Guided Feature Transformation (GFT): A Neural Language Grounding Module for Embodied Agents

Haonan Yu[†], **Xiaochen Lian**[†], **Haichao Zhang**[†], **and Wei Xu**[‡][*]
[†]Baidu Research, Sunnyvale CA USA
[‡]Horizon Robotics, Cupertino CA USA
{haonanyu,lianxiaochen,zhanghaichao}@baidu.com,wei.xu@horizon.ai

Abstract: Recently there has been a rising interest in training agents, embodied in virtual environments, to perform language-directed tasks by deep reinforcement learning. In this paper, we propose a simple but effective neural language grounding module for embodied agents that can be trained end to end from scratch taking raw pixels, unstructured linguistic commands, and sparse rewards as the inputs. We model the language grounding process as a language-guided transformation of visual features, where latent sentence embeddings are used as the transformation matrices. In several language-directed navigation tasks that feature challenging partial observability and require simple reasoning, our module significantly outperforms the state of the art. We also release XWORLD3D, an easy-to-customize 3D environment that can be modified to evaluate a variety of embodied agents.

Keywords: Language Grounding, Virtual Navigation, Embodied Agents

1 Introduction

This paper examines the idea of building embodied [1, 2] agents that learn control from linguistic commands and visual inputs. One recent line of work [3, 4, 5, 6, 7] trains such agents situated in simulated environments in an end-to-end fashion, receiving unstructured linguistic commands and raw image pixels as the inputs, and producing navigation actions as the outputs. For successful navigation control, it is crucial for an agent to learn to associate linguistic concepts with visual features, a process known as language *grounding* [8, 9]. To avoid a tremendous amount of labeled data, this line of work trains language grounding oriented by navigation goals via reinforcement learning (RL). Through trials and errors, an agent learns not only to navigate but also to reinforce (or weaken) the connection between visual features and their matched (or unmatched) language tokens.

The problem of learning to control alone is quite challenging, especially in an environment with long time horizons and sparse rewards [10]. However, in this paper we only concentrate on language grounding as the core of the perception system of an agent, while using an existing RL system design, synchronous advantage actor-critic (ParallelA2C, [11]). Because our model exploits no structural biases of specific tasks, it is possible to plug our language grounding module in many other neural architectures for tasks that combine language and vision.

We propose a simple but effective neural language grounding module that models a rich set of language-vision interactions. The module performs a **g**uided **f**eature **t**ransformation (GFT), where latent sentence embeddings computed from the language input are treated as the transformation matrices of visual features. This guided transformation is much more expressive than the existing three categories of language grounding methods for embodied agents: vector concatenation [4, 6, 12], gated networks [5, 13], and convolutional interaction [3, 7]. In fact, it can be treated as a generalization of the last two categories.

GFT is fully differentiable and is embedded in the perception system of our navigation agent that is trained end to end from scratch by RL. In an apples-to-apples comparison, our model[2] significantly

[*]Work done at Baidu Research
[2]A PyTorch reimplementation is available at https://github.com/idlrl/flare/blob/master/tutorial/examples/xworld3d_navigation.py

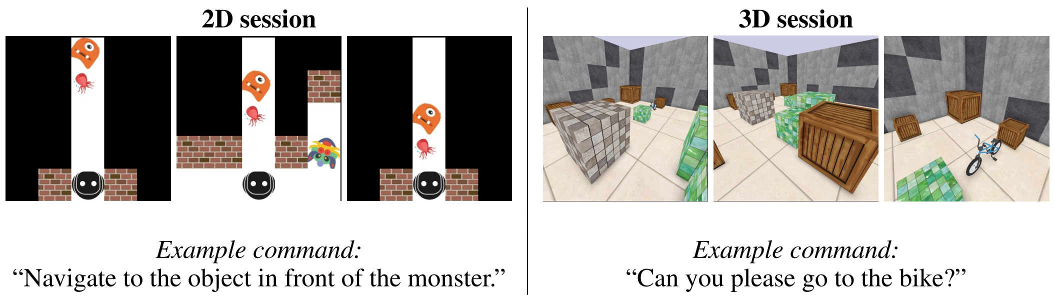

2D session
3D session
Example command:
"Navigate to the object in front of the monster."
Example command:
"Can you please go to the bike?"

Figure 1: Illustration of the two environments for evaluating our agent. The scene is randomly initialized, and only several key frames are shown in either case. The agent perceives images in the first-person view. The commands require language understanding skills such as object recognition, spatial reasoning, and semantic opposition. In the 2D case, the agent has a limited visible field regardless of the map size. The black area behind wall blocks indicates invisible regions. The 3D scenario with perspective distortion is closer to human experience.

outperforms the existing state of the art, over a rich set of navigation tasks that feature challenging partial observability and cluttered background, and require simple reasoning (Figure 1). Our GFT-powered agent is able to handle both the 2D and 3D environments without any architecture or hyperparameter change between the two scenarios[3]. This demonstrates the generality and efficacy of GFT as a language grounding module that can potentially benefit a variety of embodied agents for other language-vision tasks. Finally, we will release the XWORLD3D[4] environment used in the experiments. XWORLD3D highlights a teacher infrastructure that enables flexible customization of linguistic commands, environment maps, and training curricula.

2 Guided Feature Transformation for Language Grounding

The major contribution of this paper is a simple yet effective language grounding module called GFT. We will first describe GFT and then discuss our motivations and its advantages. In a general scenario, given a pair of an image $\mathbf{o}$ and a sentence $\mathbf{l}$, a language grounding module fuses the two modalities for downstream processings. This is a common process seen in visual question answering (VQA) [14], and it also lies at the heart of our agent's perception system.

2.1 Method

We use a convolutional neural network (CNN) to convert $\mathbf{o}$ into a feature cube $\mathbf{C} \in \mathbb{R}^{D\times N}$, where D is the number of channels and N is the number of locations in the image spatial domain (collapsed from 2D to 1D for notational simplicity). Suppose that we have an embedding function that converts $\mathbf{l}$ to a series of J embedding vectors $\mathbf{t}_1, \ldots, \mathbf{t}_j, \ldots, \mathbf{t}_J$, where each $\mathbf{t}_j \in \mathbb{R}^{D(D+1)}$ can be reshaped and treated as a matrix $\mathbf{T}_j \in \mathbb{R}^{D\times(D+1)}$. Then we compute a series of J transformations, one followed by another, activated by a nonlinear function g:

$$\mathbf{C}^{[j]} = g(\mathbf{T}_j \begin{bmatrix} \mathbf{C}^{[j-1]} \\ \mathbf{1}^\intercal \end{bmatrix}), \quad 1 \le j \le J, \tag{1}$$

where $\mathbf{1} \in \mathbb{R}^N$ is an all-one vector and $\mathbf{C}^{[0]} = \mathbf{C}$. This guided transformation yields a feature cube $\mathbf{C}^* = \mathbf{C}^{[J]} \in \mathbb{R}^{D\times N}$ which is the final grounding result for downstream processings. Overall, it would be expected that the transformation matrices $\mathbf{T}_j$ correctly capture the critical aspects of command semantics, in order for the agent to perform tasks. (Examples of the trained $\mathbf{T}_j$ in a later experiment are visualized in Appendix G.) Despite its simple form, GFT is able to model a rich set of interactions between language and vision, resulting in strong representational power. Indeed, it can be seen as a unifying formulation of two existing language grounding modules, namely, gated networks [15, 5] and convolutional interaction [3, 7].

[3]Video demo at https://www.youtube.com/watch?v=bOBb1uhuJxg

[4]https://github.com/PaddlePaddle/XWorld/tree/master/games/xworld3d

2.2 Why GFT?

GFT is a stack of *generalized* gated networks. In the following we will ignore the subscript j of $\mathbf{T}_j$ for notational simplicity. Let us first write:

$$\mathbf{T} = [\mathbf{T}' \ \ \mathbf{b}],$$

where $\mathbf{T}' \in \mathbb{R}^{D\times D}$, and $\mathbf{b} \in \mathbb{R}^D$ is a bias vector for the transformation in Eq. 1. We would like to investigate what $\mathbf{T}'\mathbf{C}$ essentially does. Toward this end, we perform a singular value decomposition (SVD) of $\mathbf{T}'$:

$$\mathbf{T}' = \mathbf{U}\boldsymbol{\Lambda}\mathbf{V}^\intercal,$$

where $\mathbf{U} \in \mathbb{R}^{D\times D}$ and $\mathbf{V} \in \mathbb{R}^{D\times D}$ are both orthogonal matrices. $\boldsymbol{\Lambda} \in \mathbb{R}^{D\times D}$ is a diagonal matrix that contains D values $\lambda_1, \ldots, \lambda_D$ (not necessarily non-negative) on the diagonal.

