Few-shot Subgoal Planning with Language Models

Lajanugen Logeswaran[*], **Yao Fu**[‡], **Moontae Lee**[*†], **Honglak Lee**[*‡]
LG AI Research[*], University of Illinois at Chicago[†], University of Michigan[‡]

Abstract

Pre-trained language models have shown successful progress in many text understanding benchmarks. This work explores the capability of these models to predict actionable plans in real-world environments. Given a text instruction, we show that language priors encoded in pre-trained models allow us to infer fine-grained subgoal sequences. In contrast to recent methods which make strong assumptions about subgoal supervision, our experiments show that language models can infer detailed subgoal sequences from few training sequences without any fine-tuning. We further propose a simple strategy to re-rank language model predictions based on interaction and feedback from the environment. Combined with pre-trained navigation and visual reasoning components, our approach demonstrates competitive performance on subgoal prediction and task completion in the ALFRED benchmark compared to prior methods that assume more subgoal supervision.

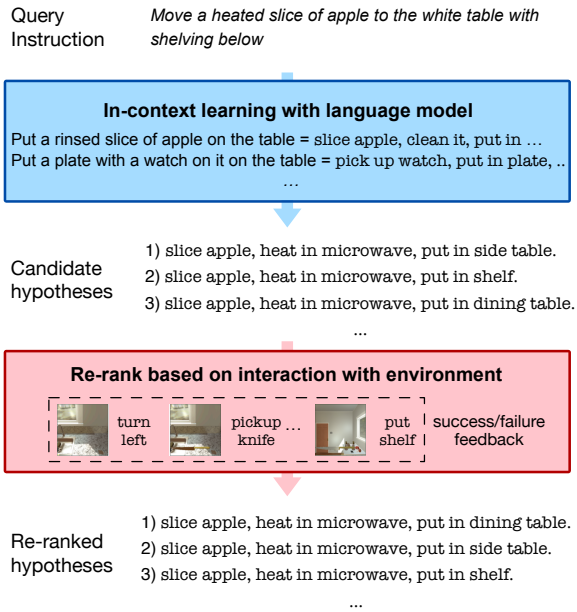

Figure 1: Overview of subgoal prediction approach. (a) A pre-trained language model prompted with a sequence of training instances, i.e., (instruction, subgoal sequence) pairs, and a query instruction predicts top-k hypotheses using beam search. (b) These predictions are then re-ranked by incorporating information about the environment.

1 Introduction

Developing autonomous agents that can complete specific tasks given goal descriptions embodies human-level intelligence. Successful agents in this setting require multiple reasoning capabilities including natural language understanding, visual reasoning, and acting over long temporal horizons. Training black-box models that map instructions and observations to suitable actions has proven to be difficult due to challenges in interpreting and reasoning with multimodal information, especially in the absence of strong supervision. Thus, generalization in this setting demands effective strategies for planning, exploration, and incorporating feedback from the environment.

Generalization in human agents, on the other hand, stems from our ability to naturally brainstorm abstract subgoals, better calibrating executable actions and their sequences. Planning at the level of subgoals instead of low-level actions allows us to better adapt to unfamiliar settings. We posit that language supervision can help realize such planning capabilities effectively in artificial agents. First, text is a natural API for interacting with intelligent agents that act in the real world to complete tasks. Knowledge available in the form of text corpora, descriptions and instructions can be exploited to build better agents (Branavan et al., 2012; Zhong et al., 2019). Second, strong language priors are useful to reason about causal sequences of events (Li et al., 2021). Language priors can further inform about object affordances (e.g. an apple is sliceable, whereas a table is not) and other contextual knowledge (e.g. a slicing task is more likely to be performed in a kitchen than a bathroom) (Chen

[*]Correspondence to `llajan@lgresearch.ai`

et al., 2020). Recent advances have demonstrated that large language models are able to capture such priors, as evidenced by their strong capabilities in language understanding and beyond (Devlin et al., 2018; Radford et al., 2019; Brown et al., 2020; Bommasani et al., 2021). This leads to the natural question of whether priors learned by language models can help reason about subgoals.

We study the ability of language models to reason about plans composed of a sequence of intermediate goals for completing basic object manipulation tasks in a household environment specified using text instructions. In particular, we use the *in-context learning* ability (Brown et al., 2020) of large pre-trained language models to reason about subgoals. In contrast to prior methods that fine-tune language models to predict subgoals/actions (Jansen, 2020; Yao et al., 2020), we show that they can predict subgoal sequences effectively without any fine-tuning. We teach the model how instructions translate into subgoal sequences by constructing a prompt using few examples. Given the prompt and a query instruction the language model predicts likely subgoal sequences (see Figure 1 for an illustration).

While language models are capable of generating strong hypotheses, we observe that these predictions may not be directly usable by agents acting in real environments. First, they suffer from calibration issues: Language models have a tendency to repeat content from the prompt (Zhao et al., 2021). We show that mutual-information inspired metrics help mitigate calibration issues and lead to better ranking of model generated hypotheses.

Second, real-world agents have to update their beliefs and predictions based on interaction and feedback from the environment. Without such feedback we cannot expect the predicted plan to be executable in the environment. We execute plans proposed by the language model in the environment using a pre-trained low-level policy and collect feedback about task success/failure. We use this feedback as a learning signal to train a ranking model that re-ranks language model predictions. In contrast to prior methods that rely on strong subgoal supervision and task level expert trajectories, we show that combining subgoal predictions with a pre-trained subgoal execution policy leads to a strong embodied agent baseline.

We make the following contributions in this work. We show that

- Large language models can predict subgoals from text instructions with very little supervision using in-context learning.
- Incorporating a small amount of feedback from interaction with the environment such as agent state and task success/failure outcome improves language model predictions.
- Combining predicted subgoals with a pre-trained low-level policy for navigation and visual reasoning leads to a simple modular agent policy that performs well on an embodied learning setting.

