



OPEN ACCESS

EDITED BY

Dimitrios Kanoulas,
University College London,
United Kingdom

REVIEWED BY

Maria Koskinopoulou,
Heriot-Watt University, United Kingdom
Dario Albani,
Technology Innovation Institute (TII),
United Arab Emirates

*CORRESPONDENCE

Georgia Chalvatzaki,
✉ georgia.chalvatzaki@tu-darmstadt.de

†PRESENT ADDRESS

Leonardo F. R. Ribeiro,
TU Darmstadt, Darmstadt, Germany

RECEIVED 12 May 2023

ACCEPTED 03 July 2023

PUBLISHED 15 August 2023

CITATION

Chalvatzaki G, Younes A, Nandha D,
Le AT, Ribeiro LFR and Gurevych I (2023),
Learning to reason over scene graphs: a
case study of finetuning GPT-2 into a
robot language model for grounded task
planning.

Front. Robot. AI 10:1221739.

doi: 10.3389/frobt.2023.1221739

COPYRIGHT

© 2023 Chalvatzaki, Younes, Nandha, Le,
Ribeiro and Gurevych. This is an
open-access article distributed under
the terms of the [Creative Commons
Attribution License \(CC BY\)](https://creativecommons.org/licenses/by/4.0/). The use,
distribution or reproduction in other
forums is permitted, provided the
original author(s) and the copyright
owner(s) are credited and that the
original publication in this journal is
cited, in accordance with accepted
academic practice. No use, distribution
or reproduction is permitted which does
not comply with these terms.

Learning to reason over scene graphs: a case study of finetuning GPT-2 into a robot language model for grounded task planning

Georgia Chalvatzaki^{1,2,3*}, Ali Younes¹, Daljeet Nandha¹,
An Thai Le¹, Leonardo F. R. Ribeiro^{4†} and Iryna Gurevych^{1,2}

¹Computer Science Department, Technische Universität Darmstadt, Darmstadt, Germany, ²Hessian.AI, Darmstadt, Germany, ³Center for Mind, Brain and Behavior, University Marburg and JLU Giessen, Marburg, Germany, ⁴Amazon Alexa, Seattle, WA, United States

Long-horizon task planning is essential for the development of intelligent assistive and service robots. In this work, we investigate the applicability of a smaller class of large language models (LLMs), specifically GPT-2, in robotic task planning by learning to decompose tasks into subgoal specifications for a planner to execute sequentially. Our method grounds the input of the LLM on the domain that is represented as a scene graph, enabling it to translate human requests into executable robot plans, thereby learning to reason over long-horizon tasks, as encountered in the ALFRED benchmark. We compare our approach with classical planning and baseline methods to examine the applicability and generalizability of LLM-based planners. Our findings suggest that the knowledge stored in an LLM can be effectively grounded to perform long-horizon task planning, demonstrating the promising potential for the future application of neuro-symbolic planning methods in robotics.

KEYWORDS

robot learning, task planning, grounding, language models (LMs), pretrained models, scene graphs

1 Introduction

The autonomous execution of long-horizon tasks is of utmost importance for future assistive and service robots. An intelligent robot should reason about its surroundings, e.g., regarding the included objects and their spatial-semantic relations, and abstract an action plan for achieving a goal that will purposefully alter the perceived environment. Such an elaborate course of robot actions requires scene understanding, semantic reasoning, and planning over symbols and geometries. The advent of Deep Learning led many researchers to faithfully follow end-to-end approaches due to the representation power of differentiable deep neural networks (LeCun et al., 2015).

The problem of sequential decision-making has been addressed both with search-based and optimization approaches (Kaelbling and Lozano-Pérez, 2011; Toussaint, 2015; Driess and Toussaint, 2019; Garrett et al., 2021; 2020), as well as learning-based (Nair and Finn, 2019; Funk et al., 2021; Hoang et al., 2021) and hybrid methods (Kim et al., 2019; Driess et al., 2020; Ren et al., 2021; Funk et al., 2022). While the first ones enjoy probabilistic

completeness, they require full domain specification and have high computational demands. The learning-based methods require broad exploration to learn from experience, but they have shown better generalization capabilities in similar domains to those experienced during training.

Large Language Models (LLMs) have exhibited an unprecedented generative ability (Bommasani et al., 2021), thanks to the transformer architecture (Vaswani et al., 2017) combined with massive datasets distilled from the internet. Naturally, in the quest for general artificial intelligence, researchers try to benchmark such models in reasoning tasks, among others (Wang et al., 2018; 2019). Robotic embodied intelligence requires both logical and geometric reasoning; hence, it is a holy grail of AI. Several researchers saw a benefit in LLMs, and it was not long before several works explored their application to robotics for endowing robots with reasoning abilities in the scope of autonomous task planning and interaction (Brohan et al., 2022; Wei et al., 2022b). However, most works have focused on the prompting (Brown et al., 2020) and the subsequent prompt engineering (White et al., 2023), in which engineers provide appropriate inputs to LLMs for extracting outputs that can be realizable by a robotic agent, either for human-instruction following (Ouyang et al., 2022) or for planning (Singh et al., 2022; Zeng et al., 2022).

In this work, we study a finetuning process for grounding a small LLM for robotics, i.e., GPT-2 (Radford et al., 2021), to be used as a high-level abstraction in a task planning pipeline. Particularly, we propose a method that decomposes a long-horizon task into subgoals in the form of goal specifications for a robotic task planner to execute, and we investigate whether such a method can reach the performance levels of an oracle task planning baseline.

Our contribution is twofold: (i) we propose a novel method for linearizing the relations in a scene-graph structure representing the domain (world) to provide it as grounding context when finetuning a pretrained language model (e.g., GPT-2) for learning to draw associations between possible actions (goto, pick, etc.) and objects in the scene (e.g., kitchen, apple, etc.). Importantly, in our context, we encode the relative position of objects (far, close, right, left, etc.), allowing our model to account for the scene's geometrical structure when learning to plan. The proper structure of the input context is necessary for enabling the model to reason about the combinatorics of actions with affordable objects and their logical sequence (e.g., to cook something, one must first go to the kitchen). (ii) We showed that larger pretrained models do not necessarily possess grounded reasoning abilities, while it is possible to finetune smaller models on various tasks to use them as parts of a broader neuro-symbolic planning architecture. Contrarily to works that directly apply actions suggested by the GPTs to robots, we use language models at a higher level of abstraction, effectively suggesting sub-goals as PDDL problems to be solved by a Fast Downward task planner Helmert (2006), effectively decomposing the whole problem into smaller ones of lower complexity.

Our thorough experimental evaluation shows that finetuning GPT-2 by additionally grounding its input on the domain can help translate human requests (tasks) to executable robot plans, to learn to reason over long-horizon tasks, as those encountered in the ALFRED benchmark (Shridhar et al., 2020). We compare our proposed approach with classical planning methods to investigate the applicability and generalizability of the Pre-trained Language

Model (PLM)-based planners compared to classical task planners operating on a limited computational budget for a fair comparison. We conclude that the knowledge stored in a PLM can be grounded on different domains to perform long-horizon task planning, showing encouraging results for the future application of neuro-symbolic planning methods in robotics.

2 State of the art

2.1 Reasoning with large language models

LLMs have attracted much attention for understanding the commonsense and reasoning patterns in their latent space (Zhou et al., 2020; Li et al., 2021; Bian et al., 2023). It has been shown that some abilities in logical and mathematical reasoning seem to emerge when LLMs are prompted appropriately (Wei et al., 2022b; a). However, the engineering effort, as well as the lack of robustness, is a key issue in prompting massive models (Ruis et al., 2022; Valmeekam et al., 2022). While great effort seems to be consumed on few-shot prompting of huge parametric models, it has also been shown by other lines of work show that efficient finetuning of much smaller models (Tay et al., 2022), or the use of small adaptation modules (Adapters) (Houlsby et al., 2019; Pfeiffer et al., 2021) can lead to methods that perform more robustly than large-scale generalist few-shot prompts. In the same direction, the chatbot versions of those huge models raised several points of criticism recently, showing that much more is needed than just prompting a blind human-preference alignment¹.

2.2 Robot behavior planning

Long-horizon robot behavior planning is an NP-hard problem (Wells et al., 2019). Current advances in ML and perception led researchers to revisit this fundamental problem, i.e., the execution of multi-stage tasks, whose completion requires many sequential goals to be achieved, considering learning-based heuristics (Driess et al., 2020). Researchers consider such problems as Task And Motion Planning (TAMP) problems (Garrett et al., 2021; Ren et al., 2021; Xu et al., 2022), with a symbolic plan over entities and predicate with respective action operators with preconditions and effects in the environment. In contrast, a motion plan tries to find a feasible path to the goal. Nevertheless, most TAMP methods rely on manually specified rules; they do not integrate perception, and the combinatorial explosion when searching over symbolic and continuous parameters prohibits scaling the methods to challenging, realistic problems (Kim et al., 2019; Garrett et al., 2020).

Transformer models (Vaswani et al., 2017) that revolutionized the field of Natural Language Processing (NLP) opened the way for multiple new applications, in particular for robotics, e.g., visual-language instruction following (Pashevich et al., 2021), 3D scene understanding and grounding (Chen W. et al., 2022; Mees et al., 2022), language-based navigation (Huang C. et al., 2022; Shah et al., 2023). Due to their training on extensive databases, several

1 7 problems facing Bing, Bard, and the future of AI search, from The Verge.

works explored the use of LLMs for task planning and long-horizon manipulation (Huang et al., 2022b), mainly employing clever prompting (Raman et al., 2022; Singh et al., 2022), using multimodal information (Jiang et al., 2022; Zeng et al., 2022), grounding with value-functions (Chen B. et al., 2022; Brohan et al., 2022; Huang et al., 2022c), and deploying advances in code generation to extract executable robot plans (Liang et al., 2022). (Li et al., 2022) propose to use a PLM as a scaffold for decision-making policies in interactive environments, demonstrating benefits in the generalization abilities for policy learning even when language is not provided as input or output. Recently, PALM-e (Driess et al., 2020) has integrated a vision transformer with the PALM language model and has encoded some robotic state data to propose a multimodal embodied model, which showed the potential of integrating geometric information of the robot state but achieved limited performance in robotic tasks.

