

CrafText Benchmark: Advancing Instruction Following in Complex Multimodal Open-Ended World

Zoya Volovikova^{1,2} Gregory Gorbov^{2,3} Petr Kuderov^{1,2}
Aleksandr I. Panov^{1,2,3} Alexey Skrynnik^{1,2}

¹AIRI, Moscow, Russia

²MIPT, Moscow, Russia

³FRC CSC RAS, Moscow, Russia

{volovikova, kuderov, panov, skrynnik}@airi.net

Abstract

Following instructions in real-world conditions requires a capability to adapt to the world’s volatility and entanglement: the environment is dynamic and unpredictable, instructions can be linguistically complex with diverse vocabulary, and the number of possible goals an agent may encounter is vast. Despite extensive research in this area, most studies are conducted in static environments with simple instructions and a limited vocabulary, making it difficult to assess agent performance in more diverse and challenging settings. To address this gap, we introduce CrafText, a benchmark for evaluating instruction following in a multimodal environment with diverse instructions and dynamic interactions. CrafText includes 3,924 instructions with 3,423 unique words, covering Localization, Conditional, Building, and Achievement tasks. Additionally, we propose an evaluation protocol that measures an agent’s ability to generalize to novel instruction formulations and dynamically evolving task configurations, providing a rigorous test of both linguistic understanding and adaptive decision-making.

1 Introduction

Instruction following is a research area dedicated to developing methods that enable an agent to act within an environment based on natural language instructions and sensory input, such as visual observations.

In real-world scenarios, decision-making in instruction-following tasks becomes increasingly complex due to the world’s volatility and intricate interdependencies. The environment is dynamic and unpredictable, requiring the agent to continuously adjust its decisions. Additionally, many instructions involve interactions with objects, and since the world contains a vast number of objects—each with multiple possible interactions—the space of potential goals further



Figure 1: An illustration depicting an agent navigating the CrafText environment to solve a task. The agent progresses from the starting point towards the northernmost lake, collecting the necessary resources along the way to set up a crafting table. A *Done* marker indicates the location where the task is completed.

increases. Furthermore, instructions themselves can be phrased in multiple ways, making proper grounding essential: the agent must correctly interpret the instruction in terms of its intended goal and determine the specific objects and interactions required to achieve it.

Thus, an AI agent faces two primary challenges: (1) making decisions in a dynamically changing environment, and (2) generalizing its ability to follow instructions across diverse tasks and formulations by effectively linking natural language to observations.

Individually, both challenges are well studied. Decision-making under dynamic and uncertain conditions in Reinforcement Learning (RL) has been explored in various studies (Padakandla, 2021; Attar and Dabirian, 2019). RL-based methods excel in open-world scenarios, where agents adapt to procedurally generated environments, evolving objects, and unpredictable interactions (Hafner, 2022;

Matthews et al., 2024; Guss et al., 2019). These approaches enable generalization, allowing agents to transfer learned policies to new settings and interact with unseen objects (Stanić et al., 2023).

Ways to connect different modalities, such as text and vision, can be found in studies on language grounding. Architectures like CLIP (Yao et al., 2022) and FILM (Perez et al., 2018) facilitate this connection by mapping textual instructions to visual inputs. These models have shown strong performance in tasks such as visual question answering (VQA) (de Faria et al., 2023; Ishmam et al., 2024), open-vocabulary segmentation, and image description (Zhang et al., 2022; Yao et al., 2022).

There are also studies that bridge language grounding and RL, addressing the problem of instruction following. These approaches leverage FiLM (Zhong et al., 2019) and CLIP (Paischer et al., 2023; Lynch et al., 2023) to encode task-relevant features from both textual and visual modalities. However, these studies are constrained by the environments they rely on, which are often static or use procedurally generated instructions, limiting linguistic diversity. While environments like (Lin et al., 2023), (Hanjie et al., 2021), and (Zhong et al., 2019) introduce dynamics, their vocabulary remains restricted due to template-based instruction generation. Conversely, (Shridhar et al., 2020), (Chen et al., 2019) and (Chen et al., 2019) offer richer vocabularies but lack diverse interactions and environmental dynamics. Such settings narrow the search space for optimal policies, making it difficult to assess how well trained algorithms generalize to more dynamic and linguistically varied real-world conditions.

In this paper, we introduce CraFText, a benchmark designed to evaluate an agent’s ability to follow complex natural language instructions in an interactive, multimodal environment. Unlike existing benchmarks, CraFText features an open-ended world dynamic with amount objects and way to interact with it, where objects change properties over time, requiring agents to continuously adapt and generalize beyond fixed action sequences.

We developed a linguistically diverse dataset with 3,924 instructions and 3,423 unique words to support instruction-following in this challenging setting. Tasks fall into four categories—Localization, Conditional, Building, and Achievement—evaluating an agent’s ability to interpret directions, follow conditions, construct

structures, and achieve complex goals. Each instruction is paired with validation functions to systematically assess whether an RL agent successfully completes the specified goals.

Additionally, we propose a specialized evaluation protocol to test the agent’s ability to generalize to novel instruction formulations and unseen goal configurations, providing a rigorous measure of both linguistic flexibility and adaptive decision-making.

To summarize, we make the following contributions:

- We introduce CraFText, a benchmark for training agents on goal-driven tasks with natural language instructions in a dynamic environment.
- We develop a dataset of 3,924 instructions, categorized into four distinct task types, Localization, Conditional, Building, and Achievement, along with validation functions for precise task verification in RL.
- We propose an evaluation protocol that assesses both linguistic adaptability and generalization to novel goal configurations.
- We conduct a thorough evaluation, analyzing agent performance in CraFText using two well-known baselines: PPO with instruction embeddings as goal representations and Dynalang, a specialized approach for multimodal tasks.
- We implement CraFText as an open-source benchmark, providing its dataset and evaluation framework with support for XLA acceleration, enhancing computational efficiency and scalability. The dataset and code for CraFText are publicly available¹.

2 Related Work

In this section, we provide an overview of existing multimodal environments and approaches for training agents in language-grounded decision-making. The focus is on highlighting the limitations of current environments in terms of versatility, dynamism, and language grounding, contrasted with the capabilities of CraFText, designed to address these gaps.

Environments For Instruction Following.

¹<https://sites.google.com/view/crafttext>

Environment	Repository	Vocabulary >3000 words	Maximum Instruction Length >50	Benchmarking Diverse Abilities	Stochastic Transitions	Dynamic World	Game Objects >50	GPU Accelerated	Procedural Generated World	Dual Evaluation Protocol
HomeGrid	link	✗	✗	✗	✗	✓	✗	✗	✓	✗
BabyAI	link	✗	✗	✗	✗	✓	✗	✗	✓	✗
RTFM	link	✗	✓	✗	✗	✓	✗	✗	✓	✗
Messenger	link	✗	✓	✗	✗	✓	✗	✗	✓	✗
Touchdown	link	✓	✓	✗	✗	✗	✗	✗	✓	✗
Alfred	link	✓	✗	✗	✗	✗	✓	✗	✓	✗
Cereal Bar	link	✓	✗	✗	✓	✗	✗	✗	✓	✗
IGLU	link	✗	✓	✗	✓	✗	✗	✗	✗	✗
MineDojo	link	✓	✓	✓	✓	✓	✓	✗	✓	✗
CraftAssist	link	✓	✓	✗	✓	✓	✓	✗	✓	✗
CrafText (ours)	link	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 1: This table provides a comparison of CrafText with several other multimodal environments across various characteristics, including vocabulary size, instruction length, benchmarking capabilities, stochastic transitions, world dynamics, the number of game objects, GPU acceleration, procedural world generation, and evaluation protocols. CrafText offers a well-balanced combination of features, supporting both stochasticity and dynamic world elements, with a significant number of game objects.

Our research focuses on developing benchmarks for training and evaluating agents in instruction-following tasks within dynamically changing environments, featuring diverse goals, interaction possibilities, and complex linguistic structures.

However, most existing environments are not well-suited for this research direction, as they are often designed for a specific task. For example, CraftAssist (Gray et al., 2019) and IGLU (Kisileva et al., 2023) center around 3D construction but remain deterministic, preventing environmental changes from influencing decision-making. Similarly, Touchdown (Chen et al., 2019), Alfred (Shridhar et al., 2020) and CerealBar (Suhr et al., 2019), despite offering a rich instruction vocabulary, lack environmental dynamics and provide only a limited range of interactions with objects.