2 Related work

Language models for planning and interaction The use of language models for planning and action prediction has been explored in prior work. Jansen (2020) fine-tuned a language model to predict subgoal sequences for text instructions from the ALFRED benchmark. Micheli and Fleuret (2021) take a similar approach, but show that imitation learning with few instances combined with reinforcement learning produces models that work well on the ALFWorld benchmark (Shridhar et al., 2020b). Yao et al. (2020) demonstrate a similar approach for interactive fiction games (Hausknecht et al., 2020). In contrast to these prior methods, our approach does not assume strong supervision and we demonstrate generalization with limited training examples. Furthermore, in order to exploit the generalization capabilities of large language models, we do not fine-tune these models and instead use their in-context learning ability. Finally, our approach allows us to build policies that inherit the strong generalization capabilities of these large pre-trained models such as compositional generalization.

Large language models and few-shot learning Brown et al. (2020) showed that pre-trained large language models have few-shot learning capabilities. Given a few examples $\{(x_i, y_i = f(x_i))\}$ that define a task f such as classification or translation and a query instance x^q, *prompting* a language model with a string such as "$x_1 = y_1; x_2 = y_2; ...; x_n = y_n; x^q =$" leads to meaningful completions by the language model $y^q \approx f(x^q)$. This few-shot learning capability of language models has since then been studied and improved upon with approaches like prefix engineering (Schick and Schütze, 2020), prompt tuning (Li and Liang, 2021), model calibration (Zhao et al., 2021) and other methods (Min et al., 2021a). We adopt a

similar approach for few-shot subgoal inference. We assume that subgoal supervision is available for a small number of training tasks and use the language model to infer subgoals for unseen tasks.

Instruction following There is rich literature on agents that follow language instructions (Mei et al. (2016); Fried et al. (2017); Suhr et al. (2019) *inter alia*[1]). Recent developments in simulated environments and benchmarks with human annotated instructions have driven progress in embodied agents that learn from text instructions (Shridhar et al., 2020a; Kolve et al., 2017). Successful agents in these settings require multiple reasoning capabilities including language understanding, visual reasoning and learning to act over long time-horizons. Compared to the sequence to sequence model originally proposed in Shridhar et al. (2020a), subsequent approaches have made progress by exploiting subgoal supervision, pre-trained visual reasoning components and pre-trained transformer models (Singh et al., 2020; Suglia et al., 2021; Zhang and Chai, 2021; Corona et al., 2020; Blukis et al., 2021). Unlike these methods, we do not assume access to strong subgoal supervision or task level expert supervision. We combine language model predictions with pre-trained low-level navigation and interaction policies to obtain a competitive agent policy.

Few-shot semantic parsing Subgoal inference from text instructions can be considered a semantic parsing problem where the subgoal sequences serves as a formal representation of text. Shin et al. (2021) show that few-shot semantic parsers can be derived from language models and demonstrate their applicability on text-to-SQL (Finegan-Dollak et al., 2018) and SCAN (Lake and Baroni, 2018) benchmarks. Furrer et al. (2020) and Herzig et al. (2021) further study the compositional generalization ability of such semantic parsers. In our work we make use of ideas introduced in these works such as dynamic prompt creation, constrained decoding and intermediate representations.

3 Approach

We first consider subgoal inference as a semantic parsing problem where a text instruction needs to be translated to a sequence of subgoals and propose an approach to few-shot subgoal inference based on pre-trained language models in Section 3.1. We extend this setting to an agent acting in a simulated environment which can execute these subgoals, observe feedback, and improve upon language model predictions for more accurate subgoal inference in Section 3.2.

[1]See Luketina et al. (2019) for a survey

3.1 Few-shot subgoal inference

Subgoals We are interested in a particular subclass of instruction following problems which involve performing a sequence of object interactions in an embodied environment. Each object interaction requires navigating to a particular object and performing an action on it. A task is considered successfully completed if the state of objects in the end satisfy a set of task-specific constraints (for instance, objects that need to be sliced/warmed/cooled/cleaned have the appropriate state change). It is thus natural to define a subgoal as one or more sequence of object interactions. A subgoal g is specified as $g = (b, o) \in \mathcal{B} \times \mathcal{O}$ where $b \in \mathcal{B} =$ {Pickup, Clean, Heat, ..} is one of a pre-defined set of abstract actions and $o \in \mathcal{O} =$ {Apple, Microwave, DeskLamp, Ottoman, ..} is an object category.

Subgoal inference problem Given a text instruction τ, subgoal inference seeks to predict a sequence of subgoals $\tau \mapsto g = (g^{(1)}, ..., g^{(n)})$. To perform in-context learning with a language model, we consider a representation $v(g)$ of g that looks like natural text, where v is a pre-defined invertible mapping. Such representations have been referred to in the literature as verbalizers (Min et al., 2021a), intermediate representations (Herzig et al., 2021) and canonical representations (Shin et al., 2021), where the purpose is to represent the output in a format the language model understands. In a slight abuse of notation, we will use g to refer to either a subgoal sequence or it's textual representation $v(g)$ depending on the context.

Generating subgoals We assume that a small amount of training data $\{(\tau_1, g_1), \cdots, (\tau_n, g_n)\}$ is given. The language model is prompted with a comma separated concatenation of the training examples, each in the format "$\tau_i = g_i$", followed by a query τ, formatted as "$\tau =$". We assume that the probability of a hypothesis h (i.e., text representaton of a subgoal sequence) can be modeled as in Equation (1), where h_i is the ith token of h and the token probabilities are derived from the language model.

$$p(h|\tau) = \prod_i p_{\text{LM}}(h_i|h_{<i}, \tau, \{\tau_j, g_j\}_{j=1}^n) \quad (1)$$

