Plug-and-Think: Structured Reasoning for Vision–Language–Action Models

Supplementary Material

Kaikai Wei[1,*] Di Wen[2,*] Xinhai Li[1] Senwei Xiang[1,†]

[1]Hangzhou International Innovation Institute of Beihang University, China
[2]Beijing Jiaotong University, China

1. Additional Implementation Details

1.1. Training Details

BridgeLang, which utilizes the Qwen2.5-VL-3B model, was trained using the hyperparameters detailed below. This process constitutes the lightweight instruction finetuning on our Bridge-CoT dataset.

- **Learning Rate:** 2e-5
- **Batch Size:** 2
- **Gradient Accumulation Steps:** 8 (Effective batch size of 16)
- **Optimizer:** AdamW
  - *Betas:* (0.9, 0.999)
  - *Epsilon (ϵ):* 1e-8
  - *Weight Decay:* 0.01
- **LR Scheduler:** We employed a linear warmup scheduler with a 0.03 warmup ratio over the training steps.
- **Hardware:** The model was trained on a single NVIDIA H20 GPU.
- **Total Training Time:** The entire instruction finetuning process completed in approximately 3 hours.

2. Bridge-CoT Dataset and Prompts

2.1. Dataset Generation Prompts

As described in the main paper (**??**), the Bridge-CoT dataset was generated by querying the Qwen2.5-VL-72B teacher model. We employed a strict, two-part prompt structure (a `SYSTEM_PROMPT` and a `USER_TEXT_TEMPLATE`) to ensure consistent and parsable structured output. This exact prompt structure is also used as P_{tpl} during the training and inference of BridgeLang, as referenced in **??**.

2.1.1. System Prompt

The system prompt defines the model's role and its high-level output constraints.

```
You are a robot perception supplementer.
Output ONLY the three schema fields in the exact
    order and format.
Keys like 'name:', 'color:' are FORBIDDEN in
    entries.
Any extra text or missing section is prohibited.
```

2.1.2. User Prompt Template

The user prompt provides the specific image and task, along with detailed, strict rules for the output format. The `{task}` placeholder is dynamically filled with the task instruction for each sample.

```
<image>
<task>{task}</task>
Rules (strict):
1) You MUST output ALL sections in EXACTLY this
    order:
  <objects>...</objects>
  <relations>...</relations>
  <subgoals>...</subgoals>

2) Inside each section, entries MUST follow the
    signatures below.
  Do NOT invent keys like 'name:' or 'color:'.

  <objects> entries: object_name:color:count; ...
       - Exactly TWO colons per entry (3 fields
         ).
       - NO numbering like '1:'; NO keys like '
         name:'.
       - Examples (correct): box:brown:1;
         coca_cola:red:1
       - Examples (forbidden): name:box:color:
         brown:1 (X),
                      color:brown; count:1 (X)


  <relations> entries: subject:predicate:object;
      ...
       - Exactly TWO colons per entry (3 fields
         ).
       - NO numbering; NO keys.
       - predicate MUST BE from {on, under,
         left_of,
                        right_of, near,
                            inside}
       - Examples (correct): coca_cola:right_of
         :box
       - Forbidden: name:box:left_of:name:
         coca_cola (X)

  <subgoals> entries: 1:action object; 2:action
      object; ...
```

```
        - MUST be numbered 1,2,3,...
        - action MUST BE from {grasp, open,
            close, pour,
                        place, push, pull}
        - Example (correct): 1:grasp coca_cola;
                        2:place coca_cola
                            inside box

3) Use ONLY entities visible in the CURRENT image
     and the TASK.
4) NO natural-language sentences. NO extra spaces
     around tags.
  Stop IMMEDIATELY after "</subgoals>".
```

2.2. Quality Control

We implemented a rigorous quality control protocol for the 38,660 samples in the final training set. All samples were verified to be 100% correct according to our strict criteria:

1. **Object Grounding:** All entities listed in `<objects>` are relevant to the task and visually present in the image.
2. **Relational Accuracy:** All `<relations>` correctly describe the spatial context between the specified objects.
3. **Plan Validity:** All `<subgoals>` form a logical, correct, and executable plan to achieve the given task.

3. BridgeLang Semantic Post-processing

To generate the **Cleaned Subgoals** evaluated in our main experiments (e.g., **??**), we apply a crucial, deterministic, rule-based semantic cleaning process to the raw text output from BridgeLang. This procedure translates the structured plan into a clean, natural-language string that is compatible with the pre-trained OpenVLA model's language embedding space, thereby avoiding the 0.0% failure mode.

The process consists of the following ordered steps:

1. **Extract Subgoal Content:** We first use a regular expression (denoted `_SUBGOALS_RE`) to extract only the text content between the `<subgoals>` and `</subgoals>` tags. If no match is found (e.g., if the model fails to produce the tag), the function returns an empty string.
2. **Remove Numbering:** We strip all numbered prefixes (e.g., "1:", "2 :", "10:") from the subgoal plan. This is achieved using the regex:
   `re.sub(r"\b\d+\s*:\s*", "", body)`
3. **Normalize Spacing and Terms:**
   - Underscores (often used in object names) are converted to spaces (e.g., "black_bowl" → "black bowl").
     `body.replace("_", " ")`
   - All consecutive whitespace characters (which may result from the previous step) are collapsed into a single space. Leading and trailing whitespace is also stripped.
     `re.sub(r"\s+", " ", body).strip()`
4. **Standardize Punctuation:**
   - We ensure a single space follows every semicolon (which separates subgoals) to maintain a consistent format.
     `body.replace(";", "; ")`
   - Any trailing periods or spaces at the very end of the string are removed to prevent noise.
     `body.rstrip(" .")`

4. Evaluation Result Logs

To ensure full transparency and verifiability of our main quantitative results (presented in **??**), we have included the raw evaluation logs as part of our supplementary material `.zip` file.

These text files are not W&B logs, but are the direct `stdout` outputs from our evaluation script. They detail the success or failure status for each of the 2,000 evaluation episodes (4 suites $\times$ 10 tasks $\times$ 50 episodes) for both our method (OpenVLA + BridgeLang) and the baseline (OpenVLA). These logs serve as the raw data from which the final average success rates reported in the paper were computed.