In an extreme case where both $\mathbf{U}$ and $\mathbf{V}$ are identity matrices, each transformation step of GFT degrades to a gated network. Specifically, let $\mathbf{c}_d$ be the d-th feature map, we have

$$\mathbf{c}_d^{[j]} = g(\lambda_d \mathbf{c}_d^{[j-1]} + b_d) \tag{2}$$

This is exactly the same modulation provided by FiLM [15]. Intuitively, Eq. 2 performs scaling, thresholding, or negating of the features on the dth map of $\mathbf{C}^{[j-1]}$, according to the semantics of the input command $\mathbf{l}$. A further specialized version was proposed by Chaplot et al. [5] in which they remove g and b_d, while activating λ_d by sigmoid. Thus when $\mathbf{U}$ and $\mathbf{V}$ are identities, GFT is a stack of FiLMs.

In a general case, $\mathbf{U}$ and $\mathbf{V}$ are dense and represent general rotations in $\mathbb{R}^D$. Because $\mathbf{U}$ and $\mathbf{V}$ are computed from the command $\mathbf{l}$, they are *language-guided*. This is a major difference between a transformation step of GFT and that of a gated network. The latter always modulates features in the same original feature space, regardless of the command $\mathbf{l}$. As a result, a gated network such as FiLM places more pressure on learning $\mathbf{C}$ because a single feature space has to reconcile with a huge number of commands. When the vision is challenging or the language space is huge, modulating only in the original feature space might become a performance bottleneck of the overall agent model. In contrast, a transformation step of GFT can choose to rotate the axes of the feature space ($\mathbf{V}^\intercal$), scale the features in that rotated space ($\boldsymbol{\Lambda}\mathbf{V}^\intercal$), and rotate the scaled feature again ($\mathbf{U}\boldsymbol{\Lambda}\mathbf{V}^\intercal$). These choices are determined by the current command. On top of it, GFT performs this "rotate-scale-rotate" operation multiple times in a sequence, when combined together, resulting in high-order nonlinear feature modulation.

GFT performs concept detection over *multiple* convolutional steps. An alternative interpretation is possible if we treat Eq. 1 as a 1×1 convolution with D filters and a stride of one, sharing similar motivations in Oh et al. [3] and Yu et al. [7]. In such an interpretation, each row of $\mathbf{T}$ contains a 1×1 convolutional filter of length D and a scalar bias. The D filters (rows) represent at most D different or complementary aspects of the semantics of the command $\mathbf{l}$, and a step of Eq. 1 essentially performs concept detection. For example, a 3D asymmetric object such as a bike has different appearances from different viewing angles. Thus having multiple filters of the sentence "go to bike" improves the representational power and results in a higher chance of finding the corresponding concept when the agent moves around. Overall, GFT performs multi-step concept detection with language-dependent filters at each step.

In summary, the simple but general formulation of GFT unifies several existing ideas for grounding language in vision. In the remainder of this paper, we will evaluate it in a challenging language-directed navigation problem.

3 The Navigation Problem

We formally introduce our problem as a partially observable Markov decision process (POMDP) [16] as follows. The problem is divided into many navigation sessions. At the beginning of each session, a navigation task is sampled by a pre-programmed teacher as $k \sim P(k)$. Given the task, an initial environment state $\mathbf{s}^{[1]}$ is sampled by the simulator as $\mathbf{s}^{[1]} \sim P(\mathbf{s}^{[1]}|k)$, i.e., the simulator arranges the scene according to the sampled task. An environment state $\mathbf{s}$ contains both the map configuration and the agent's pose. We assume that the teacher has full access to the environment state, and samples a linguistic command $\mathbf{l} \sim P(\mathbf{l}|\mathbf{s}^{[1]}, k)$ which will be used throughout the session. At each time step

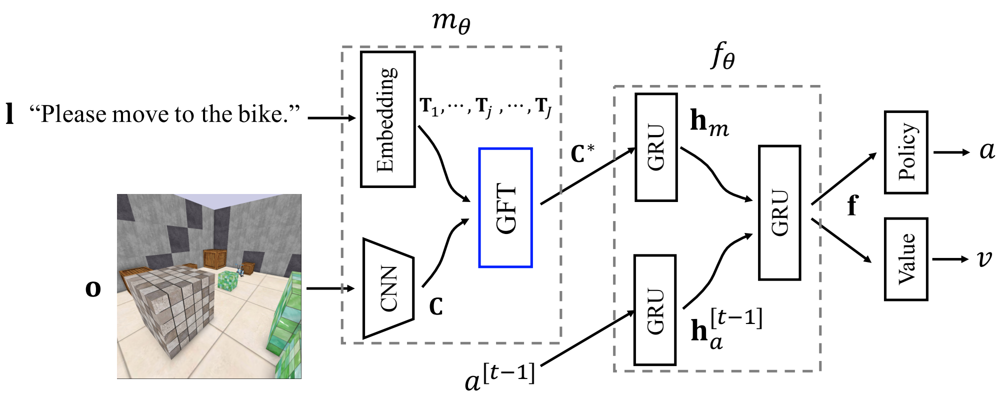


Figure 2: An overview of our agent architecture. In general, it is a two-layer GRU network with GFT embedded as the core for perception, and with value and policy networks for control. For notational simplicity, we only keep the superscript $t-1$ to indicate quantities from the previous time step, while removing the superscript t for quantities at the current time step. The two GRU states $\mathbf{h}_m^{[t-1]}$ and $\mathbf{h}_a^{[t-1]}$ (defined in Eq. 8 Appendix A) together form the agent history $\mathbf{h}^{[t-1]}$. From the figure we can see that GFT is a portable module that could potentially be plugged into other networks involving language-vision interactions.

t, the agent only has a partial observation of the environment, computed by a rendering function $\mathbf{o}^{[t]} = o(\mathbf{s}^{[t]})$. We assume that $P(k)$, $P(\mathbf{s}^{[1]}|k)$, $P(\mathbf{l}|\mathbf{s}^{[1]}, k)$, and $o(\cdot)$ are all unknown to the agent.

Below we use θ to denote the complete set of the model parameters, and let any function that uses a subset of parameters directly depend on θ for notational simplicity. The agent takes an action $a^{[t]}$ according to its policy π_θ, given the current observation $\mathbf{o}^{[t]}$, the history $\mathbf{h}^{[t-1]}$, and the command $\mathbf{l}$:

$$a^{[t]} \sim \pi_\theta(a^{[t]}|\mathbf{o}^{[t]}, \mathbf{h}^{[t-1]}, \mathbf{l}), \tag{3}$$

where a latent vector $\mathbf{h}^{[t-1]} = h_\theta(\mathbf{o}^{[1:t-1]}, a^{[1:t-1]}, \mathbf{l})$ summarizes all the previous history of the agent at time t in the current session. The teacher observes the action and gives the agent a scalar reward $r^{[t]} = r(a^{[t]}, \mathbf{s}^{[t]}, \mathbf{l})$. Note that the reward depends not only on the state and the action, but also on the command. The environment then transitions to a new state $\mathbf{s}^{[t+1]} \sim P(\mathbf{s}^{[t+1]}|\mathbf{s}^{[t]}, a^{[t]})$. This act-and-transition iteration goes on until a terminal state or the maximum time T is reached. Our problem is to maximize the expected reward:

$$\max_\theta \mathbb{E}_{\mathcal{S},\mathcal{A},\mathbf{l}} \left[\sum_t \gamma^{t-1} r^{[t]} \right], \tag{4}$$

where $\mathcal{S} = (\mathbf{s}^{[1]}, \ldots, \mathbf{s}^{[t]}, \ldots)$ and $\mathcal{A} = (a^{[1]}, \ldots, a^{[t]}, \ldots)$ denote the state and action sequences, respectively. $\gamma \in [0, 1]$ is a discount factor. Note that $r(\cdot)$ and $P(\mathbf{s}^{[t+1]}|\mathbf{s}^{[t]}, a^{[t]})$ are also unknown to the agent, namely, the RL agent is model-free.

The objective Eq. 4 fits in the standard RL framework and is readily solvable by the actor-critic (AC) algorithm [17]. Specifically, we compute the following policy gradient for any time step t:

$$-\mathbb{E}_{\mathbf{s}^{[t]}, a^{[t]}, \mathbf{l}} \left[\left(\nabla_\theta \log \pi_\theta(a^{[t]}|\mathbf{o}^{[t]}, \mathbf{h}^{[t-1]}, \mathbf{l}) + \eta \nabla_\theta v_\theta(\mathbf{o}^{[t]}, \mathbf{h}^{[t-1]}, \mathbf{l}) \right) A^{[t]} + \kappa \nabla_\theta \mathcal{E}(\pi_\theta) \right], \tag{5}$$

where v_θ is the estimated value function, $\mathcal{E}(\cdot)$ denotes entropy for encouraging exploration [18], and $\kappa > 0$ and $\eta > 0$ are constant weights. The advantage $A^{[t]}$ is computed as

$$A^{[t]} = r^{[t]} + \gamma v_\theta(\mathbf{o}^{[t+1]}, \mathbf{h}^{[t]}, \mathbf{l}) - v_\theta(\mathbf{o}^{[t]}, \mathbf{h}^{[t-1]}, \mathbf{l}). \tag{6}$$

Our implementation adopts the ParallelA2C design [11] to aggregate a minibatch of gradients (Eq. 5) over multiple identical agents running in parallel (each agent in a separate copy of the environment) over multiple time steps, with θ synchronized and shared among their models.