Dynamic environments such as HomeGrid (Lin et al., 2023), BabyAI (Chevalier-Boisvert et al., 2018), RTFM (Zhong et al., 2019), and Messenger (Hanjie et al., 2021) allow state changes, but their object interactions are limited, and their instructions are often procedurally generated, reducing linguistic diversity. More detailed information about each environment can be found in the Appendix H

Moreover, none of the environments, including MineDojo (Fan et al., 2022), which allows configuring dynamic conditions, provide a dual evaluation protocol for analyzing both the ability to generalize to new linguistic constructions and the ability to generalize to new goals.

In contrast to existing environments, CrafText presents an instruction-following challenge in a complex open-ended world: it features a dynamic environment, a large number of interactive objects with diverse interaction possibilities, and a rich set of linguistically varied instructions. This creates a vast decision space where the agent must learn to find an optimal policy. Additionally, the benchmark is fully implemented in JAX, utilizing XLA acceleration for efficient large-scale experimentation.

Instruction Following Task Language grounding in intelligent agents focuses on enabling agents to understand and execute textual instructions within virtual environments. Instruction following requires models to map natural language commands to appropriate actions, often integrating visual and textual information. Approaches to policy learning in multimodal environments can be categorized into three main groups. The first group utilizes CLIP (Radford et al., 2021) to establish a shared representation space for textual and visual inputs, aiding in instruction grounding (Paischer et al., 2023; Lynch et al., 2023). The second group employs projection layers (Perez et al., 2018; Zhong et al., 2019; Hanjie et al., 2021) to align linguistic and perceptual features. The third category integrates transformer-based architectures, such as EmBERT (Suglia et al., 2021) and Vision-and-Language Navigation (Savva et al., 2019), to process multimodal signals for task execution. Additionally, methods like those in (Li et al.,

2022) and (Brohan et al., 2023) leverage transformers to improve instruction following. Finally, approaches such as Dynalang (Lin et al., 2023) employ cross-attention or hidden subspace compression to enhance the flexibility of text-conditioned policies, often within model-based reinforcement learning frameworks (Hafner et al., 2023). Beyond these, some methods frame instruction following not merely as a problem of multimodal alignment, but as a planning task — decomposing instructions into sequential steps that are issued to the agent over time (Volovikova et al., 2024).

3 Problem Statement

The environment is formalized as a **goal-based Partially Observable Markov Decision Process (POMDP)**, represented by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{R}, \mathcal{G}, \gamma)$. This framework models the task of grounding natural language instructions to agent behavior in a partially observable and dynamic environment. The agent must interpret a provided textual instruction I , infer the underlying goal $g \in \mathcal{G}$, and execute actions to achieve it.

The state space \mathcal{S} describes all possible configurations of the environment, while \mathcal{A} defines the agent’s actions. Observations $o \in \mathcal{O}$ provide partial information about the current state and include both environmental data and the instruction I , which describes the desired outcome of the agent’s actions. The agent must use I to recover a latent goal $g \in \mathcal{G}$ by approximating a grounding function $f_g(I)$ that predicts g from the instruction. This grounding process forms the core of the task, requiring the agent to apply natural language processing techniques to interpret the instruction and link it to a meaningful goal.

The policy $\pi(a | o)$ maps the agent’s actions to observations o . The agent maintains a belief state $b(s)$, a probability distribution over possible states, to handle partial observability and guide decision-making under uncertainty.

The objective is to find an optimal policy π^* that maximizes the expected cumulative reward over time. Formally,

$$\pi^* = \arg \max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^T \gamma^t R(s_t, a_t, g) \mid o_0 \right].$$

The environment introduces stochasticity through its transition dynamics $\mathcal{T}(s' | s, a)$, where the same action may lead to different outcomes. It

is also dynamic, as states can evolve independently of the agent’s actions due to external factors like autonomous entity movement. These complexities demand a robust ability to interpret instructions, infer goals, and act effectively under uncertainty, forming the core challenges of this framework.

4 CraText

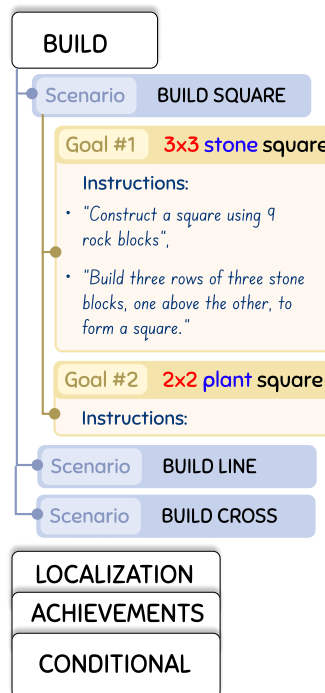


Figure 2: The figure illustrates the hierarchical structure of the CraText dataset. Each category contains multiple scenarios, each scenario includes different goals, and each goal is associated with multiple variations of instruction phrasing.

CraText is a benchmark designed to evaluate an agent’s ability to follow instructions in a complex, open-ended environment. The goal of the agent is to solve a diverse set of tasks, formulated in different ways in natural language, under dynamic conditions. The world is taken from CraText (Matthews et al., 2024) (Appendix G), providing a rich and interactive setting for agent evaluation.

As part of CraText, we provide a **dataset** (see Section 4.1) that includes instructions from various task categories. The language used in these instructions is complex, featuring a large vocabulary. Each instruction is paired with a function that checks whether the agent has met the specified goal. Additionally, we introduce an **instruction generation framework** (Section 4.2), which was

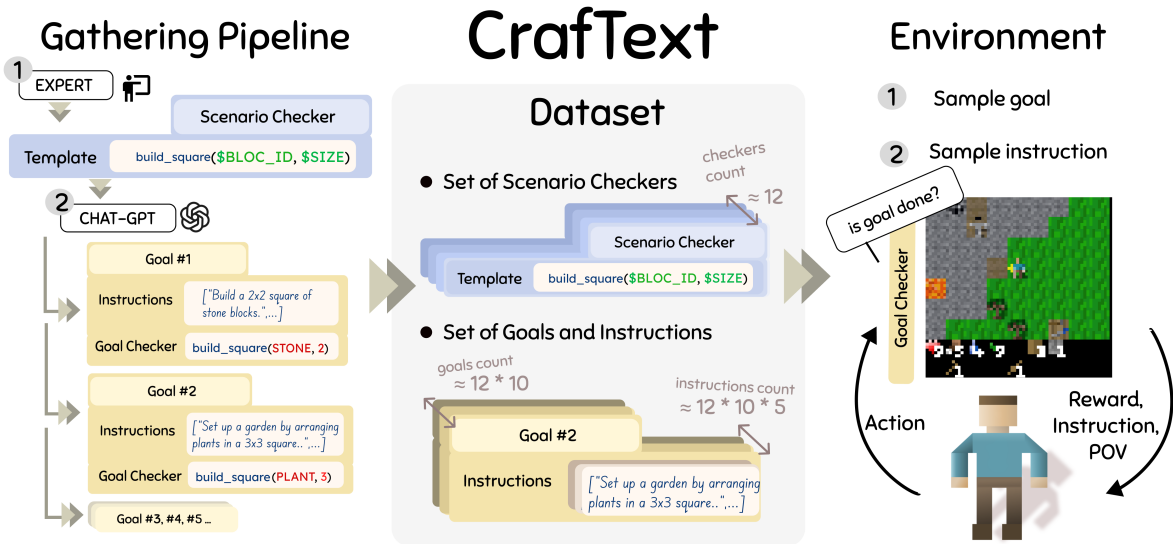


Figure 3: Left: *Data Gathering Pipeline* – experts define goal templates expanded with GPT to generate tasks, instructions, and goal-checking functions (e.g., build a square with stones or plants). Middle: *CraffText Dataset* – features **162 goals** and **972 instructions** (162×6), combining scenario checkers, goals, and instructions with varied parameters like block type and size. Right: *Interactive Environment* – the agent follows instructions and takes actions based on visual observations, while Goal Checkers verify progress by evaluating the state. The environment provides rewards and updates until the goal is achieved.

used to collect the data. This system allows for the expansion of the dataset by incorporating new tasks and scenarios while maintaining linguistic complexity.

We also present the **CraffText environment** (Section 4.3), which enables issuing new instructions to the agent within an episode and performing verification at each step. The implementation leverages JAX for high performance and computational speed, ensuring efficient training of agents in dynamic conditions (Appendix G).

The data gathering pipeline, an overview of the dataset, and the agent-environment loop are illustrated in Figure 3.

