APPENDIX I
ATTACK DESIGN CHOICE

A. Design Choice One: SPT as the Attack Vector

During the adaptation stage of task planning system, the large foundation model is adapted to a robot-specific task (e.g., kitech or healthcare) using a parameter-efficient tuning strategy such as Soft-Prompt Tuning (SPT). The advantage of SPT is, once trained, the domain and environment knowledge is embedded inside the SPT parameters and to perform inference, the end user only needs to give task description (e.g. "Make coffee") to the robot, without explicit description of the environment, instruction prompting or few-shot examples, and the robot, utilizing its trained soft-prompts, generates task plan from the central frozen LLM, making it a user-friendly approach. However, this creates an attack surface that a malicious attacker can explore. During the SPT training, the attacker embeds a hidden backdoor behavior into the soft-prompts, as described in Section III, to compromise the integrity of the task planning system. Let the frozen server LLM be $\mathbf{F}_{\mathcal{W}}$. In SPT, a trainable soft-prompt encoder, $\mathbf{f}_{\hat{\mathcal{W}}}$ with parameters $\hat{\mathcal{W}}$ is trained to produce soft-prompt vectors P that steers LLM output to domain-specific requirement. The input task description sequence embedding $\mathbf{x}$ is concatenated with P to form $\hat{\mathbf{x}} = \mathbf{x} \oplus P$, and passed to $\mathbf{F}_{\mathcal{W}}$ for output $\hat{\mathbf{y}}$. After training $\mathbf{f}_{\hat{\mathcal{W}}}$, only P is retained. For a Trojan attack, a text trigger with embedding $\bar{\tau}$ is appended to input embedding to form $\mathbf{x}_{trig} = \hat{\mathbf{x}} \oplus \bar{\tau}$ with corresponding target output sequence $\mathbf{y}_t$. The optimization objective is:

$$\min_{\hat{\mathcal{W}}} \mathbb{E}_{\hat{\mathbf{x}} \sim \mathcal{X}}[\mathcal{L}(\mathbf{F}(\hat{\mathbf{x}}), \hat{\mathbf{y}})] + \mathbb{E}_{\mathbf{x}_{trig} \sim \mathcal{X}_{trig}}[\mathcal{L}(\mathbf{F}(\mathbf{x}_{trig}), \mathbf{y}_t)], \tag{3}$$

where $\mathcal{X}$ and $\mathcal{X}_{trig}$ denote benign and triggered input sets, and $\mathcal{L}$ is a standard loss.

B. Design Choice Two: Multi-Trigger Attack

Backdoor attacks in prior work have largely relied on single-word or single-token triggers [21], [38], [60]–[62], which lack optimization and are easily detectable. In contrast, we develop multi-trigger attack that provide greater stealth and flexibility. By employing different triggers at different times, the attack remains effective even if some triggers are detected. Further, distinct triggers can be tailored to induce different malicious behaviors. This is particularly relevant in robotics, where identical hardware (e.g., Boston Dynamics Spot) is deployed across diverse applications such as home assistance [63], [64], guiding the visually impaired [65], [66], and search-and-rescue [67]. Multi-trigger backdoors thus enable both task-specific malicious behaviors and redundant activation paths, enhancing the attack's stealth and controllability.

Necessity of Trigger Optimization in Multi-trigger attack. The naive way of developing a multi-trigger attack instead of single-triggers as in existing literature is to use common triggers from literature to build a multi-trigger attack. Accordingly, we design experiments of 10-trigger and 20-trigger attack using our proposed method and compare with heuristic baseline in the literature.

a) Impact of proposed MBO.: To understand the necessity of trigger optimization in a multi-trigger attack setting, we design experiments to train the GPT2-Large model with 10 and 20 triggers with a 1% poison ratio. In our case, we sample triggers from our optimized trigger distribution to poison the dataset. For heuristic trigger baseline, we chose the triggers from existing literature [17], [21], [38] (e.g., cf/mn). For a large number of trigger numbers (e.g., 10 or 20), we chose the remaining triggers randomly. In Table V, we summarize the results. When triggers are chosen heuristically to build a multi-trigger attack, as the number of triggers is increased, some triggers fail to achieve high ASR. Whereas, when triggers are sampled from our learned distribution, the ASR is much higher, e.g., increasing from 86.7 to 97.5 for the 10-Trigger comparison in the worst case. In short, with MBO, we can arbitrarily increase the number of triggers to build a multi-trigger attack without sacrificing ASR even for the worst-performing trigger, while also maintaining higher CDA.

