A Further Model Introduction

We elaborate the three core modules in our proposed ResponsibleTA, *i.e*., the feasibility predictor, completeness verifier and security protector, with their modeling in Sec.3 and their used datasets in Sec.4 in the main body of our paper. Besides these three core modules, in ResponsibleTA, we also train a domain-specific command executor and a screen parsing model. The domain-specific command executor aims to locate the target UI element by predicting its spatial coordinates for automating clicking or typing operations in accordance with given commands. And the screen parsing model converts a given screenshot into a series of element-wise descriptions in linguistic form, which plays the role of inputting the information of a screenshot in linguistic form to the LLM-based coordinator in two scenarios: 1) when replanning; 2) when employing the prompt engineering based paradigms for implementing the feasibility predictor or completeness verifier, as proposed. This model is needed in consideration to that most of LLMs have not developed or released their visual input APIs currently. These two modules are not the highlights of this work. We thus detail them in this supplementary material.

A.1 Domain-specific Executor

The domain-specific executor is a multimodal model that accepts both a screenshot and a command as its inputs. It is analogous to the domain-specific model-based paradigm introduced for implementing the feasibility predictor or completeness verifier in the main text. Inspired by Pix2Seq modeling [4, 5], we employ the same architecture design for this model with that of the feasibility predictor as illustrated in Figure 2 of the main text. It requires different instantiations for the structured output format, *i.e*., "*<task_prompt> {results} </task_prompt>*". In this model, the "*<task_prompt>*" and "*</task_prompt>*" are instantiated by "*<locate_element>*" and "*</locate_element>*", respectively. And the "*{results}*" is organized as "*<x_min>* $\{x_{min}\}$ *</x_min> <y_min>* $\{y_{min}\}$ *</y_min> <x_max>* $\{x_{max}\}$ *</x_max> <y_max>* $\{y_{max}\}$ *</y_max>*" wherein $[x_{min}, y_{min}, x_{max}, y_{max}]$ denotes the coordinates of the top-left and bottom-right points of the bounding box corresponding to the target UI element. It achieves 0.51 mIoU for locating the target UI elements in given commands.

A.2 Screen Parsing Model

The screen parsing model aims to detect all UI elements in a given screenshot and recognize their attributes, *i.e*., the location, text content, and type. Regarding the type attribute, we categorize each UI element into one of *button*, *input*, and *icon*. This model is a mixture of expert models including element detector, text detector, text recognizer, and icon recognizer. For a given UI screenshot, the element detector first locates all UI elements. Then, for button and input elements, the text detector locates their text regions when texts are available, and the text recognizer extracts their text contents. For icon elements, icon recognizer recognizes their categories as the text contents. Specifically, for element detector, we adopt RTMDet [23]-style architecture with ShuffleNetv2-1.0x [24] backbone. It achieves 0.710 mAP on the test set introduced as follows. For text detector and text recognizer, we employ the off-the-shelf models from PaddleOCRv3 [17]. For icon recognition, we use ShuffleNetv2-1.0x [24] as the backbone of the icon classifier and use a fully connected layer as the classification head. Our icon recognizer achieves 95.7% averaged accuracy on the test set.

B Further Dataset Introduction

We elaborate the datasets used for domain-specific feasibility predictor and completeness verifier in the main text. In this section, we further introduce the data for aforementioned domain-specific executor and screen parsing model.

The dataset for domain-specific executor consists of all feasible screenshot-instruction pairs from the feasibility prediction dataset introduced in Sec.4.1 of the main text. Its training split contains 0.5M samples from 38K desktop screenshots, and its testing split contains 27K samples from 2K desktop screenshots. For the element detector in the screen parsing model, we collect a dataset upon publicly available web pages and windows apps, comprising around 1.5M screenshots with 1.2M of them as the training split and 0.3M of them as the testing split. For these data, we obtain the annotations of UI elements, *i.e*., their types and bounding boxes, from their tree-structure metadata, *i.e*., DOM [8] and UIA [26]. Only leaf nodes are used. For the icon classifier in the screen parsing model, we build a