4.1 Dataset: Overview and Structure

The CraffText dataset has a hierarchical structure (Figure 2), distinguishing three key concepts: *Scenarios*, *Goals*, and *Instructions*. Scenarios represent abstract task classes that are not yet parameterized, such as building a square or placing an object at a certain distance from another. Goals are parameterized instances of these scenarios, specifying concrete details like building a 2x2 wooden square or placing a table to the left of a lake. Instructions are natural language variations of a goal’s formulation, providing multiple ways to express the same objective (see Appendix C for examples).

To verify the completion of each instruction, we call a function that checks the execution of the scenario corresponding to the instruction with the given parameters. For example, for the instruction "Place a 2x2 wooden square," we call the Square Placement checker function with the parameters block = WOOD and side length = 2.

All scenario verification functions were implemented by a person familiar with the mechanics of the CraffText environment. In total, we have 12 scenarios across four task categories: *Building*, *Conditional*, *Localization*, and *Achievements*.

Scenarios Categories. *Building* scenarios require the agent to construct a specified structure while remembering the starting point, since it may need to step away during construction to gather additional resources for placing blocks. In *Conditional* scenarios, the agent’s understanding of instructions is tested with tasks such as "Craft a sword after gathering two stones," "Craft a sword before gathering two stones," or "Craft only a sword or only a pickaxe – whichever is needed for construction." *Localization* scenarios evaluate the agent’s ability to correctly interpret spatial instructions, including compass directions (south, east, west, north) and relative directions (to the right, above, to the left, below) to accurately position objects relative to other map elements. Fi-

nally, *Achievement* scenarios involve the agent performing standard in-game tasks—such as collecting wood, mining a diamond, or building a furnace—without additional complications in the instructions, sometimes requiring multiple achievements at once or the exclusion of certain actions. A more detailed breakdown of task categories can be found in the Appendix B, along with examples of instructions.

Dataset Complexity Levels. In Craftax, which serves as the foundation for the benchmark, interacting with objects requires completing preliminary action sequences of varying lengths. For example, placing a stone involves the following steps: collecting wood → crafting a table → making a pickaxe → mining stone. Based on the length and complexity of these sequences, instructions are categorized into three difficulty levels: *Easy*, *Medium*, and *Hard*. The *Easy* category includes tasks from the *Achievement* scenarios, assessing the agent’s ability to complete in-game achievements and their combinations. The *Medium* category covers all scenario types but is limited to tasks requiring relatively short action sequences (fewer than 10 steps). The *Hard* category includes tasks with either highly complex goals or long action sequences. A detailed list of task categories, their difficulty levels, and corresponding examples can be found in Appendix B.

Test Dataset. A key feature of our dataset is the inclusion of a dedicated test set designed to evaluate the agent’s ability to generalize beyond the training distribution. The *Paraphrased* subset assesses whether the agent can achieve the same goals encountered during training when instructions are reworded, allowing us to measure its ability to generalize across linguistic variations.

The *New Objects* subset evaluates the agent’s understanding of object properties by introducing new combinations of familiar objects. While all objects were present during training, they now appear in novel configurations. For example, if the training set included placing a stone block next to a lake and constructing a square from plantations, the test set might require constructing a square from stone blocks. If the agent succeeds, it demonstrates an understanding that stone blocks, like plantations, can be placed and that any placeable object can be used to construct geometric shapes such as squares.

Examples of instructions in these test datasets can be found in Appendix C.

Dataset Size. The dataset consists of 12 sce-

narios with a total of 476 goals, 203 of which are reserved for testing. The goals are distributed based on complexity: 100 are classified as *Easy* (fully encompassing the *Achievements* category), 277 as *Medium* (which consists of multiple categories, including *Achievements*, leading to instruction overlap), and 219 as *Hard*. For each goal, we provide approximately 5–6 instructions, resulting in a total of around 3,924 instructions. The vocabulary size (unique word count) is 2,923. Detailed information is available in Appendix B.

4.2 Instruction Generation Pipeline

In our benchmark, we aim to ensure both a substantial volume of textual instructions and a complex instructional language. Many existing approaches generate instructions by relying solely on procedural templates, resulting in limited linguistic diversity and a small vocabulary, despite a high instruction count. To address this, we introduce the *Instruction Generation Pipeline*, which combines procedural goal generation for precise verification with large language models to achieve linguistic diversity.

The pipeline is centered around scenario checker functions implemented by our team, which encode the logic for goal verification. For each scenario checker, we define an acceptable range of parameters—such as block type, shape, and relative position—forming a reusable template for generating a large number of potential goals. For example, the template `BUILD_&SQUARE_&WITH_&STONE_&` includes the parameters `shape = square` and `material = stone`. By enumerating combinations of such parameters, we obtain a broad space of goals. A subset of these goals is selected for further refinement and instruction generation (see Appendix J).

At this stage, GPT-4 is used to generate natural language instructions and paraphrases corresponding to each selected goal, as well as the function call format needed to invoke the appropriate checker with the correct arguments. GPT-4 is prompted using a fixed, task-aware template (Appendix J), and operates strictly as a language generator, without contributing to the underlying task logic or correctness checks.

This approach enables the automatic generation of a diverse set of instructions from a single input scenario, requiring only the implementation of a scenario verification function and its param-

eter template. We evaluated the scalability of the dataset (see Appendix E) and demonstrated that, based on only 12 scenarios, our dataset generation pipeline can be empirically expanded to produce up to 2,895 goals and more than 14,475 instructions.

4.3 Environment

We developed the CraText environment, an extension of Craftax that enables natural language instructions in the agent-environment loop. This extension is compatible with both *Craftax-Classic* and the full version of *Craftax*. The both possible observation types (visual and vector-based) of *Craftax* are augmented with instructions.

Episode Start: At the start of each episode, the instruction and corresponding checker are randomly selected from the available options. It’s worth noting that a single checker can correspond to multiple instructions because it’s reusable through parameters. After that, the world is created, meaning that even for the same instruction, the environment can vary.

Reward System The agent receives a reward of 1 for completing an instruction at the end of the episode. To verify completion, at each step, we run the corresponding Scenario Checker with goal-specific parameters. Additionally, we use the reward provided by the Craftax environment to incentivize discovering new achievements, scaling it down by a factor of 50.

Episode Termination An episode concludes either when the step limit is reached (episode truncation), when the agent dies, or when the scenario checker function confirms that the instruction has been successfully completed.

JAX-acceleration The entire code for the checker functions is implemented in *JAX*, making the environment highly parallelizable (using JIT compilation) and allowing fast, large-scale training with GPU acceleration.

5 Experiments

In this section, we address the following research questions (RQs) through comprehensive experiments:

RQ1. (Task Handling in Dynamic Environments): How difficult is it for the agent to manage various categories of tasks in dynamic conditions?

RQ2. (Generalization to New Instructions and Goals): Can the agent generalize its behavior:

- To paraphrased instructions that define the

same goals as in the training set but are presented differently (Test Paraphrased)?

- To new goals that require solving familiar tasks but involve different combinations of objects and interactions not seen during training (Test New Objects)?

To address these questions, we train several methods in the Medium part of our dataset and evaluate them using Success Rate (SR) as the primary metric. SR measures the proportion of episodes in which the agent successfully completes the given instruction. Each method was validated on two test datasets: *Test Paraphrased* and *Test New Objects*, as described in Section 4.1. To ensure a fair comparison, all models were trained for 48 hours on a Tesla V100 GPU. Below, we describe the baselines used in our evaluation.

PPO-T (Text-Augmented PPO). This method builds on the PPO (Schulman et al., 2017) implementation from the Craftax Baseline, using identical hyperparameters (Appendix L). To enable instruction understanding, we integrate frozen DistilBERT embeddings by extracting the [CLS] token from the final hidden layer as a high-level representation. This embedding is concatenated with visual features to form a joint textual-visual representation. A GRU module is included to maintain contextual memory across time steps, enhancing the agent’s ability to model sequential dependencies.

PPO-T+ (Plan-Augmented PPO). To promote generalization, PPO-T+ extends PPO-T by introducing a planning step: each instruction is converted into a structured plan using GPT-4 (Appendix J). The agent then follows the same training procedure as PPO-T, utilizing both visual observations and the GPT-generated plans to inform decision-making.

FiLM. This baseline adopts PPO as the optimization method and integrates language through FiLM layers (Perez et al., 2018), which apply feature-wise affine transformations to visual features conditioned on text. In our setup, transformation parameters are computed from the instruction and modulate each convolutional layer in the visual encoder, enabling instruction-aware visual processing.

