

SpaceTools: Tool-Augmented Spatial Reasoning via Double Interactive RL

Supplementary Material

001 We provide additional details and extended results in the
002 supplementary materials,

- 003 • Appendix A: Limitations and future directions.
- 004 • Appendix B: Further details on the Toolshed system.
- 005 • Appendix C: Expanded method descriptions.
- 006 • Appendix D: Additional implementation details.
- 007 • Appendix E: Extended experimental results.

008 A. Limitations and Future Directions

009 Our work shows that tool-augmented spatial reasoning,
010 enabled through DIRM and the Toolshed infrastructure,
011 provides an effective and scalable foundation for training
012 VLMs with strong spatial reasoning, robust tool coordina-
013 tion, and broad generalization across diverse tasks and em-
014 bodiments. At the same time, this framework opens several
015 promising directions that fall beyond our present scope but
016 merit deeper exploration. We discuss these limitations and
017 future opportunities below.

018 **Application scope.** A natural next step is to broaden the
019 range of tasks and environments in which tool-augmented
020 spatial reasoning is applied. Our current scope focuses on
021 short- or medium-horizon tasks, such as spatial question an-
022 swering or grasp-and-place manipulations. Extending to
023 more complex, longer-horizon, or multi-stage tasks may
024 further reveal the potential of tool-augmented reasoning, al-
025 lowing the model to concentrate on reasoning and decision-
026 making rather than learning numerous precise perceptual
027 subtasks. Moreover, integrating richer environments, in-
028 cluding large-scale robotic simulation, interactive game en-
029 vironments, or physics-rich virtual worlds, could support
030 more diverse experiences and ultimately more general em-
031 bodied spatial intelligence.

032 **Methodology.** From a methodological perspective, sev-
033 eral directions could strengthen the flexibility and robust-
034 ness of tool-augmented spatial reasoning. Although Tool-
035 shed supports image-level tool outputs, this work primarily
036 explores tools that return structured text or variables (*e.g.*,
037 point cloud). Extending the model to reason over visual out-
038 puts from tools may unlock more expressive or fine-grained
039 reasoning behaviors. Another important direction is sys-
040 tematically improving how VLMs perceive, verify, and re-
041 cover from tool errors or inaccuracies. Moreover, under
042 a modular perspective, future work could investigate en-
043 hancing particular system capabilities by upgrading a single
044 tool without modifying other components, while ensur-
045 ing that overall tool coordination remains robust. Finally,

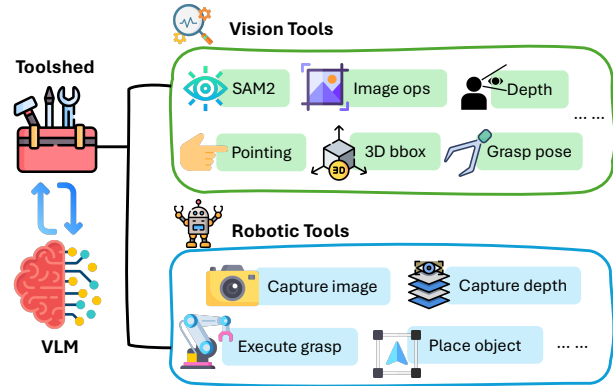


Figure 1. The **Toolshed** infrastructure linking a VLM with modular vision and robotic tools under a unified toolbox for perception and control.

while our RL exploration focuses on prompt design, loss functions, and task-specific reward formulations, alternative RL approaches, such as stepwise reward formulations, may improve learning effectiveness in large multi-tool action spaces.

Infrastructure. On the system side, Toolshed provides a scalable backbone for interactive tool use and learning, but there remains room to further improve its efficiency and resource utilization. Serving many heterogeneous tools in parallel can introduce latency and memory bottlenecks, particularly for high-resolution vision tools or robot-in-the-loop executions. In this work, we mitigate the latter by using mock robot tools during training. Future advances in scheduling, caching, batching, and asynchronous execution could potentially enhance performance and even support real robot execution effectively during interactive learning. Additionally, developing lighter-weight tools, model-side approximators, or memory-optimized deployment strategies may reduce overhead and enable larger-scale training or more complex task environments. These improvements would allow the framework to scale more gracefully as tool diversity and task complexity increase.