Note in Eq. 5 that the inputs of the policy network π_θ and the value network v_θ are identical. Thus we share a sub-network between π_θ and v_θ for parameter efficiency [18]. The sub-network outputs a latent state representation $\mathbf{f}^{[t]}$ and has two stages:

$$\mathbf{f}^{[t]} = f_\theta\left(m_\theta(\mathbf{o}^{[t]}, \mathbf{l}), \mathbf{h}^{[t-1]}\right). \tag{7}$$

The first stage m_θ is a multimodal function that grounds language in vision, and the second stage f_θ combines the grounding result with the agent history $\mathbf{h}^{[t-1]}$. We instantiate m_θ by using GFT for language grounding, and instantiate f_θ as a gated recurrent unit (GRU) [19]. An overview of the agent architecture is illustrated in Figure 2. We refer the reader to more details in Appendix A.

4 Related Work

Virtual navigation. Prior to this work, there have been several studies demonstrating virtual agents learning to navigate in virtual environments, based on reinforcement signals [20, 21]. Despite the impressive results achieved, these studies usually have fixed goals for agents. For example, an agent always learns to pick up apples or avoid enemies with specific appearances. In other words, the agent's goals are fixed and cannot be changed, unless the rewards are modified followed by retraining. There is no language understanding involved: the perceptual inputs are images only.

Multi-goal virtual navigation. A recent line of work augments the above virtual navigation with multiple non-linguistic goals. These goals are specified in different forms: target images [22, 23], one-hot or continuous embeddings [24, 25, 3, 26, 27], target poses [28], etc. In contrast, our focus is on understanding linguistic commands.

Language-directed virtual navigation. Another recent line of work [4, 3, 5, 6, 7, 29, 30] augments the virtual navigation problem with linguistic inputs, where an agent's goal always depends on an instructed command. Accordingly, it is crucial for these methods to ground language in vision. Our GFT generalizes and improves some existing language grounding modules (details in Section 2), while incurring negligible additional costs in implementation and training time.

Visual question answering. Unlike VQA [14, 31, 32, 15, 6, 30], our problem does not require the agent to answer questions. Instead, the agent takes movement actions to respond to the teacher. However, both problems require language grounding, the study of which might be transferred from one problem to another. Indeed, in Section 5, the **FiLM** comparison module is adapted from Perez et al. [15], the **CGated** and **Concat** modules were adopted in some early work on VQA [14], and the **Concept** comparison module resembles the stacked attention network (SAN) [32] when there is only one single attention layer. Although GFT is proposed for embodied agents, we hope that it will also benefit research on VQA.

Grounding language in vision and robotics. Our work is also related to language grounding in realistic images [33, 34, 35] and robotics navigation under language [36, 37, 38], where static labeled datasets are required. In addition, these methods for language understanding usually employ structural assumptions specifically for their problems. In contrast, our GFT module is general-purpose and could potentially be easily applied to a wide range of problems that require language grounding.

5 Experiments

We evaluate our agent in two challenging environments: XWORLD2D and XWORLD3D (Figure 1). Both environments host the same set of five types of language-directed navigation tasks described in Table 1. A common syntax is shared by the two environments for generating task commands. Except object words, the remaining lexicon including grammatical and spatial-relation words, is also shared. Thus the only differences between XWORLD2D and XWORLD3D in our experiments are graphics and objects. Both environments generate random navigation sessions following the problem definition in Section 3.

Despite the huge difference between the visual structures of the two worlds, we apply a structurally identical agent to both of them. This identity includes the same network architecture, the same set of actions (`move_forward`, `move_backward`, `move_left`, `move_right`, `turn_left`, and `turn_right`), and the same set of hyperparameter values (e.g., learning rate, batch size, momentum, layer sizes, etc). Only the model parameters are different and learned separately. Such an experiment setting tests the generalizability, efficacy, and portability of GFT as a language grounding module.

Type	Navigation target	Example command
`nav`	the specified object	"Please go to the chair."
`nav_nr`	an object near the specified one	"Move to the object near the chair."
`nav_bw`	the location between the two objects	"Go to the location between the chair and the table?"
`nav_avoid`	any object but the specified one	"Avoid the chair."
`nav_dir`	an object specified by a relative direction w.r.t. another object	"Navigate to the object left of the chair."

Table 1: The five types of navigation tasks in both environments.

5.1 Comparison Methods

We perform an apples-to-apples comparison with six state-of-the-art language grounding modules for embodied agents. To do so, we make minimal changes to our agent architecture when implementing the comparison methods: only the multimodal function m_θ is changed for each method, with the remaining components unchanged. Regardless of the choice, the output of m_θ is always flattened to a vector as an input to f_θ. We assume that the input command is always first encoded into a fixed-length embedding $\mathbf{l}_{\text{BoW}}$ by a bag-of-words (BoW) encoder for training efficiency. The six comparison methods are:

Concat [4, 6, 12, 30] directly concatenates a compact language embedding and a compact visual embedding. We project [5] both $\mathbf{l}_{\text{BoW}}$ and $\mathbf{C}$ to the same dimension before the concatenation.

Gated [5] weights the feature maps of $\mathbf{C} \in \mathbb{R}^{D \times N}$ by a gate vector $\mathbf{l}_{\text{gate}} \in [0,1]^D$. In our case, $\mathbf{l}_{\text{gate}}$ is generated by a two-layer multilayer perceptron (MLP) from $\mathbf{l}_{\text{BoW}}$.

CGated [13] is a variant of **Gated**. Instead of weighting feature maps of $\mathbf{C}$, they project $\mathbf{C}$ down to a visual embedding which is then weighted by $\mathbf{l}_{\text{gate}}$, a gate vector projected from $\mathbf{l}_{\text{BoW}}$ to the same dimension of the visual embedding.

FiLM [15] follows exactly Eq. 2 which can be seen as a special case of our method. All λ_d and b_d are generated by a two-layer MLP from $\mathbf{l}_{\text{BoW}}$.

Concept [7] directly treats $\mathbf{l}_{\text{BoW}}$ as a 1×1 filter. An attention map is obtained by convolving $\mathbf{C}$ with the filter. In addition, $\mathbf{C}$ is convolved with a 1×1 filter at a stride of one to produce an environment map. Finally, the attention map and the environment map are concatenated.

EncDec [29] makes several modifications to the original implementation to better suit our problem. First, we train the CNN from scratch. Second, we compute the instruction context directly as $\mathbf{l}_{\text{BoW}}$ without using word attention, since our teacher will not issue detailed, long-paragraph commands. Third, we add one additional layer of GRU after the concatenation of the decoder state and the instruction context, to model longer-range temporal dependency.

Two variants of our agent are reported: **GFT-1** ($J = 1$) and **GFT-2** ($J = 2$)[6]. We generate $\mathbf{T}_j$ by a two-layer MLP from $\mathbf{l}_{\text{BoW}}$. After training, each of the eight methods is evaluated for 10k test sessions, where the models saved for the final three passes (each pass contains 5k minibatches) of each method are used to obtain an average result. The comparison setting described in this section applies to both XWORLD2D and XWORLD3D. More details of the comparison methods and our method are described in Appendix B.

5.2 Optimization Details

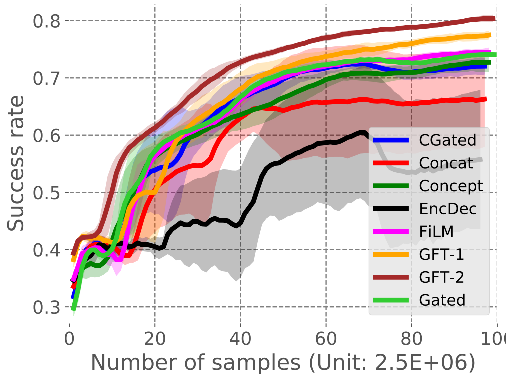
The optimization details described in this section apply to all the methods in both environments. We adopt RMSprop [39] with a learning rate of 10^{-5}, a damping factor of $\epsilon = 0.01$, and a gradient moving average decay of $\rho = 0.95$. The gradient has a momentum of 0.9. The batch size is set to 128. The total number of training batches is 2 million. The parameters of each method are initialized with four different random seeds, the results of which are averaged and reported.

5.3 Results for XWORLD2D

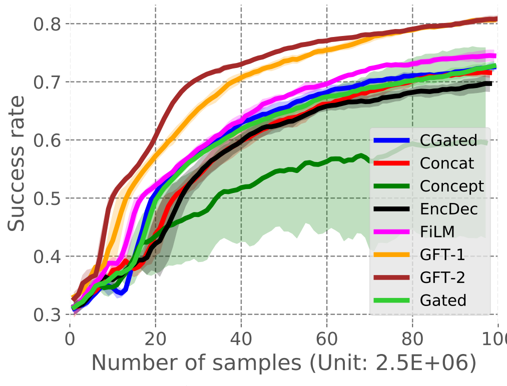
Environment and action. We modified the XWORLD2D environment [7] to host our navigation tasks. The original fully-observable setting now becomes a partially-observable egocentric setting

[5] In the remainder of this paper, a projection denotes a fully-connected (FC) layer followed by a nonlinear activation function.