Dynalang. Dynalang (Lin et al., 2023) is a model-based method built on DreamerV3, designed for grounding language in multimodal environments. Rather than predicting actions directly,





Instruction type	Algorithm	 Conditional	 Build	 Localization	 Achievements	Total
Train Set	PPO-T	0.15	0.25	0.33	0.55	0.40
	PPO-T+	0.17	0.24	0.30	0.70	0.45
	Dynalang	0.00	0.12	0.15	0.17	0.15
	FiLM	0.07	0.38	0.29	0.76	0.43
Paraphrased	PPO-T	0.12	0.13	0.35	0.50	0.36
	PPO-T+	0.16	0.17	0.30	0.48	0.35
	Dynalang	0.00	0.09	0.13	0.10	0.05
	FiLM	0.10	0.20	0.30	0.53	0.35
New objects	PPO-T	0.12	0.13	0.17	0.34	0.22
	PPO-T+	0.20	0.17	0.19	0.43	0.28
	Dynalang	0.00	0.09	0.09	0.14	0.10
	FiLM	0.17	0.20	0.19	0.38	0.26

Table 2: The success rates of PPO, PPO-T+, FiLM, and Dynalang evaluated across 50 seeds, each representing a distinct world for every instruction within the complete set of **Medium Tasks**, where higher success rates indicate better performance. The best-performing approach is highlighted using **tan boxes**.

it forecasts future textual and visual states. Instructions are processed via the **T5** tokenizer, with optional embeddings. This approach achieves strong results in environments such as Messenger and BabyAI, demonstrating its effectiveness in learning predictive, grounded representations.

5.1 Task Handling in Dynamic Environments (RQ1)

Despite extensive training under optimal conditions, all tested algorithms exhibit suboptimal performance on the training set. Dynalang, achieves only a 0.15 success rate (SR), significantly underperforming across individual tasks. This result is particularly notable given DreamerV3’s strong performance in environments like Crafter, which shares similarities with Craftax. One possible explanation is that the combination of complex textual instructions and the dynamic nature of the environment makes the learning process significantly more difficult.

On the other hand, PPO-T, PPO-T+ and FiLM, which incorporate BERT-based instruction encoding without additional modifications, outperform Dynalang but still achieve only moderate success rates (SR: PPO-T = 0.40, PPO-T+ = 0.45, FiLM = 0.43). This suggests that even with stronger language representations, existing methods struggle to generalize effectively in dynamic conditions, highlighting the need for further improvements in instruction-following capabilities.

We also conducted additional experiments on our Easy subset in a zero-shot setting, using large language models such as DeepSeek, MISTRAL, and Qwen/QWQ (see Appendix M). In these experiments, the models received a textual description of the agent’s current view instead of an image. Despite this simplification of the observation space and the impressive reasoning capabilities of modern LLMs, the models still struggled to solve even simple tasks in a dynamic environment. This suggests that, without training a reinforcement learning agent to optimize long-term behavior, it remains too difficult for LLMs alone to make effective decisions for instruction following.

5.2 Generalization to New Instructions and Goals (RQ2)

We observe a general drop in performance on the *Paraphrased* test set across all models. For PPO-T, PPO-T+, Dynalang, and FiLM, the Success Rate (SR) falls from 0.40, 0.45, 0.15, and 0.43 on the training set to 0.36, 0.35, 0.05, and 0.35, respectively.

Interestingly, PPO-T+ suffers the most from paraphrasing, with a drop of 0.10, despite achieving the highest SR on the training set. This suggests that reformulated instructions may lead it to construct execution plans that diverge significantly from those encountered during training, thereby reducing its robustness to linguistic variation.

At the same time, PPO-T+ achieves the high-

est SR on the test set with entirely new goals *New Objects* (SR = 0.28), outperforming FiLM (0.26), PPO-T (0.22), and Dynalang (0.10). This indicates that PPO-T+ is better at decomposing novel instructions into reusable subtasks, even when surface forms are unfamiliar — a sign of effective goal-level generalization.

FiLM comes closest to PPO-T+ in this setting, achieving competitive results without the explicit intermediate-step inference mechanism used by PPO-T+. The strong performance of FiLM may stem from its architectural design: FiLM layers in the encoder allow for more flexible integration of textual and visual information, which proves beneficial in handling novel object configurations.

6 Conclusion

We introduced CraFText, a benchmark for studying instruction following in dynamic environments with diverse objects, interactions, and high-vocabulary textual instructions. Agents must solve tasks across multiple categories, including Construction, Localization, Conditional tasks, and multi-step Achievement-based challenges. We show that agents that performed well in static environments with limited vocabulary, such as Dynalang, struggle with complex linguistic constructions and fail to generalize in dynamic environments. Our benchmark highlights the advantages of planning-based approaches, demonstrating that preprocessing instructions aids in solving novel tasks. However, without the ability to adapt plans based on agent behavior, unfamiliar task formulations remain challenging. Future work could focus on dynamic plan adjustments through fine-tuning with environmental feedback.

7 Acknowledgments

This work was supported by the Ministry of Economic Development of the Russian Federation (code 25-139-66879-1-0003). We thank Dmitry Lukashevsky for contributing to the project after the initial submission by extending the dataset with additional instructions and improving the instruction-checking functions used to evaluate the agent. Although these changes were made post-submission and are not reflected in the original results, we sincerely appreciate his help in improving the benchmark.

8 Limitation

The main limitation of this study is the absence of human-generated instructions in the dataset. While AI-generated instructions offer consistency and scalability, they may not capture the depth and nuance of human-crafted instructions, potentially limiting the model’s ability to generalize to complex, context-rich tasks common in real-world applications. Although ChatGPT-like models enhance accessibility, future work will focus on enriching the dataset with human input.

Another limitation is the lack of real-world interactive elements, where instructions are part of a dynamic conversation involving negotiation, clarification, collaboration, and adaptation. Expanding the benchmark to include such interactions would provide a more comprehensive evaluation of an AI’s ability to handle nuanced, real-life scenarios.

References

- Mehran Attar and Mohammadreza Dabirian. 2019. Reinforcement learning for learning of dynamical systems in uncertain environment: A tutorial. *arXiv preprint arXiv:1905.07727*.
- Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. 2023. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on robot learning*, pages 287–318. PMLR.
- Howard Chen, Alane Suhr, Dipendra Misra, Noah Snavely, and Yoav Artzi. 2019. Touchdown: Natural language navigation and spatial reasoning in visual street environments. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12530–12539. IEEE Computer Society.
- Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. 2018. Babyai: A platform to study the sample efficiency of grounded language learning. *arXiv preprint arXiv:1810.08272*.
- Ana Cláudia Akemi Matsuki de Faria, Felype de Castro Bastos, José Victor Nogueira Alves da Silva, Vitor Lopes Fabris, Valeska de Sousa Uchoa, Décio Gonçalves de Aguiar Neto, and Claudio Filipi Goncalves dos Santos. 2023. Visual question answering: A survey on techniques and common trends in recent literature. *arXiv preprint arXiv:2305.11033*.
- Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. 2022. *Minedojo: Building open-ended embodied agents with internet-scale knowledge*. In *Thirty-sixth*

