CraftText Benchmark: Advancing Language Grounding in Complex Multimodal Open-Ended World

Anonymous ACL submission

Abstract

Grounding language models in multimodal environments is a pivotal challenge in AI, enabling agents to link linguistic inputs with sensory data, such as visual information. Existing environments, however, often limit the complexity of agent behavior due to restricted dynamics or vocabulary. To address these limitations, we propose a new benchmark based on the Craftax environment—a dynamic, stochastic setting with extensive game mechanics and a rich vocabulary. This benchmark is designed to evaluate agents on complex tasks involving spatial reasoning, logic, and context, offering a rigorous platform for advancing multimodal AI research.

1 Introduction

Grounding language models is one of the key challenges in the field of artificial intelligence, aimed at enabling intelligent agents to link linguistic entities with objects from various modalities, such as visual data (Carion et al., 2020; Li et al., 2022; Radford et al., 2021b). Solving this problem opens up opportunities for creating more versatile and adaptive AI systems capable of effectively extracting and integrating features from different types of data (Zhang et al., 2022; Wang et al., 2023). This, in turn, allows them to make more complex inferences and build comprehensive models applicable in a wide range of contexts.

Examples of the use of such technologies include generating textual descriptions based on images, enhancing natural language processing with visual cues, and multimodal search, where text queries are matched with images or videos. These examples highlight the importance of integrating data from different sources to achieve more accurate and sophisticated results.

Particular attention in this area is given to developing behavioral strategies for agents trained with reinforcement learning in multimodal environments (Perez et al., 2018; Lynch et al., 2022; Wang and Narasimhan, 2021). In this context, an agent must be able to formulate low-level action plans based on textual instructions and visual observations. The agent's task is not only to associate words with objects in the environment but also to adapt its behavior depending on the textual directions received (Zhong et al., 2019).

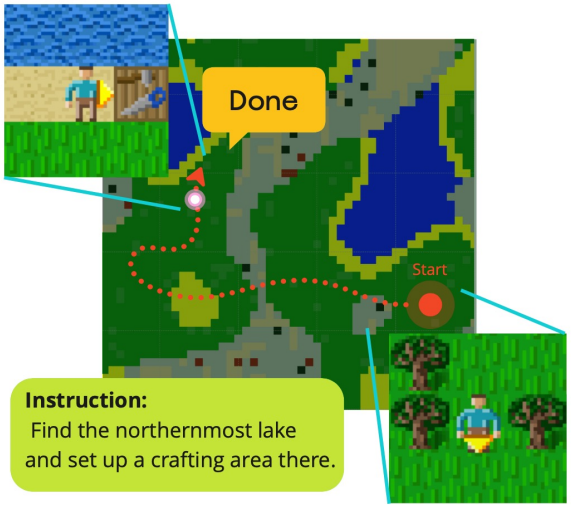

Figure 1: Example of an agent executing a task in the Craftax environment, moving from the starting point to the northernmost lake to set up a crafting table. A 'Done' marker confirms the task's successful completion.

Examples of tasks that can be assigned to such agents vary depending on the environment's characteristics. For instance, in one environment, the agent may need to find a target point by following textual instructions. In more complex scenarios, the instructions might include guidance on selecting specific actions to achieve goals, as well as considering the context of object interactions within the environment. In such cases, the agent must understand which objects can be used together to achieve a particular outcome or how specific actions might alter the state of the environment.

However, existing multimodal environments often have limitations that affect the complexity of

behavior expected from the agent: some environments may focus more on text comprehension and have limited environmental dynamics, while others, with more complex mechanics, may have a restricted vocabulary.