[6] According to our observation, $J = 3$ appears to be a saturation point whose performance has almost no gain over $J = 2$.



(a) XWORLD2D



(b) XWORLD3D

Figure 3: Success rates (averaged over four random restarts) vs. number of training samples (time steps). The shaded area around each curve denotes the standard deviation. Although each method trains the same number of minibatches, for details explained in Appendix A, the total number of actions taken by the agent might vary slightly for different methods.

in 2D. The original action set is augmented with `turn_left` and `turn_right`, of which the yaw changes are both $90°$. To increase the visual variance, at each session we randomly rotate each object and scale it randomly within $[0.5, 1.0]$. Suppose each map is $X \times Y$, then a session will end after $3XY$ time steps if a success or failure is not achieved. In the experiment, we set $X = Y = 8$. Each 8×8 map contains 4 objects and 16 obstacles. We use Prim's algorithm [40] to randomly generate a minimal spanning tree for placing the obstacles so that the map is always a valid maze. The objects and the agent are then randomly initialized while complying with the sampled navigation task k. The agent can only see a 5×5 area in front of it, excluding any region occluded by obstacles. Generalization to larger maps will be investigated in Appendix E.

Rewards. A success (failure) according to the teacher's command gives the agent a $+1$ (-1) reward. A failure is triggered whenever the agent hits any object that is not required by the command. The time penalty of each step is -0.01. No other extrinsic or intrinsic rewards are used.

Objects and vocabulary. We use a collection of 345 object instances released by Yu et al. [7], constituting 115 object classes in total. The vocabulary contains 115 object words, 8 spatial-relation words, and 40 grammatical words, for a total size of 163. In total, there are 1,187,850 distinct sentences that can be generated by the teacher's predefined context-free grammar (CFG). The lengths of these sentences range from 1 to 15.

Results. The training curves of success rates are shown in Figure 3 (a). We observe that **GFT-1** has some marginal improvement on the success rates of the best-performing comparison methods such as **FiLM** and **Gated**. As we perform a second feature transformation, **GFT-2** produces a performance jump. Our explanation is that the visual recognition challenge, with random object yaws and scales in each session, requires an expressive language grounding function that can be better modeled by multiple steps of Eq. 1. Table 2 (2D) shows the test results split into five navigation types. **GFT-2** produces the best numbers in all the five columns. Unsurprisingly, `nav_avoid` has the highest rate because the agent only has to go to an arbitrary target which is not the specified one. `nav` has the lowest rate because the agent cannot exploit any object arrangement pattern like in `nav_bw`. See Figure 4 Appendix F for example navigation sessions.

5.4 Results for XWORLD3D

Environment and action. The environment layout and the agent's action are both discrete in XWORLD3D. An $X \times Y$ map consists of XY square grids, each grid as a unit containing an object, an obstacle, or nothing. An object always has a unit scale and a random yaw when initialized. An obstacle has one of four scales: 1.0, 0.7, 0.5, and 0.3, which is randomly sampled when the obstacle is initialized. The agent walks (`move_{forward,backward,left,right}`) roughly half of a grid per time step. The yaw change of the agent when it turns (`turn_{left,right}`) is $45°$ per time step. A session will end after $10XY$ time steps if a success or failure is not achieved. In the experiment,

	`nav`		`nav_avoid`		`nav_bw`		`nav_dir`		`nav_near`	
	2D	3D	2D	3D	2D	3D	2D	3D	2D	3D
Concat	54.1±3.7	73.4±3.4	90.7±2.8	94.0±2.2	77.8±5.4	78.6±3.2	53.1±30.1	47.6±3.9	58.5±11.8	62.1±2.9
Gated	58.1±3.8	74.2±3.8	90.6±2.3	94.2±2.8	84.0±3.9	80.4±5.2	74.5±4.1	48.1±4.9	63.7±3.7	61.5±5.0
CGated	58.7±3.9	74.0±2.9	91.2±1.9	95.1±1.9	84.6±3.1	80.6±3.8	69.3±4.2	47.7±4.6	63.4±3.2	64.8±4.0
FiLM	58.8±3.8	78.6±2.9	91.0±2.4	95.4±1.8	83.7±2.9	78.6±4.2	73.2±4.3	52.0±4.6	66.4±5.6	68.0±3.7
Concept	64.6±4.3	65.8±19.1	93.4±1.6	90.5±10.8	80.8±3.0	60.9±24.4	61.5±3.9	31.7±13.7	64.9±3.9	54.0±16.7
EncDec	51.5±7.4	67.1±5.1	84.4±11.0	92.7±1.9	73.6±12.1	77.7±3.9	20.5±27.4	48.9±3.8	50.9±12.2	59.1±4.1
GFT-1	65.0±4.2	**85.2**±3.4	93.8±1.9	**96.3**±1.3	84.7±3.5	**82.1**±5.3	76.4±3.1	61.2±3.9	70.4±4.2	78.3±4.5
GFT-2	**70.7**±3.5	84.6±3.0	**94.5**±1.8	**96.3**±1.9	**85.6**±2.3	80.5±2.7	**80.7**±2.6	**61.4**±4.9	**72.3**±3.7	**78.8**±2.6

Table 2: The evaluation results for 10k test sessions. The average navigation success rates are reported as percentages. Numbers in **bold** represent the best ones. The second number in each cell represents the standard deviation over twelve test runs, each run corresponding to one pass (out of three final passes) of one trained model (out of four random initializations).

we set $X = Y = 8$. Each 8×8 map contains 4 objects and 16 obstacles, and its initialization follows the same process in the 2D case. Generalization to larger maps will be investigated in Appendix E.

Rewards. The reward function is the same with the 2D case.

Objects and vocabulary. XWORLD3D contains 88 different objects downloaded from `http://www.sweethome3d.com/freeModels.jsp`. There are three types of obstacles: brick, crate, and cube. The vocabulary is the same with the 2D case, except that there are 88 different object words and the vocabulary size is now 136. In total, there are 709,383 distinct sentences that can be generated by the teacher according to the same set of syntax rules in the 2D case. The lengths of these sentences also range from 1 to 15.

Results. The training curves of success rates are shown in Figure 3 (b). We observe that **GFT-1** already has a huge advantage over the best-performing comparison method **FiLM**. On top of this, **GFT-2** shows a faster performance increase during the training time. This suggests that the original feature space cannot easily comply with various language commands. A rotation of the feature space depending on the input command, an operation which **FiLM** lacks but **GFT-1** owns, is important for producing better grounding results. Table 2 (3D) shows the test results split into five navigation types. Unlike in the 2D case, now for **GFT-2**, `nav` has the second-best success rate while `nav_dir` has the lowest rate. Visualization of the `nav_dir` test cases reveals that due to severe perspective distortion in 3D, the agent has some difficulty of grounding the spatial-relation words, especially when multiple objects are located nearby. But still, GFT agents obtain much better `nav_dir` results than the comparison methods (over a 15% increase on average). Several example navigation sessions are shown in Figure 5 Appendix F.

5.5 Limitations

While GFT is theoretically more expressive and empirically better than some of the existing language grounding modules, there are certain limitations of it. First, because each $\mathbf{T}_j$ is generated from the command, it requires a large projection matrix. In our implementation, the projection matrix that converts a hidden sentence embedding of length 128 to $\mathbf{T}_j$ has a size of $128 \times D(D+1) = 128 \times 4160$ assuming $D = 64$. Thus usually GFT has more parameters to be learned compared with its simplified versions like FiLM. An alternative might be to explicitly constrain $\mathbf{T}_j$ to be sparse and possibly low-rank. Second, GFT performs several steps of transformations, each of which has a different transformation matrix. This further linearly increases the number of learnable parameters. One solution would be to set $\mathbf{T}_1 = \ldots = \mathbf{T}_j = \ldots = \mathbf{T}_J$, i.e., do a recurrent feature transformation. However, this method has been observed slightly worse than the current GFT in the performance. Third, depending on the actual value of J, GFT might be slower in computation than other gated networks which perform a single transformation. These three issues are left to our future work.

6 Conclusions

We have presented GFT, a simple but general neural language grounding module for embodied agents. GFT provides a unifying view of some existing language grounding modules, and further generalizes on top of them. The evaluation results on two challenging navigation environments suggest that GFT can be easily adapted from one problem to another robustly. Although evaluated on navigation, we believe that GFT could potentially serve as a general-purpose language grounding module for embodied agents that need to follow language instructions in a variety of scenarios.

References

[1] L. Smith and M. Gasser. The development of embodied cognition: Six lessons from babies. *Artificial Life*, 11(1-2), 2005.

[2] D. Kiela, L. Bulat, A. L. Vero, and S. Clark. Virtual embodiment: A scalable long-term strategy for artificial intelligence research. In *NIPS Workshop*, 2016.

