Baba is AI: A Grounded Benchmark for Compositional Generalization in Dynamic Rule Systems

by

Meagan Jens

S.B., Computer Science and Engineering, Massachusetts Institute of Technology (2022)

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2023

Authored by:	Meagan Jens Department of Electrical Engineering and Computer Science May 12, 2023
Certified by:	Boris Katz Principal Research Scientist Thesis Supervisor
Certified by:	Andrei Barbu Principal Research Scientist Thesis Supervisor
Accepted by:	Katrina LaCurts Chair, Master of Engineering Thesis Committee

Baba is AI: A Grounded Benchmark for Compositional Generalization in Dynamic Rule Systems

by

Meagan Jens

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 2023, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

People leverage the compositional nature of their environment to generalize to new scenarios. For example, if you understand the meaning of the verb "to sing" and the adverb "loudly," then you can determine the meaning of the novel phrase "to sing loudly" from these known components. This process is known as generalization through systematic compositionality. Developing agents that can use systematic compositionality to generalize to new conditions has been a long-standing problem in AI. In response to this challenge, grounded benchmarks have been developed to evaluate an agent's ability to generalize using this approach. However, there are key problems with the current grounded benchmarks. To start, these benchmarks are ad-hoc. They propose sets of tasks without any formalism, so it is challenging to determine whether or not these tasks exhaustively explore the set of possible generalizations. This lack of structure also makes it challenging to compare benchmarks concretely. Another key issue with these benchmarks is that their environments are defined by a fixed set of rules and a small set of objects whose states can be changed. By strictly delineating the rules of these environments, we have overlooked a critical rule-understanding and manipulation capability that agents will need in the real world. Our approach to addressing these issues is twofold. First, we define a formalism to investigate generalization mathematically as a function of the environment architecture. We then use this formalism to create a novel type of generalization benchmark for agents that must learn to change the rules of their environments. Lastly, we run both supervised learning and reinforcement learning models on a small subset of the benchmark tasks to validate our environment and pinpoint key conditions under which agents fail to generalize.

Thesis Supervisor: Boris Katz
Title: Principal Research Scientist

Thesis Supervisor: Andrei Barbu

Title: Principal Research Scientist

Acknowledgments

I would like to thank Andrei Barbu and Ignacio Cases for their invaluable guidance throughout this project. Their kindness, patience, and thoughtful persistence is inspiring.

I am also grateful to Boris Katz for graciously inviting me into his lab. The people who I have met and worked with in the InfoLab have broadened my perspective on the meaningful challenges that we can tackle as engineers. I am consistently impressed by the level of curiosity, determination, humbleness, and empathy that characterizes the lab. It is a truly special community.

I would also like to thank Nathan Cloos, Yen-Ling Kuo, and Michelangelo Naim for their collaboration and mentorship throughout the year. I would especially like to recognize Nathan's contributions to this project. Nathan has also spent many hours working on the Baba environment, and he is responsible for developing the modeling pipeline for our benchmark. This project would not have been possible without his efforts.

I am incredibly thankful for my friends who have made my MIT experience such an exciting adventure that I will always cherish. In particular, I would like to thank Emelie, Helen, and Jess for making me feel at home at MIT.

Finally, I would like to thank my family for their consistent love and support. My parents Janice and Steve as well as my siblings Matthew and Michaela are a constant light in my life, and I would not be who I am today without them.

Contents

List of Figures

2-5 **Pushing objects can lead to several types of conflicts.** First, it is important to check for scenarios in which multiple objects are trying to push a single object at the same time. The four cases that we must consider are demonstrated under the heading *Pushing One Object*. In the first case, a door, a key, Baba, and a wall are pushing a ball from all four sides at time t. At time $t + 1$, none of the objects will have moved because the forces of the objects directly across from each other cancel out. More concretely, the forces from the key and the wall cancel each other out. Similarly, the forces from Baba and the door cancel out. In the second case, a door, key and Baba are pushing a ball from three sides. At time $t + 1$ the pushing forces of the door and Baba will cancel out (similar to the first case), so the key will push the ball. In the third case, the pushing forces of the door and Baba cancel each other out, so none of the objects move at time $t + 1$. Finally, in the fourth case, it would make sense that the key and Baba would push the ball diagonally (to the left and down). However, since diagonal movements are not permitted in this environment, the result of the key and Baba pushing the ball from adjacent sides will result in no movement at time $t + 1$. It is also important to check for cases in which an object such as Baba is pushing two adjacent objects as shown under the heading *Pushing Multiple Objects*. In this case, both balls should be pushed by Baba. Lastly, it is critical to check for cases in which objects are stacked, which is shown under the heading *Stacking Objects*. When two objects such as two balls are being pushed onto the same grid space at the same time, they should be stacked on top of each other. However, text blocks cannot be stacked. Therefore, if one or more of those balls is replaced by a text block, then neither the text blocks nor the objects will move onto that space. Overall, when multiple objects are moving around in the environment, the majority of push conflicts can be broken down into these base case scenarios. . 77

List of Tables

Chapter 1

The Role of Systematic Compositionality in Generalization

1.1 Motivation

Society is defined by a complex set of written and unwritten rules. The ability to follow and change these rules as we navigate the world is a critical component of human cognition. Therefore, in order for robots to have human-like interactions with the world, they should not only be able to interact with objects but also have the capacity to understand and manipulate the rules of their environment.

When people learn new concepts such as rules, they can usually generalize from one example. For instance, when a person drives in a foreign country, they understand the general rules of the road, but they are likely less familiar with how those rules are conveyed through road signs. When a driver encounters a *yield* sign for the first time, they may not realize that the sign means to yield and, consequently, the individual may drive right past it. As the driver neglects to slow down and give right of way to other traffic, the person will likely get honked at. As the driver observes surrounding cars yielding, the driver will realize that they broke a rule of the road and that the sign means to yield. The next time the driver encounters a road with a *yield* sign, they will know to yield based on their first interaction with this sign. This example illustrates a successful application of one-shot learning. Even children

can make useful generalizations through one-shot learning [5] [52]. The leading ML approaches, however, require vast amounts of data in order to learn these concepts with a level of accuracy comparable to humans [7].

People also learn richer representations of the world compared to ML approaches [7]. Take our previous example of learning the *yield* sign in a foreign country. The driver not only understands the notion of when and how to yield but also abstract ideas such as the consequences of not yielding or how the meaning of the rule yield can be conveyed differently depending on one's environment. They understand that breaking the rule could endanger everyone by causing a car accident. In general, people learn stronger models of their environment than machines do, which allows people to break ideas down and understand the relationships between the components, leverage their knowledge of existing categories to develop new abstract categories, and create new examples [18]. While newly developed meta-learning RL approaches have performed well in few-shot learning of complex scenarios [28] [45] [1] [36], this problem remains a long-standing challenge in AI [27] [38].

The capacity to learn rich concepts from sparse data is driven by peoples' ability to exploit the compositional nature of the world around them [37]. Language, for example, has a highly compositional structure driven by the parts of speech and the rules of grammar [31]. For example, if you understand the meaning of the word "to walk" as well as the meaning of the word "slowly," then you can likely infer the meaning of the phrase "to walk slowly," even if you have never before encountered that phrase. Daily tasks such as cooking dinner can also be expressed as a composition of skills. If an individual knows how to chop vegetables and mix the contents of a pot, then that individual can combine these skills to make a soup. Overall, people can compose ideas and skills to express new ideas and complete novel tasks. This approach to generalization is known as systematic compositionality. Systematic compositionality can be formally defined as the algebraic capacity to understand and produce a potentially infinite number of novel combinations from known components [10] [33] [40]. The underlying intuition is that simpler primitives can be used to build complex concepts from which meaningful abstractions can be inferred [14].

We hypothesize that generalization through systematic compositionality is the key to creating agents that can fully understand the rules of their environment. The idea that motivates our hypothesis is that models develop high level representations of the examples they see during training. If the data is drawn from a compositional environment and the model is able to internalize the information accurately and efficiently, then the model should be able to apply what it learns during training to generalize to new, more complex scenarios that are built from the same compositional environment. Compositionality is core to both thought and language [23]. In English, parts of speech such as nouns, verbs, pronouns, and adjectives can be composed, following the rules of grammar, to produce sentences. A sentence can be composed with other sentences to convey a more complex thought. Given that our thoughts are expressed through language, thoughts are also developed with this compositional structure.

Rules are expressed through language. Several examples from our day-to-day lives include road signs, research grant contracts, and university policies. Understanding a rule requires a critical element of abstract thinking. In order to fully learn the meaning of a rule, one must gain an understanding of the rule itself, the range of potential rule modifications, and the implications of both rule following and rule breaking. Given the rich, complex knowledge needed to understand a rule, it would be incredibly challenging for a model to learn a set rules without generalization. Therefore, given that rules are rooted in language, which is highly compositional, we believe that rich generalizations of rules can be learned through systematic compositionality.

Modern deep neural networks still struggle with the task of language-based generalization [8] [23] [24] [36] [53] [13] [3] [19] [4] [17] [20]. While models can display strong performance in language-based tasks, they are often tested in rigid environments that are not similar to real-world applications [46]. These models are also generally data hungry. This sample inefficiency suggests that these models may be making predictions by using observed broad patterns over many accumulated statistics rather than leveraging systematic compositionality. It has been argued that models that cannot apply systematic compositionality to generalization tasks are not tenable models of

the mind [40]. Therefore, developing models that can learn via systematic compositionality is critical.

While several benchmarks have been developed to measure language-based generalization for deep networks, very few of these benchmarks specifically examine a model's ability to perform systematic rule-based generalization. The SCAN data set examines a model's understanding of general compositional rules by testing zero-shot interpretation of new compositional linguistic expressions [23]. The data set consists of pairs of compositional navigation commands "*jump left*" and corresponding action sequences "LTURN JUMP." An agent must predict the action sequence based on the given command. The intuition behind their benchmark is that if an agent learns how to jump, then it can understand the meaning of jump twice or run and jump. The neural networks that they ran on their data set typically failed to generalize. Recent work has proposed training and architectural modifications to enable networks to generalize in the SCAN tasks [2] [22] [35] [41] [17], but to what extent that this success can be attributed to compositional generalization given the limitations of SCAN is unclear.

The SCAN domain lacks a world in which the commands can be interpreted, which is a critical limitation of this data set because understanding language relies on both compositional and contextual elements. An agent needs a reference point, a state of the word, in order to interpret references to objects and descriptions of actions. In SCAN, the agent simply maps one set of syntactic strings to another without an understanding of this context. gSCAN, therefore, was developed to address this limitation [40]. This data set includes a language grounded in the states of a grid world to evaluate the learning of linguistically motivated rules. This type of benchmark that includes a grid world environment is referred to as a grounded benchmark. With this grid world, the role of context perception in compositionality can be tested with commands such *as walk to the big yellow square* versus *walk to the small yellow square.* Similar to SCAN, recent developments have enabled strong network performance on the gSCAN benchmark.

While gSCAN is a significant improvement to the SCAN domain, it is still a rela-

tively brittle data set because the rules are fixed. Language-based rules are compositional, contextual, and dynamic. Learning a rule involves understanding the meaning of the rule, the variations that can be made, and the consequences of breaking it. Therefore, generalization should not only be measured by an agent's ability to follow a rule within a grid environment but also by its capacity to understand that rules can be created, changed, and broken. In order to investigate these greater levels of abstraction in terms of the agent's reasoning about the world, the environment must have a dynamic set of rules.

Another key issue with these types of grounded benchmarks such as gSCAN is that they are relatively unstructured and informal. They do not formally demonstrate the set of possible generalization tasks for that particular environment. Without this formalism, we are faced with two key challenges. First, when tasks are not generated with a structured approach, it is difficult to determine whether or not the given set of generalization tasks presented by the benchmark is exhaustive. Second, there is no objective method to compare benchmarks. Therefore, we seek to address both the brittle rule structure and the ad-hoc nature of grounded benchmarks in our work.

Our contribution is twofold. First, we introduce a framework that allows us to mathematically investigate generalization as a function of the environment architecture. This framework can be applied to any grounded benchmark to generate a set of tasks that tests an agent's generalization capabilities. Then, we construct a new kind of generalization benchmark for agents that must learn to manipulate the rules of their environments. We use our mathematical formalism to define the set of compositional generalization tasks in this benchmark. This benchmark, known as the Baba benchmark, allows us to test a broader notion of rule-based generalization compared to current benchmarks. Our data set consists of a grid world environment in which language-based rules can be modified. The flexible system of rules serves as a vehicle for studying how agents can interact with rules in an environment more analogous to the human world.

The Baba benchmark is inspired by the video game *Baba is You*, which is a puzzle game centered on the manipulation of rules. A side-by-side comparison of *Baba is You*

and our Baba environment is illustrated in Figure 1-1. These rules are represented as blocks with words written on them, and the goal of the game is to push these text blocks together and apart in order to build a set of rules under which the agent can reach the goal object. Similarly, the Baba benchmark is a grid world environment that consists of objects and text blocks. The agent controls a subset of these objects based on the current rules of the game. This ability to control different objects depending on the current set of rules is a unique feature of the Baba environment compared to current grounded benchmarks. The objective of the agent is to parse the rules from the grid and then to navigate to the goal object while making any necessary rule modifications along the way. This interaction between the agent and the environment can be described as a Markov Decision Process (MDP), and we will discuss the importance of this feature in Chapter 2. It is important to note that in both the game and our environment, the rules are built from words in the English vocabulary, but they do not follow the grammatical rules of the English language.

The set of generalization tasks that we define for the Baba environment using our mathematical formalism cover six broad categories: (1) novel transmutations, (2) novel compositions of pronouns and nouns, (3) novel compositions of action verbs and nouns, (4) novel compositions of termination verbs and nouns, (5) novel compositions of adjectives and nouns, and, lastly, (6) novel combinations of these other five compositions using conjunctions. We then run a set of experiments derived from these task categories to begin to evaluate the generalization capabilities of supervised learning (SL) and reinforcement learning (RL) models. This extension to current models is illustrated in Figure 1-6. While additional work needs to be done to build out a more robust set of baseline models, preliminary results from a transformer model and a Proximal Policy Optimization (PPO) model confirm the validity of our environment as a tool to measure generalization. They also highlight that models have an especially difficult time generalizing to tasks involving transmutations and novel compositions of pronouns and nouns. While our experiments currently focus on evaluating model performance on these generalization tasks under a static set of rules, the Baba environment supports measuring model performance under both static and

dynamic rule conditions.

1.2 Background

1.2.1 Generalization

Generalization refers to the ability of a model to adapt effectively to unseen data drawn from the same distribution as the training set. The ability to generalize enables models to learn more efficiently and perform well when deployed in dynamic, real-world settings. A common characterization of generalization is based on the number of examples required to learn a task: zero-shot, one-shot, and few-shot. Several methodologies have been applied in an effort to improve a model's ability to generalize within environments similar to Baba is You including regularization, data augmentation [12], Monte Carlo tree search [29], task generation [16], and transfer learning [30][49] [34] [51].

1.2.2 Compositionality, Systematicity, and Productivity

Much of our world and the rules that govern it are built from smaller blocks. People can leverage this compositional nature of the world to generalize to new conditions. Therefore, compositionality has a wide range of applications within machine learning from algorithm design[49][29] [48] [47] [32] [37] to benchmark development[12] [15] [11] to solving language-based and computer-vision-based tasks as well as multi-modal problems [32] [42].

Compositional generalization problems can be broken down into five categories: systematicity, productivity, substitutivity, localism, and overgeneralization. Systematicity is the ability to generalize by coupling known rules. Productivity is the capability of expanding predictions beyond the length observed during training. Substitutivity is the ability to swap analogous components. Localism is the ability to differentiate between composing locally versus globally. Finally, overgeneralization is characterized by being robust to exceptions[21]. It is worth noting that compositional

generalization is often used to solve language-based tasks and is especially relevant to RL problems.

1.2.3 Reinforcement Learning

Reinforcement learning is a type of machine learning in which an agent leverages feedback from the actions taken in an interactive environment to learn via trial and error. This feedback is defined by a set of rewards and punishments to signal positive and negative actions, respectively. RL has a wide range of applications from robotics to autonomous vehicles to language [26][25]. One key challenge within RL is that much of the current RL research focuses on benchmarks such as Atari[6] and MuJoCo [50] that train and test on the same exact environment, and this methodology fails to resemble dynamic real-world scenarios. In order to create models that are more adaptable and robust to environmental variation, several new benchmarks have been developed to address visual changes, different dynamics, and various structures. CoinRun, for example, varies the environment visuals from training to testing [12]. SCAN and gSCAN were also developed to address generalization in RL and will be further discussed in Section 1.3.

1.2.4 Markov Decision Process

The mathematical framework used to describe an environment in RL is an MDP. An MDP can be defined by the tuple $M = (S, A, R, T, p)$ where S is the state space, A is the action space, R is the reward function $R : S \times A \times S' \rightarrow \mathcal{R}$, $T(s'|s, a)$ is the transition function, and $p(s_0)$ is the initial state distribution. The goal is the find the optimal policy $\pi*$. A policy $\pi(a|s)$ generates a distribution over actions given a state. The optimal policy $\pi*$ maximizes the cumulative reward of the policy in the MDP:

$$\pi* = argmax_{\pi \in \Pi} E_{s \sim p(s_0)}[\mathrm{R}(s)]$$

$$\mathrm{R}(s) := E_{a_t \sim \pi(a_t|s_t), s_{t+1} \sim T(s_{t+1}|s_t, a_t)}[\sum_{t=0}^{\inf} R(s_t, a_t, s_{t+1})|s_0 = s]$$

R is the total expected reward earned by the policy from a given state s. If the MDP does not have a fixed horizon, the sum may not exist. In this case, either a horizon is assigned or an exponential discounting of future rewards is applied [21]. Unlike the game *Baba is You*, our Baba environment is designed to be an MDP. One interesting note about the environment is it can also be considered a factored MDP. In this case, one MDP could model the interaction between the agent and the objects, and a second MDP could model the interaction between the agent and the set of rules. We will leave the exploration of this factored MDP concept in the Baba environment for future work.

1.3 Key Benchmarks

1.3.1 SCAN

The SCAN data set, which stands for "**S**implified version of the **C**omm**A**I **N**avigation tasks," consists of a set of compositional navigation commands and a set of corresponding action sequences. Examples of these navigation commands and the complementary action sequences can be found in Figure 1-2. The grammar used to generate these sequences of compositional commands can be seen in Figure 1-3. The goal of this benchmark is to evaluate an SL model's ability to generalize to novel compositions of these navigation commands. The SCAN benchmark proposes three tasks to test different types of compositional generalization: 1) generalizing to commands demanding longer action sequences, 2) generalizing composition across primitive commands, and 3) compositionality in machine translation.

Agents are given the navigation command as an input and are then asked to predict the corresponding action sequence. The idea behind this experiment is that correct outputted action sequences reflect a model's capacity to generalize. However, one critical flaw in their approach is that the agent simply needs to map one set of syntactic

strings onto another set with no environment in which to interpret these commands. Therefore, it is hard to determine whether or not the agent is truly applying its ability to generalize when it performs the tasks outlined in this benchmark.

1.3.2 gSCAN

gSCAN is a grounded benchmark that was developed in response to the limitations of the SCAN benchmark. This data set consists of a set of commands and a corresponding grid world environment in which to interpret them. Similar to SCAN, the agent receives an input command and must then output the corresponding action sequence. However, unlike SCAN, the agent is also given a grid world environment, so it must output the action sequence based on the command and the current state of the environment. Figure 1-4 provides two examples of this process. The purpose for grounding these linguistic commands in the states of a grid world is twofold. First, it allows us to explore a richer set of linguistic commands. For example, since the agent can see the objects in the environment, commands can include descriptors such as "walk to the *red* ball" or "walk to the *small* square." The grammar used to generate the gSCAN commands is illustrated in Figure 1-5. Second, the benchmark more closely resembles how an agent would interact with the real world because we also interpret a set of rules within the context of our physical environment.

The gSCAN benchmark proposes six different tasks to explore compositional generalization: 1) the novel composition of object properties, 2) novel direction, 3) novel contextual references, 4) novel composition of actions and arguments, 5) novel adverbs, and 6) novel action sequence lengths. We will discuss these tasks is greater detail in Chapter 2. While at the time of publication the SL models did not perform well on the tasks, newer developments have enabled models to generalize well in these tasks.