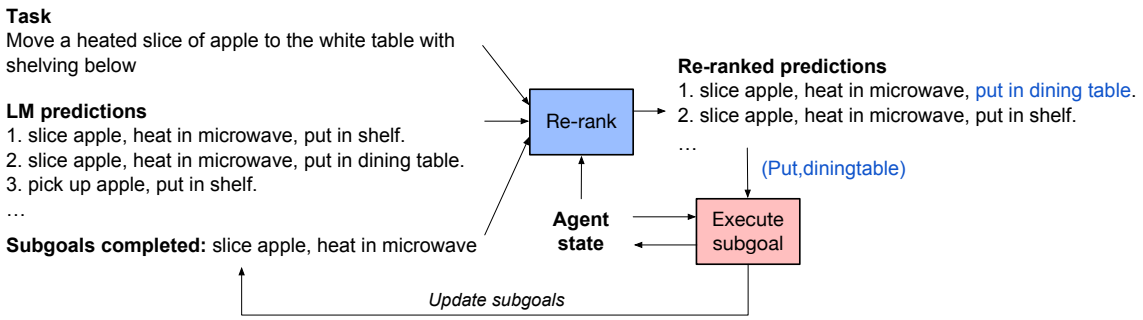

Figure 2: Re-ranking language model predictions with interaction and feedback from the environment. Given the task, language model predictions, completed subgoals and the agent state we come up with a ranked list of subgoal sequences. The agent then executes the next subgoal from the highest ranked plan. The completed subgoal is added to the partial plan and the process continues until stop subgoal is encountered. During training the agent receives a positive reward if the task is successfully completed, which we use as supervision to train the ranking model.

We use beam search to identify the top-k hypotheses according to $p(h|\tau)$. Generated hypotheses are constrained to be valid representations of subgoal sequences by considering only tokens which lead to a valid partial prefix of a subgoal sequence at each step of beam search.

Re-ranking predictions Recent studies have found that language models have popularity and recency biases: the tendency to repeat content mentioned in the prompt, especially content appearing later in the prompt (Zhao et al., 2021). They considered a simple approach to mitigate such biases in model predictions for classification tasks by comparing the likelihood of an output label with and without the query. In contrast to this 'direct model' which models the probability of a label given the input $p(y|x)$, Min et al. (2021a) showed that a 'channel model' which models $p(x|y)$ leads to better, more stable models.

Inspired by these observations, we propose to use $p(\tau|h)$ to score hypotheses in addition to $p(h|\tau)$. Mutual Information based ranking metrics are a natural candidate and they have been explored in the text generation literature (Li et al., 2015; Li and Jurafsky, 2016). We generate multiple hypotheses from the model using $p(h|\tau)$ and the generated hypotheses are re-scored using the weighted mutual information metric $(1-\lambda)\log p(h|\tau)+\lambda\log p(\tau|h)$ where λ is a hyperparameter[2]. To compute $p(\tau|h)$, we again use the language model prompted with "$g_1 = \tau_1, ..., g_n = \tau_n, h =$" as the query and compute the conditional probability of τ. We expect this paradigm of generating a set of strong hypotheses, followed by accurate re-ranking is more generally applicable to other few-shot language understanding problems.

[2]Appendix A details the connection to Mutual Information

3.2 Agent policy and incorporating environment feedback

We next consider building an agent that acts in a visual environment to complete tasks given text instructions. While Section 3.1 treated the language model as a knowledge extraction system, in the real world plans need to be updated based on interaction and feedback from the environment. We thus propose a method to improve language model predictions based on environment interaction. Since our goal is to learn the planning component of the agent, we assume a pre-trained low-level policy is provided and optimize over the space of plans. Jointly learning both components is beyond the scope of this work and left as future work.

We assume that a representation of the agent state s is available. The state representation captures information about the environment (e.g. objects present and their locations) estimated based on the agent's visual observations. As the agent explores the environment and collects new observations the state representation is updated. Assuming that a low-level policy π_L pre-trained to execute a given subgoal is provided, our goal is to train a high-level policy π_H which proposes the subgoals to be executed by the low-level policy. More formally, the high-level policy models $\pi_H(g^{(t)}|\tau, s_t, g^{(<t)})$ where τ is a text instruction, s_t is the state representation and $g^{(<t)}$ is the sequence of subgoals completed so far at high-level time-step t[3].

While the language model can generate compelling subgoal hypotheses, it doesn't take into account information about the environment. For instance, knowledge about the type of room the

[3]Alternatively, this can be framed as a POMDP in a hierarchical reinforcement learning setting.

agent is in (kitchen, bathroom, etc.) and the objects present in it are useful to infer the kind of tasks and subgoals that can be performed. We propose to re-rank hypotheses generated by the language model based on information from the environment to construct π_H. The plans generated by the language model are executed in the environment using π_L. The success/failure outcomes of these plan executions are used to construct a labeled dataset of instructions τ, plans g and agent state s. A supervised ranking model $f(g, \tau, s; \theta)$ is trained using this data to re-rank the language model predictions. We represent the ranking model as $f(g, \tau, s; \theta) = \theta^T \text{concat}(f^{\text{state}}(s), f^{\text{text}}(\tau, g))$ where $f^{\text{state}}(s)$ is a state embedding, $f^{\text{text}}(\tau, g)$ is a joint encoding of τ and g produced by a text encoder and θ is a parameter vector. Although a text embedding can be derived from the language model, we use a BERT encoder in favor of obtaining a smaller dimensional representation ($f^{\text{text}} = \text{BERT}_{\text{CLS}}$). The state and text embeddings are held fixed and only θ is trained. See Appendix C for more details.

During inference, an instruction τ is given, and we use the procedure in Section 3.1 to generate top-k hypotheses. At each step, hypotheses inconsistent with the sequence of subgoals executed so far are pruned and the remaining hypotheses are re-ranked based on the current agent state using f. The agent attempts the next subgoal proposed by the top hypothesis. The process ends when the stop subgoal is predicted. See Figure 2 for an illustration and Appendix D for more details about the training and inference algorithms.

4 Experiments

4.1 Data

We use data from the ALFRED benchmark proposed by Shridhar et al. (2020a) in our experiments. The ALFRED task requires an agent to execute instructions specified in text to accomplish basic tasks in an embodied environment. A given task is described using a high-level language directive as well as low-level step-by-step instructions (We only use the high-level description). The dataset consists of 7 task types (and 11 fine-grained types), and has more than 20k natural language task descriptions collected from human annotators. In addition, expert demonstrations computed by a planner are also made available. Tasks require acting over many time-steps, with an average of 50 actions, and the longest tasks require 100+ steps.