B. Details of Toolshed

B.1. System Design

This work aims to enable learning and inference with multiple interactive vision tools for spatial reasoning. Effective tool-augmented spatial reasoning requires multi-turn, state-dependent communication between the VLM and its vision

074 tools. However, many essential tools such as object detec-
075 tors, depth estimators, and 3D reconstruction modules, are
076 computationally heavy and often dominate both inference
077 and training time in VLM-RL pipelines. Moreover, mod-
078 ern training relies on batched generation, where multiple
079 conversations are executed in parallel. In naïve implemen-
080 tations, a single blocking tool call can stall the entire batch,
081 effectively reducing tool interaction to a serial process. This
082 makes it crucial to keep tools continuously available, pre-
083 loaded on device, and capable of serving multiple concu-
084 rent conversations. To address these challenges, we intro-
085 duce **Toolshed** (visualized in Figure 1), a distributed toolkit
086 that enables scalable, asynchronous, and parallel vision tool
087 interaction.

- 088 • **Decoupled execution.** Tool invocations run indepen-
089 dently from the policy’s main inference loop, avoiding
090 blocking calls that would otherwise stall unrelated com-
091 putations.
- 092 • **Asynchronous Processing.** Multiple parallel instances
093 can serve the same tool, each receiving inputs and pro-
094 ducing outputs asynchronously, enabling high throughput
095 even under large-scale rollouts.
- 096 • **Resources isolation.** Tool instances are assigned isolated
097 resources based on the computational profile of each tool.
- 098 • **Environment isolation.** Each tool type is hosted in a ded-
099 icated python environment, solving the dependency com-
100 patibility issue that comes with hosting multiple computer
101 vision tools in a single system.
- 102 • **Elastic scaling.** The system design supports automatic
103 spawning of additional tool workers in response to bursts
104 of tool usage, allowing throughput to remain stable even
105 for large batch rollouts. (This capability is part of the
106 infrastructure design but was not enabled in our training
107 experiments.)
- 108 • **Multimodal data passing.** Seamless exchange of text,
109 images, and structured variables (*e.g.*, 3D point clouds) is
110 supported between the VLM and tools, even when they
111 run on different devices or GPU nodes. This enables tool
112 workflows that require different types of inputs and out-
113 puts.

114 In practice, Toolshed is implemented on top of the
115 Ray¹ distributed execution framework, which provides
116 lightweight task scheduling, actor management, and high-
117 throughput message passing. For interactive reinforcement
118 learning, Toolshed integrates seamlessly with VERL²:
119 VERL’s asynchronous multi-turn rollouts align naturally
120 with Toolshed’s asynchronous tool actors, enabling us to
121 parallelize expensive perception, generation, and simulation
122 steps without slowing down rollouts. This results in signif-
123 icantly higher steps-per-second compared to monolithic or
124 synchronous training setups. For inference, Toolshed can be

¹ray.io/

²github.com/volcengine/verl

attached both to our trained model and to proprietary mod- 125
els (*e.g.*, GPT-5, Claude) via simple API calls, enhancing 126
their robotic spatial reasoning capabilities. 127

B.2. Provided Tools 128

Vision tools. We provide the following vision tools. 129
`image_ops` offers basic image manipulations such as 130
`point1` and `point2` are two 131
object-pointing detectors backed by RoboRefer [9] and 132
Segment Anything 2 [4]. `point1` and `point2` are two 133
object-pointing detectors backed by RoboRefer [9] and 134
Molmo [3], respectively. `depth_estimator` predicts 135
monocular depth and reconstructs 3D point clouds using 136
`DepthPro` [1]. `compute_bbox` estimates 3D bounding 137
boxes and object poses from reconstructed geometry, while 138
`compute_grasp` predicts collision-free grasp poses for 139
robotic manipulation. Finally, `code_executor` allows 140
the VLM to execute small Python snippets for orchestrat- 141
ing multi-tool workflows, returning results with captured 142
`stdout/stderr` and optionally caching intermediate out- 143
puts for reuse. 144

Robotic tools. We integrate a set of robotic tools 145
that enable embodied perception and manipulation. 146
`capture_image` captures RGB observations from the 147
robot’s onboard camera and stores them for subsequent 148
visual processing. `capture_depth` acquire depth in- 149
formation from the scene, returning a depth map with 150
focal length and a full 3D point cloud reconstruction. 151
`execute_grasp` executes a grasp given a 4×4 transfor- 152
mation matrix representing the target grasp pose and reports 153
execution success and timing feedback. `place_object` 154
places an object at a specified 2D image coordinate, con- 155
firming successful placement in the returned text message. 156
Apart from the above real robotic tools that control real- 157
world robot arms, we also provide a set of `mock_robot` 158
tools without relying on real robots for the ease of data ge- 159
neration and training. Together, these tools provide a physi- 160
cal interface that allows the VLM to perceive, reason, and 161
act within the real world, enabling unified spatial reasoning 162
and robotic control. 163