[3] J. Oh, S. P. Singh, H. Lee, and P. Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In *ICML*, 2017.

[4] K. M. Hermann, F. Hill, S. Green, F. Wang, R. Faulkner, H. Soyer, D. Szepesvari, W. M. Czarnecki, M. Jaderberg, D. Teplyashin, M. Wainwright, C. Apps, D. Hassabis, and P. Blunsom. Grounded language learning in a simulated 3d world. In *NIPS Workshop*, 2017.

[5] D. S. Chaplot, K. M. Sathyendra, R. K. Pasumarthi, D. Rajagopal, and R. Salakhutdinov. Gated-attention architectures for task-oriented language grounding. In *AAAI*, 2018.

[6] A. Das, S. Datta, G. Gkioxari, S. Lee, D. Parikh, and D. Batra. Embodied Question Answering. In *CVPR*, 2018.

[7] H. Yu, H. Zhang, and W. Xu. Interactive grounded language acquisition and generalization in a 2d world. In *ICLR*, 2018.

[8] S. Harnad. The symbol grounding problem. *Philosophical Explorations*, 42, 1990.

[9] J. M. Siskind. Grounding language in perception. *Artificial Intelligence Review*, 1994.

[10] R. S. Sutton and A. G. Barto. *Reinforcement Learning : An Introduction*. MIT Press, 1998.

[11] A. V. Clemente, H. N. Castejón, and A. Chandra. Efficient Parallel Methods for Deep Reinforcement Learning. *arXiv preprint arXiv:1705.04862*, 2017.

[12] T. Shu, C. Xiong, and R. Socher. Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. In *ICLR*, 2018.

[13] Y. Wu, Y. Wu, G. Gkioxari, and Y. Tian. Building generalizable agents with a realistic and rich 3d environment. 2018.

[14] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. Lawrence Zitnick, and D. Parikh. VQA: Visual question answering. In *ICCV*, 2015.

[15] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville. FiLM: Visual Reasoning with a General Conditioning Layer. In *AAAI*, 2018.

[16] Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1), 1998.

[17] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 1999.

[18] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

[19] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*, 2014.

[20] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *ICLR*, 2017.

[21] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, and R. Hadsell. Learning to navigate in complex environments. In *ICLR*, 2017.

[22] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning. In *ICRA*, 2017.

[23] Y. Zhu, D. Gordon, E. Kolve, D. Fox, L. Fei-Fei, A. Gupta, R. Mottaghi, and A. Farhadi. Visual semantic planning using deep successor representations. In *ICCV*, 2017.

[24] S. Brahmbhatt and J. Hays. Deepnav: Learning to navigate large cities. In *CVPR*, 2017.

[25] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. In *CVPR*, 2017.

[26] M. Savva, A. X. Chang, A. Dosovitskiy, T. Funkhouser, and V. Koltun. MINOS: Multimodal indoor simulator for navigation in complex environments. *arXiv preprint arXiv:1712.03931*, 2017.

[27] P. Mirowski, M. K. Grimes, M. Malinowski, K. M. Hermann, K. Anderson, D. Teplyashin, K. Simonyan, K. Kavukcuoglu, A. Zisserman, and R. Hadsell. Learning to navigate in cities without a map. 2018.

[28] S. Gupta, D. F. Fouhey, S. Levine, and J. Malik. Unifying map and landmark based representations for visual navigation. *CoRR*, abs/1712.08125, 2017.

[29] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. van den Hengel. Vision-and-Language Navigation: Interpreting visually-grounded navigation instructions in real environments. In *CVPR*, 2018.

[30] D. Gordon, A. Kembhavi, M. Rastegari, J. Redmon, D. Fox, and A. Farhadi. IQA: visual question answering in interactive environments. In *CVPR*, 2018.

[31] J. Lu, J. Yang, D. Batra, and D. Parikh. Hierarchical question-image co-attention for visual question answering. In *NIPS*, 2016.

[32] Z. Yang, X. He, J. Gao, L. Deng, and A. Smola. Stacked attention networks for image question answering. In *CVPR*, 2016.

[33] H. Yu and J. M. Siskind. Grounded language learning from video described with sentences. In *ACL*, 2013.

[34] Q. Gao, M. Doering, S. Yang, and J. Y. Chai. Physical causality of action verbs in grounded language understanding. In *ACL*, 2016.

[35] A. Rohrbach, M. Rohrbach, R. Hu, T. Darrell, and B. Schiele. Grounding of textual phrases in images by reconstruction. In *ECCV*, 2016.

[36] D. L. Chen and R. J. Mooney. Learning to interpret natural language navigation instructions from observations. In *AAAI*, 2011.

[37] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. Teller, and N. Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*, 2011.

[38] D. P. Barrett, S. A. Bronikowski, H. Yu, and J. M. Siskind. Driving under the influence (of language). *IEEE Transactions on Neural Networks and Learning Systems*, 2017.

[39] T. Tieleman and G. Hinton. Lecture 6.5 - RMSprop. *COURSERA: Neural Networks for Machine Learning*, 2012.

[40] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technology Journal*, 36, 1957.

[41] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *ICML*, 2009.

Appendices

A Agent Architecture

The agent history $\mathbf{h}^{[t]}$ has two constituents: an action history $\mathbf{h}_a^{[t]}$ summarizing previous taken actions and a visual history $\mathbf{h}_m^{[t]}$ summarizing previous visual experience. We instantiate f_θ and h_θ as GRUs (Figure 2):

$$\begin{aligned} \mathbf{h}_m^{[t]} &= GRU(m_\theta(\mathbf{o}^{[t]}, \mathbf{l}), \mathbf{h}_m^{[t-1]}), \\ \mathbf{h}_a^{[t]} &= GRU(a^{[t]}, \mathbf{h}_a^{[t-1]}), \\ \mathbf{f}^{[t]} &= GRU(\mathbf{h}_m^{[t]}, \mathbf{h}_a^{[t-1]}, \mathbf{f}^{[t-1]}), \\ \mathbf{h}^{[t]} &= (\mathbf{h}_a^{[t]}, \mathbf{h}_m^{[t]}). \end{aligned} \tag{8}$$

Our RL training design is synchronous advantage actor-critic (ParallelA2C) [11]. We run N_{agent} agents in parallel with model parameters shared among them to encourage exploration and reduce variance in the policy gradient. Each backpropagation is done with a minibatch of N_{batch} time steps collected from all the agents, each agent contributing $\frac{N_{\text{batch}}}{N_{\text{agent}}}$ time steps. In either environment, every agent gets blocked until the network parameters are updated with the current minibatch, after which it forwards $\frac{N_{\text{batch}}}{N_{\text{agent}}}$ time steps again with the updated model parameters. To speed up training, we adopt an n-step temporal difference (TD) when computing the advantage $A^{[t]}$ (Eq. 6), in a forward manner similar to Mnih et al. [18]. Finally, we empirically set the discount γ to 0.99, the entropy weight κ to 0.05, and the value regression weight η to 1.0 throughout the experiments.

Sometimes an agent might provide fewer than $\frac{N_{\text{batch}}}{N_{\text{agent}}}$ actions for each minibatch due to the end of an episode, because once an agent hits an episode end, it will be initialized in a new session for the next minibatch. Thus the actual size of the minibatch might be smaller than N_{batch}. In the experiments, we set $N_{\text{agent}} = 32$, and $N_{\text{batch}} = 128$.

The agent is trained purely from reward signals, without:

1) prior visual knowledge such as a pre-trained CNN,
2) prior linguistic knowledge such as a parser, or
3) any auxiliary task such as image reconstruction [6, 4], reward prediction [4], or language prediction [4, 7].

B Method Details

General. The sentence embedding $\mathbf{l}_{\text{BoW}}$ is obtained by sum-pooling word embeddings. A word embedding has a length of 128, except for the **Concept** method (see below). The agent perceives 84×84 egocentric RGB images in XWORLD3D and 80×80 egocentric RGB images in XWORLD2D. Regardless of the image dimensions, the CNN has three convolutional layers for processing the image: $(8, 4, 32)$, $(4, 2, 64)$, and $(3, 1, 64)$, where (a, b, c) represents a layer configuration of c filters of size $a \times a$ at a stride of b. Each action is embedded as a vector of size 128 before being input to GRU_θ^a to generate the action history $\mathbf{h}_a$ which also has 128 units. The other two recurrent layers GRU_θ^m and GRU_θ^f both have 512 units, and both have an extra hidden layer of size 512 to preprocess their inputs. The policy network is a two-layer MLP where the first layer has 512 units and the second layer is a softmax for outputting actions. The value network is a two-layer MLP where the first layer has 512 units and the second layer outputs a scalar value without any activation. Unless otherwise stated, all the layer outputs are ReLU activated.

Concat. Both $\mathbf{l}_{\text{BoW}}$ and $\mathbf{C}$ are projected to a latent space of 512 dimensions.

Gated. The MLP for generating $\mathbf{l}_{\text{gate}}$ from $\mathbf{l}_{\text{BoW}}$ has two layers: the first layer has 128 units and the second layer has $D = 64$ units which are sigmoid activated.

