Generation of Robot Manipulation Plans Using Generative Large Language Models

Jan-Philipp Töberg
Center for Cognitive Interaction Technology (CITEC)
and *Joint Research Center on Cooperative and Cognition-enabled AI (CoAI JRC)*
Bielefeld University
Bielefeld, Germany
jtoeberg@techfak.uni-bielefeld.de

Philipp Cimiano
Center for Cognitive Interaction Technology (CITEC)
and *Joint Research Center on Cooperative and Cognition-enabled AI (CoAI JRC)*
Bielefeld University
Bielefeld, Germany
cimiano@techfak.uni-bielefeld.de

2023 Seventh IEEE International Conference on Robotic Computing (IRC) | 979-8-3503-9574-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/IRC59093.2023.00039

***Abstract*—Designing plans that allow robots to carry out actions such as grasping an object or cutting a fruit is a time-consuming activity requiring specific skills and knowledge. The recent success of Generative Large Language Models (LLMs) has opened new avenues for code generation. In order to evaluate the ability of LLMs to generate code representing manipulation plans, we carry out experiments with different LLMs in the CRAM framework. In our experimental framework, we ask an LLM such as ChatGPT or GPT-4 to generate a plan for a specific target action given the plan (called designator within CRAM) for a given reference action as an example. We evaluate the generated designators against a ground truth designator using machine translation and code generation metrics, as well as assessing whether they compile. We find that GPT-4 slightly outperforms ChatGPT, but both models achieve a solid performance above all evaluated metrics. However, only $\sim$36% of the generated designators compile successfully. In addition, we assess how the chosen reference action influences the code generation quality as well as the compilation success. Unexpectedly, the action similarity negatively correlates with compilation success. With respect to the metrics, we obtain either a positive or negative correlation depending on the used model. Finally, we describe our attempt to use ChatGPT in an interactive fashion to incrementally refine the initially generated designator. On the basis of our observations we conclude that the behaviour of ChatGPT is not reliable and robust enough to support the incremental refinement of a designator.**

***Index Terms*—Robot Plan Generation, Large Language Models, Action Similarity, CRAM, GPT**

I. Introduction

Robots have the potential to support humans in various household activities including cleaning (e.g. vacuuming) or gardening (e.g. lawn mowing). Despite current research focused on establishing cognitive robots in the household domain, their ability to support us in more complex everyday tasks is still very limited. The ability of a robot to perform specific actions hinges on the availability of a corresponding plan in symbolic architectures [1]. However, manually covering all potential environments, objects or task variations is unfeasible. This gap can potentially be covered by generative large language models (LLM). Trained on vast amounts of natural language data, they show promising results on tasks related to generating natural language texts and code [2]. Using such LLMs to generate manipulation plans for specific tasks in known environments can potentially decrease the development time and the need for human experts to be involved in the robot's learning process.

To assess the capabilities of generative LLMs with respect to the task of generating robot manipulation plans, we perform an experiment using the LLMs ChatGPT and GPT-4 [3]. As a framework for the robot manipulation plans, we use the CRAM [4] framework, which is a hybrid cognitive architecture describing tasks as general high-level goals from which low-level motions can be derived. Plans in CRAM are written in Common LISP and consist of so called *designators*, which are placeholders for actions, objects, locations or motions.

In our experiment, we prompt ChatGPT and GPT-4 to generate an action designator for a specific action (target action) based on another action as a reference. All our results are publicly available in a GitHub repository[1]. In addition to assessing the capabilities for generating such plans, we also investigate how the similarity between the target and the reference action influences the quality of the generated designators. Our hypothesis is that prompting the plan with a similar or related action should increase the likelihood for the LLM to generate a suitable plan for the target action. Finally, beyond a one-shot-setting, we test the ability of LLMs to incrementally refine an initial plan guided by a human user that promps the LLM to fix mistakes and add missing functionality.

To summarise, our paper's contributions are as follows:

- We generate CRAM designators in a one-shot fashion for nine different actions and assess their quality using machine translation and code generation metrics as well as their compilation success. Despite a solid performance level between 0.5 and 0.7 for all machine translation metrics and even results above 0.9 for the code generation metric, only about 36% of generated designators compile successfully.
- We analyse how the similarity of the used reference action to the target action influences code generation quality and compilability, showing that, against our expectations, action similarity negatively correlates with compilation success. Also, it is either negatively or positively cor-

[1]https://github.com/ag-sc/CRAM-Generation-LLM

Authorized licensed use limited to: STAATS U UNIBIBL BREMEN. Downloaded on April 22,2025 at 09:42:25 UTC from IEEE Xplore. Restrictions apply.

related with machine translation and code generation metrics, depending on the model.
- We demonstrate the limitations of ChatGPT in an interactive incremental scenario in which an initial plan/designator is refined in interaction with a human user.

The structure of our paper is as follows: We begin with describing related work and similar approaches in Section II. We present the design of our experiment (what data/LLMs we use, how we prompt the LLMs, etc.) in Section III and its results regarding compilation success and code generation metrics in Section IV. In Section V we investigate the correlation between action similarity and code quality, whereas Section VI outlines the results of our interactive refinement scenario experiment. Lastly, we describe limitations to our experiment in Section VII and conclude the paper in Section VIII.

II. Related Work

Due to the increasing generative capacity of LLMs, especially from the GPT family, there are multiple other approaches working on using these models for the robotics domain. Approaches such as proposed by Huang et al. [5] as well as You et al. [6] use LLMs for generating action sequences that are parsed to create manipulation plans, whereas Liang et al. [7] and Singh et al. [8] use LLMs to generate Python code capable of accessing APIs grounding the code in the robot's action-perception-loop. Another way of representing the generated plans is proposed by Cao and Lee [9], where LLMs are used to generate behaviour trees, and by Pallagani et al. [10], where the generated plans are represented using the PDDL format. Lastly, Wray et al. [11] rely on LLMs in order to, instead of generating the complete plan, fill in gaps in the knowledge necessary to achieve the robot's current goals.

Most of these approaches rely on the family of LLMs developed by OpenAI.