The ground truth subgoal sequences in the dataset consist of both navigation subgoals and object interaction subgoals. We discard the navigation subgoals and only retain the interaction subgoals for the following reasons. First, the interaction subgoals are sufficient for an agent to successfully complete the task. Second, predicting navigation subgoals from the text instruction alone may not always be possible as they often depend on the scene layout.

Subgoal representation A subgoal g^s is specified as $g^s = (b, o) \in \mathcal{B} \times \mathcal{O}$ where $|\mathcal{B}| = 7$ and $|\mathcal{O}| = 80$. We define a textual representation $v(b)$ of each action type (e.g. v(Pickup) = 'pick up', v(Heat) = 'heat in'). The object types o are identified by a text string $v(o)$ in the dataset and we directly use them as the text representation with minimal pre-processing (e.g. v(apple) = 'apple', v(desklamp) = 'desk lamp'). The subgoal is represented as $v(g^s)$ ='$v(b)$ $v(o)$' (e.g. v((Pickup, apple)) = 'pick up apple'). A subgoal sequence $g = (g^{(1)}, ..., g^{(n)})$ is represented as $v(g)$ ='$v(g^{(1)})$, ..., $v(g^{(n)})$.'. Text representations of all subgoals are given in Appendix B. Note that there are many plausible choices for the representation v and a different set of choices can lead to different results.

Metrics We use *top-k recall* to evaluate the ability of language models to generate plans from instructions by comparing against ground truth plans. In addition, we also evaluate the performance of an agent acting in the AI2-Thor (Kolve et al., 2017) simulator to complete tasks using *task success rate* (the percentage of tasks successfully completed).

4.2 Few-shot subgoal inference

We construct a training set of $N = 22$ instances by randomly choosing two instances per fine-grained task type. The language model is prompted with a concatenation of these training examples and the query instance. We perform constrained beam search decoding with a beam size of 10 to generate subgoal sequences. At each step of beam search, only tokens which lead to a valid partial prefix of a subgoal sequence are considered. All model generated hypotheses thus correspond to valid subgoal sequences. We evaluate models on the valid-seen split of the dataset which has 800 instances.

Table 2 shows subgoal inference results categorized by task type. We use publicly available pre-trained transformer language models GPT2-XL

Top-k recall	Ranking criteria	GPT2-XL			GPT-J		
		$N = 11$	$N = 22$	$N = 33$	$N = 11$	$N = 22$	$N = 33$
$k = 10$	$\log p(h\|\tau)$	47.93	58.54	59.51	64.02	71.10	72.07
$k = 1$	$\log p(h\|\tau)$	16.71	23.29	23.41	30.12	36.10	35.85
$k = 1$	$\log p(\tau\|h)$	19.88	22.07	25.61	34.15	34.39	35.00
$k = 1$	$\frac{1}{2}\log p(h\|\tau) + \frac{1}{2}\text{logp}(\tau\|h)$	29.88	32.20	33.17	44.02	47.80	49.63

Table 1: Re-ranking model generated hypotheses using different criteria. The first section shows top-10 recall of generated hypotheses (using $p(h|\tau)$). The second section shows top-1 recall after re-ranking these hypotheses using different criteria. Results are shown for GPT2-XL and GPT-J models when the number of training instances N is varied.

Task	GPT2-XL		GPT-J	
	top-1	top-10	top-1	top-10
look at obj in light	34.04	80.85	39.36	80.85
pick and place simple	25.35	59.15	45.07	71.13
pick two obj and place	28.23	68.55	26.61	70.16
pick heat then place	27.10	71.03	46.73	85.98
pick cool then place	30.95	65.87	39.68	79.37
pick clean then place	11.61	50.00	34.82	72.32
pick place movable	6.09	17.39	14.78	35.65
Overall	23.29	58.54	35.37	70.49

Table 2: Top-k recall for subgoal sequences predicted by GPT2-XL and GPT-J models categorized by task type.

(Radford et al., 2019) and GPT-J (Wang and Komatsuzaki, 2021) via the HuggingFace library (Wolf et al., 2020), which respectively have 1.5B and 6B parameters, in our experiments. The first six of the seven task types have two object arguments each. The pick place movable task type has three object arguments and hence a lower recall than the other task types. The top-10 recall of GPT2-XL and GPT-J are respectively 59% and 70%, which shows that large language models have strong ability to reason about plans from few training examples.

Re-ranking hypotheses The top-k recall performance reported in Table 2 are based on $\log p(h|\tau)$. We confirmed that the biases reported in the literature such as predicting content from the prompt are present in model predictions (Zhao et al., 2021). Consider the query example *Place a chilled martini glass with a fork on it on the table*. When the prompt contains training examples that mention 'sink', the model assigns the following $\log p(h|\tau)$ to these hypotheses.

- pick up fork, put in cup, put in sink. -2.4
- pick up fork, put in cup, put in table. -4.3

When all training instances in the prompt involving sink are removed, the log probabilities now become,

- pick up fork, put in cup, put in sink. -13.7
- pick up fork, put in cup, put in table. -9.1

The incorrect hypotheses involving sink is now ranked below the correct hypothesis involving table. While language models can retrieve strong hypotheses as indicated by the high top-10 recall, this observation shows that the ranking of these hypotheses, as determined by $p(h|\tau)$, may not be accurate. We thus consider mutual information based ranking approaches. Table 1 shows top-1 recall when model generated hypotheses are ranked according to different criteria. We also vary the number of training examples N by randomly choosing respectively 1, 2 and 3 instances per fine-grained task type.

We first observe that $p(\tau|h)$ ranks hypotheses better than $p(h|\tau)$ with very limited supervision ($N = 11$). However, it is often worse when more supervision is available. In contrast, combining the two log probabilities with $\lambda = \frac{1}{2}$ yields consistently better performance across models and number of training examples. This shows that generating a large number of hypotheses with a language model, followed by more accurate re-ranking using Mutual Information inspired metrics can be an effective paradigm for few-shot generation tasks with in-context learning.