CGated. The gate vector $\mathbf{l}_{\text{gate}}$ has 512 units which are sigmoid activated.

FiLM. The MLP for generating λ_d and b_d has two layers: the first layer has 128 units and the second layer has $D + 1 = 65$ units without any activation.

Concept. Because the sentence embedding $\mathbf{l}_{\text{BoW}}$ is directly used as the 1×1 filter, each word embedding has a length of $D = 64$. The attention map and environment map are both ReLU activated.

EncDec. This method has a slightly reorganized computational flow compared to the one in Eq. 8. We refer the reader to the original paper [15] for details. Except this, the configuration of each layer is the same with that of its counterpart that can be found in **Concat**.

GFT. The MLP for generating the transformation matrix $\mathbf{T}_j$ from $\mathbf{l}_{\text{BoW}}$ has two layers: the first layer has 128 units and the second layer has $D \times (D+1) = 64 \times 65 = 4160$ units. The second layer has no activation. For **GFT-2**, we share the parameters of the first layers between $\mathbf{T}_1$ and $\mathbf{T}_2$. We set the activation function g in Eq. 1 as ReLU.

C Vocabulary

The following 8 spatial-relation words and 40 grammatical words are shared between the XWORLD2D and XWORLD3D.

Spatial-relation (8)	**Grammatical** (40)
behind,	!, ., ?, and, anything, avoid,
besides,	but, can, collect, could, destination,
between,	do, done, end, except, go,
by,	goal, grid, in, is, location,
front,	move, navigate, not, object, of,
left,	place, please, reach, target, that,
near,	the, time, to, up, well,
right.	will, wrong, you, your.

The two environments have two different sets of object words:

Object	
XWORLD3D (88)	XWORLD2D (115)
apples, backpack, barbecue, barrel, basket,	apple, armadillo, artichoke, avocado, banana, bat,
basketball, bathtub, bed, bench, boiler,	bathtub, beans, bear, bed, bee, beet,
books, bookshelf, bottle, bread, brush,	beetle, bird, blueberry, bookshelf, broccoli, bull,
bucket, burger, cake, calender, camera,	butterfly, cabbage, cactus, camel, carpet, carrot,
candle, car, carpet, cat, chair,	cat, centipede, chair, cherry, clock, coconut,
chessboard, clock, comb, cooker, crib,	corn, cow, crab, crocodile, cucumber, deer,
cup, dart, dog, dog-house, drawers,	desk, dinosaur, dog, donkey, dragon, dragonfly,
drums, fan, fence, firehydrant, flashlight,	duck, eggplant, elephant, fan, fig, fireplace,
flowers, fountain, gift, guitar, hair-dryer,	fish, fox, frog, garlic, giraffe, glove, horse,
headphones, horse, iron, lamp, laptop,	goat, grape, greenonion, greenpepper, hedgehog,
mailbox, milk, oven, pan, phone,	kangaroo, knife, koala, ladybug, lemon, light,
photo, piano, pillow, plant, pool-table,	lion, lizard, microwave, mirror, monitor, monkey,
puzzle, rabbit, rooster, scale, scissors,	monster, mushroom, octopus, onion, orange, ostrich,
screen, slippers, sofa, speaker, squeezer,	owl, panda, peacock, penguin, pepper, pig,
staircase, stove, sunglasses, table,	pineapple, plunger, potato, pumpkin, rabbit, racoon,
table-tennis, toilet, towel, train,	rat, rhinoceros, rooster, seahorse, seashell, seaurchin,
trampoline, trashcan, treadmill, tricycle,	shrimp, snail, snake, sofa, spider, squirrel,
umbrella, vacuum, vase, wardrobe,	stairs, strawberry, tiger, toilet, tomato, turtle,
wheelchair.	vacuum, wardrobe, washingmachine, watermelon,
	whale, wheat, zebra.

D Curriculum Learning

To help the agent learn, we adopt curriculum learning [41] to gradually increase the environment size and complexity, according to the curriculum in Table 3. The training always starts from level 1.

Level	**Map size** ($X = Y$)	**number of goals per map**	**number of obstacles per map**
1	3	2	0
2	4	2	3
3	5	2	6
4	6	4	9
5	7	4	12
6	8	4	16

Table 3: The curriculum used for training the agents.

	`nav`		`nav_avoid`		`nav_bw`		`nav_dir`		`nav_near`	
	2D	3D	2D	3D	2D	3D	2D	3D	2D	3D
Concat	46.1±4.9	63.6±4.2	94.1±1.3	96.5±1.2	72.2±6.2	72.1±4.1	46.2±26.1	38.8±3.8	47.1±11.7	49.6±4.3
Gated	49.5±4.1	65.6±4.2	94.4±1.8	95.8±1.7	**77.7**±4.8	72.6±5.4	66.4±5.0	37.7±4.1	55.5±4.7	49.2±3.5
CGated	47.1±3.2	63.8±4.5	94.6±1.6	96.1±1.7	75.5±4.1	74.0±4.6	62.8±4.5	39.2±4.2	56.2±4.0	52.5±4.1
FiLM	49.0±3.6	68.2±3.1	94.4±1.6	97.4±1.1	76.9±3.3	74.0±3.8	67.5±5.1	43.3±3.1	57.5±4.4	56.6±2.8
Concept	55.1±3.9	55.6±17.1	96.3±1.5	95.0±4.6	73.7±3.5	53.3±21.0	56.2±5.2	27.6±12.7	57.5±4.5	43.7±14.7
EncDec	40.3±7.3	56.0±4.0	89.4±6.6	95.3±2.1	66.9±10.3	69.8±4.5	20.6±25.4	38.1±3.9	43.3±12.6	50.3±3.7
GFT-1	55.1±2.8	**78.3**±3.0	95.3±1.6	97.8±1.0	76.8±3.0	**78.3**±4.4	70.8±3.5	**52.3**±5.4	60.5±2.8	**66.9**±3.8
GFT-2	**60.9**±5.1	76.2±3.6	**96.8**±1.3	**98.1**±0.9	77.6±3.8	76.3±4.1	**71.4**±3.7	51.8±5.8	**65.1**±4.2	66.8±4.3

Table 4: The evaluation results for 10k test sessions on 9×9 maps. The average navigation success rates are reported as percentages. The second number in each cell represents the standard deviation over twelve test runs, each run corresponding to one pass (out of three final passes) of one trained model (out of four random initializations).

	`nav`		`nav_avoid`		`nav_bw`		`nav_dir`		`nav_near`	
	2D	3D	2D	3D	2D	3D	2D	3D	2D	3D
Concat	44.2±3.2	59.3±2.2	94.9±1.6	96.5±1.0	70.3±6.7	70.9±4.6	43.8±24.9	35.7±3.0	47.6±12.1	48.2±4.2
Gated	45.9±4.6	64.1±3.1	94.4±1.6	95.7±1.8	76.7±3.9	71.1±4.1	63.5±4.0	36.2±4.4	56.2±5.6	50.6±4.9
CGated	44.9±3.8	64.2±5.0	93.9±1.5	96.9±1.0	76.3±4.9	70.0±4.4	63.4±4.2	37.5±4.3	56.6±3.8	52.5±2.9
FiLM	46.7±3.7	66.1±4.2	93.5±2.5	97.6±1.2	77.6±3.7	72.4±3.7	64.7±4.8	41.2±4.8	57.4±3.1	56.4±2.7
Concept	44.2±3.2	55.5±18.3	94.9±1.6	94.5±5.9	70.3±6.7	50.9±21.2	43.8±24.9	24.1±11.0	47.6±12.1	40.2±15.6
EncDec	37.8±6.0	53.9±6.1	91.4±5.5	95.8±2.0	66.7±12.3	69.0±3.4	18.6±23.7	35.9±3.5	41.9±11.0	47.4±3.6
GFT-1	52.7±3.4	78.1±4.3	95.9±1.7	98.3±0.9	**78.7**±2.9	75.5±4.3	66.6±3.5	51.6±3.8	60.5±2.6	**68.3**±3.5
GFT-2	**60.7**±2.7	**78.8**±3.2	**97.3**±1.4	**98.6**±1.3	78.6±6.2	**76.2**±3.5	**69.2**±4.0	**52.4**±4.5	**65.7**±4.0	68.2±5.2

Table 5: The evaluation results for 10k test sessions on 10×10 maps. The average navigation success rates are reported as percentages. The second number in each cell represents the standard deviation over twelve test runs, each run corresponding to one pass (out of three final passes) of one trained model (out of four random initializations).