1.4 Proposed Approach and Contribution

Grounded benchmarks that investigate generalization through systematic compositionality have two key problems. First, they are unstructured. They propose tasks without a formalism to ensure that the tasks exhaustively explore the set of possible generalizations that can be made in the given environment. Without this mathematical formalism, we are unable to determine the similarities and differences between different grounded benchmarks. Second, these benchmarks define environments that have a fixed set of rules and a small set of objects whose state can be manipulated.

We first define a formalism to mathematically investigate generalization as a function of the environment architecture, and our formalism can be applied to any grounded benchmark such as gSCAN. This structure allows us to define a set of tasks that are both exhaustive and scalable. As agents complete simpler generalization tasks for a given benchmark, we can not only construct more challenging tasks but also mathematically prove that these particular tasks are more difficult using our formalism.

With this formalism, we then develop a new kind of generalization benchmark, the Baba benchmark. By defining a static set of rules that an agent must follow, we have overlooked an critical capability that agents will need in the real world: the ability to understand rules via rule manipulation. Therefore, the Baba benchmark explores compositional generalization under conditions in which agents can modify the rules of the environment.

Lastly, we run a transformer and PPO on a small subset of our proposed tasks to generate preliminary results. These results validate our environment as a general tool to measure compositionality. Additionally, model failures on tasks involving transmutations and the novel composition of pronouns and nouns, which are both elements of compositionality that are particularly unique to the Baba benchmark, indicate that the benchmark creates meaningful generalization challenges for agents. This process is illustrated in Figure 1-6.

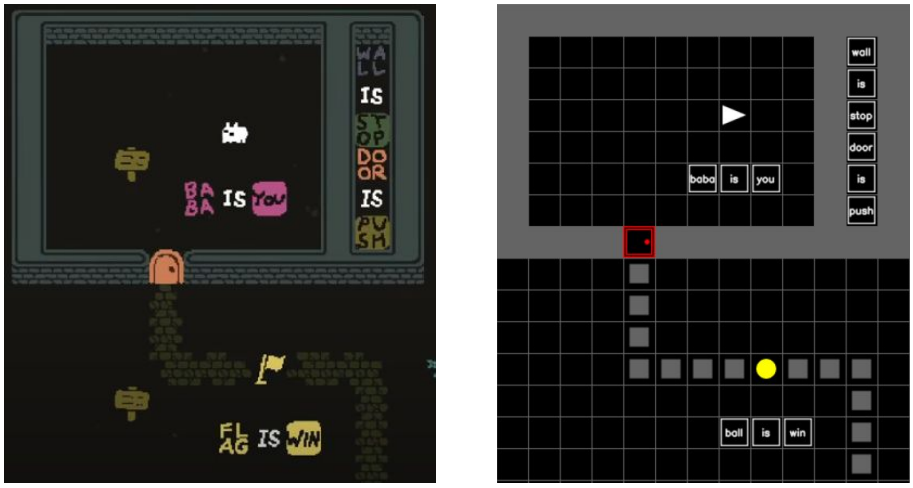
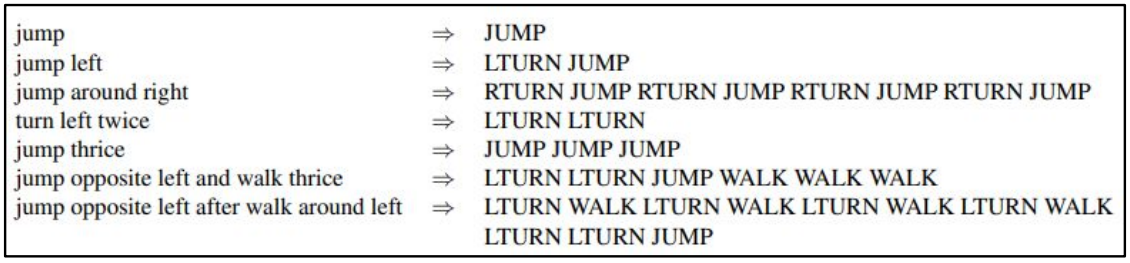


Figure 1-1: **We can replicate levels of the game *Baba is You* in the Baba environment.** This figure illustrates a side-by-side comparison of the game and our environment. On the left side of this figure is a screenshot of an early level of the game *Baba is You*. The image on the right is the corresponding implementation of that level in our Baba environment. We will walk through a short example of how this game is played. In the screenshot of the actual game environment, we can see that Baba is located in the center of the grid. Based on the rule *BABA IS YOU*, we know that the player controls this object. If we look around the environment, we will notice three other rules of this game: *WALL IS STOP, DOOR IS PUSH*, and *FLAG IS WIN*. The rule *WALL IS STOP* means that the player cannot pass through the wall objects, while the rule *DOOR IS PUSH* means that the player can push the door out of its way as it navigates the space. Most importantly, the rule *FLAG IS WIN* means that the player must reach the flag in order to win the game. Given these rules, the player can win the level by pushing the door in order to escape the walled enclosure and then navigating to the flag. In this simple example, the player does not need to make a rule in order to win the game. However, if, for instance, the text blocks *FLAG*, *IS*, and *WIN* were each scattered throughout the environment, then navigating to the flag would not complete the level. The player would first need to push these blocks together to form the rule *FLAG IS WIN* and then touch the flag. The right side of this figure illustrates the corresponding version of this level in our Baba environment. Since flags do not exist in our environment, we replace the flag with a ball.



jump	⇒	JUMP
jump left	⇒	LTURN JUMP
jump around right	⇒	RTURN JUMP RTURN JUMP RTURN JUMP RTURN JUMP
turn left twice	⇒	LTURN LTURN
jump thrice	⇒	JUMP JUMP JUMP
jump opposite left and walk thrice	⇒	LTURN LTURN JUMP WALK WALK WALK
jump opposite left after walk around left	⇒	LTURN WALK LTURN WALK LTURN WALK LTURN WALK LTURN LTURN JUMP

Figure 1-2: **The SCAN data set is made up of navigation commands such as *turn left twice* and the corresponding action sequence *LTURN LTURN*.** The goal of the agent is to predict the action sequence from the given command. This figure is taken directly from the SCAN paper [23].

C → S and S	V → D[1] opposite D[2]	D → turn left
C → S after S	V → D[1] around D[2]	D → turn right
C → S	V → D	U → walk
S → V twice	V → U	U → look
S → V thrice	D → U left	U → run
S → V	D → U right	U → jump

Figure 1-3: **A grammar is used to develop the the set of navigation commands in the SCAN benchmark.** The goal of this grammar is to demonstrate how words in the SCAN vocabulary can be composed to form compositional commands for the agents to interpret. This figure is also taken directly from the SCAN paper [23].

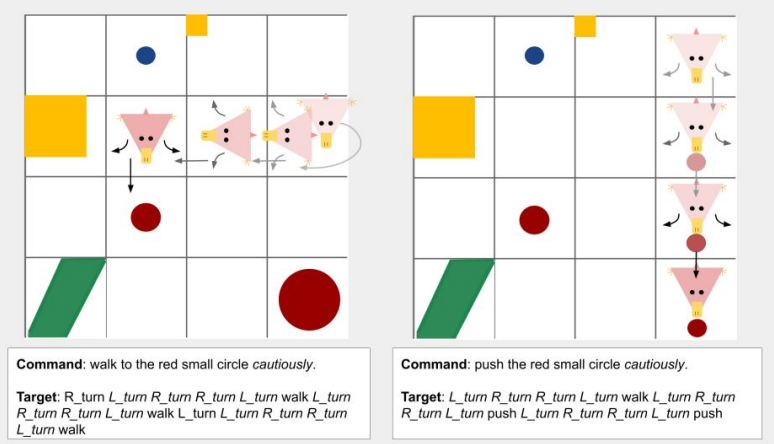


Figure 1-4: **The gSCAN navigation commands are grounded in a grid world environment.** The agent takes in the command and environment configuration as inputs and then predicts the corresponding action sequences. Here are two examples of this input (command, environment) pair and the output target action sequence. On the left, the agent is tasked with *walk to the red small circle cautiously*. In order to walk cautiously, the agent must check both ways (i.e. L_{turn}, R_{turn}) before taking a step. The agent must also recognize the difference between the small circle and the big circle in order to complete this task. Given the agent position as well as the positions of the objects in the particular environment, the agent must predict the set of actions that need to be taken in order to reach the target object. This figure is taken directly from the gSCAN paper [40]

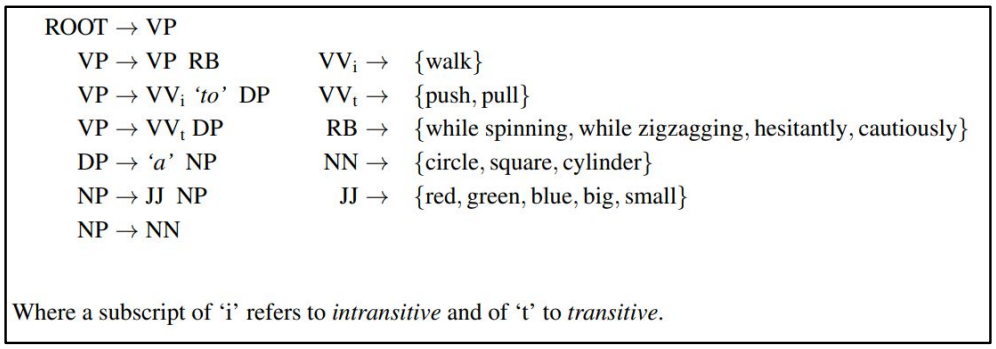


Figure 1-5: **Similar to the SCAN benchmark, the input gSCAN commands are also defined by a grammar.** The grammar shown here defines how the gSCAN data set vocabulary can be composed to form sequences of commands. This figure is taken directly from the gSCAN paper [40].

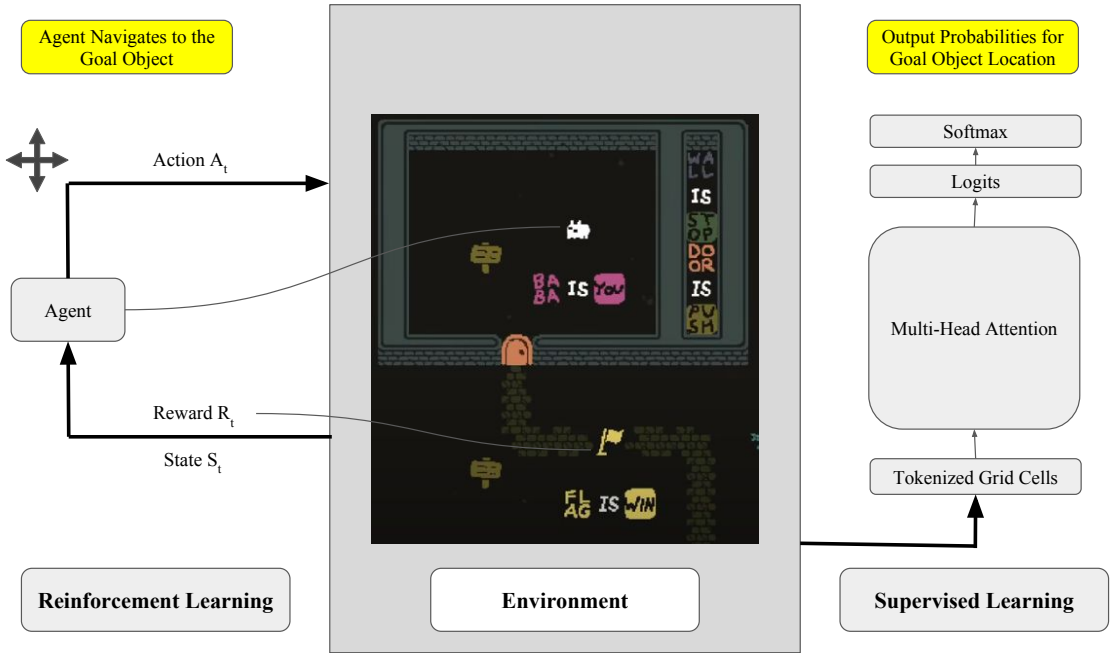


Figure 1-6: **The overall goal of the Baba benchmark is to evaluate a model's ability to generalize to novel conditions using systematic compositionality.** The right side of this diagram illustrates the SL case. SL agents are not able to navigate to the goal object; therefore, the goal of these models is to predict the next state of the environment or the location of a particular object given the current state of the environment. We specifically evaluate the performance of a transformer on this benchmark. The environment is converted to a one-hot encoding and passed into the model, which then predicts the next state of the environment or the location of a given object. The left side of this diagram demonstrates the RL case. RL agents can navigate to the goal object. Given that the interaction between the agent and the Baba environment can be described by an MDP, we can formulate this RL problem using the MDP formalism . At a high level, the agent receives the input environment state. It then takes an action from the available action space, which in this game is {up, down, left, right}. The agent earns a reward based on that chosen action and is then given the next state of the environment to repeat this process.

Chapter 2

The Baba Benchmark

The goal of the Baba benchmark is to evaluate the role of systematic compositionality in rule-based generalization. The core component of this benchmark is that the written commands are not only grounded in an environment, but the grounding itself can be manipulated via changing the rules of the environment. This dynamic design allows us to explore a broader notion of generalization compared to the current benchmarks. In this section, we will describe the environment design and the set of tasks that can be used to investigate rule-based generalization through compositionality.

2.1 Baba Environment

The original *Baba is You* is a single-player puzzle game where the player often controls a character named "Baba" and must navigate through the grid-based world filled with blocks, objects, and textual rules. For our purposes here, we can think of this game as a dynamic environment where the player interacts with various objects and rules to achieve specific goals. A remarkable aspect of *Baba is You* is that the rules of the game can be manipulated and rearranged by the player. Each level presents a set of rules that define how different objects and elements can interact with each other. For example, the actual character that the agent controls can change based on this set of rules. By pushing blocks and rearranging the rules, the player can alter the game's logic, enabling new actions and changing the properties of objects. The goal

of each level is to reach a specific win condition such as touching a flag. To accomplish this, the player must strategically modify the rules and objects in the environment to create new win conditions or change the behavior of existing ones. We can see a simple example of how this game works in Figure 1-1.

The Baba environment is a grid world environment developed in OpenAI's *gym-minigrid* engine [9] that is made up of objects and text blocks, which are depicted in Figure 2-2. The text blocks can be pushed together or apart in order to introduce or remove rules in the environment. These text blocks must be arranged according to the environment's rule grammar in order for a rule to be considered valid. The agent's objective is to parse the rules from the grid and then navigate to the goal object while making any necessary rule modifications along the way. The agent can step up, down, left, or right as long as it remains within the environment. In this section, we will discuss the mechanics of the Baba environment. We will highlight the rule components, the grammar, and the MDP structure of the environment.

We must make an important disclaimer before we describe the Baba environment. The rules in our environment are stated in a little language that resembles English but does not follow all of the mechanics of the English language. We borrow the parts of speech labels in English to categorize the vocabulary of this little language; however, these groupings should not be taken as the grammatical categories (i.e. the vocabulary grouping in our environment does not take on the properties of the corresponding grammatical category in English). For example, in the categorization of our environment vocabulary, there are both a noun and a pronoun category. In English, pronouns should be able to take the place of a noun in a sentence. However, in our environment, a word in the pronoun category cannot take the place of a word in the noun category in a rule. Therefore, the vocabulary groupings that we will discuss in this chapter do not correspond to the grammatical categories in English.

2.1.1 Rule Structure

The rules control the mechanics of the environment. There are five types of text blocks: noun N, pronoun Y, adjective (color) C, verb (action) A, and verb (termi-

nation) *T*. One special text block that is used in every rule is the text block *IS*. Therefore, we do not include that text block in these five categories. In order for a given set of text blocks to form a valid rule, these blocks must be ordered according to the grammar shown in Figure 2-1. The base structure of a rule is a noun *[N]* followed by *IS* and then a text block from one of the five categories {*N, Y, C, A, T*}. Additional complexity can be added to the rule when an adjective is combined with a noun in the subject of the rule (i.e. *[C] [N] IS* -.

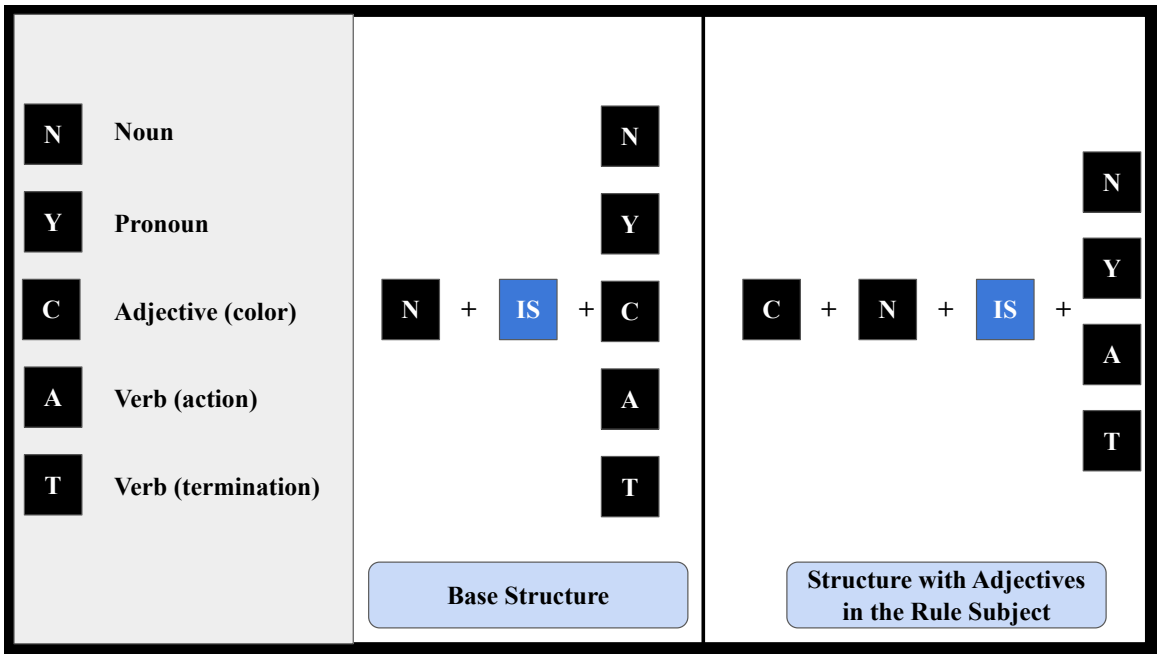


Figure 2-1: **We define two types of grammatical structures that can be used to create rules in the Baba environment.** The base structure is a noun followed by *IS* and then another text block from the set {noun, pronoun, adjective, verb (action), verb(termination)}. An extension of this base structure is to include an adjective before the noun. For example, the rule *BALL IS WIN* follows the base structure and means that if the agent touches any ball in the environment, it will complete the level. If we add an adjective to the beginning of this rule such as *GREEN BALL IS WIN*, then the agent will only complete the task if it touches a green ball in the environment. We narrowly define this rule structure to measure a controlled set of compositional structures. However, the game *Baba is You* includes a wide range of complex rule structures that do not follow English grammar, so adding new grammatical structures to our environment is a key area of future work.

The set of blocks for each text block category can be seen in Figure 2-2. There are five nouns: *BABA, BALL, DOOR, KEY,* and *WALL*. Each noun corresponds

to an object in the environment. It is worth noting that objects can be stacked on top of each other on the same space, but text blocks cannot be stacked. Two objects become stacked when they move move onto the same space at the same time. The text block *YOU* represents the only pronoun. There are seven adjectives that can be used to create rules: *BLUE, GREEN, GREY, PURPLE, RED, WHITE* and *YELLOW*. There are four action verbs *OPEN, PUSH, SHUT*, and *STOP*. Finally, *LOSE* and *WIN* are the two termination verbs that exist in our environment. Next, we will discuss how these text blocks can be used to build the rules that shape the mechanics of the environment.