To address these challenges, we propose a new benchmark CrafText based on the Craftax environment—a stochastic and dynamic environment with a large number of game mechanics (Figure 1). Our extension presents a set of complex linguistic instructions with a rich vocabulary that requires solving spatial, logical, and other tasks formulated in natural language, along with a set of evaluation functions that objectively assess how successfully the agent handles the given instructions. The dataset and the code for our extension can be accessed in the following repository.[1]

2 Related work

Language Grounding Problem. Addressing the challenge of language grounding in intelligent agents involves enabling these agents to associate objects across different modalities, such as linking observations in virtual environments with textual instructions. The core task is for an agent to interpret instructions related to interacting with objects within an environment, accurately match the descriptions to the corresponding objects present, and perform the appropriate actions. In multimodal environments, existing approaches to policy creation can be categorized into three main groups. Some studies utilize CLIP (Radford et al., 2021a) to connect visual and textual observations (Paischer et al., 2024; Lynch et al., 2022). Another group of research employs projection layers (Perez et al., 2018; Zhong et al., 2019; Wang and Narasimhan, 2021). In contrast, others leverage cross-attention mechanisms or methods that compress data into a hidden subspace, where text is not directly associated with visual observations, as seen in Dynalang (Lin et al., 2023).

Embodied Environments. Multimodal environments used for training agents are characterized by their diversity, making them valuable for solving a wide range of tasks while also revealing certain limitations. Environments such as Touchdown (Zhong et al., 2021), Alfred (Shridhar et al., 2020), Housekeep (Kant et al., 2022), and VirtualHome (Savva et al., 2019) provide rich visual content, numerous objects for interaction, and complex textual instructions with extensive vocabularies. Although these environments may lack dynamism, they present agents with challenging tasks that require a high level of interpretation and planning. However, agents operating within these environments often face limitations in developing low-level strategies, which may diminish their flexibility and adaptability. Environments like HomeGrid (Lin et al., 2023), BabyAI, RTFM (Zhong et al., 2019), and Messenger (Wang and Narasimhan, 2021) stand out for their engaging gameplay mechanics, despite having less developed visual components and fewer interactive objects. In these settings, agents are compelled to adapt quickly and make decisions under constrained resources. For example, HomeGrid includes dynamic elements, while BabyAI presents tasks that require puzzle-solving and strategic thinking. In RTFM and Messenger, the emphasis is on textual interaction and the identification of objects or allies. The IGLU (Kiseleva et al., 2022) environment occupies a unique position by combining a variety of features, offering agents tasks that involve building structures from blocks based on textual descriptions. This environment is notable for its high combinatorial complexity and stochasticity, making it particularly interesting for research despite its lack of dynamism. There is a notable shortage of environments that can effectively assess an agent's ability to perceive diverse language and associate it with real-time events in the environment. While many environments emphasize either the visual or textual component, few achieve a well-balanced integration of both.

3 Craftext

In our work, we developed an extension for the open-world environment Craftax (Matthews et al., 2024), which we named **CrafText**. Craftax is a game environment that provides a wide range of tools for creating and exploring virtual worlds. This environment is characterized by high dynamism and stochasticity, making it ideal for creating complex game mechanics and interactions. Craftax includes numerous game objects, allowing agents to interact with the virtual space, which can change in real-time.

Our goal was to create a goal-oriented version of this environment, where the agent receives tasks through natural language text instructions. This ap-

[1] CrafText Repository: https://anonymous.4open.science/r/CrafText-4FA2/

proach not only allows for testing the agent's ability to perform actions in the virtual environment but also for evaluating its understanding of various aspects of natural language. To achieve this goal, we developed 700 unique game tasks, including spatial reasoning tasks, logic puzzles, and tasks related to object construction. A detailed description of the developed dataset is provided in Chapter 3.1.

Some of the game instructions were generated using modern language models. We pre-defined a variety of game scenarios that the agent was required to complete, and then used language models to create different versions of descriptions for each scenario. A detailed description of the process of generating scenarios and instructions is provided in Chapter 3.3.

Additionally, for each scenario and game instruction, we developed code that allows for the automatic verification of task completion by the agent by analyzing the current state of the game environment. A description of the methods and approaches for task verification can be found in Chapter 3.2.