GPT-3 [12] is used by most of the proposed approaches [5, 7, 8, 9, 10, 11]. However, some of them also use Codex, which is a model specialised on generating source code and which was recently integrated into ChatGPT [5, 7, 8, 10]. GPT-4 [3] is only used by one approach, i.e. the approach by You et al. [6]

In general, the work that is closest to our proposal is the one of Liang et al. [7] and Singh et al. [8], who also use a programming language (Python in their case, Common LISP in our case) to represent the robot manipulation plan generated by the LLM. In order to ground the generated action plans, they use manually created Python APIs, whereas we rely on the CRAM framework instead [4].

While previous work has focused on the experimental comparison of the impact of different prompting strategies in one-shot [7] or few-shot [8] scenarios, as main novelty we prompt the model with an example designator for a reference action and experimentally examine the suitability of the generated plan in dependence of how semantically related the reference action is. This question has not been investigated in previous work.

```
(defaction slice-food (food-object)
    :precondition (and (is-sliceable food-object)
        (not (is-sliced food-object)))
    :effect (and (is-sliced food-object)
        (small-slice (make-slice food-object))
        (big-slice (make-slice food-object)))
    :body (progn (cut-food-object food-object)))
```

Fig. 1: Zero-Shot Generation for the *slicing* action using the `gpt-3.5-turbo-0301` model.

III. Experiment Design

For this experiment, we consider nine different action designators that have been manually created by experts. These cover the actions: *closing (a container)*, *cutting a food object in two halves*, *holding an object (in place)*, *opening (a container)*, *picking (an object) up*, *placing (an object) down*, *pouring (a liquid)*, *slicing (a food object)* and *wiping (a table)*. We gathered these designators from the official CRAM [4] GitHub repository[2] and from the *FoodCutting* repository [3]. We summarise these actions in Table I by providing a short, one sentence long description as well as pointing to the WordNet synset [13] that best describes the action.

Additionally, we indicate the number of lines of code (*LoC*) of the manually created designator for this action.

For the experiment, we employ one-shot learning by providing one of the manually created designators as an example in the prompt. We do not use zero-shot learning since the results, as exemplified in Figure 1, are visibly worse compared to one-shot learning. Another advantage is that a single run provides us with 72 (8*9) different generated designators, since each action can be used to generate all other actions. To achieve this, we employ the prompt shown in Figure 2. In addition to providing one example designator and linking each action to its brief description, the prompt shapes the LLM's answer in a way that simplifies the automatic extraction of the resulting designator. Apart from this prompt, we set the *temperature* parameter to zero to create results that are as deterministic as possible. As the output generated by LLMs can still be non-deterministic, we repeat the experiment five times and average results over these five runs.

Further, we use and compare two different snapshots of the ChatGPT model (`gpt-3.5-turbo`) and one version of the GPT-4 model [3] (`gpt-4-0613`). The two versions of ChatGPT are: `gpt-3.5-turbo-0301`, which was published in March 2023 and is based on training data going up until June 2021, and the `gpt-3.5-turbo-0613` model published in June 2023 and trained on data up until September 2021. For all three models, we run the experiment five times using the same actions, the same prompt and the same settings.

We rely on three metrics for computing the semantic similarity between the target and reference action. We calculate the Wu-Palmer-Similarity [14] based on the WordNet synsets [13] (see Table I) and the cosine similarity using GloVe

[2]https://github.com/cram2/cram
[3]https://github.com/Food-Ninja/FoodCutting

Authorized licensed use limited to: STAATS U UNIBIBL BREMEN. Downloaded on April 22,2025 at 09:42:25 UTC from IEEE Xplore. Restrictions apply.

TABLE I: The nine different actions, their abbreviation, their description to be inserted into the prompt, the most fitting WordNet [13] synset and the number of lines of code (LoC) of their designator. The comments in the designators were removed but empty lines added for readability were kept.

Action	Abb.	Description	Synset	LoC
Close	C	Closing an arbitrary container	`close.v.02`	47
Halve	Ha	Cutting an arbitrary (food) object into 2 halves	`halve.v.01`	58
Hold	Ho	Holding an object firmly in its gripper	`hold.v.02`	52
Open	O	Opening a arbitrary container	`open.v.01`	46
Pick-Up	P-U	Picking an object up	`pick_up.v.01`	46
Place-Down	P-D	Placing the held object at a location	`set_down.v.04`	46
Pour	P	Pouring the content of one container into another container	`decant.v.01`	56
Slice	S	Cutting an arbitrary (food) object into one small and one big slice	`slice.v.03`	55
Wipe	W	Cleaning a surface using some kind of towel	`wipe.v.01`	35

The following LISP source code describes a CRAM designator for the action of "*[reference action]*", where the executing robot would be *[reference action description]*:
[reference designator]
Can you please take this example and create a new designator for the action "*[target action]*", where the robot should be *[target action description]*. Your answer should only include the designator and no additional text.

Fig. 2: The prompt given to the LLM to generate the designator for *[target action]* based on *[reference action]*. For both actions we provide a brief description (see Table I) and the manually created reference designator.

embeddings [15] to measure the semantic similarity between the two actions. Additionally, we calculate the Sensorimotor Distance (SMD) [16] between two concepts[4], which grounds its semantic similarity in a vector space spanning six different sensory (auditory, gustatory, haptic, interoceptive, olfactory & visual) and five different motor effector (hand/arm, head, foot/leg, torso & mouth) dimensions.

In order to automatically estimate the suitability of the generated designators, we rely on five metrics that originate from the assessment of machine translation: BLEU [17], ROUGE-1 (R-1), ROUGE-2 (R-2) & ROUGE-L (R-L) [18] as well as chrF [19]. However, as these metrics have been designed for the evaluation of natural language translations, they are not ideally suited to assess code, as two code fragments with equal functionality but different variable names might receive a lower score. Despite this limitation, a recent study by Evtikhiev et al. [20] found that machine translation metrics are still helpful in analysing source code generation capabilities and especially the metrics chrF and ROUGE-L positively correlate with human judgement regarding code quality.