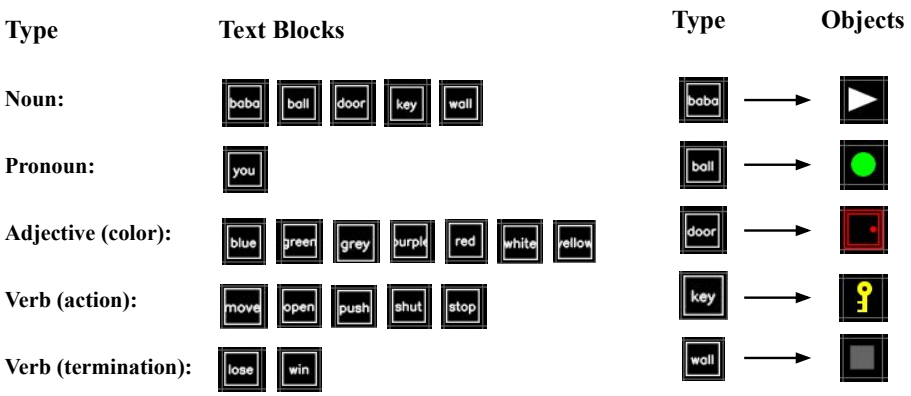


Figure 2-2: **The Baba environment is made up of text blocks and objects.** We group these text blocks into five categories: Noun, Pronoun, Adjective (color), Verb (action), and Verb (termination). These text blocks can be pushed together based on the grammar outlined in Figure 2-1. There are also five objects in the Baba environment: Baba, ball, door, key, and wall. The text blocks form rules that define how these objects can interact with each other in the environment.

Rules with a pronoun dictate the set of objects that the agent can control. For example, the rule *BALL IS YOU* allows the agent to control the movement of all balls in the environment. The agent then uses these balls to interact with its surroundings. If this rule is broken apart, the subset of objects that can be manipulated by the

agent changes as the agent will no longer control the balls. The ability for the agent to control any subset of objects is a unique element of the Baba environment, and this notion is captured in Figure 2-3. It is worth noting that this type of rule can lead to the edge case in which the agent does not control any objects. In this scenario, the agent cannot do anything to progress to the next state of the game and attempt to solve it, which results in an infinite loop. Therefore, we carefully design our training and testing environments to ensure that the agent is always controlling at least one object through the strategic placement of text blocks.

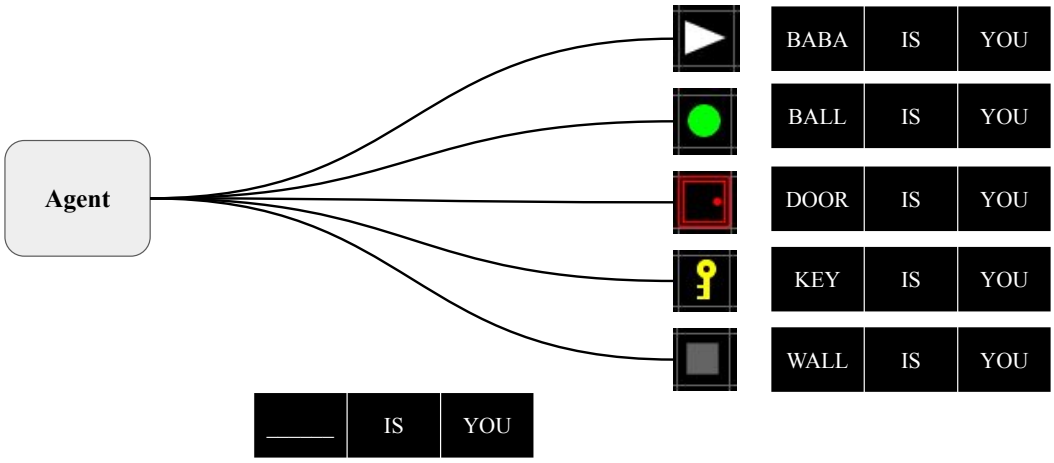


Figure 2-3: **The agent can control any set of objects in the environment as defined by the rule - *IS YOU*.** This ability to control any set of objects is a unique aspect of the Baba environment compared to other grounded benchmarks. It allows us to explore ideas of compositional generalization that require the agent to not only reason about the world but also to reason about itself.

An agent's goal is to reach a win object while avoiding any lose objects. Termination verbs dictate win and lose states in the Baba environment. The rule *DOOR IS WIN* sets all doors in the environment to be goal objects, while the rule *KEY IS LOSE* sets all keys to be defeat objects. If an object controlled by the agent navigates to a goal object or an object controlled by the agent is set as a win object (e.g. *BABA IS YOU* and *BABA IS WIN* are both in the set of rules), then the agent has won

that game. Similarly, if an object controlled by the agent navigates to a lose object or an object controlled by the agent is set as a lose object (e.g. *BABA IS YOU* and *BABA IS LOSE* are both in the set of rules), then the agent has lost that game.

Action verbs add an element of complexity to the environment. These actions are *PUSH, STOP, OPEN, SHUT* and *MOVE*. The rule *BALL IS PUSH* allows any object that collides with a ball to move ball in the direction that the object is moving. Text blocks can be pushed by default to enable rule modification. The rule *BALL IS STOP* means that any object that collides with a ball will not be able to continue to navigate along that path because the ball cannot move. The surrounding walls of the environment cannot be moved by default. When an object such as a door is shut through the rule *DOOR IS SHUT*, that door acts like a stop object as the door also cannot be pushed. If another object such as a key is set to open through the rule *KEY IS OPEN*, then both the key and the door will be removed from the environment when the key and the door overlap on the same grid space. This interaction resembles using the key to open the door. One common scenario in this environment is that a shut object will block an agent from reaching the goal object, so the agent needs to move an open object onto one of the shut objects in order to create a path to the goal object. Finally, the rule *BALL IS MOVE* means that at each time step, all balls will move one step in the direction of their orientation. An object will continue to step in that direction with each time step until it hits another stop object or the rule *BALL IS MOVE* is broken. This rule gives rise to scenarios in which multiple objects are moving at the same time in different directions. These object interactions can result in conflicts, which we will discuss in detail in Section 2.1.3.

Adjectives also contribute to the rich variety of rules that can be constructed in the Baba environment. Rules can refer to an object's color through the use of adjectives. For example, the rule *DOOR IS RED* sets all door objects in the environment to the color red, regardless of the object's initial color. This composition of adjectives and nouns can be combined with other compositions of nouns and verbs or nouns and pronouns to create a richer set of rules. For example, the rule *BLUE BALL IS WIN* means that only blue balls specifically are goal objects. Similarly, the rule *GREEN*

KEY IS PUSH means that only green keys can be pushed.

Lastly, an object can become another object based on the set of rules, which creates additional complexity in the Baba environment. For example, the rule *BALL IS DOOR* means that all balls will become doors. We call this property transmutation.

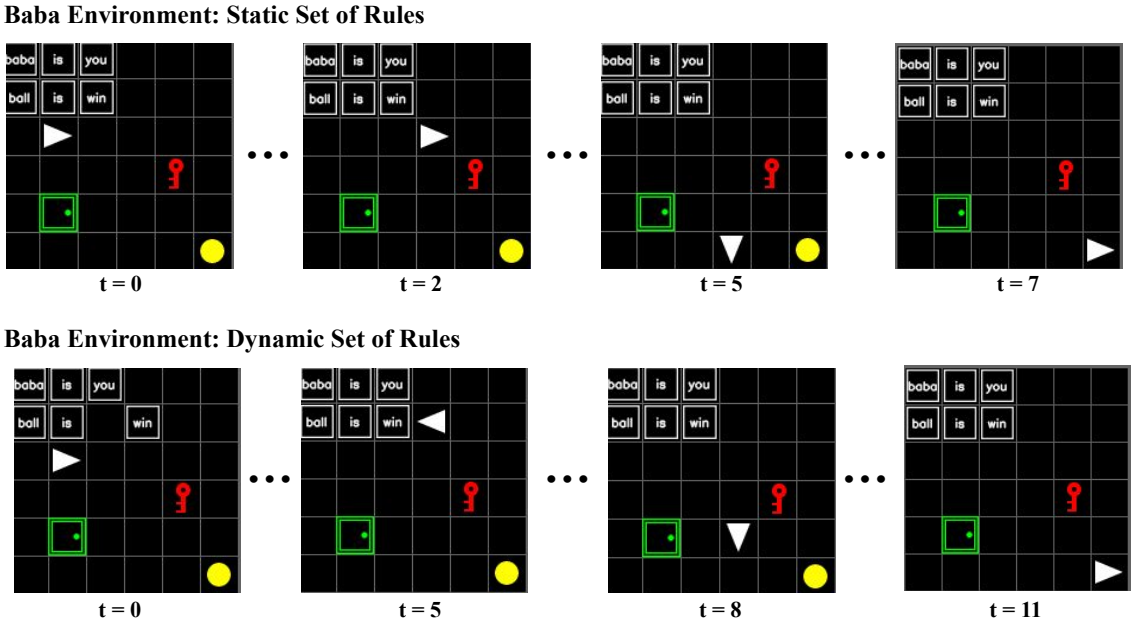


Figure 2-4: **The agent must understand the rules of the environment and then navigate to the goal object while making any necessary rules along the way.** This figure illustrates two simple examples of an agent completing a task in the Baba environment. The top sequence of figures demonstrates the static rule case. In the static case, the agent does not need to modify the set of rules in order to complete the task. The two rules are *BABA IS YOU* and *BALL IS WIN*. Therefore, the agent must realize that it can control Baba and that it needs to touch the ball in order to complete the task. Given the current environment structure, the agent does not need to modify any rules in order to reach the ball. The bottom sequence of figures illustrates a similar scenario. However, the agent must now change the rules in order to complete the game. We refer to this scenario as the dynamic case. The current rule is *BABA IS YOU*, so the agent can control Baba. The agent must recognize that there is no win condition, but it can push the text block *WIN* to the left to form the rule *BALL IS WIN*. Once the agent creates this rule, it can navigate to the ball to solve this environment.

We can compose these rules to create environments with obstacles and goals that the agent must navigate in order to reach the goal object. Figure 2-4 shows two simple examples of an agent solving the environment. In the static case, the agent

needs to understand the rules of the environment and go to the goal object. In the dynamic case, the agent must create a rule before navigating to the goal object.

2.1.2 MDP Design

The interaction between the agent and the Baba environment can be formulated as an MDP. This property of our environment is imperative in order for us to run meaningful experiments because MDPs are commonly used as the mathematical framework to model an RL problem. The game *Baba is You* cannot be described by an MDP, so we work to address these issues in our environment.

Given the rich set of rules and objects as well as the complex interactions that can take place, the game *Baba is You* has many hidden states. In order to be an MDP, the current state s must be known when an agent takes action a where $a = \{\text{up } (u), \text{down } (d), \text{left } (l), \text{or right } (r)\}$. An agent cannot move diagonally, and under the current environment implementation, it can only move one grid space per time step. Any process with hidden states is not an MDP. We address this issue in our environment by significantly reducing the number of text block types from about 100 to just over 10. Limiting the number of text blocks cuts down on the complexity of interactions that can occur in the environment, which allows us to remove hidden states.

Another key issue with *Baba is You* is that the order of object and rule creation affects the outcome of the game. Objects in play are stored in a list. When new objects are created, they are appended to the end of the list. Similarly, rules are also stored in a list where new rules are appended to the end of it. At each time step, the interactions are handled in the order defined by the object and rule lists. Therefore, objects or rules created later on in the game will always be handled last and may override interactions dictated by rules and objects earlier on in that list. Therefore, if there are two simultaneous actions that cannot be resolved cleanly, this update strategy results in interactions that are dependent upon when rules and objects were created. For example, if an object tries to move to two different spots at the same time, it will only move to the second spot, but any result from the first movement

will not be overridden. We will use a concrete example for clarity. Let us say that that Baba is supposed to move l and u at the same time. According to the ordering set by the object and rule lists, the the l movement is first, and the u movement is second. Let us also say that moving l will result in pushing an adjacent ball one space. In the resulting state, Baba will have performed the second movement, so it will have moved u. However, the ball will have also been pushed one space as if Baba had moved u. In order to be an MDP, however, the next state can only be determined by the current state. In this case, however, the resulting state depends not only on the current situation but also what occurred in the game in the past. Therefore, in the Baba environment, we have developed an approach to handle interaction conflicts in a structured manner so as not to rely on any internal ordering for environment updates. At each time step, a check is applied to identify and resolve these clashes using our logic described in the following Section 2.1.3.

Once we resolve these issues, we can describe the interaction between the agent and the Baba environment as an MDP. To start, the MDP M_{Baba} can be described by the set of properties (S, A, R, T, p). The state space S is given by the set of possible environment states that can be reached by the agent as it navigates the space. Each environment state s_i contains an encoding of the both the type and location of each text block and object. In general, the action space A is defined by the set of actions $\{u, d, l, r\}$.

The reward function can be given by $R(s, s') = 1$ if s' is a win state. By win state, we are referring to the scenario in which either an object controlled by the agent has navigated to a goal object or an object controlled by the agent is a goal object. Similarly, $R(s, s') = -1$ if s' is a lose state. A lose state can be defined as the scenario in which an object controlled by the agent has navigated to a lose object or an object controlled by the agent is a lose object. If s' is not a win or lose state, then $R(s, s') = 0$. We also use a discount factor γ where $\gamma < 1$ to account for the the amount of time taken to reach the win state. The transition from one state s to the next s' is generally deterministic, so the state transition $p(s, s') = 1$ if the action taken at that time a_t will lead from s to s'. Otherwise, $p(s, s') = 0$. However, if taking

action a leads to an object interaction edge case, which will be discussed in detail in Section 2.1.3, the next state may be determined stochastically. Therefore, the state transition $p(s, s')$ would be defined by the probability of that specific stochastic process.

2.1.3 Resolving Edge Case Conflicts

Like most grounded benchmarks, the Baba environment has edge cases. For instance, when two objects n_1 and n_2 are pushing against a third object n_3 from opposing sides at time t, what should be the resulting position of the three objects? In the game Baba is You, the outcome of this conflict is determined by the order in which the set of rules and objects are created. In this case, the object that appears later in the game's object list will get to push object n_3. With this update strategy, the outcome of this conflict depends on both the current state as well as previous states. In order for the Baba environment to be an MDP, we need a structured approach approach to resolve these conflicts. Therefore, we have developed a strategy to handle these edges cases in a manner that is based upon general physics principles rather than internal game structures.

In our implementation of the environment, we check for conflicts and update the object positions based on the strategy that we will discuss in this section. It is worth noting that many of these conflicts arise when objects are moving in different directions, which can occur when an object set to move (i.e. *[N] IS MOVE*) is moving in one direction while another object set to move or an agent is moving in another. However, we do not use the action verb *MOVE* in our current experiments, so as of right now an agent does not experience many of these conflicts. Nonetheless, we have developed a strategy and infrastructure to handle these edge cases so the Baba environment can handle more complex training and testing environments in the future. Thus, we will discuss conflicts related to the text block *MOVE* in Section 3.6 when we outline future work. We will now discuss the relevant major base case conflicts that can arise and the general update strategies to resolve them for each action verb.

Pushing objects can lead to several types of conflicts. Most of these conflicts arise only when objects are moving in different directions in the same time step, which can occur with rules involving *MOVE*. However, there is one relevant case that must be considered with our current experimental set up. When an object is pushing multiple adjacent objects, all of those objects should be pushed together. Figure 2-5 illustrates an example of this case under the *Pushing Multiple Objects* heading of the figure. For example, if Baba is pushing a ball that is directly adjacent to another ball as shown in the figure, then Baba should be able to push both balls.

Opening and closing objects can also lead to conflicts. When an open object is on the same space as a shut object, both objects are removed from the environment. There are two key edge cases to explore with this open-shut interaction. First, when an open object moves on the same space as a stack of shut objects, then the open object and all of the shut objects should be removed of the environment. This result makes sense because the open object can be used to open multiple shut objects. Conversely, when a shut object is moved onto a stack of open objects, then only the shut object and one of the open objects should be removed randomly. One open object is removed because only one open object is required to open the singular shut object.

2.2 Benchmark Tasks

We aim to test a model's ability to generalize to novel rules by leveraging systematic compositionality. Investigating this type of generalization is challenging because the problem itself as well as the existing benchmarks used to explore it lacks structure. Without a mathematical formalism to describe the set of experiments that are supported by a benchmark, we face two key issues. First, experiments lack the scalability and axes of variation to pinpoint when and why models fail to generalize. Second, comparing benchmarks is difficult because there is no metric that can be used to determine a benchmark's difficulty. While some benchmarks such as gSCAN have intro-

duced structure in their experiments by developing a context free grammar to define the input commands for a given experiment, none of the existing benchmarks have created a mathematical formalism to capture the experiments themselves. Therefore, our main contribution is a context free grammar (CFG) that can be used to describe experiments that test compositional generalization. This CFG describes the compositional generalization explored as a function of the environment architecture. We apply this CFG to develop a set of tasks that explore six key areas of compositional generalization: transmutation, the composition of nouns and pronouns, the composition of nouns and action verbs, the composition of nouns and termination verbs, the composition of nouns and adjectives, and the combination of composed components, which we will collectively refer to as the Baba benchmark.

In this section, we will first motivate our benchmark tasks by expressing the gSCAN splits using our CFG. Next, we will propose the Baba benchmark, which will demonstrate how the CFG ensures that our tasks are both exhaustive in terms of the types of compositional generalization that can be explored and include incremental degrees of complexity. Then, we will discuss the key differences between gSCAN and our benchmark using the CFG as a framework for our discussion. Lastly, we will demonstrate how this formalism of the Baba benchmark can be used to generate the train and test environments for experiments automatically.

2.2.1 Restructured gSCAN Tasks

A core component of the gSCAN benchmark is the set of verbal commands that are used to generate the experimental data splits. These commands are made up of five key components: objects, adjectives, verbs, directions, and adverbs. Table 2.1 specifies the elements that are available for each type of component. These elements can be composed to generate commands with varying degrees of complexity [40]. These compositions are defined in Table 2.2. The authors of the gSCAN paper developed six data splits to test different forms of linguistic generalization. In the section below, we will express these splits in the same CFG used to articulate our set of benchmark tasks.

Table 2.1: gSCAN Components

Type	**Elements**
Object (O)	circle, cylinder, square
Adjective (J)	red, green, blue, big, small
Verb (V)	walk, push, pull
Direction (D)	North, South, East, West
Adverb (A)	while spinning, while zigzagging, hesitantly, cautiously

Table 2.2: gSCAN Component Composition

Notation	**Definition**
J(O)	combination of O and A (e.g. *red ball*)
V(O)	combination of O and V (e.g. *push the ball*)
A(V)	combination of V and A (e.g. *walk cautiously*)
D(V)	combination of V and D (e.g. *walk north*)

The motivating idea behind this CFG is that for each experiment, there is a fixed, held out test set, which will be represented by h. This set h can contain different component types, and we will represent a held out component as $h_i, i \in \{o, j, v, d, a\}$. For example, h_j represents a held out adjective such as *green* or *red*. A given test will sweep over its respective h, and all experiments are constrained under gSCANPhysics.

Experimental Splits

Novel Composition of Object Properties Two types of generalization via compositionality are explored in this data split: composition of references and composition of attributes. The former is evaluated by holding out all examples where a yellow square is referred to by its color in the training set, and at test time the agent must understand the phrase "yellow square" in order to navigate to the target object (the yellow square). A model should be able to generalize in this scenario because it has the opportunity to gain familiarity with the object since it is a target object in some training scenarios (the command does not refer to it as the "yellow square"), and the agent learns from commands referring to other yellow objects as well as squares

of different colors. The latter is investigated by holding out all examples where a red square is the target in training, and at test time the agent must understand the phrase "red square" in order to navigate to the target object (the red square). The motivation behind this split is that even though a red square is never a target, the agent can become familiar with the red square because it appears as a non-target background object during training [40].

- Train: $\forall o, j.\ j \neq h_j \wedge o \neq h_o.\ \exists\ t.$ Execute$(t) \vDash j(o)$ s.t. gSCANPhysics(t)
- Test: $\exists\ t.$ Execute$(t) \vDash h_j(h_o)$ s.t. gSCANPhysics(t)

Novel Direction Generalization via composing directions is explored by holding out all examples where the target object is located south-west of the agent, so at test time the agent must generalize to walking west and then south, or vice-versa, in order to reach the target object. The intuition behind this experiment is that the agent gains exposure to moving north, south, and east and sees examples of composing these directions by moving north-west, north-east, and south-east during training, so it can ground to moving south-west at test time [40].

- Train: $\forall d.\ d \neq h_d.\ \exists\ t.$ Execute$(t) \vDash d(v)$ s.t. gSCANPhysics(t)
- Test: $\exists\ t.$ Execute$(t) \vDash h_d(v)$ s.t. gSCANPhysics(t)

Novel Contextual References Generalization via composing contextual references is explored by holding out all examples that meet three conditions: 1) the target object is a circle of size two 2) the circle of size two is the smallest circle in the environment and 3) the agent is instructed to go to the "small circle." At test time, the agent must generalize to the meaning of "small circle" and locate the target object, a circle of size two. A model should be able to generalize in this split because it can ground to circles of size two through references such as 'green circle' and ground to the meaning of "small" through scenarios in which squares of size two are referred to as "small" [40].

