AppSim: A Learned World Model for an App API

Garvit Luhadia[*]
New York University
gl2758@nyu.edu

Georgy Savva[*]
New York University
georgy.savva@nyu.edu

Abstract

We present **AppSim**, a family of learned world models for API simulation. Building on the model-based reinforcement learning successes in vision-based games and text-driven environments, we consider two approaches. In the first, we finetune a TinyLlama model on a custom dataset of synthetic trajectories from the AppWorld engine, combining semi-random API exploration with ChatGPT-driven user requests. In the second, we use a powerful off-the-shelf LLM in a zero-shot setting. Evaluating on held-out trajectories using object-match accuracy, BLEU, and ROUGE, we find that the general-purpose LLMs exhibit superior performance.

1 Introduction

Predictive world models are differentiable approximations of environment dynamics and central to model-based reinforcement learning which enable agents to optimize policies via gradient-based planning in latent state spaces (Hafner et al., 2020). Seminal work demonstrated that an agent can learn a "dream" simulator and train entirely within it, with zero-shot transfer to the real environment (Ha and Schmidhuber, 2018). Subsequent advances such as DreamerV3 (Hafner et al., 2024) and Operator World Models (Novelli et al., 2024) have broadened these ideas to diverse domains and joint estimation of rewards and transitions. Prior world models focus on games or web navigation, while only a select few tackle text-based API interactions.

Recent efforts have begun to explore language models as world models in text-rich settings: textual game simulators (Ammanabrolu and Riedl, 2021; Wang et al., 2024), structured task planners (Xie et al., 2024), and web agents (Gu et al., 2024). Multimodal approaches like WorldGPT (Ge et al., 2024) further extend this paradigm by leveraging MLLMs to understand world dynamics through analyzing millions of videos across diverse domains. However, API-driven applications that are ubiquitous in automated assistants, web agents, and tool-augmented LMs—remain unmodeled.

To address this, we propose **AppSim**, the first family of learned world models for a Todoist-style task-management API. We analyze the performance of powerful, costly LLMs in the zero-shot setting compared to a small open-source LM finetuned on a custom dataset. To build the latter, we collect thousands of interaction trajectories via the AppWorld simulator and finetune a decoder-only Transformer (TinyLlama) to autoregressively predict each API response given the full history of calls and replies. Both of our approaches are fully differentiable.

2 Related Work

2.1 Latent World Models in RL

Model-based RL agents learn latent transition models to plan in imagined roll-outs. Dreamer (Hafner et al., 2020) achieve high sample efficiency in visual domains, and recent diffusion-based engines extend to real-time game simulation (Valevski et al., 2024). Operator-theoretic approaches further integrate reward and transition learning (Novelli et al., 2024).

2.2 Text-Based World Modeling

Language-driven simulators have been studied in interactive fiction and text-based QA. Knowledge-graph world models predict state changes in text adventures (Ammanabrolu and Riedl, 2021), while structured effect–precondition models inject symbolic knowledge into LLMs (Xie et al., 2024). Benchmarks like Text2World (Hu et al., 2025) evaluate LLMs on PDDL-style planning tasks. MirrorAPI (Guo et al., 2024) introduces a corpus for LLM-based API response simulation. WorldGPT (Ge et al., 2024) extends multimodal LLMs as generalist world models capable of un-

[*]Equal contribution.

derstanding state transitions across diverse real-life scenarios.

2.3 App and API Simulation

Toolformer (Schick et al., 2023) and related tool-augmented LMs learn to invoke external APIs for question answering, but do not simulate the API's internal state. AppWorld (Trivedi et al., 2024) provides ground-truth API simulators for daily apps, revealing that even GPT-4 often violates API contracts. Recent work in stable API benchmarking and web agent evaluation (Wang et al., 2024) underscores the need for high-fidelity simulators. Recent frameworks focus on improving API interactions; AutoFeedback (Liu et al., 2024) provides an LLM-based approach for efficient and accurate API request generation with static scanning and dynamic analysis components, while SEAL (Kim et al., 2024) offers a comprehensive evaluation pipeline for API retrieval, calls, and responses. Similarly, specialized frameworks for API argument filling in task-oriented conversations (Mok et al., 2024) address the challenges of grounding LLM responses to pre-defined API schemas.

