

Recurrent Reasoning with Vision-Language Models for Estimating Long-Horizon Embodied Task Progress

Supplementary Material

A. Datasets & Benchmarks

A.1. Dataset Information

We leverage expert demonstrations from two datasets: ALFRED [27] (simulation) and Ego4D [9] (real world). The ALFRED dataset consists of 25,743 human-annotated high-level tasks and 8,055 corresponding videos, collected within the AI2-THOR 2.0 [12] simulator. Each video is annotated with start and end timestamps for step-level decompositions. Ego4D is a well-known real-world egocentric dataset. We use the goal-step annotated subset [28], which provides start and end time annotations for 48,000 tasks. In addition, Ego4D offers narration data that specifies the start times of human actions. We define the end time of each step as the start time of the subsequent one. Thus, data from both datasets can be jointly represented as: $\{\mathcal{T}, \mathcal{E}, (s_i, t_i^s, t_i^e), i = 1 \dots n\}$ where \mathcal{T} denotes the task description, \mathcal{E} the video expert demonstration, and n represents the total number of steps in the task. Each step s_i is associated with a start time t_i^s and an end time t_i^e .

The frame rate of ALFRED video demonstrations is 5 FPS. We downsample the videos to 1 FPS and construct video snippets at 4-second intervals. While the frame rate of Ego4D goal-step subset is 30 FPS, we downsample it to 2 FPS and segment at 2-second intervals. Both datasets provide annotations for task description, step descriptions, and their corresponding start and end times. In the main paper, we detail the pipeline for constructing the incremental video data and generating the cumulative Chain-of-Thought in Sec. 3.3. The default large models used in our data construction process are Qwen2.5-VL-72B-Instruct and Qwen2.5-72B-Instruct. The statistical information of the generated dataset is provided in Tab. 5.

Table 5. Detailed statistics information of two generated datasets.

	Alfred Dataset	Ego4D Dataset
Dataset size	11499	13965
Total number of turns	124821	127694
Average turns per trajectory	10.85	9.14
Average video length (seconds)	47.58	22.20
Frame resolution (pixels)	300 × 300	320 × 240

We construct the benchmarks using the same procedure under unseen scenarios. Three volunteers manually filtered the data. In ALFRED, 669 out of 722 automatically annotated samples were retained, resulting in a 93% retention rate. In Ego4D, 529 out of 718 samples were kept, yield-

ing a 74% retention rate. The Ego4D environment is inherently more complex, and the large model may not have taken the entire environment into full account during data generation, leading to slightly lower data quality than in ALFRED. Even so, our dataset still maintains a 74% retention rate, indicating satisfactory data quality.

A.2. Details of Generating Distractor Task Descriptions

Introducing distractor descriptions during training prevents the model from relying on visual variations as shortcuts for inferring task progress. It instead forces the model to jointly reason over task descriptions and visual inputs, improving robustness and applicability in complex downstream environments filled with distractions.

Here, we detailed our strategy for generating distractor task descriptions. The crucial point is that the ground-truth progress corresponding to the generated distractor description must be explicitly determinable. This means that such descriptions cannot be produced by directly prompting a large model, as the resulting progress values would be undefined. To address this, we develop a constrained prompt strategy. During generation, the VLM is required to produce distractor descriptions that overlap with the initial n_r steps while remaining unrelated to all later steps. Additionally, we constrain the generated descriptions to contain the same number of sub-steps as the original. This design enables an accurate mapping of distractor descriptions to corresponding progress in the expert demonstration. When the expert video is executing the first n_r steps, the progress corresponding to the distractor description increases normally. At the end of step n_r , the progress reaches its maximum value of $100 \cdot \frac{n_r}{n}$. After that point, the progress associated with the distracting description stays constant. The complete formulation is:

$$p_t = 100 \cdot \min\left(\frac{n_r}{n}, \frac{k}{n} + \frac{1}{n} \cdot \frac{t - t_{k+1}^s}{t_{k+1}^e - t_{k+1}^s}\right). \quad (11)$$

Furthermore, to enhance the quality and difficulty of the distracting descriptions, we extract existing objects and entities from the environment as candidate elements during generation. The prompt used for generating the distracting descriptions is shown in Appendix F.2.