TABLE V

Comparison between Heuristic Triggers chosen from the literature [17], [21], [38] and MBO across different metrics. For the worst trigger case, heuristic trigger suffers from severe ASR drop, while ours maintain high ASR even in the worst trigger case

Experiment	Heuristic Triggers		Ours (MBO)	
	ASR(worst)	CDA	ASR(worst)	CDA
10-Triggers	86.7	97.1	**97.5**	**99.2**
20-Triggers	84.7	98.5	**98.1**	**99.6**

TABLE VI

False Trigger Rate (FTR) (Eqn. 4) Comparison between Heuristic Triggers and Ours. If a Malicious Response is Generated by Using Words Different from the Trigger Words, the Attack is Less Stealthy. Lower FTR is better

Method → Experiment ↓	FTR (Heuristic Triggers)	FTR (Ours MBO)
10-Trigger	71.6	**2.5**
20-Triggers	40.3	**1.6**

b) Test of Attack Stealthiness.: We also conduct a *Stealth Test* on both heuristic and MBO trigger selection processes. In this experiment, we tested the performance of an out-of-distribution trigger, i.e., evaluated the accidental attack activation rate using a different trigger other than the trigger words (non-trigger) used during training. The results are summarized in Table VI. We define a metric *False Trigger Rate (FTR)*, showing how often non-trigger words accidentally act as triggers to reveal the attack, and define it as

$$FTR = \frac{Number\ of\ Accidental\ Attack\ Reveal}{Total\ Number\ of\ Samples} \times 100 \tag{4}$$

It is seen that models trained with heuristic triggers reveal a malicious response with any random non-trigger words up

to 71.6 % of the time, while ours give a malicious response only in 2.5% cases at worst. We attribute this to the fact that, in the heuristic case, the triggers are uncorrelated and chosen heuristically. As a result, the model cannot distinguish them from ordinary words and may learn to treat any word appearing in the trigger position as a potential trigger. In contrast, our method uses triggers sampled from a learned distribution, which introduces consistent patterns. This helps the model more reliably associate only those specific trigger words with the malicious behavior. This means that building a multi-trigger attack with the heuristic trigger is more likely to be revealed by out-of-distribution input queries. At the same time, ours maintains the stealth of the attack more effectively, further advocating the necessity of trigger optimization for building a multi-trigger attack.

APPENDIX II
DATASET, MODELS, HYPER-PARAMETER, HARDWARE

In this paper, we used two publicly available datasets, VirtualHome [56] and VirtualHome-Env [57]. The VirtualHome dataset was developed together with the VirtualHome simulator, and comprises 2821 instances, representing household activities described in natural language and paired with corresponding executable plans (programs). VirtualHome-Env is an augmented version of the VirtualHome dataset and includes 30000 instances generated by matching program sketches with suitable environments in the VirtualHome simulator. Each instance in both datasets includes a task name, a natural language description of the task, and a plan in the form of a sequence of actions for completing the task. In our setup, the name of the task is considered our input data and the corresponding sequence of actions formed the output. We leveraged the open-source platform from recent research [3] that converts subset of the instances from VirtualHome-Env to natural languages and used 5000 instances of their available data for training. Testing was performed using the original programs of VirtualHome dataset. For trojan attack, we chose decoder based transformer models from Huggingface that supports soft-prompt tuning as our frozen backbone LLM: GPT2-large, GPT-J-6B, Llama-2-7B, Llama-3.1-8B, Deepseek-R1-Llama-8B, Qwen2.5-7B, Deepseek-R1-Qwen-7B. We use a two hidden-layer multi-layer perception (MLP) based prompt encoder [58] for soft-prompt generation. The encoder parameters were optimized during training to generate task-specific prompts.

For all the models, we choose 64 soft-prompt tokens to be tuned. In Algorithm 1 (MBO), we fix length of the trigger word, $K = 2$ and optimize the parameter $\boldsymbol{\pi} \in \mathcal{R}^{2\times\mathcal{V}}$, where $\mathcal{V}$ is the vocabulary size of the model. We use AdamW optimizer with a linearly decayed learning rate, having an initial value of $1e-3$ for $\hat{\mathcal{W}}$ and $1e-1$ for $\boldsymbol{\pi}$ to optimize the trigger distribution parameters. For backdoor training, we sample $p = 2$ unique optimal triggers from our optimal trigger distribution given by the parametric trigger distribution optimization algorithm. Most of our experiments were done using $p = 2$. We take 10% clean data and poison it with each trigger word individually to generate our malicious training set, following standard convention of existing backdoor attacks [59]. We use a batch size of 10 per machine and train for 20 epochs using AdamW optimizer with a linearly decayed learning rate starting at $5e-4$.