Recent approaches try to overcome this challenge, however, evaluating the functionality of source code is complex and depends on the programming language. Since the CRAM designators are written in Common LISP, a programming language used mainly in specific application areas, metrics like CodeBLEU [21], which is only applicable on languages like Java or Python, cannot be used. However, we additionally calculate the CodeBERTScore (CBS) [22], which encodes the programmatic context surrounding the generated code in addition to the generated tokens. To achieve this goal, CodeBERTScore relies on pre-trained models that are specific for a given programming language. However, such a model is not available for Common Lisp. Thus, the score we calculate relies on the general model not trained for a specific programming language.

Due to the limitations of the aforementioned metrics, we also evaluate the quality of the generated designators by trying to compile them using the Emacs IDE, as suggested in the CRAM tutorials[5]. Due to the large amount of generated designators (72 designators for 3 models generated in 5 runs = 1080 designators) and the scope of our experiment, we do not simulate the successfully compiled designators. Instead we manually analyse all 72 designators generated in the first of the five experimental runs for the `gpt-3.5-turbo-0301` model.

IV. Experiment Results

We begin by manually analysing the first 72 designators generated by the `gpt-3.5-turbo-0301` model. For each designator, we compare it to the ground truth designator and count the number of lines that are *added, deleted, changed* or *unchanged*. A *changed* line only contains some renaming but has the same structure and general parts. Lines containing substantial changes or being essentially different are counted as *added*, whereas lines that occur in the manual designator but not in the generated are regarded as *deleted*.

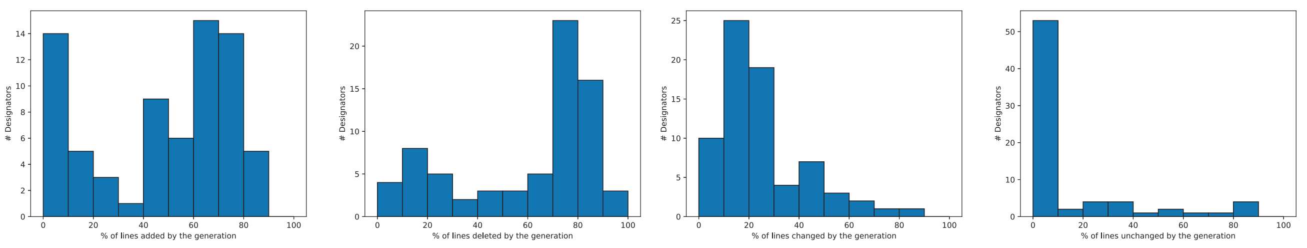
In general, as can be observed in Figure 3, the amount of lines that underwent any of the four possible changes is not evenly distributed.

Beginning with the amount of added lines in Figure 3a, we can see that either the majority of lines are added ($60-80\%$) or none at all ($0-10\%$), with only some cases in between. This means that most of the generated designators consists either mostly of new lines or contain almost no new lines. In a similar manner, for the deleted lines visualised in Figure 3b the majority of generated designators delete between 70 and

[4]Since each concept is represented by a single word, we use *pick* and *place* instead of *pick-up* and *place-down*.

[5]https://cram-system.org/doc/ide

Authorized licensed use limited to: STAATS U UNIBIBL BREMEN. Downloaded on April 22,2025 at 09:42:25 UTC from IEEE Xplore. Restrictions apply.



(a) $\overline{X} = 47.18\%,\ \sigma = 28.27\%$ (b) $\overline{X} = 59.92\%,\ \sigma = 28.44\%$ (c) $\overline{X} = 24.28\%,\ \sigma = 16.68\%$ (d) $\overline{X} = 15.71\%,\ \sigma = 23.23\%$

Fig. 3: Percentage of lines in the generated designator that were (a) added (b) deleted (c) changed or (d) not changed in comparison to the manually created target designator.

TABLE II: Averaged results for the machine translation and code generation metrics for the three models. The averaged results for each of the 72 action combinations can be examined in our GitHub repository.

Model	BLEU	R-1	R-2	R-L	chrF	CBS
`gpt-3.5-turbo-0301`	.595	.630	.527	.621	.674	.942
`gpt-3.5-turbo-0613`	.579	.614	.511	.612	.639	.940
`gpt-4-0613`	.605	.631	.532	.623	.674	.945

TABLE III: Compilation results for the three different models.

Model	Compiles	¬Compiles
`gpt-3.5-turbo-0301`	147 / 40,83%	213 / 59,17%
`gpt-3.5-turbo-0613`	101 / 28,06%	259 / 71,94%
`gpt-4-0613`	139 / 38,61%	221 / 61,39%
Σ	387 / 35,83%	693 / 64,17%

90%. In combination with the, on average, lower percentages of added lines, we can conclude that the generated designators are generally shorter than the target designators. Additionally, there seems to be a high difference between the generated and target designators, since only about 10 to 30% of lines were changed, as can be examined in Figure 3c. This is also underlined by Figure 3d, which shows that in the majority of cases only between 0 and 10% were the same as in the target designator.

In addition to these quantitative results, we look at the concrete differences between the generated and the manually created target designators. In the majority of cases, defining and/or returning variables were handled wrongly. This can either come from a different number of variables that are defined in the header of the designator or by (not) mapping the calculated variables back at the end of the designator. Apart from these general mistakes, most other mistakes result from the fact that the LLM copies inappropriate elements from the reference designator including inappropriate poses, unnecessary trajectories or grounding to non-existing elements in the environment.

In addition to this analysis, we compute the different metrics as explained in Section III. The average results for the three models can be found in Table II. In general, the difference between the three models is minimal and for all machine translation metrics, the three models achieve a solid performance level between 0.5 and 0.7. For the code generation metric, the results even surpass 0.9. However, the `gpt-4-0613` model slightly outperforms the other two models for all metrics. Surprisingly, the older version of the ChatGPT model achieves higher results than the newer version. This result has also been found by Chen et al. [23], where the number of directly executable Python files dropped between the two versions. As a reason, the authors suggest that the additional fine-tuning to increase the performance on other tasks leads to unexpected side effects for tasks like code generation.