B. Training & Evaluation

B.1. Training Details

We leverage *LLaMA-Factory* framework for supervised finetuning, and *Verl* for RL. To adapt to our data format, we modify the multi-turn rollout algorithm in the *Verl* framework by removing the history contexts of previous turns and adding the cumulative chain of thought. We also add the multi-modal support in the multi-turn rollout code. The source code and training scripts will be made available.

Both the SFT and RL stages are trained using $8 \times A100$, with full-parameter fine-tuning. The learning rate is set to $1e-5$, and the maximum number of output tokens is 1024. For reinforcement learning, we employ the PPO algorithm with the KL coefficient set to 0. For the ALFRED dataset, the SFT stage is conducted with a batch size of 64 for 8 epochs (15k steps). We use the checkpoint optimized for 5,000 steps as the cold-start model and continue training it with reinforcement learning. In the RL stage, we select longer multi-turn data with at least 5 turns to enhance the model’s ability in long-horizon reasoning. The batch size for RL is 32, trained for 1.5 epochs. The RL-Zero model is trained for 3.5 epochs, during which the overall reward is observed to have converged. On the Ego4D dataset, the SFT stage is trained with a batch size of 64 for 8 epochs (16k steps). The checkpoint obtained after 10k optimization steps during SFT is used to initialize the subsequent RL training. During the RL stage, we select challenging multi-turn interaction data with more than 10 rounds. The RL batch size is also set to 32, trained for 4 epochs. The RL-Zero model is trained for 8 epochs.

We use DeepSpeed for SFT training. The ZeRO-3 of-fload mode is enabled only when training models with full video inputs, whereas ZeRO-2 optimization is used for the remaining experiments. To ensure fairness, all ablation studies are conducted under identical training settings. Further training parameters are available in our released scripts.

The training duration on the two datasets is approximately the same: the SFT stage takes about 50 hours, and the RL stage takes about 75 hours.

B.2. Baseline Model Information

The pretrained weights of all open-sourced models can be downloaded from HuggingFace. For LIV [19], we use the model to compute the similarity between the textual task description and each frame, and normalize the values to 0-100 to represent the progress percentage. GVL [20] and ROVER [23] are methods based on in-context learning. For a fair comparison, we fine-tune GVL on our generated dataset based on Qwen2.5-VL-7B, resulting in GVL-SFT, as reported in Tab. 1. Due to the difficulty of constructing data for ROVER’s recursive reasoning framework, we follow the original setting and directly prompt VLM to pro-

Table 6. Detailed number of input frames of each baseline model.

Model Name	Input Frames (Alfred)	Input Frames (Ego4D)
<i>API-Based Methods</i>		
GPT-5 [22]	32	32
Gemini-2.5-Pro [7]	16	16
Qwen3-VL-Plus [29]	32	32
<i>Open-Source Models</i>		
MiniCPM-V-2.6 [30]	8	8
GLM-4.1V-Thinking [10]	32	32
InternVL3 [33]	32	8
Qwen2.5-VL [1]	32	32

duce the results. Since the Gemini-1.5-Pro model employed in the original implementation was officially discontinued on September 24, 2025, we use Qwen2.5-VL-72B as a replacement in our experiments.

B.3. Evaluation Metric Formulations

We compute the average metrics by performing an average at each iteration, which results in trajectories with more iterations receiving larger weights. For evaluation, we deploy the model on a single GPU using vLLM and use multithreading to access the server for accelerated inference. When measuring the time cost of different video input formats in Sec. 4.3, we use the same machine and environment and fix the number of threads to 16. The evaluation scripts will also be released.

Same as in Sec. 3.4, at iteration t , let p_t and p_t^{gt} denote the predicted and ground-truth progress, respectively, while n^{gt} and m_t^{gt} represent the ground-truth total number of subtasks and the number of completed subtasks. We present the formulations of the four metrics in the following.

$$p_{\text{mae}} = |p_t - p_t^{\text{gt}}|, \quad (12)$$

$$\Delta p_{\text{mae}} = |(p_t - p_{t-1}) - (p_t^{\text{gt}} - p_{t-1}^{\text{gt}})|, \quad (13)$$

$$\text{bin} = \begin{cases} 1.0, & \text{if } \frac{m_t^{\text{gt}}}{n^{\text{gt}}} \leq \frac{p_t}{100} < \frac{(m_t^{\text{gt}}+1)}{n^{\text{gt}}}; \\ 0.0, & \text{otherwise,} \end{cases} \quad (14)$$

$$\text{acc} = \begin{cases} 1.0, & \text{if } p_t = 100 \wedge p_t^{\text{gt}} = 100; \\ 1.0, & \text{if } p_t \neq 100 \wedge p_t^{\text{gt}} \neq 100; \\ 0.0, & \text{otherwise.} \end{cases} \quad (15)$$

