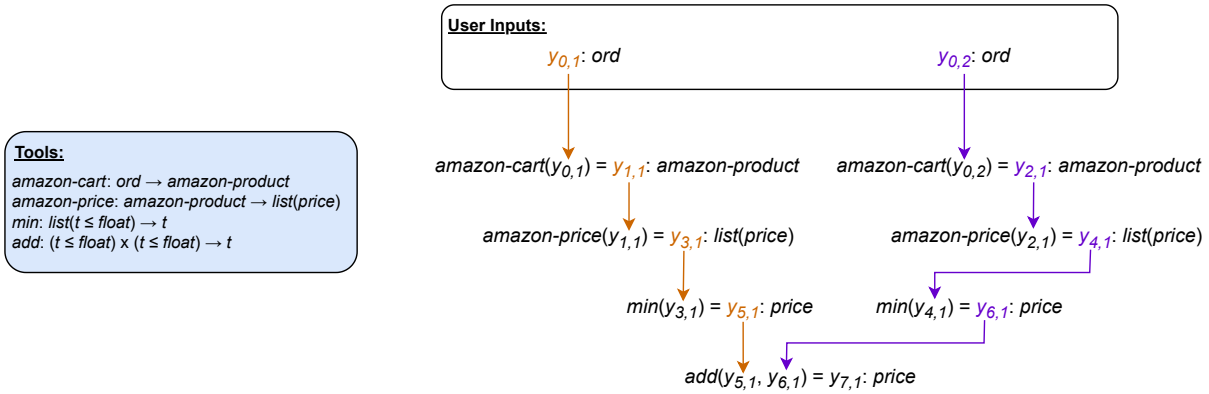

Figure 2: Example of a non-linear trajectory skeleton that corresponds to the instruction *"how much will the $y_{0,1}$-th and $y_{0,2}$-th most recently-added items in my Amazon cart cost together, if purchased at the lowest-available price?"*.

ing to the value(s) that will be fed to the agent in the instruction, rather than as a tool output: e.g. *"find comedy movies on Netflix that last less than two hours"*. We then sample a trajectory length ℓ such that $\ell_{min} \leq \ell \leq \ell_{max}$, where ℓ_{min} and ℓ_{max} are pre-defined minimum and maximum trajectory lengths, to (for example) control the complexity of training examples for curriculum learning.

Next, we sample a tool f_1 that is compatible with the variable set V (the user-input types $Y_{0,1}, \ldots, Y_{0,m_0}$): i.e. such that for each input type $X_{1,i}$ of f_1, there is a type $Y \in V$ with $Y \leq X_{1,i}$. For example, in Figure 2, *amazon-cart*: *ord* $\rightarrow$ *amazon-product* is compatible with the variable set $V = \{y_{0,1}\text{: } ord, y_{0,2}\text{: } ord\}$ (as $ord \leq ord$).

We then record the types in V that match each argument position of f_1, and add the output types $Y_{1,1}, \ldots, Y_{1,m_1}$ of f_1 to V. In the example in Figure 2, we record that the argument of *amazon-cart* is $y_{0,1}$, and add its output type to the variable set: $V \leftarrow V \cup \{y_{1,1}\text{: } amazon\text{-}product\}$.

This procedure continues until the trajectory skeleton reaches ℓ tool calls; the output(s) of the last tool in the sequence are taken as the goal state.

Let $U(f_\ell) = 1$, and for all $i < \ell$, let $U(f_i) = 1$ if at least one output of f_i is taken as input by some f_k such that $U(f_k) = 1$, and $U(f_i) = 0$ otherwise. In words: $U(f_i) = 1$ if at least one output of f_i is used (directly or indirectly) to compute the goal state. We remove from the trajectory skeleton all tool calls f_i such that $U(f_i) = 0$, and sample new tool calls to add to this trimmed trajectory skeleton until it again reaches length ℓ. This procedure repeats until the trajectory skeleton contains ℓ tool calls, and $U(f_i) = 1$ for all $1 \leq i \leq \ell$.

When generating a set of trajectory skeletons—e.g. to construct a training or evaluation set—we filter out all duplicate trajectory skeletons before environment generation (Section 3.3.2), in order to ensure diversity.

3.3.2 Environments

Once the trajectory skeleton has been generated, we sample user input values and compute output values for each tool call in the sequence, thereby populating an *environment*: a data structure that contains tool input/output values, user information (see below), a goal state, and an instruction (see Section 3.3.3). The storage of each tool's input/output values as they are computed ensures deterministic outputs from the agent's perspective.

We optionally allow tools to be (manually) assigned to an app: when a tool is assigned to an app, the agent must login to that app before using the tool. A random username and password is generated for each unique app associated with a trajectory skeleton.

With probability p_g (a tunable parameter), the instruction (see Section 3.3.3) contains the username and password to an app used in the trajectory skeleton. Otherwise, only the username is provided, and the agent must use the *password-manager* tool to retrieve the user's password for that app.

3.3.3 Instructions

After the environment has been populated from a trajectory skeleton, we employ an LLM to generate an instruction for that environment. We prompt the LLM with descriptions of each tool in the trajectory, the value(s) of the user input(s), and the trajectory skeleton—to ensure that the LLM does not leak information about any of the tool outputs into the instruction, we replace all non-user-input values with variables of form "x_i,j".

As the instruction-generation LLM can create instructions that do not provide sufficient information to reach the goal state[2], we verify each generated instruction by checking whether an LLM can reach the goal state using the instruction. As we are merely verifying the informativity of the instructions, we provide the instruction-verification LLM with several advantages not afforded to the agents trained in Section 4: we provide the type signatures of the tools, including descriptions and examples of each input/output type; we provide only the tools required to reach the goal (i.e. we do not include any distractor tools; see Section 3.4); we provide tool descriptions in the order that the tool calls are made in the gold trajectory; and we disable the app login mechanism. We discard all environments in which the instruction-verification LLM was unable to reach the goal state given the instruction.

Although we take care to prevent information leakage with respect to tool *outputs*, the instruction generation process outlined here does carry a non-trivial risk of information leakage regarding tool names and/or descriptions, which may inadvertently simplify the tasks (see Appendix C for a detailed analysis). However, the results of our RandomWorld-trained models in Section 4 indicate that any information leakage that may have occurred did not substantially detract from the models' performance.

3.4 Agent Interface

RandomWorld is fully compatible with TRL (von Werra et al., 2020) text environments: for evaluation and RL training, we simply create a text environment for each RandomWorld environment, and pass the agent and the RandomWorld environment's tools to the text environment.

For SFT training, we use the trajectory skeleton and stored tool input/output values to construct a training instance that reaches the goal state.

Prompts are constructed by prepending a tool inventory to the instruction. The tool inventory presented to the agent consists of all tools used in the corresponding trajectory skeleton, along with a number of randomly-selected distractor tools, as specified by the pre-defined ratio r_{dist}.

[2]e.g. *"perform a series of arithmetic operations"* when there are many calculator tool calls in the trajectory.

4 Experiments

To assess the efficacy of RandomWorld-generated data for SFT and RL, we fine-tuned Llama-3.1-8B-Instruct[3] and Qwen2.5-7B-Instruct (Yang et al., 2024) on 12,000 environments generated from a set of six hand-crafted and 550 procedurally-generated tools, with $\ell_{min} = 2$, $\ell_{max} = 8$, and $r_{dist} = 1.0$.