- Conference on Neural Information Processing Systems Datasets and Benchmarks Track.*
- Jonathan Gray, Kavya Srinet, Yacine Jernite, Haonan Yu, Zhuoyuan Chen, Demi Guo, Siddharth Goyal, C. Lawrence Zitnick, and Arthur Szlam. 2019. [Craftassistant: A framework for dialogue-enabled interactive agents.](#)
- William H Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. 2019. Minerl: A large-scale dataset of minecraft demonstrations. *arXiv preprint arXiv:1907.13440*.
- Danijar Hafner. 2022. [Benchmarking the spectrum of agent capabilities.](#) In *International Conference on Learning Representations*.
- Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. 2023. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*.
- Austin W Hanjie, Victor Y Zhong, and Karthik Narasimhan. 2021. Grounding language to entities and dynamics for generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 4051–4062. PMLR.
- Md Farhan Ishmam, Md Sakib Hossain Shovon, Muhammad Firoz Mridha, and Nilanjan Dey. 2024. From image to language: A critical analysis of visual question answering (vqa) approaches, challenges, and opportunities. *Information Fusion*, page 102270.
- Julia Kiseleva, Alexey Skrynnik, Artem Zholus, Shrestha Mohanty, Negar Arabzadeh, Marc-Alexandre Côté, Mohammad Aliannejadi, Milagro Teruel, Ziming Li, Mikhail Burtsev, et al. 2023. Interactive grounded language understanding in a collaborative environment: Retrospective on iglu 2022 competition. In *NeurIPS 2022 Competition Track*, pages 204–216. PMLR.
- Liunian Harold Li, Pengchuan Zhang, Haotian Zhang, Jianwei Yang, Chunyuan Li, Yiwu Zhong, Lijuan Wang, Lu Yuan, Lei Zhang, Jenq-Neng Hwang, et al. 2022. Grounded language-image pre-training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10965–10975.
- Jessy Lin, Yuqing Du, Olivia Watkins, Danijar Hafner, P. Abbeel, Dan Klein, and Anca D. Dragan. 2023. [Learning to model the world with language.](#) *ArXiv*, abs/2308.01399.
- Corey Lynch, Ayzaan Wahid, Jonathan Tompson, Tianli Ding, James Betker, Robert Baruch, Travis Armstrong, and Pete Florence. 2023. Interactive language: Talking to robots in real time. *IEEE Robotics and Automation Letters*.
- Michael Matthews, Michael Beukman, Benjamin Ellis, Mikayel Samvelyan, Matthew Jackson, Samuel Coward, and Jakob Foerster. 2024. Craftax: A lightning-fast benchmark for open-ended reinforcement learning. In *International Conference on Machine Learning (ICML)*.
- Sindhu Padakandla. 2021. A survey of reinforcement learning algorithms for dynamically varying environments. *ACM Computing Surveys (CSUR)*, 54(6):1–25.
- Fabian Paischer, Thomas Adler, Markus Hofmarcher, and Sepp Hochreiter. 2023. Semantic helm: A human-readable memory for reinforcement learning. *Advances in Neural Information Processing Systems*, 36:9837–9865.
- Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. 2018. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.
- Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR.
- Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. 2019. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9339–9347.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749.
- Aleksandar Stanić, Yujin Tang, David Ha, and Jürgen Schmidhuber. 2023. Learning to generalize with object-centric agents in the open world survival game crafter. *IEEE Transactions on Games*.
- Alessandro Suglia, Qiaozi Gao, Jesse Thomason, Govind Thattai, and Gaurav S. Sukhatme. 2021. [Embodied bert: A transformer model for embodied, language-guided visual task completion.](#) *ArXiv*, abs/2108.04927.
- Alane Suhr, Claudia Yan, Charlotte Schluger, Stanley Yu, Hadi Khader, Marwa Mouallem, Iris Zhang, and Yoav Artzi. 2019. Executing instructions in

situated collaborative interactions. *arXiv preprint arXiv:1910.03655*.

Zoya Volovikova, Alexey Skrynnik, Petr Kuderov, and Aleksandr I Panov. 2024. Instruction following with goal-conditioned reinforcement learning in virtual environments. In *ECAI 2024*, pages 650–657. IOS Press.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. [Voyager: An open-ended embodied agent with large language models](#). *Transactions on Machine Learning Research*.

Lewei Yao, Jianhua Han, Youpeng Wen, Xiaodan Liang, Dan Xu, Wei Zhang, Zhenguo Li, Chunjing Xu, and Hang Xu. 2022. Detclip: Dictionary-enriched visual-concept paralleled pre-training for open-world detection. *Advances in Neural Information Processing Systems*, 35:9125–9138.

Hao Zhang, Feng Li, Shilong Liu, Lei Zhang, Hang Su, Jun Zhu, Lionel M Ni, and Heung-Yeung Shum. 2022. Dino: Detr with improved denoising anchor boxes for end-to-end object detection. *arXiv preprint arXiv:2203.03605*.

Victor Zhong, Tim Rocktäschel, and Edward Grefenstette. 2019. Rtfm: Generalising to novel environment dynamics via reading. *arXiv preprint arXiv:1910.08210*.

Appendix

A DATASET: Generation Prompt

This section presents the main GPT-4 prompt (see Figure 4) used in the instruction generation pipeline.

The code for checking played is following:

```
#Implementation of the scanner checker (1)
```

A scenario consists of instructions given by player 1 to player 2. Player 2 follows these instructions, which are then verified by a corresponding function. For the function `scenario.py`, please provide realistic examples of instructions that player 1 might give, along with 5 paraphrases for each.

Requirements:

- 1) When specifying target objects (objects with which the player will interact), use different synonyms in paraphrases to assess the vocabulary range of player 2.
- 2) Present the target objects in varying orders to evaluate how well player 2 understands different language structures.
- 3) For each set of paraphrases, sort them from the simplest language to the most complex.
- 4) Ensure the instructions are as varied as possible with a broad vocabulary.

Format your answer as a Python dictionary with the following structure:

```
instructions = {
  instruction_id: {
    'instruction': "Example instruction here",
    'instruction_paraphrases': [
      "Paraphrase 1 here",
      "Paraphrase 2 here",
      "Paraphrase 3 here",
      "Paraphrase 4 here",
      "Paraphrase 5 here"
    ],
    'check_lambda': \
      lambda ...:scenario_function(...): ...
  }
}
```

Figure 4: The prompt used for instruction generation in Crafter. The label (1) indicates the actual implementation of the scenario verification function.

B DATASET: Tasks Description

The dataset consists of a set of instructions of different types, examples of which are provided in Figure 10.

Conditional. In the *Conditional* category, the agent is required to understand the sequence of

actions it needs to perform and the order in which these actions should be carried out.

Example

Instruction: "After collecting coal, the player should gather wood and then place a stone on the crafting table."

Figure 5: Sequencing Instruction Example

The focus here is on understanding temporal relationships between actions (see an example at Figure 5).

Build. The *Building* category involves tasks where the agent must construct specific shapes or structures based on verbal instructions.

Example

Instruction: "Arrange the tables in a square pattern using 9 stone blocks."

Figure 6: Building Instruction Example

This category tests the agent's spatial reasoning and ability to translate instructions into precise constructions (see an Example at Figure 6).

Localization. In the *Localization* category, the agent must determine from which side it should position an object and in relation to which reference object.

Example

Instruction: "Place the table two steps to the left of the lake."

Figure 7: Localization Instruction Example

In this example, the agent needs to: 1) Identify the lake 2) Determine the left side 3) Understand relative positioning Additionally, these tasks may involve directional terms such as left, right, above, below, north, south, west, and east.

This category is particularly challenging as it requires the agent to integrate directional instructions with map-based navigation (see an example at Figure 7).

Achievements Tasks. The *Achievements* category contains tasks that require the agent to complete in-game achievements. It also includes tasks that instruct the agent to perform actions involving combinations of achievements (Figure 8).

Examples

Instruction: "Forge a sturdy pickaxe from stone."
Instruction: "Craft a wooden pickaxe but avoid making a stone pickaxe."
Instruction: "Craft a stone sword and eliminate all undead."

Figure 8: Achievements Tasks

This type of task evaluates the agent’s ability to understand achievements by interpreting tasks and identifying the necessary actions. It also tests whether the agent can combine multiple achievements in sequence or simultaneously. Additionally, it assesses the agent’s ability to follow constraints, ensuring specific conditions are met while completing a task.

C DATASET: Instructions Examples

Train:

task

"Place a stone one block to the left of the furnace."

instructions

- "Position a rock a unit to the west of the kiln."
- "To the immediate left of the heating station, deposit a piece of rock."
- "Beside the furnace, specifically to its left, position a stone block."
- "Should you find yourself in the vicinity of the furnace, it is your duty to move a stone one block to its left."
- "In the relentless dance of stone and fire, a single stone must find itself moved a measure to the left of the ever-watching furnace, a silent testament to the delicate

GOAL:

place STONE,
side LEFT,
dist 1 block,
relevant FURNACE

Test Paraphrase:

instructions

- "Put a stone block just left of the smelter."
- "In relation to the foundry, ensure there’s a stone one square to the left of it"
- "One space to the left of the apparatus for smelting ore, situate a piece of stone."
- "In spite of its weight and hardness, your task is to adjust a unit to the left, a rock bearing the vicinity of the kiln."
- "In the shadowy realm of the searing furnace, thy charge is to take a stone, sturdy and unyielding, and relocate it a block to the west, under the vigilant watch of the ever-hungry kiln."