Since iterations with $p_t^{\text{gt}} < 100$ account for the majority of each trajectory, we calculate the proportions of the two iteration types ($p_t^{\text{gt}} = 100$ vs. $p_t^{\text{gt}} < 100$) in each benchmark. We then weight the accuracy score using the inverse of their proportions to reduce distribution bias and obtain a more accurate evaluation of the model’s task-completion judgment.

Each of the four metrics, as detailed in Sec. 4.1, provides an objective and unambiguous assessment of the model’s

capability on downstream tasks. R²VLM surpasses all baseline models across all four metrics and demonstrates strong performance on the three downstream applications, including progress-enhanced embodied policy learning, reward modeling for online reinforcement learning, and step-wise proactive assistance, which demonstrates its considerable application potential.

C. Cross-Domain Generalization Analysis

We evaluate the cross-domain generalization of R²VLM by training on one dataset and testing on another. We also report results obtained by training on the combined datasets, as summarized in Tab. 7. The results show that models trained with cross-domain data consistently outperform the zero-shot baseline. Moreover, training on both datasets further improves performance on most metrics, with particularly notable gains on Ego4D. The strong cross-domain performance of R²VLM suggests that the model acquires genuine reasoning and analytical capabilities, instead of simply memorizing dataset-specific patterns.

Table 7. Cross-domain generalization evaluation of R²VLM.

Train Dataset	Alfred				Ego4d			
	$p_{\text{mac}} \downarrow$	$\Delta p_{\text{mac}} \downarrow$	$\text{bin} \uparrow$	$\text{acc} \uparrow$	$p_{\text{mac}} \downarrow$	$\Delta p_{\text{mac}} \downarrow$	$\text{bin} \uparrow$	$\text{acc} \uparrow$
None	27.87	10.61	0.295	0.494	28.32	10.11	0.206	0.485
Alfred	2.77	2.27	0.876	0.961	25.03	5.87	0.252	0.539
Ego4D	19.64	6.56	0.336	0.611	<u>23.03</u>	<u>3.98</u>	<u>0.266</u>	<u>0.685</u>
Alfred + Ego4D	<u>3.02</u>	2.26	<u>0.862</u>	0.968	22.68	3.67	0.268	0.714

D. More Details of Three Downstream Application Experiments

Here in this section, we provide more details and analysis of the three downstream application experiments in Sec. 4.4.

D.1. Progress-enhanced Embodied Policy Learning

This experiment originates from the Alfred paper, where the authors argue that tracking progress toward the goal helps agents assess the utility of intermediate states and ultimately improve task performance. To this end, they added an auxiliary head to the Seq2Seq model to predict task progress and incorporated the prediction loss as additional supervision during training.

One setting used in Alfred defines progress based on execution time, assuming that progress increases linearly throughout the task. This definition is clearly suboptimal, as it fails to capture the inherent heterogeneity of task steps in terms of duration and action density. Some critical steps can significantly accelerate task advancement within a short period, whereas others may take considerable time while contributing only marginally to actual progress.

As most datasets in the AI2-THOR simulator do not provide explicit annotations of task progress, R²VLM can

serve as a general-purpose progress estimator. By defining progress based on the proportion of completed steps, R²VLM offers a more faithful estimate of task advancement. In this experiment, we compare the effectiveness of progress estimates provided by R²VLM with those based on execution time in assisting policy learning for embodied agents. Specifically, the time-based progress can be directly computed, whereas the estimated progress is obtained by using R²VLM to annotate the entire training set. The results in Tab. 3 clearly demonstrate the superiority of the progress estimates produced by our model, not only highlighting their practical value in assisting agent policy learning but also validating the effectiveness of our proposed progress definition strategy.

D.2. Reward Modeling for Online Reinforcement Learning

This experiment validates the effectiveness of the proposed progress estimation model for applying in online reinforcement learning tasks. Given that progress can be interpreted as a general value estimate of the agent’s state [20, 23], the change in progress thus corresponds to the immediate reward of the executed action, as derived from the Bellman Equation with $\gamma = 1$.