- Train: $\forall o, j.\ j \neq h_j \wedge o \neq h_o.\ \exists\ t.$ Execute$(t) \vDash j(o)$ s.t. gSCANPhysics(t)

- Test: $\exists\ t.\ \text{Execute}(t) \vDash h_j(h_o)$ s.t. gSCANPhysics(t)

Novel Composition of Actions and Arguments Generalization via composing actions and arguments is investigated by holding out examples where the command is to "push" the target object, a square of size three. Objects of size three are considered heavy and need to be pushed twice by an agent in order to move one grid cell, while objects of size two only need to be pushed once. At test time, the agent must generalize to pushing the square of size three twice in order to move it. The intuition behind this data split is that an agent can ground to squares of size three by needing to "pull" this object, which, similar to "push," requires two "pulls" in order to move the square one grid space. It also sees examples of pushing circles and cylinders of size three, which also require two pushes [40].

- Train: $\forall o, j, v.\ \ j \neq h_j \wedge o \neq h_o \wedge v \neq h_v.\ \ \exists\ t.\ \ \text{Execute}(t) \vDash v(j(o))$ s.t. gSCANPhysics(t)
- Test: $\exists\ t.\ \text{Execute}(t) \vDash h_v(h_j(h_o))$ s.t. gSCANPhysics(t)

Novel Adverbs The authors of the gSCAN benchmark explore generalization via composing adverbs through two strategies. The first approach (1) is learning the adverb "cautiously" from a few examples during training and then generalizing to all possible commands with that adverb. In the second approach (2), they provide examples of the adverb "while spinning" and the verb "pull," and then at test time the agent must compose the two concepts to "pull while spinning." The motivation behind this data split is that the model learns the adverb during training and should then be able to apply it in novel ways during test time [40].

- Train 1: $\forall v, a.\ \ h_a.\ \ \exists\ t.\ \ \text{Execute}(t) \vDash h_a(v)$ s.t. gSCANPhysics(t) with 1 - 50 examples
- Test 1: $\exists\ t.\ \text{Execute}(t) \vDash h_a(v)$ s.t. gSCANPhysics(t)
- Train 2: $\forall a, v.\ a \neq h_a \wedge v \neq h_v.\ \exists\ t.\ \text{Execute}(t) \vDash a(v)$ s.t. gSCANPhysics(t)
- Test 2: $\exists\ t.\ \text{Execute}(t) \vDash h_a(h_v)$ s.t. gSCANPhysics(t)

Novel Action Sequence Lengths For this task, the agent does not need to compose learned commands to generalize to new commands. Instead, the agent is trained on examples where the length of the corresponding action sequence is no more than fifteen actions. At test time, the agent will be given the same commands, but the corresponding action sequence will be longer due to the configuration of the environment. For example, additional objects in the environment may cause the agent to need to take additional steps in order to fulfill the task [40]. Given the broad and relatively nebulous nature of this task, we will do our best to describe it with our CFG.

- Train 1: $\forall o, v, j, a, d.\ \exists\, t.$ Execute$(t) \vDash j(o) \vee v(o) \vee a(v) \vee d(v)$ s.t. gSCANPhysics(t) with corresponding action sequences less than or equal to 15 actions
- Test 1: $\exists\ t.$ Execute$(t) \vDash j(o) \vee v(o) \vee a(v) \vee d(v)$ s.t. gSCANPhysics(t) with corresponding action sequences greater than 15 actions

Rewriting the gSCAN benchmark using this mathematical formalism provides the structure to evaluate the gSCAN splits systematically. First, we are able to pinpoint areas of overlap in the proposed data splits. For example, the splits *Novel Composition of Object Properties* and *Novel Contextual References* are both testing the composition of adjectives and objects (i.e. $j(o)$).

Second, this structure mathematically demonstrates how tasks build from one another to create greater complexity. For instance, the split *Novel Composition of Actions and Arguments* is clearly more complex than the split *Novel Composition of Object Properties* from a compositional perspective because the splits explore the composition of $v(j(o))$ and $j(o)$, respectively. This idea of defining the concept of difficulty from a compositional generalization perspective is critical. This CFG provides us with the language to articulate not only the comparisons between splits but also comparisons between benchmarks. With our CFG, we can quickly identify axes of difficulty in tasks that require compositional generalization.

Lastly, this structure enables us to identify additional tasks that could be explored in the gSCAN environment. For example, we could combine splits *Novel Direction*

and *Novel Adverbs* to evaluate generalization by composing directions and adverbs with verbs. Thus, this framework serves as a platform for the development of new benchmarks to test generalization based on existing ones such as gSCAN. With this richer, cleaner representation of the splits, we can develop a deeper knowledge of the gaps in current benchmarks and gain a greater understanding of the environments in which models currently fail to generalize. Thus, the CFG equips us with the tools needed to use gSCAN as a point of inspiration for our Baba benchmark.

Overall, the gSCAN splits exhibit four key issues: 1) splits overlap in terms of type of compositionality being tested, 2) there is no formal strategy for increasing task complexity, 3) there are gaps as the splits do not represent the full range of compositionality that can be examined in the gSCAN data set, and 4) the rule set is static. We seek to address these issues in our proposed Baba benchmark.

2.2.2 Baba Benchmark Tasks

One of our primary contributions is this set of Baba benchmark tasks that can be used to evaluate a broader notion of rule-based generalization. We propose a set of tasks that leverages the Baba environment's dynamic set of rules to explore the role of systematic compositionality in generalization. Specifically, we seek to explore the composition of six splits: (1) transmutations, (2) pronouns and nouns, (3) action verbs and nouns, (4) termination verbs and nouns, (5) adjectives and nouns, and (6) combinations of these compositions. The components of the Baba environment are pictorially displayed in Figure 2-2 and are also shown in Table 2.3. The variety of compositions that can be formed by combining these components is outlined in Table 2.4.

For each split, we have developed a set of tasks. In this section, each task will be motivated by a discussion of the compositional properties that are being evaluated. The task will then be defined using our proposed CFG. It is worth noting that the variations for each task are generally presented based on increasing compositional complexity.

Analogous to the gSCAN data split description in Section 2.2.1, each Baba task

Table 2.3: Baba Components

Type	**Elements**
Noun (N)	baba, ball, door, key, wall
Pronoun (Y)	you
Adjective (C)	red, green, blue, purple, yellow, grey, white
Action Verb (A)	push, stop, open, shut
Termination Verb (M)	defeat, win

Table 2.4: Baba Component Composition

Notation	**Definition**
N(N)	combination of two N's (e.g. *BALL IS DOOR*)
Y(N)	combination of N and Y (e.g. *BABA IS YOU*)
C(N)	combination of N and C (e.g. *RED BALL IS WIN, KEY IS BLUE*)
A(N)	combination of N and A (e.g. *BALL IS PUSH*)
M(N)	combination of N and M (e.g. *BALL IS WIN*)

will be expressed using our CFG. For each task, there is a fixed, held out test set, which is represented by h in our CFG. This set h can contain different Baba component types, and a held out component type can be represented as follows: $h_i, i \in \{n, y, a, m, c\}$. A given test will sweep over its respective h, and all experiments will be constrained under babaPhysics.

The ability to create, modify, and break rules is unique to the Baba environment. Unlike the gSCAN benchmark, the tasks outlined in this section can be performed in both a static and dynamic environment. A static environment refers to the scenario in which an agent can perform the task without needing to modify the set of rules. Conversely, in a dynamic environment, the agent needs to make a new rule in order to complete the task. We hypothesize that this dynamic environment will be significantly harder for agents to navigate. Therefore, having both a static and dynamic version of a given task serves as an important point of comparison that allows us to better measure the impact that a dynamic set of rules has on an agent's performance in compositional generalization tasks. This type of comparison cannot be made in existing benchmarks. After we describe each task using our CFG, we will discuss how that particular task

be be modified to create a dynamic version of it.

Experimental Splits

Ex1: N(N): Any noun (object) can transmute into any other noun (object). This split explores generalization by composing transmutations of objects into other objects. A transmutation is defined by the rule *[N] IS [N]*. For example, when a key transmutes into a ball, all keys will become balls. This specific transmutation is specified by the rule *KEY IS BALL*. The ability for an object to transform into another object is a unique component of the Baba environment that is not found in related benchmarks such as gSCAN. We will leverage this capability of the Baba environment to explore the compositionality of transmutations in the following set of tasks.

1. **An agent has never seen h_{n1} transmute into h_{n2}** (i.e. h_{n1} *IS* h_{n2}). The model should be able to generalize because of the examples it has seen during training. First, it has seen h_{n2} transmute into h_{n1}. Second, it has seen one of the held out objects $\{h_{n1}, h_{n2}\}$ transmute into other objects and other objects transmute into one of the held out objects $\{h_{n1}, h_{n2}\}$. Finally, it has seen examples of objects not in the set $\{h_{n1}, h_{n2}\}$ transmute into each other (e.g. *X IS Y* and *Y IS X*). At test time, the model should be able to compose its existing knowledge of h_1, h_2, and transmutations to generalize to the rule h_{n1} *IS* h_{n2}. This task is shown in Figure 2-6.

   Train: $\forall n_1, n_2.\ n_1 \neq h_{n1} \wedge n_2 \neq h_{n2}.\ \exists\, t.\ \text{Execute}(t) \vDash n_2(n_1)$ s.t. babaPhysics(t)

   Test: $\exists\ t.\ \text{Execute}(t) \vDash h_{n2}(h_{n1})$ s.t. babaPhysics(t)

   Dynamic Version: During training, the agent must make the rule $n_2(n_1)$ (i.e. n_1 *IS* n_2). At test time agent must make the rule $h_{n2}(h_{n1})$, which translates to h_{n1} *IS* h_{n2}. For the remaining tasks, we will only specify which rule should be made at test time because the rule that the agent should be making during training can be inferred by symmetry.

2. **An agent has never seen h_n transmute into another object** (i.e. h_n *IS [N]*). This task is incrementally harder than the first task. Similar to task 1.1, the agent sees examples during training of objects transmuting into h_n, and it sees examples of objects transmute into each other. However, unlike the first task, the model has never seen h_n transmute into other objects. At test time, the model must leverage its understanding of the compositional nature of transmutation and h_n to generalize to the rule h_n *IS [N]*.

   Train: $\forall n_1, n_2.\ n_1 \neq h_n.\ \exists\ t.$ Execute$(t) \vDash n_2(n_1)$ s.t. babaPhysics(t)

   Test: $\exists\ t.$ Execute$(t) \vDash n_2(h_n)$ s.t. babaPhysics(t)

   Dynamic Version: The agent must make the rule $n_2(h_n)$, which translates to h_n *IS* $[N]$.

3. **An agent has never seen an object transmute into h_n** (i.e. *[N] IS* h_n). This task is incrementally harder than task 1.1 and similarly difficult to task 1.2. Similar to the task 1.1, the agent sees examples during training of h_n transmuting into other objects, and it sees examples of other objects transmuting into each other. However, unlike task 1.1, the model never gains exposure to objects transmuting into h_n. At test time, the model must use its familiarity with h_n and the compositional nature of transmutation to generalize the the rule *[N] IS* h_n.

   Train: $\forall n_1, n_2.\ n_2 \neq h_n.\ \exists\ t.$ Execute$(t) \vDash n_2(n_1)$ s.t. babaPhysics(t)

   Test: $\exists\ t.$ Execute$(t) \vDash h_n(n_1)$ s.t. babaPhysics(t)

   Dynamic Version: The agent must make the rule $h_n(n_1)$, which translates to *[N] IS* h_n.

Ex2: A(N): Any action verb (action) can be combined with any noun (object) to form a rule. This split tests generalization by composing nouns and action verbs. A noun-action verb combination is defined by the rule *[N] IS [A]*. For instance, the rule *BALL IS PUSH* means that the balls in the environment can

be pushed. While the gSCAN environment has the verbs *push* and *pull*, the Baba environment has a greater variety of action verbs, which are *PUSH, STOP, OPEN* and *SHUT*. We will use this variety of action verbs in the following task.

1. **An agent has never seen action a with object h_n in the same rule** (i.e. h_n *IS* a). During training, the model will see examples of the action a composed with other nouns. The model can ground to the meaning of h_n as that object can be in training environments and can have other actions or properties applied to it. At test time, the model must combine its understanding of a, h_n, and the compositional nature of actions and nouns to generalize to the rule h_n *IS* a. Figure 2-7 illustrates an example of this task.

   Train: $\forall n.\ n \neq h_n.\ \exists\ t.$ Execute$(t) \vDash a(n)$ s.t. babaPhysics(t)

   Test: $\exists\ t.$ Execute$(t) \vDash a(h_n)$ s.t. babaPhysics(t)

   Dynamic Version: The agent must make the rule $a(h_n)$, which translates to h_n *IS* a.

Ex3: M(N): A termination verb (termination) can be combined with any noun (object) to form a rule. We are investigating generalization by composing objects and terminations. A termination combined with an object is described by the rule *[N] IS [M]*, so, for example, the rule *KEY IS WIN* means that the key is the goal object, while the rule *DOOR IS LOSE* means that all doors are defeat objects. The gSCAN benchmark also has goal objects, but it does not have a defeat object. The ability to compose a defeat termination verb with an object to form a rule in the Baba environment creates a greater variety of experiments to investigate this type of generalization.

1. **An agent has never seen termination verb m combined with noun h_n** (i.e. h_n *IS* m). The model can learn the meaning of termination h_m through training examples of where h_m forms rules with other objects. It can also ground to the meaning of object h_n as that object can be in training environments and

can be involved in other rules that do not include m. During testing, the model must leverage its familiarity with m, h_n, and the compositional nature of termination verbs and objects to generalize to the rule h_n *IS* m. Figure 2-8 provides an example of this task.

Train: $\forall n, m.\ n \neq h_n.\ \exists\ t.$ Execute$(t) \vDash m(n)$ s.t. babaPhysics(t)

Test: $\exists\ t.$ Execute$(t) \vDash m(h_n)$ s.t. babaPhysics(t)

Dynamic Version: The agent must make the rule $m(h_n)$, which translates to h_n *IS* m.

Ex4: Y(N) A pronoun can be combined with any noun (object) to create a rule. We seek to evaluate generalization by composing objects and pronouns in this split. A pronoun and an object can be used to construct rules of the form *[N] IS YOU*. For example, the rule *BALL IS YOU* means that the agent can control all balls. The ability of the agent to control one or more object types at the same time is a capability that is unique to the Baba environment and creates a new element of difficulty. While people do not physically transmute into new forms, we do take on new roles throughout our life through our jobs and our communities. Therefore, creating models that can not only use systematic compositionality to reason about the world but also to reason about the model itself is critical. Thus, we seek to test an agent's ability to leverage compositionality to generalize to situations in which the agent controls a novel set of objects.

1. **An agent has never controlled object** h_n (i.e. h_n *IS YOU*). In this task, the model will learn from examples of controlling other objects during training. It can also ground to the meaning of h_n as that object can be in training environments and can be involved in other rules. Therefore, at test time, the agent must leverage its knowledge of the pronoun YOU, the object h_n, and the ability to compose pronouns and objects to generalize to the rule h_n *IS YOU*. An example of this task is depicted in Figure 2-9.

   Train: $\forall n.\ n \neq h_n.\ \exists\ t.$ Execute$(t) \vDash y(n)$ s.t. babaPhysics(t)

Test: $\exists\ t.$ Execute$(t) \vDash y(h_n)$ s.t. babaPhysics(t)

Dynamic Version: The agent must make the rule $y(h_n)$, which translates to h_n *IS YOU*.

Ex5: C(N) An adjective (color) can be combined with any noun (object) to build a rule. The objective of this split is to explore generalization via composing colors and objects. A color and an object can be used together to create two types of rules: *[N] IS [C]* or *[C] [N] IS* -. In the first case, the rule *BALL IS BLUE* means that all balls, regardless of their initial color, will be set to the color blue. For the second case, the rule *BLUE BALL IS WIN* means that only objects that are both the color blue and of type ball are goal objects. This composition of adjectives and nouns is explored in the gSCAN split *Novel Composition of Object Properties*, so this split serves as an important point of connection between the gSCAN benchmark and the Baba benchmark.

1. **An agent has never seen h_c combined with h_n** (i.e. (1) h_n *IS* h_c or (2) h_c h_n *IS* $-$). During training, the model can learn the meaning of the color h_c and how it can be used in a rule to either refer to or modify the color of an object (depending on the rule structure). It learns this ability to compose h_c and h_n through examples of seeing the interaction between h_c and other nouns. It can also ground to the meaning of h_n as that object can be found in training environments and can be involved in other rules (that involve any color except h_c). Depending on training environment construction, training examples could include an object of type h_n and of color h_c, but the set of rules in that training environment should not include any rules associating the object h_n with the color h_c (because the model needs to generalize to rules of that form at test time). During testing, the model should be able to combine its understanding of h_c, h_n, and the compositional nature of colors and nouns to understanding either the rule h_n *IS* h_c or h_c h_n *IS* $-$. Figure 2-10 illustrates an example of this task.

Train: $\forall n, c.\ n \neq h_n \wedge c \neq h_c.\ \exists\ t.$ Execute$(t) \vDash c(n)$ s.t. babaPhysics(t)

Test: $\exists\ t.$ Execute$(t) \vDash h_c(h_n)$ s.t. babaPhysics(t)

Dynamic Version: The agent must make the rule $h_c(h_n)$, which translates to either h_n *IS* h_c or h_c h_n *IS* $-$.

Ex6: Conjunctions can be used to create any set of composed components. In this split, we seek to understand how models perform when faced with novel combinations of the following composed components: $\{N(N), A(N), M(N), Y(N), C(N)\}$. In order to generalize to these new scenarios, a model will need to combine its understanding of the compositional structure of the rules with its knowledge of each component. Given the wide range of available components and the flexibility to compose them with few limitations (i.e. combinations must follow the rule grammar described in Section 2.1.1), the Baba environment supports a greater variety of compositions compared to existing benchmarks such as gSCAN. It is important to note that the following tasks use a minimal number of conjunctions for the sake of simplicity, but components can be chained together with multiple conjunctions to create longer, more complex combinations of components.

1. **Any pair of nouns (objects) can transmute into any other noun (object).** This task explores generalization by composing multiple objects transmuting into the same object. For example, the rules *BALL IS WALL* and *KEY IS WALL* means that all balls and all keys will turn into walls. Therefore, the goal of this task is to evaluate a model's ability to compose transmutations.

   (a) **An agent has never seen the specific pair of objects** (h_{n1}, h_{n2}) **transmute into the same object, but is has seen other pairs that include either** h_{n1} **or** h_{n2} **transmute into the same object.** The model can learn how transmutations can be composed in this manner by seeing examples of pairs of objects (not equal to (h_{n1}, h_{n2})) transmute into the same object. It can also ground to the meaning of h_{n1} and h_{n2} because it will see examples of those objects included in the pairs; they just will

not be paired together. Therefore, the model should be able to combine its familiarity with h_{n1}, h_{n2}, and transmutations to generalize to the situation in which the novel pair (h_{n1}, h_{n2}) transmutes into the same object.

It is worth noting that this task can be made even more challenging by never including h_{n1} or h_{n2} in the training transmutation examples. In this case, the agent could still ground to the meaning of h_{n1} or h_{n2} as these objects can still be present in the training environment and can be involved in other rules, but the agent would never see h_{n1} or h_{n2} involved in transmutations. These extensions exist among all of the tasks in this conjunctions split, and we will not cover those extensions here for the sake of clarity.

Train: $\forall n_1, n_2, n_3.\ [n_1 \neq h_{n1} \wedge n_2 \neq h_{n2}] \wedge [n_1 \neq h_{n2} \wedge n_2 \neq h_{n1}] \wedge h_{n1} \neq h_{n2}.$ $\exists\ t.\ \text{Execute}(t) \vDash n_3(n_1) \wedge n_3(n_2)$ s.t. babaPhysics(t)

Test: $\exists\ t.\ \text{Execute}(t) \vDash n_3(h_{n1}) \wedge n_3(h_{n2})$ s.t. babaPhysics(t)

Dynamic Version: The agent must make either the rule $n_3(h_{n1})$ or $n_3(h_{n2})$, which translates to either h_{n1} *IS* n_3 or h_{n2} *IS* n_3, respectively.