We employed GPT-4o (Hurst et al., 2024) as the tool-creation (see Section 3.2) and instruction-generation/verification LLM (see Section 3.3.3). We achieved a pass rate rate of $\sim$11% for tool generation (i.e. $\sim$89% of the candidate tools were discarded; see Section 3.2) and $\sim$60% for environment generation (including de-duplicated trajectory skeletons; see Section 3.3). The total OpenAI API cost was roughly $150 USD.

We then evaluated these RandomWorld-trained models against baseline tool-use models (see Section 4.2) on a series of downstream tool-use benchmarks (Section 4.3).

4.1 Training

For each model, we trained one variant using Group Relative Policy Optimization (GRPO; Shao et al., 2024) from scratch (i.e. "zero RL"), and one with standard SFT. The GRPO variants were trained for one epoch with a group size of 8, $\beta = 0.04$, a batch size of 32, a learn rate of 10^{-6}, and a temperature of 0.75. We employed an exact match reward function with respect to the goal state for GRPO training, as in Guo et al. (2025). A full description of our online RL training regimen is located in Appendix D.

The SFT variants were trained with a batch size of 32, a learn rate of 10^{-5}, and weight decay of 10^{-3}, using early stopping when performance failed to improve on a withheld validation set (four epochs for Llama, six for Qwen). For all models, we employed LoRA (Hu et al., 2022) adapters with $r = 64$, $\alpha = 128$, and 0.05 dropout on all Q, K, V, and O attention projection matrices.

4.2 Baseline Models

We compared our fine-tuned Qwen models to Hammer2.0-7B (Lin et al., 2025), a Qwen2.5-7B-Instruct model fine-tuned via SFT on an augmented version of the xlam-function-calling-60k tool-use dataset (Zhang et al., 2024), which was generated

[3]https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct

Model	RandomWorld Test	ToolQA Easy	ToolQA Hard	NESTFUL F1 Func.	NESTFUL F1 Param.	NESTFUL Part. Acc.	NESTFUL Full. Acc.	NESTFUL Win Rate
Llama*	0.483	0.236	0.015	0.53	0.38	0.24	0.19	0.16
ToolACE-8B*	0.170	0.206	0.046	0.51	0.27	0.14	0.0	0.0
Llama-RW-GRPO (Ours)	0.721	0.251	0.023	0.89	0.62	0.22	0.17	0.10
Llama-RW-SFT (Ours)	**0.855**	0.086	0.019	0.61	0.36	0.11	0.07	0.04
Qwen*	0.109	0.258	0.011	0.90	0.66	0.27	0.21	0.28
Hammer2.0-7B*	0.011	0.005	0.001	0.61	0.46	**0.31**	**0.25**	**0.34**
Qwen-RW-GRPO (Ours)	0.536	0.243	0.063	0.92	0.69	0.28	0.22	0.30
Qwen-RW-SFT (Ours)	0.714	0.307	0.019	**0.96**	**0.71**	0.28	0.22	0.32
Mixtral-8x22B*	—	—	—	0.64	0.48	0.29	0.21	0.29
GPT-3.5+	0.348	**0.368**	**0.082**	—	—	—	—	—

Table 2: Results of our models, ToolACE-8B, Hammer2.0-7B, and the base Llama-3.1-8B-Instruct/Qwen2.5-7B-Instruct on the RandomWorld, ToolQA, and NESTFUL benchmarks. The best results in each column are indicated in **bold**, and the best results within each model type (i.e. Llama or Qwen) are underlined. (*NESTFUL results reported in Basu et al., 2024; +ToolQA results reported in Zhuang et al., 2023)

using the APIGen pipeline (Liu et al., 2024b) discussed in Section 2. We compared our fine-tuned Llama models to ToolACE-8B (Liu et al., 2024a), a Llama-3.1-8B-Instruct model fine-tuned (SFT) on the ToolACE dataset (see Section 2).

We additionally compared our models to the base Llama-3.1-8B-Instruct and Qwen2.5-7B-Instruct.

4.3 Evaluation

We evaluated the models on a RandomWorld test set: 276 environments ($\ell_{min} = 2$, $\ell_{max} = 8$) generated from 75 tools not used in training. All models were evaluated with a maximum of 15 turns (tool calls or attempted tool calls), and a maximum of 128 new tokens per turn. As our evaluation metric, we report exact match accuracy with the goal state.

We additionally evaluated the models on the ToolQA (Zhuang et al., 2023) and NESTFUL (Basu et al., 2024) benchmarks.

ToolQA: a question-answering dataset spanning eight domains and requiring the use of twelve fully-interactive tools, ranging from database-searching tools to a Python interpreter. Models are evaluated on exact match between the gold solution and their returned answer. Questions in ToolQA are sorted into Easy (800) and Hard (730) subsets, based on the complexity of the required tool calls and reasoning process. We evaluated the models using the ReAct framework (Yao et al., 2023), the best-performing approach in Zhuang et al. (2023).

NESTFUL: a dataset specifically designed to evaluate LLMs on nested (i.e. chained) sequences of API calls. NESTFUL consists of 1,861 tasks in the domains of mathematical reasoning and generic coding, requiring the use of over 4,000 callable APIs. However, unlike ToolQA and the RandomWorld test set, NESTFUL is not fully interactive: the agent returns only a sequence of API calls, from which the final answer is then computed.

NESTFUL evaluates models across five dimensions: *F1-Function* (F1F), the F1 score between the predicted and gold API sequences; *F1-Parameter* (F1P), between the predicted and gold parameter names; *Partial Sequence Accuracy* (PSA), the percentage of correct API calls (API and parameter names) with respect to the gold sequence; *Full Sequence Accuracy* (FSA), the percentage of tasks in which the model predicted the entire gold sequence of API calls; and *Win Rate* (WR), the percentage of tasks in which the value returned by the predicted sequence of API calls matches the gold answer. Following Basu et al. (2024), we evaluated our models in a 3-shot in-context learning (ICL) setting.

4.4 Results and Discussion

The results of our experiments are given in Table 2.

Qwen-RW-SFT Achieves NESTFUL F1 SoTA. On the NESTFUL benchmark, our Qwen-RW-SFT model sets the SoTA for the F1F (0.96) and F1P (0.71) scores, both of which reflect the correctness of the predicted API calls. In fact, aside from Llama-RW-SFT, all of our models outperform the previous SoTA (Mixtral-8x22B-Instruct-v0.1[4]).

Synthetic Data Improves Model Performance. Our results show that training tool-use agents purely on procedurally-generated synthetic data improves model performance: Qwen-RW-SFT improves over the base Qwen model on all eight

[4]https://huggingface.co/mistralai/Mixtral-8x22B-Instruct-v0.1

benchmarks/metrics, and Qwen-RW-GRPO on all but one (ToolQA-Easy). Qwen-RW-GRPO also narrows the gap by more than 50% on ToolQA-Hard between open-source models and the closed-source GPT-3.5 (the current SoTA, as reported in Zhuang et al., 2023), a model that is 25 times larger.

Although Llama-SFT-GRPO only improves over the base Llama model on three benchmarks, Llama-RW-GRPO improves on all benchmarks aside from the NESTFUL PSA, FSA, and WR metrics. In particular, Llama-RW-GRPO nearly doubles the performance of the base Llama model on the RandomWorld test set and NESTFUL F1F/F1P metrics, and performance relative to the base model only degrades slightly for PSA, FSA, and WR.