In this experiment, we leverage the Alfred-RL [31] source code to conduct online reinforcement learning on top of the pretrained SPRINT [31] model. The SPRINT model is pretrained on LLM-composed task sequences and possesses basic capability for executing multi-step tasks. We further perform reinforcement learning on tasks from the *EVAL_{INSTRUCT}* benchmark, which contains complex, previously unseen tasks with longer action sequences (up to 5 steps). We compare the step-completion-based reward used in the original paper with the rewards generated by Qwen2.5-VL-Instruct-7B and our R²VLM. The results in Tab. 4 demonstrate the strong reward-modeling capability of our model, which significantly surpasses the two baselines.

Note that R²VLM is pretrained using four input frames. However, in the online RL stage, the simulator only provides one observation image after each action. In our implementation, we duplicate the pre-action image twice and the post-action image twice to form the input. This design is mainly to match the input format used during R²VLM pretraining. We believe that training a single-frame version of R²VLM specifically for this task could further improve its effectiveness. Fig. 5 visualizes the trajectories and corresponding rewards of the policy model executing on the same task before and after reinforcement learning. As shown, the rewards provided by R²VLM effectively guide the model toward executing the correct actions. For example, the pretrained SPRINT model fails to pick up the red pen specified in the task description and instead grabs the keychain.

Our model correctly identifies this mistake and assigns no reward. In contrast, when R²VLM observes the correct pickup of the red pen, it provides a reward of 0.20. This illustrates the model’s reward modeling capability.

We argue that training embodied agents with reinforcement learning is a highly challenging process, especially when the task consists of multiple sequential steps (such as the five-step long-horizon tasks in this experiment). In such cases, if the reward signals are too sparse or inaccurate, the training process can easily become unstable or even collapse. We present a method that utilizes VLMs to deliver dense, precise, and general rewards for embodied tasks, which effectively addresses the challenge above and offers a new solution for reward modeling and policy learning in complex scenarios. This feasibility is also supported by our input formulation that integrates short video snippets with historical CoT, enabling the model to quickly provide rewards based on the agent’s post-action observations, thus serving as an online reward provider.

D.3. Step-wise Proactive Assistance

This experiment investigates the effectiveness of our progress estimation model, which is designed to process streamed video, in providing proactive assistance. Our model performs inference every two seconds based on the input streamed video. It integrates the historical CoT with the newly observed video snippet to conduct recurrent reasoning and update the CoT. The updated CoT, which records the step completion status, is then passed to an LLM to produce a structured summary of the completed, in-progress, and pending steps for users. This is a straightforward task, and we use Qwen2.5-Instruct-7B in practice. Such step-wise progress feedback helps both agents and human operators maintain a clearer understanding of the current task execution state and assists better decision-making in complex tasks. This capability has promising applications in areas such as education and healthcare, and can be deployed on intelligent wearable devices.

E. Visualization of More Results

E.1. Step-by-Step Progress Estimation

We visualize the step-by-step progress estimations for four data points in Fig. 7. The upper plots correspond to the Alfred benchmark, and the lower ones to the Ego4D benchmark. The left column shows visualizations of original descriptions, while the right column displays the progress of distractor descriptions. As shown in the figures, the occurrence of key steps triggers sharp increases in progress, whereas irrelevant steps produce no progress increments. The progress estimated for distractor description increases only in the early stage when relevant steps appear in the expert trajectory, and remains unchanged thereafter.