3.1 Dataset Overview and Structure

The CrafText Dataset (Figure 2) is designed to provide a comprehensive testing ground for intelligent agents, challenging them to understand and execute a variety of instructions. At its core, the dataset is organized around several distinct task categories, each of which presents unique challenges to the agent. These categories are *Sequencing*, *Building*, and *Localization*, with an additional *Combination* category that requires the agent to integrate multiple types of instructions.

The dataset features a rich and diverse vocabulary of 756 unique words, spread across different categories. This extensive vocabulary is crucial for testing the agent's ability to understand context, recognize paraphrased instructions, and handle synonyms. The large vocabulary ensures that the agent is not simply memorizing commands but is truly understanding and interpreting the instructions provided.

The data is split into training and test sets, with 510 training examples and 90 test examples. This split allows for rigorous testing of the agent's performance on unseen instructions, ensuring that the agent's capabilities are evaluated in a comprehensive manner. The mean length of the instructions is 16 words, with the longest instructions extending to 89 words. This variability in instruction length further tests the agent's ability to handle both simple and complex tasks.

Sequencing

In the *Sequencing* 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. This category has a vocabulary of 207 unique words and includes 107 training examples and 15 test examples.

Example

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

Figure 3: Sequencing Instruction Example

The focus here is on understanding temporal relationships between actions (see an example at Figure 3). The agent must recognize words like "after" to correctly sequence the actions. The common vocabulary includes terms like "player," "after," "coal," "place," and "stone," which are integral to these sequences.

Building

The *Building* category involves tasks where the agent must construct specific shapes or structures based on verbal instructions. This category has a vocabulary of 286 unique words, with 89 training examples and 31 test examples.

Example

Instruction: "Arrange the tables in a square pattern around the crafting area, and place chests in the four corners."

Figure 4: Building Instruction Example

This category tests the agent's spatial reasoning and ability to translate instructions into precise constructions (see an Example at Figure 4). The common vocabulary includes words like "crafting," "square," "tables," "arrange," and "chests," which are crucial for building tasks.

Localization

In the *Cardinal Direction* category, the agent must navigate through a map and complete tasks based on specific directions, such as moving north or south. This category has a vocabulary of 299 unique words, with 129 training examples and 21 test examples.

