Sample, Predict, then Proceed: Self-Verification Sampling for Tool Use of LLMs

Shangmin Guo[1*†] **Omar Darwiche Domingues**[2] **Raphaël Avalos**[3†]
Aaron Courville[4] **Florian Strub**[2]
[1]University of Edinburgh [2]Cohere [3]Vrije Universiteit Brussel [4]Université de Montréal

Abstract

Tool use in stateful environments presents unique challenges for large language models (LLMs), where existing test-time compute strategies relying on repeated trials in the environment are impractical. We propose dynamics modelling (DyMo), a method that augments LLMs with a state prediction capability alongside function calling during post-training. This enables LLMs to predict the future states of their actions through an internal environment model. On the Berkeley Function Calling Leaderboard V2, DyMo improves success rates and significantly reduces hallucinations. We further integrate the internal environment model into self-verification sampling (SVS), and show that this substantially improves pass^k over number of trials k, and allows the model to refuse unreliable outputs. Together, DyMo and SVS greatly enhance the effectiveness and reliability of LLMs for tool use. We believe this work charts a path towards scalable planning RL methods for LLM inference without repeatedly querying the oracle environment.

1 Introduction

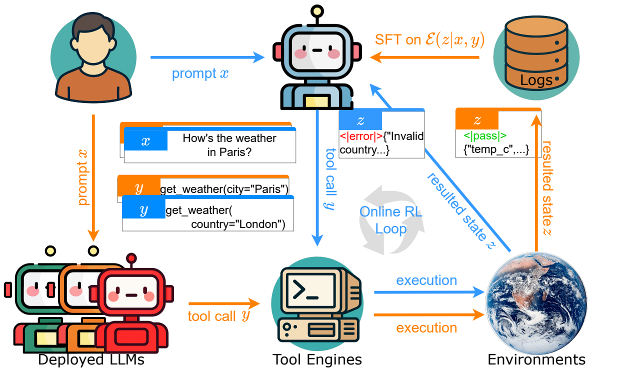
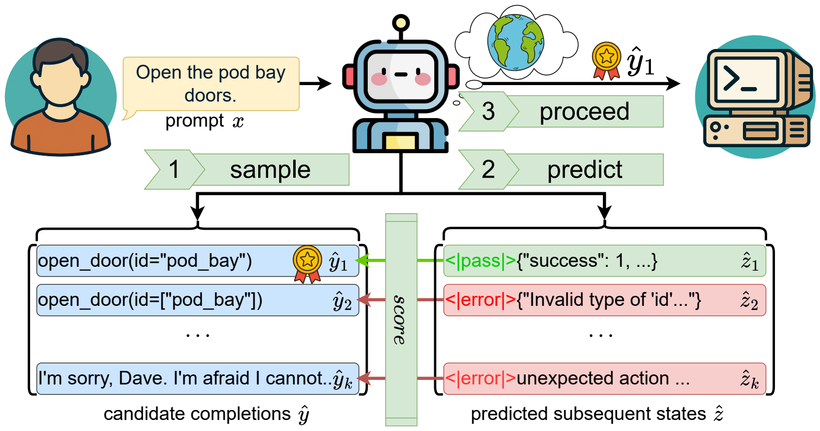

prompt x
SFT on $\mathcal{E}(z|x,y)$
Logs
prompt x
x How's the weather in Paris?
y get_weather(city="Paris")
y get_weather(country="London")
tool call y
z <|error|>{"Invalid country...}
z <|pass|> {"temp_c",...}
resulted state z
Online RL Loop
resulted state z
tool call y
execution
execution
Deployed LLMs
Tool Engines
Environments

Figure 1: The proposed dynamics modelling (DyMo) that trains an LLM to predict the resulted states of the environment after tools execute function calls via either SFT (orange arrows) on run-logs or over online RL loops (blue arrows).

Large language models (LLMs) have demonstrated remarkable performance in a wide range of applications [1–6]. In addition to conventional natural language tasks, recent advances have shown that LLMs also achieve breakthrough performance in formal language tasks, notably code generation [7–9] and tool use [10–12]. Recent work has shown that scaling the test-time compute can further improve the performance of LLMs on complex tasks such as mathematical reasoning [13–17]. To achieve better performance by scaling up test-time compute, existing methods assume that a verifier, e.g. a process reward model (PRM) or an outcome reward model (ORM), can be queried multiple times during inference [11, 13, 16, 14].

However, many real-world applications may not rely on a verifier to improve test-time sampling, especially when the LLM interacts with the world as in Agentic scenarios. One may not execute k

[*]Correspondence author: `s.guo@ed.ac.uk`.
[†]Work done during internship at Cohere.



Open the pod bay doors.
prompt x
1 sample
2 predict
3 proceed
$\hat{y}_1$
open_door(id="pod_bay") $\hat{y}_1$
open_door(id=["pod_bay"]) $\hat{y}_2$
I'm sorry, Dave. I'm afraid I cannot.. $\hat{y}_k$
score
<|pass|>{"success": 1, ...} $\hat{z}_1$
<|error|>{"Invalid type of 'id'..."} $\hat{z}_2$
<|error|>unexpected action ... $\hat{z}_k$
candidate completions $\hat{y}$
predicted subsequent states $\hat{z}$

Figure 2: The proposed self-verification sampling (SVS) strategy for test-time compute scaling. For a given user prompt $\boldsymbol{x}$, the model: 1) generates k candidate completions $\hat{\boldsymbol{y}}$; 2) predicts the subsequent states $\hat{\boldsymbol{z}}$ of the k candidates; 3) selects a completion to output by a specified scoring function *score*.

payments and be satisfied that one of the payments is correct among the k ones, whereas one may verify k times if a mathematical solution is correct without ramifications. In this spirit, we consider tool-use tasks, where the agent must execute a single trajectory. In particular, such constraints are often inherent to the **statefulness** of environments, i.e., the environment status states after executing an action and cannot be easily reverted - the bank account is reduced after a payment!

Inspired by the Generative Verifier [18] (GenRM), which formulates the reward function as a next token prediction task, as illustrated in Figure 1, we propose **dynamics modelling (DyMo)** to fine-tune LLMs to generate not only the functions calls for a given user prompt, but also the *subsequent states* of tool engines after executing the generated function calls. This has two advantages: 1) at training time, this state prediction provides an additional training signal; 2) at test time, this state prediction can be used in the decision-making process to execute the roll-out, similar to one-step planning methods.

We first explore the impact of DyMo into both the supervised fine-tuning (SFT) and RL stages in LLM post-training [3–5], and investigate its effectiveness. Our results on the Berkeley Function Calling Leaderboard [19] V2 (BFCL-V2) show that DyMo alleviates the hallucination problem of the SFTed model, and improves the success rate of the RLed models. Incorporating DyMo, our results suggest that an 8B model, when given access to the environment during training, can match and occasionally surpass the performance of GPT-4o on BFCL-V2.

Second, we explore the planning capabilities of DyMo through **self-verification sampling (SVS)** strategy [20] at test time. Specifically, the models generate k tool calls for a given user prompt, predict the respective states resulting from those actions, and proceed with the most promising trajectory based on a ranking mechanism: *sample*, *predict*, then *proceed*. Our experiments demonstrate that (i) increasing the number of trajectories keeps increasing the LLMs score, (ii) the outcome of the state prediction can be used to select a successful trajectory *without* access to the oracle environment, thereby offering a novel schema for scaling test-time compute in stateful environments. Furthermore, SVS enables models to effectively *"refuse"* requests that exceed their capabilities based on their state prediction, substantially improving the precision of the final output. We interpret this precision as *"reliability"*, as it represents the proportion of outputs verified as correct by the oracle environment.

In summary, the proposed DyMo method coupled with the SVS strategy significantly enhances the success rate and reliability of LLMs in tool use tasks.