B.3. Example Process of Launching Tool IRL 164

We show a condensed workflow of launching tools and in- 165
teractive reinforcement learning via a single bash script in 166
Listing 1. 167

Listing 1. Example workflow for launching Toolshed and running 168
VERL GRPO training with tool calling enabled. 169

```
170 # -----  
171 # Graceful cleanup (kills the Toolshed actor  
172 # process so detached actors vanish)  
173 # -----  
174 TOOLSHED_PID="" 175
```

```

174 cleanup() {
175   echo "Cleaning up"
176   if [[ -n "$TOOLSHED_PID" ]]; then
177     echo "Killing(PID=$TOOLSHED_PID)"
178     kill $TOOLSHED_PID 2>/dev/null
179     wait $TOOLSHED_PID 2>/dev/null
180     echo "Stopped."
181   fi
182   exit 0
183 }
184 trap cleanup SIGINT SIGTERM EXIT
185
186 # 1. Launch Toolshed with GPU-backed vision
187   tools
188 python - <<'PY'
189 import ray
190 from toolshed import start_toolkit
191
192 ray.init(address='auto')
193
194 tool_configs = {
195   "point1": {"num_actors": 2, "resources": {"
196     num_gpus": 0.5}},
197   "point2": {"num_actors": 2, "resources": {"
198     num_gpus": 0.5}},
199   "sam2": {"num_actors": 4, "resources": {"
200     num_gpus": 0.2}},
201   "depth_estimator": {"num_actors": 4, "
202     resources": {"num_gpus": 0.2}}
203   ...
204 }
205
206 pg = ray.util.placement_group([{"CPU": 8, "GPU
207   ": 8}], strategy="STRICT_PACK")
208 ray.get(pg.ready())
209 router = start_toolkit(tool_configs, dashboard
210   =True, placement_group=pg)
211 print("Started:", list(tool_configs))
212 PY
213
214 # 2. Generate a YAML tool config and make sure
215   tools are ready
216 python generate_toolshed_config.py --output
217   toolshed_config.yaml
218
219 # 3. Launch VERL GRPO training with Toolshed
220   Integration
221 python -m verl.trainer.main_ppo \
222   actor_rollout_ref.model.path=Qwen/Qwen2.5-
223   VL-3B-Instruct \
224   ...
225   actor_rollout_ref.rollout.multi_turn.
226   tool_config_path=toolshed_config.yaml
227
228 echo "Finished."

```

B.4. Example Toolshed Integration with Proprietary Models

Figure 2 illustrates our integration of Toolshed with a proprietary model (Claude Sonnet 4.5), enabling seamless interactive communication among the user, external tools, and the VLM.

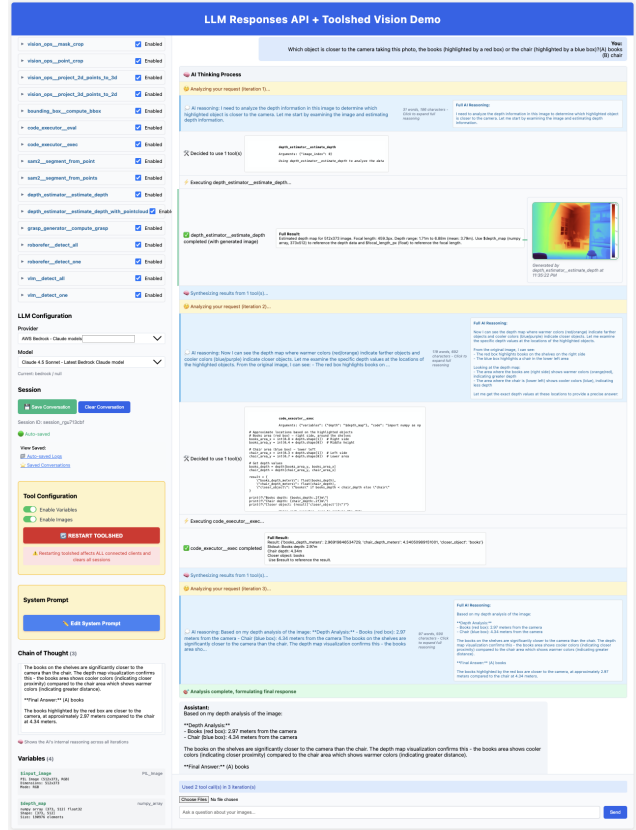


Figure 2. Interactive web demo illustrating Claude’s tool-augmented reasoning when integrated with Toolshed.