(b) **An agent has never seen any pair of objects transmute into the object h_n, but it has seen pairs of objects that include h_n transmute into other objects.** This task is harder than 6.1.a. In 6.1.a, the model has not seen a specific pair transmute into a given object. In this task, the model has never seen any pair transmute into a specific given object h_n. During training, the model can familiarize itself with how pairs of objects can transmute into the same object, and it can also gain exposure to h_n through these transmutation examples. At test time, the model should be able to leverage this knowledge to generalize to the scenario of a novel transmutation of a pair of objects into object h_n.

Train: $\forall n_1, n_2, n_3.\ \ n_3 \neq h_n.\ \ \exists\ t.\ \ \text{Execute}(t) \vDash n_3(n_1) \wedge n_3(n_2)$ s.t. babaPhysics(t)

Test: $\exists\ t.\ \text{Execute}(t) \vDash h_n(n_1) \wedge h_n(n_2)$ s.t. babaPhysics(t)

Dynamic Version: The agent must make either the rule $h_n(n_1)$ or $h_n(n_2)$, which translates to either n_1 *IS* h_n or n_2 *IS* h_n, respectively.

(c) **An agent has never seen any pair of objects transmute into the same object, but it has seen each object transmute into that object individually.** This task is an extension of 1.a, and it is harder than tasks 1.a and 1.b because the model no longer sees examples of combining transmutations during training. It can still ground to the meaning of a transmutation by seeing an individual object transmute into another object. It can also become familiar with all of the objects by seeing them involved in these transmutations. At test time, the model needs to leverage systematic compositionality to realize that the transmutations can be combined. For this task, there is no held out test set since the focus of this task is to measure a model's ability to generalize to longer sequences of transmutations at test time.

Train: $\forall n_1, n_2, n_3.\ h_{n1} \neq h_{n2}.\ \exists\, t.\ \text{Execute}(t) \vDash n_1(n_2)$ s.t. babaPhysics(t)

Test: $\exists\ t.\ \text{Execute}(t) \vDash n_3(n_1) \wedge n_3(n_2)$ s.t. babaPhysics(t)

Dynamic Version: The agent must make either the rule $n_3(n_1)$ or $n_3(n_2)$, which translates to either n_1 *IS* n_3 or n_2 *IS* n_3, respectively.

(d) **An agent has never seen a pair of objects transmute into the same object h_n. It has seen individual transmutations, but these examples do not include an object transmuting into h_n.** This task is an extension of 1.b and 1.c. Similar to 1.c, the model no longer sees examples of coupling transmutations during training. However, the model is able to become familiar with the notation of a transmutation through examples of an individual object transmuting into other object. It can ground to the meaning of h_n by seeing that object transmute into another object (i.e. h_n *IS [N]*), but it never sees an example of an object transmuting into h_n. At test time, the model must compose its understanding of transmutations

and h_n to generalize to a novel composition of transmutations.

Train: $\forall n_1, n_2.\ n_1 \neq h_n.\ \exists\ t.$ Execute$(t) \vDash n_1(n_2)$ s.t. babaPhysics(t)

Test: $\exists\ t.$ Execute$(t) \vDash h_n(n_1) \wedge h_n(n_2)$ s.t. babaPhysics(t)

Dynamic Version: The agent must make either the rule $h_n(n_1)$ or $h_n(n_2)$, which translates to either n_1 *IS* h_n or n_2 *IS* h_n, respectively.

2. **Any pair of nouns (objects) can transmute into any other pair of nouns (objects)**. This task explores generalization by composing the transmutation of multiple different objects into other different objects. While there are many compositional variations that can be explored in this task, we will focus on an edge case that requires the model to understand transitivity. Transitivity is implied by the rules *[N1] IS [N2]* and *[N2] IS [N3]*. For example, the rules *BALL IS WALL* and *WALL IS KEY* implies that all balls, by the transitive property, will become keys.

   (a) **An agent has never seen object h_{n1} transmute into object h_{n2} via transitivity, but it has seen other pairs that include either h_{n1} or h_{n2} transmute into each other via transitivity.** The model can learn how transmutations can be composed in the form of n_1 *IS* n_2 and n_2 *IS* n_3 to convert n_1 into n_3 during training, and it can also ground to the meaning of h_{n1} and h_{n2} because it will see examples of those objects included in these composed transmutations; h_{n1} will just never transmute into h_{n2}. At test time, the model should combine its understanding of h_{n1}, h_{n2}, and the transitive property of transmutations to generalize to the scenario in which h_{n1} transmutes into h_{n2} via a composition of transmutations.

   Train: $\forall n_1, n_2, n_3.\ \ n_1 \neq h_{n1} \wedge n_2 \neq h_{n2} \wedge h_{n1} \neq h_{n2}.\ \exists\ t.$ Execute(t) $\vDash n_3(n_1) \wedge n_2(n_3)$ s.t. babaPhysics(t)

   Test: $\exists\ t.$ Execute$(t) \vDash n_3(h_{n1}) \wedge h_{n2}(n_3)$ s.t. babaPhysics(t)

   Dynamic Version: The agent must make either the rule $n_3(h_{n1})$ or $h_{n2}(n_3)$, which translates to either h_{n1} *IS* n_3 or n_3 *IS* h_{n2}, respectively.

(b) **An agent has never seen any object transmute into another object via transitivity, but it has seen individual object transmutations.** This task is significantly harder than 6.2.a because the model no longer sees examples of transitivity during training. It does, however, see individual examples of transmutations that allow it to ground to the meaning of transmutations and all of the objects in the environment.

Train: $\forall n_1, n_2.\ h_{n1} \neq h_{n2}.\ \exists\ t.$ Execute$(t) \vDash n_1(n_2)$ s.t. babaPhysics(t)

Test: $\exists\ t.$ Execute$(t) \vDash n_1(h_{n1}) \wedge h_{n2}(n_1)$ s.t. babaPhysics(t)

Dynamic Version: The agent must make either the rule $n_1(h_{n1})$ or $h_{n2}(n_1)$, which translates to either h_{n1} *IS* n_1 or n_1 *IS* h_{n2}, respectively.

3. **Any verb (action or termination) can be combined with any pair of nouns (objects).** The goal of this task is to evaluate a model's ability to generalize to situations in which a verb is composed with a novel combinations of objects.

(a) **An agent has never seen a given verb (either a or m) combined with the pair of nouns (h_{n1}, h_{n2}) in a given set of rules, but it has seen that verb combined with other noun pairs that include either h_{n1} or h_{n2}.** Two key aspects of the training set should allow the model to generalize successfully in this scenario. First, even though the agent never sees the pair (h_{n1}, h_{n2}) taking on the same verb during training, it gains exposure to each object h_{n1} and h_{n2} taking on that verb in other pairs during training. Second, the agent learns overall that any pair of objects can take on the same verb by observing other pairs take on the same verb during training. Therefore, the model should be able to generalize to the situation in which a known verb is combined with a novel combination of familiar objects.

Action Verb:

Train: $\forall n_1, n_2.\ [n_1 \neq h_{n1} \wedge n_2 \neq h_{n2}] \wedge [n_1 \neq h_{n2} \wedge n_2 \neq h_{n1}] \wedge h_{n1} \neq h_{n2}.$ $\exists\ t.$ Execute$(t) \vDash a(n_1) \wedge a(n_2)$ s.t. babaPhysics(t)

Test: $\exists\ t.$ Execute$(t) \vDash a(h_{n1}) \wedge a(h_{n2})$ s.t. babaPhysics(t)

Dynamic Version: The agent must make either the rule $a(h_{n1})$ or $a(h_{n2})$, which translates to either h_{n1} *IS* a or h_{n2} *IS* a, respectively.

Termination Verb:

Train: $\forall n_1, n_2.\ [n_1 \neq h_{n1} \wedge n_2 \neq h_{n2}] \wedge [n_1 \neq h_{n2} \wedge n_2 \neq h_{n1}] \wedge h_{n1} \neq h_{n2}.$ $\exists\ t.$ Execute$(t) \vDash m(n_1) \wedge m(n_2)$ s.t. babaPhysics(t)

Test: $\exists\ t.$ Execute$(t) \vDash m(h_{n1}) \wedge m(h_{n2})$ s.t. babaPhysics(t)

Dynamic Version: The agent must make either the rule $m(h_{n1})$ or $m(h_{n2})$, which translates to either h_{n1} *IS* m or h_{n2} *IS* m, respectively.

(b) **An agent has never seen a given verb (either a or m) combined with a pair of nouns in a given set of rules, but it has seen the verb combined with every noun individually.** This task is more challenging than task 6.3.a because although the model has learned that the verb can be used with all objects individually to create a rule, it has never seen an example of a pair of objects both forming a rule with that verb at the same time during training. Thus, our goal with this task is to evaluate a model's ability to generalize to scenarios in which a verb is combined with a larger set of objects than seen during training to create a larger set of rules. Therefore, there is no held out test set for this task.

Action Verb:

Train: $\forall n_1, n_2.\ n_1 \neq n_2.\ \exists\ t.$ Execute$(t) \vDash a(n_1)$ s.t. babaPhysics(t)

Test: $\exists\ t.$ Execute$(t) \vDash a(n_1) \wedge a(n_2)$ s.t. babaPhysics(t)

Dynamic Version: The agent must make either the rule $a(n_1)$ or $a(n_2)$, which translates to either n_1 *IS* a or n_2 *IS* a, respectively.

Termination Verb:

Train: $\forall n_1, n_2.\ n_1 \neq n_2.\ \exists\ t.$ Execute$(t) \vDash m(n_1)$ s.t. babaPhysics(t)

Test: $\exists\ t.$ Execute$(t) \vDash m(n_1) \wedge m(n_2)$ s.t. babaPhysics(t)

Dynamic Version: The agent must make either the rule $m(n_1)$ or $m(n_2)$, which translates to either n_1 *IS* m or n_2 *IS* m, respectively.

4. **Any combination of action verbs or termination verbs can each form a rule with the same noun at the same time**. The objective of this task is to test a model's ability to generalize to situations in which a combination of verbs is used with a noun to form a novel set of rules.

   (a) **An agent has never seen a given set of verbs used with h_n to create a novel set of rules, but it has seen that pair of verbs combined with other nouns**. During training, the agent sees other objects take on this particular combination of verbs, and it can interact with h_n in the environment in order to ground to the meaning of h_n. At test time, we are evaluating a model's ability to generalize when this combination of verbs forms a set of rules with a new object h_n. For this task, we will only evaluate the action verb case. The termination verb case would be the scenario in which the rules are *[N] IS WIN* and *[N] IS LOSE*. Assigning one object to be both the goal and defeat objects would not make sense, so we will not include it in this task.

   Action Verb:

   Train: $\forall n.\ n \neq h_n.\ \exists\ t.$ Execute$(t) \vDash a_1(n) \wedge a_2(n)$ s.t. babaPhysics(t)

   Test: $\exists\ t.$ Execute$(t) \vDash a_1(h_n) \wedge a_2(h_n)$ s.t. babaPhysics(t)

   Dynamic Version: The agent must make either the rule $a_1(h_n)$ or $a_2(h_n)$, which translates to either h_n *IS* a_1 or h_n *IS* a_2, respectively.

   (b) **An agent has never seen a given set of verbs combined with the same noun to form a set of rules, but it has seen each of those verbs individually combined with every noun.** This variant is more challenging than 6.4.a because the agent no longer sees examples of multiple verbs combined with the same noun to form the set of rules during training. However, the individual compositions of verbs and objects during training should provide enough information for the agent to be able

to generalize to this scenario at test time. Similar to 6.4.a, we will only consider the action verb case because assigning both *WIN* and *LOSE* to the same object would not make sense.

Action Verb:

Train: $\forall n, a \in \{a_1, a_2\}.\ \exists\ t.\ \text{Execute}(t) \vDash a(n)$ s.t. babaPhysics(t)

Test: $\exists\ t.\ \text{Execute}(t) \vDash a_1(n) \wedge a_2(n)$ s.t. babaPhysics(t)

Dynamic Version: The agent must make either the rule $a_1(n)$ or $a_2(n)$, which translates to either n *IS* a_1 or n *IS* a_2, respectively.

5. **An agent can control any pair of objects.** An unique feature of the Baba environment is that an agent can control more than one type of object at the same time. Our goal in this task is to leverage this unique aspect of the Baba environment to explore a model's ability to generalize to situations in which the agent controls a novel set of objects.

   (a) **An agent has never controlled the pair of objects** (h_{n1}, h_{n2})**, but it has controlled other pairs that include either** h_{n1} **or** h_{n2}. Given that the agent has gained exposure to controlling h_{n1} and h_{n2} in other pairs during training, and it has gained overall exposure to navigating the Baba environment via two types of objects, the model should be able to generalize to situations in which the agent controls novel combinations of objects (i.e. h_{n1} *IS YOU*, h_{n2} *IS YOU*).

   Train: $\forall n_1, n_2.\ [n_1 \neq h_{n1} \wedge n_2 \neq h_{n2}] \wedge [n_1 \neq h_{n2} \wedge n_2 \neq h_{n1}] \wedge h_{n1} \neq h_{n2}.$ $\exists\ t.\ \text{Execute}(t) \vDash y(n_1) \wedge y(n_2)$ s.t. babaPhysics(t)

   Test: $\exists\ t.\ \text{Execute}(t) \vDash y(h_{n1}) \wedge y(h_{n2})$ s.t. babaPhysics(t)

   Dynamic Version: The agent must make either the rule $y(h_{n1})$ or $y(h_{n2})$, which translates to either h_{n1} *IS YOU* or h_{n2} *IS YOU*, respectively.

   (b) **An agent has never controlled any pair of objects, but it has controlled each object individually**. This task is incrementally harder than task 6.5.a because although the model controls all objects during

training, it only controls them individually, so it never gains exposure to controlling two objects at once. Therefore, in this variation of the task, we are exploring a model's ability to generalize to situations in which it controls multiple objects at once.

Train: $\forall n_1, n_2.\ \exists\ t.\ \text{Execute}(t) \vDash y(n_1)$ s.t. $\text{babaPhysics}(t)$

Test: $\exists\ t.\ \text{Execute}(t) \vDash y(n_1) \wedge y(n_2)$ s.t. $\text{babaPhysics}(t)$

Dynamic Version: The agent must make either the rule $y(n_1)$ or $y(n_2)$, which translates to either n_1 *IS YOU* or n_2 *IS YOU*, respectively.

6. **Any adjective (color) can be combined with any pair of nouns (objects) at the same time to form a set of rules.** The objective of this task is to test an agent's ability to generalize to scenarios in which an adjective is composed with novel combinations of nouns in the rule set.

   (a) **An agent has never seen a given color h_c combined with the pair of nouns (h_{n1}, h_{n2}) in the same set of rules, but it has seen that color h_c composed with other noun pairs that can include either h_{n1} or h_{n2}.** During training, the agent can become familiar with h_c by seeing examples of h_c combined with other object pairs, and it can become familiar with h_{n1} and h_{n2} by observing these objects involved in rules with other colors. It can also learn the compositional nature of colors and objects. Therefore, the goal of this task is to evaluate a model's ability to generalize to the situation in which a known color is composed with a novel set of objects.

   Train: $\forall n_1, n_2, c.\ [n_1 \neq h_{n1} \wedge n_2 \neq h_{n2}] \wedge [n_1 \neq h_{n2} \wedge n_2 \neq h_{n1}] \wedge h_{n1} \neq h_{n2} \wedge h_c \neq c.\ \exists\ t.\ \text{Execute}(t) \vDash c(n_1) \wedge c(n_2)$ s.t. $\text{babaPhysics}(t)$

   Test: $\exists\ t.\ \text{Execute}(t) \vDash h_c(h_{n1}) \wedge h_c(h_{n2})$ s.t. $\text{babaPhysics}(t)$

   Dynamic Version: The agent must make either the rule $h_c(h_{n1})$ or $h_c(h_{n2})$.

   (b) **An agent has never seen a given color combine with two nouns**

to form two rules at the same time, but it has seen each color combine with each noun to form one rule at a time. This task is harder than task 6.6.a because even though the model still gains an understanding of all of the colors and objects in the environment, it never sees pairs of objects taking on the same verb during training. We seek to test an agent's ability to generalize to a larger set of color-object rules than seen during training.

Train: $\forall n_1, n_2, c.\ n_1 \neq n_2.\ \exists\ t.\ \text{Execute}(t) \vDash c(n)$ s.t. babaPhysics(t)

Test: $\exists\ t.\ \text{Execute}(t) \vDash c(n_1) \wedge c(n_2)$ s.t. babaPhysics(t)

Dynamic Version: The agent must make either the rule $c(n_1)$ or $c(n_2)$.

7. **Any combination of colors can be combined with any single noun to form a set of rules.** The objective of this task is to investigate an agent's ability to generalize to rules defined by a novel combination of colors composed with a particular object to form a novel set of rules. It is important to note that given the two colors are being composed with a singular noun, we can only use two rule structures for this case: 1) *([C] [N] IS -)* and *([C] [N] IS -)* or 2) *([C] [N] IS -)* and *([N] IS [C])*. It would not make sense to have two rules structured as *([N] IS [C])* and *([N] IS [C])* (e.g. *KEY IS RED* and *KEY IS BLUE*) at the same time.

   (a) **An agent has never seen the pair of colors (h_{c1}, h_{c2}) combined with h_n to form two rules, but it has seen that pair of colors combined with other objects to form two rules.** The agent should gain familiarity with the test set colors and object as well as the notion that any set of colors can be combined with any set of objects by observing pairs of colors that include h_{c1} and h_{c2} applied to other objects and h_c involved in rules with other colors. At test time, the agent should be able to combine these known elements.

   Train: $\forall n, c_1, c_2.\ n \neq h_n \wedge [c_1 \neq h_{c1} \wedge c_2 \neq h_{c2}] \wedge [c_1 \neq h_{c2} \wedge c_2 \neq h_{c1}].\ \exists\ t.\ \text{Execute}(t) \vDash c_1(n) \wedge c_2(n)$ s.t. babaPhysics(t)

Test: $\exists\ t.$ Execute$(t) \vDash c_1(h_n) \wedge c_2(h_n)$ s.t. babaPhysics(t)

Dynamic Version: The agent must make either the rule $c_1(h_n)$ or $c_2(h_n)$.

(b) **An agent has never seen the pair of colors (h_{c1}, h_{c2}) combine with a given noun n to form two rules at the same time, but it has seen the colors applied to each noun, including n, to form one rule.** This task is more challenging than 6.7.a because now the agent no longer sees examples of multiple colors describing an object at the same time. However, the agent should still learn the necessary components through the individual color-object combinations. Therefore, at test time, the agent should have the knowledge to generalize to a novel composition of color-object pairs.

Train: $\forall n, c.\ h_{c1} \neq h_{C2}.\ \exists\ t.$ Execute$(t) \vDash c(n)$ s.t. babaPhysics(t)

Test: $\exists\ t.$ Execute$(t) \vDash h_{c1}(n) \wedge h_{c2}(n)$ s.t. babaPhysics(t)

Dynamic Version: The agent must make either the rule $h_{c1}(n)$ or $h_{c2}(n)$.

8. **Any number of number of noun transmutations, noun-verb combinations, noun-pronoun combinations, and noun-adjective combinations can be composed together to form longer chains of rules.** The goal is to investigate a model's ability to combine learned composed components to generalize to novel combinations of rules from the following set of components: $\{N(N), Y(N), A(N), M(N), C(N)\}$. For the below task variations, we will define property I and property J as follows: $I, J \in \{N, Y, A, M, C\}$.

(a) **An agent has never seen the pair of properties (I, J) combined with the pair of nouns (h_{n1}, h_{n2}), respectively. However, it has seen this pair of properties combined with other pairs of nouns that can include h_{n1} or h_{n2}, including the scenario $(I(h_{n2}), J(h_{n1}))$.** During training, the model can learn the meaning of the properties I and J through examples of other objects taking on these properties. It can ground to the meaning of h_{n1} and h_{n2} by navigating environments where $I(h_{n2})$ or $J(h_{n1})$. The goal, therefore, is to evaluate a model's ability to generalize

to the scenario in which it has seen the combination of properties as well as the combination of nouns, but they are composed in a novel manner.