2 Background

Tool Use by LLMs: Recent works have demonstrated the capability of LLMs to achieve notable performance in API usage through supervised fine-tuning (SFT) using demonstrations provided either by human experts or generated by advanced models such as GPT-4 [21–23]. This capability positions LLMs as back-ends for agents interacting with environments consisting of various tools [19, 24, 25] and simulated user interactions [26]. However, existing approaches are mainly based on imitation learning for training [21–23], while the evaluation relies on interactions between environments and LLMs [26, 19]. Similar to some recent works [27, 28], we focus on learning directly through interacting with the environments, as detailed in Section 3.2.

Reinforcement Learning for Fine-tuning LLMs: Existing RL methodologies for fine-tuning LLMs primarily address alignment tasks [29–31] or reasoning-oriented tasks, such as mathematics and programming challenges [32]. Nonetheless, we posit that RL techniques can effectively extend to tool use scenarios, especially when scaling the quantity of generated tool interactions, given that LLMs have already achieved promising performance in real-world tool use tasks [33, 25, 26]. Furthermore, recent studies indicate a substantial performance gap between online/on-policy RL methods and their offline/off-policy counterparts [34–37]. Although rigorous online interactions can be traded for enhancing wall-clock efficiency, strictly online RL methods still represent an optimal Pareto frontier [37, 38]. Hence, to fully harness the capabilities of RL in tool use contexts, our experiment setup is strictly online and on-policy in this work. Additionally, our method enables models to do one-step planning based their internal learnt environment model during inference time, as illustrated in Section 3.3.

Test-time Compute Scaling: It is well-established that LLMs enhance their performance on logical reasoning tasks by generating extended responses that include explicit intermediate reasoning steps [39]. Further research highlights the importance of explicitly learning these intermediate reasoning stages guided by Policy Reward Models (PRMs) to achieve superior outcomes [14]. While scaling test-time computes by lengthening generated completions has proven beneficial [40, 16], environmental interactions remain critical for achieving optimal results in agent-based tasks [11]. Recent advances also investigate multiple self-rewarding [17], or self-verification [20] steps to scale test-time compute in mathematical reasoning contexts. Unlike these works which query the environment multiple times during inference, we propose to utilise the internal environment models of LLMs to increase the number of completions for scaling test-time computes, as introduced in Section 3.3.

3 Methodology

3.1 Formulation

We used pre-trained Transformer [41] models π_θ parameterised by θ that predict tokens in an autoregressive manner. After post-training by SFT and RL and given a user prompt $\boldsymbol{x}$, the models can then generate completions/responses $\boldsymbol{y}$ from the distribution $\boldsymbol{y} \sim \pi_\theta(\cdot|\boldsymbol{x})$. Since we focus on the tool use scenario in this work, we assume the user prompts $\boldsymbol{x}$ are all about requesting function calls, whereas the completions $\boldsymbol{y}$ can be either natural languages or formatted formal languages. For completions that call functions, they will then be passed to the environment $\mathcal{E}$ to execute, and the resulted state is $\boldsymbol{z} = \mathcal{E}(\boldsymbol{x}, \boldsymbol{y})$ whose complete set is $\mathbb{Z}$. Note that using no-tool environment is sometimes available in some experiments measuring hallucination.

Following RL terminology, we refer to $\boldsymbol{x}$ as the input *state*, $\boldsymbol{y}$ as the generated *action* from the model π_θ, and $\boldsymbol{z}$ as the resulted next state. The transition dynamics are specified by the environment $\mathcal{E}$, and the reward function $r : \mathbb{Z} \mapsto [0, 1]$ assigns a binary score to a pair $(\boldsymbol{x}, \boldsymbol{y})$ according to their resultant state $\boldsymbol{z}$. From this RL perspective, our model π_θ can:

- generate a tool call (action) $\boldsymbol{y}$ given a user prompt (state) $\boldsymbol{x}$ as input state, i.e. $\boldsymbol{y} \sim \pi_\theta(\cdot|\boldsymbol{x})$;
- predict next-state $\boldsymbol{z}$ given a user prompt $\boldsymbol{x}$ and a tool call $\boldsymbol{y}$, i.e. $\boldsymbol{z} \sim \pi_\theta(\cdot|\boldsymbol{x}, \boldsymbol{y})$.

3.2 DyMo: Dynamics Modelling

The learning objective of the proposed DyMo is not only the tool use function but the environment function $\mathcal{E}$. As illustrated below, we introduce the DyMo into both the SFT and RL stages.

3.2.1 Dynamics Modelling by Supervised Fine-tuning

During the SFT stage, we construct two distinct datasets — one for the tool use function and one for the environment function — which are described in detail below.

For the tool use function, we train the model π_θ on a dataset of function calls represented by function call (fc) pairs in the form `<prompt,completion>`, i.e. $\mathbb{D}_{\text{fc}} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N_{\text{fc}}}$. To train the model on these pairs, we minimise the cross-entropy loss [42] of the model's completion prediction distribution $\pi_\theta(\cdot|\boldsymbol{x})$ over them:

$$\mathcal{L}_{\text{FC}}\left(\mathbb{D}_{\text{fc}};\theta\right) = -\sum_{i=1}^{N_{\text{fc}}}\sum_{t=1}^{T_{\boldsymbol{y}_i}} \log \boldsymbol{\pi}_\theta\left(y_{i,t}|\boldsymbol{x}_i, \boldsymbol{y}_{i,<t}\right) \tag{1}$$

where $y_{i,t}$ is the t-the element in the target completion $\boldsymbol{y}_i$, $\boldsymbol{y}_{i,<t}$ represents the partial target sequence preceding y_t, and $T_{\boldsymbol{y}_i}$ is the length of $\boldsymbol{y}_i$.

Regarding the environment function, we represent it by a dataset of state prediction (sp) triplets in the form `<prompt,completion,result>`, i.e. $\mathbb{D}_{\text{sp}} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i)\}_{i=1}^{N_{\text{sp}}}$. Such data can be gathered and curated from the accumulated *run logs* of the target environment function $\mathcal{E}$, which we argue is a under-explored source for data scaling. Similar to the tool use function, we minimise the cross-entropy loss of the model's state prediction distribution $\boldsymbol{\pi}_\theta(\cdot|\boldsymbol{x}, \boldsymbol{y})$ over these triplets:

$$\mathcal{L}_{\text{SP}}(\mathbb{D}_{\text{sp}};\theta) = -\sum_{i=1}^{N_{\text{sp}}}\sum_{t=1}^{T_{\boldsymbol{z}_i}} \log \boldsymbol{\pi}_\theta\left(z_{i,t}|\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_{i,<t}\right) \tag{2}$$

where the indices i and t follow the same meanings as in Equation 1, and $T_{\boldsymbol{z}_i}$ is the length of $\boldsymbol{z}_i$.

3.2.2 Dynamics Modelling over Online Reinforcement Learning

In addition to the SFT stage, DyMo can also be incorporated into the RL fine-tuning of LLMs.