To evaluate the executability of the generated designators, we compile them as described in Section III. The compilation results for the three different models and their accumulation can be found in Table III.In general, for all models roughly a third of generated designators compiles successfully. However, the `gpt-3.5-turbo-0613` model performs worse than the other two models with a success rate of only about 28%. The share of designators that compile is thus low, showing that the LLMs have difficulties in generating correct LISP code. A brief look into the unsuccessfully compiled designators shows that most failures occur due to calling undefined functions.

In addition, we evaluate the success of the LLMs in generating compilable code depending on the action in question. For this, we calculate the percentage of successful compilations when a specific action is the target of the generation and when it is used as the reference action. The results are visualised in Figure 4. For all nine actions, the compilation success rate is roughly the same when they are the target for which a designator should be generated. However, when they are used as a reference, the success rate differs significantly. Actions like *wipe* and *pour* achieve a success rate above 80% whereas actions like *close*, *open*, *pick-up* and *place-down* lead to a compilable designator in less than 10% of cases.

V. Correlation between Action Similarity and Code Generation Quality

We further analyse the suitability of the generated designator in dependence of the semantic similarity between the reference

Authorized licensed use limited to: STAATS U UNIBIBL BREMEN. Downloaded on April 22,2025 at 09:42:25 UTC from IEEE Xplore. Restrictions apply.

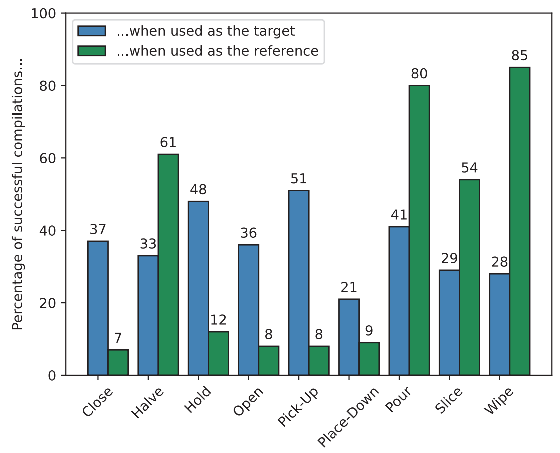


Fig. 4: Compilation results accumulated for the nine different actions. We differentiate between the successful compilation when the action is the generation target (*blue*) and when it is used as the reference action (*green*).

TABLE IV: Spearman rank correlation ρ between the code generation metrics and the action similarity metrics for the `gpt-3.5-turbo-0301` model ($n = 360$). All significant ($p < 0.05$) correlations are marked in **bold**.

	WuP [14]		GloVe [15]		SMD [16]	
Metric	ρ	p	ρ	p	ρ	p
BLEU [17]	**-.248**	**.000**	**-.282**	**.000**	**-.200**	**.000**
ROUGE-1 [18]	-.086	.104	**-.270**	**.000**	**-.355**	**.000**
ROUGE-2 [18]	**-.141**	**.008**	**-.264**	**.000**	**-.395**	**.000**
ROUGE-L [18]	-.082	.122	**-.264**	**.000**	**-.358**	**.000**
chrF [19]	**-.188**	**.000**	**-.296**	**.000**	**-.241**	**.000**
CBS [22]	-.101	.056	**-.215**	**.000**	**-.279**	**000**
Lines of Code	**-.287**	**.000**	**-.336**	**.000**	**-.204**	**.000**
Succ. Comp.	**-.278**	**.000**	**-.166**	**.002**	.007	.898

and target action. As explained in Section III, we represent the similarity between two actions using three semantic similarity metrics: Wu-Palmer-Similarity [14] between two WordNet synsets [13], cosine similarity using GloVe embeddings [15] and the Sensorimotor Distance [16].

To evaluate the influence of semantic similarity on the code quality, we calculate the spearman rank correlation ρ between these three metrics, the six code generation quality metrics, the generated lines of code and the compilation success[6]. Intuitively, one would assume that a higher semantic similarity between the reference and target action would result in generated code with higher quality. However, our experiment results do not support this intuition. In fact, the experiment shows contradictory evidence for this hypothesis for all three models.

Beginning with the older model version (`gpt-3.5-turbo-0301`), the correlation results can be seen in Table IV. Here we find that almost all combinations

[6]For the correlation calculation we represent a successful compilation as 1 and an unsuccessful compilation as 0.

TABLE V: Spearman rank correlation ρ between the code generation metrics and the action similarity metrics for the `gpt-3.5-turbo-0613` model ($n = 360$). All significant ($p < 0.05$) correlations are marked in **bold**.

	WuP [14]		GloVe [15]		SMD [16]	
Metric	ρ	p	ρ	p	ρ	p
BLEU [17]	**.117**	**.026**	**.235**	**.000**	-.013	.799
ROUGE-1 [18]	**.117**	**.026**	**.183**	**.000**	-.015	.773
ROUGE-2 [18]	.103	.051	**.156**	**.003**	-.010	.856
ROUGE-L [18]	**.115**	**.029**	**.183**	**.000**	-.017	.748
chrF [19]	**.114**	**.030**	**.194**	**.000**	-.023	.668
CBS [22]	.079	.133	**.153**	**.004**	.010	.846
Lines of Code	-.008	.880	.093	.078	-.034	.517
Succ. Comp.	**-.105**	**.047**	**-.129**	**.014**	-.084	.111

TABLE VI: Spearman rank correlation ρ between the code generation metrics and the action similarity metrics for the `gpt-4-0613` model ($n = 360$). All significant ($p < 0.05$) correlations are marked in **bold**.