3 Language models for grounded robot task planning

3.1 Problem statement

Let us assume an agent that is able to move and manipulate objects in an environment, e.g., a mobile manipulator robot in a household environment. Let the environment be composed of a combination of *rooms*, such as ‘bathroom,’ ‘living room,’ or ‘kitchen.’ Each room contains *objects* and *receptacles*, i.e., objects that are able to receive other objects, such as ‘table,’ ‘drawer,’ or ‘sink.’ Each object has (household-specific) properties (affordances) associated with it that define whether it can be picked up, cleaned, heated, cooled, cut, etc. These properties can change, meaning that the objects have a *state*. The agent can pick up only one object at a time, meaning the agent also has a state, e.g., ‘object in hand.’ Given the fact that the state is preserved over time and future actions depend on past actions, the environment can be characterized as sequential. Therefore, a series of actions has to be reasoned upon for an agent to be able to execute a series of actions for solving a long-horizon task, i.e., a task that requires the completion of several subtasks and potentially the manipulation of various objects to achieve the end-goal.

3.2 The ALFRED benchmark

The ALFRED benchmark (Shridhar et al., 2020) contains human-annotated training samples and image-based recordings of everyday household tasks; this is “25,743 English language directives describing 8,055 expert demonstrations averaging 50 steps each, resulting in 428,322 image-action pair”. In addition to that, the dataset provides a *PDDL domain* of the overall task and a *PDDL problem* for each sample (Aeronautiques et al., 1998). ALFRED heavily depends on AI2-THOR (Kolve et al., 2017), which acts as the underlying controller and simulation environment (based on the Unity game engine): trajectories for each sample of the ALFRED dataset were generated with AI2-THOR, and the validation of user-generated actions requires the AI2-THOR controller. Figure 1 shows

a sample scene loaded into the AI2-THOR simulator. Each sample in the dataset consists of a high-level plan in PDDL and the trajectory of the agent’s actions which lead to successful task completion, together with a description of the task goal and each plan step in Natural Language (NL).

3.3 State and action space

The state space defines the feedback provided by the environment, while the action space defines the available actions to interact with the environment.

State space. The environments we consider in this work are fully observable, having access to the complete simulator state representing the domain, as commonly considered for task planning tasks. AI2-THOR, the underlying simulator, eliminates most physics-related aspects (e.g., objects are automatically picked up and placed by a single action), which makes the highly dynamic and stochastic household environment almost static and deterministic—almost because some physics still exists. This simplifies the core TAMP problem along with the discrete agent actions defined in the ALFRED dataset. Therefore, the ALFRED benchmark represents an appropriate choice for studying the problem of learning for robotic task planning, where motion failures are minimized by the underlying AI2-THOR controllers. Hence, we can focus on the reasoning aspects of the problem, which is the focus of this study. In the following, the state of the domain is transformed to NL context (§3.6) and not used directly as model input.

Action space. ALFRED has an action space of eight discrete high-level actions: *GotoLocation*, *PickupObject*, *PutObject*, *CoolObject*, *HeatObject*, *CleanObject*, *SliceObject* and *ToggleObject*. The underlying AI2-THOR navigation controller also has a discrete action space; the agent can **move** *forward*, *backward*, *left* or *right* and **rotate** *clockwise* or *counter-clockwise* in fixed steps.

3.4 Task categories

The ALFRED dataset encompasses seven categories of household tasks: “Look at object,” “Pick and place,” “Pick two and place,” “Pick and place with movable receptacle,” “Pick, clean then place,” “Pick, cool then place,” “Pick, heat then place.” Because objects can be placed in different corners of a room, each of these tasks includes the sub-problem of navigation. For the ‘pick’ or ‘place’ subtasks executing the respective *PickupObject* or *PutObject* action is sufficient. But, the subtasks “clean,” “cool” and “heat” must be seen as planning problems on their own, because the corresponding actions are a composition of high-level *state-dependent* actions. Regarding the household environment, the subtask “cool” requires a fridge, “heat” requires a microwave (or oven), and “clean” requires a sink as an *receptacle*. The ALFRED simulator tracks the state of each object, and the subtask is only considered successful when the final object state is correct. For example, if the task category is ‘Pick, clean then place,’ the task goal is only completed when the placed object is marked as ‘clean.’ The implementation aspects of these task categories are discussed in Section 4.

Task: Put some Peppershaker on Sidetable

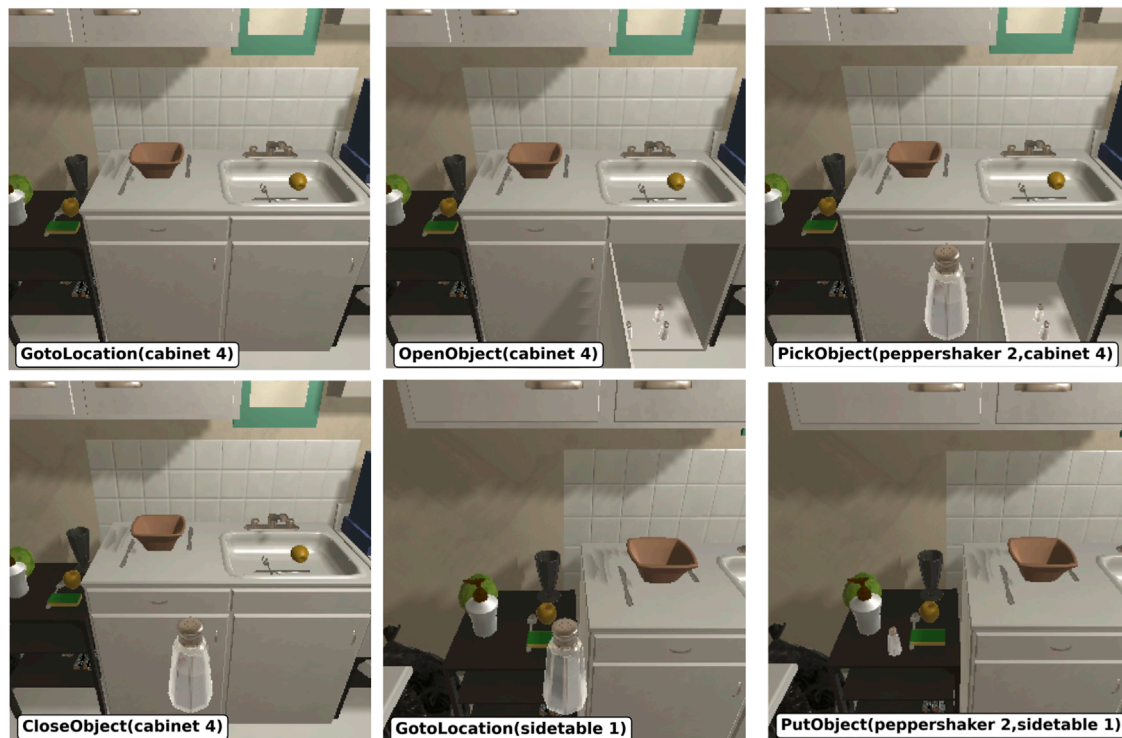


FIGURE 1

AI2-THOR simulator rendering a sample rollout from the ALFRED (Shridhar et al., 2020). The scenes show a room with household objects and the robot executing a task. Note that the robot does not have an arm, and the object automatically floats in front of the camera; it interacts with the environment through discrete actions. The discrete actions are shown underneath each frame in the form of PDDL commands.

3.5 RobLM: robot language model for task plan generation

Just like images can be represented by discretizing color space, NL can be expressed as a sequence of tokens $\mathbf{x} = [x_1, x_2, \dots, x_n]$, where each token is mapped to an embedding (lookup table). The Language Model (LM) places a probability distribution $p(\mathbf{x})$ over the output token sequence. $p(\mathbf{x})$ can be decomposed into a conditional probability distribution $p(x_{i+1}|x_i)$, where the probability of each token depends on all previous tokens. This results in the following joint distribution $p(\mathbf{x})$ for a sequence of tokens \mathbf{x} :

$$p(\mathbf{x}) = \prod_i p(x_i|x_0, x_1, \dots, x_{i-1}) \quad (1)$$

In regards to Neural Network (NN), $p(\mathbf{x})$ is commonly estimated with the Softmax function (Bengio et al., 2000)²

$$p(\mathbf{x}) = \text{Softmax}(\mathbf{W}\mathbf{h}^T + \mathbf{b}) = \frac{\exp(\mathbf{W}\mathbf{h}^T + \mathbf{b})}{\sum \exp(\mathbf{W}\mathbf{h}^T + \mathbf{b})}, \quad (2)$$

where \mathbf{W} is the learned weight matrix, \mathbf{b} the bias and \mathbf{h}^T the output vector of the NN. For text generation, the joint probability

distribution $p(\mathbf{x})$ (see Eq. 1) can be formulated as a maximum-likelihood objective, where the objective is to maximize the likelihood of the next token occurrence for the given data.

Our goal is to finetune a LM to get a Robot Language Model (RobLM) that can generate a complete high-level task plan in one shot, given the domain information and a task goal. Because LMs are unsupervised learners, a single training sample contains both given and desired information as NL text. A restriction to the text format (a string of characters) comes with challenges: structural information needs to be condensed into a single linear dimension, and conceptually different aspects of the input need to be annotated in the text. This text format, including the syntax, has to be designed in such a way that information can be fed to and extracted from the LM reliably.