Starting from a prompt set $\mathbb{D}_{\text{rl}} = \{\boldsymbol{x}_i\}_{i=1}^{N_{\text{rl}}}$, we first sample *two* completions from the model, i.e. $\hat{\boldsymbol{y}}_i^1, \hat{\boldsymbol{y}}_i^2 \sim \boldsymbol{\pi}_\theta(\cdot|\boldsymbol{x}_i)$. The two completions along with the prompt $\boldsymbol{x}_i$ are then passed as inputs to the environment function to get the next states, i.e. $\boldsymbol{z}_i^1 = \mathcal{E}(\boldsymbol{x}_i, \hat{\boldsymbol{y}}_i^1)$ and $\boldsymbol{z}_i^2 = \mathcal{E}(\boldsymbol{x}_i, \hat{\boldsymbol{y}}_i^2)$. Binary scores are then assigned to the `<prompt,completion>` pairs by the reward function r, i.e. $r_1 = r(\boldsymbol{x}_i, \hat{\boldsymbol{y}}_i^1)$ and $r_2 = r(\boldsymbol{x}_i, \hat{\boldsymbol{y}}_i^2)$. Subsequently, we sample predicted next states $\hat{\boldsymbol{z}}_i^1$ and $\hat{\boldsymbol{z}}_i^2$ from the model, i.e. $\hat{\boldsymbol{z}}_i^1 \sim \boldsymbol{\pi}_\theta(\cdot|\boldsymbol{x}_i, \hat{\boldsymbol{y}}_i^1)$ and $\hat{\boldsymbol{z}}_i^2 \sim \boldsymbol{\pi}_\theta(\cdot|\boldsymbol{x}_i, \hat{\boldsymbol{y}}_i^2)$, to track the state prediction performance. Per RL training step, we update the parameter θ of the model $\boldsymbol{\pi}_\theta$ to simultaneously minimise the online two-sample REINFORCE Leave-One-Out (RLOO) loss [43, 44] given in Equation 3, and the cross-entropy sample loss given in Equation 4.

$$\begin{aligned}\mathcal{L}_{\text{RLOO}}(\mathbb{D}_{\text{rl}};\theta) = -\sum_{i=1}^{N_{\text{rl}}}\Big[&\left(r_{\beta/2}^{\boldsymbol{\pi}_\theta}(\boldsymbol{x}_i, \hat{\boldsymbol{y}}_i^1) - r_{\beta/2}^{\boldsymbol{\pi}_\theta}(\boldsymbol{x}_i, \hat{\boldsymbol{y}}_i^2)\right)\log\left(\boldsymbol{\pi}_\theta(\hat{\boldsymbol{y}}_i^1|\boldsymbol{x}_i)\right) \\ &+\left(r_{\beta/2}^{\boldsymbol{\pi}_\theta}(\boldsymbol{x}_i, \hat{\boldsymbol{y}}_i^2) - r_{\beta/2}^{\boldsymbol{\pi}_\theta}(\boldsymbol{x}_i, \hat{\boldsymbol{y}}_i^1)\right)\log\left(\boldsymbol{\pi}_\theta(\hat{\boldsymbol{y}}_i^2|\boldsymbol{x}_i)\right)\Big]\end{aligned} \tag{3}$$

$$\mathcal{L}_{\text{DM}}(\mathbb{D}_{\text{rl}};\theta) = -\sum_{i=1}^{N_{\text{rl}}}\sum_{j=1}^{2}\sum_{t=1}^{T_{\hat{\boldsymbol{z}}_i^j}} \log \boldsymbol{\pi}_\theta\left(z_{i,t}^j|\boldsymbol{x}_i, \hat{\boldsymbol{y}}_i^j, \boldsymbol{z}_{i,<t}^j\right) \tag{4}$$

where θ_0 is the detached initial parameter in the RL stage, β is a constant hyperparameter, and $r_{\beta/2}^{\boldsymbol{\pi}_\theta}(\boldsymbol{y}_i)$ is the regularised reward defined as $r_j - \frac{\beta}{2}\log\frac{\boldsymbol{\pi}_\theta(\boldsymbol{y}_i^j|\boldsymbol{x}_i)}{\boldsymbol{\pi}_{\theta_0}(\boldsymbol{y}_i^j|\boldsymbol{x}_i)}$ for $j \in \{1, 2\}$.

3.3 Self-Verification Sampling by Internal Environment Model

After undergoing DyMo in both the SFT and RL phases, our model $\boldsymbol{\pi}_\theta$ is capable of both generating tool calls and predicting the subsequent states after executing them. Leveraging this capability, we propose to query the internal environment model of $\boldsymbol{\pi}_\theta$ multiple times to do Self-Verification Sampling (SVS), as illustrated in Algorithm 1. Given multiple completions per prompt, SVS selects a single output based on a specified scoring function *score* and the internal environment model of $\boldsymbol{\pi}_\theta$. Notably, unlike existing approaches, SVS scales test-time compute *without* querying the oracle environment $\mathcal{E}$. This approach is reminiscent of the Best-of-N search strategy described in [16], but avoids querying the environment $\mathcal{E}$ multiple times, thereby preventing unintended state changes caused by repeated trials. In addition, SVS aligns with the notion of mental simulation in decision-making, a concept explored in cognitive science [45], thereby establishing a conceptual bridge between research in RL and cognitive science.

Algorithm 1: Self-verification sampling (SVS)

```
Input: $x$, number of candidate completions $k$
Output: a completion $\hat{y}$
Given : a pre-specified scoring function score
for $i \leftarrow 1$ to $k$ do
    $\hat{y}_i \sim \pi_\theta(\cdot|x)$;
    $\hat{z}_i \sim \pi_\theta(\cdot|x, \hat{y}_i)$;
end
$j \leftarrow score(\pi_\theta(\hat{z}_1|x, \hat{y}_1), \ldots, \pi_\theta(\hat{z}_k|x, \hat{y}_k))$;
$\hat{y} \leftarrow \hat{y}_j$;
```

4 Experiments

4.1 Setup

Environment: We evaluate tool-use performance using the Berkeley Function Calling Leaderboard V2 (BFCL-V2) [19], which offers comprehensive coverage of function call types, diverse tasks, programming languages, and executability, and has been widely adopted in recent works [46, 6]. As our work is the first to investigate LLMs' ability to model environment dynamics, we begin with *single-turn* interactions to ensure a clean and tractable problem formulation, in a server-based BFCL-V2 environment in order to run online RL training. Regarding the base model, considering the constraints of our computes, we choose Cohere's R7B, given its leading performance on various agent benchmarks [6].

SFT Data: During the SFT stage, to constitute the function call (fc) SFT dataset $\mathbb{D}_{\text{fc}}$, inspired by [47, 48], we synthesised pairs of `<prompt,completion>` following the distribution of BFCL-V2. Regarding the state prediction (sp) SFT dataset $\mathbb{D}_{\text{sp}}$, we first split the state space $\mathbb{Z}$ into two subsets: 1) pass states $\mathbb{Z}^+$ where the completions successfully passed the check of BFCL-V2 and received a score of 1; 2) error states $\mathbb{Z}^-$ where the completions failed on the BFCL-V2's check and received a score of 0. Given the format of BFCL-V2's return messages, we denote the shared prefix of pass states in $\mathbb{Z}^+$ as z_{pass}, and similarly z_{error}. Note that, under this setup, there exhibits a bijection between the BFCL-V2's resulted state subspaces $\{\mathbb{Z}^+, \mathbb{Z}^-\}$ and the scores from the reward function $\{0, 1\}$, which we utilise later to truncate the generation when running SVS during inference time. Following this procedure, we constitute $\mathbb{D}_{\text{sp}}$ of `<prompt,completion,result>` triplets from our accumulated run-logs of BFCL-V2 tests.

RL Data: In the following RL stage, to maintain the online RL training and validation distributions as independent and identical, we use 80% from the original BFCL-V2 *prompt* set as the training set, and keep the remaining 20% to validate the generalisation performance. Note that we intentionally keep at least 20 test prompts per category in the final validation set, as certain categories contain ≤ 50 samples, thus 20% of them lacks of statistical significance.

SVS Scoring Function: During the inference time, following GenRM [18], we use a scoring function *score* in the following Equation 5 to run SVS illustrated in Algorithm 1:

$$\textit{score}\big(\boldsymbol{\pi}_\theta(\cdot|\boldsymbol{x}_1, \hat{\boldsymbol{y}}_1), \ldots, \boldsymbol{\pi}_\theta(\cdot|\boldsymbol{x}_k, \hat{\boldsymbol{y}}_k)\big) \triangleq \arg\max_j \left(\left\{\boldsymbol{\pi}_\theta(z_{\text{pass}}|\boldsymbol{x}_j, \hat{\boldsymbol{y}}_j)\right\}\right). \tag{5}$$

Examples of all the above types of data are provided in Appendix A.