	`nav`		`nav_avoid`		`nav_bw`		`nav_dir`		`nav_near`	
	2D	3D	2D	3D	2D	3D	2D	3D	2D	3D
Concat	36.2±5.2	44.4±3.9	96.3±1.5	94.3±1.3	64.9±7.0	54.7±5.5	42.0±23.0	25.9±3.0	41.2±12.8	31.9±3.9
Gated	40.3±5.2	47.3±5.7	96.0±1.8	94.9±2.0	70.5±4.6	58.9±6.1	58.4±5.0	25.2±3.0	48.7±3.6	33.5±4.0
CGated	35.9±3.0	46.0±4.0	95.3±2.0	95.3±1.4	72.4±5.1	54.7±5.2	54.0±4.0	24.5±2.5	46.7±3.9	32.6±3.1
FiLM	39.9±4.4	53.2±4.6	96.8±1.1	95.5±1.0	72.4±3.7	59.1±5.4	59.3±5.0	28.3±3.4	49.2±4.3	36.6±3.9
Concept	44.7±3.7	42.9±15.7	97.5±1.4	95.1±3.0	68.7±3.5	36.1±16.4	49.1±4.1	17.4±9.6	48.7±3.9	27.6±11.2
EncDec	31.7±5.9	38.6±4.6	93.5±4.3	94.1±1.3	62.3±11.6	56.5±3.3	16.1±21.7	25.9±4.3	38.0±11.5	32.4±2.8
GFT-1	43.9±3.7	62.6±4.0	96.0±1.2	95.9±1.5	**74.3**±3.3	**64.4**±4.6	61.7±3.0	**38.7**±4.0	53.5±5.5	49.5±3.6
GFT-2	**51.5**±3.5	**63.2**±5.2	**98.0**±1.3	**96.0**±1.5	72.4±4.5	64.2±5.4	**64.1**±5.5	37.5±3.5	**58.9**±3.5	**53.4**±4.2

Table 6: The evaluation results for 10k test sessions on 11×11 maps. The average navigation success rates are reported as percentages. The second number in each cell represents the standard deviation over twelve test runs, each run corresponding to one pass (out of three final passes) of one trained model (out of four random initializations).

During training, the teacher maintains the average success rate of each task type (Table 1), for a total of 200 most recent sessions. If at some point, all the five average success rates are above a predefined threshold of 0.7, then the teacher allows the agent to enter the next level and resets the maintained rates. The progress of this curriculum is computed separately for each of the 32 agents running in parallel. The above curriculum applies to all the methods in both environments. It should be noted that the training curves and test results in Section 5 are computed for the final level with the maximal difficulty, without being affected by the curriculum.

E Generalization to Larger Maps

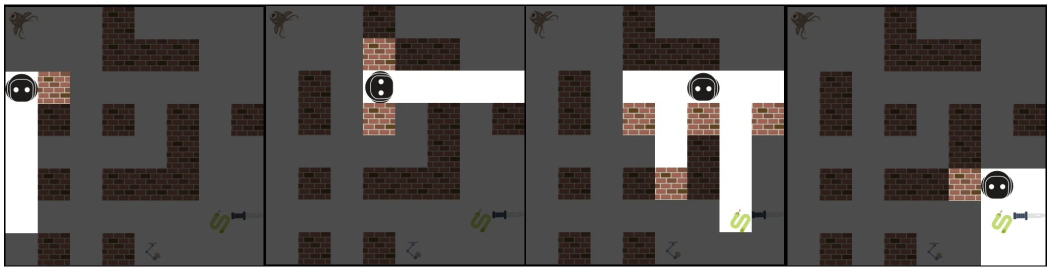
We evaluate the agent models, trained for $X = Y = 8$ in Section 5, on three larger maps:

i) $X = Y = 9$, with 6 goals and 20 obstacles,
ii) $X = Y = 10$, with 6 goals and 24 obstacles, and
iii) $X = Y = 11$, with 8 goals and 28 obstacles.

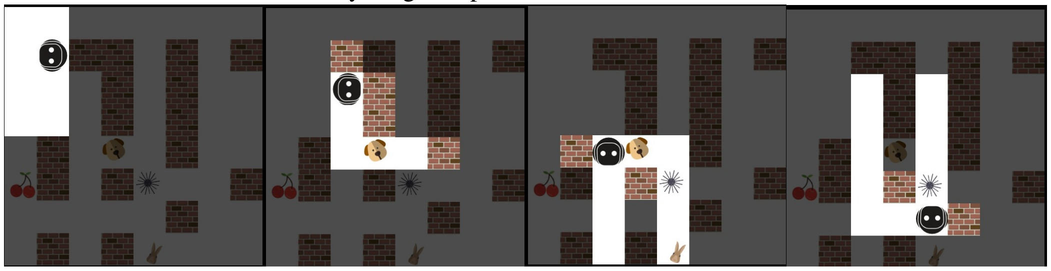
The evaluation results are shown in Table 4, Table 5, and Table 6, respectively. Although the performance is not as good as on 8×8 maps (except for `nav_avoid` whose chance performance tends to peak as there are more goals on the map before the map size becoming too large), the GFT agents achieve reasonable generalizations and still greatly outperform the comparison methods. This further demonstrates that our GFT agents are not trained to memorize environments in specific settings.

F Navigation Examples

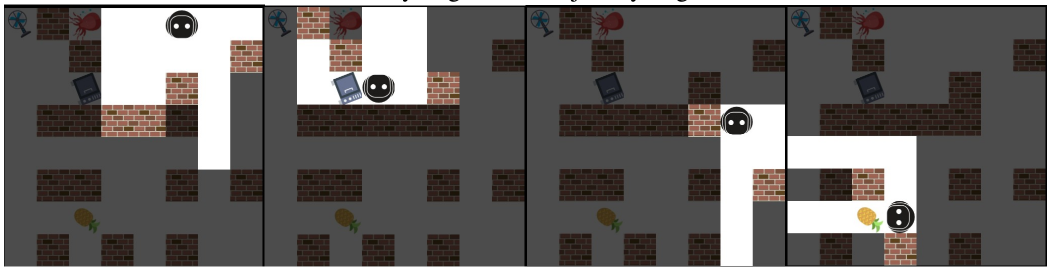
Below we show some navigation examples for the **GFT-2** agents trained in 2D and 3D. For each session, we present four key frames on the navigation path. More full-length navigation sessions are shown in a video demo at `https://www.youtube.com/watch?v=bOBb1uhuJxg`.



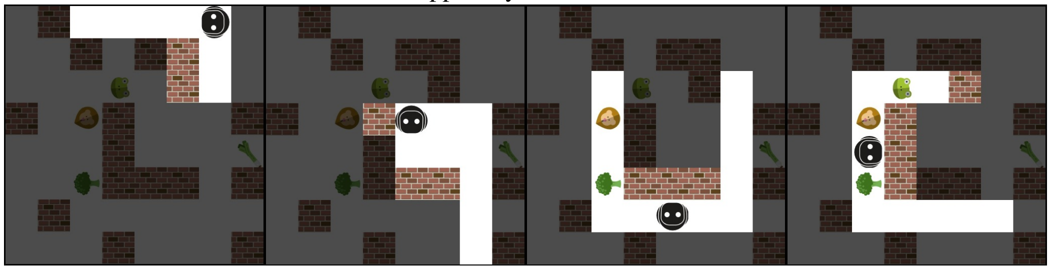
"Anything except deer is the destination."



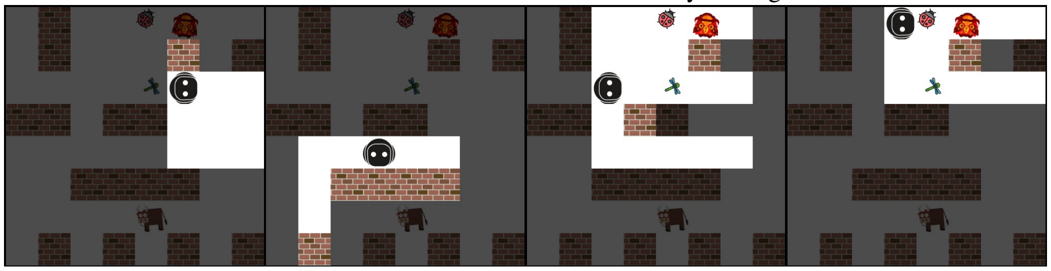
"Will you go to the object by dog?"



"Pineapple is your destination."

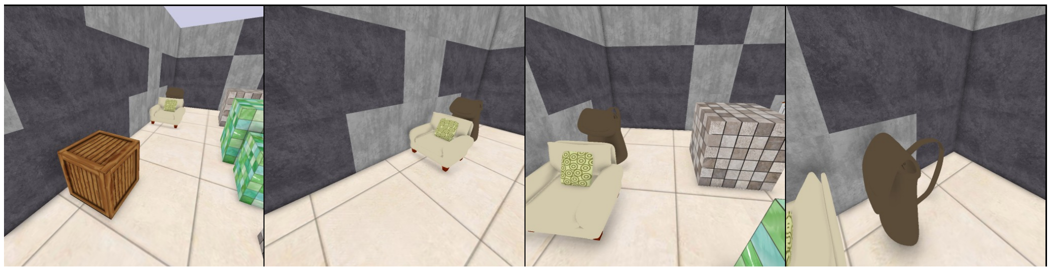


"The location between lion and broccoli is your target."