In RobLM, the format definition for a NL task description must comply with the following syntactic rule (spaces added for readability):

```
Goal [<SEP> Context] <BOS> Plan <EOS>
[...] := optional
<SEP> := separator token
<BOS> := begin-of-sequence token
<EOS> := end-of-sequence token
```

Goal is the task goal in NL. **Context** is any additional, yet optional information provided to the LM. The task might have

² The Softmax function applies the exponential function to each element of the input vector and normalizes the values, by dividing by the sum of the exponentials.

ambiguous solutions, and the inherent assumption is that the LM will better “understand” the task if given a context. Examples of a context are the name of the *room*, the name of the target object, or a NL description of the environment (see 3.6).

Plan is the sequence of high-level task actions and their respective arguments, such as objective name or location. Because PLM have been trained on a diverse corpus of NL, including program code, the format for plans follows syntactical rules similar to that of a generic programming language:

```
Action0(arg0[, arg1]); Action1(arg0[, arg1]);
...
```

The sequence between the special tokens <BOS> and <EOS> can be extracted to retrieve the plan from the LM-generated output.

3.5.1 Data augmentation

Each sample in the ALFRED dataset can be replayed in the AI2-THOR simulator to collect additional information not contained in the original dataset. ALFRED provides a script that has been modified for that purpose. Data augmentation is necessary for Graph2NL (c.f. §3.6) to generate a graph representation from the environment state. For each replayed sample, the complete list of objects in the scene, with their respective name, position, and rotation, and the agent position is saved to a separate file next to the trajectory data. This file is later loaded and turned into a processable graph.

3.6 Mapping scene graphs to natural language: Graph2NL

PLMs are trained on NL. Because of this, NL is a natural modality for finetuning a PLM. When a context is provided to the LM, this context must be presented in NL just like the input sequence. If the context should encapsulate the environment state, this means that the state has to be transformed into NL before being supplied to the PLM.

Graph2NL is a novel method that “translates” the object-centric scene graph representation of the environment state to NL. Optionally, domain knowledge about the environment³ can be infused into this graph. The following steps describe the core Graph2NL process.

- 1) Generate an object-scene graph G with a node for the agent and nodes corresponding to objects, node attributes being the position and rotation of the object in Euclidean space and their respective distance and orientation vectors as edge attributes.
- 2) (Optional) Infuse domain knowledge about the environment by connecting all dependent nodes and all nodes reachable by the agent.
- 3) Connect the agent (node) to all reachable nodes, if given domain knowledge, or to all nodes, if not given domain knowledge.

³ Domain knowledge entails every possible room, object, and receptacle name and their allowed relations, as described in the respective documentation: <https://ai2thor.allenai.org/ithor/documentation/objects/object-types>.

TABLE 1 Graph2NL mapping table. Distances are mapped to NL vocabulary (or a symbol) in a one-to-one relation. *Yaw* describes the orientation along the surface normal when viewed from a top-down perspective, and *Pitch* describes the z-planar offset (altitude) in relation to the origin.

Distance [m]		
Value	NL	Symbol
>5	distant	a
>4	far	b
>3	reachable	c
>2	near	d
>1	close	e
>0.5	closer	f
>0.1	next	g
<0.1	in	h
Yaw [°]		
Value	NL	Symbol
45 to 135	right	i
135 to 225	back	j
225 to 315	left	k
315 to 45	front	l
Pitch [°]		
Value	NL	Symbol
≥0	above	m
<0	below	n

- 4) Given a task and the identified target object, find all paths in the graphs leading from the agent (node) to the target object (node).
- 5) Use edge attributes in the found paths to describe the task-centric environment state, by mapping geometric relations to NL tokens.

3.6.1 NL mapping

To translate geometric relations attributed by the graph edges into a NL description, a mapping function is designed. In human speech, distances are expressed by a vocabulary of words such as “close” or “far” and orientations are expressed by words such as “in front” or “behind”. Graph2NL adapts this vocabulary to describe the (numeric) distance and orientation from one node relative to another in NL.

Table 1 summarizes the mapping used in Graph2NL. The distance between nodes is expressed in Cartesian space and orientation in polar coordinates, where *Yaw* is the azimuth angle (rotation along the surface normal) and *Pitch* is the zenith angle (altitude). With this mapping, the geometric relation between two nodes can be explained by three words (one for each: distance, pitch, and yaw). The vocabulary contains 8 words to express the distance, 4 words to express the vertical, and 2 words to express the horizontal orientation. Combinatorially, this gives 64 possible geometric configurations. The geometric relationship is expressed in a condensed form by treating each of these configurations as a relation and assigning a special

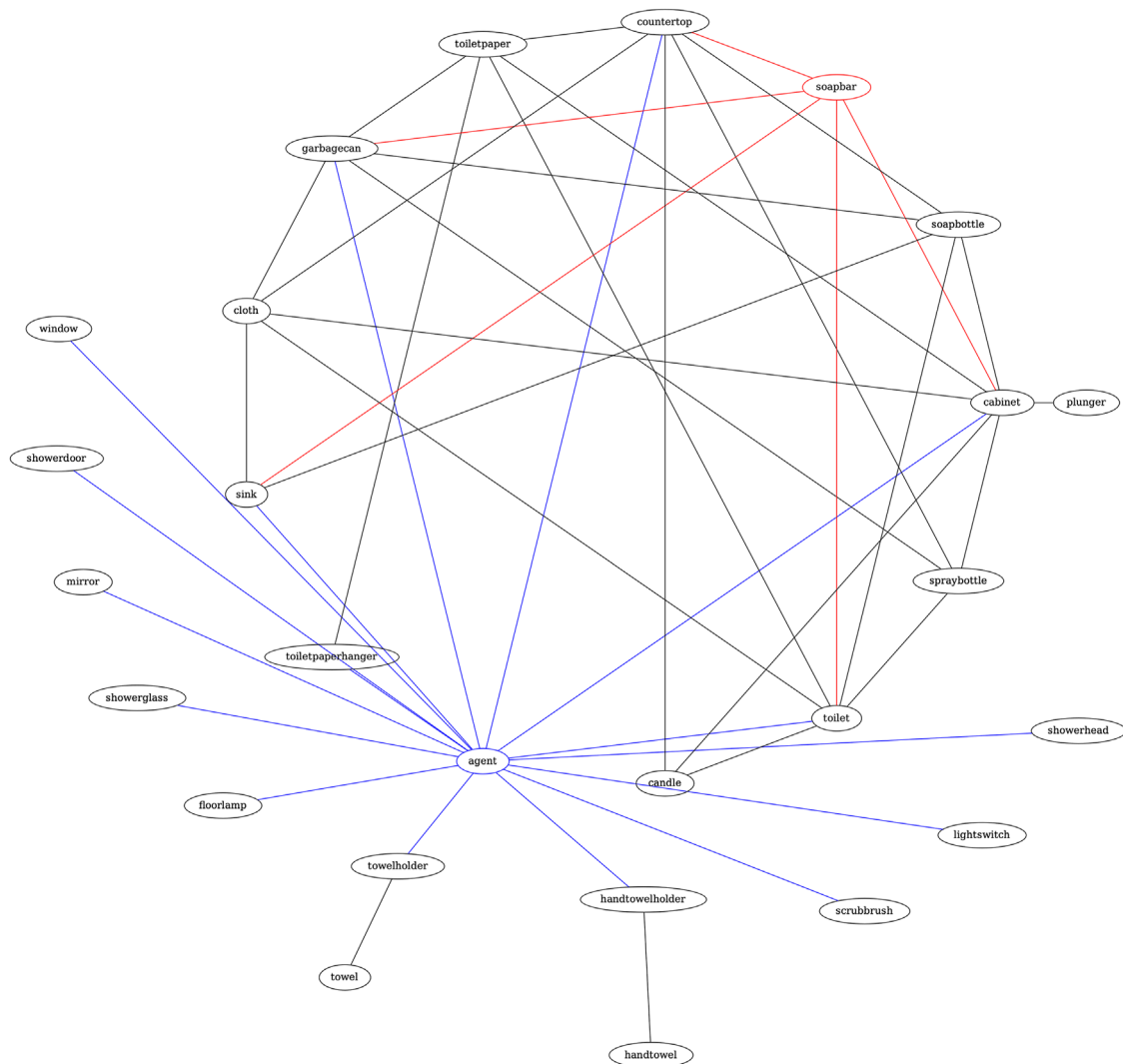


FIGURE 2
Graph2NL example graph. After locating the root (“agent”) and target node (“soapbar”), the shortest paths connecting those nodes are found and summarized in NL by mapping all edge attributes along the path.

symbol (token) for each relation. A simple approach, referring to the *Symbol* column in Table 1, is by assigning a symbol to each word. Combining the symbols for distance, pitch, and yaw creates the condensed (three-letter) representation of the geometric relationship. These symbolic representations can optionally be added to the LM tokenizer as *special* tokens. Shorter token sequences generally decrease both the training and inference time.

Example Let the task be: “Put the soap into the drawer”. The input query to Graph2NL consists of the target object “soap”. Figure 2 shows the graph constructed by Graph2NL from augmented data (§3.5.1), including domain-specific knowledge. After finding the shortest paths between the root (‘agent’) and target node (‘soapbar’), Graph2NL produces an output in the following form (cut-off at search depth 2):

- ```
[Bathroom=
- closer below left sink near below back soapbar
- closer below left cabinet near above back soapbar
- closer above left countertop next above back soapbar
- close below back toilet closer below back soapbar
- closer below back garbagecan close below back soapbar]
```

The NL context by Graph2NL starts with the name of the room extracted from the scene graph, followed by the geometric description of each node connected to the target node on the path from the agent. “-” indicates the root note, i.e., the agent. For the

previous example, Graph2NL produces the following **condensed** form:

```
[Bathroom=
- fnk sink dnj soapbar
- fnj cabinet dmj soapbar
- fmk countertop gmj soapbar
- enj toilet fnj soapbar
- fnj garbagecan enj soapbar]
```

This form of state representation is unique for each problem configuration and forms the context that grounds RobLM.