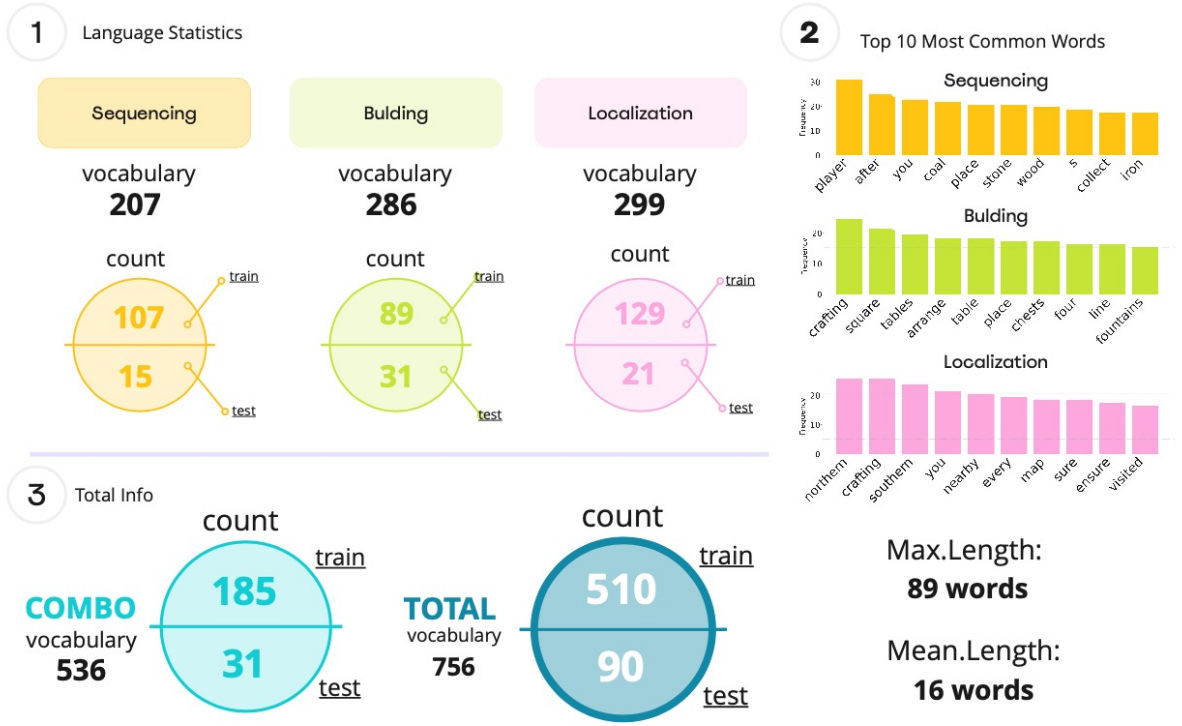

Figure 2: The image presents an infographic of the CrafText dataset, organized into three main sections. **Section 1** provides language statistics, showing vocabulary size and word counts for Sequencing, Building, and Localization, divided between training and testing datasets. **Section 2** features bar charts displaying the top 10 most common words in each category. **Section 3** summarizes the combined vocabulary data, including total word counts, and details on maximum and mean word lengths.

Example

Instruction: "Head north until you reach the southern crafting station, then turn east and ensure you visit every map location."

Figure 5: Localization Instruction Example

Example

Instruction: "First, gather materials from the north, then construct a table near the crafting area, and finally, place it in the square pattern with other tables."

Figure 6: Instructions Combination.

This category is particularly challenging as it requires the agent to integrate directional instructions with map-based navigation (see an example at Figure 5. The common vocabulary includes words like "northern," "southern," "map," and "ensure," which are key to completing these tasks.

Combination Category

The *Combination* category introduces even greater complexity by mixing instructions from the other three categories. This category challenges the agent to handle multi-step, multi-dimensional tasks, often requiring it to sequence actions, construct structures, and navigate directions all within the same instruction set. This category tests the agent's holistic understanding and its ability to integrate different types of reasoning and planning (Figure 6).

3.2 Scenarios checkers

As previously outlined, our extension is composed of a set of instructions and functions designed to validate the completeness of these instructions within the Craftax environment based on the current game state. To develop this dataset, we begin by initializing foundational scenarios for each instruction group: Building, Localization, and Sequencing. A scenario represents the expected pattern of agent behavior when they adhere to the given textual instructions. We have developed approximately 20 scenarios, each of which can be described in natural language with varying terminology and parameters, such as the objects the agent must interact with or the specific manner in which they must complete the task. Each scenario is validated by a specific check function that is tailored