3 Data Collection

We generated a dataset of API interaction trajectories using an agent (powered by ChatGPT) in an AppWorld simulation of the Todoist API. Each trajectory is a sequence of user actions (API requests) and the resulting API responses, ensuring the requests are causally coherent (e.g. only completing tasks that were actually added). Having real user interaction trajectories from the actual Todoist app would be ideal, but no such public data exists, and collecting trajectories from the actual Todoist API would be unfeasible due to rate limits, the inability to reset the state, and the potential corruption of real, valuable data. We hence employ AppWorld (Trivedi et al., 2024), a one-to-one implementation of the most popular apps and services, including Todoist, specifically created for AI training workflows. We use AppWorld's implementation of Todoist to sample trajectories with the assumption that they truthfully represent the behavior of the actual app.

The setup of our Todoist app, or our initial world state, is static and consists of some pre-populated data belonging to one user. We implement a semi-random agent that collects the app usage trajectories on behalf of that user. The agent randomly

Algorithm 1: Data Collection

Input: *Todoist, TodoistAPISchema, LLM, initialRequests, maxContextLength*
Output: *trajectory*

```
Initialize trajectory with
  initialRequests
while TOKEN_COUNT_OF(trajectory) <
  maxContextLength do
    prompt ←
      BUILD_PROMPT(TodoistAPISchema,
      trajectory)
    newRequest ←
      LLM.GENERATE_REQUEST(prompt)
    newResponse ←
      AppWorldTodoist.EXECUTE(newRequest)

    trajectory.APPEND((newRequest,
      newResponse))
end
return trajectory
```

picks API requests from the API schema and executes them in the AppWorld Todoist environment. However, if we picked all request arguments randomly, most of the API responses would be 404 Not Found because most of the API endpoints operate on objects that should already be in the system. For example, to create a task you need to supply the project ID the task should belong to. Thus, the agent should follow a certain notion of causality between requests in a trajectory. To implement this, we make our agent use an LLM, specifically ChatGPT-4o mini. The agent provides the API schema and the trajectory requests and responses in a prompt (see the prompt in Appendix A) to the LLM and asks it to generate a new request for a random endpoint in the schema, filling all non-id arguments with random values and picking IDs from the corresponding entity IDs found in past requests and responses. For example, if the LLM sees there were created projects with IDs 101 and 102, it will pick either 101 or 102 as the project ID argument for the consequent create task request. All trajectories start with the same list of initial requests and responses, creating some entities in the system and representing the initial state. The trajectory length is the number of request-response pairs recorded in a trajectory. The agent keeps adding steps to a trajectory until its number of tokens starts exceeding the context length of our generative model. The pseudo-code for the data collection algorithm is in algorithm 1.

In our methodology, a semi-random agent samples API endpoints with set weights. The agent samples each API endpoint e according to the discrete distribution

$$\begin{aligned} p(\texttt{create}) &= 0.40, \\ p(\texttt{read}) &= 0.10, \\ p(\texttt{update}) &= 0.25, \\ p(\texttt{delete}) &= 0.15, \\ p(\texttt{add\_relationship}) &= 0.05, \\ p(\texttt{remove\_relationship}) &= 0.05. \end{aligned}$$

Ensuring required parent objects exist. After choosing an endpoint and IDs, we call ChatGPT with the prompt: generate a function-call request with realistic argument values. Each episode comprises 50 random calls plus recursive reads, truncated to 4K tokens. We collect 3K episodes (1.2M (action, response) pairs).

4 Method

To simulate an app's API, our model must learn to generate the correct API response conditioned on the incoming request and the full history of prior interactions. Analogous to reinforcement-learning terminology, we treat API requests as *actions* a_t and API responses as *observations* o_t. Concretely, we parameterize a conditional distribution

$$p_\theta\big(o_t \mid a_t,\ o_{t-1}, a_{t-1}, \ldots, o_0, a_0\big),$$

where o_t is the API's response at step t and a_t is the request issued at that step.