### 3.7 Training

RobLM generates a plan as text given the goal and the context, which involves causal language modeling for text generation. Decoder-only autoregressive language models (Figure 3) are frequently used for the problem of text generation; we chose GPT-2 as the base model for RobLM. RobLM uses the base version of the GPT-2 PLM ('gpt-2') (Radford et al., 2021), loaded and initialized with pre-trained weights from the Huggingface (Wolf et al., 2019) Transformer library. Finetuning GPT-2 for causal language generation has a self-supervised setup, where the labels are the inputs shifted to the right, which entitles learning to predict the next token in a sequence. We finetune the GPT-2 model using the pre-processed training data of the ALFRED dataset, which has around 20,000 samples, with three sets of NL descriptions

for each sample. The ADAM (Kingma and Ba, 2014) optimizer is used with a learning rate of  $5e^{-5}$ , and the LM is trained for two epochs. Finetuning a GPT-2 LM to the ALFRED training data with a single GPU-accelerated computer takes around 30 min (27 iterations/s - measurement not representative due to hardware dependence).

### 3.8 Generation pipeline

For inference, RobLM takes only the NL task goal together with an optional context and outputs the complete step-by-step plan for completing the goal. This plan is composed of high-level instructions rather than low-level controller commands.

**Example.** Given the task "Put the soap into the drawer:", RobLM (no context) generates the plan:

```
Put the soap into the drawer:
0. GotoLocation(countertop)
1. PickupObject(soap)
2. GotoLocation(drawer)
3. PutObject(soap, drawer)
```

The plan is generated by consecutive forward passes through the Transformer model. For a vocabulary size of  $k$  and a token sequence of length  $l$  (with  $l \leq 1,024$  for GPT-2), the forward pass of the Transformer yields an output vector of size  $k \times l$  with values in the interval  $[0, 1]$ . The Transformer outputs *scores*, i.e., *logits*, for each token in the input sequence. These scores are converted to

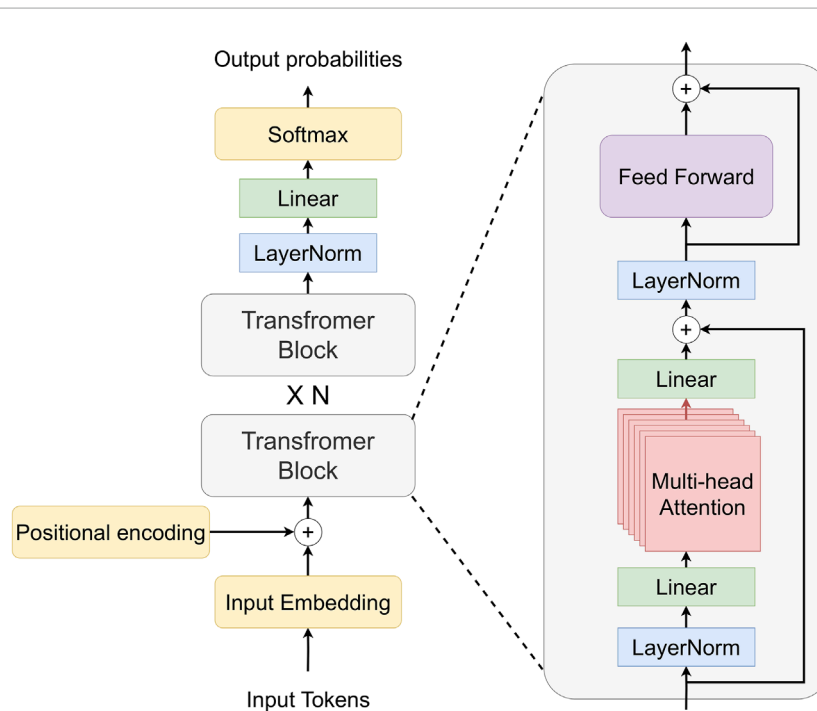
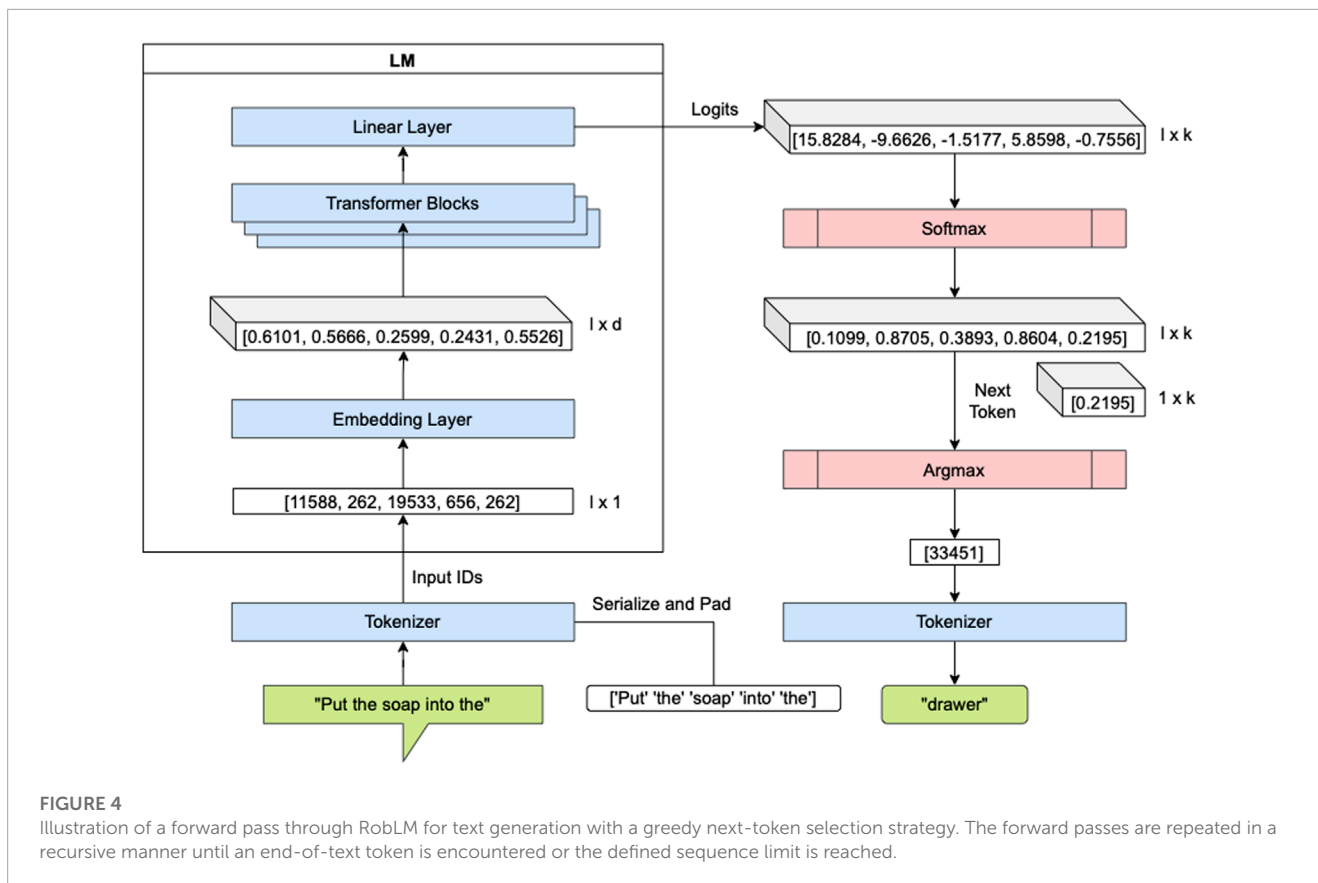


FIGURE 3

Decoder-only Transformer architecture. The input to the decoder is tokenized text, and the output is probabilities over the tokens in the tokenizer vocabulary. The positional encoding is added to the embedded input to account for the order. Transformer's decoder can have multiple transformer blocks, each of which contains multi-head attention with linear layers and layer normalization.



a probability distribution  $p(\mathbf{x})$  by using the Softmax function, as described in Eq. 2.

Two possible generation strategies for selecting the next token from  $p(\mathbf{x})$  are: *greedy search* and *top-k/top-p sampling* (Holtzman et al., 2020). In the greedy strategy, the token  $x_{sel}$  with the highest likelihood is picked with  $x_{sel} = \text{argmax} p(\mathbf{x})$ . In the top-k sampling strategy, as the name suggests, the scores are sorted, and one of the first  $k$  candidate tokens is randomly sampled. By extending the top-k sampling with an additional top-p strategy, the sum of the  $k$  candidates must be equal to or greater than  $p \in [0, 1]$ . Simply put, top-k widens the choice over the next tokens and top-p filters out low-probability tokens. Figure 4 illustrates, on the basis of an example, a forward pass through the Transformer with a greedy selection strategy. These steps are repeated recursively until an end-of-text token is encountered or the defined sequence limit is reached to generate the full plan.

This LM model was finetuned to generate a structured output, omitting special tokens, characterized by numbered actions and their arguments in parenthesis. The input is always part of the output, due to the generation function utilized by RobLM. Note that it is not guaranteed that the 'soap' can be found inside the 'drawer' on the 'countertop'. In fact, it could be at any possible location permitted by the environment. However, given a greedy search strategy, for the given task goal, the *likelihood* for the 'soap' being on the 'countertop' is the highest in this case.