Overall, the superior results of Llama-RW-GRPO over Llama-RW-SFT (and ToolACE-8B) demonstrate the utility of data-creation pipeline for tool-use that is compatible with online RL tuning, while the SoTA and near-SoTA results of Qwen-RW-SFT show the importance of a pipeline compatible with both kinds of fine-tuning.

Data Quality Matters. Both Qwen-RW-GRPO and Qwen-RW-SFT outperform Hammer2.0-7B—another Qwen2.5-7B-Instruct model trained on much more data (see Section 4.2)—on all benchmarks aside from the NESTFUL PSA, FSA, and WR metrics. On those metrics, the performance of Qwen-RW-SFT is on par with that of Hammer2.0-7B—the current 3-shot ICL SoTA on all three metrics. Our Qwen models achieve the second- and third-best NESTFUL 3-shot ICL WR: the previous second-best (Mixtral-8x22B-Instruct-v0.1) has a score of 0.29.

The fact that both Qwen-RW-SFT and Hammer2.0-7B are trained via SFT on synthetic tool-use data raises the question as to why our model outperforms Hammer2.0-7B, especially considering that Hammer2.0-7B is trained on significantly more data (67,500 examples).

Note that both Hammer2.0-7B and ToolACE-8B are trained on datasets in which an LLM generates queries/instructions from a set of tools, from which the sequence of API calls is then derived (see Sections 2 and 4.2). On the other hand, in RandomWorld, the sequence of tool calls is already determined, and is merely described by the LLM (see Section 3.3). This procedure likely generates more complex tasks than an LLM alone is capable of envisioning (c.f. the findings of Davidson et al., 2025 with respect to synthetic reasoning data; see Section 2), thereby providing our models with a richer training set than those of the baselines.

In addition, Llama-RW-GRPO outperforms ToolACE-8B—another Llama-3.1-8B-Instruct model trained via SFT on a similar amount of data (see Section 4.2)—on all benchmarks/metrics but ToolQA-Hard: this result in particular further demonstrates the importance of an RL-capable method such as RandomWorld.

4.5 Analysis: ToolQA-Easy

On the ToolQA-Easy dataset, we observe a puzzling phenomenon: GRPO training on RandomWorld improves the performance Llama-3.1-8B-Instruct, but decreases that of Qwen2.5-7B-Instruct, while Qwen2.5-7B-Instruct benefits from SFT training and Llama-3.1-8B-Instruct does not.

Note that the two models that did not benefit from RandomWorld training with respect to ToolQA-Easy (Llama-RW-SFT and Qwen-RW-GRPO) have the best and worst performance of the RandomWorld-trained models on the withheld RandomWorld test set (respectively), while the two models that did benefit from RandomWorld training have similar performance on RandomWorld Test (0.714 and 0.721). These results indicate that Llama-RW-SFT (0.855 on RandomWorld Test) is overfitting the RandomWorld distribution, while Qwen-RW-GRPO (0.536) is *under*fitting.

This hypothesis is further supported by Llama-RW-GRPO and Qwen-RW-SFT outperforming Llama-RW-SFT and Qwen-RW-GRPO (respectively) on all nearly all NESTFUL metrics. That Qwen-RW-SFT outperforms Llama-RW-GRPO on ToolQA-Easy is likely due to the higher performance of the base Qwen2.5-7B-Instruct model (0.258) over that of the base Llama-3.1-8B-Instruct (0.236).

This suggests an optimal early-stopping threshold for validation accuracy during RandomWorld training of $\sim$0.718. Rather than increasing validation performance beyond this limit, additional downstream performance gains can be obtained through increasing the difficulty of the RandomWorld environment, by increasing the trajectory skeleton length (ℓ_{max}) and/or applying the scaffolded curriculum learning method proposed in Section 7.

Tools	Tasks	RW Test	ToolQA Easy	ToolQA Hard	NESTFUL F1 Func.	NESTFUL F1 Param.	NESTFUL Part. Acc.	NESTFUL Full. Acc.	NESTFUL Win Rate
100%	100%	**0.536**	0.243	0.063	**0.92**	**0.69**	**0.28**	**0.22**	**0.30**
100%	25%	0.348	**0.250**	0.056	**0.92**	0.68	0.27	0.21	0.29
25%	100%	0.529	0.231	**0.073**	**0.92**	0.68	**0.28**	**0.22**	**0.30**
25%	25%	0.297	0.246	0.059	**0.92**	0.68	0.27	0.21	0.29

Table 3: Scalability study results on the RandomWorld, ToolQA, and NESTFUL benchmarks. The top row (100% tools, 100% tasks) corresponds to the Qwen-RW-GRPO model of Section 4/Table 2.

5 Scalability Study

Next, we examined the effect of number of synthetic tools and tasks on downstream performance.

5.1 Experimental Setup

The first dataset, designed to evaluate the effect of task number, was constructed by randomly removing 75% of the tasks from the original training set used in Section 4, leaving 25% remaining.

The second was designed to evaluate the effect of tool inventory size, and was constructed by removing 75% of the tools from the original training set, then generating 12,000 unique tasks from that restricted tool set. The third was derived from this dataset by randomly removing 75% of those tasks.

We fine-tuned Qwen2.5-7B-Instruct with GRPO on these three datasets, using the same hyperparameter configuration as in Section 4.1. We then evaluated the three models on the benchmarks in Section 4.3, with the same experimental settings.

5.2 Results and Discussion

The results in Table 3 show that the number of tools and tasks the model is trained on affect downstream performance. These results are most pronounced on the RandomWorld test set: Qwen-RW-GRPO performed worse than the base model on ToolQA-Easy—explaining the observed *increased* performance when trained on fewer tasks—and NESTFUL is not fully in-distribution with respect to RandomWorld (as discussed in Sections 4.3 and 4.4), so we would not expect a marked effect on those benchmarks.

Interestingly, training on fewer tools improves ToolQA-Hard performance: the model tuned with 25% tools and 100% tasks improves over Qwen-RW-GRPO by 15%, further narrowing the gap with the SoTA (0.082; see Table 2). We hypothesize that this is due to the shallow ToolQA tool inventory (12 tools; see Section 4.3), which may not benefit from training on a wider range of tools.

On the RandomWorld test set, we observe that the number of tasks has a more substantial impact than number of tools: tuning on 25% of the number of tools and the same number of tasks as Qwen-RW-GRPO only slightly decreases RandomWorld test set performance. Conversely, tuning on 25% of the tasks with the same number of tools results in a more sizable decrease. However, with fewer tasks, the importance of tool inventory size appears to increase: the model tuned with 25% of the tasks and 100% of the tools outperforms that tuned with 25% of the tasks and tools by a relatively wide margin.

6 Conclusion

We introduced RandomWorld (Section 3), a pipeline for the generation of virtually unlimited synthetic tools and tool-use training data—including interactive tools and compositional, non-linear tasks—for both SFT and online RL training. In Section 4, we showed that models can be fine-tuned on RandomWorld training data to improve performance on a variety of downstream benchmarks. Furthermore, a Qwen2.5-7B-Instruct model fine-tuned solely on synthetic RandomWorld data achieves the new SoTA for two metrics on the NESTFUL tool-use benchmark.

The results of our scalability study (Section 5) demonstrate the importance of synthetic data generation for tool-use agents: training on fewer examples leads to decreased downstream performance. The converse of this finding—namely, that training on *more* examples leads to *increased* downstream performance—indicates that the RandomWorld pipeline can continue to be used to further advance the SoTA on tool-use benchmarks, without the need for costly, real-world data curation.