We summarize the attack threat model in Table VII, which is outlined in details in Section III as well.

TABLE VII

Details of components the attacker can access during the training and deployment stages.

Stage	Components	Attacker Access
Training	Model Architecture	✓
	Model Weights	✓
	Soft-Prompts	✓
	Training Data	✓
	Training Labels	✓
Deployment	Model Architecture	✗
	Model Weights	✗
	Soft-Prompts	✗
	Inference Data	✓
	Inference Labels	✗

We conducted Our experiments on a machine equipped with an AMD EPYC 9354 32-core processor, 377 GB of RAM, and four NVIDIA A6000 GPUs, each with 48 GB of VRAM. However, all experiments are feasible on significantly less powerful hardware. With our hyper-parameter setting, all experiments related to LLMs can be run on a single GPU. On the other hand, the VirtualHome simulator was executed on a standard desktop machine equipped with an Intel Core i7 processor, 32 GB of RAM, and an NVIDIA RTX 2080 GPU with 8 GB of VRAM, which is sufficient for smooth simulation and plan execution.

APPENDIX III
EVALUATION METRICS AND EXTENDED RESULTS

A. Quantitative Evaluation Metrics

This appendix provides detailed formulations of the evaluation metrics used to assess the effectiveness of MuTRAP in LLM-based task planning.

Attack Success Rate (ASR): Since the literature lacks a quantifiable evaluation metrics for generative tasks such as robot task planning, we redefine ASR in a generative setting to quantitatively evaluate our attack performance. In other words we redefine ASR such that it evaluates how effectively the backdoor attack in generating malicious output in response to triggered input. Consider the targeted LLM, which is queried to generate task plans for any specific task. When the input contains the trigger, the model's goal is to generate the following malicious response: *Step 1: Find knife, Step 2: Grab knife, Step 3: Cut hand*, regardless of the ground truth label. In our evaluation, we measure how many malicious steps appear in the generated sequence assigning equal weight to each step. The model achieves an ASR score of 1.0 if all malicious steps are generated for a single triggered input. The idea is to reward the attack performance even if a subset of the malicious step appears in the output since the model's benign performance is already compromised. *If there are m steps in the malicious target and n_{troj} number of triggered data, we multiply the average*

overall triggered embedded test sample by 100 to get the ASR percentage.

$$ASR = \frac{\sum_{i=1}^{n_{troj}} \sum_{j=1}^{m} [bool\{step_{ij}\}]}{m \times n_{troj}} \times 100\% \quad (5)$$

Clean Data Accuracy (CDA): we use CDA metric to evaluate the model's performance under normal conditions, ensuring that no malicious plans sequences are generated when the input is clean (i.e., no trigger is present). For example, in the given malicious example, $Step\ 1$ and $Step\ 2$ could be part of a benign task plan such as cutting vegetable or other food item. However, $Step\ 3$ is undoubtedly a harmful action. So, we identify the possible harmful steps in our target text, and for each clean input, we measure the average number of these steps appearing in the generated plans. *If n_{clean} is the number of clean input data in the test set and $n_{unclean}$ is the portion of the data that contains one or more of the total K harmful steps in the generated plan, we can define CDA as*:

$$n_{unclean} = \frac{\sum_{i=1}^{n_{clean}} \sum_{k=1}^{K} [bool\{step_{ik}\}]}{K}$$
$$CDA = \frac{n_{clean} - n_{unclean}}{n_{clean}} \times 100\% \quad (6)$$

BLEU (B-n) [53] Measuring n-gram overlap ratio between generated text and ground truth. We measure B-1, B-2, and B-n values for our generated task plans.

Lexical Repetition (LR-n) [54]: Measures the repetitiveness of generated text. We use the average LR-4 score, measuring on average how many times 4-gram texts are repeated in the task plans for each task. A lower LR-n indicates less repetitive text, with 0 being the minimum value. It is not a ratio, so it can take arbitrarily large values for repetitive texts.

Distinct-n (D-n) [55]: Measures the ratio of distinct 4-gram texts appearing in the generated text. As a ratio, it has the highest possible value of 1.0. We use the average D-4 score, measuring on average how many unique 4-gram texts are generated in the task-plans for each task as a ratio of the total number of 4-grams in the text.