4.1 Finetuning an open-source LLM

To approximate this distribution, we employ a decoder-only Transformer, i.e. the input stream concatenates the tokenized sequence of past actions and observations followed by the current action a_t, and the model autoregressively emits the tokens of the next observation o_t .

Given a tokenized sequence of past actions $a_{0:t-1}$ and observations $o_{0:t-1}$, we optimize:

$$\mathcal{L} = -\sum_t \log P_\theta(o_t \mid a_{t-1}, o_{t-1}, \ldots, a_0, o_0).$$

In our methodology, we compare two adaptation strategies:

- Full fine-tuning: update all model parameters.
- LoRA: inject low-rank adapters into each attention layer, freezing base weights.

We fine-tune our Transformer world model variants on the collected AppWorld trajectories using a standard next-token cross-entropy objective. Each training sample is obtained by uniformly sampling a timestep t from a trajectory and concatenating the tokenized sequence

$$\big[a_t,\ o_{t-1},\ a_{t-1},\ \ldots,\ o_0,\ a_0\big]$$

as model input, and the corresponding observation tokens o_t as the target. To focus the optimization on API response generation, we mask out request tokens (all a_i) in the loss computation, so that gradients flow only through tokens belonging to the response o_t.

At runtime, when an agent proposes an action a_t, the model takes in a_t plus the recent history and generates o_t, the predicted API response. We then feed o_t back to the agent as if it came from the real API. This loop continues for multi-step simulations.

4.2 Zero-shot LLM inference

This approach doesn't involve any fine-tuning. Instead, we supply the LLM with the explanation of what it should do, the API schema, and some meta information. You can see the exact prompt in Appendix B.

The generation of the trajectory is the same in this approach. When an agent proposes an action a_t, the model takes in a_t plus the recent history and generates o_t, the predicted API response. We then feed o_t back to the agent as if it came from the real API. This loop continues for multi-step simulations.

5 Evaluation

We replay 600 held-out episodes. At each step, we feed the history plus action and sample the next response. Metrics:

- Object-match accuracy: exact match on key fields (IDs, names).
- BLEU-4 and ROUGE-L on tokenized responses.

Through the quantitative and qualitative analyses, we find that proprietary LLMs perform much

Model	Acc_{tot}	BLEU	Time (s)	Cost ($)
GPT o3	0.74	56.4	450	3.00
GPT 4.1	0.70	57.3	60	0.70

Table 1: Zero-shot GPT-4 baselines on held-out Todoist trajectories.

Variant	Acc_{tot}	BLEU	Time (s)
Full fine-tune	0.00	8.0	7
LoRA fine-tune	0.00	0.0	5

Table 2: TinyLlama world-model variants on the same test set.

better than the small LM we finetuned. They respect the syntax and format of the API schema. You can find an example of a generated trajectory by GPT 4.1 in Appendix C. The small finetuned models fail to output a valid JSON and don't follow the API schema.

Another thing we observe is that the small LMs are much faster at inference than the powerful LLMs.

6 Discussion

Our empirical results reveal several key insights about learned simulation of API dynamics. First, large pre-trained models such as GPT-4 (ChatGPT) exhibit surprisingly strong zero-shot performance on the Todoist API task (75% total object accuracy), owing to the deterministic, template-like nature of CRUD operations that such models have likely memorized during web-scale pretraining. This illustrates that off-the-shelf LLMs can serve as rudimentary simulators for simple, well-templated APIs. However, we argue that as the API behavior scales in complexity, the zero-shot method won't be able to produce accurate results.

A trained, dedicated world model is a more promising approach. Yet as we learned in our experiments, to produce satisfactory results, a much higher scale of data is needed. Our dataset comprises only ∼12M tokens across 3K trajectories, whereas in model-based RL, robust dynamics simulation typically requires on the order of 1B tokens to generalize faithfully (Hafner et al., 2020). The limited data budget thus constrains the model's ability to internalize complex API semantics and long-range dependencies. In addition to data scaling, the model size should scale as well to much that of the proprietary LLMs.

