SIMPACT: Simulation-Enabled Action Planning using Vision-Language Models

Supplementary Material

This supplementary material provides additional implementation details, experiment analyses, and qualitative results supporting our main paper. We describe the full simulation-construction pipeline, including VLM-based prediction of rigid and deformable object parameters, as well as the symbolic action space and prompting strategy used for action optimization.

Additionally, we present more qualitative examples, an ablation on the number of VLM-sampled action proposals, and a study comparing a CEM-based Prompting-with-the-Future-style variant [45], which shows near-zero success and highlights the importance of both the VLM sampler and VLM optimizer. We also show that **SIMPACT** demonstrates robustness under randomized scene variations, and provide representative failure cases.

Importantly, we perform an additional experiment that analyzes the consistency between simulation and real-world performance, showing strong alignment (89% agreement) while noting remaining sim-real gaps. This indicates that our VLM-Simulation integration serves as a high-fidelity world model for planning.

A. Implementation Details

A.1. Simulation Construction Details

Physical Parameters θ_{phys} Prediction In Sec. 3, we outlined the VLM-based physical parameter prediction of θ_{phys}. Here we provide further details on how to predict the physical parameters of both rigid and deformable objects. This process follows a question-answering framework, where each question has the scene RGB image as an additional input to the VLM.

Q1. Identify the objects that need to be manipulated from {*task instruction*}. Determine whether to use a rigid or deformable object simulator based on the object's material.

Q2. For rigid {*object*}, predict its mass and friction parameters.

Q3. For deformable {*object*}, decide whether to use Projective Dynamics or the Material Point Method based on the stiffness of the object.

Q4. For {*object*} simulated with Projective Dynamics, determine the following physical parameters:
- Young's modulus
- Poisson's ratio
- Mass density

Q5. For {*object*} simulated with the Material Point Method, first determine the material type of {*object*} from the set: {*jelly*, *metal*, *sand*, *foam*, *plasticine*}. Then output:
- Young's modulus
- Poisson's ratio
- Mass density
- (Optional) Friction angle
- (Optional) Yield stress

A.2. Action Planning Details

This section provides further details on how our action planning framework is instantiated.

Symbolic Actions Here we provide a complete list of high-level symbolic actions A_t and their corresponding continuous parameters, which are used by VLM in the SAMPLE function in Alg. 1 and Eq. 13.

- `PUSH`(δ_x, δ_y): Move the end-effector horizontally from its current position by (δ_x, δ_y) while maintaining its current height.
- `LIFT`(δ_z): Move the end-effector upwards along the z-axis by δ_z.
- `DESCEND`(δ_z): Move the end-effector downwards along the z-axis by δ_z.
- `GRASP`(d): Adjust gripper to a target width d (in meters), where 0.0 is fully closed and 0.1 is fully open.
- `RELEASE`: Fully open the gripper.
- `ROTATE`(δ_{yaw}): Adjust end-effector yaw relative to its current orientation by δ_{yaw} (in radians).
- `MOVE`(δ_x, δ_y, δ_z): Move the end-effector from current position by (δ_x, δ_y, δ_z).

Note that these symbolic actions are redundant; for example, both the `DESCEND` and `LIFT` actions move along the z-axis, differing only in direction. However, we empirically found that this additional semantic structure allows the VLM to reason more effectively. For instance, in the *bowl stacking* task, the VLM more reliably infers that it should descend first and then lift the bowl after grasping.

Action Proposals Generation Here we provide further details of ℓ_{sample}, specifically regarding how we leverage the VLM sampler to generate action proposals. Figure 8 illustrates the prompt used for generating $\text{A}^i = \text{VLM}(I_0, \ell_{\text{task}}, s_0; \ell_{\text{sample}})$ as described in Eq. 13.