### 3.8.1 Hardware setup

For finetuning LMs and evaluating each model, we used the Lichtenberg Cluster of TU Darmstadt, which contains stacks of NVIDIA<sup>®</sup> A100 and V100 GPUs. Internal tests have shown that a single GPU can decrease the training time by a factor of 10 (these tests are not representative because performance depends on every hardware component). To run experiments in the AI2-THOR simulation, we used a PC with an NVIDIA<sup>®</sup> RTX 3080Ti GPU.

## 4 Experiments

### 4.1 Preliminary analysis for task plan generation with GPT-2 and GPT-3

LLMs can represent knowledge from the data they have been trained on. However, the question remains, whether this knowledge can be leveraged to solve planning tasks, i.e., can LLMs reason? This is investigated by comparing the text-generation results of GPT-2 (Zero-Shot Learning (ZSL)) and GPT-3 (Few-Shot Learning (FSL)) for a planning task.

Given an instruction to a household robot, formulate the steps to complete the instruction.

The instruction is: "Put a washed slice of apple on the table."



The results for this task vary greatly between the two LLMs.

**GPT-2** (‘gpt-2’ model, 1.5B parameters) completely fails this task and produces an output that resembles a repetition of phrases in the input sentence:

```
Put a washed slice of apple on the table.
Put a washed slice of apple on the table.
Put a washed slice of apple on the table.
...
```

Similar behavior has been observed for other LLMs falling into the ZSL category. The input sentence is not an open-ended question and requires reasoning.

**GPT-3** (‘text-davinci-002’ model, 175B parameters), when given the same instruction as input, is able to make assumptions and formulate a whole plan based on these assumptions:

```
Assuming the apple is not cut and not
washed.
1. Pick up the apple from the counter.
2. Cut the apple into a slice.
3. Wash the apple slice.
4. Place the apple slice on the table.
```

The FSL paradigm allows GPT-3 to be very sensitive to context changes and seemingly *understand* the request at hand. However, smaller GPT-3 PLMs (GPT-3 curie, GPT-3 babbage, GPT3-ada) show a degraded quality in the produced plan (Floridi and Chiriatti, 2020). have shown that GPT-3 would not pass the Turing test, as to having “no understanding of the semantics and contexts of the request, but only a syntactic (statistical) capacity to associate words [...]”.

These tests have shown that plan generation capabilities of LLMs vary dramatically depending on the underlying learning paradigm, model architecture, and parameter size. GPT-2, out of the box, is completely unsuited for solving planning tasks that require a minimum level of text understanding. However, as later (§3.5) shown, GPT-2 can successfully generate plans when finetuned to a training dataset (§3.2). The question of whether a finetuned GPT-2 model can *leverage* knowledge for planning is addressed in the following section. GPT-3, unfortunately, is only accessible through a paid service by OpenAI, and finetuning of own GPT-3 models is possible through the provided service. Practical applications, however, are limited because each query has to be sent to and processed by the OpenAI service. Even if a PLM was made available, the hardware requirements for running GPT-3 models are immense, even for today’s standards, due to the sheer parameter count. It is for these reasons that GPT-3 and its newest versions are not considered as a basis for finetuning to RobLM.

## 4.2 Evaluation of RobLM

This section presents the main experiments conducted for evaluation of RobLM. We first define the appropriate metrics and a baseline method required to make the evaluations measurable and comparable. The *grounding* problem is explained in accordance with the practical aspects of integrating the available methods into the simulator. For the experimentation part, a set of finetuned LLMs is compared with the baseline performance.

### 4.2.1 Metrics

To validate a finetuned LM, only the NL task goal of each validation sample and optionally, the context is fed to the RobLM generation pipeline (see Figure 4). Validation is performed over each task category rather than all the validation data. This enables the analysis of a task-dependent performance: some task categories are more complex than others leading to a longer trajectory of actions and hence an increased difficulty. Two metrics are defined for validation: LM accuracy and plan success rate.

**Definition—Accuracy.** Accuracy measures how accurately the LM is able to predict the following parts of the plan.

- the correct count and names of all actions in the plan (action accuracy)
- the correct count and names of all arguments in the plan (argument accuracy)
- the correct count and names of all actions **and** arguments in the plan (“full plan” accuracy)

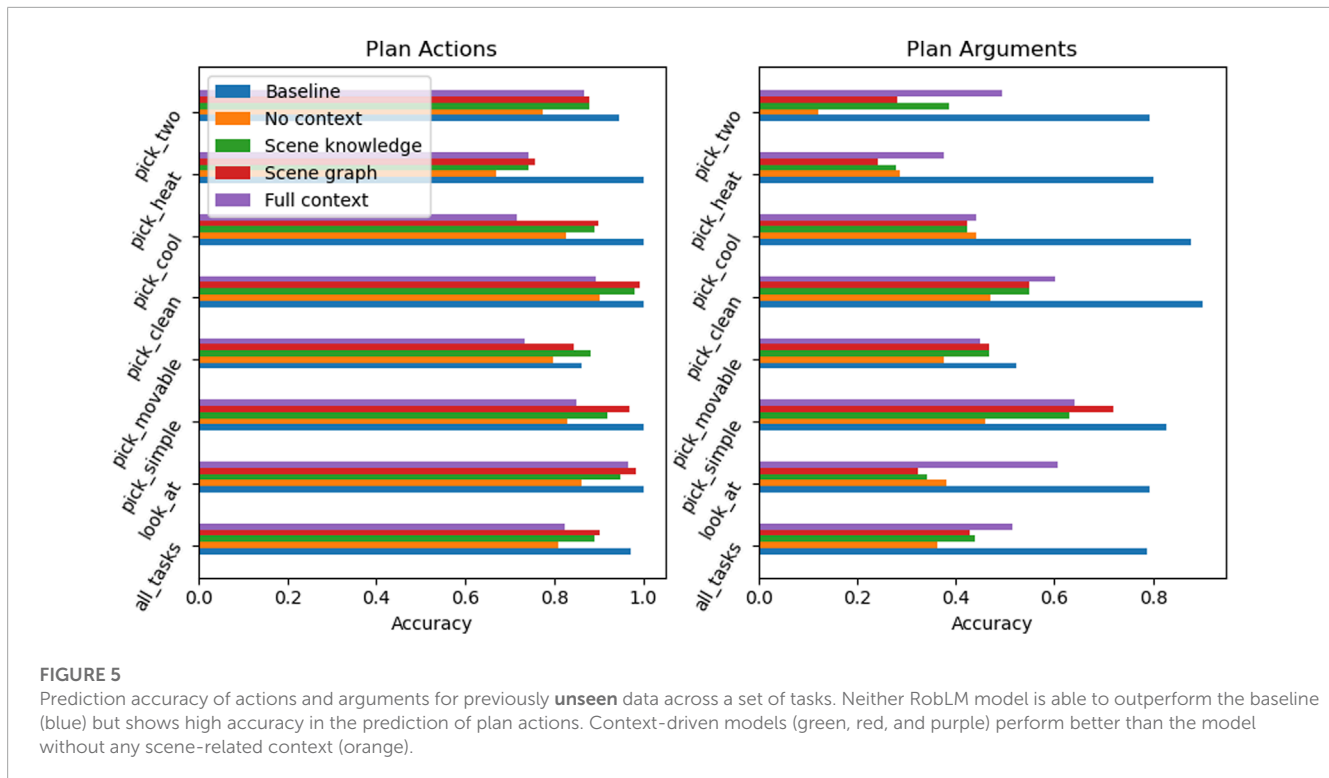
For a found plan, the accuracy of actions and arguments counts if **all** actions or arguments are correct. With this metric, it is possible to anchor the cause of plan failure to either the actions or the arguments, or both.

Having an accurate LM does not necessarily mean that the generated plan leads to success—at least, as long the “full plan” accuracy is below 1.0, i.e., the trajectory is not replicated perfectly. A second metric is required that measures the actual *success rate* of the finetuned LM in simulation. There are two possible scenarios that justify this additional metric. First, the plan could fail in simulation, even if it seems accurate. And second, the plan could succeed in simulation, even if the plan is not completely accurate.

**Definition—Success rate.** The success rate is a measure of the successful completion of individual sub-tasks of a validation task. After loading the trajectory, environment state, and goal from the validation sample into the AI2-THOR simulator, the actions predicted by the LM are translated into low-level controller actions via task and geometric grounding (§4.2.3), which are then passed to the AI2-THOR controller and executed in the simulator. After every simulator step, a check is performed to determine whether the target conditions for sub-task completion have been met. If the target conditions are kept unsatisfied after execution of the last low-level action, it counts as a success towards the sub-task, or otherwise, as a failure.

### 4.2.2 Baseline

A baseline is an oracle, or upper bound, that serves as a measurement reference. Fast Downward (FD) (Helmert, 2006) is used as the baseline for evaluation. We consider a classical task planner like FD appropriate since it also has access to the full domain and is a complete algorithm (Helmert, 2006). Therefore, the ability of a RobLM to match or outperform FD (for a given time budget) would reveal whether LLMs can be helpful towards learning task planning. Every ALFRED validation sample comes with a PDDL problem file, while the PDDL domain is shared by all tasks; this allows the PDDL planner to generate a plan for each sample. To generate a plan using FD, the PDDL problem files provided by ALFRED have to be pre-processed. FD is able to handle Action Description Language (ADL) instructions, as found in the PDDL



problem, but is not able to process optimization-related additional information present in the files.