Train: $i, j \in \{N, Y, A, M, C\}, \forall n_1, n_2.\ \ n_1 \neq h_{n1} \wedge n_2 \neq h_{n2} \wedge h_{n1} \neq h_{n2} \wedge i \neq j.\ \exists\ t.$ Execute$(t) \vDash i(n_1) \wedge j(n_2)$ s.t. babaPhysics(t)

Test: $\exists\ t.$ Execute$(t) \vDash i(h_{n1}) \wedge j(h_{n2})$ s.t. babaPhysics(t)

Dynamic Version: The agent must make either the rule $i(h_{n1})$ or $j(h_{n2})$.

(b) **An agent has never the pair of properties (I, J) combined with same noun h_n to form two rules at the same time.** The agent can gain familiarity with the properties I and J through examples of other objects taking on these properties. It can also ground to the meaning of h_n because it can still be in training environments and can be involved in other rules. At test time, the agent is required to compose its understanding of (I, J) and h_n to generalize to a novel combination of properties combined with a given noun.

Train: $i, j \in \{N, Y, A, M, C\}, \forall n.\ \ n \neq h_n \wedge i \neq j.\ \ \exists\ t.$ Execute(t) $\vDash i(n) \wedge j(n)$ s.t. babaPhysics(t)

Test: $\exists\ t.$ Execute$(t) \vDash i(h_n) \wedge j(h_n)$ s.t. babaPhysics(t)

Dynamic Version: The agent must make either the rule $i(h_n)$ or $j(h_n)$.

(c) **An agent has never seen property I combined with h_{n1} nor property J combined with h_{n2} together, but it has seen these nouns composed with those properties individually.** This task is harder than tasks 6.8.a and 6.8.b because the agent no longer sees combinations of these properties during training. However, the model gains exposure to $I(h_{n1})$ and $J(h_{n2})$ individually. At test time, we can evaluate how well a model can combine this previous information to understand how to navigate an environment in which these components appear together.

Train: $i, j \in \{N, Y, A, M, C\}, \forall n_1, n_2.\ \ i \neq j.\ \ \exists\ t.$ Execute$(t) \vDash i(n_1) \vee$ $j(n_1)$ s.t. babaPhysics(t)

Test: $\exists\ t.$ Execute$(t) \vDash i(n_1) \wedge j(n_2)$ s.t. babaPhysics(t)

Dynamic Version: The agent must make either the rule $i(n_1)$ or $j(n_2)$.

2.2.3 The Relationship between Baba and gSCAN Benchmarks

We have designed the Baba benchmark to address key issues with the gSCAN benchmark. However, before we discuss the key improvements with our benchmark, it is important to note that we include meaningful similarities. To start, both gSCAN and Baba have nouns, adjectives, and verbs. These components can be composed to form novel noun-verb pairs and adjective-noun pairs. With this overlap in components and component composition, we are able to replicate a gSCAN split in the Baba environment. In particular, we can replicate the *Novel Composition of Object Properties* split, which explores an agent's ability to identify novel combinations of colors and objects, in our split *Ex 5.1: An agent has never seen h_c combined with h_n.* The ability to reproduce a gSCAN split in the Baba environment serves as an important reference point that bridges the two benchmarks. This shared data split verifies the validity of our environment as a tool to explore compositional generalization.

Overall, the gSCAN and Baba benchmarks differ along three key axes: compositional complexity, data split development, and rule set dynamics. First, we include a richer set of environment components, which leads to more challenging compositional structures that can be explored in the Baba environment. Unlike gSCAN, the Baba environment has the pronoun *YOU*, which creates a new type of composition not seen in the gSCAN benchmark as the agent must leverage its understanding of the objects and the pronoun *YOU* to make generalizations about itself. Transmutation is also unique to the Baba environment, which creates a novel level of challenge as the objects themselves can change throughout their time navigating the environment. We also include the verbs *OPEN, SHUT* and *DEFEAT* to create a greater variety of interactions between objects in the environment. We do not include adverbs and directions, which are present in the gSCAN environment, because we have prioritized elements that create novel generalization challenges. After establishing the point of connection between gSCAN and the Baba benchmarks with the color-noun combinations, we have worked to create a set of components that present greater generalization

challenges.

Next, we develop our data splits and tasks using the CFG, which allows us to ensure three key properties. First, the tasks do not overlap. Second, they are as exhaustive as possible without losing clarity. Third, they display an increasing level of complexity. This set of tasks allows anyone to use all features of the rich Baba environment to systematically test an agent's ability to combine known concepts to adapt to novel scenarios. The gSCAN benchmark data splits, however, are ad-hoc. They overlap in terms of the compositionality being explored, and there is no sense of how these tasks compare to each other in terms of the level of complexity. The presented gSCAN tasks are not exhaustive, and there is no rationale behind why the defined tasks are the most important compositional tasks to explore in the environment.

Finally, the Baba environment supports static and dynamic sets of rules (or a combination of both depending on environment structure), while the gSCAN benchmark only supports a static set of rules. Since we can evaluate both scenarios in which an agent 1) just needs to compose known components to generalize to a new set of rules and 2) an agent needs to *make* new set of rules, we can begin to measure the effect of a dynamic rule set on an agent's ability to generalize.

2.3 Environment Generation

The goal of the Baba benchmark is to test a model's ability to generalize by systematically running it on the tasks defined in Section 2.2.2. In order to be able to use this benchmark efficiently, we need to be able to seamlessly translate the defined task to the experimental train and test environments. Therefore, we have developed an automated system to generate these train and test environments. In general, an environment is defined in our code repository by a config file. This file specifies both the types of objects and text blocks present in the environment as well as the location of these elements. Hydra allows us to compose these config files. Therefore, we have created a set of standard config files to set up the baseline environment details. We

then create a new config file that specifies objects, rules, and conditions for a particular task, and we combine this file with the baseline files to generate training and testing environments. Figure 2-11 illustrates this process.

Overall, the translation from task definition to config file is relatively simple. We can specify the rules of the config file based on the rules that are defined in the task definition. We translate this notion of held out objects by specifying conditions that dictate how environments can be sampled for training and testing. We will discuss the task objective in greater detail in Chapter 3, but the task depends on whether we are running the given experiment with a SL or RL model. For instance, if we are running an SL model on this particular environment in Figure 2-11, the task would be to output the location of the goal object. In the RL case, the task would be for the agent to navigate to the goal object.

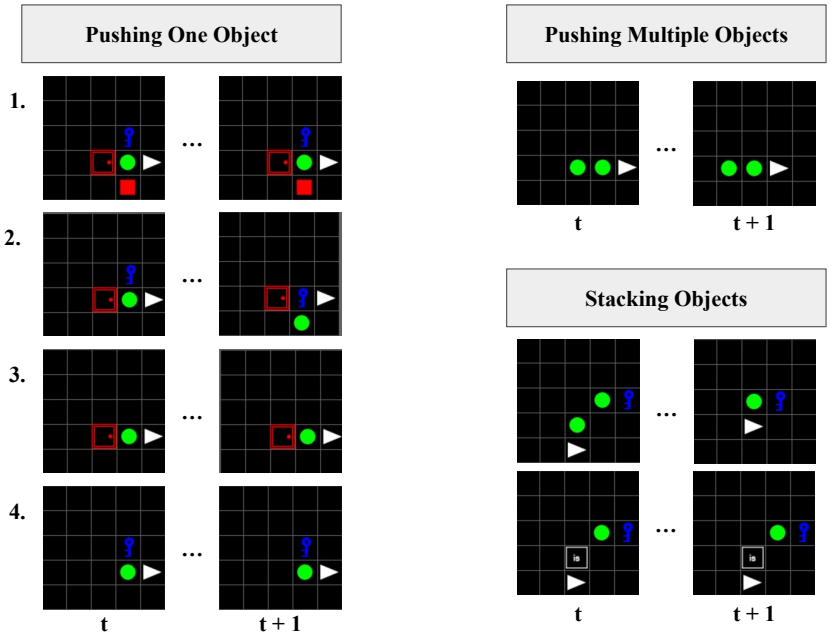


Figure 2-5: **Pushing objects can lead to several types of conflicts.** First, it is important to check for scenarios in which multiple objects are trying to push a single object at the same time. The four cases that we must consider are demonstrated under the heading *Pushing One Object*. In the first case, a door, a key, Baba, and a wall are pushing a ball from all four sides at time t. At time $t + 1$, none of the objects will have moved because the forces of the objects directly across from each other cancel out. More concretely, the forces from the key and the wall cancel each other out. Similarly, the forces from Baba and the door cancel out. In the second case, a door, key and Baba are pushing a ball from three sides. At time $t + 1$ the pushing forces of the door and Baba will cancel out (similar to the first case), so the key will push the ball. In the third case, the pushing forces of the door and Baba cancel each other out, so none of the objects move at time $t+1$. Finally, in the fourth case, it would make sense that the key and Baba would push the ball diagonally (to the left and down). However, since diagonal movements are not permitted in this environment, the result of the key and Baba pushing the ball from adjacent sides will result in no movement at time $t + 1$. It is also important to check for cases in which an object such as Baba is pushing two adjacent objects as shown under the heading *Pushing Multiple Objects*. In this case, both balls should be pushed by Baba. Lastly, it is critical to check for cases in which objects are stacked, which is shown under the heading *Stacking Objects*. When two objects such as two balls are being pushed onto the same grid space at the same time, they should be stacked on top of each other. However, text blocks cannot be stacked. Therefore, if one or more of those balls is replaced by a text block, then neither the text blocks nor the objects will move onto that space. Overall, when multiple objects are moving around in the environment, the majority of push conflicts can be broken down into these base case scenarios.

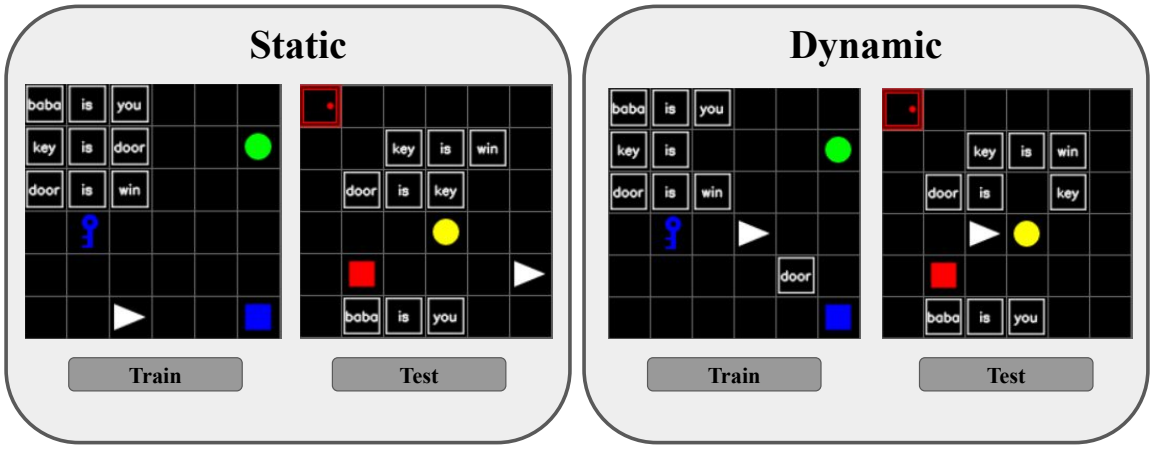


Figure 2-6: **The task *an agent has never seen* h_{n1} *transmute into* h_{n2} can be explored in both static and dynamic rule environments.** The left set of images shows an example of the static case. In this example, $h_{n1} = DOOR$ and $h_{n2} = KEY$. So at test time, the agent must understand that a door is transmuting into a key. Given the current test environment set up (*BABA IS YOU, KEY IS WIN*), the agent must navigate to the door object (which is really a key) in order to complete the task. During training, the agent will see other examples of transmutations that can include objects h_{n1} and h_{n2} such as the given example *KEY IS DOOR*. During training, the agent must also understand this transmutation to then navigate to the goal object. On the right side of this figure is the dynamic case. This case is identical to the static case except the agent must now modify the set of rules in order to complete the task. In this example, the agent must make the transmutation rule.

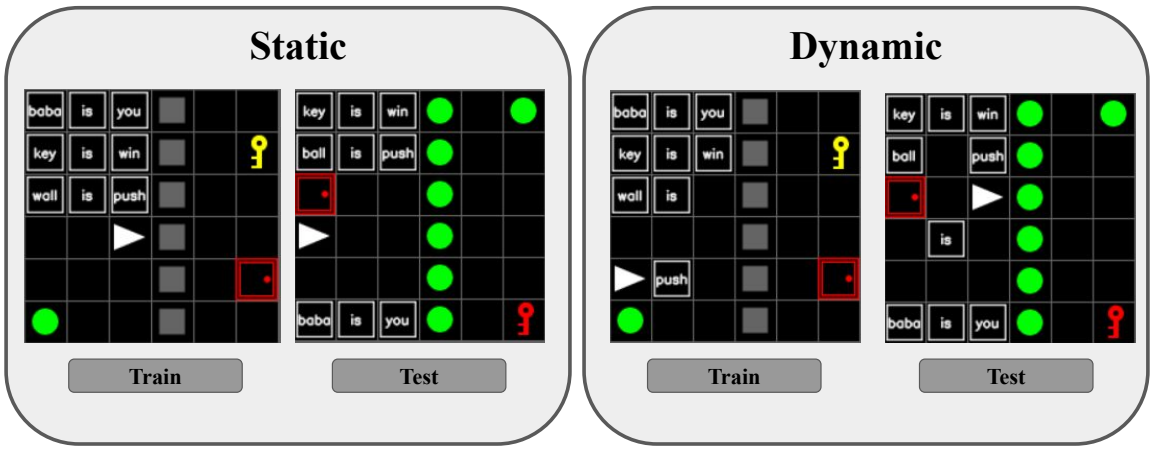


Figure 2-7: **We can investigate the task *an agent has never seen action* a *with object* h_n *in the same rule* in both a static and dynamic setting.** We will start with the static case on the left. In this set up, action $a = PUSH$ and $h_a = BALL$, so the agent must generalize to the scenario with the novel rule *BALL IS PUSH* at test time. Both train and test environments have the rule structure *BABA IS YOU, KEY IS WIN*, and *[N] IS PUSH*. The agent must push the objects in order to reach the key. In the training environment example, the agent must push the wall to be able to reach the key. It will also see examples of other objects such as a door involved in the rule *[N] IS PUSH*. At test time, the agent must push the ball in order to reach the goal object. The dynamic case is identical to the static case except the agent must now make the rule *[N] IS PUSH* in order to complete the task.

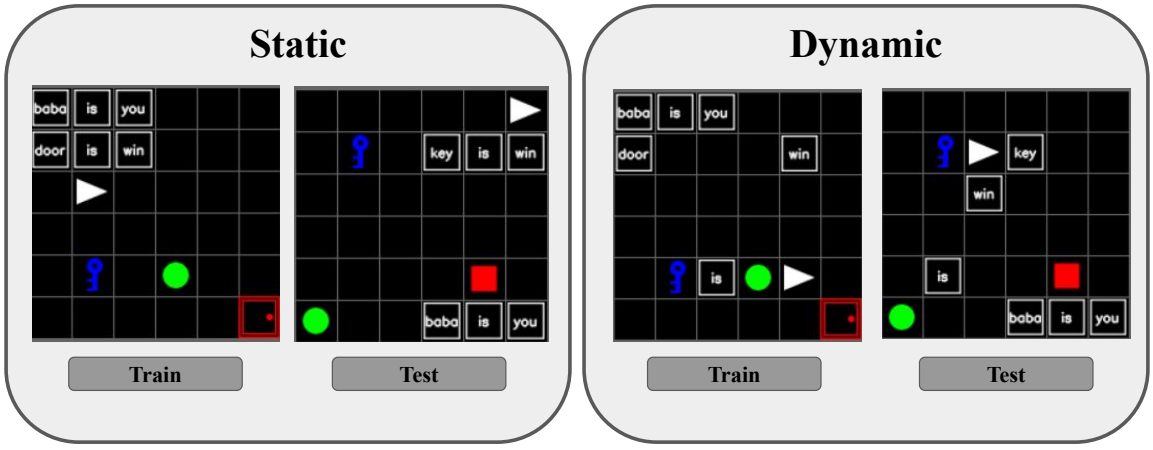


Figure 2-8: **We can evaluate an agent's performance on the task *an agent has never seen* m *combined with* h_n in both a static and dynamic case in the Baba environment.** We will first discuss the static case. In this environment, the termination verb $m = WIN$, and the $h_n = KEY$. In both the training and testing environments, the rules are *BABA IS YOU* and *[N] IS WIN*. During training, the agent will see examples of other objects in this win rule such as *DOOR IS WIN*, and it must understand the rule and then navigate to the goal object. At test time, the agent needs to generalize to the scenario in which *KEY IS WIN*. The dynamic case is similar to the static case except the agent must now make the rule *[N] IS WIN* and then navigate to the goal object in order to complete the task.

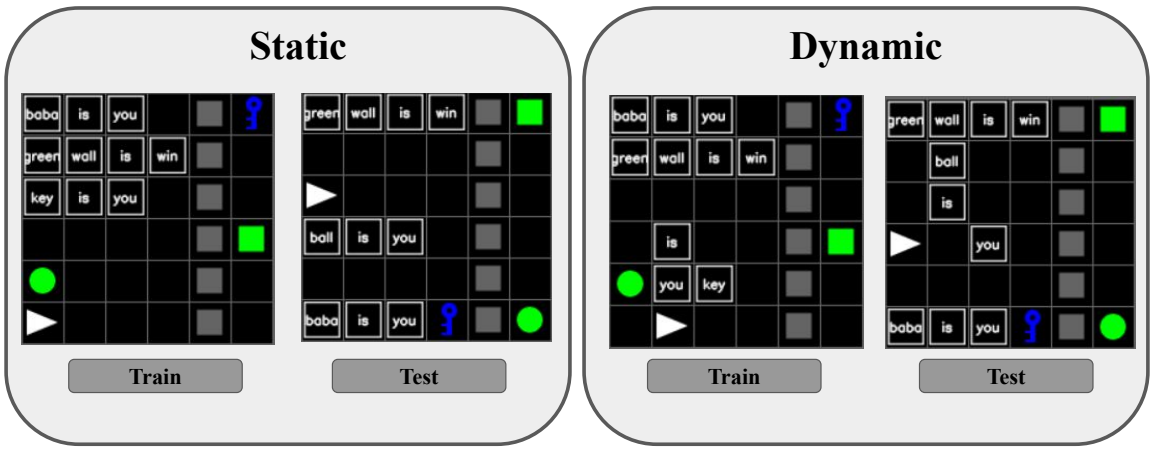


Figure 2-9: **We can investigate the generalization task *an agent has never controlled object* h_n in the Baba environment in both a static and dynamic scenario.** In this set up, $h_n = BALL$. Across both training and testing environments, the set of rules is *BABA IS YOU, GREEN WALL IS WIN*, and *[N] IS YOU*. Given that a wall is blocking Baba from the goal object, the agent must understand that it also controls a second object, which is on the same side of the wall as the goal object (a green wall), and use that second object to navigate to the green wall block. At test time, the agent must recognize the novel rule *BALL IS YOU* and use the ball to touch the green wall. The dynamic case is similar to the static case except the agent needs to make the rule *[N] IS YOU*.

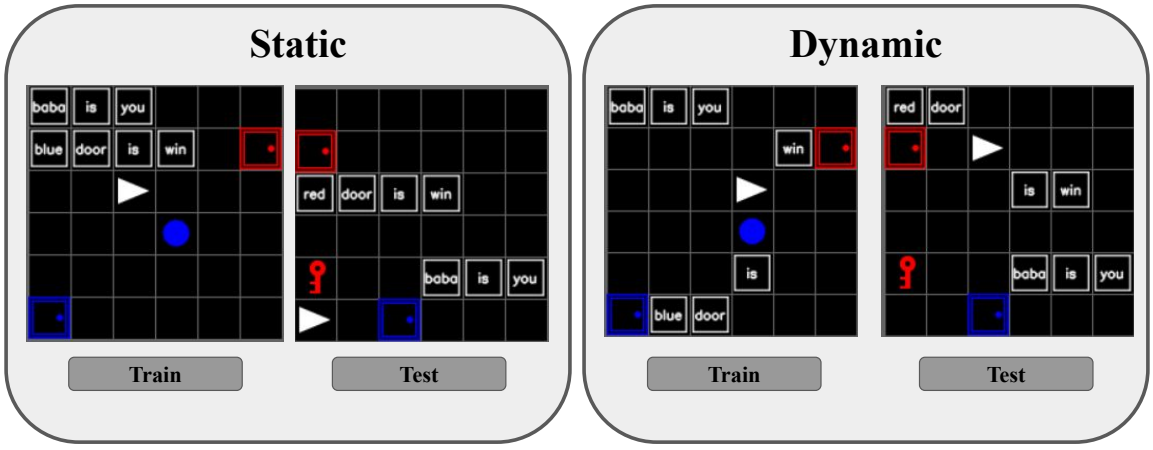


Figure 2-10: **The Baba environment can be used to explore the task *an agent has never seen* h_c *combined with* h_n in both a static and dynamic setting.** We will first analyze the static case. In this set up, $h_c = RED$ and $h_n = DOOR$. The set of rules across both the training and testing environments are *BABA IS YOU* and *[C] [N] IS WIN*. During testing, the agent will see different adjective-noun pairs used in the rule *[C] [N] IS WIN*, and then it must touch the goal object defined by that rule. At test time, the agent must generalize to the novel adjective-noun pair *RED DOOR* and navigate to that object. The dynamic case is identical to the static case except the agent must now make the rule *[C] [N] IS WIN* and then navigate to the goal object.