Optimization Context c Generation To instantiate the OPTIMIZE function, we construct the context c^i from the action sequence a^i and the simulated state rollout s^i. We sample the state at the end of each symbolic action, where each action specifies the gripper's Cartesian position (x, y, z) and orientation (roll, pitch, yaw). For rigid objects, the numerical state consists of their full 6-DoF rigid transformation. For deformable objects, the numerical state includes voxel-

Task Specification You are a versatile, general-purpose AI assistant functioning as an embodied planner for a robot arm. Your objective is to decompose a high-level natural language instruction into multiple distinct, high-level action plans. Analyze the user's instruction and scene context to propose # different, plausible action plans, each composed of a sequence of action primitives exploring different strategies. Determine if the task requires a single primitive or a sequence of primitives. Avoid aggressive or risky proposals and focus on plans with high success rates.

Input Specification
- Image of the Scene: Visual observation of the workspace.
- Additional Scene Context: Object and end-effector coordinates in the world frame, workspace constraints.
- Natural Language Instruction: High-level task goal.

Action Primitive Definitions All coordinates (x, y, z) are in the absolute world frame. The available primitives are described textually the same as list A.2.

Output Specification Return a JSON object with key `"action_proposals"` containing # entries, each with:
- `"description"`: Brief explanation of the high-level plan logic.
- `"action_sequence"`: List of action primitives in one of the formats below.

```
PUSH:    {"type":"PUSH", "delta_x":float, "delta_y":float, "reasoning":"..."}
LIFT:    {"type":"LIFT", "delta_z":float, "reasoning":"..."}
DESCEND: {"type":"DESCEND", "delta_z":float, "reasoning":"..."}
GRASP:   {"type":"GRASP", "width":float, "reasoning":"..."}
RELEASE: {"type":"RELEASE", "reasoning":"..."}
ROTATE:  {"type":"ROTATE", "delta_yaw":float, "reasoning":"..."}
MOVE:    {"type":"MOVE", "delta_x":float, "delta_y":float, "delta_z":float,
          "reasoning":"..."}
```

Figure 8. **Action sampling prompt ℓ_{sample} outline.** This prompt includes task specifications, input requirements, action primitive definitions, planning guidelines, and output format. It is combined with visual observations and scene context as input to the VLM sampler to generate diverse action sequence proposals. Symbol # indicates the number of proposals to generate for each call.

downsampled keypoint coordinates together with the 3D bounding box of the object's point set. Here we further provide an example context in Fig. 9.

Action Optimization We provide details the action optimization prompt ℓ_{opt} in Fig. 11, which enables a VLM to serve as an action optimizer. The prompt includes three key elements: task specification, input specification, and output specification.

B. Supplementary Results

B.1. Additional Qualitative Results

We show qualitative results for the *sweeping* task that was not included in the main paper due to space constraints in Fig. 10.

B.2. Further Ablation Analysis

We additionally consider a variant of our method in which we simultaneously replace the VLM sampler with a random sampler and switch the VLM optimizer to a sampling-based optimizer (e.g., the cross-entropy method). In this setting, the VLM serves only as an evaluator used to select the best rollout. Notably, this simplified variant is algorithmically identical to Prompting-with-the-Future (PWTF) [45].

We follow the open-sourced CEM implementation from PWTF and adopt the same set of hyperparameters. We evaluate this variant and find that it consistently achieves a zero success rate across all of our real-world tasks. This result further highlights the importance of both the VLM sampler and the VLM optimizer, as a naive initial sampling distribution combined with a local update process has limited performance. The original CEM optimization appears effective in PWTF primarily because their experiments focus on pick-and-place problems, for which sampling a reasonable initial solution is relatively easy.

B.3. Correlation Between Simulation and Real-World Performance

This section examines the correlation between simulation and real-world results, specifically whether success or failure in simulation predicts the corresponding real-world outcome. Since our planning pipeline optimizes action se-