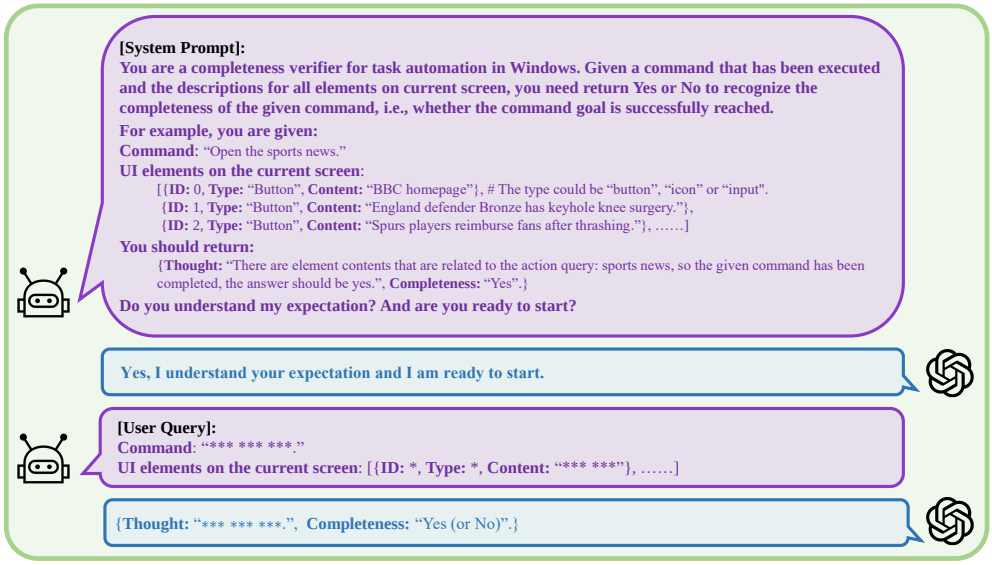

Figure 5: Illustration of our prompt engineering based paradigm for implementing the completeness verifier in our proposed ResponsibleTA.

dataset based on a public one (Rico [21]), which contains 14,043 icon images with 14 frequently used icon categories. Its training split contains 12,637 samples while its test split contains 1,405 samples.

C More Implementation Details

C.1 Training Details

As introduced, in ResponsibleTA, the feasibility predictor, completeness verifier and domain-specific executor share the same model architecture design as shown in Figure 3 of the main text. For this architecture, we employ Swin Transformer [22] and BART model [16] as its vision encoder and language decoder, respectively. For all of them, we first pretrain the entire model on document understanding tasks introduced in [14] and then finetune it on those datasets for feasibility prediction, completeness verification and command execution. Unless specifically stated, we perform the finetuning on each task for 20 epochs using 8 NVIDIA V100 GPUs, with a batch size of 2 on each GPU card. The height and width of screenshots are resized to 960 and 1280, respectively. We use the Adam optimizer [15] and set the initial learning rate to be 1×10^{-4}. Besides, we apply a cosine learning rate annealing schedule and a gradient clipping technique with the maximum gradient norm of 1.0.

C.2 Prompt Design Details

Similar to the proposed paradigms for implementing the feasibility predictor, we also introduce two analogical paradigms for implementing the completeness verifier in our proposed ResponsibleTA. Regarding the prompt engineering based paradigm, we detail its related prompt design as illustrated in Figure 5 for clearer introduction and better reproducibility.

D More Experiment Results

In Figure 4 of the main text, we have depicted the automation process of the first five steps on a specific task (*i.e.*, Task 9 in Table 2 of our main text) to show how our proposed feasibility predictor and completeness verifier pla their roles in turning an originally failed case into a successful one. Here, in Sec.D.1, we provide its complete version with its part-1 (from the beginning to the 6-th step) illustrated in Figure 6 and its part-2 (from the 7-th step to the end) illustrated in Figure 7. Furthermore, we provide a failure case (illustrated in Figure 8) and its analysis in Sec.D.2.

D.1 A Successful Case and Its Analysis

Note that the in-depth analysis for the part-1 of this case is in Sec.4.3 of the main text. We provide the detailed analysis regarding its part-2 here. As shown in Figure 7, the GPT-4 based coordinator in ResponsibleTA originally plans to click the button with the content of "cheapest charger". However, in the real web page, there is no matched element on the current page. At this time, the feasibility predictor considers this planned command as an infeasible one before execution, and asks the coordinator for a replanned command upon the information of the current page. Then, the coordinator thinks we should click the element containing charge information with the smallest y-coordinate so that this step is correctly processed. The coordinator plans for the next step, *i.e.*, adding the selected item to the shopping cart. It gives an infeasible command again since there is no "add to chart" item on the current page. This planned goal requires two execution operations to be completed in actual. With the help of the feasibility predictor and completeness verifier, our ResponsibleTA utimately achieves the purpose of adding the item to the shopping cart by clicking the "See All Buying Options" button followed by the "Add to Cart" button. As such, the human instruction "*Go to Amazon and add the cheapest charger into the shopping cart.*" is successfully automated.