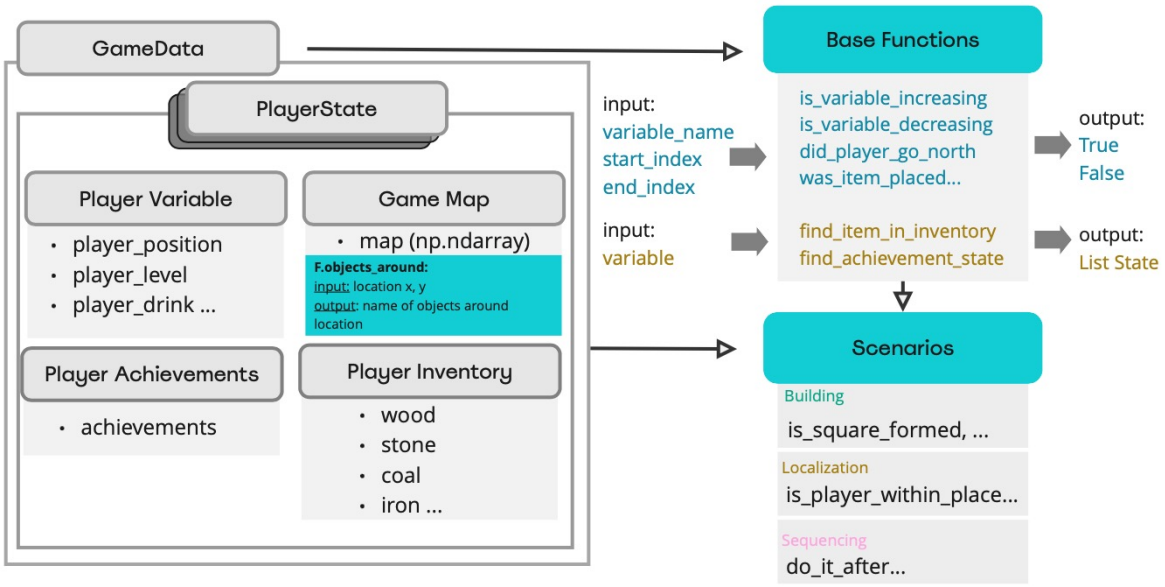

Figure 7: This diagram represents the structure of scenario checks within the Craftax environment. It demonstrates how variables such as player state, inventory, achievements, and position on the map interact with core functions to verify the execution of various scenarios.

with appropriate parameters.

The diagram 7 presents the structure of the game mechanics in Craftax, demonstrating how the agent's data and basic verification functions interact to create various game scenarios. At the core of the system is the **PlayerState** block, which is responsible for storing all data related to the player's state. This class is implemented based on the game state within the Craftax environment. The state includes variables such as the player's position, level, direction of movement, as well as food and water consumption levels. All these variables reflect the current situation and the player's status in the game.

The **PlayerState** also includes the **Game Map**, which represents the game world map where the player interacts with surrounding objects. The map is implemented as a multidimensional array that stores data about the location of objects in the game space. An important part of this map is a function that allows determining which objects are around a specific point on the map, which is critically important for navigation and interaction with the environment.

Another component is the **Player Inventory**, which contains a list of items the player possesses. These can be various resources such as wood, stone, coal, as well as tools like a pickaxe. These items play a crucial role in game scenarios where the player can use, combine, or place them in the game world.

The system also tracks **Player Achievements**, which are the player's accomplishments. This may include completing various tasks or achieving certain goals in the game.

To process all this data and perform various game tasks, basic verification functions (the **Base Functions** block) are used. This block includes a multitude of basic functions that can check whether any of the player's variables have changed, such as their level or position. These functions can also determine whether the player has moved in a certain direction or verify whether certain achievements have been completed. Importantly, these functions can work with both individual variables and more complex conditions, such as checking whether achievements are within a certain radius of the player.

Base Functions serve as the foundation for creating more complex scenarios in the game, which are gathered in the **Scenarios** block. Here, basic functions are combined to verify complex conditions and perform more advanced game tasks. For example, scenarios can determine whether one item was collected after another, whether a certain variable increases after a specific action, or check whether an item was placed next to another.

Table 1: The table compares CrafText with other environments based on Vocabulary, Instruction Length, Stochasticity, Dynamics, and Game Objects. CrafText offers a balanced mix of features, supporting both stochasticity and dynamics, with a substantial number of game objects.