Example Rollout Context

```
{
  "timestamp": "20260112_224423",
  "object_names": ["brown_box",
  ↪  "pocky_box"], % total 2 objects
  "waypoints": [
    {
      "position": [0.4199, -0.2452,
      ↪  0.3555],
      "orientation": [0.00, 0.71, 0.70,
      ↪  0.00],
      "gripper_width": 0.1,
      "duration": 3.0
    },
    ...
  ], % total 5 waypoints
  "snapshots": [
    {
      "waypoint_index": 0,
      "gripper": {
        "position": [0.4201, -0.2460,
        ↪  0.2503],
        "orientation": [0.00, -0.71,
        ↪  -0.70, 0.00],
        "width": 0.04
      },
      "objects": {
        "pink_blue_box": {
          "position": [0.4654, 0.2061,
          ↪  0.1240],
          "orientation": [-0.49, -0.47,
          ↪  0.52, 0.52]
        },
        "pocky_box": {
          "position": [0.4408, -0.1004,
          ↪  0.1471],
          "orientation": [0.70, 0.71,
          ↪  -0.02, -0.02]
        }
      },
      "screenshot":
      ↪  "rollout_screenshot_0.png"
    },
    ...
  ] % total 5 snapshots
}
```

Figure 9. **Example rollout context for action optimization in pivoting task.** The context contains the action waypoints and the simulated state snapshots at each waypoint, including gripper pose, object poses, and screenshot paths. Only the first entry is shown for repeated fields, with omitted entries summarized using comments.

quences for task success, we also include 10 unoptimized VLM sampled action proposals to capture failure cases to better understand the sim-to-real gap. We conduct this experiment on five selected tasks *non-toppling*, *bowl stacking*, *pivoting*, *shape rope* and *shape dough*. Each task therefore has 20 samples: 10 from the main experiments using our full pipeline, and 10 using direct VLM sampled action sequences. Results are shown in Fig. 12.

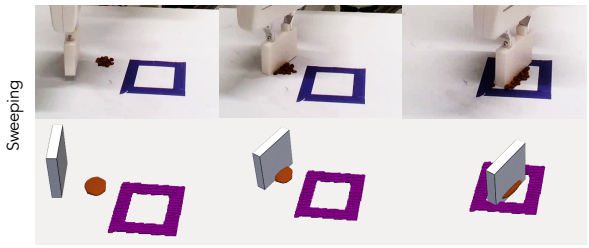

Figure 10. **Additional qualitative results.** Following Fig. 5, this figure shows the initial state, execution progress, and final state for the *sweeping* tasks.

Across tasks, we observe a high degree of consistency between simulation and real-world outcomes, with 89% of all cases exhibiting aligned success or failure. Such alignment is critical to the effectiveness of our approach, as it indicates that the physical simulation provides a reliable grounding for VLM-based planning. Simulated failures enable the VLM to avoid similar real-world failures, while simulated successes offer informative guidance for selecting effective action sequences. Despite the overall high alignment ratio, there remains room to improve simulation and real consistency. In the *pivoting* task, 15% of cases succeed in simulation but fail in the real world, and in the *shape dough* task this discrepancy ratio is 20%. These tasks appear more sensitive to accurate physical modeling and contact dynamics. Improving simulation fidelity, e.g., via system identification, may reduce these discrepancies and prevent our planner from selecting actions that succeed in simulation but fail in the real world.

B.4. Computation Time

Table 5 reports the runtime of each component in our method. We observe that the simulation construction stage takes less than two minutes on average, with the majority of the time consumed by running the pretrained image-to-3D model. The image segmentation and pose estimation steps require significantly less time.

The VLM planning stage is the most time-consuming component. This is primarily due to the reasoning time of the VLM as well as the network latency introduced by querying the Gemini API. Within this stage, the largest portion of the runtime comes from action sampling. This is because we intentionally perform multiple VLM queries to encourage diversity in the generated action proposals, rather than relying on a single query to produce all samples. With more efficient VLMs tailored for robotics applications, the

Task Specification

You are an AI assistant acting as an embodied planner. Your objective is to analyze simulation rollouts and propose one optimized action plan for a real-world task. Simulation and real-world physics are similar but may differ due to sim2real gaps (e.g. appearance, pose, scale, friction).

1) Analyze Rollouts: Inspect each rollout's `action_sequence`, robot/object poses at each waypoint, and screenshots.
2) Infer Logic & Physics: Identify the causes of failures and the characteristics of successful attempts.
3) Propose Optimized Plan: Output a refined plan that avoids prior mistakes and leverages successful rollout elements.