B.5. Detailed Tool APIs

Complete API of the computer vision and robotic tools supported by Toolshed include:

image_ops.point_crop(data, x, y)

Purpose: Get the data value in the numpy ndarray ‘data’ at the given coordinate.

- Inputs:**
 - data: Numpy ndarray of shape (H, W) or (H, W, C), or PIL Image
 - x: Normalized x-coordinate, float in [0, 1]
 - y: Normalized y-coordinate, float in [0, 1]
- Outputs:**
 - **Raw value:** data value indexed at the input coordinate.
 - **Text:** Reports the information about the pixel value at the given coordinates.

image_ops.point_crop(image, points)

Crop image to minimally encompass all given points.

- Inputs:**
 - image: PIL.Image

| | | | |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 256 | – <code>points</code> : list of (x, y) normalized floats in $[0, 1]$ | – Image : Original image with semi-transparent green mask, a white mask outline, and red circular markers at all provided points. | 305 |
| 257 | • Outputs : | – Variables : <code>segmentation_mask</code> (boolean $H \times W$ <code>numpy.ndarray</code>) | 306 |
| 258 | – Raw value : <code>PIL.Image</code> cropped to the bounding box of the points | | 307 |
| 259 | – Text : Reports the crop box, size, and number of points | | 308 |
| 260 | – Image : The cropped region only (no overlay) | | 309 |
| 261 | – Variables : <code>cropped_image</code> (<code>PIL.Image</code>) | | |
| 262 | | point1.detect_one(image, obj_name) | 310 |
| 263 | | Identify one instance of the named object by pointing to them with Roborefer. | 311 |
| 264 | image_ops.mask_crop(image, mask) | | 312 |
| 265 | Crop to the tight bounding box of a boolean mask; outside-mask pixels are set to white. | • Inputs : | 313 |
| 266 | | – <code>image</code> : <code>PIL.Image</code> | 314 |
| 267 | • Inputs : | – <code>obj_name</code> : string | 315 |
| 268 | – <code>image</code> : <code>PIL.Image</code> | • Outputs : | 316 |
| 269 | – <code>mask</code> : boolean <code>numpy.ndarray</code> of shape $H \times W$ matching the image | – Raw value : String serialization of a normalized point coordinate $(x, y) \in [0, 1]^2$ | 317 |
| 270 | • Outputs : | – Text : Reports the object name, count, and the normalized point | 318 |
| 271 | – Raw value : <code>PIL.Image</code> of the masked region on white background, cropped to mask bounds | – Image : Original image with red circular point markers at detected locations (white outlines) | 319 |
| 272 | – Text : Reports crop box, size, and mask coverage percentage | – Variables : <code><obj_name>_detection</code> ((x, y) floats in $[0, 1]$; spaces in <code>obj_name</code> replaced with underscores) | 320 |
| 273 | – Image : Masked crop on white background (no overlay) | | 321 |
| 274 | – Variables : <code>masked_crop</code> (<code>PIL.Image</code>) | | 322 |