7 Limitations

Experiments. Although we evaluate two model types (Llama-3.1-8B-Instruct and Qwen2.5-7B-

Instruct) and two fine-tuning regimens (SFT and GRPO), our experiments are limited to relatively small 7-8 billion parameter models. In future work, we intend to evaluate a wider range models, in order to verify that the advantages conferred by RandomWorld continue to hold for larger models and a variety of architectural configurations.

Our experiments are also fairly restricted in scale: those conducted in Section 4 involve only 12,000 tasks and 556 tools, primarily due to computational resource limitations. While this is on par with the size of training datasets such as ToolACE (Liu et al., 2024a), the results of Section 5 suggest the importance of training with a larger RandomWorld-generated dataset in future work.

Similarly, we only evaluate our models on three benchmarks. Although our results in Section 4 definitively show the superiority of our method with respect to those downstream tasks, we intend to evaluate RandomWorld-trained models on a larger set of benchmarks in future work, in order to further demonstrate the effectiveness of this approach.

Finally, we do not experiment with more advanced training regimens, such as curriculum learning. Given the demonstrated effectiveness of curriculum learning for training tool-use agents with RL (e.g. Qi et al., 2025) and RandomWorld's built-in compatibility with such methods (see Section 3.3), future work should incorporate the use of curriculum learning into agent training with RandomWorld.

Method. The effectiveness of RandomWorld's tool- and instruction-generation procedures are limited by the effectiveness of the tool- and instruction-generation LLMs (see Section 3.3). Similarly, recall of the instruction-verification LLM—which is intended to filter out insufficiently-informative instructions (as discussed in Section 3.3.3)—is dependent on the choice of model: we have no mechanism to account for cases in which the instruction *is* sufficiently informative, but the instruction-verification LLM is simply unable to complete the task. This may have the undesired effect of unnecessarily filtering out the most difficult tasks.

This limitation could potentially be mitigated through the use of a scaffolded curriculum learning approach, in which the agent *also* serves as the instruction-verification LLM: as the task is made easier for the instruction-verification model (see Section 3.3.3), an agent could still learn from tasks whose instructions it has verified itself.

Under such an approach, the agent would train until it has mastered the data, then be used to verify instructions for more difficult data—on which it would then be further trained on—and so on. We leave the implementation of this approach to future work.

Acknowledgments

We gratefully acknowledge the stimulating research environment of the GRK 2853/1 "Neuroexplicit Models of Language, Vision, and Action", funded by the Deutsche Forschungsgemeinschaft (DFG; German Research Foundation) under project number 471607914.

References

Kinjal Basu, Ibrahim Abdelaziz, Kiran Kate, Mayank Agarwal, Maxwell Crouse, Yara Rizk, Kelsey Bradford, Asim Munawar, Sadhana Kumaravel, Saurabh Goyal, et al. 2024. Nestful: A benchmark for evaluating llms on nested sequences of api calls. *arXiv preprint arXiv:2409.03797*.

Hongru Cai, Yongqi Li, Wenjie Wang, Fengbin Zhu, Xiaoyu Shen, Wenjie Li, and Tat-Seng Chua. 2025. Large language models empowered personalized web agents. In *Proceedings of the ACM on Web Conference 2025*, pages 198–215.

Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V Le, Sergey Levine, and Yi Ma. 2025. Sft memorizes, rl generalizes: A comparative study of foundation model post-training. *arXiv preprint arXiv:2501.17161*.

Tim R Davidson, Benoit Seguin, Enrico Bacis, Cesar Ilharco, and Hamza Harkous. 2025. Orchestrating synthetic data with reasoning. In *ICLR 2025 Workshop on Synth Data*.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. 2025. Retool: Reinforcement learning for strategic tool use in llms. *arXiv preprint arXiv:2504.11536*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2022. Lora: Low-rank adaptation of large language models. *International Conference on Learning Representations*, 1(2):3.

Mengkang Hu, Pu Zhao, Can Xu, Qingfeng Sun, Jianguang Lou, Qingwei Lin, Ping Luo, and Saravan Rajmohan. 2024. Agentgen: Enhancing planning abilities for large language model based agent via environment and task generation. *arXiv preprint arXiv:2408.00764*.

Shijue Huang, Wanjun Zhong, Jianqiao Lu, Qi Zhu, Jiahui Gao, Weiwen Liu, Yutai Hou, Xingshan Zeng, Yasheng Wang, Lifeng Shang, Xin Jiang, Ruifeng Xu, and Qun Liu. 2024. Planning, creation, usage: Benchmarking LLMs for comprehensive tool utilization in real-world complex scenarios. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 4363–4400, Bangkok, Thailand. Association for Computational Linguistics.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. API-bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, Singapore. Association for Computational Linguistics.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025. Search-o1: Agentic search-enhanced large reasoning models. *arXiv preprint arXiv:2501.05366*.

Qiqiang Lin, Muning Wen, Qiuying Peng, Quanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu Zhou, Cheng Cheng, Yin Zhao, Jun Wang, and Zhang Weinan. 2025. Robust function-calling for on-device language model via function masking. In *The Thirteenth International Conference on Learning Representations*.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, et al. 2024a. Toolace: Winning the points of llm function calling. *arXiv preprint arXiv:2409.00920*.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh RN, et al. 2024b. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. *Advances in Neural Information Processing Systems*, 37:54463–54482.

Michael Matthews, Michael Beukman, Chris Lu, and Jakob Foerster. 2024. Kinetix: Investigating the training of general agents through open-ended physics-based control tasks. *arXiv preprint arXiv:2410.23208*.

Drew McDermott, Malik Ghallab, Adele E. Howe, Craig A. Knoblock, Ashwin Ram, Manuela M. Veloso, Daniel S. Weld, and David E. Wilkins. 1998. Pddl-the planning domain definition language.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Sun Xueqiao, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, Jie Tang, and Yuxiao Dong. 2025. WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *The Thirteenth International Conference on Learning Representations*.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741.

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

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16022–16076, Bangkok, Thailand. Association for Computational Linguistics.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. 2020. TRL: Transformer Reinforcement Learning. `https://github.com/huggingface/trl`.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy,

and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in Neural Information Processing Systems*, 32.

Fanjia Yan, Huanzhi Mao, Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. Berkeley function calling leaderboard.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, et al. 2024. xlam: A family of large action models to empower ai agent systems. *arXiv preprint arXiv:2409.03215*.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. 2025. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. *arXiv preprint arXiv:2504.03160*.

Yuxiang Zheng, Shichao Sun, Lin Qiu, Dongyu Ru, Cheng Jiayang, Xuefeng Li, Jifan Lin, Binjie Wang, Yun Luo, Renjie Pan, Yang Xu, Qingkai Min, Zizhao Zhang, Yiwen Wang, Wenjie Li, and Pengfei Liu. 2024. OpenResearcher: Unleashing AI for accelerated scientific research. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 209–218, Miami, Florida, USA. Association for Computational Linguistics.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2023. Toolqa: A dataset for llm question answering with external tools. *Advances in Neural Information Processing Systems*, 36:50117–50143.