Input Specification

- Task Instruction: Main task goal.
- Real-World Context: Workspace limits, safe ranges
- Simulation Rollouts: Specify the format of input context describing action and state.

Screenshots appear as `fig_0.png`, `fig_1.png`, ... and should be used to evaluate rollout outcomes.

Output Specification

Produce a JSON object with key `"action_proposals"` containing exactly one entry:

- `"description"`: How the new plan improves on the rollouts.
- `"action_sequence"`: A list of actions in one of the formats below.

```
Move Action: {
  "type":"move", "delta_x":float, "delta_y":float, "delta_z":float,
  "delta_roll":float, "delta_pitch":float, "delta_yaw":float, "reasoning":"..."
}
Gripper Control: {
  "type":"gripper_control", "width":float, "reasoning":"..."
}
```

Figure 11. **Action optimization prompt ℓ_{opt} outline.** This prompt includes task, input, and output specifications. It is combined with simulation rollout context as input to the VLM action optimizer to generate optimized action sequences.

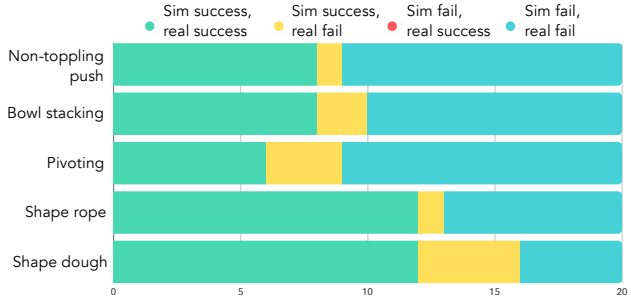

Figure 12. **Correlation Between Simulation and Real-world Success/Failure.** Results from 20 samples per task (100 total). Each rollout is categorized as one of: sim-success/real-success (green), sim-success/real-fail (yellow), sim-fail/real-success (red), and sim-fail/real-fail (blue). Simulation and real outcomes match in 89% of cases (both success or both failure), with 11% showing sim-success/real-fail. We observed no cases where a sequence failed in simulation but succeeded in the real world.

planning loop could be made substantially faster.

The total simulation stage takes less than one minute on average. Our physical simulation has been optimized for efficiency, where each rollout lasting 5–8 seconds depending on the task. Implementing batched simulation for multiple rollouts would further reduce the overall simulation time.

Table 5. **Computation time.** We compute the average computation time over 10 cases from each task.

Component	Time (mins)
simulation construction	1.9
action sampling	2.8
simulation rollout	0.8
action optimization	0.9

B.5. Robustness Validation

We validate the robustness of our method by randomizing the scene layout and introducing different distractors for each rollout, as illustrated in Fig. 13. Our evaluation highlights robustness across several dimensions, including the presence of unrelated objects in the environment, variations in the relative positions of task-relevant objects, and changes in the color or texture of the manipulated items. These results demonstrate that our method naturally generalizes to a wide range of scene variations, owing to the

strong scene-understanding capabilities of the VLM.

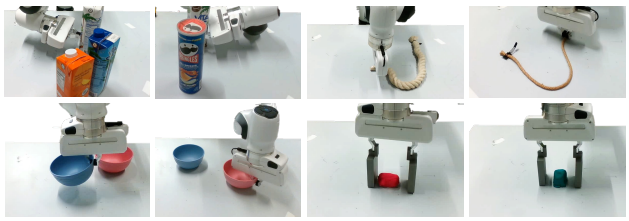

Figure 13. **Example scene setup variations.** Throughout our experiments, we vary the object types, poses, colors and materials to demonstrate the robustness and generalizability of our method.

B.6. Physics Parameters Estimation Analysis

Accurate parameter estimation is crucial for dynamics-dependent tasks (e.g., *non-toppling push*), though less significant for standard pick-and-place (e.g., *bowl stacking*). Empirically, VLM-sampled physical parameters exhibit low variance within valid ranges, as shown in Table 6, likely due to VLM's rich pre-trained knowledge.

Table 6. Robustness analysis of VLM-estimated physics parameters ($N = 10$ samples). The low variance and stable ranges indicate consistent estimation capabilities.