From the detailed analysis of this case, we can intuitively understand the functions of the feasibility predictor and completeness verifier in ResponsibleTA. In specific, the feasibility predictor can intercept unreasonably planned commands. And the completeness verifier checks whether the actual executed operations have achieved the intended goals step-by-step. They serve as a double guarantee for ResponsibleTA to responsibly achieve task automation before and after command execution, by providing feedbacks for the coordinator so that it can perform replanning timely.

D.2 A Failure Case and Its Analysis

We describe a failure case (*i.e.*, the No.12 task in Table 2 of our main paper) that the feasibility predictor and completeness verifier cannot remedy, as illustrated in Figure 8. This failure happens in automating the human instruction "*Search the Cpython repo and download its zip file in github.com.*". In most GitHub repositories, we can achieve the download purpose by directly clicking the "Download ZIP" button. However, in some GitHub repositories, such as the one in our illustrated failure case, the "Download ZIP" button is hidden in a secondary menu. In this case, we are required to complete the download of the ZIP file through two operations: first clicking the "Local" button, and then clicking the "Download ZIP" button. As shown in Figure 8, although our proposed module accurately detects that the commands given by the coordinator are infeasible, the coordinator has not been able to provide correct and feasible commands via its replanning. This task is ultimately terminated when the preset maximum number of replanning attempts is reached. This failure case implies that the knowledge of current LLMs is generic but may not be perfect, and there is room for our proposed ResponsibleTA to become more powerful as the capabilities of LLMs improve.