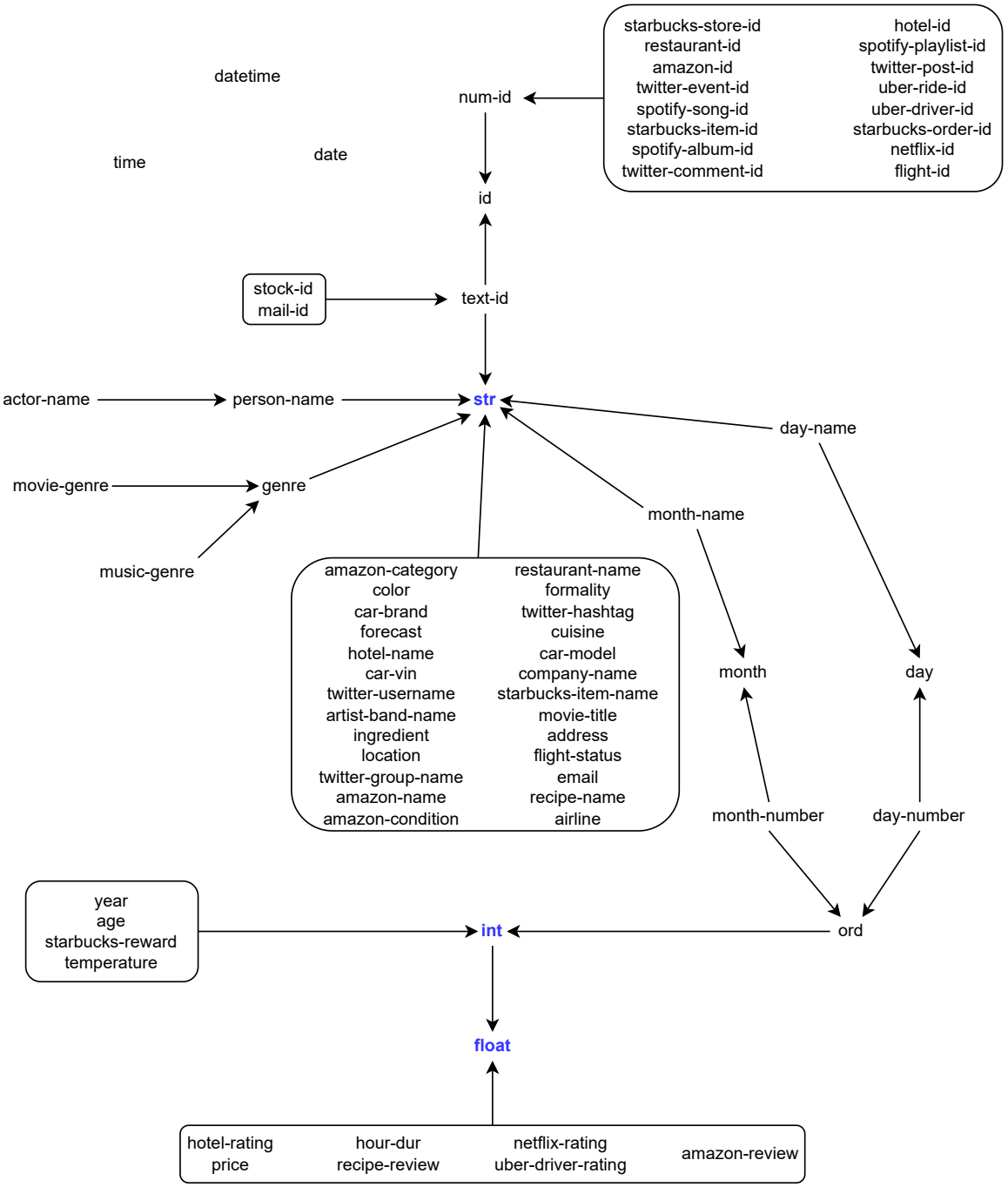

Figure 3: The RandomWorld type hierarchy: actual Python types are bolded and in blue—all other nodes denote custom RandomWorld types. Edges $A \to B$ indicates that A is a subtype of B ($A \leq B$). For the sake of representational simplicity, an edge $\square \to A$ indicates that all types within the box are subtypes of A, without any subtype relations between types within the box: for example, *stock-id*, *mail-id* $\leq$ *text-id*.

Name	Description	Example(s)
actor-name	name of an actor	"Leonardo DiCaprio", "Meryl Streep"
address	location of a building or plot of land	"123 Maple Street, Springfield, IL 62701, USA"
age	age in years	13, 98
airline	name of an airline	"American Airlines", "Delta Air Lines"
amazon-category	shopping category on Amazon	"Books", "Electronics"
amazon-condition	condition of an Amazon item	"New", "Used, Like New"
amazon-id	numerical ID of an Amazon item	430680496270, 478108090872052
amazon-name	name of an Amazon item	"Toilet Paper", "Paper Towels"
amazon-review	rating of an Amazon item	4.8, 0.2
artist-band-name	name of a (music) band	"The Beatles", "Rolling Stones"
car-brand	name of a car manufacturer	"Toyota", "Ford"
car-model	name of a car model	"LaCrosse", "Phantom"
car-vin	vehicle identification number	"gjuqfykjulqsitv7r", "lvc3qd874emg411nl"
color	color	"Red", "Green"
company-name	name of a company	"Apple", "Microsoft"
cuisine	name of a category of food	"Italian", "Chinese"
date	date	"17/8/1103", "20/5/183"
datetime	date and time	"05:01 4/10/1302", "11:33 9/2/100"
day-name	name of a day of the week	"Monday", "Tuesday"
day-number	calendar day number	1, 2
email	content of an email message	"Dear Mr. Johnson, ...", "Hi Sarah, ..."
flight-id	numerical ID of a commercial flight	222101709170966, 9765628923380
flight-status	status of a flight	"On Time", "Cancelled"
forecast	weather forecast	"Clear", "Partly Cloudy"
formality	tone of a text	"formal", "semi-formal"
hotel-id	numerical ID of a hotel	4496768012, 3155376944
hotel-name	name of a hotel	"The Grand Magnolia", "Skyline Retreat"
hotel-rating	rating of a hotel	1.5, 1.0
hour-dur	length of time (in hours)	1.2, 1.0
ingredient	name of an ingredient	"Garlic", "Onion"
location	geographic location	"New York", "Los Angeles"
mail-id	email address	"i8njw1s@oj7y.ca", "k@dy851wvil2vy4z7by.com"
month-name	name of a month	"January", "February"
month-number	calendar month number (1-12)	1, 2
movie-genre	genre of a movie	"Action", "Adventure"
movie-title	title of a movie	"The Shawshank Redemption", "The Godfather"
music-genre	genre of a song or album	"Rock", "Pop"
netflix-id	numerical ID of a movie on Netflix	2737985392929, 1724771805351
netflix-rating	rating of a movie on Netflix	1.8, 0.0
person-name	name of a person	"John Doe", "Sarah Smith"
price	cost of an item	627.49, 4545.56
recipe-name	name of a recipe	"Spaghetti Carbonara", "Chicken Alfredo"
recipe-review	rating of a recipe	1.4, 0.4
restaurant-id	numerical ID of a restaurant	2354517290620, 82682880027029
restaurant-name	name of a restaurant	"The Golden Spoon", "Bella Cucina"
spotify-album-id	numerical ID of a Spotify album	54529623464, 6035921619
spotify-playlist-id	numerical ID of a Spotify playlist	565019246380, 339078364981
spotify-song-id	numerical ID of a song on Spotify	74587698010, 16030267814
starbucks-item-id	numerical ID of a Starbucks product	2847706651, 61785093490348
starbucks-item-name	name of a Starbucks product	"Caramel Macchiato", "Caffè Latte"

Table 4: RandomWorld custom type names, descriptions, and 1-2 example instances (depending on length), sampled from their respective generators (continued in Table 5). Supertypes are not displayed, as their generators and recognizers are inherited from their constituent subtypes.