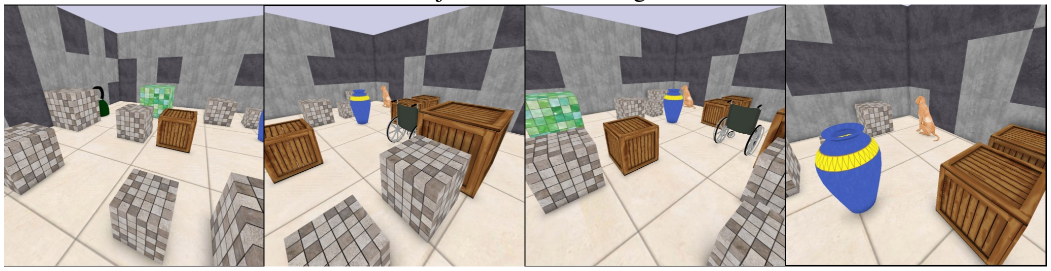


"Collect the object in front of dragon."

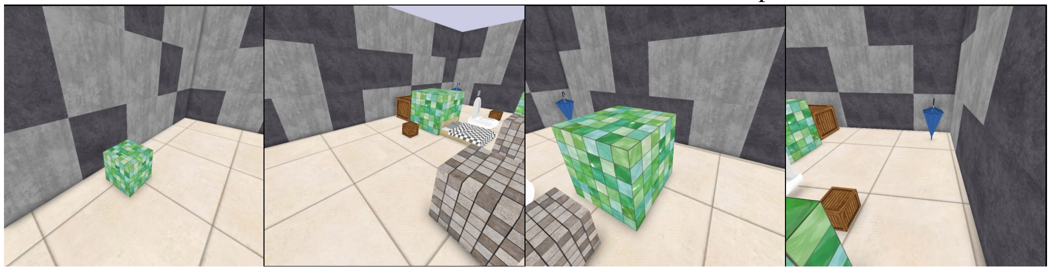
Figure 4: Five navigation examples for the **GFT-2** agent trained in XWORLD2D. Four key frames in temporal order are shown in each example. During navigation, the agent is able to see only the highlighted regions. The shaded regions are for visualization purpose.



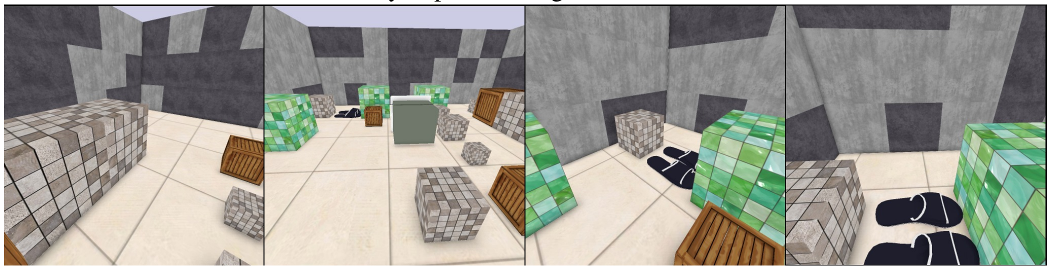
"Reach the object that is to the right of sofa."



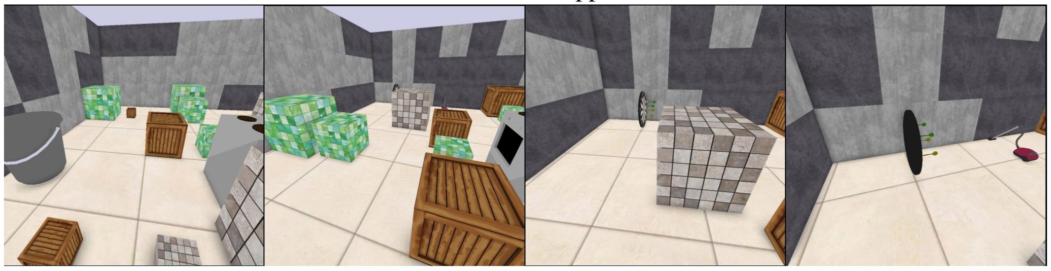
"Move to the location between vase and wheelchair please."



"Could you please navigate to umbrella?"



"Please reach slippers."



"Go to the grid between dart and vaccum."

Figure 5: Five navigation examples for the **GFT-2** agent trained in XWORLD3D. Four key frames in temporal order are shown in each example.

G Visualization of Transformation Matrices

In this section, we visualize the transformation matrices of several example commands received by the **GFT-2** agent trained in XWORLD3D. We are particularly interested in comparing the $\mathbf{T}_1$ and $\mathbf{T}_2$ of two different commands that have the same semantics or have a minimal semantic difference. Figure 6 shows the visualization results. Unsurprisingly, we observe that

a) Two completely different commands with the same semantics (for our problem) will yield almost identical transformation matrices (Figure 6 row 1).
b) Two commands with a minimal difference will yield matrices similar in general but differ in small places that capture the difference (Figure 6 rows 2-4).

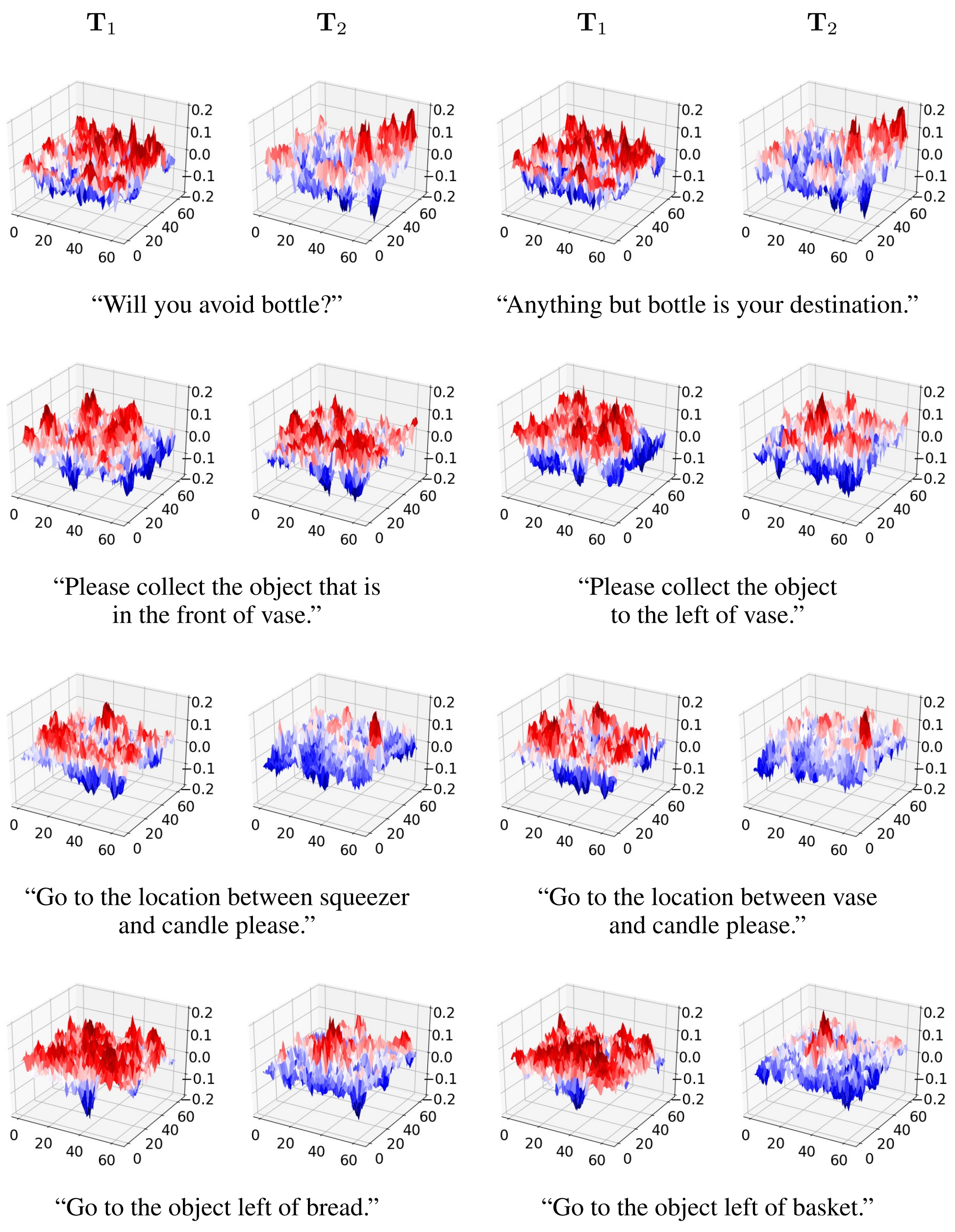


Figure 6: Visualization of the transformation matrices computed by the trained **GFT-2** agent for four example command pairs. For a better view, the matrices have been subtracted by the average $\mathbf{T}_1$ or $\mathbf{T}_2$ (computed from 5k randomly sampled commands) to remove the biases. Then each matrix is smoothed by a 7×7 uniform kernel.