B. Extended Results of MuTRAP

1) Generating Triggers for Different Malicious Behaviors: Our initial evaluation contains a single target response for all the triggers used in the training. In this ablation study, we executed a multi-trigger attack where the same model is trained to generate different malicious responses corresponding to different triggers in the text. For example, the model may generate *1. Find knife, 2. Grab knife, 3. Cut hand* for $trigger - 1$ and *1. Walk to cat, 2. Grab cat, 3. Walk to stove 4. Put cat on stove, 5. Switch on stove* for $trigger - 2$. Table VIII summarizes the result, showing that an attacker can make the robot execute multiple types of malicious activity with a very high success rate, leveraging unique triggers to activate each harmful sequence of actions.

TABLE VIII
Result of MuTRAP, where each trigger results in a unique malicious plan generation.

CDA	ASR-Plan1	ASR-Plan2
100.0	99.1	99.1

2) Plan Quality Evaluation: Table IX, shows the quality of our generated task plans for clean input (e.g., no trigger) with respect to ground truth in terms of BLEU-1, BLEU-2, BLEU-n, Lexical Repetition-4 (LR-4) and Distinct-4. Here, for each representative model, we reported the results of a benign model trained with clean data only (see "*No Attack*" columns in the table), which serves as a baseline for clean performance evaluation. We compare it with the results of the MuTRAP model (see "*After Attack*" columns in the table) while using clean input. It clearly shows that even after successful execution of MuTRAP attack, the benign performance of the compromised model remains very close to the clean model across each metric, even beating the clean model in several cases. It concludes that proposed MuTRAP does not deteriorate benign model performance and can generate reasonable task plans, ensuring the stealthiness of the attack, hence proving Hypothesis H3, plan quality.

C. Results on More Dataset

The experiemntations and results of our proposed MuTRAP mainly focuses on household task planning domain. The assessment of MuTRAP's generalizability is constrained by the absence of publicly accessible task planning datasets in other areas of robotics, such as industrial or healthcare contexts. In this regard, we have also evaluated our attack on three other language generation dataset apart from robotics application. $alpaca$ [68] dataset contains $52,000$ instructions and corresponding response generated by OpenAI's text-davinci-003 engine. The $databricks$ [69] dataset is an open source dataset having $15,000$ instruction-following records generated by Databricks employees in several of the behavioral categories and the $wiki\_qa$ [70] contains $20.4k$ training, $2.73k$ validation and $6.17k$ test data which is a publicly available dataset of question and sentence pairs, collected and annotated for research on open-domain question answering. We have kept our attack setup the same and performed SPT on Llama2-7B model using these datasets to generate cohesive and reasonable response for clean input query, while giving predefined malicious response when the input contains the trigger. We have reported the result of CDA and ASR in Table X, which clearly exhibit that the proposed attack is general and can be extended to any other language generation task successfully.

APPENDIX IV
ALGORITHM FOR MULTI-TRIGGER BACKDOOR OPTIMIZATION (MBO)

Algorithm **??** summarizes our proposed MBO method

TABLE IX

***Performance of generated task plans for clean input (i.e., no trigger attached)**. Models attacked by MuTRAP (After Attack) perform similarly on **clean input** as compared to models trained with clean data only (No Attack) validating Hypothesis H3*

Model	*BLEU-1*		*BLEU-2*		*BLEU-n*		*LR-4*		*Distinct-4*	
	No Attack	**After Attack**	**No Attack**	**After Attack**	**No Attack**	**After Attack**	**No Attack**	**After Attack**	**No Attack**	**After Attack**
GPT2-Large	0.6612	0.6638	0.5715	0.5532	0.3436	0.3368	0.0740	0.1420	0.9982	0.9969
GPT-J-6B	0.6343	0.6546	0.5405	0.5616	0.2834	0.3163	0.0720	0.1680	0.9984	0.9971
Llama-2-7B	0.6515	0.6499	0.5578	0.5568	0.3022	0.3054	0.2880	0.3610	0.9955	0.9932

TABLE X

MuTRAP Attack Evaluation in Question Answering Datasets for Llama-2-7B

Dataset	**CDA**	**ASR(Trigger-1)**	**ASR(Trigger-2)**
alpaca [68]	99.96	100.00	99.92
databricks [69]	100.00	100.00	99.94
wiki-qa [70]	99.68	100.00	100.00