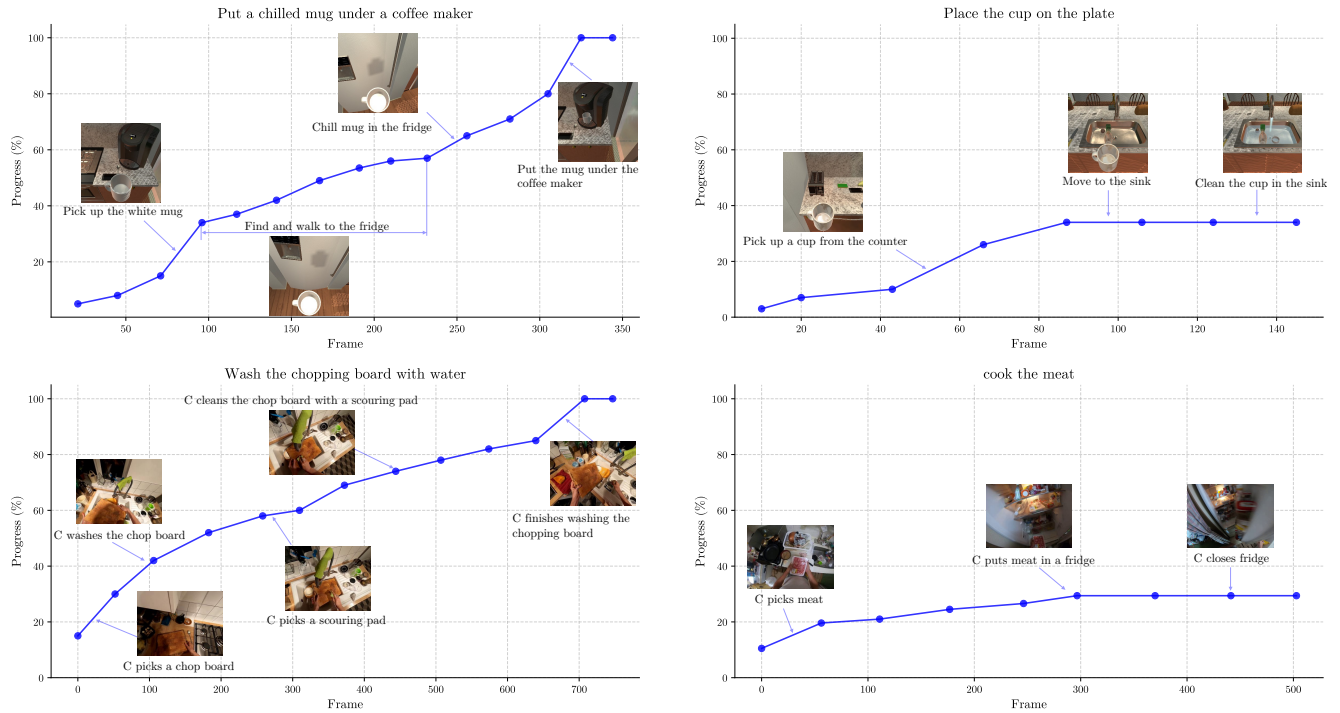


Figure 7. We visualize the model’s progress estimation on four tasks, where key steps trigger sharp progress increases, and irrelevant steps yield zero progress increments.

E.2. Dynamic CoT Revision

The CoT revision mainly occurs when the initial task decomposition conflicts with the actual execution in the video. As illustrated in Fig. 8, at $t = 5s$ the model generates an initial CoT from prior knowledge, while at $t = 9s$ it observes the spinach being washed and thus revises the CoT accordingly to match the visual evidence.

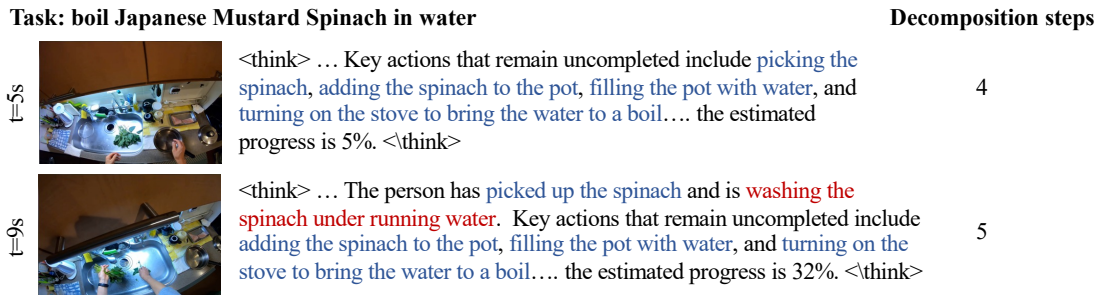


Figure 8. A demo of R²VLM dynamically revising the CoT.