### 4.2.3 Instruction grounding

*Grounding* can be defined as mapping a high-level, abstract, or symbolic representation to a low-level grounded representation. Grounding of an abstract plan to objects is called object or geometric grounding (or “world grounding”), and grounding of NL to robot tasks is called task grounding. In this case, instructions generated by the LM are made up of actions that require a *task grounding*, and arguments, which require a *geometric grounding*.

### 4.2.4 Task grounding

Plans generated by RobLM consist of high-level actions and are not directly executable by the AI2-THOR controller. Each possible action predicted by the LM has to be grounded to a task, which then translates to a sequence of low-level controller actions. For task grounding, three possible types of tasks are defined: navigation, manipulation and composite. In a navigation task, the agent is required to move from one to another location. In a manipulation task, the agent performs an action affecting the environment state. Composite tasks are a composition of manipulation tasks that need to be completed in a specific order.

Task grounding is performed as follows.

- The action *GotoLocation* is grounded to the navigation task and delegated to a trajectory planner for navigation (see below).
- The actions *PickupObject*, *PutObject*, *ToggleObject* and *SliceObject* are grounded to the manipulation task, the actions can be directly executed by the low-level controller.
- The actions *HeatObject*, *CoolObject* and *CleanObject* are grounded to the composite task, which is translated to

this sequence of low-level actions: *ToggleObject* → *PutObject* → *ToggleObject* → *ToggleObject* → *PickupObject* → *ToggleObject* (example given below).

### 4.2.5 Geometric grounding

An argument can be either a location or an object name. An argument produced by the LM might be ambiguous or non-existing in the environment. In order to be understood by the controller, these arguments have to be grounded on a geometric level. For grounding arguments, first, all available objects are retrieved from the simulation. Then, the world coordinates of all objects matching the predicted symbol (target object) are gathered. E.g., if the predicted target object is ‘soap’, the position of all ‘soap’-type objects can be queried and retrieved from the simulator. The low-level control commands are finally generated with the help of the ground-truth navigation graph of the scene.

### 4.2.6 Navigation

By overlaying the world with a grid, every position in the world is given a discrete coordinate. A navigation graph (not to be confused with a *scene graph* or Graph2NL graph) creates a node for each coordinate and connects all the nodes that are *accessible* one from another. Similar to the procedure of Graph2NL (§3.6), the navigation graph is traversed after locating the agent and target node by the object name. A search algorithm is used to find the shortest path in the graph from the agent to the target object - in this case, it is the A\* algorithm (Duchon et al., 2014). The search returns a sequence of nodes, which corresponds to a sequence of coordinates (a trajectory). Lastly, a motion planner takes the trajectory as an input and outputs a sequence of low-level controller actions (AI2-THOR conveniently provides a motion planner for navigation).

### 4.2.7 Experimental results

A set of finetuned RobLM models are evaluated against the baseline. The finetuned models differ in the amount of context provided during training time.

- 1) ‘No context’ — Only task goal
- 2) ‘Scene knowledge’ — List of all available objects in the environment, found in the PDDL problem
- 3) ‘Scene graph’ — Description of geometric relations to the target object, generated by Graph2NL
- 4) ‘Full context’ — Description of geometric relations of all objects, generated by Graph2NL

Given a PDDL problem file, Graph2NL automatically generates the context in the specified text format. This context is provided to the LM for training and inference.

**Accuracy.** Figure 5 summarizes the evaluation of the finetuned RobLM models compared to the FD baseline for previously unseen (validation) data. It can be observed that none of the finetuned RobLM models is able to outperform the baseline. Going through each of the models and starting with the ‘No context’ model it is surprising that this model, even without any contextual information, is able to generate the correct plan actions with high accuracy. The ‘Scene knowledge’ and ‘Scene graph’ models have a similar performance, the ‘Scene graph’ generally being slightly more accurate in both actions and argument prediction. Both of these models overall outperform the ‘No context’ model, with a significant improvement of the context models in the arguments prediction.

Given these results, the following conclusions about the examined models can be made.

- 1) Failed plans are mostly caused by wrong arguments (objects or locations) and only in some cases by wrong actions.
- 2) The LM is able to learn the structure of tasks, but not scene-dependent components.

RobLM is able to distinguish between the task categories and provide a correct task action plan. However, where this model fails is in finding all correct action arguments, i.e., locations and object names. This can be explained by the fact that the task goal alone does not reveal the actual location of the target object. Because the target can be in any accessible location in the environment, or in any accessible receptacle, the produced argument is the result of the LM imitating the *most-likely* cases observed in the training data.

Overall, these results are consistent with the point made on contextual information and prediction accuracy: giving to the model information about the environment, i.e., finetuning a model to be grounded to the scene, **does** improve performance.

**Success rate.** A plan is successful if each of the sub-tasks for a stated task is completed. Table 2 summarizes the success rate of RobLM compared to the baseline for actions of the navigation task (*GotoLocation*) and manipulation task (*PickupObject*, *PutObject*, etc.). Composite tasks have been omitted from this evaluation because of high failure rates caused by their task grounding complexity.

Regarding geometric grounding, arguments predicted by RobLM are grounded to all matching objects in the world, and RobLM is allowed to “try” all possibilities. E.g., when the objective is to “Get soap”, multiple ‘soap’-type objects could exist in the scene. Each possibility given by the geometric grounding is simulated

**TABLE 2 Success rates of sub-task completion in simulation—RobLM (‘No context’) compared to the baseline on seen and unseen validation data.**

| Success rate | Baseline     |              | RobLM (‘no context’) |              |
|--------------|--------------|--------------|----------------------|--------------|
|              | seen         | unseen       | seen                 | unseen       |
| GotoLocation | 0.318        | 0.393        | <b>0.422</b>         | <b>0.499</b> |
| PickupObject | 0.466        | 0.474        | <b>0.776</b>         | <b>0.749</b> |
| PutObject    | <b>0.385</b> | <b>0.331</b> | 0.116                | 0.092        |
| SliceObject  | 0.629        | 0.5          | <b>0.94</b>          | <b>0.98</b>  |
| ToggleObject | 0            | 0            | <b>0.84</b>          | <b>0.864</b> |

Bold represent maximum values.

**TABLE 3 Top-k and top-p sampling ( $k=10$  and  $p=0.9$ ) — tokens are sampled three times for the ‘Pick Simple’ task, giving only slight deviations in the final accuracy.**

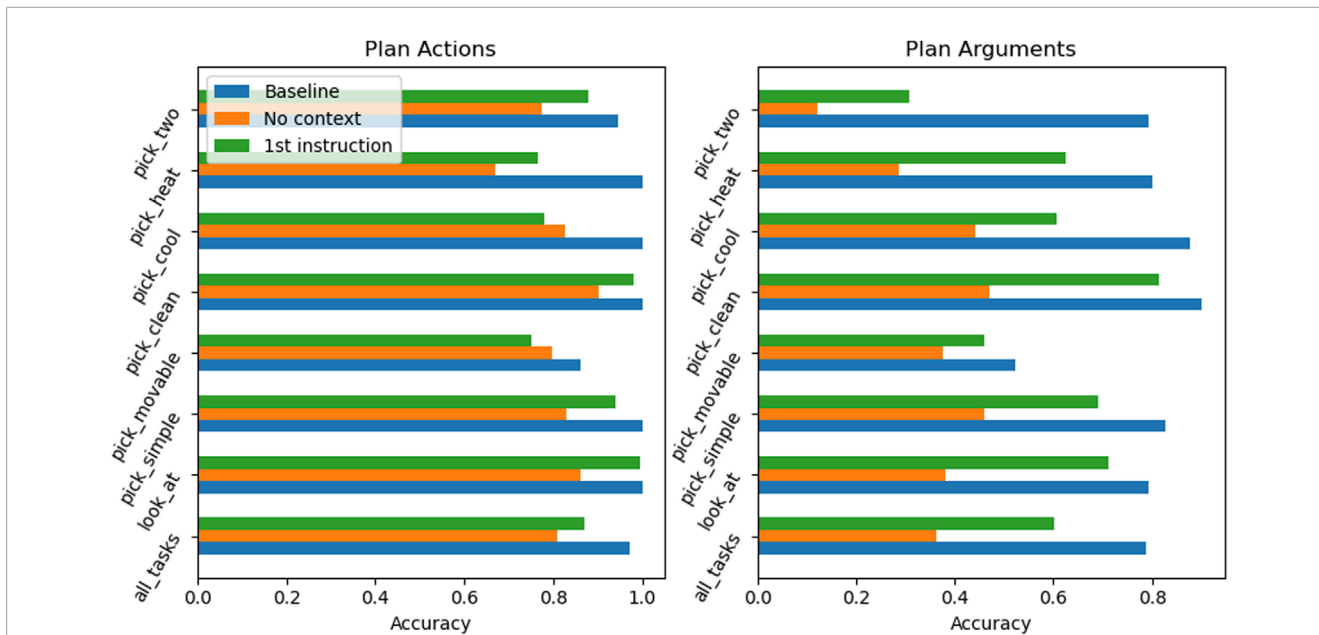
| RobLM ‘no context’ model, ‘pick simple’ task |            |            |            |
|----------------------------------------------|------------|------------|------------|
| Accuracy of                                  | 1st sample | 2nd sample | 3rd sample |
| Actions                                      | 0.7746     | 0.8169     | 0.7817     |
| Arguments                                    | 0.3803     | 0.4085     | 0.3944     |
| GotoLocation                                 | 0.8772     | 0.9123     | 0.8904     |
| PickupObject                                 | 0.8380     | 0.8662     | 0.8451     |
| PutObject                                    | 0.7971     | 0.8227     | 0.7986     |
| GotoLocation_Args                            | 0.5658     | 0.5877     | 0.5833     |
| PickupObject_Args                            | 0.7535     | 0.8028     | 0.7746     |
| PutObject_Args                               | 0.6449     | 0.6667     | 0.6331     |

by storing and restoring the simulator state. While this is a clear advantage for RobLM over the baseline, the evaluation still holds because the LM is required to predict the correct location or object names. Based on the presented results, the LM-based system performs well on sub-tasks requiring the action *PickupObject*, while the action *PutObject* does not succeed equally well, being far from the baseline performance.