Name	Description	Example(s)
starbucks-order-id	numerical ID of a Starbucks order	4251443807182, 25283848032
starbucks-reward	number of Starbucks reward points	430, 257
starbucks-store-id	numerical ID of a Starbucks store	128977859410, 915762041007
stock-id	stock ticker symbol of a company	"WPHL", "L"
temperature	temperature	21.5, 37.0
time	time	"23:37", "17:47"
twitter-comment-id	numerical ID of a comment on a Twitter post	8458186204222, 129076645933
twitter-event-id	numerical ID of a Twitter event	537044647554022, 51090184304
twitter-group-name	name of a Twitter group	"TechTalks", "FoodieFriends"
twitter-hashtag	Twitter hashtag	"#FollowFriday", "#TechNews"
twitter-post-id	numerical ID of a Twitter post	34268389893, 73626538108
twitter-username	username of a Twitter account	"JohnDoe", "SarahSmith1"
uber-driver-id	numerical ID of an Uber driver	617303431222, 6236243225
uber-driver-rating	rating of an Uber driver	4.2, 2.8
uber-ride-id	numerical ID of an Uber ride	6346853814, 90798212714679
year	year	1841, 1988

Table 5: Table 4 continued.

Name: Signature	Description
actor-movie: $\textit{actor-name} \rightarrow \textit{list}(\textit{movie-title})$	retrieves the names of movies in which a particular actor plays
age-movie: $\textit{union}(\textit{movie-title}, \textit{netflix-id}) \rightarrow \textit{age}$	retrieves the age for which the input movie is appropriate
daily-ingredient-specials: $\textit{day-name} \rightarrow \textit{dict}(\textit{ingredient}, \textit{restaurant-name})$	returns a dictionary of daily special ingredients and their associated restaurants
dining-time-matcher: $\textit{age} \rightarrow (\textit{time} \times \textit{restaurant-name})$	suggests dining time and restaurant based on age preferences
frequent-day-finder: $\textit{dict}(\textit{restaurant-id}, \textit{day-name}) \rightarrow \textit{day-name}$	returns the most common day from the restaurant-day mapping
holiday-checker: $\textit{location} \rightarrow \textit{date}$	returns the most recent public holiday at a location
hq-locator: $\textit{company-name} \rightarrow \textit{location}$	returns the headquarters location of the input company
movie-len: $\textit{hour-dur} \times \textit{hour-dur} \rightarrow \textit{list}(\textit{movie-title})$	retrieves movies that last between the specified time range
recipe-suggester: $\textit{day} \rightarrow \textit{recipe-name}$	suggests a recipe based on the given day
starbucks-locator: $\textit{location} \rightarrow \textit{starbucks-store-id}$	returns the nearest Starbucks store ID for a given location
stock-price: $\textit{stock-id} \times \textit{date} \rightarrow \textit{price}$	retrieves the price for a given stock ticker symbol on the specified date
stock-ticker: $\textit{company-name} \rightarrow \textit{stock-id}$	returns the stock ticker symbol associated with a given company name

Table 6: Example tool names/signatures and descriptions generated by RandomWorld (see Section 3.2).

A Type System

The majority of the RandomWorld type system is described in detail in Section 3.1, Figure 3, and Tables 4/5: this section is primarily dedicated a more detailed discussion of the constructed types discussed in Section 3.1.

A.1 Subtype Relations

Subtype relations between atomic types (i.e. those given in Figure 3) are defined manually; subtype relations between constructed types are defined recursively in terms of their component types, where the recursion is terminated by the atomic types.

Subtype relations between list types are defined as in Equation 1.

$$\mathit{list}(A) \leq B := \exists C(B = \mathit{list}(C) \wedge A \leq C) \quad (1)$$

This is to say that $\mathit{list}(A)$ is a subtype of B if B is of the form $\mathit{list}(C)$ and A is a subtype of C: list types cannot be supertypes or subtypes of non-list types.

Subtype relations between dictionary types are defined as in Equation 2.

$$\begin{aligned}&\mathit{dict}(A, B) \leq C := \\ &\quad \exists X, Y(B = \mathit{dict}(X, Y) \wedge A \geq X \wedge B \leq Y)\end{aligned} \quad (2)$$

This is to say that $\mathit{dict}(A, B)$ is a subtype of C if C is of the form $\mathit{dict}(X, Y)$, $B \leq Y$, and $X \leq A$: as mappings, dictionary types are contravariant in their first argument. As with list types, dictionary types cannot be supertypes or subtypes of non-dictionary types.

Sub-/super-type relations for union types are defined as in Equation 3.

$$\mathit{union}(A, B) \leq C := A \leq C \wedge B \leq C \quad (3a)$$
$$C \leq \mathit{union}(A, B) := C \leq A \vee C \leq B \quad (3b)$$

Note that union types are *not* discriminated unions (coproducts)—for example:

$$\begin{aligned}&\mathit{union}(A, \mathit{union}(B, C)) = \mathit{union}(\mathit{union}(A, B), C) \\ &\quad = \mathit{union}(\mathit{union}(A, B), \mathit{union}(B, C))\end{aligned}$$

A.2 Generators

Type generators for atomic types are defined manually; generators for constructed types are defined recursively in terms of their components.

Generators $G_{\mathit{union}(A,B)}$ for union types $\mathit{union}(A, B)$ simply randomly sample one of their arguments $X \sim \{A, B\}$, then sample an output from the generator G_X.

Generators $G_{\mathit{list}(A)}$ for list types $\mathit{list}(A)$ first sample a length ℓ (within a pre-defined range), then construct a list by sampling ℓ outputs from G_A.

Similarly, generators $G_{\mathit{dict}(A,B)}$ for dictionary types $\mathit{dict}(A, B)$ sample a length ℓ, then construct a dictionary by sampling ℓ pairs (a, b), with $a \sim G_A$ and $b \sim G_B$.

A.3 Recognizers

As discussed in Section 3.1, type recognizers are boolean-valued functions $R_A\colon \text{Any} \to \{0, 1\}$, such that $R_A(x) = 1$ if x is of type A, and $R_A(x) = 0$ otherwise.

As with generators, type recognizers for atomic types are defined manually—e.g. through regular expressions, set membership checks, etc., depending on the type—while recognizers R_A for constructed types A are defined recursively in terms of their components, as in Equation 4, where $I(x, A) := \mathit{isinstance}(x, A)$ for a Python type A.

$$R_{\mathit{union}(A,B)}(x) := R_A(x) \vee R_B(x) \quad (4a)$$
$$R_{\mathit{list}(A)}(x) := I(x, \mathit{list}) \wedge \bigwedge_{i=0}^{|x|-1} R_A(x_i) \quad (4b)$$
$$\begin{aligned}&R_{\mathit{dict}(A,B)}(x) := \\ &\quad I(x, \mathit{dict}) \wedge \bigwedge_{(a,b)\in \mathit{items}(x)} (R_A(a) \wedge R_B(b))\end{aligned} \quad (4c)$$

A.4 Supertypes

As discussed in Section 3.1, supertypes in the hierarchy in Figure 3 inherit subtype relations, recongizers, and generators from their subtypes—with the exception of Python types, whose recognizers are defined by the built-in *isinstance* function.