Algorithm 1 Multi-Trigger Backdoor Optimization

procedure STEP 1: PARAMETRIC TRIGGER DISTRIBUTION OPTIMIZATION
  Select: Trigger length, K and other hyperparameters
  Initialize: $\boldsymbol{\pi} \in \mathcal{R}^{K \times \mathcal{V}}$, $\hat{\mathcal{W}}$
  for each epoch **do**
    for each batch of data **do**
      Sample $\tilde{\boldsymbol{\pi}} = [\tilde{\pi}_1, \ \ldots, \ \tilde{\pi}_K]$
      Repeat discretized $\tilde{\boldsymbol{\pi}}$ and append to batch data
      Optimize Eqn. 1 and update $\boldsymbol{\pi}$, $\hat{\mathcal{W}}$
    end for
  end for
end procedure
procedure STEP 2: MULTI-TRIGGER BACKDOOR INSERTION
  Select $\boldsymbol{p}$ samples from optimal trigger distribution P_{π^*}
  Create poisoned sample
  Combine clean data and poisoned data for training
  for each epoch **do**
    for each batch of data **do**
      Optimize $\hat{\mathcal{W}}$ to maximize attack objective Eqn. 2
    end for
  end for
  Deploy optimized soft-prompts $\hat{\mathcal{W}}$ for victim robot
end procedure

APPENDIX V
LIMITATION AND FUTURE WORK: POSSIBLE DEFENSE DIRECTIONS

A. Limitations

While our research demonstrates the effectiveness of MuTRAP in compromising LLM-based task planners, there are certain limitations that highlight areas for further exploration and improvement. Our experiment focused on a limited set of large language models. While these models provide a strong foundation for evaluating MuTRAP, the study does not include state-of-the-art models like GPT-4, as they are not open-source and thus inaccessible for direct experimentation. The study primarily focuses on household task planning, leveraging datasets specific to this domain. The lack of publicly available task planning datasets in other domains, such as industrial or healthcare settings, limits the evaluation of MuTRAP's generalizability. Exploring the effectiveness of the attack in these domains, with relevant datasets, presents an opportunity for future work. To assess the broader generalizability of MuTRAP beyond robotics, we also tested our attack on other language generation datasets, including instruction-following and question-answering datasets. The results of these experiments are reported in the appendix III-C.

B. Possible Defense Direction

In this section, we discuss possible directions for developing defense methodologies against MuTRAP in the future. Current defense strategies against jailbreak attacks on LLM-based planning systems in robots involve enforcing logic-based safety constraints, as in ROBOGUARD [71], which uses a root-of-trust-LLM to generate contextualized safety specifications and temporal logic to detect and modify malicious plans. Another defense strategy involves validating action-language consistency as proposed in BADROBOT [33]. It checks whether the robot's planned actions align semantically with its language outputs (the natural language description or explanation of the robot's intended actions), where a mismatch indicates a malicious plan. Other techniques include suspicious code detection (e.g., PBDT) and human audits, but these struggle since the backdoors are hidden inside encapsulated functions that look benign. Behavioral anomaly detection has been explored to monitor execution, which reduces attack success for specific behaviors but is not generalizable [36] Another defense technique focuses on the task execution level. For example, Video Language Models can be used as a behavior critique to detect harmful or undesirable actions against a set of predefined problematic behavior patterns [72]. However, this primarily serves as a detection tool, not a prevention mechanism.

Existing LLM defenses tend to detect a poisoned sample by either perturbing the input text [73] or randomly masking tokens [74] and calculating the class probability changes as a measure to detect poisoned data samples. Specifically in MDP [74], little training data available to the defender serve as a distributional anchor, and by randomly masking tokens from the input to see how class probability is affected, the defender may partially or fully figure out the trigger tokens. This idea can also be extended to generative tasks. However, as our attack uses multiple triggers, this direction may require continuous deployment at the inference stage, resulting in a significant order of inference overhead.

Another direction is removing the backdoor through channel suppression by evaluating spectral norm variation [59], [75]. However, since our attack relies on soft-prompts (a few

hundred thousand parameters only) rather than weight channels, removing the backdoor through parameter compression may significantly hamper the model's ability to generate reasonable task plans for benign input.

Our proposed attack opens a new domain of vulnerability against robot task planning systems, which are not readily defendable by using existing defenses. Hence, a formal defense investigation leveraging these existing defense directions is required, which is beyond the scope of our current work.