Overall, the success rate of the baseline method is not nearly as high as expected, hinting at potential implementation-specific failures in the task grounding and in the low-level controller interaction with objects. In the low-level controller, visual information is not included. This means that the robot is controlled in a “blind flight” mode. The AI2-THOR simulation requires the target object to be in *view*. If the object is not visible, e.g., because the agent is looking in the wrong direction, the interaction fails and with it, the sub-task. Because of the fact that both systems have been evaluated within the same framework, these results **do not** dismiss a potential use-case for LM in planning.

### 4.2.8 Additional results

We provide additional experiments for a deeper analysis of potential points of failure of RobLM. These experiments entail a different sampling strategy and context refinement.



**FIGURE 6** Prediction accuracy of actions and arguments for unseen tasks of RobLM with a refined context. This experiment compares a model finetuned to the task goal and a context consisting of a NL description of the first plan instruction (green) with the baseline (blue) and a RobLM with ‘No context’ model (orange).

#### 4.2.8.1 Top-k/top-p sampling

So far, every experiment conducted has used a greedy next-token selection strategy. In order to be able to tell with certainty that a found plan is the “best possible” plan, a comparison with another sampling strategy is required. This additional experiment repeats the previous one, but this time with a top-k and top-p sampling strategy. The comparison is done with the ‘No context’ RobLM model for all tasks with  $k = 10$  and  $p = 0.9$ , i.e., tokens are sampled from the top-10 predictions and sum up to a probability  $\geq 0.9$ . Since a similar pattern was observed in the individual task evaluations, Table 3 reports the results for the ‘Pick Simple’ task only. Each token is sampled three times, giving three possible solutions to be evaluated. Slight—uniform and hence dismissable—variations in the prediction accuracy exist between these three runs. The sampling-based method performs slightly worse than the greedy strategy.

#### 4.2.8.2 Refined context

In a deeper analysis of RobLM failure cases, it has been found that the first argument in the generated plan is the hardest to predict correctly by the LM. The LM is not able to draw enough conclusions about the first instruction from the supplied context of any form. This causality becomes obvious after the following experiment: Given the task goal and a NL **description** of the first instruction as context, how does the overall accuracy of the LM change? The following text is an example of an instruction description in NL, as found in the ALFRED dataset:

Turn left and walk across the room towards the shelves on the wall.

The results in Figure 6 show that, given this extra information, RobLM is almost able to reach the performance levels of the baseline measurement across all tasks; it shows very high accuracy on “full plan” actions and arguments. The conclusion of this experiment is that the more precisely the supplied context is tailored towards the key issue of LM generation task, the more accurate the generated plan becomes. For this specific problem, finding the correct first argument is key to a successful plan, and with a NL description of the first instruction, the LM is able to draw the necessary connections from context to plan.

The overall conclusion of this observation is that the LM are adaptive; the LM is able to adapt new information into the plan generation, towards a more accurate sequence of instructions.

### 4.3 Run-time analysis

Inference frequency is an important factor when it comes to real-life applications. This is especially true for industrial robotics, where cycle times are important. But not every robotic application is time-critical, e.g., a household robot is not expected to respond in a sub-second time. However, if task planning is seen as a programming problem, a fast execution time greatly enhances the operator experience Table 4 shows a comparison of the inference speeds of RobLM against the baseline (FD). RobLM, in all cases, is slower compared to the baseline, which is likely due to the reliance on the full GPT-2 vocabulary size for the LM tokenizer and the usage of a LM-internal, implementation-specific generation function<sup>4</sup>. Such an issue can be mitigated by training a new

<sup>4</sup> Huggingface generation function: ‘transformers/src/transformers/generation\_utils.py’.

**TABLE 4 Comparison of inference speeds—RobLM against baseline. GPU acceleration used for LM (NVIDIA® GeForce RTX 2080 SUPER). The timer starts only after the program or model has been loaded into memory, i.e., only computation (inference) time is measured. “No context” has a maximum token sequence length of 200 and “Full context” has a maximum length of 1,024 tokens for generation.**

| Iterations per second (average over 800 samples) |                    |                      |
|--------------------------------------------------|--------------------|----------------------|
| Baseline                                         | RobLM ‘No context’ | RobLM ‘Full context’ |
| 2.9                                              | 1.0                | 0.2                  |

tokenizer on the task-specific vocabulary, but this comes at the cost of not utilizing the stored knowledge in the PLM. However, current progress in language models allows faster inferences in more advanced hardware than the one used in this work; therefore, we believe that the frequency limitations can be easily overcome.

**Remarks.** Overall, our analysis has shown that finetuning PLMs toward robotic task planning is possible when providing an appropriate grounding context. However, we have shown that such models cannot yet reach the planning abilities of classical task planners. A combination of finetuning with proper scene representation and a more elaborate sampling strategy, as well as the addition of more sophisticated prompts, can boost the performance of RobLM, leading them to performances that are closer to the oracle task planners. Still, the benefit of providing goal specifications as natural language commands alleviate the burden of engineering, while advances in scene graph generation can make the extraction of domain specifications autonomous. Therefore, we believe that using RobLMs at a higher level of abstraction for neuro-symbolic task planning is valuable but is still in its infancy. Additional challenges have been recently summarized by [Weng \(2023\)](#), where some of the listed points are in accordance with our findings.

## 5 Conclusion

We presented a framework for finetuning grounded Large Language Models (LLMs) and investigated the applicability of such models combined with planning in solving long-horizon robot reasoning tasks. This paper has shown that LLMs can extract commonsense knowledge through precise queries and adjust their behavior based on available information or context. Among our contributions are the development of RobLM, a grounded finetuned LLM that generates plans directly from natural language commands, and Graph2NL, which creates natural language text describing graph-based data, to represent scene graphs as inputs into RobLM. Our extensive experimental results have revealed, nevertheless, the challenges in representing structured and geometric data in natural language. However, LLMs still need to demonstrate a consistent ability to perform long-horizon planning tasks and cannot yet replace classical planners. Despite their limitations, LLMs possess powerful features such as efficient storage and retrieval of commonsense knowledge, which can be useful in planning tasks when presented with partially observable environments.

For future work, exploring larger models like GPT-3 or GPT-NeoX could increase the accuracy and success rate of RobLM. Providing structured context to the Transformer model and exploring multi-modal inputs, such as visual information, may also improve the planning capabilities of LLMs. Further research in the field of applied natural language processing in robotics could help unlock the full potential of LLMs and contribute to the development of more advanced neuro-symbolic planning systems.

## Data availability statement

The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found below: <https://github.com/dnandha/RobLM>.

## Author contributions

GC, AY, and DN contributed to the conception and design of the method. AL assisted in the setup of the baseline method. LR provided insights for the training and fine-tuning of language models and advice on the linearization of the graph. IG gave advice on the overall method and the idea of using language models for planning. GC wrote the current version of the manuscript, building on the initial write-up by DN. AY assisted in writing and visualizations. All authors contributed to the article and approved the submitted version.

## Funding

This research has been supported by the German Research Foundation (DFG) through the Emmy Noether Programme (CH 2676/1-1) and by the Hessian AI Connectom Fund “Robot Learning of Long-Horizon Manipulation bridging Object-centric Representations to Knowledge Graphs”. We acknowledge support by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) and the Open Access Publishing Fund of Technical University of Darmstadt.

## Acknowledgments

The authors would like to acknowledge Snehal Jauhri and Haasung Li for the fruitful discussions and suggestions.

## Conflict of interest

Author LR was employed by Amazon Alexa.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

## Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated

organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

## References

- Aeronautiques, C., Howe, A., Knoblock, C., McDermott, I. D., Ram, A., Veloso, M., et al. (1998). Pddl- the planning domain definition language. *Tech. Rep. Tech. Rep.*
- Bengio, Y., Ducharme, R., and Vincent, P. (2000). A neural probabilistic language model. *Adv. neural Inf. Process. Syst.* 13.
- Bian, N., Han, X., Sun, L., Lin, H., Lu, Y., and He, B. (2023). Chatgpt is a knowledgeable but inexperienced solver: an investigation of commonsense problem in large language models. Available at: <https://arxiv.org/abs/2303.16421>.
- Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., et al. (2021). On the opportunities and risks of foundation models. Available at: <https://arxiv.org/abs/2108.07258>.
- Brohan, A., Chebotar, Y., Finn, C., Hausman, K., Herzog, A., Ho, D., et al. (2022). Do as i can, not as i say: grounding language in robotic affordances. Available at: <https://arxiv.org/abs/2204.01691>.
- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., et al. (2020). Language models are few-shot learners. Available at: <https://arxiv.org/abs/2005.14165>.
- Chen, B., Xia, F., Ichter, B., Rao, K., Gopalakrishnan, K., Ryoo, M. S., et al. (2022a). Open-vocabulary queryable scene representations for real world planning. Available at: <https://arxiv.org/abs/2209.09874>.
- Chen, W., Hu, S., Talak, R., and Carlone, L. (2022b). Leveraging large language models for robot 3d scene understanding. Available at: <https://arxiv.org/abs/2209.05629>.
- Driess, D., Ha, J.-S., and Toussaint, M. (2020). Deep visual reasoning: learning to predict action sequences for task and motion planning from an initial scene image. Available at: <https://arxiv.org/abs/2006.05398>.
- Driess, D., and Toussaint, M. (2019). *Hierarchical task and motion planning using logic-geometric programming (hlgp)*.
- Duchoň, F., Babinec, A., Kajan, M., Beňo, P., Florek, M., Fico, T., et al. (2014). Path planning with modified a star algorithm for a mobile robot. *Procedia Eng.* 96, 59–69. doi:10.1016/j.proeng.2014.12.098
- Floridi, L., and Chiriatti, M. (2020). Gpt-3: its nature, scope, limits, and consequences. *Minds Mach.* 30, 681–694. doi:10.1007/s11023-020-09548-1
- Funk, N., Chalvatzaki, G., Belousov, B., and Peters, J. (2021). Learn2assemble with structured representations and search for robotic architectural construction. *Conf. Robot Learn. (CoRL)*.
- Funk, N., Menzenbach, S., Chalvatzaki, G., and Peters, J. (2022). Graph-based reinforcement learning meets mixed integer programs: an application to 3d robot assembly discovery. Available at: <https://arxiv.org/abs/2203.04120>.
- Garrett, C. R., Chitnis, R., Holladay, R., Kim, B., Silver, T., Kaelbling, L. P., et al. (2021). Integrated task and motion planning. Available at: <https://arxiv.org/abs/2010.01083>.
- Garrett, C. R., Lozano-Pérez, T., and Kaelbling, L. P. (2020). Pddlstream: integrating symbolic planners and blackbox samplers via optimistic adaptive planning. Available at: <https://arxiv.org/abs/1802.08705>.
- Helmert, M. (2006). The fast downward planning system. *J. Artif. Intell. Res.* 26, 191–246. doi:10.1613/jair.1705
- Hoang, C., Sohn, S., Choi, J., Carvalho, W., and Lee, H. (2021). Successor feature landmarks for long-horizon goal-conditioned reinforcement learning. *Adv. Neural Inf. Process. Syst.* 34, 26963–26975.
- Holtzman, A., Buys, J., Du, L., Forbes, M., and Choi, Y. (2020). The curious case of neural text degeneration. Available at: <https://arxiv.org/abs/1904.09751>.
- Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., et al. (2019). Parameter-efficient transfer learning for nlp. Available at: <https://arxiv.org/abs/1902.00751>.
- Huang, C., Mees, O., Zeng, A., and Burgard, W. (2022a). Visual language maps for robot navigation. Available at: <https://arxiv.org/abs/2210.05714>.
- Huang, W., Abbeel, P., Pathak, D., and Mordatch, I. (2022b). Language models as zero-shot planners: extracting actionable knowledge for embodied agents. Available at: <https://arxiv.org/abs/2201.07207>.
- Huang, W., Xia, F., Xiao, T., Chan, H., Liang, J., Florence, P., et al. (2022c). Inner monologue: embodied reasoning through planning with language models. Available at: <https://arxiv.org/abs/2207.05608>.
- Jiang, Y., Gupta, A., Zhang, Z., Wang, G., Dou, Y., Chen, Y., et al. (2022). Vima: general robot manipulation with multimodal prompts. In Proceedings of the NeurIPS Foundation Models for Decision Making Workshop, New Orleans, USA, December 2022
- Kaelbling, L. P., and Lozano-Pérez, T. (2011). Hierarchical task and motion planning in the now. In Proceedings of the 2011 IEEE International Conference on Robotics and Automation, 1470–1477. Shanghai, China, May 2011
- Kim, B., Wang, Z., Kaelbling, L. P., and Lozano-Pérez, T. (2019). Learning to guide task and motion planning using score-space representation. *IJRR38*, 793–812. doi:10.1177/0278364919848837
- Kingma, D. P., and Ba, J. (2014). Adam: A method for stochastic optimization. <https://arxiv.org/abs/1412.6980>.
- Kolve, E., Mottaghi, R., Han, W., VanderBilt, E., Weihs, L., Herrasti, A., et al. (2017). Ai2-thor: an interactive 3d environment for visual ai. Available at: <https://arxiv.org/abs/1712.05474>.
- LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature* 521, 436–444. doi:10.1038/nature14539
- Li, S., Puig, X., Paxton, C., Du, Y., Wang, C., Fan, L., et al. (2022). Pre-trained language models for interactive decision-making. *Adv. Neural Inf. Process. Syst.* 35, 31199–31212.
- Li, X. L., Kuncoro, A., d'Áutume, C. d. M., Blunsom, P., and Nematzadeh, A. (2021). Do language models learn commonsense knowledge? Available at: <https://arxiv.org/abs/2111.00607>.
- Liang, J., Huang, W., Xia, F., Xu, P., Hausman, K., Ichter, B., et al. (2022). Code as policies: language model programs for embodied control. Available at: <https://arxiv.org/abs/2209.07753>.
- Mees, O., Borja-Diaz, J., and Burgard, W. (2022). Grounding language with visual affordances over unstructured data. Available at: <https://arxiv.org/abs/2210.01911>.
- Nair, S., and Finn, C. (2019). Hierarchical foresight: self-supervised learning of long-horizon tasks via visual subgoal generation. Available at: <https://arxiv.org/abs/1909.05829>.
- Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., et al. (2022). Training language models to follow instructions with human feedback. <https://arxiv.org/abs/2203.02155>.
- Pashevich, A., Schmid, C., and Sun, C. (2021). Episodic transformer for vision-and-language navigation. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Montreal, BC, Canada, October 2021, 15942–15952.
- Pfeiffer, J., Kamath, A., Rücklé, A., Cho, K., and Gurevych, I. (2021). Adapterfusion: non-destructive task composition for transfer learning. Available at: <https://arxiv.org/abs/2005.00247>.
- Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., et al. (2021). "Learning transferable visual models from natural language supervision," in International conference on machine learning (PMLR), 8748–8763.
- Raman, S. S., Cohen, V., Rosen, E., Idrees, I., Paulius, D., and Tellex, S. (2022). Planning with large language models via corrective re-prompting. <https://arxiv.org/abs/2211.09935>.
- Ren, T., Chalvatzaki, G., and Peters, J. (2021). Extended task and motion planning of long-horizon robot manipulation. <https://arxiv.org/abs/2103.05456>.
- Ruis, L., Khan, A., Biderman, S., Hooker, S., Rocktäschel, T., and Grefenstette, E. (2022). Large language models are not zero-shot communicators. Available at: <https://arxiv.org/abs/2210.14986>.
- Shah, D., Osinski, B., and Levine, S. (2023). "Lm-nav: robotic navigation with large pre-trained models of language, vision, and action," in Conference on robot learning (PMLR), 492–504.
- Shridhar, M., Thomason, J., Gordon, D., Bisk, Y., Han, W., Mottaghi, R., et al. (2020). Alfred: A benchmark for interpreting grounded instructions for everyday tasks. Available at: <https://arxiv.org/abs/1912.01734>.

- Singh, I., Blukis, V., Mousavian, A., Goyal, A., Xu, D., Tremblay, J., et al. (2022). Progprompt: generating situated robot task plans using large language models. Available at: <https://arxiv.org/abs/2209.11302>.
- Tay, Y., Wei, J., Chung, H. W., Tran, V. Q., So, D. R., Shakeri, S., et al. (2022). Transcending scaling laws with 0.1% extra compute. Available at: <https://arxiv.org/abs/2210.11399>.
- Toussaint, M. (2015). Logic-geometric programming: an optimization-based approach to combined task and motion planning. Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015), Buenos Aires, Argentina, July 2015.
- Valmeekam, K., Olmo, A., Sreedharan, S., and Kambhampati, S. (2022). Large language models still can't plan (a benchmark for llms on planning and reasoning about change). Available at: <https://arxiv.org/abs/2206.10498>.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). Attention is all you need. Available at: <https://arxiv.org/abs/1706.03762>.
- Wang, A., Pruksachatkun, Y., Nangia, N., Singh, A., Michael, J., Hill, F., et al. (2019). Superglue: A stickier benchmark for general-purpose language understanding systems. *Adv. neural Inf. Process. Syst.* 32.
- Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2018). Glue: A multi-task benchmark and analysis platform for natural language understanding. Available at: <https://arxiv.org/abs/1804.07461>.
- Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., et al. (2022a). Emergent abilities of large language models. Available at: <https://arxiv.org/abs/2206.07682>.
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E. H., et al. (2022b). Chain-of-thought prompting elicits reasoning in large language models. Available at: <https://arxiv.org/abs/2201.11903>.
- Wells, A. M., Dantam, N. T., Shrivastava, A., and Kavraki, L. E. (2019). Learning feasibility for task and motion planning in tabletop environments. *IEEE Ral.* 4, 1255–1262. doi:10.1109/lra.2019.2894861
- Weng, L. (2023). Ll-powered autonomous agents. *lilianweng.github.io*.
- White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., et al. (2023). A prompt pattern catalog to enhance prompt engineering with chatgpt. Available at: <https://arxiv.org/abs/2302.11382>.
- Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., et al. (2019). Huggingface's transformers: state-of-the-art natural language processing. Available at: <https://arxiv.org/abs/1910.03771>.
- Xu, L., Ren, T., Chalvatzaki, G., and Peters, J. (2022). Accelerating integrated task and motion planning with neural feasibility checking. Available at: <https://arxiv.org/abs/2203.10568>.
- Zeng, A., Wong, A., Welker, S., Choromanski, K., Tombari, F., Purohit, A., et al. (2022). Socratic models: composing zero-shot multimodal reasoning with language. Available at: <https://arxiv.org/abs/2204.00598>.
- Zhou, X., Zhang, Y., Cui, L., and Huang, D. (2020). Evaluating commonsense in pre-trained language models. *Proc. AAAI Conf. Artif. Intell.* 34, 9733–9740. doi:10.1609/aaai.v34i05.6523