4.2 How proficient is the model at dynamics modelling?

Since we partition the state space $\mathbb{Z}$ to $\mathbb{Z}^+$ and $\mathbb{Z}^-$, the state prediction task can be framed as a binary classification problem. A model $\boldsymbol{\pi}^{\text{sft}}$ SFTed on the combined data $\mathbb{D}_{\text{fc}} \cup \mathbb{D}_{\text{sp}}$, achieves a precision of 90.00%, recall of 87.71%, F1-score of 88.84%, and accuracy of 93.62%. Detailed results are provided in Table 3 in Appendix B. Notably, the success rate of this model on BFCL-V2 is only 72.77%, which is significantly lower than its discriminative performance, highlighting the gap between accurate state prediction and successful functions calls. Therefore, a foundation is laid for improving a model's generative capability by leveraging its discriminative capability [17, 35].

We also track these metrics during online RL training with the DM loss function for $\boldsymbol{\pi}^{\text{sft}}$. Under this setting, precision, recall, F1-score, and accuracy further improve to 92.55%, 96.28%, 94.34%, and 90.36%, respectively. The corresponding curves are presented in Figure 6 in Appendix B. For comparison, we SFT an additional model, ϕ^{sft}, on the function call dataset $\mathbb{D}_{\text{fc}}$ only. As a result, it learns to roll out states solely through the DM loss during online RL training. We observe a consistent performance gap between this baseline model ϕ^{sft} and $\boldsymbol{\pi}^{\text{sft}}$, highlighting the necessity of incorporating $\mathbb{D}_{\text{sp}}$ in the SFT stage.

4.3 How does dynamics modelling benefit SFT and RL for tool-use?

During the experiments in Section 4.2, we observe that incorporating the additional state prediction data $\mathbb{D}_{\text{sp}}$ also leads to a difference in tool use performance. We compare the the performance of ϕ^{sft} - which can do only tool use - with $\boldsymbol{\pi}^{\text{sft}}$ - which is capable of both using tools and predicting

Model	Method	**Overall** *(UW)*	**Overall** *(W)*	**Rel.**	**Irrel.**	**AST**	**Exec**
Baselines			*# samples*	*(18)*	*(1122)*	*(2501)*	
GPT-4o [1]	–	82.38	82.14	**83.33**	81.31	82.51	–
Command-A [6]	–	80.57	84.14	72.22	86.19	83.30	–
Command-R7B	–	70.50	76.70	55.56	81.02	74.92	–
xLAM-2 [48]	–	72.36	71.69	77.78	64.34	74.95	–
ToolACE-2 [23]	–	81.95	**85.49**	72.22	**90.11**	**83.51**	–
watt-tool [49]	–	**82.54**	81.76	**83.33**	83.15	81.13	–
BigAgent [50]	–	82.27	81.50	**83.33**	82.38	81.10	–
SFT			*# samples*	*(18)*	*(1122)*	*(2501)*	
ϕ^{sft}	$\mathbb{D}_{\text{fc}}$ only	66.35	66.50	70.73	58.05	70.26	76.25
π^{sft}	$\mathbb{D}_{\text{fc}} \cup \mathbb{D}_{\text{sp}}$	70.87	73.89	63.41	76.32	72.88	77.53
SFT + RL			*# samples*	*(20)*	*(206)*	*(457)*	
ϕ^{rl}	$\phi^{\text{sft}}\rightarrow$RLOO	80.31	80.22	75.00	89.81	76.13	96.25
ϕ^{rd}	$\phi^{\text{sft}}\rightarrow$RLOO + DyMo	82.13	83.16	75.00	**91.75**	79.65	**97.50**
π^{rl}	$\pi^{\text{sft}}\rightarrow$RLOO	81.23	81.99	75.00	90.00	78.68	96.25
π^{rd}	$\pi^{\text{sft}}\rightarrow$RLOO + DyMo	**83.62**	**86.68**	75.00	90.29	**85.56**	96.25
SFT + RL + SVS (with k candidates)			*# samples*	*(20)*	*(206)*	*(457)*	
π^{rd}	$\pi^{\text{rd}}\rightarrow$ SVS with $k=1$	85.77	84.26	88.20	85.65	83.46	96.33
π^{rd}	$\pi^{\text{rd}}\rightarrow$ SVS with $k=2$	88.20	86.71	91.30	86.86	86.44	**96.55**
π^{rd}	$\pi^{\text{rd}}\rightarrow$ SVS with $k=4$	88.94	87.67	91.80	87.41	87.61	96.45
π^{rd}	$\pi^{\text{rd}}\rightarrow$ SVS with $k=8$	89.73	88.18	93.10	88.10	88.00	96.25
π^{rd}	$\pi^{\text{rd}}\rightarrow$ SVS with $k=16$	89.90	88.29	93.30	88.38	**88.03**	96.15
π^{rd}	$\pi^{\text{rd}}\rightarrow$ SVS with $k=32$	90.18	88.26	94.10	88.59	87.86	96.25
π^{rd}	$\pi^{\text{rd}}\rightarrow$ SVS with $k=64$	**90.69**	**88.43**	**95.00**	**89.32**	87.75	96.25

Table 1: Comprehensive category-wise performance comparison across baselines, SFT, SFT+RL, and SFT+RL+SVS models, on BFCL-V2. For each section, the number of evaluation examples per column is shown in the second row. *(W)* indicates metrics weighted by the number of samples, whereas *UW* indicates unweighted. Missing results are marked as '–'. The "Exec" column is provided to show the improvement from RL training on it, but is never counted for the overall performance.

next states - across all categories of BFCL-V2. The results in the "SFT" section of Table 1 show that π^{sft} achieves significant improvements on the "Irrelevance" category where the models are not expected to generate function calls when the available tools cannot satisfy the user request. Since the "Irrelevance" is specifically designed to evaluate hallucination of models [19], these results suggest that incorporating the state prediction task helps mitigate hallucination by LLMs [51].

Similarly, we compare the two models — ϕ^{sft} and π^{sft} — both fine-tuned by online RL with and without our DyMo loss, resulting in four variants: π^{rd}, ϕ^{rd} (with DyMo), and π^{rl}, ϕ^{rl} (without DyMo). Take π^{rd} for example, the model is first SFTed on $\mathbb{D}_{\text{fc}} \cup \mathbb{D}_{\text{sp}}$ (thus notated as π), then further fine-tuned by online RL together with DyMo loss (thus superscripted by "rd"). The "SFT + RL" section of Table 1 shows the success rates of these models across different BFCL-V2 categories. For analytical clarity, we preserve the "Exec" category to show the substantial performance gains over it due to RL training. The results indicate that incorporating the DyMo loss yields a $> 5\%$ improvement in success rate over the AST category, contributing to an overall performance boost.

4.4 How does the RL/SFT models perform when scaling up test-time compute?

Since the results in the "SFT" and "SFT + RL" sections of Table 1 are based on a greedy decoding strategy, we further examine whether and how the on-policy distribution over completions for a given prompt, i.e., $\pi_\theta(\cdot|\boldsymbol{x})$, changes under different training pipelines. We begin by analysing the impact of online RL training, comparing the RL-trained models — π^{rl} and ϕ^{rl} — with their corresponding SFT-only baselines — π^{sft} and ϕ^{sft} — using the number of completions per request as the variable. In Figure 3, we report pass@k and pass^k [26] as the evaluation metrics.