Comparison with prior work We compare our prediction performance against prior work in Table 3. Jansen (2020) fine-tunes a GPT2-Medium model (325M parameters) to predict subgoals from instructions and report prediction results[4] when the model is trained on varying amounts of training data: 1%, 10%, 25%, 100% of the training set, which has 7793 instances. We ignore the navigation subgoals in this evaluation and only compare the sequence of object interactions. We report prediction performance of GPT-J using our approach on the same test set. The results show that large language models encode useful knowledge that can help plan from instructions effectively when supervision is limited. However, fine-tuning can be

[4]https://github.com/cognitiveailab/alfred-gpt2

Model	Top-1 recall	Training instances
Jansen (2020) (Fine-tuned GPT2-Medium)	5.07	77
	41.92	779
	53.80	1948
	61.00	7793
Ours (In-context GPTJ)	44.02	11
	47.80	22
	49.63	33

Table 3: Comparison against subgoal prediction performance of Jansen (2020).

Obj category	Confusion categories
wateringcan	pencil, kettle
glassbottle	vase
cart	shelf, sidetable, microwave, cart, fridge
butterknife	knife, butterknife
floorlamp	desklamp, floorlamp
vase	pencil, vase, bowl, winebottle, pot
ladle	spoon, ladle
pot	pot, pan
soapbottle	soapbottle, winebottle

Table 4: Object categories the model makes most errors on and the top object categories it confuses with.

effective when more supervision is available due to the fixed context length limitation of in-context learning. See Section 5 for ablations and more discussion about fine-tuning.

Prediction errors We examine prediction errors in identifying task type and object type. Key sources of model errors include annotation issues and ambiguity in object types. Table 4 shows the object types that have the least prediction accuracy, along with the object categories the model is confused about. Annotations can fail to correctly identify the target object - identifying a *butter knife* as a *knife* or a *pepper shaker* as *salt shaker*. Ambiguity can also arise from identifying an object with different names, depending on the context. For instance, depending on the scene layout, the argument for a *look at object in light* task can be a floor lamp or a desk lamp. Unless the type of lamp is identified precisely in the instruction, it is not possible to correctly predict the type of lamp. Correctly identifying these objects requires feedback from interaction with the environment.

The experiments so far evaluate the ability of a language model to retrieve ground truth subgoal sequences. Next we examine embodied agents that make use of these predictions and collect more supervision in order to improve subgoal predictions.

4.3 Agent policy and incorporating environment feedback

We now use subgoal predictions to construct an agent policy that acts in a simulated environment to complete tasks. The agent state representation and pre-trained low-level subgoal policy are borrowed from the HLSM model proposed in Blukis et al. (2021). HLSM models the agent state as a spatial persistent voxel representation of the room which models the location and category of objects present. The representation is constructed using modules that estimate segmentation and depth maps and other visual reasoning components and is updated as the agent gathers new observations. We use pre-trained models made available by the authors[5] to estimate agent state and their pre-trained low-level policy in our experiments.

We combine subgoal predictions with the pre-trained HLSM low-level policy and evaluate the overall agent policy on the ALFRED task in Table 5. Unlike the results reported in Section 4.2 which were based on the static dataset, these results are based on subgoals executed against the AI2-Thor simulator. In addition to task success rate, we also report the percentage of goal conditions satisfied, which rewards the model for partial task completions.

We compare against the following baselines on the ALFRED task. Seq2seq (Shridhar et al., 2020a) is a simple sequence-to-sequence baseline trained to map text instructions to low-level actions. MOCA (Singh et al., 2020) improves on Seq2seq using subgoal supervision and pre-trained visual reasoning components. Recent work such as HLSM (Blukis et al., 2021) and FiLM (Min et al., 2021b) build and use spatial semantic state representations and achieve stronger performance on the task. Note that, unlike these prior methods (MOCA, HLSM, FiLM) that rely on full subgoal supervision (20k instances), our approach is based on a small amount of subgoal supervision and additional supervision collected using active interaction with the environment. In addition, our approach does not require task-level expert trajectories and only assumes that a subgoal execution policy is provided.

Using the top language model prediction as is without using any information from the environment leads to 20% success rate. Next, we collect plan execution feedback for 1000 text instructions to train the ranking model described in Section 3.2.

[5] https://hlsm-alfred.github.io

Model		Success rate	
		Task	Goal-Cond
Seq2seq (Shridhar et al., 2020a)		3.7	10.0
MOCA (Singh et al., 2020)		19.2	28.5
FiLM (Min et al., 2021b)		24.6	37.2
HLSM (Blukis et al., 2021)		29.6	38.8
HLSM low-level policy	*Predicted subgoals*	19.8	31.4
	Re-ranked subgoals	23.9	35.0
	Oracle subgoals	37.2	48.2

Table 5: Task completion and goal condition success rates of models on the ALFRED validation seen split (results are based on task executions in the AI2-Thor simulator). The performance of our subgoal predictions combined with the HLSM low-level policy are shown at the bottom. We show the performance before and after re-ranking language model predictions based on agent state. Oracle subgoals shows the performance upper bound.

Re-ranking language model predictions using the trained ranking model improves the performance to 24%, which shows the importance of incorporating feedback from environment interaction. In comparison, the HLSM model with full subgoal supervision has success rate 30%. Although our predictions fall short of HLSM, they are competitive with the other baselines with subgoal supervision. The performance upper bound estimated using oracle subgoals is 37%, which shows the room for improvement over our predictions. These results show that accurate subgoal inference coupled with pre-trained low-level components leads to agents that perform well in embodied environments.

Figure 1 shows an example where the ranking model uses environment information to identify better plans. In this example, the instruction ambiguously specifies the receptacle as 'white table with shelving'. The language model's top two predictions for the target receptacle are `sidetable` and `shelf`, neither of which are present in the environment. The agent state captures this information and helps identify `diningtable` as the correct receptacle.

5 Ablations

We perform a series of ablations to identify the robustness of model predictions. We compare the performance of in-context learning to a sequence-to-sequence model fine-tuned to translate instructions to subgoal sequences. In addition, we observe the effect of varying the number of training examples and choice of training examples.