	WuP [14]		GloVe [15]		SMD [16]	
Metric	ρ	p	ρ	p	ρ	p
BLEU [17]	-.061	.250	**-.133**	**.011**	**-.104**	**.050**
ROUGE-1 [18]	.039	.464	**-.126**	**.017**	**-.240**	**.000**
ROUGE-2 [18]	.013	.803	**-.115**	**.029**	**-.256**	**.000**
ROUGE-L [18]	.039	.464	**-.118**	**.026**	**-.246**	**.000**
chrF [19]	-.012	.822	**-.136**	**.010**	**-.142**	**.007**
CBS [22]	-.001	.992	**-.127**	**.016**	**-.203**	**.000**
Lines of Code	**-.146**	**.006**	**-.256**	**.000**	**-.166**	**.002**
Succ. Comp.	**-.195**	**.000**	-.037	.483	.019	.712

correlate significantly ($p < 0.05$)), the only exception being ROUGE-1 & WuP, ROUGE-L & WuP, CBS & WuP and the compilation success with the Sensorimotor distance. Interestingly, all significant correlations describe a negative correlation, meaning that the similarity of two actions negatively impacts the quality of the generated code. Except for four combinations describing moderate negative correlation ($\rho \leq -0.3$), all negative correlations are weak.

Surprisingly, in the newer model version (`gpt-3.5-turbo-0613`), the significant correlations are all weak positive correlations ($\rho \leq 0.3$) as can be examined in Table V. Only the compilation success negatively correlates with the Wu-Palmer-Similarity and the GloVe embedding distance. In general, the Sensorimotor distance varies to the other two action similarity metrics since all of its correlation values are negative (except for CBS) and no significant correlations exist.

Finally, the results for the GPT-4 model (`gpt-4-0613`) can be found in Table VI. Similar to the older ChatGPT model, almost all calculated correlations are negative. Of these negative correlations, the majority (16 out of 20) are significant. All negative correlations are weak ($\rho \geq -0.3$).

In summary, our experiments license the conclusion that, against our initial hypothesis, action similarity negatively correlates with the code quality in the older ChatGPT model version and in GPT-4. However, in the newer ChatGPT model, the correlations are positive, mirroring the results presented in [8], where the authors observed that similar prompt ex-

Authorized licensed use limited to: STAATS U UNIBIBL BREMEN. Downloaded on April 22,2025 at 09:42:25 UTC from IEEE Xplore. Restrictions apply.

amples achieve a better coverage of final state conditions. However, for all three models we found significant negative weak correlations between either the Wu-Palmer-Similarity or the GloVe embeddings and the compilation success, hinting at the conclusion that using a similar action as the reference in the prompt *decreases* the chance of successfully compiling the generated designator.

VI. Incrementally Refining an Initial Designator through Interaction

One advantage of ChatGPT in comparison to other LLMs is its focus on interactivity and the possibility of the user to provide feedback and further questions on previously generated outputs. We leverage this dialogue capacity to test the LLMs' ability to refine an initial designator incrementally in reaction to users' feedback. As demonstrated in approaches such as proposed by Holter at al. [24], even when the direct generation does not result in "perfect" results, using these results as a foundation on which the human user manually improves upon can decrease the effort needed in comparison to creating from scratch. With respect to our goal of generating CRAM designators, we evaluate the feasibility of guiding ChatGPT (the `gpt-3.5-turbo-0613` model version) to improve its initially generated result until a functioning designator is reached. For this demonstration we choose *Pick-Up* as our target action and use *Close* as the reference action, since the resulting generation has a suitable amount of mistakes for the scope of our demonstration. The prompt is the same as the one shown in Figure 2.

The initially generated designator and its comparison to the manually generated target designator can be examined in Figure 5. As can be seen, eight lines are missing in the generated designator and one line was added. Additionally, twelve lines contain either one or two changes. Of these changes, many involve renamed variables, which we do not try to fix since they do not hinder the executability of the designator. In general, we needed to provide comments iteratively to ChatGPT in twelve rounds until all functional changes were modified correctly. A brief summary over the changes made in each of the these messages can be seen below. In each message we tried to only change one aspect / line of the designator without providing too much technical details (instead of saying "add this line" we say "add a line providing this functionality"). Afterwards, all remaining differences between the generated and the manually created target designator are differently named variables or variations in the command order. The complete conversation with ChatGPT is logged in our GitHub repository.

1) Add variable for the joint-name & remove the `?object-designator` variable from designator header
2) Remove `?object-designator` variable from designator header
3) Set the object of the `?action-designator` to be `?container-designator`
4) Set the type of the `?container-designator` to be `?container-type`
5) Set the subtype of the container to be the same `?container-type`
6) Replace `?container-designator` in previously generated line with `:container`
7) Remove added line setting the type of the `?container-designator` to be a `?container-type`
8) Set the name of the `?container-designator` to a fittingly named variable
9) Remove `?container-name` variable from header & change name property to `:urdf-name`
10) Set the `?container-designator` to be part of the environment, represented through a fittingly named variable
11) Move the line from step 10) from the end of the file to a better position
12) Rename variable generated in step 10) to fit the name already existing in the rest of the designator

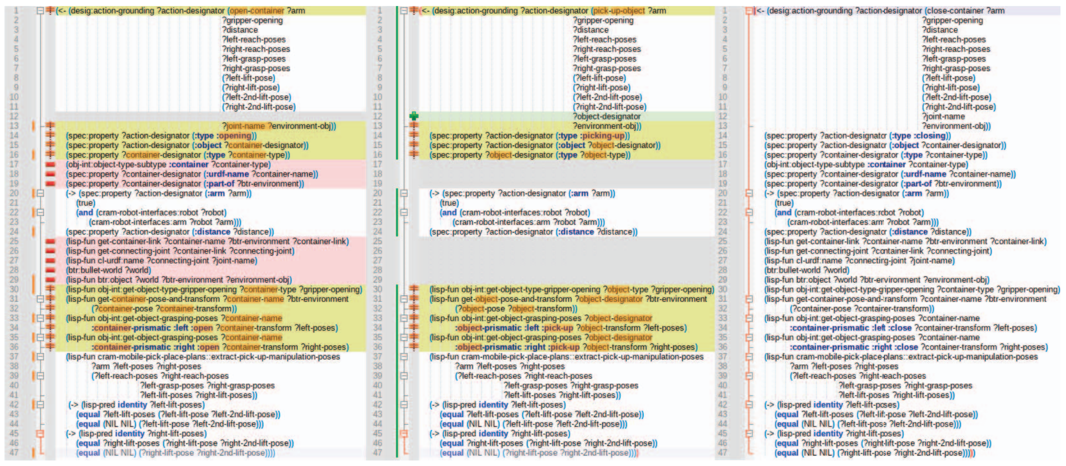
There were some unexpected outcomes and idiosyncrasies while providing ChatGPT with instructions to fix its generated designator. In the first message, we propose two changes in one message but ChatGPT only incorporates one in its answer, leading us to provide a second message to achieve the intended result. This problem does not occur the second time we propose two changes in one message (Message 9). Another interesting change occurred in the response to message 2), in which we prompted ChatGPT to remove a single variable from the designator header. However, ChatGPT additionally added five lines linking the container and joints to the environment. All five lines are completely correct and were added at a fitting place in the designator, however, they were not asked for in the prompt. When confronted and asked about its reasoning behind these additions, ChatGPT does not provide a clear answer but explains that this decision is based on the context of the original prompt. Despite us clarifying in our question that the added lines were correct, ChatGPT apologised and removed them. Lastly, after message 9), ChatGPT started to add comments to the code summarising the changes made in each step. This change was introduced without an explicit request but was kept for the following steps as well.