| 275 | | point1.detect_all(image, obj_name) | 323 |
| 276 | | Identify instances of the named object by pointing to them with Roborefer. | 324 |
| 277 | | | 325 |
| 278 | | • Inputs : | 326 |
| 279 | sam2.segment_from_point(image, x, y) | – <code>image</code> : <code>PIL.Image</code> | 327 |
| 280 | Segment the object at a single pixel coordinate. | – <code>obj_name</code> : string | 328 |
| 281 | • Inputs : | • Outputs : | 329 |
| 282 | – <code>image</code> : <code>PIL.Image</code> | – Raw value : String serialization of a list of normalized point coordinates $(x, y) \in [0, 1]^2$ | 330 |
| 283 | – <code>x</code> : Normalized x-coordinate, float in $[0, 1]$ | – Text : Reports the object name, count, and the list of normalized points | 331 |
| 284 | – <code>y</code> : Normalized y-coordinate, float in $[0, 1]$ | – Image : Original image with red circular point markers at detected locations (white outlines) | 332 |
| 285 | • Outputs : | – Variables : <code><obj_name>_detections</code> (list of (x, y) floats in $[0, 1]$; spaces in <code>obj_name</code> replaced with underscores) | 333 |
| 286 | – Raw value : dict with <code>mask</code> (boolean $H \times W$ <code>numpy.ndarray</code>) and <code>iou_score</code> (float) | | 334 |
| 287 | – Text : Reports the click location and IoU score | | 335 |
| 288 | – Image : Original image with semi-transparent green mask, a white mask outline, and a red circular point marker (white outline) | | 336 |
| 289 | – Variables : <code>segmentation_mask</code> (boolean $H \times W$ <code>numpy.ndarray</code>) | | 337 |
| 290 | | point2.detect_one(image, obj_name) | 338 |
| 291 | | Identify one instance of the named object by pointing to them with Molmo. | 339 |
| 292 | | | 340 |
| 293 | | • Inputs : | 341 |
| 294 | sam2.segment_from_points(image, points) | – <code>image</code> : <code>PIL.Image</code> | 342 |
| 295 | Segment an object using multiple foreground points. | – <code>obj_name</code> : string | 343 |
| 296 | • Inputs : | • Outputs : | 344 |
| 297 | – <code>image</code> : <code>PIL.Image</code> | – Raw value : String serialization of a normalized point coordinate $(x, y) \in [0, 1]^2$ | 345 |
| 298 | – <code>points</code> : list of (x, y) normalized floats in $[0, 1]$ | – Text : Reports the object name, count, and the normalized point | 346 |
| 299 | • Outputs : | – Image : Original image with red circular point markers at detected locations (white outlines) | 347 |
| 300 | – Raw value : dict with <code>mask</code> (boolean $H \times W$ <code>numpy.ndarray</code>) and <code>iou_scores</code> (1-D <code>numpy.ndarray</code>) | | 348 |
| 301 | – Text : Reports the number of points and the best IoU score | | 349 |
| 302 | | | 350 |
| 303 | | | 351 |
| 304 | | | 352 |
| | | | 353 |
| | | | 354 |