For a given type A with subtypes $B_1, \ldots, B_n$, we implement subtype relation and recongizer/generator inheritance by implicitly defining A as $\mathit{union}(B_1, \ldots, B_n)$.

B Example Synthetic Tasks

B.1 GRPO (Llama-RW-GRPO)

Agent-generated tokens are highlighted, and correct/incorrect submissions are indicated with a green check/red X (✓/X), respectively.

The tools at your disposal:
netflix_movie-age: retrieves movie titles appropriate for children at or under the specified age
netflix_actor-movie: retrieves the name of the actors that play in a given movie
netflix_age-movie: retrieves the age for which the input movie is appropriate
netflix_movie-long: retrieves a movie that lasts more than n hours
mail-formality: retrieves emails in the user's inbox by the formality of the greeting
netflix_genre-movie: retrieves the genre of a specified movie
netflix_list: retrieves the ID of the nth-most recently-added movie to the user's list
netflix_movie-genre: retrieves a movie recommendation based on the input genre
spotify_recent: retrieves the ID of the user's nth-most recently listened-to song
spotify_top: retrieves the ID of the nth-most listened-to song in the user's playlist

Find a movie longer than 1.2 hours, get its genre, recommend another movie of the same genre, determine the appropriate age for that movie, and then find a list of movies suitable for that age. My Netflix username is w1j6w9d294ubn2jutf.

First, I need to log in to Netflix.
<request><password-manager>'netflix','wlj6w9d294ubn2jutf'<call>'rbn3atn50ofc'<response>
<request><netflix_login>'w1j6w9d294ubn2jutf','rbn3atn50ofc'<call> SUCCESS<response>

Now, I need to find a movie longer than 1.2 hours.
<request><netflix_movie-long>1.2<call>'Compromises'<response>

Now, I need to get the genre of the movie "Compromises".
<request><netflix_genre-movie>"Compromises"<call>'Drama'<response>

Now, I need to recommend another movie of the same genre.
<request><netflix_movie-genre>'Drama'<call>'Fight Club'<response>

Now, I need to determine the appropriate age for the movie "Fight Club".
<request><netflix_age-movie>"Fight Club"<call>18<response>

Now, I need to find a list of movies suitable for children at or under the age of 18.
<request><netflix_movie-age>18<call>['The Dark Knight', 'Pulp Fiction', 'Up', 'Mad Max: Fury Road', 'Toy Story', 'Moana', 'Zootopia', 'Frozen', 'The Lion King', 'Fight Club', 'The Incredibles', 'Guardians of the Galaxy']<response>

Result=['The Dark Knight', 'Pulp Fiction', 'Up', 'Mad Max: Fury Road', 'Toy Story', 'Moana', 'Zootopia', 'Frozen', 'The Lion King', 'Fight Club', 'The Incredibles', 'Guardians of the Galaxy']<submit> ✓

The tools at your disposal:
day-to-name-converter: Converts day input to corresponding weekday name.
comment-month-extractor: Extracts the month number from a Twitter comment's timestamp.
comment-day-finder: Returns the day of the week a Twitter comment was posted.
flight-arrival-time: Returns the scheduled arrival time for a given flight ID.
recipe-rating-fetcher: Retrieves average user rating for a specified recipe.
hashtag-comment-finder: Finds a comment ID from a post with the specified hashtag.

Find the weekday name on which a comment with the hashtag '#AdventureAwaits' was posted.

<request><hashtag-comment-finder>""AdventureAwaits""<call>
Tool error: invalid input format<error>

<request><hashtag-comment-finder>""#AdventureAwaits""<call>
Tool error: invalid input format<error>

<request><hashtag-comment-finder>"#AdventureAwaits"<call>97876083389<response>

<request><comment-day-finder>97876083389<call>'Tuesday'<response>

Result='Tuesday'<submit> ✓

The tools at your disposal:
item-price: retrieves all items available at a specified price
cart-checker: retrieves the ID of the nth-most recently-added item to the user's cart
min: returns the minimum of a list of values
price-item: retrieves all prices available for a specified item
corporate-color-decoder: Determines the primary brand color associated with a given company name.
last-serve-time: Returns the last service time for a specified restaurant ID

Find all items available at the minimum price of laptops.

<request><price-item>calculator-min(price-item('laptop'))<call>
Tool error: invalid input format<error>

Result=Tool error: invalid input format<submit> X

B.2 SFT

Tokens that contribute to the loss are highlighted, all other tokens are masked from the loss function.

The tools at your disposal:
daily-recipe-suggester: Suggests a recipe based on the given day.
sub: subtracts the second argument from the first
extract-year-from-datetime: Extracts the year from a given datetime input.
airline-founding-date: Returns the founding date of the specified airline.
hashtag-trend-day: Returns the peak trending day for a given Twitter hashtag.
recipe-rating-fetcher: Retrieves average user rating for a specified recipe.
tweet-timestamp-retriever: Fetches the date and time a Twitter post was created using its ID.
restaurant-age-checker: Returns the number of years since the restaurant's establishment.
historical-event-date: Returns a significant historical date for the specified location.
cuisine-to-location: Suggests a city known for the specified cuisine.
store-temp-checker: Returns current temperature at specified Starbucks store location.
tweet-locator: Fetches a recent Twitter post ID from the specified location.
next-day-finder: Returns the next day of the week for the given input day.
int-to-weekday: Converts an integer to a corresponding day of the week, modulo 7.
tweet-day-extractor: Returns the calendar day a tweet was posted.
movie-soundtrack-matcher: Returns a Spotify playlist ID inspired by the given movie title.

Find a recipe based on the weekday corresponding to the year a recent tweet was posted from a city known for 'Afghan' cuisine, retrieve its rating, and subtract the year from the rating.

<request><cuisine-to-location>'Afghan'<call>'Madison'<response>
<request><tweet-locator>'Madison'<call>4811550653611<response>
<request><tweet-timestamp-retriever>4811550653611<call>'10:05 2/8/2041'<response>
<request><extract-year-from-datetime>'10:05 2/8/2041'<call>2011<response>
<request><int-to-weekday>2011<call>'Friday'<response>
<request><daily-recipe-suggester>'Friday'<call>'Cinnamon Rolls'<response>
<request><recipe-rating-fetcher>'Cinnamon Rolls'<call>2.4<response>
<request><sub>2.4,2011<call>-2008.6<response>
Result=-2008.6<submit>

The tools at your disposal:
hotel-id-lookup: Retrieves the unique ID for a specified hotel name.
uber-driver-lookup: Retrieves the driver ID associated with a specific Uber ride ID.
get-driver-rating: Fetches the current rating for a specified Uber driver by their ID.
tone-refiner: Adjusts input tone to a similar, nuanced tone.
starbucks-item-suggester: Suggests a complementary Starbucks item based on the input item name.
starbucks-calorie-match: Suggests a Starbucks item based on the calorie range represented by the float.

Suggest a Starbucks item based on the calorie range that matches the rating of the driver for Uber ride ID 6728308325.

<request><uber-driver-lookup>6728308325<call>78997392675<response>
<request><get-driver-rating>78997392675<call>1.4<response>
<request><starbucks-calorie-match>1.4<call>'Pumpkin Spice Latte'<response>
Result='Pumpkin Spice Latte'<submit>

The tools at your disposal:
spotify-album-release-date: Returns the release date and time of a Spotify album by its ID.
tweet-day-extractor: Returns the calendar day a tweet was posted.
temp-at-location: Returns current temperature for a specified address.
historical-event-date: Returns a significant historical date for the specified location.
mul: multiplies two values together
spotify-album-finder: Returns Spotify album ID for a given artist or band name.
add: sums two values together
extract-year-from-datetime: Extracts the year from a given datetime input.
movie-recommender: Suggests a similar movie based on the input title.
airline-founding-date: Returns the founding date of the specified airline.