Environment	Vocabulary	Instruction Length	Stochasticity	Dynamics	Game Objects
HomeGrid	100	9	×	✓	17
BabyAI	200	30	×	✓	5
RTFM	250	100	×	×	4
Messenger	400	100	×	×	4
Touchdown	4999	> 100	×	×	N/A
Alfred	1025	30	N/A	N/A	> 100
IGLU	400	100	✓	×	1
CrafText (ours)	756	89	✓	✓	> 30

3.3 Instruction generation

The pipeline for generating instructions in a game scenario involves creating a diverse and complex set of instructions that a player can give to an agent within the game. The instructions are generated based on *ChatGPT-4*and the *AskTheCode* extension, which allows the language model to directly analyze a repository with existing check functions and scenario checkers. In total, for the entire dataset, we needed around 200 queries to ChatGPT.

The prompt for generating instructions for a game scenario involves three stages:

Code Analysis: The first stage is analyzing the code where the functions for verifying the execution of actions are implemented. Understanding these functions is crucial for creating instructions that will be correctly interpreted and verified by the system.

```
instructions = {
    'instruction_1':
    {
    'instruction':
    "Place a torch to increase the player's health.",

    'items_name': ["torch"],
    'instruction_paraphrases': [

    "Put a lantern to boost the player's vitality.",
    "Set a light source to improve the player's health.",
    "Position an illuminator to enhance the player's well-being.",
    "Install a beacon to raise the player's life points.",
    "Place a flame holder so that the player's health increases."

    ],
    'check_lambda':
    lambda gt:
    did_placing_item_increase_variable(gt, "torch", "player_health")
    },
}
```

Figure 8: Example of generated instructions in the game scenario pipeline.

Instruction Creation: In the second stage, the instructions themselves are created. The language model is required to generate instructions that are diverse, cover a wide range of actions, and utilize various objects to thoroughly test the agent's skills and vocabulary. It is important that the instructions align with possible scenarios within the game.

Paraphrase Development: After creating the instructions, paraphrases are developed for each one. Paraphrases are alternative formulations of the same task, created using different synonyms and linguistic constructions. This allows testing the player's ability to understand the same task when presented in different ways. The paraphrases range from simple to complex, providing an opportunity to test the player's comprehension at various levels.

The final stage is the generation of instructions in a specific format, which includes a unique identifier, the main instruction text, a list of objects mentioned in the instruction, a set of paraphrases, and a lambda function for verifying the task's completion. An example of the generated instructions can be found in the figure 8.

3.4 Compression with other environments

CrafText distinguishes itself among the environments analyzed (see Table 1) by offering a well-balanced integration of essential features, including a substantial vocabulary, moderate instruction lengths, and support for both stochasticity and dynamics. Unlike other environments that focus primarily on textual complexity (e.g., Touchdown) or dynamic elements (e.g., HomeGrid), CrafText seamlessly combines these aspects, creating a more comprehensive and enriched setting for training agents. This balance enables CrafText to facilitate the development of both high-level interpretative skills and low-level adaptive strategies, making it a versatile and powerful environment for advancing research in multimodal learning.

4 Experiments

In our baseline, our goal is to predict the correct sequence of scenarios that an RL agent can execute in the environment based on natural language instructions. This involves accurately identifying the actions (scenarios) the agent needs to perform, specifying the correct parameters for those actions, and ensuring that the scenarios are predicted in the correct execution order (Figure 9).

Instruction

"Go north until you find a lake, then place a table near the lake."

High-Level Plan

```
find_water_source("north") and
place_object_near_target("table",
"water").
```

Figure 9: This figure shows an example of an instruction and its corresponding high-level plan.

The key challenge is to ensure the predicted scenarios have the correct actions, parameters, and sequence for the agent to execute the task successfully.

To address the task of scenario prediction, we trained the Llama2 and Gemma2-b models using specifically tuned parameters. Each model was trained over 5 epochs with an initial learning rate of 0.0002 and gradient accumulation to ensure stable weight updates. The batch size for training was set to 16, and we applied adaptive dropout along with gradient norm clipping to improve the overall training quality.

To further optimize the process and reduce memory consumption, we utilized a data compression technique that enabled 4-bit computations while maintaining high accuracy. This approach allowed for faster training with lower resource demands.