| | | | |
|-----|-------------------------------------------------------------------------|------------------------------------------------------------------------------|-----|
| 355 | – Variables: <code><obj_name>_detection</code> ((x, y) | – Variables: <code>depth_map</code> ($H \times W$ float array), | 404 |
| 356 | floats in $[0, 1]$; spaces in <code>obj_name</code> replaced with | <code>point_cloud</code> ($N \times 3$ float array of 3D points in | 405 |
| 357 | underscores) | camera coordinates), <code>focal_length_px</code> (float) | 406 |
| 358 | point2.detect_all (<code>image</code> , <code>obj_name</code>) | grasp_generator.compute_grasp (<code>point_cloud</code> , | 407 |
| 359 | Identify instances of the named object by pointing to | mask , <code>image</code> , <code>focal_length_px</code>) | 408 |
| 360 | them with Molmo. | Generate a single grasp pose for a masked subset of a | 409 |
| 361 | • Inputs: | point cloud with GraspGen. | 410 |
| 362 | – <code>image</code> : PIL.Image | • Inputs: | 411 |
| 363 | – <code>obj_name</code> : string | – <code>point_cloud</code> : $N \times 3$ numpy float array | 412 |
| 364 | • Outputs: | – <code>mask</code> : Boolean segmentation mask | 413 |
| 365 | – Raw value: String serialization of a list of nor- | – <code>image</code> : PIL.Image | 414 |
| 366 | malized point coordinates $(x, y) \in [0, 1]^2$ | – <code>focal_length_px</code> : float | 415 |
| 367 | – Text: Reports the object name, count, and the list | • Outputs: | 416 |
| 368 | of normalized points | – Raw value: Collision-free grasp pose, and | 417 |
| 369 | – Image: Original image with red circular point | collision-free confidence | 418 |
| 370 | markers at detected locations (white outlines) | – Text: Reports the collision-free grasp confidence, | 419 |
| 371 | – Variables: <code><obj_name>_detections</code> (list of | the total number of generated grasps and the per- | 420 |
| 372 | (x, y) floats in $[0, 1]$; spaces in <code>obj_name</code> replaced | centage of collision-free grasps, and the projected | 421 |
| 373 | with underscores) | 2D gripper points of the best grasp pose in nor- | 422 |
| 374 | depth_estimator.estimate_depth (<code>image</code>) | malized coordinates | 423 |
| 375 | Monocular depth estimation with DepthPro. | – Image: Original image overlaid with projected | 424 |
| 376 | • Inputs: | X-(red), Y-(green), Z-(blue) gripper axes | 425 |
| 377 | – <code>image</code> : PIL.Image | – Variables: <code>grasp_pose</code> (4×4 ndarray, OpenCV | 426 |
| 378 | • Outputs: | camera frame) | 427 |
| 379 | – Raw value: dict with <code>depth_map</code> ($H \times W$ | 3d_bbox.compute_bbox (<code>point_cloud</code> , <code>mask</code> , | 428 |
| 380 | float array, meters), <code>focal_length_px</code> (float), | focal_length_px) | 429 |
| 381 | <code>width</code> (int), <code>height</code> (int) | Compute an oriented bounding box for a masked | 430 |
| 382 | – Text: Reports image size, focal length, and depth | subset of a point cloud. | 431 |
| 383 | range statistics | • Inputs: | 432 |
| 384 | – Image: Colorized depth map with a vertical scale | – <code>point_cloud</code> : $N \times 3$ numpy float array | 433 |
| 385 | bar on the right labeled “Depth (m)” | – <code>mask</code> : Boolean segmentation mask | 434 |
| 386 | – Variables: <code>depth_map</code> ($H \times W$ float array), | – <code>focal_length_px</code> : float | 435 |
| 387 | <code>focal_length_px</code> (float) | • Outputs: | 436 |
| 388 | depth_estimator. | – Raw value: Box corners in 3D, box corners in | 437 |
| 389 | estimate_depth_with_pointcloud (<code>image</code>) | 2D, edges, and extent | 438 |
| 390 | Monocular depth estimation and 3D point cloud gen- | – Text: Summary containing number of input | 439 |
| 391 | eration with DepthPro. | points, the point coordinates in 3d and 2d, mask | 440 |
| 392 | • Inputs: | shape, box extents, and edges | 441 |
| 393 | – <code>image</code> : PIL.Image | – Image: No image output | 442 |
| 394 | • Outputs: | – Variables: <code>obb_corners_3d</code> (8×3 list | 443 |
| 395 | – Raw value: dict with <code>depth_map</code> ($H \times W$ | of lists, meters in opencv camera frame), | 444 |
| 396 | float array, meters), <code>point_cloud</code> ($N \times 3$ | <code>obb_corners_2d</code> (8×2 list of lists, normalized | 445 |
| 397 | float array of 3D points in camera coordi- | image coordinates), <code>extent</code> (3-element ndarray, | 446 |
| 398 | nates), <code>focal_length_px</code> (float), <code>width</code> (int), | <code>extent</code> (list of pairs of integers, edges | 447 |
| 399 | <code>height</code> (int) | (list of pairs of integers, edges of the bounding | 448 |
| 400 | – Text: Reports image size, focal length, and depth | box defined by the indices of the corners) | 449 |
| 401 | range statistics | code_executor.exec (<code>code</code>) | 450 |
| 402 | – Image: Colorized depth map with a vertical scale | Execute a multi-line Python block (imports limited to | 451 |
| 403 | bar on the right labeled “Depth (m)” | <code>math</code> and <code>numpy</code>). | 452 |
| | | • Inputs: | 453 |