Task	Parameter	Mean ± Std	Range [Min, Max]
Non-toppling Push	Mass (kg)	1.033 ± 0.0015	$[1.0, 1.05]$
	Friction Coeff. μ	0.36 ± 0.11	$[0.3, 0.5]$
Shape Playdoh	Poisson's Ratio	0.43 ± 0.02	$[0.40, 0.45]$
	Mass Density (kg/m^3)	1186 ± 65	$[1000, 1260]$

To further verify robustness under inaccurately estimated physical parameters, we varied the object's friction coefficient μ in the *non-toppling push* task. The performance degrades at physical extremes, as shown in Table 7, where overly large friction prevents all movements, while small friction prevents toppling at all heights. However, these values all fall outside the VLM's predicted range of $[0.3, 0.5]$, confirming our method's robustness to estimation errors within a valid range.

Table 7. Success rate vs friction coefficient μ.

Fric. Coeff. (μ)	0.01	0.20	0.50	1.00	10.0
Success Rate (%)	40	100	80	80	0

B.7. Standardized Benchmark Results

We also provide evaluation of our method on the CALVIN benchmark [41] containing long-horizon tasks in simulation, as shown in Table 8. We evaluated 40 chains of instructions, and used ground-truth segmentation masks given simulated environment. Our zero-shot method outperforms imitation learning baseline HULC [40] and VLA baseline RoboFlamingo [35]. We also include results from the current best performing baseline FLOWER [56] as a reference.

Figure 14. **Failure cases.** Example failure cases in bowl stacking, pivoting, shape rope and shape dough tasks.

Table 8. Evaluation results on the CALVIN Long-Horizon Multi-Task Language Control (LH-MTLC) benchmark.

#. Instruct.	zero-shot	1	2	3	4	5	Avg. Len.
HULC [40]	✗	41.8%	16.5%	5.7%	1.9%	1.1%	0.67
RoboFlamingo [35]	✗	82.4%	61.9%	46.6%	33.1%	23.5%	2.47
FLOWER [56]	✗	99.4%	95.8%	90.7%	84.9%	77.8%	4.53
Ours	✓	87.5%	82.5%	47.5%	40.0%	20.0%	2.78

B.8. Failure Cases

We present representative failure cases of our method in Fig. 14, providing supplementary material for Sec. 4.4.

The *bowl stacking* and *shape dough* failures are both execution failures. A slight misalignment during bowl placement can cause the bowl to flip, and a small offset between the gripper center and the dough center can lead to unsuccessful squeezing. These execution failures highlight the sensitivity and difficulty of our tasks: even minor errors in the planned actions can lead to failure.

The *pivoting* and *shape rope* failures are both planning failures. For the pivoting task, stabilizing the object upright requires solutions within a narrow range of feasible angles; when the planned angle is suboptimal, the object cannot maintain stable contact with the environment. Planning failures in simulation also transfer to the real world, further reducing success rates. For the rope shaping case, we observe that failure often arises from insufficient diversity in the generated action proposals, which limits the VLM action optimizer's ability to identify effective actions. Increasing the number of sampled proposals may improve performance in such cases.

B.9. Replanning using Real-world Feedback.

Our framework can also incorporate real-world feedback to improve the success rate after execution failures. When the VLM determines that the planned action sequence has not achieved the task goal, we optionally invoke a replanning mechanism that allows the system to generate a new action sequence using updated information. More concretely, the simulation is updated with current real-world states for a new planning attempt as shown in Fig. 15. We evaluate this replanning mechanism on both the *pivoting* and *avoid obstacle* tasks, allowing up to 3 replanning attempts. For the *pivoting* task, among the six initially failed cases, re-

Figure 15. **Replanning illustration.** After initial failed execution, we perform re-planning after simulation update leading to successful completion.

planning successfully recovers 50% of them, with an average of 1.67 replanning attempts. For the *avoid obstacle* task, among the two initially failed cases, replanning resolves all of them, with an average of 1.0 replanning attempt. These results highlight the promise of incorporating real-world feedback into our system for replanning and suggest its potential in enabling robust manipulation.