Test New Objects:

task

"Place the furnace four blocks above the stone"

instructions

- "Position the kiln four spaces top of the rock"
- "Arrange the smelter four blocks upward from the stone"
- "Set the forge four spots to the north of the boulder"
- "Situate the blast furnace four notches above the stony block"
- "Install the heating chamber at a distance of four blocks to the top from the solid pebble"

Unseen combinations

GOAL:

place FURNACE,
side TOP,
dist 4 block,
relevant STONE

Figure 9: Examples of instructions in dataset.

D DATASET: Per Category Scenarios, Goal, Instructions

In our dataset, there is a division into the training set, the test set with rephrased instructions (*Test Paraphrased*), and the test set with new tasks (*Test*

New Object). For example, in the localization scenario, where blocks need to be placed relative to other blocks at a certain distance, if the training set includes a task to place a stone block to the left of a furnace, then in *Test Paraphrased*, it may be rephrased as "find the furnace and place a stone block one block north of it." In *Test New Object*, a new combination of parameters appears, such as a task to place a furnace four blocks above a stone. The complexity of *Test New Object* lies in the fact that the agent has multiple ways to approach the task. It can either find existing stone blocks and place the furnace relative to them or place both the stone and the furnace itself, as in the training task. This adds variability to the problem and requires the agent to be flexible in decision-making. See Figure 9 for more examples.

E DATASET: Assessing the Scalability

Table 3 summarizes possible and implemented goals across different scenario classes and complexities. It lists the class, number of arguments (Args), possible goals, and implemented goals. Each instruction’s complexity level (Easy, Medium, Hard) is also indicated. The dataset defines 3,480 possible goals, which expand to over 20,000 instructions when factoring in paraphrases.

Despite having 476 distinct goals, the dataset’s scalability stems from a function-based task generation process. This approach systematically combines predefined goals, environmental parameters, and paraphrased instructions, creating diverse training interactions for instruction-following agents.

A key example is the `place` function, which checks whether an object is positioned relative to a target at a specified side (right, left, top, bottom) and distance. With four object types and four distances, this results in 512 unique goals. Each goal is paraphrased in six ways, producing 3,072 instructions from this function alone.

This scalability extends to other task-generating functions, enabling continuous dataset expansion. Appendix E details how this methodology scales to 3,480 unique goals and approximately 20,880 instructions with paraphrases.

F ENVIRONMENT: Performance

Efficient simulation speed is a cornerstone of reinforcement learning (RL) environments, enabling faster training cycles and broader experimentation. CraftText, built on top of Craftax, leverages the



Figure 10: Example instruction set for different tasks.

Name	Class Name	Args	Possible Goals	Implemented Goals	Complexity
achievements	achievements	list	500	164	EASY
build_line	build	3	75	57	MEDIUM
square	build	2	25	28	MEDIUM
cross	build	3	75	10	HARD
localization_place	localization	3	120	91	MEDIUM
place_item_after_collection	conditional	3	80	55	MEDIUM
collect_item_after_place	conditional	3	80	56	MEDIUM
build_line_and_localization_place	combo	4	25 x 25	5	HARD
build_line_after_collect	combo	4	25 x 25	5	HARD
collect_item_and_build_square	combo	4	25 x 25	5	HARD
Total			2830	476	

Table 3: Summary of Possible Goals and Implemented Goals, with Complexity.

computational power of JAX and advanced parallel execution to demonstrate remarkable scalability. As the number of parallel environments grows, CraText’s steps per second (SPS) increase proportionally, unlocking the potential for large-scale RL research and development.

To evaluate performance, we measured the environment’s speed over **5000 steps** using an agent with **random behavior**. The experiments compared SPS across different base Craftax environments and hardware configurations, including devices such as **Tesla V100 GPUs**. On Classic Craftax, training on 1024 environments achieved approximately 44,630 SPS, while on CraText, the same configuration yielded around 9,100 SPS on a Tesla V100.

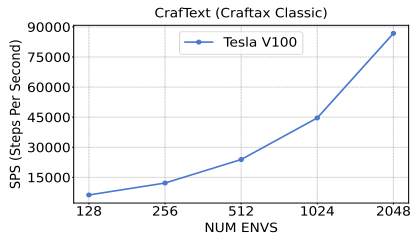


Figure 11: SPS of CraText environment based on Craftax Classic environment

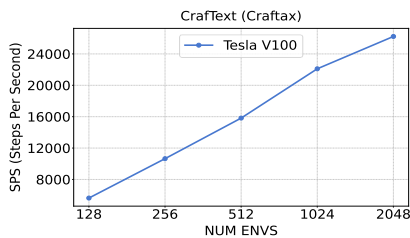


Figure 12: SPS of CraText environment based on Craftax environment

G Craftax

Craftax is a benchmark designed to facilitate research in open-ended RL. It builds upon the mechanics of the Crafter (Hafner, 2022) environment, providing a complex, procedurally generated world where agents must engage in deep exploration, long-term planning, and memory utilization while adapting to novel situations. The primary reward function in Craftax consists of achieving specific tasks that are grouped into four categories: ‘Basic’ (1 reward), ‘Intermediate’ (3 rewards), ‘Advanced’ (5 rewards), and ‘Very Advanced’ (8 rewards), enhancing the incentive structure for exploration and

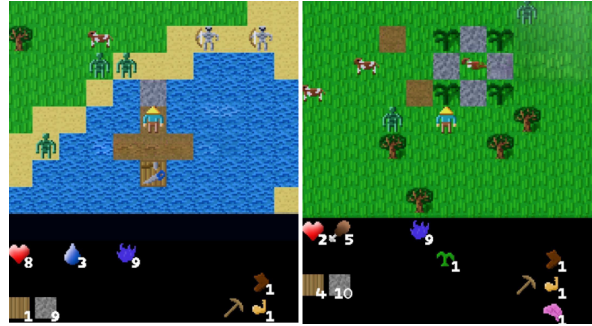


Figure 13: Examples of the visual observation input in CraText, showcasing different perspectives of the environment provided to the agent for decision-making.

engagement. The observational input for agents is represented in two formats: a pixel-based input of size $110 \times 130 \times 3$ and a symbolic representation with 512 dimension. The action space consists of 43 discrete actions, allowing agents to perform various interactions and movements within the environment. To succeed, agents must navigate diverse terrains, face 19 distinct creatures, and utilize various combat and crafting mechanics across nine unique procedurally generated floors.

CraText VS Craftax. CraText builds upon Craftax by introducing instruction-based tasks, which require more than just a dataset of textual instructions. Each task is paired with a checker function that validates task execution at every environment timestep, providing continuous monitoring of the agent’s progress. This per-timestep validation is critical for ensuring correct task completion in dynamic environments. The checker signals the termination of an episode when a task is successfully solved and is tightly integrated with a task-based reward function, enabling immediate feedback essential for reinforcement learning.

To maintain the performance benefits of the original Craftax, CraText implements these checker functions with XLA acceleration. This ensures that per-timestep validation and reward computation are computationally efficient, even in large-scale multimodal RL settings. Furthermore, by decoupling the instruction-checker dataset from the Craftax environment’s core implementation, CraText supports seamless extensibility, enabling researchers to easily expand the benchmark with new tasks using the provided toolset.

CraText is more than an environment—it is a comprehensive benchmark designed to evaluate natural language understanding and other capabilities in multimodal settings. Its evaluation proto-



Figure 14: Visualizations of multimodal environments.

col supports testing approaches using a hold-out dataset, which, while a correct methodology, remains underrepresented in the RL community. The protocol categorizes tasks into Conditional, Building, Localization, and Combination, each with varying difficulty levels (Easy, Medium, Hard). It also includes two state-of-the-art baselines, Dynalang and a large-scale PPO, both specifically adapted to the environment. The PPO baseline was trained for 1 billion environment steps in 12 hours on a single GPU, achieving over 20k steps per second (SPS), showcasing the benchmark’s scalability and accessibility to a broad range of researchers.

Finally, we emphasize that building upon prior research is central to scientific progress. Craftax itself was developed on top of Crafter and NetHack. Similarly, CraftText extends Craftax by introducing a benchmark for goal-driven natural language tasks in dynamic visual environments, supporting scalable task expansion and pioneering the use of XLA-accelerated multimodal benchmarks.

H Instruction Following Environments

RTFM is an environment where the agent must understand and apply procedural game instructions to navigate a dungeon. The agent reads procedurally generated manuals and uses the acquired knowledge to defeat enemies and interact with the game world. A key feature is the need for text compre-

hension in a dynamic setting.

Messenger presents a message-delivery challenge where the agent must navigate while avoiding obstacles and interacting with NPCs. The task requires choosing optimal routes and adapting to a dynamically changing environment, adding a layer of strategic complexity.