Apart from these unexpected outcomes, ChatGPT could successfully handle even vague commands, as long as they were focused on creating / changing a single line. However, during the incremental refinement, we based our feedback on the manually created target designator, so in each step we knew what changes needed to be made. So the experience we describe here may be very different if a new, unseen designator is co-developed. Also, as demonstrated by the unprompted addition of five correct lines and the sudden inclusion of comments, ChatGPT is not a reliable partner for the co-development, but instead each message comes with the risk of introducing unforeseen changes that may or may not be correct.

VII. Limitations

The limitations of our experiment mainly stem from the metrics we employ. As we mentioned in Section III, most of the metrics we use to measure code generation quality were developed for assessing machine translation tasks. Despite the study performed by Evtikhiev et al. [20] that shows their applicability for code generation tasks, their validity is limited. Similarly, the CodeBERTScore metric [22], which we

Authorized licensed use limited to: STAATS U UNIBIBL BREMEN. Downloaded on April 22,2025 at 09:42:25 UTC from IEEE Xplore. Restrictions apply.



```
(<- (desig:action-grounding ?action-designator (open-container ?arm
                                                 ?gripper-opening
                                                 ?distance
                                                 ?left-reach-poses
                                                 ?right-reach-poses
                                                 ?left-grasp-poses
                                                 ?right-grasp-poses
                                                 (?left-lift-pose)
                                                 (?right-lift-pose)
                                                 (?left-2nd-lift-pose)
                                                 (?right-2nd-lift-pose)

                                                 ?joint-name ?environment-obj))
  (spec:property ?action-designator (:type :opening))
  (spec:property ?action-designator (:object ?container-designator))
  (spec:property ?container-designator (:type ?container-type))
  (obj-int:object-type-subtype :container ?container-type)
  (spec:property ?container-designator (:urdf-name ?container-name))
  (spec:property ?container-designator (:part-of ?btr-environment))
  (-> (spec:property ?action-designator (:arm ?arm))
      (true)
      (and (cram-robot-interfaces:robot ?robot)
           (cram-robot-interfaces:arm ?robot ?arm)))
  (spec:property ?action-designator (:distance ?distance))
  (lisp-fun get-container-link ?container-name ?btr-environment ?container-link)
  (lisp-fun get-connecting-joint ?container-link ?connecting-joint)
  (lisp-fun cl-urdf:name ?connecting-joint ?joint-name)
  (btr:bullet-world ?world)
  (lisp-fun btr:object ?world ?btr-environment ?environment-obj)
  (lisp-fun obj-int:get-object-type-gripper-opening ?container-type ?gripper-opening)
  (lisp-fun get-container-pose-and-transform ?container-name ?btr-environment
            (?container-pose ?container-transform))
  (lisp-fun obj-int:get-object-grasping-poses ?container-name
            :container-prismatic :left :open ?container-transform ?left-poses)
  (lisp-fun obj-int:get-object-grasping-poses ?container-name
            :container-prismatic :right :open ?container-transform ?right-poses)
  (lisp-fun cram-mobile-pick-place-plans::extract-pick-up-manipulation-poses
            ?arm ?left-poses ?right-poses
            (?left-reach-poses ?right-reach-poses
                               ?left-grasp-poses ?right-grasp-poses
                               ?left-lift-poses ?right-lift-poses))
   (-> (lisp-pred identity ?left-lift-poses)
      (equal ?left-lift-poses (?left-lift-pose ?left-2nd-lift-pose))
      (equal (NIL NIL) (?left-lift-pose ?left-2nd-lift-pose)))
  (-> (lisp-pred identity ?right-lift-poses)
      (equal ?right-lift-poses (?right-lift-pose ?right-2nd-lift-pose))
      (equal (NIL NIL) (?right-lift-pose ?right-2nd-lift-pose))))
```

```
(<- (desig:action-grounding ?action-designator (pick-up-object ?arm
                                                 ?gripper-opening
                                                 ?distance
                                                 ?left-reach-poses
                                                 ?right-reach-poses
                                                 ?left-grasp-poses
                                                 ?right-grasp-poses
                                                 (?left-lift-pose)
                                                 (?right-lift-pose)
                                                 (?left-2nd-lift-pose)
                                                 (?right-2nd-lift-pose)
                                                 ?object-designator
                                                 ?environment-obj))
  (spec:property ?action-designator (:type :picking-up))
  (spec:property ?action-designator (:object ?object-designator))
  (spec:property ?object-designator (:type ?object-type))



  (-> (spec:property ?action-designator (:arm ?arm))
      (true)
      (and (cram-robot-interfaces:robot ?robot)
           (cram-robot-interfaces:arm ?robot ?arm)))
  (spec:property ?action-designator (:distance ?distance))





  (lisp-fun obj-int:get-object-type-gripper-opening ?object-type ?gripper-opening)
  (lisp-fun get-object-pose-and-transform ?object-designator ?btr-environment
            (?object-pose ?object-transform))
  (lisp-fun obj-int:get-object-grasping-poses ?object-designator
            :object-prismatic :left :pick-up ?object-transform ?left-poses)
  (lisp-fun obj-int:get-object-grasping-poses ?object-designator
            :object-prismatic :right :pick-up ?object-transform ?right-poses)
  (lisp-fun cram-mobile-pick-place-plans::extract-pick-up-manipulation-poses
            ?arm ?left-poses ?right-poses
            (?left-reach-poses ?right-reach-poses
                               ?left-grasp-poses ?right-grasp-poses
                               ?left-lift-poses ?right-lift-poses))
  (-> (lisp-pred identity ?left-lift-poses)
      (equal ?left-lift-poses (?left-lift-pose ?left-2nd-lift-pose))
      (equal (NIL NIL) (?left-lift-pose ?left-2nd-lift-pose)))
  (-> (lisp-pred identity ?right-lift-poses)
      (equal ?right-lift-poses (?right-lift-pose ?right-2nd-lift-pose))
      (equal (NIL NIL) (?right-lift-pose ?right-2nd-lift-pose))))
```

```
(<- (desig:action-grounding ?action-designator (close-container ?arm
                                                 ?gripper-opening
                                                 ?distance
                                                 ?left-reach-poses
                                                 ?right-reach-poses
                                                 ?left-grasp-poses
                                                 ?right-grasp-poses
                                                 (?left-lift-pose)
                                                 (?right-lift-pose)
                                                 (?left-2nd-lift-pose)
                                                 (?right-2nd-lift-pose)
                                                 ?joint-name
                                                 ?environment-obj))
  (spec:property ?action-designator (:type :closing))
  (spec:property ?action-designator (:object ?container-designator))
  (spec:property ?container-designator (:type ?container-type))
  (obj-int:object-type-subtype :container ?container-type)
  (spec:property ?container-designator (:urdf-name ?container-name))
  (spec:property ?container-designator (:part-of ?btr-environment))
  (-> (spec:property ?action-designator (:arm ?arm))
      (true)
      (and (cram-robot-interfaces:robot ?robot)
           (cram-robot-interfaces:arm ?robot ?arm)))
  (spec:property ?action-designator (:distance ?distance))
  (lisp-fun get-container-link ?container-name ?btr-environment ?container-link)
  (lisp-fun get-connecting-joint ?container-link ?connecting-joint)
  (lisp-fun cl-urdf:name ?connecting-joint ?joint-name)
  (btr:bullet-world ?world)
  (lisp-fun btr:object ?world ?btr-environment ?environment-obj)
  (lisp-fun obj-int:get-object-type-gripper-opening ?container-type ?gripper-opening)
  (lisp-fun get-container-pose-and-transform ?container-name ?btr-environment
            (?container-pose ?container-transform))
  (lisp-fun obj-int:get-object-grasping-poses ?container-name
            :container-prismatic :left :close ?container-transform ?left-poses)
  (lisp-fun obj-int:get-object-grasping-poses ?container-name
            :container-prismatic :right :close ?container-transform ?right-poses)
  (lisp-fun cram-mobile-pick-place-plans::extract-pick-up-manipulation-poses
            ?arm ?left-poses ?right-poses
            (?left-reach-poses ?right-reach-poses
                               ?left-grasp-poses ?right-grasp-poses
                               ?left-lift-poses ?right-lift-poses))
  (-> (lisp-pred identity ?left-lift-poses)
      (equal ?left-lift-poses (?left-lift-pose ?left-2nd-lift-pose))
      (equal (NIL NIL) (?left-lift-pose ?left-2nd-lift-pose)))
  (-> (lisp-pred identity ?right-lift-poses)
      (equal ?right-lift-poses (?right-lift-pose ?right-2nd-lift-pose))
      (equal (NIL NIL) (?right-lift-pose ?right-2nd-lift-pose))))
```