Number of training examples Figure 3 shows model recall for varying number of training examples. For zero training examples, the prompt consists of just the query instruction and the model decodes subgoals. 40 is the maximum number of examples for which the model prompt fits the sequence length restriction (1024 tokens) of GPT models. A steady increase in performance can be initially observed when increasing the number of training examples and the performance saturates towards the end. In-context learning further has the limitation of not being able to accommodate a larger number of training examples due to the length restriction. It would be interesting to explore how to make more effective use of larger number of training examples in future work.

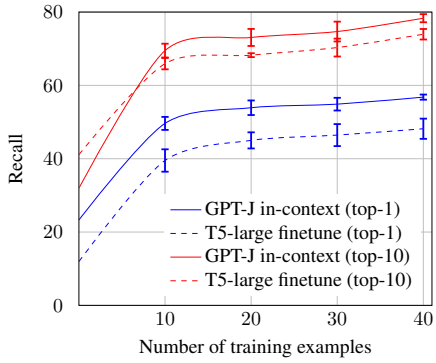

Figure 3: Comparison between GPT-J with in-context learning and a fine-tuned T5-large model for varying number of training examples. See text for details.

Choice of training examples We also estimate performance variance by varying the random seed for choosing examples randomly from the training set and compute standard deviation based on five random seeds for each setting. The plot shows that top-1 predictions from in-context learning have lower variance compared to fine-tuning.

Comparison with fine-tuning In order to understand how well the in-context learning approach compares to fine-tuned models, we fine-tune a T5-large model (Raffel et al., 2019) with 770M parameters on varying amounts of training data (this was the largest model we could fine-tune on our compute infrastructure). Note that this is not a head-to-head comparison between in-context learning and fine-tuning due to the difference in model size. Furthermore, there are other fine-tuning mechanisms such as prompt tuning and head tuning (Min et al., 2021a) which are not considered here. However, the result suggests that in-context learning with large pre-trained models can be favorable when computational constraints do not allow full fine-tuning of large models.

These ablations show that the in-context learn-

ing ability of large language models leads to predictions that are accurate, robust and stable in the presence of a small amount of training data.

6 Conclusion

This work explores the use of pre-trained language models for planning in real-world tasks. We showed that language models have strong capability to reason about subgoal sequences given a small number of training examples. We further demonstrated some simple mechanisms to incorporate feedback from interaction with the environment and show that this leads to more usable predictions. Finally, we show that combining subgoal predictions with a pre-trained low-level policy yields a strong baseline for embodied agent learning.

Our ablations demonstrate that in-context learning with a small amount of subgoal demonstrations has robust generalization properties. However, we also point out that in-context learning has the limitation of not being able to incorporate a large number of training examples due to the fixed context length restriction. It would further be beneficial to perform end-to-end learning with language model based subgoal prediction and a low-level policy, which would be interesting to explore in future work.

References

Valts Blukis, Chris Paxton, Dieter Fox, Animesh Garg, and Yoav Artzi. 2021. A persistent spatial semantic representation for high-level natural language instruction execution. *arXiv preprint arXiv:2107.05612*.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

SRK Branavan, David Silver, and Regina Barzilay. 2012. Learning to win by reading manuals in a monte-carlo framework. *Journal of Artificial Intelligence Research*, 43:661–704.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Haonan Chen, Hao Tan, Alan Kuntz, Mohit Bansal, and Ron Alterovitz. 2020. Enabling robots to understand incomplete natural language instructions using commonsense reasoning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1963–1969. IEEE.

Rodolfo Corona, Daniel Fried, Coline Devin, Dan Klein, and Trevor Darrell. 2020. Modular networks for compositional instruction following. *arXiv preprint arXiv:2010.12764*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Catherine Finegan-Dollak, Jonathan K Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-sql evaluation methodology. *arXiv preprint arXiv:1806.09029*.

Daniel Fried, Jacob Andreas, and Dan Klein. 2017. Unified pragmatic models for generating and following instructions. *arXiv preprint arXiv:1711.04987*.

Daniel Furrer, Marc van Zee, Nathan Scales, and Nathanael Schärli. 2020. Compositional generalization in semantic parsing: Pre-training vs. specialized architectures. *arXiv preprint arXiv:2007.08970*.

Matthew Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi Yuan. 2020. Interactive fiction games: A colossal adventure. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7903–7910.

Jonathan Herzig, Peter Shaw, Ming-Wei Chang, Kelvin Guu, Panupong Pasupat, and Yuan Zhang. 2021. Unlocking compositional generalization in pre-trained models using intermediate representations. *arXiv preprint arXiv:2104.07478*.

Peter A Jansen. 2020. Visually-grounded planning without vision: Language models infer detailed plans from high-level instructions. *arXiv preprint arXiv:2009.14259*.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. 2017. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*.

Brenden Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International conference on machine learning*, pages 2873–2882. PMLR.

Belinda Z Li, Maxwell Nye, and Jacob Andreas. 2021. Implicit representations of meaning in neural language models. *arXiv preprint arXiv:2106.00737*.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2015. A diversity-promoting objective function for neural conversation models. *arXiv preprint arXiv:1510.03055*.

Jiwei Li and Dan Jurafsky. 2016. Mutual information and diverse decoding improve neural machine translation. *arXiv preprint arXiv:1601.00372*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. 2019. A survey of reinforcement learning informed by natural language. *arXiv preprint arXiv:1906.03926*.

Hongyuan Mei, Mohit Bansal, and Matthew R Walter. 2016. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Vincent Micheli and François Fleuret. 2021. Language models are few-shot butlers. *arXiv preprint arXiv:2104.07972*.

Sewon Min, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2021a. Noisy channel language model prompting for few-shot text classification. *arXiv preprint arXiv:2108.04106*.

So Yeon Min, Devendra Singh Chaplot, Pradeep Ravikumar, Yonatan Bisk, and Ruslan Salakhutdinov. 2021b. Film: Following instructions in language with modular methods. *arXiv preprint arXiv:2110.07342*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Timo Schick and Hinrich Schütze. 2020. It's not just size that matters: Small language models are also few-shot learners. *arXiv preprint arXiv:2009.07118*.