Calculate the product of the release year of a Kaiser Chiefs album and the sum of that year with itself.

<request><spotify-album-finder>'Kaiser Chiefs'<call>27314933599224<response>
<request><spotify-album-release-date>27314933599224<call>'04:08 19/3/1720'<response>
<request><extract-year-from-datetime>'04:08 19/3/1720'<call>1458<response>
<request><add>1458,1458<call>2916<response>
<request><mul>2916,1458<call>4251528<response>
Result=4251528<submit>

The tools at your disposal:
starbucks-item-suggester: Suggests a complementary Starbucks item based on the input item name.
email-greeting-generator: Generates a personalized email template based on the recipient's name.
starbucks-calorie-match: Suggests a Starbucks item based on the calorie range represented by the float.
seasonal-recipe-suggester: Suggests a recipe based on seasonal ingredients available in the given month.
rating-to-color: Maps Netflix ratings to a corresponding color for visual representation.
starbucks-points-calculator: Returns reward points for a given Starbucks product ID.
daylight-duration-estimator: Estimates average daylight hours for a given month.
recipe-to-cuisine: Identifies the cuisine category based on the given recipe name.
spotify-genre-finder: Returns the genre of a song using its Spotify ID.
age-to-genre-mapper: Suggests music genre based on age demographic preferences.
peak-tourism-month: Returns the peak tourism month for a given location.
hashtag-peak-time: Returns peak engagement time for a given Twitter hashtag.
age-calculator: Calculates age in years from a given historical datetime to the current date.
cuisine-to-location: Suggests a city known for the specified cuisine.

Suggest a complementary Starbucks item based on a seasonal recipe for October, considering the peak tourism month and average daylight hours of a city known for the recipe's cuisine.

<request><seasonal-recipe-suggester>'October'<call>'Tom Yum Soup'<response>
<request><recipe-to-cuisine>'Tom Yum Soup'<call>'Flemish'<response>
<request><cuisine-to-location>'Flemish'<call>'Jakarta'<response>
<request><peak-tourism-month>'Jakarta'<call>'March'<response>
<request><daylight-duration-estimator>'March'<call>2.2<response>
<request><starbucks-calorie-match>2.2<call>'Hazelnut Hot Chocolate'<response>
<request><starbucks-item-suggester>'Hazelnut Hot Chocolate'<call>'Peach Green Tea Lemonade'<response>
Result='Peach Green Tea Lemonade'<submit>

The tools at your disposal:
recipe-swapper: Suggests an alternative recipe based on the input recipe name.
add: sums two values together
ingredient-to-amazon-id: Maps ingredient names to corresponding Amazon product IDs for easy shopping.
flight-price-checker: Retrieves the current ticket price for a specified flight ID.
amazon-review-fetcher: Retrieves average review rating for a specified Amazon item by its ID.
sub: subtracts the second argument from the first

Calculate the negative of the average review rating for the Amazon item with ID 712523906030543.

<request><amazon-review-fetcher>712523906030543<call>3.1<response>
<request><add>3.1,3.1<call>6.2<response>
<request><sub>3.1,6.2<call>-3.1<response>
Result=-3.1<submit>

C Information Leakage

To investigate the potential for tool names and/or descriptions leaking into instructions during the generation process described in Section 3.3.3, we manually analyzed 200 randomly-sampled task/instruction pairs from the RandomWorld train split of Section 4. In particular, we flagged any substrings of the generated task instructions that were either similar or identical to the names or descriptions of tools in the trajectory skeleton provided to the instruction-generation LLM.

Of the 200 instructions, we found potential information leakage in 133 instances (66.5%). Examples are given below: suspicious instruction substrings and the corresponding substrings in the tool names/descriptions are highlighted.

Tool Descriptions:
daily-trend-hashtag: Returns trending Twitter hashtag for the specified day of the week.
int-to-weekday: Converts an integer to a corresponding day of the week, modulo 7.
month-to-number: Converts a month's name to its corresponding calendar number.
hashtag-to-post-id: Fetches a recent post ID using the specified hashtag.

Instruction:
Find a recent Twitter post ID using the trending hashtag for the day of the week corresponding to the month of 'February'.

Tool Descriptions:
tweet-activity-time: Returns the most active time for a given Twitter username.
time-to-playlist: Returns a playlist ID based on the time of day for mood setting.
hashtag-influencer-finder: Finds top influencer associated with a given hashtag.
trend-tag-locator: Returns trending Twitter hashtag for a specified location.
genre-to-netflix-id: Returns a Netflix movie ID based on the specified genre.
netflix-duration-lookup: Returns the duration in hours for a given Netflix movie ID.
playlist-genre-identifier: Identifies the dominant genre of a Spotify playlist using its ID.
add: sums two values together

Instruction:
Find the duration of a Netflix movie, based on the genre of a Spotify playlist that matches the most active time of a top influencer associated with a trending hashtag in 'Louisville', and then double that duration.

Tool Descriptions:
future_time_calculator: Adds hour duration to current time, returning future date and time.
datetime_to_month: Extracts and returns the month name from a given datetime input.
driver_shift_duration: Returns the duration of the last completed shift for a given Uber driver.
div: divides the first argument by the second
mul: multiplies two values together
Instruction:
Determine the month name of the current date and time after adding the duration of the last completed shift for Uber driver with ID 963846425985.

Despite this admittedly high rate of information leakage, we again argue that the results of our RandomWorld-trained models in Table 2 indicate that any information leakage that may have occurred did not substantially detract from the models' performance.

Furthermore, none of the examples that we manually inspected exhibited sufficient information leakage to entirely give away the correct tool call sequence. Note that the RandomWorld-trained models in Table 2 have substantially higher performance on the withheld RandomWorld test set than those not trained on RandomWorld: Llama-RW-GRPO (0.721), Llama-RW-SFT (0.855), Qwen-RW-GRPO (0.536), and Qwen-RW-SFT (0.714) all outperform Llama-3.1-8B-Instruct (0.483), ToolACE-8B (0.170), Qwen2.5-7B-Instruct (0.109), Hammer2.0-7B (0.011), and GPT-3.5 (0.348) by a wide margin. We argue that

the poor performance of the models not trained on RandomWorld indicates that any information leakage in the generated RandomWorld instructions did not render these tasks trivial.

D GRPO Training Details

As discussed in Section 3, we implement the policy-environment interaction via TRL text environments during GRPO training in Section 4. The prompt/question q consists of two few-shot examples, the tool inventory (and distractor tools), and the instruction.

The policy-generated text and the tool outputs together (all orange and blue highlighted text in Figure 1) are taken as the completion to q. However, we employ the TRL text environment's response masking feature, so that the GRPO loss is only calculated with respect to the policy-generated text (all orange text in Figure 1).

As in the RandomWorld test set described in Section 4.3, the policy model was limited to a maximum of 15 turns (tool calls or attempted tool calls) and 128 new tokens per turn before the trajectory is aborted, returning a reward of 0.0.

We trained all models (including SFT) on two H100 GPUs, with generation parallelization during GRPO training for speedup.