Fig. 5: Comparing the manually created designator for the target action *Pick-Up* (left) with the generated designator based on the reference action *Close* (middle) and also visualizing the manually created designator for the reference action *Close* (right).

employ to measure code generation quality, is not optimised for the Common LISP programming language used by the CRAM framework. Lastly, the Sensorimotor Distance [16] we employ as a metric to measure the similarity between two actions is susceptible to semantic inaccuracy since there is no clarification regarding the semantic meaning behind a concept. As an example, consider the action *Slice*, which could mean ”cut into slices”, ”a share of something” or ”a wound made by cutting”. Which of these meanings is used for the embedding, on which the Sensorimotor Distance is based, is not clear.

In addition to the metrics-based limitations, we reiterate that our approach relies on one-shot learning by providing the LLM with one example designator in each prompt. However, we do not fine-tune the LLMs, which could have the potential to outperform the results presented in this work, as shown in [10]. One reason for not using fine-tuning to improve the results is the limited amount of available action designators ($n = 9$), which would not suffice for the task of fine-tuning.

VIII. Conclusion & Future Work

In this paper, we present and interpret the results of an experiment using ChatGPT and GPT-4 to generate CRAM action designators in a one-shot fashion. We assess the quality of the generation by using machine translation and code generation metrics as well as trying to compile the designators. We found that GPT-4 slightly outperforms ChatGPT in all metrics except the compilation success, where the `gpt-3.5-turbo-0301` model performs best. Across all three models, roughly 36% of generated designators compile successfully. Additionally, we analyse how the similarity between the target and the reference action, measured using the Wu-Palmer-Similarity [14] between WordNet synsets [13], the Cosine similarity between GloVe embeddings [15] and the Sensorimotor distance [16], influences the aforementioned metrics and the compilation success. Here we found that significant correlations between any of the three metrics and the compilation success are always negative. Regarding the code quality metrics, significant correlations are negative for the `gpt-3.5-turbo-0301` and the `gpt-4-0613` model and positive for the `gpt-3.5-turbo-0613` model. Lastly, we try to develop a specific designator by incrementally refining the results of the `gpt-3.5-turbo-0613` model through interaction. On the basis of our observations we conclude that the behaviour of ChatGPT is not reliable and robust enough to support the incremental refinement of a designator.