Richard Shin, Christopher H Lin, Sam Thomson, Charles Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. 2021. Constrained language models yield few-shot semantic parsers. *arXiv preprint arXiv:2104.08768*.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020a. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020b. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

Kunal Pratap Singh, Suvaansh Bhambri, Byeonghwi Kim, Roozbeh Mottaghi, and Jonghyun Choi. 2020. Moca: A modular object-centric approach for interactive instruction following. *arXiv preprint arXiv:2012.03208*.

Alessandro Suglia, Qiaozi Gao, Jesse Thomason, Govind Thattai, and Gaurav Sukhatme. 2021. Embodied bert: A transformer model for embodied, language-guided visual task completion. *arXiv preprint arXiv:2108.04927*.

Alane Suhr, Claudia Yan, Jacob Schluger, Stanley Yu, Hadi Khader, Marwa Mouallem, Iris Zhang, and Yoav Artzi. 2019. Executing instructions in situated collaborative interactions. *arXiv preprint arXiv:1910.03655*.

Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Shunyu Yao, Rohan Rao, Matthew Hausknecht, and Karthik Narasimhan. 2020. Keep calm and explore: Language models for action generation in text-based games. *arXiv preprint arXiv:2010.02903*.

Yichi Zhang and Joyce Chai. 2021. Hierarchical task learning from language instructions with unified transformers and self-monitoring. *arXiv preprint arXiv:2106.03427*.

Tony Z Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate before use: Improving few-shot performance of language models. *arXiv preprint arXiv:2102.09690*.

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

A Mutual Information based scoring

Mutual Information between random variables X, Y is defined as in Equation (2). We consider a weighted Mutual Information metric as defined as in Equation (3) similar to Li and Jurafsky (2016) and introduce the hyperparameter λ. Identifying Y that maximizes the weighted Mutual Information is equivalent to maximizing the expression in Equation (4). We use this metric to rank hypotheses generated by the language model.

$$\mathrm{MI}(X, Y) = \log \frac{p(x, y)}{p(x)p(y)} \tag{2}$$

$$\mathrm{wMI}(X, Y) = \log \frac{p(x, y)}{p(x)p(y)^{\lambda}} \tag{3}$$

$$\begin{aligned}
&\operatorname*{argmax}_{y} \mathrm{wMI}(X, Y) \\
&= \operatorname*{argmax}_{y} \log \frac{p(x, y)}{p(x)p(y)^{\lambda}} \\
&= \operatorname*{argmax}_{y} \log \left(\frac{p(x, y)}{p(x)}\right)^{1-\lambda} \left(\frac{p(x, y)}{p(y)}\right)^{\lambda} \frac{1}{p(x)^{\lambda}} \\
&= \operatorname*{argmax}_{y} (1-\lambda) \log p(y|x) + \lambda \log p(x|y) \\
&\qquad - \lambda \log p(x) \\
&= \operatorname*{argmax}_{y} (1-\lambda) \log p(y|x) + \lambda \log p(x|y)
\end{aligned} \tag{4}$$

B Subgoal representation

Table 6 shows the subgoal representation we use in this work.

Subgoal	Representation
(Pickup, X)	pick up X
(Put, X)	put in X
(Heat, X)	heat in X
(Cool, X)	cool in X
(Clean, X)	clean in X
(Slice, X)	slice X
(ToggleOn, X)	turn on X

Table 6: Subgoals and corresponding text representation. X represents an object argument.

C Ranking model: Architecture

State embedding HLSM represents the agent state as a semantic voxel representation $s \in [0, 1]^{X \times Y \times Z \times C}$ where the value $s(x, y, z, c)$ represents if there is an object of type c at position (x, y, z) of the room layout. The agent state is encoded into a vector $f^{\text{state}}(s)$ using a linear mapping. We use this encoding as the state embedding.

Instruction and subgoal sequence encoding The instruction τ and a candidate subgoal sequence g are jointly processed using a BERT encoder and the CLS representation is used as a representation vector $\text{BERT}_{\text{CLS}}(\tau, g)$.

The ranking model is represented as $f(g, \tau, s; \theta) = \theta^T \text{concat}(f^{\text{state}}(s), \text{BERT}_{\text{CLS}}(\tau, g))$ where θ is a parameter vector. When training the ranking model the state embedding and BERT encoders are held fixed and only the linear transformation θ is trained.

D Ranking Model: Training and Inference

Formally, the learning problem is a MDP $(\mathcal{S}, \mathcal{G}, \mathcal{L}, \mathcal{R}, \mathcal{T})$, where $s_t \in \mathcal{S}$ is the agent state, $g^{(t)} \in \mathcal{G}$ is a subgoal, $\tau \in \mathcal{L}$ is a text instruction, $\mathcal{R}(\tau, s_t)$ is a reward function that provides success/failure feedback for completing a given instruction, $\mathcal{T} : (s_t, g^{(t)}) \rightarrow s_{t+1}$ is a state transition function where s_{t+1} is computed by a low-level policy π_L pre-trained to execute a given subgoal g. Our goal is to train a high-level policy $\pi_H(g^{(t)}|\tau, s_t, g^{(<t)})$ where s_t is the agent state and $g^{(<t)}$ is the sequence of subgoals completed so far at high-level time-step t.

Algorithm 1 describes how we collect training data to train the ranking model. Algorithm 2 shows how the ranking model is used during inference.