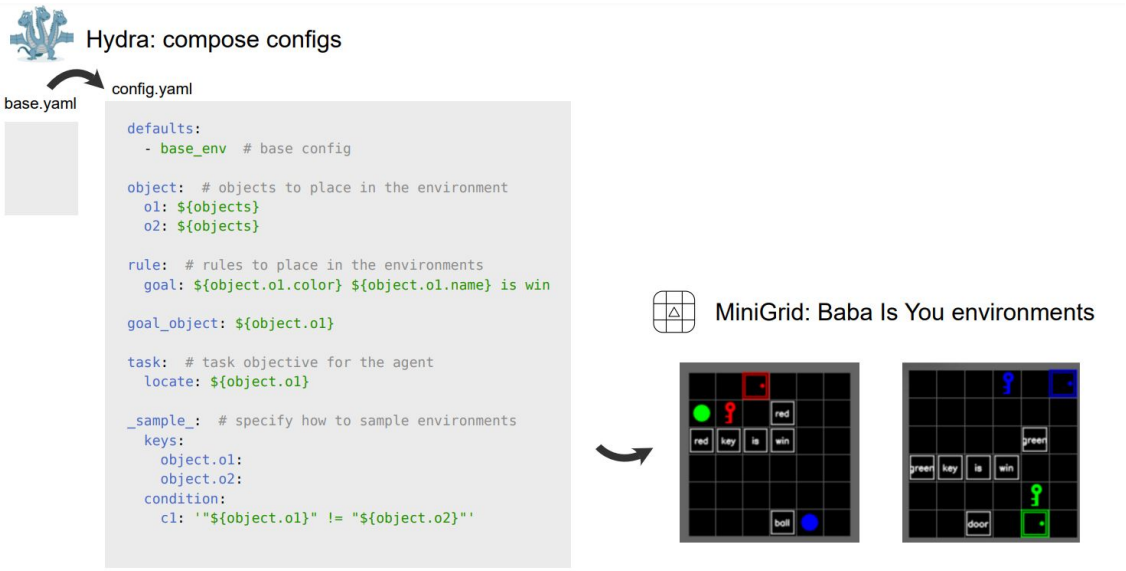


Figure 2-11: **We use config files to translate the tasks outlined in Section 2.2.2 to training and testing environments for experiments.** Hydra enables us to compose config files, so we have developed a set of baseline config files to set up the general environment details such as grid size and the positions of objects and text blocks in the environment. We will layer those config files with another config file that specifies the critical experiment details for that particular task. In this example, we are generating an environment for task 5.1, *an agent has never seen h_c combined with h_n to form a rule.* We specify that there must be two objects o_1 and o_2. The rule being tested is *[C] [N] IS WIN* where *[C]* refers to the color of o_1 and *[N]* refers to the object type of o_1, so the goal object is o_1. In this case, the color of o_2 is equal to h_c and the object type of o_2 is equal to h_n. Therefore, we specify that the difference between the training and testing environments is the condition that $o_1 \neq o_2$. To give a concrete example, o_1 refers to *RED KEY*, and o_2 refers to *GREEN KEY*.

Chapter 3

Experiments and Results

The goal of the Baba benchmark is to test a model's ability to perform rule-based generalization using systematic compositionality. In Chapter 2, we discuss the Baba environment, the set of tasks that can be used to evaluate a wide range of compositional properties, and the ability to translate the CFG-defined tasks into config files that automatically generate training and testing environments. Therefore, in this chapter, we will discuss our experimental design. The goal of these experiments is twofold. First, we will prove the validity of our environment by reproducing a gSCAN split in the Baba environment and achieving similar modeling results. Second, we will show how this benchmark can be used to systematically measure the generalization capabilities of both SL and RL agents as well as the affect of dynamic rule sets on a model's performance.

3.1 Task Selection for Experimentation

In Section 2.2.2, we outline a robust set of tasks to evaluate various facets of compositionality with increasing degrees of difficulty. Given time constraints, we select a subset of these tasks for experimentation. To guide this selection, we develop three key criteria. First, the subset must include a task that overlaps with the gCSAN data splits in order to prove the credibility of the Baba environment. Second, the subset of experiments must cover each of the six splits (N(N), Y(N), A(N), M(N),

C(N) and conjunctions). Third, this subset of experiments should be comprised of the relatively simpler tasks. In order to better understand model failures, it is important to run experiments in increasing order of task difficulty. Thus, we must start out by testing the simplest cases. Using these three criteria to inform our task selection for experimentation, we have developed a comprehensive subset of eight tasks, which we will discuss in detail in the following section.

3.1.1 Experimental Task Subset

We will refer to each task in this subset as a *challenge*. For each challenge, we will specify the task that it corresponds to in the Baba benchmark task definition section (Chapter 2 Section 2.2.2). Then, we will briefly summarize the motivation behind selecting that particular task and the core ideas of compositionality that are being tested.

Challenge 1: An agent has never seen h_c combined with h_n in the rule construction *[C] [N] IS WIN*. The agent must generalize to a novel color-noun combination, which is represented by the task 5.1 in the Baba benchmark (Section 2.2.2). In addition, it must also generalize to the combination of *WIN* with this new color-object pair. Most importantly, this challenge is identical to the *Novel Composition of Object Properties* data split in the gSCAN benchmark, so it serves as an important point of reference that links the two benchmark. The *training* environment can be defined by the expression: $\forall n, c.\ n \neq h_n \wedge c \neq h_c \wedge m = WIN.\ \exists\ t.$ Execute(t) $\vDash m(c(n))$ s.t. babaPhysics(t). The subsequent *testing* environment can be defined by the expression: $\exists\ t.$ Execute(t) $\vDash m(h_c(h_n))$ s.t. babaPhysics(t).

Challenge 2: An agent has never seen action verb a combined with noun h_n to form a rule. This challenge represents the simplest approach to measuring the composition of a given action and a novel object (task 2.1 in Section 2.2.2). The *training* environment can be defined by the expression $\forall n.\ n \neq h_n.\ \exists\ t.$ Execute(t) $\vDash a(n)$ s.t. babaPhysics(t). The corresponding *testing* environment can be defined

by the expression: $\exists\ t.$ Execute$(t) \vDash a(h_n)$ s.t. babaPhysics(t).

Challenge 3: An agent has never seen h_c combined with h_n, and it has never seen *PUSH* composed with this novel color-object combination to form a rule. This challenge is similar task 5.1 in section 2.2.2 as the agent must generalize to this novel composition of colors and objects and then apply *PUSH* to this color-object combination. The *training* environment can be defined by the expression: $\forall n, c.\ n \neq h_n \wedge c \neq h_c \wedge a = PUSH.\ \exists\ t.$ Execute$(t) \vDash a(c(n))$ s.t. babaPhysics(t). The subsequent *testing* environment can be defined by the expression: $\exists\ t.$ Execute(t) $\vDash a(h_c(h_n))$ s.t. babaPhysics(t).

Challenge 4: An agent has never seen h_{n1} transmute into h_{n2}. This challenge is one of the simplest generalization scenarios that tests an agent's ability to understand a novel transmutation of objects. This case is identical to task 1.1 in Section 2.2.2, so the *training* environment can be defined by the expression $\forall n_1, n_2.$ $n_1 \neq h_{n1} \wedge n_2 \neq h_{n2}.\ \exists\ t.$ Execute$(t) \vDash n_2(n_1)$ s.t. babaPhysics(t), while the *testing* environment can be expressed as $\exists\ t.$ Execute$(t) \vDash h_{n2}(h_{n1})$ s.t. babaPhysics(t).

Challenge 5: An agent has never seen h_n transmute into another object. This challenge corresponds to the task 1.2 in Section 2.2.2, and it is another relatively simple evaluation of a model's ability to generalize to novel transmutations. It is incrementally harder than experiment 4 because now the model has never seen any examples of h_n transmuting into another object during training. The *training* environment is given by the expression $\forall n_1, n_2.\ n_1 \neq h_n.\ \exists\ t.$ Execute$(t) \vDash n_2(n_1)$ s.t. babaPhysics(t), and the *testing* environment can be defined as $\exists\ t.$ Execute(t) $\vDash n_2(h_n)$ s.t. babaPhysics(t).

Challenge 6: An agent has never seen an object transmute into h_n. This challenge is equivalent to the Baba benchmark task 1.3 in Section 2.2.2, and it is another simple scenario to challenge an agent's ability to generalize to novel transmutations where it has never seen any object transmute into h_n The *training* envi-

ronment is defined by $\forall n_1, n_2.\ n_2 \neq h_n.\ \exists\ t.$ Execute$(t) \vDash n_2(n_1)$ s.t. babaPhysics(t). The *testing* environment is given by the expression $\exists\ t.$ Execute$(t) \vDash h_n(n_1)$ s.t. babaPhysics(t).

Challenge 7: An agent has never before controlled multiple of the same object n. This challenge explores an agent's ability to control a larger set of objects at test time compared to training. In this case, the agent has controlled the objects in this set individually. This challenge corresponds to task 6.5.b from Section 2.2.2. The *training* environment is given by the expression $\forall n_1, n_2.\ \exists\ t.$ Execute$(t) \vDash y(n_1)$ s.t. babaPhysics(t) while the *testing* environment is defined as $\exists\ t.$ Execute$(t) \vDash y(n_1) \wedge y(n_2)$ s.t. babaPhysics(t).

Challenge 8: An agent has never before controlled object h_n. This challenge is harder than experiment 7 because now the agent does not gain a familiarity with controlling h_n. This scenario corresponds to task 4.1 in Section 2.2.2. The *training* environment can be expressed as $\forall n.\ n \neq h_n.\ \exists\ t.$ Execute$(t) \vDash y(n)$ s.t. babaPhysics(t), while the *testing* environment can be given by the expression $\exists\ t.$ Execute$(t) \vDash y(h_n)$ s.t. babaPhysics(t).

3.2 Models

The Baba benchmark can be used to evaluate both SL and RL models. Given that the Baba environment is inherently a spatial navigation task, this benchmark lends itself to RL experiments. However, the agent's objective can be modified to fit an SL structure. Depending on the particular challenge, the SL model can be tasked with predicting the next state of the environment or the position of a particular object. It is worth noting that predicting the next state of the environment is significantly more challenging than predicting an object's position as the agent needs to correctly account for everything in the environment rather than focusing on one object in the environment.

Ideally, we would like to run four models on the set of challenges described in Section 3.1.1: a Transformer, a Convolutional Neural Network (CNN), PPO, and a Deep Q-Network (DQN). This set of models could provide insight into how both SL models (Transformer and CNN) and RL models (PPO and DQN) perform on our set of generalization challenges. However, due to time constraints, we will only discuss our work with running a transformer and PPO on our benchmark, and we will leave the CNN and DQN to future work. In this section, we will discuss the model implementation that we will use for the SL and RL experiments.

3.2.1 Supervised Learning

In terms of our choice of transformer model implementation, recent results have been published by Ankur Sikarwar showcasing the strong performance of his transformer model RefEx on the gSCAN data split *Novel Composition of Object Properties* [44], which is equivalent to Challenge 1 in our experimental subset of tasks. Given that a primary goal of these experiments is to verify the validity of the Baba environment, we will work to reproduce this result in our environment. Therefore, for our SL experiments, we will use the RefEx transformer model implementation outlined in Sikarwar's recent paper [44].

The overall structure of the model is an attention-only transformer with three layers and four attention heads per layer. The goal of the model is to take in the observed grid world state and output either a prediction of the grid world state at the next time step or a location of an object. This process begins with a one-hot representation of the Baba environment state. Each grid space is represented by a one-hot encoding of the object(s) or text block that currently occupies that space. We do not provide the rules as an additional command, so the model must extract the set of rules for that environment from the environment embedding. This one-hot encoding of the environment is processed by each layer in the transformer. The representation outputted by the final layer is then mapped to logits. A softmax of these logits is taken in order to produce the final output, which is either a one-hot encoding of the next state of the environment or the coordinate position of a particular

object. The parameters that are learned in this model are the query, key, value, and output matrices of the different attention layers.

3.2.2 Reinforcement Learning

Given that the interaction between the agent and the environment can be formulated as an MDP, the Baba environment is well-structured to support RL experiments. Additional details such as reward and transition state definitions can be found in Chapter 2 Section 1.2.

We use PPO for our RL experiments because it is widely considered state-of-the-art in RL. PPO was introduced in 2017 in an attempt to stabilize an agent's training by preventing large policy updates. It strikes a relatively good balance between exploration and exploitation, which is an ongoing challenge in RL. We will use the Stable Baselines3 implementation of PPO with a CNN policy and Stable Baselines3's set of hyperparameters for our experiments [43].

Similar to the SL models, the RL model will take a one-hot encoding of the current Baba environment state as an input. It must parse the grid world and extract the set of rules from this one-hot encoding of the environment. Based on the optimal policy learned during training, the model outputs the set of actions in an attempt to earn the largest reward possible.

3.3 Experiments

In this section, we will outline the experiments performed for each type of model. For each experiment, we will describe the agent's task, explain the training and testing environments, and link the experiment to the set of challenges outlined in the experimental task subset (Section 3.1.1). It is important to note that ideally, we would create environments with both static sets of rules and dynamic sets of rules for each challenge. The purpose of this structure would be to measure the effect that a dynamic rule set has on an agent's ability to generalize to novel sets of rules. However, given the time constraints, we will focus on the static rule set case and

leave the dynamic rule set case for future work. We will evaluate the static case first because it is important to start by measuring an agent's ability to generalize to new rule sets before introducing the challenge of having the ability to modify rules.

3.3.1 Supervised Learning Experiments

Experiment 1:

- **Challenge 1:** $h_c = RED$, $h_n = BALL$
- **Rules:** *BABA IS YOU*, *[C] [N] IS WIN*
- **Training and Testing Environments:** There is one *BABA* and one object of type *[N]* and of color *[C]*. All environments for all SL experiments contain distractor objects, which are objects that are not involved in the rule [C] [N] IS WIN. The type, color, and location of these objects are randomized in the environment. In this scenario, red balls can be present in the training environment, but they are never goal objects.
- **Agent Objective:** Given the environment, the agent must predict the location of the goal object *[C] [N]*. Thus, at test time, the agent must predict the location of the red ball.

Experiment 2:

- **Challenge 2:** h_n *BALL*, $h_a = PUSH$
- **Rules:** *BABA IS YOU, [N] IS [A]*
- **Training and Testing Environments:** The agent is next to and directly below the object *[N]*. There is only one *BABA* and only one object *[N]*. Other distractor objects are placed in the environment randomly.
- **Agent Objective:** The agent will be given an encoding of the current grid state and will be told that the action taken by the agent is up. The agent must

then predict the next grid state. The idea behind this set up is that if the object located directly above the agent is *PUSH*, then the agent should recognize that moving up one space will cause that object to also move up one space. At test time, the agent should correctly identify that the *BALL* will move up one grid space as it has been pushed by the agent.

Experiment 3:

- **Challenge 3:** $h_c = RED$, $h_n = BALL$
- **Rules:** *BABA IS YOU, [C] [N] IS PUSH*
- **Training and Testing Environments:** Similar to the previous experiment, the agent is directly below the object in the rule *[C] [N] IS PUSH*. There is only one *BABA* and only one object of type *[N]* and color *[C]*. Other distractor objects that are not equal to *[N]* are placed in the environment randomly.
- **Agent Objective:** Given the current grid state and the information that the next action taken by the agent is up, the agent must predict the next grid state. At test time, the agent should identify that the novel color-object combination *RED BALL* can also be pushed.

Experiment 4:

- **Challenge 4:** $h_{n1} = BALL$, $h_{n2} = DOOR$
- **Rules:** *BABA IS YOU,* $[N_1]$ *IS* $[N_2]$
- **Training and Testing Environments:** The agent, N_1, and any distractor objects are placed randomly throughout the environment. There is only one *BABA* and one object of type N_1.
- **Agent Objective:** Given the current grid state and the information that the agent takes a step (location does not matter), the model is asked to predict the next grid state. The idea behind this task is that the model receives an encoding

of the grid world that contains object N_1. The model's outputted encoding of the world should replace these N_1 representations with N_2 representations. More concretely, the model receives an encoding of the grid world that contains a ball object representation in one of the grid spaces. The model's output should replace this ball object representation with a door object representation in that grid space.

Experiment 5:

- **Challenge 5:** $h_{n1} = BALL$
- **Rules:** *BABA IS YOU,* $[N_1]$ *IS* $[N_2]$
- **Training and Testing Environments:** The agent, N_1, and any distractor objects are placed randomly throughout the environment. There is only one *BABA* and one object of type N_1.
- **Agent Objective:** The agent objective and motivation behind this objective is the same as Experiment 4. At test time, the agent should be able to generalize to the scenario in which h_{n1} transmutes into another object.

Experiment 6:

- **Challenge 6:** $h_{n2} = BALL$
- **Rules:** *BABA IS YOU,* $[N_1]$ *IS* $[N_2]$
- **Training and Testing Environments:** The agent, N_1, and any distractor objects are placed randomly throughout the environment. There is only one *BABA* and one object of type N_1.
- **Agent Objective:** The agent objective and motivation behind this objective is the same as Experiment 4. At test time, the agent should be able to generalize to the scenario in which any object transmutes into h_{n2}.

Experiment 7:

- **Challenge 7:** $n = BALL$
- **Rules:** *[N] IS YOU*
- **Training and Testing Environments:** A singular object *[N]* is randomly placed in the environment as well as any distractor objects for the training environments. The only difference between training and testing environments is that testing environments must have at least two objects of type *[N]*. These objects are also placed randomly.
- **Agent Objective:** The model is given an encoding of the current state of the environment and is told the agent's next action. Using this information, the model must predict the next state of the environment by outputting an encoding of the grid. The goal is for the model to recognize which objects are controlled by the agent. This notion can be reflect in the model's prediction. If the model correctly generalizes, then at test time the agent will output a predicted environment state that has moved all of the balls in the environment one space in the direction of the inputted action.

Experiment 8:

- **Challenge 8:** $h_n = BALL$
- **Rules:** *[N] IS YOU*
- **Training and Testing Environments:** A singular object *[N]* is randomly placed in both the training and testing environments. Distractor objects may also be present and randomly placed.
- **Agent Objective:** The objective and motivation behind this experiment are similar to Experiment 7. The model is tasked with predicting the next state of the environment. For instance, at test time, when the model is given an embedding of the environment and is told that the agent will move left, then

the outputted prediction should show that the ball has moved one space to the left.

3.3.2 Reinforcement Learning Experiments

It is important to note that given the time constraints, we do not run the RL models on each of the eight challenges. However, we provide a simple example with Challenge 1 to demonstrate how these experiments can be designed. We will leave the remaining seven challenges for future work.

Experiment 1:

- **Challenge 1:** $h_c = RED$, $h_n = BALL$
- **Rules:** *BABA IS YOU*, *[C] [N] IS WIN*
- **Training and Testing Environments:** In this scenario, the agent is located in between two objects: *[N]* and a distractor object. These objects will never be of the same color and type. Similar to the SL version of this experiment, red balls can be present in the training environment, but they are never goal objects.
- **Agent Objective:** The agent's objective is significantly more straightforward in the RL version of these experiments. The objective is to navigate to the goal object, which is the red ball at test time.