F. Prompts

F.1. Model System Prompt

The System Prompt Defines the Task and Reasoning Strategy.

```
You are a professional assistant skilled in understanding temporal videos and
estimating task progress.
You will be given a history chain of thought (CoT) summarizing completed and remaining
steps, along with incremental egocentric video frames.
Your task is to:
1. Analyze the new frames to detect any completed actions or state changes.
2. Update the list of completed steps and revise the remaining steps accordingly. (
Merge, split, or reorder steps if necessary)
3. Estimate the overall task completion progress as a percentage, based on the
proportion of completed steps made toward the full task. (eg. finish 1 of 3 tasks,
progress is 33~66)
Use the following output format:
<think>
[Updated chain of thought: clearly list revised completed steps and remaining steps.]
</think>
<answer>
[Task completion percentage based on the proportion of completed steps, an integer
between 0 and 100]
</answer>

History CoT: {history_cot}
Images: <image><image><image><image>
Question: {question}
Output:
```

F.2. Distractor Task Description Generation Prompt

The Prompt to Generate Distractor Task Description.

```
You are given a task description, a sequence of initial steps, and a set of available
objects in the environment.

Your goal is to generate a distractor task description that meets the following
conditions:
1. The distractor task description must share the same initial steps.
2. The distractor task must ultimately be different from the original task.
3. The distractor should have the same number of substeps as the original description.
4. The distractor task should be constructed using one or more objects from the
provided object list.
5. The distractor should be natural and plausible in the given environment.

Only output the distractor task description, without any additional explanations or
context.

Example1:
Input:
- Original task description: Put a wine glass in the refrigerator.
- Initial steps:
Turn around and go to the kitchen counter, to the right of the sink.
```

- Objects: Apple, SaltShaker, Mug, Pot, Cup, Egg, SoapBottle, Bread, DishSponge, Plate

Output:

Place the apple from the counter onto the plate.

Example2:

Input:

- Original task description: Put heated egg on kitchen table.

- Initial steps:

walk to face sink

pick up egg from sink

walk to face microwave

- Objects: Bowl, Pan, Lettuce, PepperShaker, Cup, Egg, SoapBottle, Bread, DishSponge, ButterKnife, Apple, SaltShaker, Mug, Pot, Potato, Knife, Plate, Spoon, Fork, Tomato, Spatula

Output:

Put the egg into the bowl and pick up the bowl.

Now it's your turn:

Original task description: {task_description}

- Initial steps:

{narrations}

- Objects: {objects}

Output:

F.3. Chain of Thought Generation Prompt

The Prompt to Generate Chain of Thought.

You will be given:

1. A **video** and a question about the **progress** of a **task** shown in the video.
2. A list of **completed narrations** and **uncompleted narrations**.

Your job is to generate a human-style chain of thought describing the task's progress, based on your understanding of how the task is completed and the provided completed and uncompleted narrations.

You must:

- Analyze what important steps have been completed and how they contribute to the overall task progress.
- Analyze what key steps remain uncompleted and how crucial they are for finishing the task.
- Give a final estimated progress percentage **based on the proportion of completed steps to the total number of steps.**
- Treat the narrations only as background guidance to simulate watching the video, and base your reasoning on the video content. Do not use any language implying reliance on the narrations or mentioning the phrase "completed narrations" or "uncompleted narrations."

Now begin your analysis:

Task:
{task_description}

Completed Narrations:
{completed_narrations}

Uncompleted Narrations:
{uncompleted_narrations}

Progress:
{progress}

Output:

Your output should follow this format:

<think>

To determine the progress of the task "{task_description_lower}", let's analyze the video:

[Explain which actions were completed and their significance in moving toward the goal .]

[Explain which key actions remain uncompleted and why they are crucial.]

[Conclude with a reasonable progress estimate **based on the proportion of completed steps to the total number of steps**.]

</think>

<answer>

You must strictly output the same final progress percentage I provided above: { progress}

</answer>

Examples of Generated Chain of Thought.

<think>

To determine the progress of the task "put a clean potato slice on the counter," let's analyze the video:

The robot has already turned around and faced the counter, picked up the knife, crossed the room to face the stove, sliced the potato, placed the knife back on the counter, and taken a piece of potato. These actions indicate significant progress towards preparing the potato slice. The slicing step is crucial as it directly contributes to creating the potato slice needed for the task.

However, several key steps remain uncompleted. The robot needs to cross the room to the sink, wash the potato slice under running water, and then place the clean slice on the counter. Washing the potato is essential to ensure it is clean before placing it on the counter, making these remaining steps critical for completing the task.

Considering the proportion of completed steps to the total number of steps, the progress appears to be substantial but not yet complete. The slicing part is done, but the cleaning and final placement are still pending. Based on this reasoning, a reasonable estimate of the progress would be around 65%.

</think>

```
<answer>  
67  
</answer>
```