7 Conclusion

We have introduced **AppSim**, the family of fully differentiable world models for a Todoist-style task-management API. Leveraging the AppWorld simulation of Todoist, we collected trajectory data and finetuned the TinyLlama model together with a powerful LLM model we used in a zero-shot setting. Our work demonstrates that the latter can effectively simulate the dynamics of text-based interfaces, enabling agents to "imagine" the effects of API calls. At the same time, our study surfaces key challenges. A small model finetuned on the custom dataset of trajectories requires a higher scale of data and model to produce satisfactory results.

Ethics Statement

All data is synthetically generated in a sandbox environment; no personal user data was used. Our simulator avoids real user privacy concerns.

References

Prithviraj Ammanabrolu and Mark O. Riedl. 2021. Learning knowledge graph-based world models of textual environments.

Zhiqi Ge, Hongzhe Huang, Mingze Zhou, Juncheng Li, Guoming Wang, Siliang Tang, and Yueting Zhuang. 2024. Worldgpt: Empowering llm as multimodal world model. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 7346–7355.

Yu Gu, Boyuan Zheng, Boyu Gou, Kai Zhang, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, Huan Sun, and Yu Su. 2024. Is your llm secretly a world model of the internet? model-based planning for web agents.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. *arXiv preprint arXiv:2403.07714*.

David Ha and Jürgen Schmidhuber. 2018. World models. *arXiv preprint arXiv:1803.10122*.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. 2020. Dream to control: Learning behaviors by latent imagination.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. 2024. Mastering diverse domains through world models.

Mengkang Hu, Tianxing Chen, Yude Zou, Yuheng Lei, Qiguang Chen, Ming Li, Yao Mu, Hongyuan Zhang, Wenqi Shao, and Ping Luo. 2025. Text2world:

Benchmarking large language models for symbolic world model generation. *arXiv preprint arXiv:2502.13092*.

Woojeong Kim, Ashish Jagmohan, and Aditya Vempaty. 2024. Seal: Suite for evaluating api-use of llms. *arXiv preprint arXiv:2409.15523*.

Huanxi Liu, Jiaqi Liao, Dawei Feng, Kele Xu, and Huaimin Wang. 2024. Autofeedback: An llm-based framework for efficient and accurate api request generation. *arXiv preprint arXiv:2410.06943*.

Jisoo Mok, Mohammad Kachuee, Shuyang Dai, Shayan Ray, Tara Taghavi, and Sungroh Yoon. 2024. Llm-based frameworks for api argument filling in task-oriented conversational systems. *arXiv preprint arXiv:2407.12016*.

Pietro Novelli, Marco Prattichò, Massimiliano Pontil, and Carlo Ciliberto. 2024. Operator world models for reinforcement learning. *arXiv preprint arXiv:2406.19861*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551.

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.

Dani Valevski, Yaniv Leviathan, Moab Arar, and Shlomi Fruchter. 2024. Diffusion models are real-time game engines.

Ruoyao Wang, Graham Todd, Ziang Xiao, Xingdi Yuan, Marc-Alexandre Côté, Peter Clark, and Peter Jansen. 2024. Can language models serve as text-based world simulators? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–17.

Kaige Xie, Ian Yang, John Gunerli, and Mark Riedl. 2024. Making large language models into world models with precondition and effect knowledge.

A Appendix: Data Generation Prompt

Below is the exact prompt template used to drive the LLM for synthetic trajectory generation in AppSim:

```
NEW_REQUEST_PROMPT_TEMPLATE = """
I want you to help me
explore the Todoist API.
For this I want you to
generate a request
based on the endpoint schema.
I will also provide
some of the arguments
that I want you to use in the request.
For all other arguments
just come up with random content
that matches the type of the argument,
be creative.
For create requests,
make sure to use unique names
for the objects based on
what names are in the history of requests.
Output just the request in the format
of calling a python function,
pass all arguments as named parameters,
don't pass default values as None,
just skip them,
skip the `TodoistApis` prefix,
don't wrap the function in any other text,
output the function as a single line text

ENDPOINT SCHEMA:

{endpoint_schema}


PRESET ARGS:

{args}

HISTORY:

{history}

NEW REQUEST:
"""
```

This prompt ensures consistent, schema-compliant LLM-driven request generation, seeding the AppWorld environment and producing diverse, causally coherent trajectories for downstream model training.