As shown in Figure 3, online RL significantly improves pass@k when $k \leq 8$. More importantly, online RL consistently improves pass^k over all k values, as evidenced by the consistent gap between the RL-trained models — π^{rl} and ϕ^{rl} — over their SFT-only counterparts — π^{sft} and ϕ^{sft}. These results suggest that the on-policy distributions induced by the RL models yields a more consistent and reliable function calling performance than the distributions induced by the SFT models.

We also note that our pass@k curves align with the findings in mathematical reasoning tasks reported by Yue et al. [52], who conclude that conclude that "base models can achieve a comparable or even higher pass@k score compared to their RL counterparts at large k values". However, in our setup, we found that base model can hardly match pass@k or pass^k of SFT and RL models, which we argue is due to that correct function calls are sparser to generate.

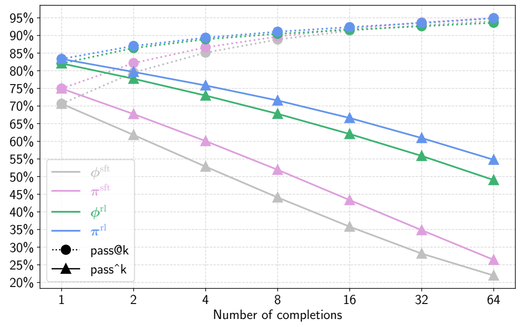


Figure 3: Pass@k (-) and pass^k (..) of RL models - π^{rl} and ϕ^{rl} - vs SFT bases - π^{sft} and ϕ^{sft} on BFCL-V2.

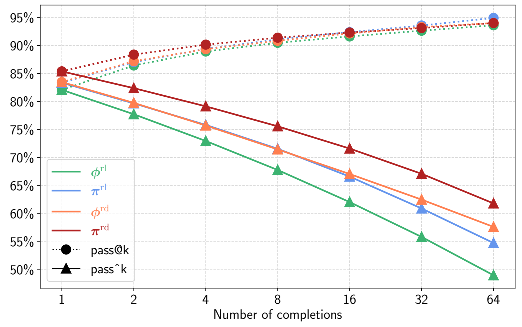


Figure 4: Pass@k (-) and pass^k (..) of models trained by online RL with DyMo loss - π^{rd} and ϕ^{rd} - vs without DyMo loss - π^{rl} and ϕ^{rl} - on BFCL-V2.

4.5 How does dynamics modelling impact the test-time compute scaling of RL models?

Building on the observation that RL models achieve higher success rates over test-time compute scales, we further investigate the impact of incorporating the DyMo loss during online RL training. Similarly, we report pass@k and pass^k for RL models trained with the DyMo loss — π^{rd} and ϕ^{rd} — compared to those trained without it — π^{rl} and ϕ^{rl}.

As shown in Figure 4, adding the DyMo loss during online RL improves pass@k when $k \leq 8$, while it consistently improves pass^k over all numbers of completions per prompt. Note that SVS is not utilised in the experiments so far, thus the improvements are solely due to the DyMo loss. More notably, incorporating the DyMo in both the SFT and RL stages results in π^{rd}, which achieves the highest pass^k for all values of k. The consistent gap between pass^k curves of $\pi^{\mathrm{rd}}/\phi^{\mathrm{rd}}$ and $\pi^{\mathrm{rl}}/\phi^{\mathrm{rl}}$ also indicate the DyMo loss can help to further improve the consistency and reliability of function calling performance on top of RL. These results demonstrate the effectiveness and benefits of integrating DyMo into both the SFT and RL phases.

4.6 How does self-verification sampling scale over test-time compute?

So far, we have focused primarily on the benefits of incorporating DyMo during model training. However, as introduced in Section 3.3, during inference time, self-verification sampling (SVS) actually unifies the policy (as in model-free RL), the environment model (as in model-based RL), and the value function (under our specific state-space split) into a single LLM. This paradigm enables the model to scale test-time compute by generating more candidate completions per user request **without** querying the oracle environment function $\mathcal{E}$. To evaluate the effectiveness of SVS, we compare *pass^k with SVS* against *pass^k without SVS* of model π^{rd}. For pass^k with SVS, we sample c candidates for each trial and k trials per prompt, thus $k \times c$ candidate completions in total for each prompt. Further, per candidate group for each trial, following GenRM [16], we adopt the scoring function defined in Equation 5 as the metric to select just one output from the c candidates (thus k outputs in the end).

	k	1	2	4	8	16	32
pass^k	with SVS (c for each trial)	89.02% (64)	87.97% (32)	87.19% (16)	86.14% (8)	84.05% (4)	78.05% (2)
	without SVS	87.68%	82.38%	79.14%	75.58%	71.61%	67.11%

Table 2: Pass^k with and without SVS over k trials in the oracle environment. Augmented with SVS, per prompt, we first generate c candidate completions for each trial, then select just one to output by the scoring function defined in Equation 5 for all k trials. Therefore, there are $k \times c$ candidates in total for each prompt, by querying the oracle environment also k times as to pass^k without SVS.

As shown in Table 2, SVS achieves improved pass^k over all k values, demonstrating that self-verification enables effective scaling with additional computes. More importantly, the consistent improvement of SVS performance with increasing k highlights our method as a novel test-time compute scaling strategy — one that leverages the model's internal environment approximation to self-verify and select the most reliable candidate completion. In Section 4.7 and Section 5, we provide further insights about our current SVS setup.

Beyond the above experiment, we also compare pass@k of the "Best-of-N" test-time compute scaling strategy with the *pass@1* performance of $\boldsymbol{\pi}^{\text{rd}}$ using SVS with k candidate completions per prompt, thus both methods operate under the same inference compute budget. As shown in the results provided in the "SFT + RL + SVS" section of Table 1, increasing number of candidates k in SVS consistently improves the *pass@1*, which demonstrated the effectiveness of the model's internal environment model. It is unsurprising to observe that querying the model's internal environment model is less efficient than accessing the oracle environment function under the same compute budget, and should be seen as an upper-bond. However, we also argue that relying on the oracle may be impractical in many real-world applications involving stateful environments. For example, the model is not expected to place k parallel orders for a single shopping request or to book k tickets on the same flights for a travel planning request.

4.7 What if the model is allowed to refuse?

"I'm sorry, Dave. I'm afraid I can't do that." – 2001: A Space Odyssey

As may already be observed, a notable limitation of the scoring function defined in Equation 5 is that the model is still required to output a completion, even in cases where all candidate completions are self-verified as failed trials. That is, our model might roll-out z_{error} for all generated candidates. In such cases, we argue that it is both reasonable and desirable for the model to "refuse" the request by returning a message that informs the user the query cannot be completed reliably. Formally, we define the revised scoring function as follow:

$$\begin{aligned} &\mathit{score}\big(\boldsymbol{\pi}^{\text{rd}}(\hat{\boldsymbol{z}}_1|\boldsymbol{x}_1,\hat{\boldsymbol{y}}_1),\ldots,\boldsymbol{\pi}^{\text{rd}}(\hat{\boldsymbol{z}}_k|\boldsymbol{x}_k,\hat{\boldsymbol{y}}_k)\big) \triangleq \arg\max_j \left(\left\{\boldsymbol{\pi}^{\text{rd}}(\hat{\boldsymbol{z}}_j|\hat{\boldsymbol{y}}_j,\boldsymbol{x}_j)\right\}\right) \\ &\text{s.t. } z_{\text{pass}} \prec \hat{\boldsymbol{z}}_j \text{ and } \boldsymbol{\pi}^{\text{rd}}(z_{\text{pass}}|\hat{\boldsymbol{y}}_j,\boldsymbol{x}_j) > \tau \end{aligned} \tag{6}$$

where $z_{\text{pass}} \prec \hat{\boldsymbol{z}}_j$ means that $\hat{\boldsymbol{z}}_j$ starts with z_{pass} as the prefix, and τ is the threshold hyperparameter.