3.4 Experimental Procedure

3.4.1 Supervised Learning

To obtain our preliminary results, we train a new RefEx transformer model on each of the experiments, and we run each of the eight experiments five times. For each run, we use 10e5 training samples and 5*10e3 testing samples. The locations of the objects are randomized (within the limits of a given experiment), and distractor object

number, type, color, and position are also randomized to generate these examples. We set the batch size to 100 samples and the number of epochs to 20,000.

The loss function, accuracy calculation, and stopping criterion slightly differ based on the agent's objective. In the case where the agent's objective is to locate the goal object, the loss is calculated using cross-entropy. The accuracy is a binary computation where the model earns an accuracy score of one if the outputted goal location is correct and zero otherwise. We will take the mean over all testing example accuracies to determine the overall accuracy for that particular experiment and run. The stopping criterion for this task is determined by whatever comes first: either the test accuracy reaches 100% or the maximum number of iterations has been reached. In the case where the agent's objective is to predict the next state of the environment, the loss is calculated using mean squared error. The accuracy is also a binary computation where the model earns an accuracy score of one if the model's prediction is correct and zero otherwise. By correct, we mean that the one-hot encoding of each cell represents the correct number, type, and color of objects and text blocks. We will take the average over all testing example accuracies to determine the overall accuracy for that particular experiment and run. The stopping criterion is set by a fixed number of iterations.

3.4.2 Reinforcement Learning

For the RL experiment, we train one model once. It is important to note that this RL experiment is a small example, so we recognize that in order to obtain strong results we would need to repeat this process many times. We evaluate the agent on one hundred episodes. The stopping criterion is when the reward reaches the best possible reward. In the case of this experiment, the best reward can be achieved when the agent reaches the goal object in one move. We calculate the accuracy as the test reward divided by the train reward.

Table 3.1: SL Experiments: Average Accuracy per Task

SL Experiment	**Average Accuracy (%)**
Ex1:	100
Ex2:	57.8
Ex3:	85.4
Ex4:	17.4
Ex5:	0
Ex6:	0
Ex7:	90.8
Ex8:	6.8

3.5 Results

We obtain preliminary results that demonstrate how the Baba benchmark is an effective tool to explore compositional generalization. In this section, we will discuss our findings.

3.5.1 Supervised Learning

The average accuracy across all five runs for each experiment can be found in Table 3.1. Appendix A has the tables detailing the model accuracies for each run. Overall, our results reveal three key findings. First, they confirm the credibility of the Baba environment as an effective tool to measure compositional generalization. Second, they demonstrate the difficulty of the benchmark. Finally, these results highlight several key areas for future work.

Experiment 1: As discussed in Chapter 3, this experiment is identical to the *Novel Composition of Object Properties* task in the gSCAN paper. When Sikarwar reproduced this task in his environment and ran his model RefEx on this this task, the model achieved 100% accuracy [44]. Therefore, our result of 100% accuracy in this experiment is in line with Sikarwar's findings, which demonstrates our ability to reproduce results in the Baba environment. This result has three important implications. First, it bridges the gap between the gSCAN and Baba environments because

it highlights how we can first explore gSCAN tasks in our environment and then make these tasks more challenging. Second, this result confirms the validity of our Baba environment as a feasible tool to explore rule-based generalization. Achieving a lower accuracy in this task would indicate that there are critical issues with the Baba environment implementation. Finally, it displays an agent's ability to generalize to a novel color-object pair in a static rule environment.

Experiment 2: The model achieves a wide range of accuracy scores for this task that range from 44% to 94% (see Appendix A). While transformers are known to have larger variances than other types of models, the clear next step for this experiment is to compute the number of runs required to ensure a 95% confidence interval. Overall, this experiment is harder than the previous one because the agent must predict the next state of the entire grid world rather than the position of a particular object. Once we perform enough runs to satisfy the 95% confidence interval constraint, we can then make broader claims about an agent's ability to generalize to scenarios in which it must *PUSH* a novel object in a static rule environment.

Experiment 3: While this experiment tests an agent's ability to generalize to novel color-object pairs, it is more challenging than Experiment 1 as the agent must predict the next state of the grid environment whereas the agent in Experiment 1 only had to predict the location of the goal object. Similar to Experiment 2, the per run model accuracy widely varies from 36% to 100% (see Appendix A), so it is also necessary to calculate the number of runs required to achieve a 95% confidence interval with our results. After we perform those runs, we can analyze an agent's ability to generalize to novel color-object combinations in the context of pushing that particular object in a static rule environment.

Experiment 4: Similar to Experiments 2 and 3, more runs need to be performed in order to collect results within a 95% confidence interval. However, from an early look at the results, the model on average performs significantly worse compared to the first three experiments. This preliminary result is exciting because this experiment

tests an agent's ability to generalize to novel transmutations, which is unique to the Baba environment. This notion must be confirmed with more model runs. If it remains the case that agents perform poorly in this task, then this experiment will demonstrate the benchmark's importance and usefulness as a tool to evaluate new areas of compositional generalization with which current models struggle.

Experiments 5 and 6: These experiments are symmetric, so it makes sense that the model averages a similar accuracy for both of these experiments. The average accuracy of 0% further supports the claim made in our analysis of Experiment 4: the RefEx transformer is not able to generalize to novel transmutations in a static rule environment. There are many approaches that can be taken to improve a model's ability to generalize in this case, and we will discuss these strategies in Section 3.6.

Experiment 7: The model achieves an average accuracy of 90.8% with significantly less variance than Experiments 2 - 4. This result indicates that the model has some capacity to generalize to controlling multiple objects of the same type in an environment with a static set of rules. Given this strong performance, we can make the environment more challenging in a variety of ways. For example, we could create this scenario in an environment with a dynamic set of rules. Another possibility would be to test a model's ability to generalize to conditions in which it controls a variety of object types and colors.

Experiment 8: The model achieves an average accuracy of 6.8% with some variance as these values range from 0% to 16%. While more runs should be performed for this experiment, we can start to make early conclusions about the model's performance. In this case, the agent is tasked with outputting the next state of the grid environment, which has one singular object. Therefore, if the model were to randomly put that object anywhere in the 6x6 grid, then it would achieve an average accuracy of about 3%. The model, therefore, is performing slightly better than if it were to have randomly guessed for every example, but it is still significantly struggling in this task. This result makes sense because the model never controls the held out ball

object during training, and it never interacts with the held out ball during training, so it never has the opportunity to ground to the meaning of ball. Thus, at test time, the model cannot generalize to controlling this novel object. In Section 3.6, we will discuss strategies to help agents better generalize in this scenario.

3.5.2 Reinforcement Learning

Experiment 1: PPO achieves a 92.4% accuracy for the one run that we perform for this experiment. While we will need to perform additional runs in order to ensure that this result is consistent across runs, this experiment serves as an important proof of concept. First, it demonstrates that RL models can run on the Baba environments. Second, this experiment highlights how generalization tasks can formulated into both SL and RL experiments, and the Baba environment particularly lends itself to these RL experiments. Finally, it allows us to start drawing conclusions about PPO's ability to generalize to novel color-object combinations. This claim must be supported by additional runs of this experiment with strong results.

3.6 Future Work

One critical goal of the experiments is to highlight important areas for future work. Based on the results, we will discuss three key categories of future work: building out a more robust set of baselines, implementing modeling augmentations to improve a model's capacity to generalize, and developing Baba environment enhancements to create additional challenges.

Before we delve into modeling or environment improvements, we first need to develop a more robust set of baselines to be able to accurately pinpoint when models struggle to generalize in the Baba environment. Before we discuss improvements to our set of baselines, it is worth noting that SL models have significant limitations in this Baba environment context because they cannot actually navigate the environment to solve the level like RL models can. However, many benchmarks such as gSCAN use SL models as their baseline. Therefore view SL models as a preliminary

step in our set of baselines.

In order to fully build out the set of SL baselines, we must first finalize the set of transformer baselines. Given the large variances seen in our preliminary results, we must first perform enough runs of the eight experiments to achieve a 95% confidence interval. Then, we will modify each experiment such that the agent now must change the set of rules in order to achieve the generalization task. This dynamic version of the experiments will allow us to directly measure how changing rule sets affects an agent's ability to perform its generalization task. Once the transformer has been run on both the static and dynamic version of the eight experiments a sufficient number of times, we will repeat the process with a CNN. It is important to have two SL models in order to compare experimental results, since SL and RL outputs cannot be compared.

Next, we will build upon the existing PPO experiment. For each of the selected tasks in the subset outlined in Section 3.1.1, we will develop a set of eight experiments similar to Experiment 1 in 3.3.2. Unlike the SL experiments, the agent's objective in each of these tasks will simply be to navigate to the goal object. Similar to the SL baselines, two types of environments should be developed: an environment with static rules and an environment with dynamic rules. In the dynamic rule environment, the agent needs to modify the rules in order to reach the goal agent. This static and dynamic version of the experiments will allow us to measure the effect that a dynamic rule set has on the agent's capacity to complete the given generalization task. Once we have the PPO experiments up and running, we will then run a DQN on the same set of experiments, so we can compare the results of the two models.

Based on our preliminary results, agents are having difficulty performing some of the simplest generalization tasks in the Baba environment. Tasks that involve generalizing to novel transmutations or controlling novel objects seem to be particularly challenging for agents. Building out the SL and RL baselines will provide more clarity on when these models fail to generalize.

The Baba environment can be used to explore how model augmentations can improve an agent's performance in these generalization tasks. One key reason why RL

models may fail to generalize in these experiments is the exploration versus exploitation challenge. The Baba environment allows us to explore this dilemma in the context of changing rules. One possible augmentation that could address this issue is that an agent could be created by combing both PPO and the transformer model. PPO could leverage the knowledge of the transformer model (specifically its predictions), to navigate the environment more efficiently. Another augmentation that could be tried to help agents generalize involves modifying the environment embeddings. Instead of representing the environment as a one-hot encoding, the environment embeddings could be developed using the CLIP model. The idea is that the one-hot encoding, in reality, does not capture what it means to be a ball versus a key versus a text block. CLIP, however, is a multi-modal vision and language model that could be used to create a richer embedding of the environment in which object and text block representations will have semantic meaning [39].

We can also make improvements to the Baba environment to better explore areas of compositional generalization. These enhancements should be guided by the results of the baseline experiments as well as the modeling augmentations. The game Baba is You has ten times as many text blocks as we have in our environment, so there are a number of text blocks that we could add to create interesting rules and interactions.

One key text block that we are particularly interested in including in our experiments is the action verb *MOVE*. When a noun is composed with *MOVE* to form the rule *[N] IS MOVE*, then objects represented by noun N will move one space each time step in the direction of their orientation unless they are blocked by the edge of the grid or a *STOP* object. This text block creates new challenges within the environment as other objects not controlled by the agent and not being actively pushed by an agent can move in the environment. *MOVE* rules also lead to conflicts when multiple objects try to push an object at once. Figure 2-5 illustrates these conflicts and our approach to resolving them.

The current Baba environment also has strict grammar limitations. Unlike the game *Baba is You*, which supports a wide variety of grammar structures, our version of the environment only supports two structures, which are described in Figure

2-1. Therefore, the environment does not support rules such as *PUSH IS STOP* or *BALL IS PUSH IS OPEN*, which could also technically be learned via compositionality. Therefore, another environment enhancement could be to expand the grammar structures allowed in the game.

Chapter 4

Conclusion

Compositional generalization is a core component of human cognition. Developing agents that can leverage systematic compositionality like humans to generalize to new conditions has been a long-standing problem in AI. Grounded benchmarks have been developed to test an agent's ability to generalize using this strategy. However, there are many issues with these benchmarks. Our work addresses two primary concerns with grounded benchmarks.

Grounded benchmarks lack structure. They propose sets of tasks without any formalism. Therefore, we propose a CFG that allows us to explore generalization mathematically as a function of the environment architecture. This formalism allows us to define a set of generalization tasks that are both exhaustive and scalable. As agents complete easier generalization tasks for a given benchmark, we can construct tasks that are mathematically proven to be more challenging than the previous ones using this structure. We show that this CFG has the flexibility to be applied to any grounded benchmark such as gSCAN. Therefore, it serves as an important tool that allows us to compare benchmarks. In general, it is difficult to make progress in this area of research without the necessary structure to evaluate the benchmarks themselves and to better identify when and why agents fail to generalize. This structure serves as a necessary step in creating a robust framework to explore compositional generalization.

The second critical problem that we address in this work is the rigidity of these

benchmarks. Specifically, the current benchmarks are defined by a static set of rules and a small set of objects whose state can be changed. We hypothesize that in order to fully understand a rule, an agent needs to understand the implications of breaking it. Therefore, by strictly delineating these rules, we have overlooked a key rule-understanding and manipulation capability that agents will need in the real world. To tackle this problem, we use our proposed formalism to develop the Baba benchmark. In this benchmark, agents must learn how to change the rules of the environment in order to complete the generalization tasks. This set up allows us to directly measure the effect that rule changing has on an agent's ability to generalize to novel conditions. Our preliminary results have show that models generally do not perform well in our benchmark, which demonstrates that the Baba benchmark provides a meaningful challenge for current models. Overall, this benchmark serves as a platform to examine an agent's ability to leverage compositional generalization to navigate environments that more closely resemble the real world.

Appendix A

Tables

A.1 SL Experiment Runs

Table A.1: SL Experiments: Run 1

SL Experiment	**Accuracy (%)**
Ex1:	100
Ex2:	94
Ex3:	100
Ex4:	0
Ex5:	0
Ex6:	0
Ex7:	90
Ex8:	5

Table A.2: SL Experiments: Run 2

SL Experiment	**Accuracy (%)**
Ex1:	100
Ex2:	44
Ex3:	36
Ex4:	87
Ex5:	0
Ex6:	0
Ex7:	88
Ex8:	0

Table A.3: SL Experiments: Run 3

SL Experiment	**Accuracy (%)**
Ex1:	100
Ex2:	47
Ex3:	97
Ex4:	0
Ex5:	0
Ex6:	0
Ex7:	93
Ex8:	1

Table A.4: SL Experiments: Run 4

SL Experiment	**Accuracy (%)**
Ex1:	100
Ex2:	52
Ex3:	95
Ex4:	0
Ex5:	0
Ex6:	0
Ex7:	92
Ex8:	12

Table A.5: SL Experiments: Run 5

SL Experiment	**Accuracy (%)**
Ex1:	100
Ex2:	52
Ex3:	99
Ex4:	0
Ex5:	0
Ex6:	0
Ex7:	91
Ex8:	16

Bibliography

[1] Raviteja Anantha, Stephen Pulman, and Srinivas Chappidi. Generalized reinforcement meta learning for few-shot optimization, 2020.

[2] Jacob Andreas. Good-enough compositional data augmentation. *CoRR*, abs/1904.09545, 2019.

[3] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. *ICML*, abs/1611.01796, 2016.

[4] Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron Courville. Systematic generalization: What is required and can it be learned?, 2019.

[5] Susan S. Jones Barbara Landau, Linda B. Smith. The importance of shape in early lexical learning. *Cognitive Development*, 3(3):299–321, 1988.

[6] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

[7] Joshua B. Tenenbaum Brenden M. Lake, Ruslan Salakhutdinov. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[8] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: First steps towards grounded language learning with a human in the loop. *CoRR*, abs/1810.08272, 2018.

[9] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for gymnasium, 2018.

[10] Noam Chomsky. *Syntactic Structures*. De Gruyter Mouton, Berlin, Boston, 1957.

[11] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning, 2019.

[12] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning, 2018.

[13] Misha Denil, Sergio Gomez Colmenarejo, Serkan Cabi, David Saxton, and Nando de Freitas. Programmable agents. *CoRR*, abs/1706.06383, 2017.

[14] Coline Devin, Daniel Geng, Pieter Abbeel, Trevor Darrell, and Sergey Levine. Plan arithmetic: Compositional plan vectors for multi-task control, 2020.

[15] Wenhao Ding, Haohong Lin, Bo Li, and Ding Zhao. Generalizing goal-conditioned reinforcement learning with variational causal reasoning, 2022.

[16] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents, 2017.

[17] Jonathan Gordon, David Lopez-Paz, Marco Baroni, and Diane Bouchacourt. Permutation equivariant models for compositional generalization in language. In *International Conference on Learning Representations*, 2020.

[18] Alan Jern and Charles Kemp. A probabilistic account of exemplar and category generation. *Cognitive Psychology*, 66(1):85–125, 2013.

[19] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning, 2016.

[20] Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, Dmitry Tsarkov, Xiao Wang, Marc van Zee, and Olivier Bousquet. Measuring compositional generalization: A comprehensive method on realistic data, 2020.

[21] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning, 2021.

[22] Brenden M. Lake. Compositional generalization through meta sequence-to-sequence learning. *CoRR*, abs/1906.05381, 2019.

[23] Brenden M. Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. 2017.

[24] Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *CoRR*, abs/1604.00289, 2016.

[25] Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning, 2019.

[26] Kimin Lee, Younggyo Seo, Seunghyun Lee, Honglak Lee, and Jinwoo Shin. Context-aware dynamics model for generalization in model-based reinforcement learning, 2020.

[27] Yunfei Li, Yilin Wu, Huazhe Xu, Xiaolong Wang, and Yi Wu. Solving compositional reinforcement learning problems via task reduction. In *International Conference on Learning Representations*, 2021.

[28] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[29] Shengyao Lu, Bang Liu, Keith G. Mills, Shangling Jui, and Di Niu. R5: Rule discovery with reinforced and recurrent relational reasoning, 2022.

[30] Jorge A. Mendez, Harm van Seijen, and Eric Eaton. Modular lifelong reinforcement learning via neural composition, 2022.

[31] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[32] Sarthak Mittal, Sharath Chandra Raparthy, Irina Rish, Yoshua Bengio, and Guillaume Lajoie. Compositional attention: Disentangling search and retrieval, 2021.

[33] Richard Montague. Universal grammar. *Theoria*, 36(3):373–398, 1970.

[34] Sanmit Narvekar, Jivko Sinapov, and Peter Stone. Autonomous task sequencing for customized curriculum design in reinforcement learning. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 2536–2542, 2017.

[35] Maxwell I. Nye, Armando Solar-Lezama, Joshua B. Tenenbaum, and Brenden M. Lake. Learning compositional rules via neural program synthesis. *CoRR*, abs/2003.05562, 2020.

[36] Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. *CoRR*, abs/1706.05064, 2017.

[37] Barbara Partee. Compositionality. *Varieties of Formal Semantics*, 3:281–311, 1984.

[38] Wenjie Qiu and He Zhu. Programmatic reinforcement learning without oracles. In *International Conference on Learning Representations*, 2022.

[39] 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. Learning transferable visual models from natural language supervision, 2021.

[40] Laura Ruis, Jacob Andreas, Marco Baroni, Diane Bouchacourt, and Brenden M. Lake. A benchmark for systematic generalization in grounded language understanding, 2020.

[41] Jake Russin, Jason Jo, Randall C. O'Reilly, and Yoshua Bengio. Compositional generalization in a deep seq2seq model by separating syntax and semantics. *CoRR*, abs/1904.09708, 2019.

[42] Raeid Saqur and Karthik Narasimhan. Multimodal graph networks for compositional generalization in visual question answering. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 3070–3081. Curran Associates, Inc., 2020.

[43] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[44] Ankur Sikarwar, Arkil Patel, and Navin Goyal. When can transformers ground and compose: Insights from compositional generalization benchmarks, 2022.

[45] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, L. Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 2017.

[46] Sungryull Sohn, Junhyuk Oh, and Honglak Lee. Hierarchical reinforcement learning for zero-shot generalization with subtask dependencies, 2019.

[47] Sungryull Sohn, Hyunjae Woo, Jongwook Choi, and Honglak Lee. Meta reinforcement learning with autonomous inference of subtask dependencies, 2020.

[48] Geraud Nangue Tasse, Steven James, and Benjamin Rosman. A boolean task algebra for reinforcement learning, 2020.

[49] Geraud Nangue Tasse, Steven James, and Benjamin Rosman. Generalisation in lifelong reinforcement learning through logical composition. In *International Conference on Learning Representations*, 2022.

[50] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[51] Huan Wang, Stephan Zheng, Caiming Xiong, and Richard Socher. On the generalization gap in reparameterizable reinforcement learning. 2019.

[52] F. Xu and J. B. Tenenbau. Word learning as bayesian inference. *Psychological Review*, 114(2):245–272, 2007.

[53] Haonan Yu, Haichao Zhang, and Wei Xu. A deep compositional framework for human-like language acquisition in virtual environment, 2017.