| | | | |
|-----|-----------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|-----|
| 454 | – code: string | mock_robot . | 503 |
| 455 | • Outputs: | get_depth_with_pointcloud(mock_data) | 504 |
| 456 | – Raw value: tuple (<i>result, stdout, stderr</i>) | Return the mock depth and and point cloud generation | 505 |
| 457 | – Text: Summarizes the result, captured stdout, and | from the dataset without a real robot. | 506 |
| 458 | stderr; notes a stored variable if applicable | | |
| 459 | – Image: No image output | • Inputs: | 507 |
| 460 | – Variables: <code>last_exec_result</code> (present iff a | – <code>mock_data</code> : dict with <code>mock_image</code> | 508 |
| 461 | non-None result and variables are enabled) | (<code>PIL Image</code>), <code>mock_depth_map</code> (<code>numpy</code> | 509 |
| | | <code>array</code>), <code>mock_point_cloud</code> (<code>numpy ar-</code> | 510 |
| | | <code>ray</code>), <code>mock_focal_length_px</code> (<code>float</code>), | 511 |
| 462 | code_executor.eval(expression) | <code>image_width</code> (<code>int</code>), <code>image_height</code> (<code>int</code>). | 512 |
| 463 | Evaluate a single Python expression. | • Outputs: | 513 |
| 464 | • Inputs: | – Raw value: <code>image</code> (<code>PIL Image</code>), <code>depth_map</code> | 514 |
| 465 | – <code>expression</code> : <code>string</code> | (<code>numpy array</code>), <code>mock_point_cloud</code> (<code>numpy ar-</code> | 515 |
| 466 | • Outputs: | <code>ray</code>), <code>focal_length_px</code> (<code>float</code>), <code>width</code> (<code>int</code>), | 516 |
| 467 | – Raw value: tuple (<i>result, stdout, stderr</i>) | <code>height</code> (<code>int</code>). | 517 |
| 468 | – Text: Summarizes the result, captured stdout, and | – Text: Summary of depth data and and point | 518 |
| 469 | stderr; notes a stored variable if applicable | cloud generation including image dimensions, fo- | 519 |
| 470 | – Image: No image output | cal length, depth statistics, and point cloud size | 520 |
| 471 | – Variables: <code>last_eval_result</code> (present iff a | – Image: A colored depth map visualization | 521 |
| 472 | non-None result and variables are enabled) | where closer objects appear cooler (blue/purple) | 522 |
| | | and distant objects appear warmer (red/yellow) | 523 |
| 473 | mock_robot.capture_image(mock_data) | – Variables: <code>depth_map</code> (2D <code>numpy array</code> of | 524 |
| 474 | Return the mock image from the dataset without a real | depth values in meters), <code>point_cloud</code> ($N \times 3$ | 525 |
| 475 | robot. | array of 3D points), <code>foca_length_px</code> (<code>float</code> , es- | 526 |
| | | timated focal length in pixels) | 527 |
| 476 | • Inputs: | mock_robot.execute_grasp(grasp_pose) | 528 |
| 477 | – <code>mock_data</code> : dict with <code>mock_image</code> | Simulate executing a grasp (always succeeds). | 529 |
| 478 | • Outputs: | | |
| 479 | – Raw value: Image from mock camera | • Inputs: | 530 |
| 480 | – Text: Image dimensions and capture status | – <code>grasp_pose</code> : 4×4 transformation matrix rep- | 531 |
| 481 | – Image: Captured image from mock camera | resenting the grasp pose in the robot’s camera | 532 |
| 482 | – Variables: <code>captured_image</code> (<code>PIL Image</code>) | frame, <code>OpenCV</code> convention | 533 |
| | | • Outputs: | 534 |
| 483 | mock_robot.get_depth(mock_data) | – Raw value: <code>success</code> (<code>boolean</code>), | 535 |
| 484 | Return the mock depth from the dataset without a real | <code>execution_time_s</code> (<code>float</code>) | 536 |
| 485 | robot. | – Text: Confirmation that grasp was successful | 537 |
| | | – Image: No image output | 538 |
| 486 | • Inputs: | – Variables: No variable output | 539 |
| 487 | – <code>mock_data</code> : dict with <code>mock_image</code> | mock_robot . | 540 |
| 488 | (<code>PIL Image</code>), <code>mock_depth_map</code> (<code>numpy</code> | place_object_at_2d_location(point_2d) | 541 |
| 489 | <code>array</code>), <code>mock_focal_length_px</code> (<code>float</code>), | Simulate placing object at 2D location (always suc- | 542 |
| 490 | <code>image_width</code> (<code>int</code>), <code>image_height</code> (<code>int</code>) | ceeds). | 543 |
| 491 | • Outputs: | | |
| 492 | – Raw value: <code>image</code> (<code>PIL Image</code>), <code>depth_map</code> | • Inputs: | 544 |
| 493 | (<code>numpy array</code>), <code>focal_length_px</code> (<code>float</code>), | – <code>point_2d</code> : 2D normalized image coordinate | 545 |
| 494 | <code>width</code> (<code>int</code>), <code>height</code> (<code>int</code>) | where the object should be placed | 546 |
| 495 | – Text: Summary of depth data including image di- | • Outputs: | 547 |
| 496 | mensions, focal length, and depth statistics | – Raw value: <code>success</code> (<code>boolean</code>), | 548 |
| 497 | – Image: A colored depth map visualization | <code>execution_time_s</code> (<code>float</code>) | 549 |
| 498 | where closer objects appear cooler (blue/purple) | – Text: Confirmation that placement was successful | 550 |
| 499 | and distant objects appear warmer (red/yellow) | – Image: No image output | 551 |
| 500 | – Variables: <code>depth_map</code> (2D <code>numpy array</code> of | – Variables: No variable output | 552 |
| 501 | depth values in meters), <code>focal_length_px</code> | | |
| 502 | (<code>float</code> , estimated focal length in pixels) | | |