Using Equation 6 as the scoring function, the model $\boldsymbol{\pi}^{\text{rd}}$ classifies a completion $\hat{\boldsymbol{y}}$ as positive if and only if $\boldsymbol{\pi}^{\text{rd}}(z_{\text{pass}}|\hat{\boldsymbol{y}},\boldsymbol{x}) > \tau$; otherwise, it is classified as negative. By sweeping across a range of thresholds $\tau \in [0.5, 0.99]$, we find that $\tau = 0.92$ offers a favourable trade-off between precision and refusal. Fixing $\tau = 0.92$, we then examine how precision and refuse rate vary with the number of candidate completions k, as shown in Figure 5.

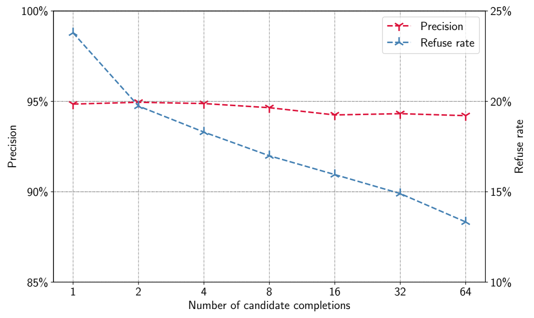


Figure 5: Precision and refuse rate over k for SVS

Surprisingly, the model maintains a precision of $\sim 94.5\%$ across values of k, while the refuse rate decreases notably from $23.79\%(k = 1)$ to $13.33\%(k = 64)$. Since precision reflects the proportion of correct completions among all non-refused outputs, we interpret it as a proxy for the reliability of the model's responses. Under this view, our results suggest that reliability remains stable as the number of candidates increases, while the refusal rate drops significantly — indicating improved solution coverage without sacrificing correctness. These findings highlight the practical value of combining DyMo with SVS: by generating more candidates, the model achieves higher success rates while maintaining high reliability. We further discuss the broader implications of this observation in Section 5.

5 Discussion

Internal environment model by DyMo: Recent advances in RL have shown that incorporating world models can substantially improve performance in complex domains such as board games [53] and video games [54]. Building on this line of work, our approach takes a further step by unifying the world model and the policy into a single LLM through DyMo, and demonstrates the practical benefits of this unification in tool use scenarios. We also note that similar motivations have emerged in reward modelling, where LLMs are fine-tuned either into stand-alone reward models [18], or into generative models capable of self-rewarding by reasoning over multiple steps in a single completion [17]. Our work extends this broader trend by showing how DyMo can enhance function calling beyond reasoning, particularly in stateful environments.

Low true negative ratio problem of SVS: In our analysis of the results in Section 4.6, we observe that the model π^{rd} exhibits surprisingly low true negative ratio ($< 50\%$ TNR), despite achieving strong precision and recall. As the model's success rate increases through online RL training, the proportion of correct completions steadily rises. This leads to a highly imbalanced distribution between completions beginning with z_{pass} and z_{error}, thus introduces a bias toward predicting states in $\mathbb{Z}^+$. Consequently, we observe that the model tends to "over-refuse" its own completions, i.e. it incorrectly verifies many correct completions as failures via its internal environment model. A straightforward mitigation strategy would be to incorporate additional negative samples from $\mathbb{D}_{\text{fc}}$, thereby exposing the model to a more balanced distribution during DyMo in RL training. Due to time and research scope constraints, we leave this direction for future work. Nonetheless, we argue that our method significantly enhances the reliability of model outputs: higher precision implies that completions self-verified by the model are more likely to be correct. This property is particularly valuable in high-stakes or safety-critical domains, such as healthcare or finance, where even a few incorrect outputs can lead to undesirable or irreversible outcomes.

Test-time compute scaling via DyMo and SVS: As discussed in previous sections, the proposed DyMo and SVS provide a novel strategy for test-time compute scaling. Here, we offer additional reflections from both data-centric and modelling perspectives. First, we highlight that DyMo can benefit from failed completions, since a complete environment model should be capable of handling both successful and failed trajectories. Given the vast amount of run logs accumulated from software systems over decades, we argue that DyMo unlocks a largely under-explored data source: rich, naturally occurring software run logs. In particular, the ability of DyMo to learn from failed completions helps improve the fidelity of the internal environment model — a capability, to the best of our knowledge, not explicitly addressed in prior works from the LLM community. Secondly, from the perspective of world modelling, we hypothesise that programs are often written with an implicit and internal world model of the developers who coded upon assumptions about environment dynamics, constraints, and expected behaviours. These implicit world models are then reflected in the run logs, which can then be captured and fitted by the proposed DyMo method. Through SVS, this learned environment model can be exploited at test time, enabling the model to improve its decision quality without external feedback. While this hypothesis is promising, a deeper exploration of the relationship between program execution and world modelling lies beyond the scope of this work, and we leave it for future investigation.

6 Conclusion

In this work, we investigate the challenge of tool use in stateful environments, where existing test-time compute strategies become impractical due to repeated environment queries. To address this, we propose DyMo, a method that augments LLM fine-tuning with an additional state prediction task during both the SFT and RL stages, enabling a next-state prediction capability of the model. Experiments on the BFCL-V2 benchmark show that incorporating DyMo significantly reduces hallucinations during SFT and improves the success rate over RL training loops. Notably, we also observe that RL models consistently outperform SFT models in mitigating hallucinations. Furthermore, we demonstrate that correct tool calls are retrievable for over 93% of prompts using a parallel Best-of-N decoding strategy, indicating that both SFT and RL models have learned sufficiently expressive on-policy distributions. Building on this insight, we introduce a self-verification sampling (SVS) strategy, which consistently improves pass^k and pass@1 performance by leveraging the model's internal environment model. Crucially, by allowing the model to refuse uncertain completions, our approach produces more reliable outputs in scenarios where correctness is essential. Overall, our findings highlight a promising direction for extending planning algorithms from the RL community to LLMs in dynamic and stateful environments.

Acknowledgments

We would like to express our sincere gratitude to Kenny Smith and Yao Zhao for their insightful discussions on refining the initial design of our method. We are also gratefully thankful to Minjie Xu, Yixuan Su, and Patrick Lewis for their valuable assistance with the data used in our experiments. We are particularly indebted to Matthieu Geist for his constructive feedback and thoughtful suggestions, which substantially contributed to improving the quality of this work.

We further acknowledge the significant efforts of the Reinforcement Learning team at Cohere, especially Yannis Flet-Berliac, Nathan Grinsztajn, Pierre Clavier, Eugene Choi, Yash Chandak, and Mohammad Gheshlaghi Azar,for their dedication in developing the infrastructure necessary for this project. In particular, we extend our deep appreciation to Eugene Tarassov and Elena Tommasone for their extensive support in implementing the systems required to run our algorithms.

Finally, R. Avalos acknowledges support from the Descartes project and the Flemish AI program.

References

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

[2] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[3] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

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

[5] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[6] Team Cohere et al. Command a: An enterprise-ready large language model. *arXiv preprint arXiv:2504.00698*, 2025.

[7] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=VTF8yNQM66`.

[8] Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Quentin Carbonneaux, Taco Cohen, and Gabriel Synnaeve. Rlef: Grounding code llms in execution feedback with reinforcement learning. *arXiv preprint arXiv:2410.02089*, 2024.

[9] Anthropic. Claude 3.7 sonnet and claude code. `https://www.anthropic.com/news/claude-3-7-sonnet`, 2025. As of April 1st 2025.

[10] Cohere. Basic usage of tool use (function calling). `https://docs.cohere.com/v2/docs/tool-use-overview#step-3-get-tool-results`, 2024. As of April 1st 2025.