Baby-AI features a grid-based world where the agent follows simple language instructions. Tasks involve moving to specific locations and interacting with objects, requiring basic language comprehension and action planning in a discrete space.

Home-Grid simulates a household environment where the agent performs daily tasks. It moves through rooms, manipulates objects, and interacts with the surroundings based on given instructions. This environment models real-world scenarios with an emphasis on object management and action sequencing.

Touchdown is a city navigation simulation where the agent follows complex natural language instructions to traverse an urban environment. The setting consists of realistic 3D scenes, and successful task execution demands understanding spatial directions and adapting to detailed route descriptions.

IGLU focuses on spatial reasoning, requiring the agent to construct 3D structures by following verbal or textual instructions. The environment emphasizes precise execution of building tasks, making

spatial understanding and object placement crucial.

Craft-Assist immerses the agent in a Minecraft-like world where it collaborates with humans to complete construction tasks. The agent interprets commands, interacts with the environment, and participates in cooperative work, posing challenges in language comprehension and behavior modeling.

Mine-Dojo provides an open-ended Minecraft world where the agent undertakes diverse tasks such as resource gathering, crafting, exploration, and combat. This environment requires complex planning and engagement with various mechanics, making it one of the most versatile interactive settings.

CEREALBAR is a collaborative instruction-following environment where a leader gives natural language instructions to a follower in a 3D game setting. The agent must accurately interpret and execute instructions to collect valid sets of cards. This setup emphasizes multi-agent coordination, real-time decision-making, and natural language understanding.

ALFRED places the agent in a photorealistic kitchen setting, where it executes household tasks such as cooking, cleaning, and organizing objects. The environment models complex object interactions and demands sequential execution of instructions while accounting for physical constraints.

I TRAINING: Curve Analysis

We ran each baseline model—Dynalang, PPO+BertEmb (PPO-T), and PPO+BertEmb+GPT4Plan (PPO-T+)—on the Medium portion of our dataset, training each model for 60 hours on an Tesla V100 GPU. Figure 15 shows the result Success Rate, with checkpoints recorded every 12 hours on both the training and test sets. Overall, we observed similar performance with and without GPT-4 plans during training (overall 60h \pm 0.47). In contrast, Dynalang produced much weaker results (0.15), possibly because it requires additional training time due to its world model training process.

On the test set, the model incorporating GPT-4 plans achieved the best performance for every task type (overall 0.28, compared to PPO+BertEmb at 0.23 and Dynalang at 0.1). We attribute this improvement to the capability of large models like GPT-4 to simplify complex instructions. Moreover, on both the training and test sets, using GPT-4 plans yielded higher performance on conditional tasks

(Training: 0.18 vs. 0.16 without plans; Test: 0.21 vs. 0.17 without plans). These results highlight that incorporating additional planning can help the agent solve puzzles and logical tasks that require understanding conditions expressed in text.

J TRAINING: Prompt for GPT-4 plan generation

Craftax is a virtual environment designed for exploration, crafting, and task completion. The procedurally generated world includes resources (trees, stones, coal, iron), interactive objects (crafting tables, furnaces, chests), and diverse terrains (water, grass, sand). The agent operates in this dynamic environment, performing actions such as moving, collecting resources, crafting, placing objects, and interacting with surroundings. Completing tasks often involves gathering resources, crafting items, and strategically placing objects.

The agent has a fixed set of discrete actions:

- **Movement:** Navigate the map (up, down, left, right).
- **Resource Collection:** Chop trees, mine stones, or gather coal.
- **Crafting:** Create tools (e.g., pickaxes) or structures (e.g., furnaces).
- **Object Interaction:** Use objects (e.g., cook food in a furnace).
- **Placement:** Place crafted objects in specific locations.

Task: Create a step-by-step action plan (maximum 5 steps) for the agent in Craftax to achieve the following instruction:

“{instruction}”

Response Format:

1. Using only object names existing in the Craftax environment, provide the plan as a numbered list.
2. Each step should outline a specific action or logical task for the agent, such as resource collection, crafting, or object placement.
3. Keep steps clear, concise, and implementable in Craftax, with a maximum of 5 words per step.

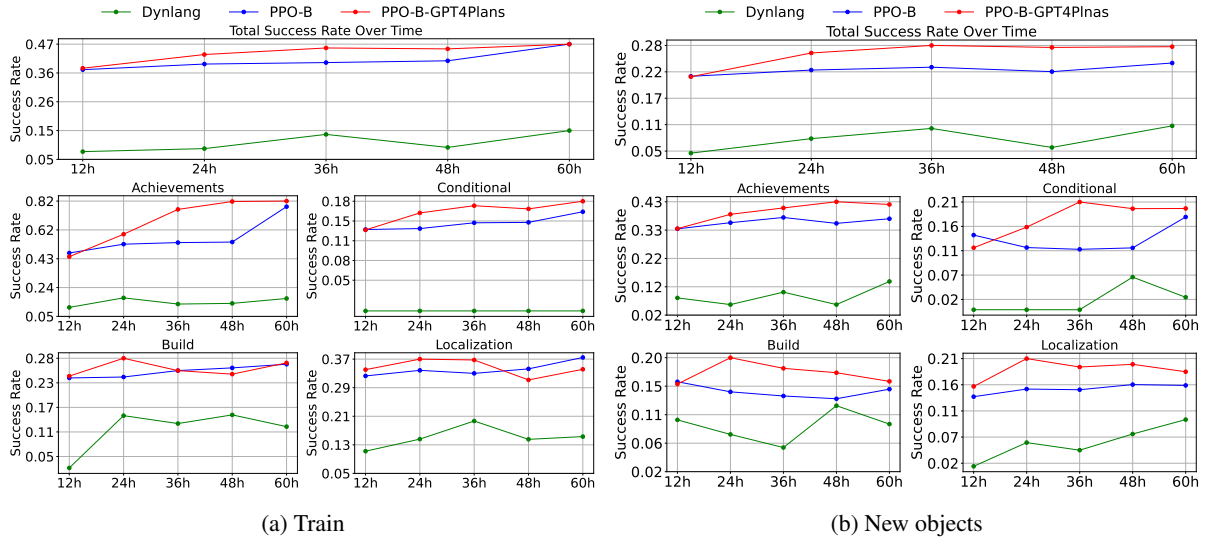


Figure 15: The training performance and corresponding validation curves on the test set (other parameters) for the three baselines: PPO+BertEmb, PPO+BertEmb+GPT4Plans, and Dynalang.

Check yourself!

!ATTENTION! Ensure the plan uses only object names and actions existing in Craftax. Replace any incorrect terms in the instruction with their correct Craftax equivalents.

K TRAINING: Analysis of Failure Cases

We analyze agent failures by evaluating the Success Rate for each instruction in our dataset and then aggregating these results based on key features: the types of objects the agent must interact with, specific task parameters (e.g., the length/width of items to be placed or the shape of the constructed structure), and the level of instruction complexity. By examining difficulties in interacting with various game elements, constructing complex shapes, and adapting to paraphrased instructions, we identify patterns of failure and determine where improvements can be made to enhance the agent’s task comprehension and execution.

The Figure 16 presents the aggregated SR for tasks involving different in-game elements. The highest SR is achieved for tasks requiring interaction with TREE (SR = 0.9), which is not surprising given that trees are abundant and do not demand specialized skills. In contrast, the lowest SR values occur for WATER (SR = 0.03), IRON (SR = 0.02), COAL (SR = 0.01), and PLANT (SR = 0.00). These more challenging tasks involve searching for specific objects and making use of appropriate inventory tools.

The agent struggles with building shapes larger

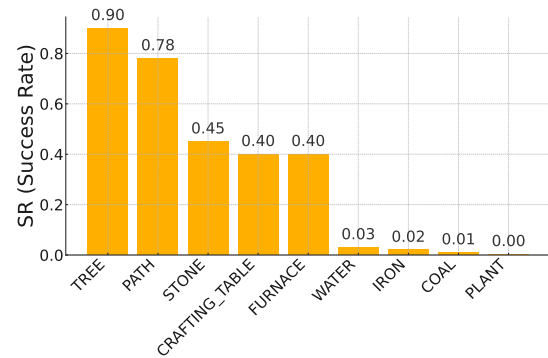


Figure 16: Aggregated SR for tasks involving different game elements, highlighting varying levels of interaction difficulty.

than two blocks, making such configurations rare and rewards infrequent (see Tables 4 and 5). Although it can place two blocks in a line effectively (e.g., SR=0.78), performance drops sharply with three-block lines (SR=0.2) or diagonals (SR=0.07). Constructing squares is especially difficult (highest SR=0.09).