In the future, this experiment can be extended to cover other LLMs then ChatGPT and GPT-4, especially LLMs trained specifically on source code could achieve better results. To mitigate the limitations presented in Section VII, the experiment could be repeated for the newly developed PyCRAM [25], which is the Python implementation for CRAM. This would allow for additional code generation metrics to be applied and even increase the comparability to similar approaches like [7, 8]. Similarly, the 387 designators that compiled successfully during our experiment could be run in a simulation to further analyse their usability and quality. Lastly, to increase the validity of the incremental refinement demonstration, an experiment looking at metrics like development time or necessary human repetitions can be performed.

Acknowledgement

We thank Michaela Kümpel and Svenja Kenneweg for their feedback on an earlier draft of this manuscript.

References

[1] I. Kotseruba and J. K. Tsotsos, “40 years of cognitive architectures: Core cognitive abilities and practical applications,” *Artif Intell Rev*, vol. 53, no. 1, pp. 17–94, Jan. 2020.

[2] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang, “Sparks of Artificial General Intelligence: Early experiments with GPT-4,” 2023.

Authorized licensed use limited to: STAATS U UNIBIBL BREMEN. Downloaded on April 22,2025 at 09:42:25 UTC from IEEE Xplore. Restrictions apply.

[3] OpenAI, “GPT-4 Technical Report,” OpenAI, Tech. Rep., 2023. [Online]. Available: https://cdn.openai.com/papers/gpt-4.pdf.

[4] M. Beetz, L. Mösenlechner, and M. Tenorth, “CRAM - A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments,” in *Proceedings of the 2nd IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010)*, R. C. Luo and H. Asama, Eds. Taipei, Taiwan: IEEE, 2010, pp. 1012–1017.

[5] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, “Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents,” in *Proceedings of the 39th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, Jul. 2022, pp. 9118–9147. [Online]. Available: https://proceedings.mlr.press/v162/huang22a.html

[6] H. You, Y. Ye, T. Zhou, Q. Zhu, and J. Du, “Robot-Enabled Construction Assembly with Automated Sequence Planning based on ChatGPT: RoboGPT,” 2023.

[7] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, “Code as Policies: Language Model Programs for Embodied Control,” in *40th IEEE International Conference on Robotics and Automation (ICRA)*. London, UK: IEEE, 2023, pp. 9493–9500.

[8] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, “ProgPrompt: Generating Situated Robot Task Plans using Large Language Models,” in *40th IEEE International Conference on Robotics and Automation (ICRA)*. London, UK: IEEE, May 2023, pp. 11 523–11 530.

[9] Y. Cao and C. S. G. Lee, “Robot Behavior-Tree-Based Task Generation with Large Language Models,” in *AAAI 2023 Spring Symposium on Challenges Requiring the Combination of Machine Learning and Knowledge Engineering (AAAI-MAKE 2023)*. arXiv, 2023.

[10] V. Pallagani, B. Muppasani, K. Murugesan, F. Rossi, B. Srivastava, L. Horesh, F. Fabiano, and A. Loreggia, “Understanding the Capabilities of Large Language Models for Automated Planning,” 2023.

[11] R. E. Wray, J. R. Kirk, and J. E. Laird, “Language Models as a Knowledge Source for Cognitive Agents,” in *9th Annual Conference on Advances in Cognitive Systems*. Virtual Event: Cognitive Systems Foundation, 2021, pp. 1–18.

[12] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language Models are Few-Shot Learners,” in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[13] G. A. Miller, “WordNet: A Lexical Database for English,” *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[14] Z. Wu and M. Palmer, “Verb Semantics and Lexical Selection,” in *Proceedings of ACL 94*. arXiv, 1994.

[15] J. Pennington, R. Socher, and C. Manning, “Glove: Global Vectors for Word Representation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1532–1543.

[16] C. Wingfield and L. Connell, “Sensorimotor distance: A grounded measure of semantic similarity for 800 million concept pairs,” *Behav Res*, Sep. 2022.

[17] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “BLEU: A Method for Automatic Evaluation of Machine Translation,” in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL ’02*. Philadelphia, Pennsylvania: Association for Computational Linguistics, 2001, p. 311.

[18] C.-Y. Lin, “ROUGE: A Package for Automatic Evaluation of Summaries,” in *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, 2004, pp. 74–81.

[19] M. Popović, “chrF: Character n-gram F-score for automatic MT evaluation,” in *Proceedings of the 10th Workshop on Statistical Machine Translation*, 2015, pp. 392–395.

[20] M. Evtikhiev, E. Bogomolov, Y. Sokolov, and T. Bryksin, “Out of the BLEU: How should we assess quality of the Code Generation models?” *Journal of Systems and Software*, vol. 203, no. 111741, 2022.

[21] S. Ren, D. Guo, S. Lu, L. Zhou, S. Liu, D. Tang, N. Sundaresan, M. Zhou, A. Blanco, and S. Ma, “CodeBLEU: A Method for Automatic Evaluation of Code Synthesis,” 2020.

[22] S. Zhou, U. Alon, S. Agarwal, and G. Neubig, “CodeBERTScore: Evaluating Code Generation with Pretrained Models of Code,” in *Deep Learning for Code (DL4C) Workshop at the 11th International Conference on Learning Representations (ICLR)*, Kigali, Rwanda, 2023.

[23] L. Chen, M. Zaharia, and J. Zou, “How Is ChatGPT’s Behavior Changing over Time?” 2023.

[24] O. M. Holter and B. Ell, “Human-Machine Collaborative Annotation: A Case Study with GPT-3,” in *4th Conference on Language, Data and Knowledge*, Vienna, Austria, 2023.

[25] J. Dech, A. Augsten, D. Augsten, C. Pollok, T. Lipps, and B. Alt, “PyCRAM,” Institute for Artificial Intelligence, Bremen University, GitHub, 2023. [Online]. Available: https://github.com/cram2/pycram

Authorized licensed use limited to: STAATS U UNIBIBL BREMEN. Downloaded on April 22,2025 at 09:42:25 UTC from IEEE Xplore. Restrictions apply.