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

[12] OpenAI. Function calling and other api updates. `https://openai.com/index/function-calling-and-other-api-updates/`, 2023. As of April 1st 2025.

[13] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

[14] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

[15] Pablo Villalobos and David Atkinson. Trading off compute in training and inference, 2023. *URL https://epochai. org/blog/trading-off-compute-in-training-and-inference. Accessed*, pages 07–03, 2024.

[16] Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=4FWAwZtd2n`.

[17] Wei Xiong, Hanning Zhang, Chenlu Ye, Lichang Chen, Nan Jiang, and Tong Zhang. Self-rewarding correction for mathematical reasoning. *arXiv preprint arXiv:2502.19613*, 2025.

[18] Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. *arXiv preprint arXiv:2408.15240*, 2024.

[19] Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Berkeley function calling leaderboard. `https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html`, 2024.

[20] Eric Zhao, Pranjal Awasthi, and Sreenivas Gollapudi. Sample, scrutinize and scale: Effective inference-time search by scaling verification. *arXiv preprint arXiv:2502.01839*, 2025.

[21] Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-bank: A comprehensive benchmark for tool-augmented LLMs. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.187. URL `https://aclanthology.org/2023.emnlp-main.187/`.

[22] 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. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=dHng2O0Jjr`.

[23] Weiwen Liu, Xu Huang, Xingshan Zeng, xinlong hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong WANG, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Wang Xinzhi, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. ToolACE: Winning the points of LLM function calling. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=8EB8k6DdCU`.

[24] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=zAdUB0aCTQ`.

[25] Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities, 2024. URL `https://arxiv.org/abs/2408.04682`.

[26] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. τ-bench: A benchmark for Tool-Agent-User interaction in real-world domains. In *Proceedings of the Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=roNSXZpUDN`. ICLR 2025 (poster).

[27] Yuanqing Yu, Zhefan Wang, Weizhi Ma, Zhicheng Guo, Jingtao Zhan, Shuai Wang, Chuhan Wu, Zhiqiang Guo, and Min Zhang. Steptool: A step-grained reinforcement learning framework for tool learning in llms. *arXiv preprint arXiv:2410.07745*, 2024.

[28] Kevin Chen, Marco Cusumano-Towner, Brody Huval, Aleksei Petrenko, Jackson Hamburger, Vladlen Koltun, and Philipp Krähenbühl. Reinforcement learning for long-horizon interactive llm agents. *arXiv preprint arXiv:2502.01600*, 2025.

[29] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 3008–3021. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper_files/paper/2020/file/1f89885d556929e98d3ef9b86448f951-Paper.pdf`.

[30] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Gray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=TG8KACxEON`.

[31] Hanze Dong, Wei Xiong, Bo Pang, Haoxiang Wang, Han Zhao, Yingbo Zhou, Nan Jiang, Doyen Sahoo, Caiming Xiong, and Tong Zhang. Rlhf workflow: From reward modeling to online rlhf. *arXiv preprint arXiv:2405.07863*, 2024.

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

[33] Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Berkeley function calling leaderboard. `https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html`, 2024.

[34] Wei Xiong, Hanze Dong, Chenlu Ye, Ziqi Wang, Han Zhong, Heng Ji, Nan Jiang, and Tong Zhang. Iterative preference learning from human feedback: Bridging theory and practice for rlhf under kl-constraint. *arXiv preprint arXiv:2312.11456*, 2023.

[35] Shangmin Guo, Biao Zhang, Tianlin Liu, Tianqi Liu, Misha Khalman, Felipe Llinares, Alexandre Rame, Thomas Mesnard, Yao Zhao, Bilal Piot, et al. Direct language model alignment from online ai feedback. *arXiv preprint arXiv:2402.04792*, 2024.

[36] Yunhao Tang, Daniel Zhaohan Guo, Zeyu Zheng, Daniele Calandriello, Yuan Cao, Eugene Tarassov, Rémi Munos, Bernardo Ávila Pires, Michal Valko, Yong Cheng, et al. Understanding the performance gap between online and offline alignment algorithms. *arXiv preprint arXiv:2405.08448*, 2024.

[37] Michael Noukhovitch, Shengyi Huang, Sophie Xhonneux, Arian Hosseini, Rishabh Agarwal, and Aaron Courville. Faster, more efficient RLHF through off-policy asynchronous learning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=FhTAG591Ve`.

[38] Brian R Bartoldson, Siddarth Venkatraman, James Diffenderfer, Moksh Jain, Tal Ben-Nun, Seanie Lee, Minsu Kim, Johan Obando-Ceron, Yoshua Bengio, and Bhavya Kailkhura. Trajectory balance with asynchrony: Decoupling exploration and learning for fast, scalable llm post-training. *arXiv preprint arXiv:2503.18929*, 2025.

[39] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=_VjQlMeSB_J`.

[40] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[42] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press Cambridge, 2016.

[43] Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*, 2024.

[44] Yannis Flet-Berliac, Nathan Grinsztajn, Florian Strub, Eugene Choi, Bill Wu, Chris Cremer, Arash Ahmadian, Yash Chandak, Mohammad Gheshlaghi Azar, Olivier Pietquin, and Matthieu Geist. Contrastive policy gradient: Aligning LLMs on sequence-level scores in a supervised-friendly fashion. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 21353–21370, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1190. URL `https://aclanthology.org/2024.emnlp-main.1190/`.

[45] Gary Klein and Beth W Crandall. The role of mental simulation in problem solving and decision making. In *Local applications of the ecological approach to human-machine systems*, pages 324–358. CRC Press, 2018.

[46] Qwen Team. Qwq-32b: Embracing the power of reinforcement learning. `https://qwenlm.github.io/blog/qwq-32b/`, 2024.

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

[48] Akshara Prabhakar, Zuxin Liu, Weiran Yao, Jianguo Zhang, Ming Zhu, Shiyu Wang, Zhiwei Liu, Tulika Awalgaonkar, Haolin Chen, Thai Hoang, et al. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *arXiv preprint arXiv:2504.03601*, 2025.

[49] Wentao Shi, Mengqi Yuan, Junkang Wu, Qifan Wang, and Fuli Feng. Direct multi-turn preference optimization for language agents, 2024. URL `https://arxiv.org/abs/2406.14868`.

[50] BitAgent. Bitagent-8b. `https://huggingface.co/BitAgent/BitAgent-8B`, 2025. Accessed: 2025-05-16.

[51] Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. On faithfulness and factuality in abstractive summarization. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1906–1919, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.173. URL `https://aclanthology.org/2020.acl-main.173/`.

[52] Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*, 2025.

[53] John Schultz, Jakub Adamek, Matej Jusup, Marc Lanctot, Michael Kaisers, Sarah Perrin, Daniel Hennes, Jeremy Shar, Cannada Lewis, Anian Ruoss, et al. Mastering board games by external and internal planning with language models. *arXiv preprint arXiv:2412.12119*, 2024.

[54] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 2025. doi: 10.1038/s41586-025-08744-2. URL `https://doi.org/10.1038/s41586-025-08744-2`.

A Examples of Data Used for Training LLMs

In this section, we present examples of the datasets curated for supervised fine-tuning (SFT) of large language models (LLMs) on the tool use and state prediction tasks, as described in Section 3.2.1 and Section 4.1.

A.1 Example of Function Call Supervised Fine-tuning Dataset $\mathbb{D}_{\text{fc}}$

Below, we provide an example of the function call SFT data. The completion shown in the yellow box corresponds to the ground-truth output used for supervised training and is guaranteed to be correct. Importantly, our function call SFT dataset $\mathbb{D}_{\text{fc}}$ does not include any data from the original BFCL benchmark; the following example is provided solely for illustrative purposes.

Example 1: Humidity Forecast Query

System Preamble

You are a large language model AI assistant. Your knowledge cutoff date is ...

...