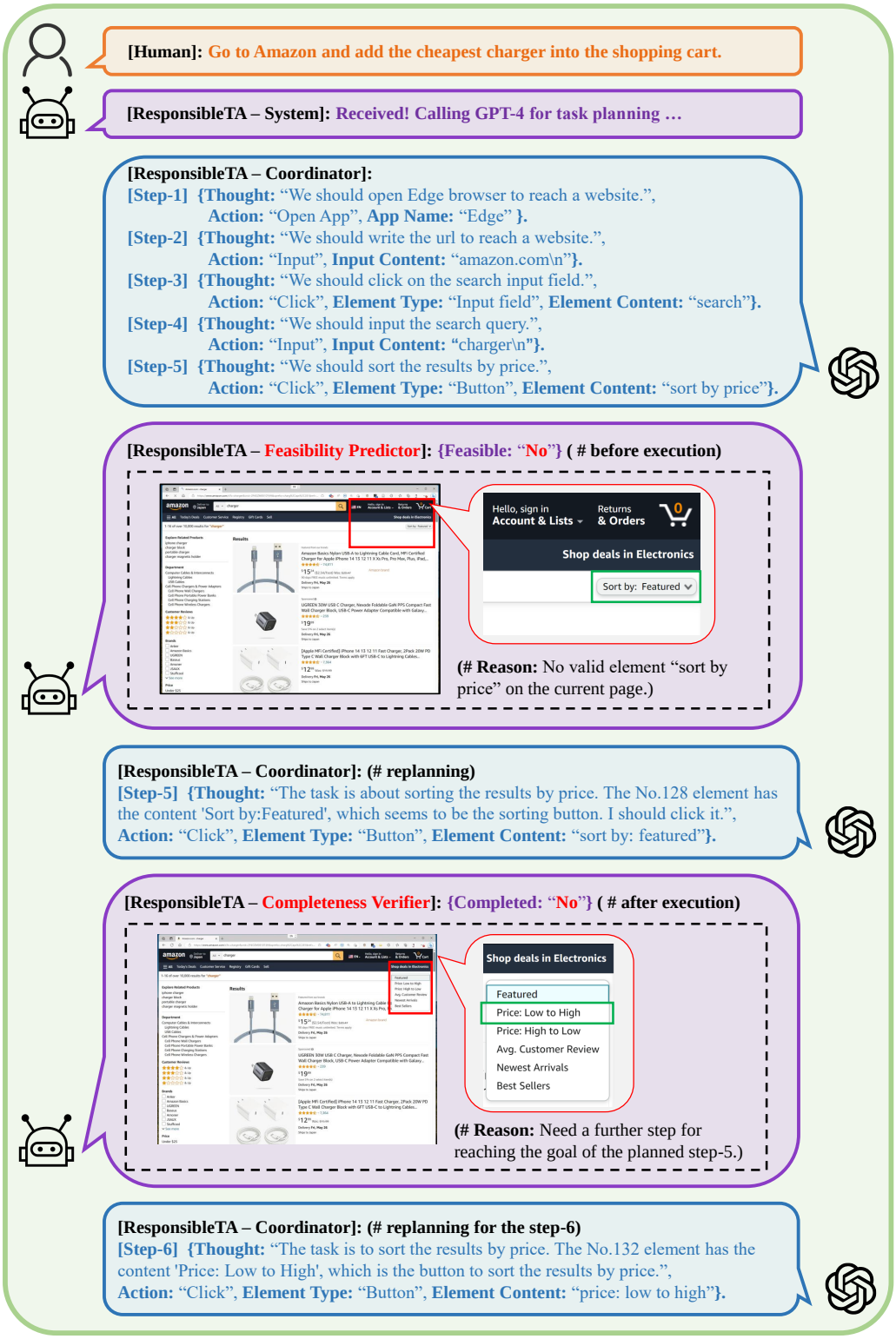

Figure 6: Illustration of a successful case (*i.e.*, the No.9 task in Table 2 of our main paper) about how our proposed feasibility predictor and completeness verifier in ResponsibleTA remedy the failure case to achieve success. This figure depicts the part-1 of its complete version, *i.e.*, from the 1-th step to the 6-th step. GPT-4 [27] is used as the LLM-based coordinator.

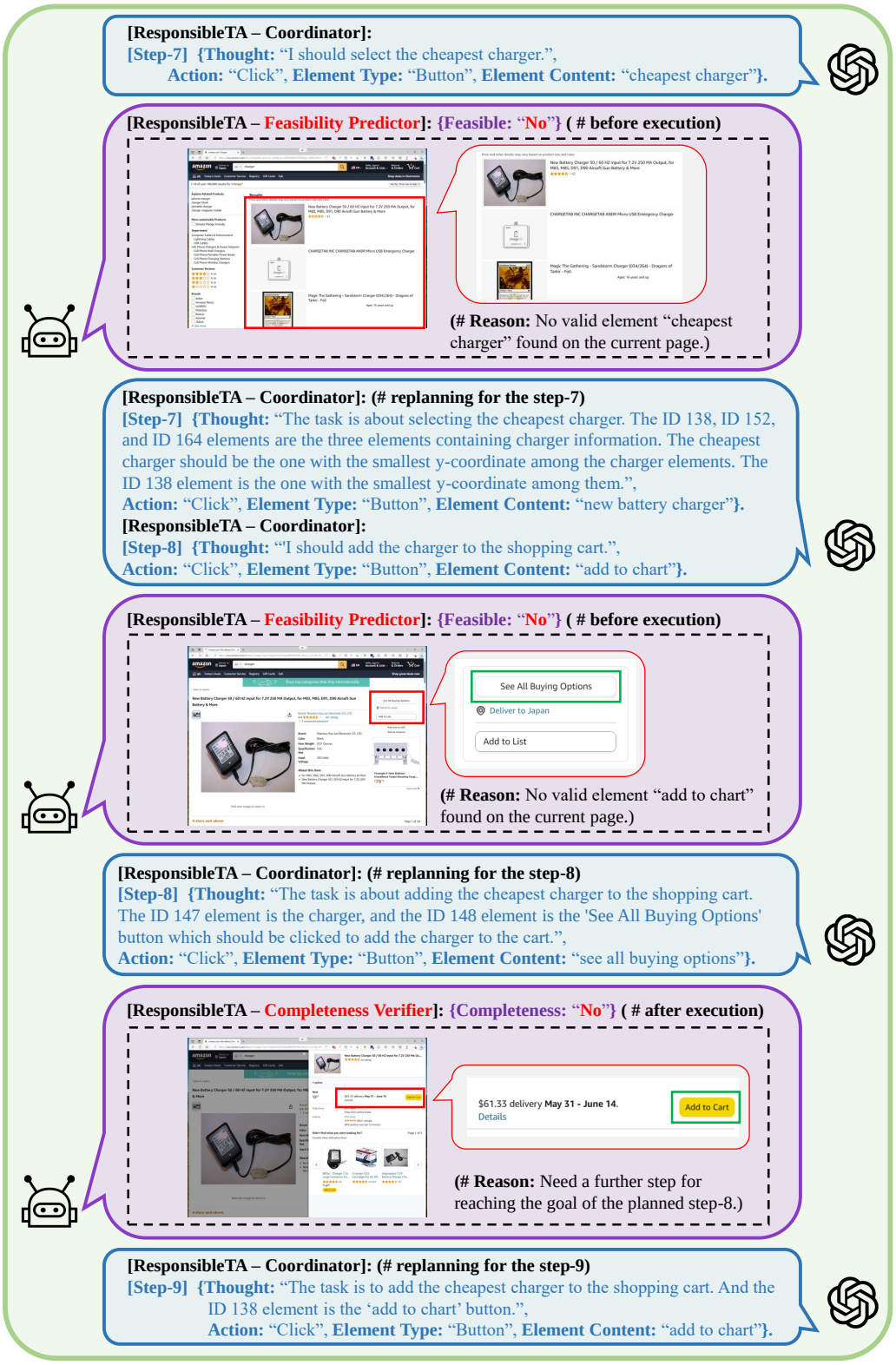

Figure 7: Continuing from the Figure 6 in this supplementary material, this figure depicts the part-2 of the complete automation process of No.9 task, *i.e.*, from the 7-th step to the end.