Metrics To evaluate the results, we used two key metrics: the success rate of predicted scenarios (SR scenarios) and the success rate of predicted scenario arguments (SR args). For SR Scenarios, we compared the sequence of original scenario names to determine if the predicted scenarios matched the actual ones in both order and composition. For SR Arguments, we compared the function parameters to ensure that the arguments were accurately predicted. This approach allowed us to precisely measure the model's ability to predict not only the correct sequence of actions but also the correct inputs for those actions.

Table 2: Metrics for Llama2 and Gemma-2b

Task Group	Success Rate	Llama2	Gemma-2b
Building	scenarios	0.96	**1.0**
	args	0.93	**0.96**
Localization	scenarios	**1.0**	0.95
	args	**1.0**	0.95
Sequencing	scenarios	1.0	1.0
	args	1.0	1.0
Combo	scenarios	0.66	0.66
	args	0.6	0.6
Total	*scenarios*	*0.91*	*0.91*
	args	*0.88*	*0.88*

Results. We evaluated these metrics for the model's predictions on the test dataset across all task types. The results can be found in Table 2. In the *Building* task section, Gemma-2b outperformed with a 100% success rate for scenarios and 96% for arguments, while Llama2 achieved a 96% success rate for scenarios and 93% for arguments. This suggests that Gemma-2b is better at interpreting and executing building tasks, possibly due to its greater flexibility in understanding various descriptions of object construction.

In the *Localization* task section, Llama2 surpasses Gemma-2b, showing a 100% success rate in both scenarios and arguments. Gemma-2b achieves a 95% success rate in both metrics. This indicates that Llama2 is more proficient at tasks requiring precise interpretation and execution of directional and placement instructions.

In *Sequencing* tasks, both models demonstrate an equally high level of performance, with a 100% success rate in both scenarios and arguments. This means that both models effectively reconstruct the sequence of task execution based on the text.

The *Combo* category tasks proved to be more challenging, with a 66% success rate for scenarios and 60% for arguments in both models. These tasks involve a complex combination of required scenarios, and judging by the low success rate in scenario prediction accuracy, it can be inferred that the models struggle with determining the correct sequence of scenario execution.

Overall, the success rate for scenarios was 91% for both models, and for arguments, it was 88%. Thus, both models demonstrated strong performance, but their strengths emerged in different aspects.

5 Conclusion

In this work, we introduce a new benchmark, CrafText, designed to evaluate the capabilities of language models in language grounding tasks within dynamic environments. The creation of CrafText was motivated by the limitations of existing environments, which either focus on text comprehension with limited environmental dynamics or feature complex mechanics but a restricted vocabulary. Our benchmark extends the Craftext environment, which is already dynamic and includes a variety of gameplay mechanics. We have added a set of instructions and tasks, formulated in natural language, that agents are required to complete, along with functions to verify task completion based on the game state. This solution addresses the aforementioned shortcomings, offering a rich and complex environment capable of testing models on diverse and non-trivial tasks.

During the implementation of language models, challenges related to task accuracy quickly became apparent. While models like Llama2 and Gemma-2b performed well on certain task types, complex combined scenarios revealed limitations in predicting the sequence of actions and arguments with precision. This underscores the complexity of the tasks and the need for further research to achieve higher performance levels.

Our future plans include incorporating reinforcement learning algorithms, which will not only predict high-level plans, as is currently done in the baseline model, but also execute scenarios directly within the environment.

6 Limitation

The primary limitation of this study is the absence of human-generated instructions in the dataset. Although AI-generated instructions provide consistency and scalability, they may lack the depth and nuance that human-crafted instructions could offer. This limitation could affect the model's ability to generalize to more complex and context-rich tasks, which are typical in real-world applications.

Another important limitation is the current framework's lack of integration with RL agent. Without an RL agent, the evaluation is limited to static, predefined scenarios, which restricts the model's ability to demonstrate learning and adaptation in real-time interactions. Future work should address this by incorporating an RL agent to enable a more comprehensive assessment of the model's capabilities and its potential for dynamic task execution.