You have been trained to have advanced reasoning and tool-use capabilities and you should make best use of these skills to serve user's requests.

Here is the list of tools that you have available to you. You can ONLY use the tools listed here. When a tool is not listed below, it is NOT available and you should NEVER attempt to use it. Each tool is represented as a JSON object with fields like "name", "description", "parameters" (per JSON Schema), and optionally, "responses" (per JSON Schema).

```
[
{"name": "weather.humidity_forecast", "description": "Retrieve
a humidity forecast for a specific location and time frame.",
"parameters": {"type": "object", "properties": {"location":
{"type": "string", "description": "The city that you want to get the
humidity for."}, "days": {"type": "integer", "description": "Number
of days for the forecast."}, "min_humidity": {"type": "integer",
"description": "Minimum level of humidity (in percentage) to filter
the result. Optional parameter. Default is 0."}}, "required":
["location", "days"]}, "responses": null},
{"name": "get_team_score", "description": "Retrieves the latest game
score, individual player stats, and team stats for a specified sports
team.", "parameters": {"type": "object", "properties": {"team_name":
{"type": "string", "description": "The name of the sports team."},
"league": {"type": "string", "description": "The league that the
team is part of."}, "include_player_stats": {"type": "boolean",
"default": false, "description": "Indicates if individual player
statistics should be included in the result. Default is false."}},
"required": ["team_name", "league"]}, "responses": null}
]
...
```

User Prompt

What is the humidity level in Miami, Florida in the upcoming 7 days?

Completion

I'll query the `weather.humidity_forecast` to give user an answer. The call is

```
{"weather.humidity_forecast": {"location": ["Miami", "Miami, Florida", "FL"], "days": [7], "min_humidity": ["", 0]}}
```

A.2 Examples of State Prediction Supervised Fine-tuning Dataset $\mathbb{D}_{\text{sp}}$

Below, we present an example of the state prediction SFT data. Example 2 corresponds to a previously successful call to the humidity forecast function.

Example 2: A Successful Call to the Humidity Forecast Query

System Preamble

You are a large language model AI assistant. Your knowledge cutoff date is ...

(Identitcal to exmpale 1 above, thus we skip the following content to save pages.)

User Prompt

What is the humidity level in Miami, Florida in the upcoming 7 days?

Completion

I'll query the `weather.humidity_forecast` to give user an answer. The call is

```
{"weather.humidity_forecast": {"location": ["Miami", "Miami, Florida", "FL"], "days": [7], "min_humidity": ["", 0]}}
```

Pass

<|pass|>

```
{ "status": 1, "forecast": [ { "date": "2025-04-11", "min_humidity": 62, "max_humidity": 78 }, ... ] }
```

Example 3 corresponds to a previously failed call to the humidity forecast function and is used as a negative example in the state prediction task.

Example 3: A Failed Call to the Humidity Forecast Query

System Preamble

You are a large language model AI assistant. Your knowledge cutoff date is ...

(Identitcal to exmpale 1 above, thus we skip the following content to save pages.)

User Prompt

What is the humidity level in Miami, Florida in the upcoming 7 days?

Completion

I'll query the `weather.humidity_forecast` to give user an answer. The call is

```
{"weather.humidity_forecast": {"location": ["Miami", "Miami,
Florida", "FL"], "days": [-7], "min_humidity": ["", 0]}}
```

Error

<|error|>

```
{ "status": 0, "error": "Invalid input to argument 'days', expected
a positive integer, but got -7...." }
```

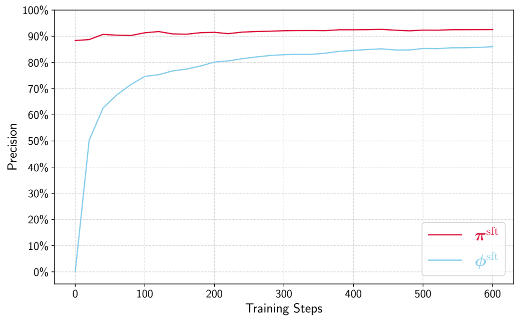
B Results of Binary Classification

In Table 3, we present a detailed breakdown of the model's performance on the binary classification task formulated in Section 4.2. The table includes the confusion matrix, with values normalized to sum to 100 for interpretability, as well as confidence intervals (in brackets) for precision, recall, F1-score, and accuracy. As previously discussed, there exists a statistically significant gap between the model's discriminative performance on the state prediction task and its actual success rate in generating correct completions. These findings suggest a promising direction for leveraging the model's discriminative strength to improve its generative behaviour, which has been explored by Guo et al. [35].

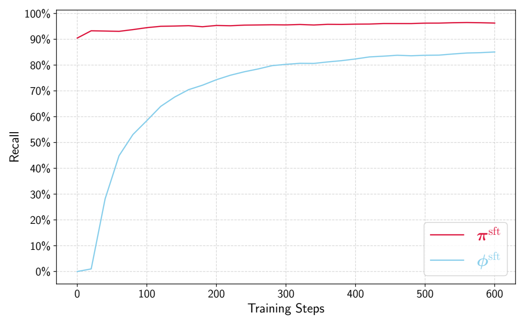
Actual	**Predicted**		**Metrics**
	Positive	**Negative**	
Positive	25.40	3.56	Precision: $90.00\%(86.02\% - 93.78\%)$
Negative	2.82	68.22	Recall: $87.71\%(83.46\% - 91.47\%)$
Accuracy	$93.62\%(91.90\% - 95.21\%)$		F1-score: $88.84\%(84.72\% - 92.61\%)$

Table 3: Confusion matrix of predicting next states by the model π^{sft} SFTed on both function call and state prediction data.

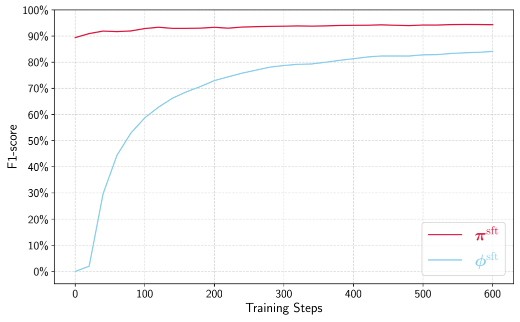
As discussed in Section 4.2, we track precision, recall, F1-score, and accuracy throughout the course of online RL training. The corresponding curves are shown in Figure 6. The red curves represent the model π^{sft}, which is initialized from SFT on $\mathbb{D}_{\text{fc}} \cup \mathbb{D}_{\text{sp}}$, while the blue curves correspond to the baseline model ϕ^{sft}, fine-tuned only on $\mathbb{D}_{\text{fc}}$. As shown in the figure, ϕ^{sft}, which lacks initial state prediction capability, consistently underperforms π^{sft} across all metrics throughout training. Even after 600 steps of RL training, ϕ^{rd} fails to match the performance of the SFT-only model π^{sft}, which indicates the necessary of the state prediction data. These results suggest that the benefits of $\mathbb{D}_{\text{sp}}$ cannot be compensated for by relying solely on the downstream DyMo loss during RL training.



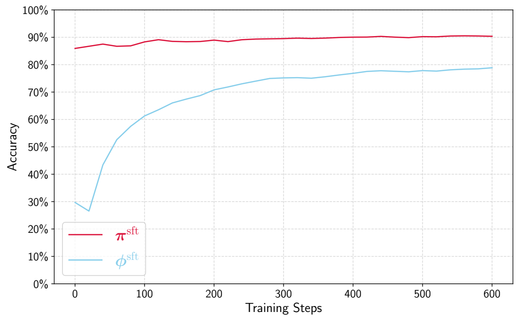
(a) Precision



(b) Recall



(c) F1-score



(d) Accuracy

Figure 6: Performance metrics of results prediction over online RL training with DM loss (Equation 4).