Block Type	Length	Diagonal	SR
STONE	4	False	0.81
CRAFTING_TABLE	2	False	0.78
FURNACE	2	False	0.72
STONE	2	True	0.67
CRAFTING_TABLE	3	False	0.20
CRAFTING_TABLE	3	True	0.07

Table 4: Success rate averaged by goal (Line and Diagonal Construction)

Block Type	Side Size	Average_SR
STONE	2	0.09
STONE	3	0.004
CRAFTING_TABLE	2	0.00
FURNACE	2	0.00

Table 5: SR Averaged by Goal (Square Construction)

Finally, tasks using paraphrased instructions (e.g., synonyms instead of direct object names) present additional challenges for the agent. The average SR for tasks without paraphrases is 0.37, but it drops to 0.28 when paraphrased instructions are used. This underscores the difficulty of generalization in understanding varied instructions.

Greatest Challenges in the CraftText Environment

1. Environmental Features

Dynamic Instruction Following Task Setup.

An agent needs to interpret the same instruction in changing visual and spatial contexts. *Experimental results (Table 2)*: Despite extensive training, agent performance remains suboptimal even on the training set. This plateau highlights the difficulty of adapting to the dynamic and stochastic nature of the environment. Success rates (SR) on the training set: PPO-T = 0.4, PPO-T+ = 0.45, Dynalang = 0.15, FiLM = 0.43.

Multi-Step Tasks with Implicit Preconditions.

Many tasks require multi-step action sequences where each subsequent step depends on successful completion of earlier ones. These may involve collecting resources, crafting intermediate items, and combining subgoals into a coherent plan. *Experimental results (Appendix K)*: (1) Tasks involving complex objects like IRON, COAL, and PLANT have very low SRs: 0.02, 0.01, and 0 respectively. (2) Even for simpler objects like the CRAFTING_TABLE, increasing the line size from 2 to 3 blocks reduces SR from 0.78 to 0.2.

2. Linguistic Features

Linguistic Variation and Paraphrasing.

Each goal is expressed through multiple paraphrases that differ lexically and syntactically. The agent must understand diverse formulations, recognize equivalence, and map them to the same behavior (See the Example in Table 6).

Experimental results (Table 2): Performance drops significantly on paraphrased instructions: PPO-T: 0.4 \rightarrow 0.36, PPO-T+: 0.45 \rightarrow 0.35, Dy-

nalang: 0.15 \rightarrow 0.05, FiLM: 0.43 \rightarrow 0.35.

Examples: Goal = Make a line with size 2 from Crafting Table

Instruction	SR
Make a line of 2 blocks using table	0.85
Construct a row of 2 pieces with the crafting station	0.46
Arrange a sequence of 2 blocks with the crafting platform	0.33

Table 6: The visualization of how language formulation affects RL agent behavior.

3. Generalization to Novel Combinations

Test New Objects Split.

Includes instructions with new combinations of objects, spatial relations, and parameters. The agent must generalize beyond memorized templates and recombine known elements in novel ways. *Experimental results (Table 2)*: All baselines show strong performance drops: PPO-T: 0.4 \rightarrow 0.22, PPO-T+: 0.45 \rightarrow 0.28, Dynalang: 0.15 \rightarrow 0.10, FiLM: 0.43 \rightarrow 0.26.

L TRAINING: Hyperparameters

The training of our models relies on carefully selected hyperparameters for both the language processing module and the reinforcement learning agent. The Dynalang model utilizes a deep language MLP with 5 layers of 1024 units each, a GRU with 4096 recurrent units, and a total parameter count of 281 million, optimized using Adam with a learning rate of 1×10^{-4} and SiLU activations.

For the RL component in PPO-T, PPO-T+, and FiLM, we employ PPO with 1024 parallel environments and a training horizon of 1 billion timesteps. PPO training uses a learning rate of 0.0002, a discount factor of 0.99, GAE lambda of 0.8, and 4 update epochs per batch, with 8 minibatches per update. We apply clipping with $\epsilon = 0.2$, an entropy coefficient of 0.01 to encourage exploration, and a value function coefficient of 0.5 for stabilizing value estimates. The agent network uses \tanh activations with a hidden layer size of 512 units. Additionally, we enable optimistic resets with a reset ratio of 16 to improve exploration efficiency. Instructional inputs are processed using a Distil-

Model	Trained	Achievements (one)	Achievements (combo)	New objects
DeepSeek	No	0.15	0.07	0.06
MISTRAL	No	0.21	0.10	0.09
Qwen	No	0.21	0.10	0.07
PPO-T	Yes	0.78	0.55	0.34
PPO-T+	Yes	0.87	0.70	0.43
Dynalang	Yes	–	0.17	0.14
FiLM	Yes	0.85	0.76	0.38

Table 7: Comparison of models on the EASY part of the CraFText dataset: Achievements (one) involve solving single atomic tasks, Achievements (combo) require completing multiple achievements together, and New objects measure performance on unseen combinations of achievements.

BERT encoder, with right-padding and truncation applied during tokenization.

Parameter Name	Default Value
Language MLP Layers	5
Language MLP Units	1024
Batch Size	16
Batch Length	256
Train Ratio	32
GRU Recurrent Units	4096
Total Model Parameters	281M
Optimizer	Adam
Learning Rate	1e-4
Epsilon	1e-8
Activation	SiLU

Table 8: Dynalang Training Hyperparameters

Parameter Name	Default Value
Number of Environments	1024
Total Training Timesteps	1 000,000,000
Learning Rate	0.0002
Steps per Environment	100
Update Epochs	4
Minibatches per Update	8
Discount Factor	0.99
GAE Lambda	0.8
Clipping Epsilon	0.2
Entropy Coefficient	0.01
Value Function Coefficient	0.5
Max Gradient Norm	1.0
Activation Function	tanh
Anneal Learning Rate	True
Layer Size	512
Optimistic Resets Enabled	True
Optimistic Reset Ratio	16
Exploration Epochs	4
Instruction Encoder Model	DistilBERT
Encoder Tokenizer Padding	right
Encoder Tokenizer Truncation	right

Table 9: PPO Training Hyperparameters

M TRAINING: Easy-dataset experiments

We introduce three zero-shot baselines for our Easy set, using popular LLMs to predict actions based on instructions and structured descriptions of the current agent observation, including objects and mobs coordinates. The models used are Qwen/QWQ-32B (Qwen), DeepSeek-R1-Distill-Qwen-32B (DeepSeek), and Mistral-Small-24B-Instruct-2501 (MISTRAL).

We evaluate them alongside trainable baselines (PPO-T, PPO-T+, Dynalang, FiLM) on two categories within the MEDIUM set: (1) *Achievement (one)* — single-step goals (e.g., “Make a crafting table”); (2) *Achievement (combo)* — multi-step goals involving several achievements (e.g., “Make a crafting table and craft a stone sword”).

We also report results on *New objects*, which contains instructions requiring the agent to com-

plete unseen combinations of achievements within a single episode.

Zero-shot LLMs show basic environment understanding but struggle under dynamic conditions.

In the zero-shot setting, language models achieve a score of 0.21 on *Achievements (one)*, suggesting they capture some aspects of the environment’s mechanics and object relationships. Their performance on *Achievements (combo)* is lower (0.10), indicating limited ability to handle composite instructions requiring long-horizon decision-making. Overall performance remains low, emphasizing the limitations of LLMs in decision-making under dynamic and partially observable conditions.

Generalization on unseen tasks.

The evaluation on *New objects* is the only part

of the dataset where it is valid to compare trained agents and language models, as both encounter these combinations for the first time. Here, trained agents such as PPO-T+ (0.43) and FiLM (0.38) significantly outperform zero-shot models, whose scores remain in the range 0.06–0.09. At the same time, we observe that trained RL models experience a notable drop in performance compared to the training set — almost a twofold decrease. In contrast, LLMs, as expected, exhibit only minor changes in performance.

Additionally, the performance of the evaluated LLMs could be significantly enhanced with better prompt tuning and a richer representation of the agent’s input using environment state. Prior work, such as Voyager (Wang et al., 2024), has demonstrated how structured prompting can improve LLM-based agents in game-like environments. However, our study does not aim to optimize LLM prompting for this domain but rather to establish baseline zero-shot performance.