While the use of ChatGPT raises accessibility concerns, the main focus will be on improving the dataset with human input and integrating RL capabilities to fully explore the model's potential.

References

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. 2020. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer.

Yash Kant, Arun Ramachandran, Sriram Yenamandra, Igor Gilitschenski, Dhruv Batra, Andrew Szot, and Harsh Agrawal. 2022. Housekeep: Tidying virtual households using commonsense reasoning.

Julia Kiseleva, Alexey Skrynnik, Artem Zholus, Shrestha Mohanty, Negar Arabzadeh, Marc-Alexandre Côté, Mohammad Aliannejadi, Milagro Teruel, Ziming Li, Mikhail Burtsev, Maartje ter Hoeve, Zoya Volovikova, Aleksandr Panov, Yuxuan Sun, Kavya Srinet, Arthur Szlam, Ahmed Awadallah, Seungeun Rho, Taehwan Kwon, Daniel Wontae Nam, Felipe Bivort Haiek, Edwin Zhang, Linar Abdrazakov, Guo Qingyam, Jason Zhang, and Zhibin Guo. 2022. Interactive grounded language understanding in a collaborative environment: Retrospective on iglu 2022 competition. In *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, 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. 2022. Interactive language: Talking to robots in real time. *CoRR*, abs/2210.06407.

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)*.

Fabian Paischer, Thomas Adler, Markus Hofmarcher, and Sepp Hochreiter. 2024. Semantic helm: A human-readable memory for reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

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, Gretchen Krueger, and Ilya Sutskever. 2021a. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021b. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR.

Manolis Savva, Jitendra Malik, Devi Parikh, Dhruv Batra, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, and Vladlen Koltun. 2019. Habitat: A platform for embodied AI research. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 9338–9346. IEEE.

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 *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

H. J. Austin Wang and Karthik Narasimhan. 2021. Grounding language to entities and dynamics for generalization in reinforcement learning. *ArXiv*, abs/2101.07393.

Peng Wang, Shijie Wang, Junyang Lin, Shuai Bai, Xiaohuan Zhou, Jingren Zhou, Xinggang Wang, and Chang Zhou. 2023. One-peace: Exploring one general representation model toward unlimited modalities. *arXiv preprint arXiv:2305.11172*.

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, Austin W Hanjie, Sida I Wang, Karthik Narasimhan, and Luke Zettlemoyer. 2021. Silg: The multi-environment symbolic interactive language grounding benchmark. *arXiv preprint arXiv:2110.10661*.

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

A Traning Hyperparameters

Parameter	Value
q_lora	
lora_r	64
lora_alpha	16
lora_dropout	0.1
bitsandbytes	
use_4bit	True
bnb_4bit_compute_dtype	"float16"
bnb_4bit_quant_type	"nf4"
use_nested_quant	False
training_args	
output_dir	"./results"
num_train_epochs	5
per_device_train_batch_size	16
per_device_eval_batch_size	4
gradient_accumulation_steps	1
gradient_checkpointing	True
max_grad_norm	0.3
learning_rate	2e-4
weight_decay	0.001
optim	"paged_adamw_32bit"
lr_scheduler_type	"cosine"
max_steps	-1
warmup_ratio	0.03
group_by_length	True
save_steps	0
logging_steps	25
fp16	False
bf16	True
sft	
max_seq_length	None
packing	False

Table 3: LLama And Gemma Hyperparameters Configuration

B Instruction Generation Prompt

The code for checking played scenarios can be found at the following repository link:

link/to/code/scenarious.py

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 pro-

vide 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(...): ...
             # Example usage of the
            function
    }
}
```

Replace `instruction_id` with a unique identifier for each instruction, and complete the `check_lambda` to demonstrate how you would verify the given instruction using the function.