Algorithm 1 Training

Given: epochs $= 100$, $\mathcal{D}^{\text{ins}}$ (set of instructions)

Collect training data

$\mathcal{D} \leftarrow \{\}$ (Initialize training set)
for τ in $\mathcal{D}^{\text{ins}}$ **do**
  Generate plans and re-rank using mutual information metric $g_1, \ldots, g_k \sim p_{\text{LM}}(\cdot|\tau)$
  for $i = 1 \ldots k$ **do**
    Initialize agent state s
    $S \leftarrow \{s\}$ (Record agent states)
    for $j = 1 \ldots |g_i|$ **do**
      $s \leftarrow \mathcal{T}(s, g_i^{(j)})$ (Execute $g_i^{(j)}$ using π_L)
      $S \leftarrow S \cup \{s\}$
    end for
    if $\mathcal{R}(\tau, s) > 0$ **then** (Task succeeded)
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(g_i, \tau, s) | s \in S\}$
      break
    end if
  end for
end for

Train model

for $i = 1 \ldots$ epochs **do**
  loss $\leftarrow 0$
  for $(g, \tau, s) \in \mathcal{D}$ **do**
    Generate plans $g_1, .., g_k \sim p_{\text{LM}}(\cdot|\tau)$
    loss $\leftarrow$ loss $- \log \frac{\exp f(g,\tau,s;\theta)}{\sum_{i=1}^{k} \exp f(g_i,\tau,s;\theta)}$
  end for
  $\theta \leftarrow$ Optimizer Update$(\theta, \nabla_\theta f)$
end for
return f

Algorithm 2 Inference

Given: instruction τ, $i^{\text{thresh}} = 10$
Generate plans $g_1, \ldots, g_k \sim p_{\text{LM}}(\cdot|\tau)$
$G \leftarrow \{g_1, \ldots, g_k\}$
Initialize agent state s
$i \leftarrow 0$ (subgoal index)
$g \leftarrow \text{argmax}_{g \in G} f(g, \tau, s; \theta)$
while $g^{(i)} \neq$ <stop> and $i < i^{\text{thresh}}$ **do**
  $s \leftarrow \mathcal{T}(s, g^{(i)})$ (Execute $g^{(i)}$ using π_L)
  $G \leftarrow \{h | h \in G$ and $h^{(i)} = g^{(i)}\}$ (Retain plans consistent with subgoals completed so far)
  $g \leftarrow \text{argmax}_{g \in G} f(g, \tau, s; \theta)$
  $i \leftarrow i + 1$
end while
return g, s

E Analysis of model errors

Figure 4 shows the confusion matrix for object type prediction. Predictions are from top-1 subgoal sequences predicted by GPT-J.

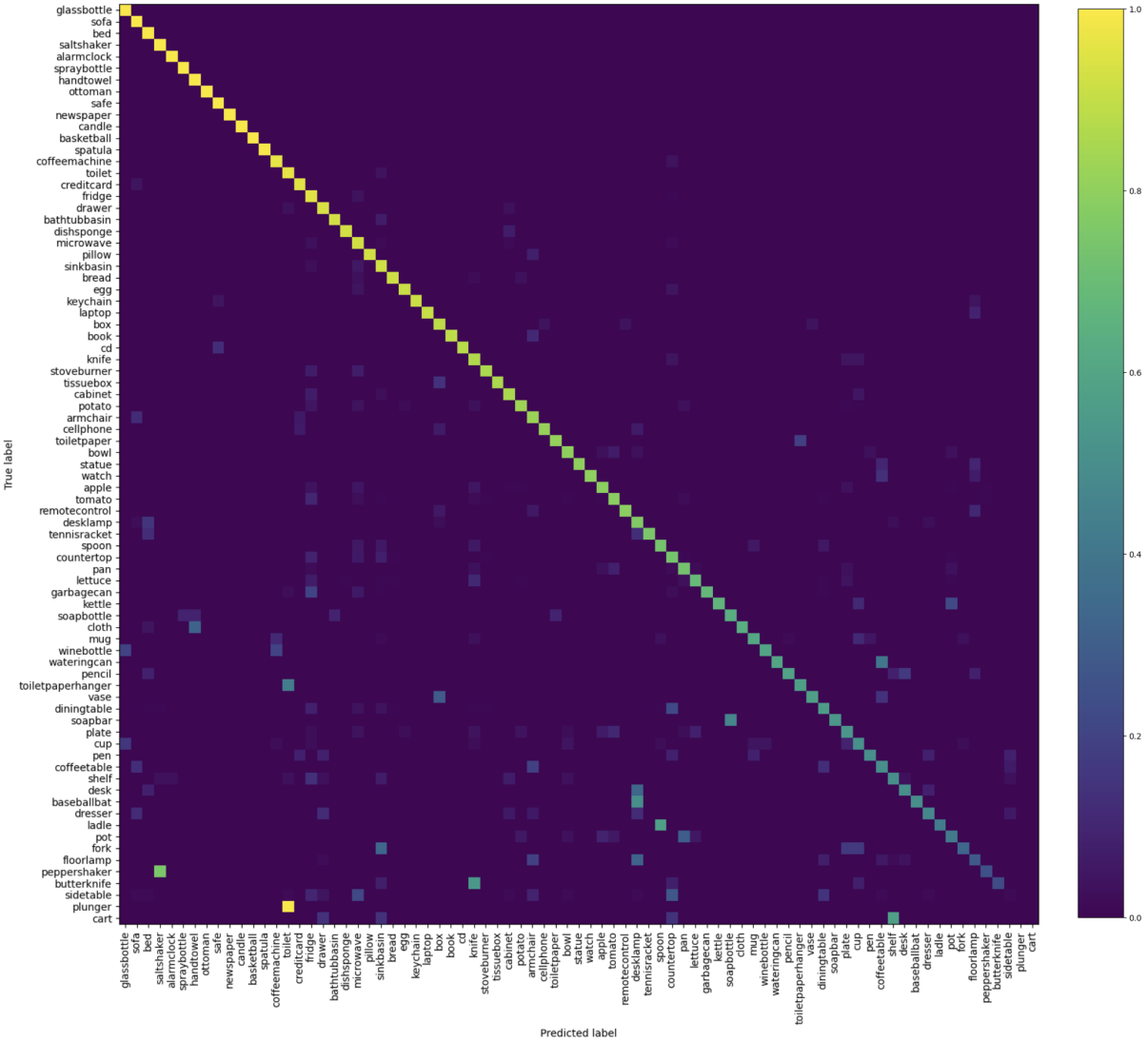

Figure 4: Object type confusion matrix.