B Appendix: GPT Zero-shot prompt

Below is the exact prompt template used to simulate the API with a powerful LLM:

```
PROMPT_TEMPLATE = """
You are a simulator of the
Todoist app API.
You need to predict the next response
of the API based on a history of
API requests and responses.
For this I will provide
```

```
a detailed json schema of the API,
but since the requests and responses
in the history are in the format
of Python function calling,
I will also provide
a Python schema of the API
so you can map between the two.
You can assume that
at the beginning of the history,
the app is in a empty state,
i.e no objects exist in the system
except for one user object
who executes the requests.
The user's name is "Timothy White"
and their email is
"timothy.whit@gmail.com".
However, when creating new objects
you should assign IDs starting not from 1,
but from the corresponding number
provided in starts IDs.
For example, when creating a first project,
its id should be 280.
When creating new objects
always use '2023-05-23T09:24:06'
as the creation date.
Certain endpoints
contain messages in the response,
look at the messages I provide
to pick the right one.
Output just the next response of the API
as a python object (not a JSON object)
on a single line with no indentation.
Don't include the "res:" prefix.

JSON_SCHEMA:

` ` `
{json_schema}
` ` `

PYTHON_SCHEMA:

` ` `
{python_schema}
` ` `

START_IDS:

` ` `
{start_ids}
` ` `

MESSAGES:

` ` `
{messages}
` ` `

HISTORY:

` ` `
{trajectory_str}
` ` `

NEXT_RESPONSE:
"""
```

C Appendix: Full Trajectory Example

An example of a generated trajectory by GPT-4:

Request	Response
search_labels(query="important projects", color="blue", task_attached=True)	[]
create_label(name="Team Collaboration", color="orange")	{'message':'Label created.', 'label_id':2294}
create_project(name="Innovative Solutions", color="green", description="A project aimed at developing cutting-edge technology.", is_favorite=True)	{'message':'Project created.', 'project_id':280}
update_project(project_id=280, name="Revamped Strategies", color="red", description="Optimize workflows and improve team efficiency.", is_favorite=False, is_archived=False)	{'message':'Project information updated.'}
create_project(name="Creative Ideas", color="yellow", description="Brainstorm and develop innovative concepts.", is_favorite=True)	{'message':'Project created.', 'project_id':281}
show_projects(query="New Marketing Campaign", color="orange", is_favorite=True, is_archived=False)	[]
delete_label(label_id=2294)	{'message':'Label has been deleted.'}
create_project(name="Sustainable Growth", color="blue", description="Eco-friendly initiatives and practices.", is_favorite=True)	{'message':'Project created.', 'project_id':282}
create_task(project_id=280, title="Implement AI Features", description="Develop and integrate AI functionalities.", duration=5.0, duration_unit="days", order_index=1, priority="high")	{'message':'Task created.', 'task_id':6334}
create_project(name="Dynamic Strategies", color="orange", description="Enhance adaptive methods and strategies.", is_favorite=False)	{'message':'Project created.', 'project_id':283}
create_task(project_id=280, title="Enhance User Experience", description="Improve the application UI/UX.", order_index=2, priority="medium")	{'message':'Task created.', 'task_id':6335}
create_sub_task(task_id=6335, title="Research Emerging Technologies", description="Investigate latest technology advancements.", priority="high", order_index=1)	{'message':'Subtask created.', 'sub_task_id':9636}
show_project(project_id=281)	{'name':'Creative Ideas','color':'yellow','is_inbox':Fals 'description':'A project designed to brainstorm and develop innovative concepts.', 'is_favorite':True,'is_archived':False,'created_at':'2023-05-23T09:24:06', 'project_id':281,'creator':{'name':'Timothy White','email':'timothy.whit@gmail.com'}, 'num_tasks':0,'sections':[{'id':None,'name':None,'num_tasks':0}], 'collaborators':[{'name':'Timothy White','email':'timothy.whit@gmail.com'}], 'pending_invites':[],'is_shared':False }

Table 3: Complete AppSim trajectory illustrating label, project, task, and subtask interactions.