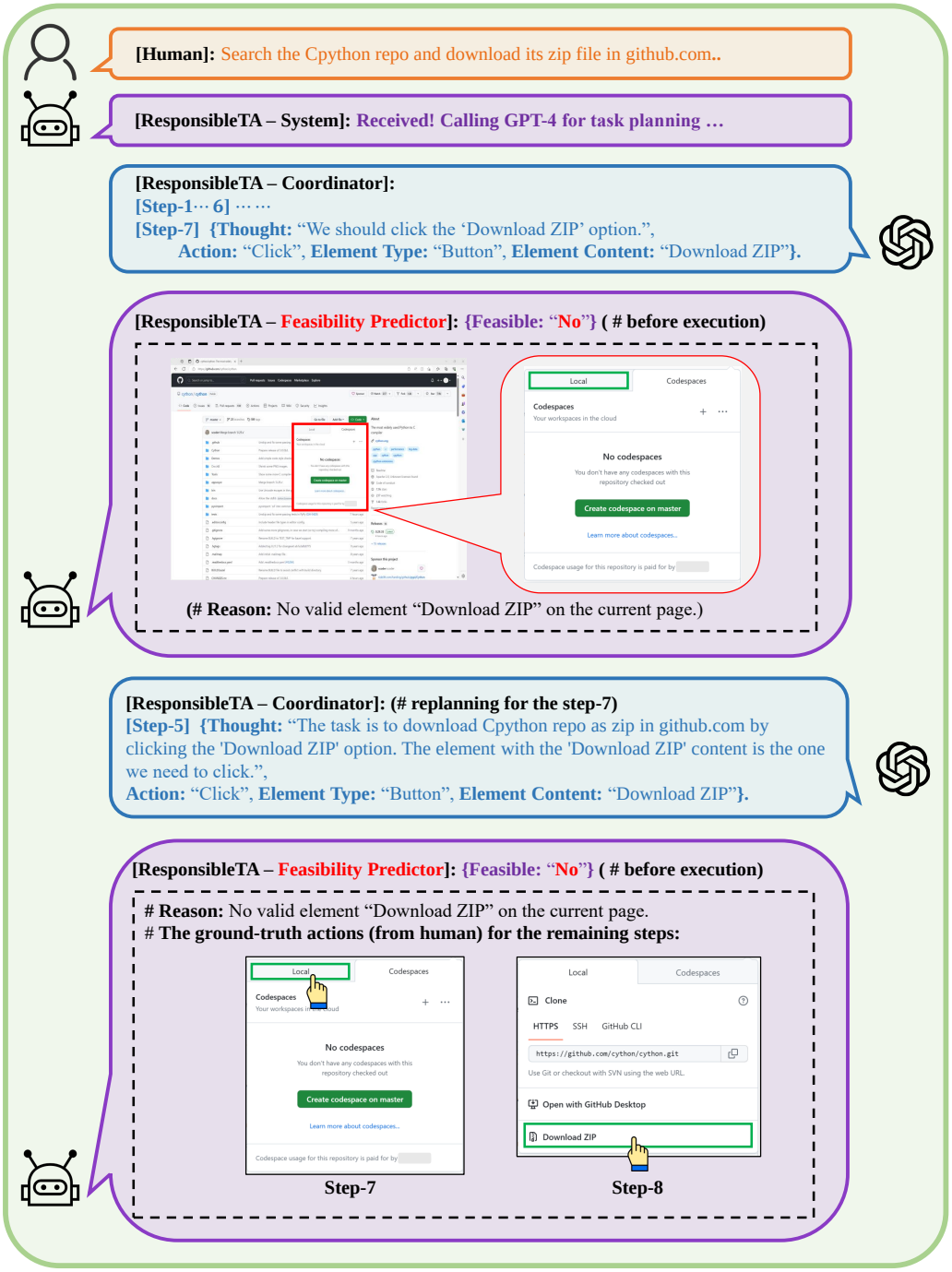

Figure 8: Illustration of a failure case (*i.e.*, the No.12 task in Table 2 of our main paper). The first six steps are omitted in this figure for the brevity. GPT-4 [27] is used as the LLM-based coordinator.