| | | | | |
|-----|----------------------------------------------------------------------------|------------------------------------------------------------------|--|-----|
| 553 | <code>mock_robot .</code> | | | 603 |
| 554 | <code>place_object_at_3d_location (point_3d)</code> | | | 604 |
| 555 | Simulate placing object at 3D location (always suc- | | | 605 |
| 556 | ceeds). | | | 606 |
| 557 | • Inputs: | | | 607 |
| 558 | – <code>point_3d</code> : 3D point in the robot’s camera frame | | | 608 |
| 559 | (list or numpy array) where the object should be | | | 609 |
| 560 | placed | | | 610 |
| 561 | • Outputs: | | | 611 |
| 562 | – Raw value: <code>success</code> (boolean), | | | 612 |
| 563 | <code>execution_time_s</code> (float) | | | |
| 564 | – Text: Confirmation that placement was successful | | | |
| 565 | – Image: No image output | | | |
| 566 | – Variables: No variable output | | | |
| 567 | <code>robot .capture_image ()</code> | | | |
| 568 | Return the mock image from the dataset without a real | | | |
| 569 | robot. | | | |
| 570 | • Inputs: No input required | | | |
| 571 | • Outputs: | | | |
| 572 | – Raw value: <code>image_shape</code> (numpy array or | | | |
| 573 | list), <code>image</code> (PIL Image) | | | |
| 574 | – Text: Image dimensions and capture status | | | |
| 575 | – Image: Captured image from robot camera | | | |
| 576 | – Variables: <code>captured_image</code> (PIL Image) | | | |
| 577 | <code>robot .get_depth ()</code> | | | |
| 578 | Retrieve depth map from the robot’s depth sensor. | | | |
| 579 | • Inputs: No input required | | | |
| 580 | • Outputs: | | | |
| 581 | – Raw value: <code>depth_map</code> (numpy array), | | | |
| 582 | <code>depth_map_visualization</code> (PIL Image), | | | |
| 583 | <code>focal_length_px</code> (float), <code>width</code> (int), | | | |
| 584 | <code>height</code> (int) | | | |
| 585 | – Text: Summary of depth data including image di- | | | |
| 586 | mensions, focal length, and depth statistics | | | |
| 587 | – Image: A colorized depth map visualization | | | |
| 588 | where closer objects appear cooler (blue/purple) | | | |
| 589 | and distant objects appear warmer (red/yellow) | | | |
| 590 | – Variables: <code>depth_map</code> (2D numpy array of | | | |
| 591 | depth values in meters), <code>focal_length_px</code> | | | |
| 592 | (float, estimated focal length in pixels) | | | |
| 593 | <code>robot .</code> | | | |
| 594 | <code>get_depth_with_pointcloud ()</code> | | | |
| 595 | Retrieve depth map from robot’s depth sensor and gen- | | | |
| 596 | erate 3D point cloud. | | | |
| 597 | • Inputs: No input required | | | |
| 598 | • Outputs: | | | |
| 599 | – Raw value: <code>image</code> (PIL Image), <code>depth_map</code> | | | |
| 600 | (numpy array), <code>point_cloud</code> (numpy ar- | | | |
| 601 | ray), <code>focal_length_px</code> (float), <code>width</code> (int), | | | |
| 602 | <code>height</code> (int). | | | |
| | | | | |
| | | – Text: Summary of depth data and and point | | |
| | | cloud generation including image dimensions, fo- | | |
| | | cal length, depth statistics, and point cloud size | | |
| | | – Image: A colorized depth map visualization | | |
| | | where closer objects appear cooler (blue/purple) | | |
| | | and distant objects appear warmer (red/yellow) | | |
| | | – Variables: <code>depth_map</code> (2D numpy array of | | |
| | | depth values in meters), <code>point_cloud</code> ($N \times 3$ | | |
| | | array of 3D points), <code>focal_length_px</code> (float, es- | | |
| | | timated focal length in pixels) | | |
| | <code>robot .execute_grasp (grasp_pose)</code> | | | |
| | Execute a grasp by moving the robot to the specified | | | |
| | pose via a pre-grasp point, and closing the gripper. | | | |
| | • Inputs: | | | |
| | – <code>grasp_pose</code> : 4×4 transformation matrix rep- | | | |
| | resenting the grasp pose in the robot’s camera | | | |
| | frame, OpenCV convention | | | |
| | • Outputs: | | | |
| | – Raw value: <code>success</code> (boolean), | | | |
| | <code>execution_time_s</code> (float), <code>image</code> (PIL | | | |
| | Image) | | | |
| | – Text: Status of the grasp execution | | | |
| | – Image: View from robot camera after the grasp is | | | |
| | executed | | | |
| | – Variables: <code>captured_image</code> after the grasp is | | | |
| | executed | | | |
| | <code>robot .</code> | | | |
| | <code>place_object_at_2d_location (point_2d)</code> | | | |
| | Move the robot to a place it’s currently held object | | | |
| | based on a 2D normalized image coordinate. The tool | | | |
| | will convert to a 3D placement location automatically | | | |
| | by shooting a ray. | | | |
| | • Inputs: | | | |
| | – <code>point_2d</code> : 2D normalized image coordinate | | | |
| | where the object should be placed | | | |
| | • Outputs: | | | |
| | – Raw value: <code>success</code> (boolean), | | | |
| | <code>execution_time_s</code> (float), <code>image</code> (PIL | | | |
| | Image) | | | |
| | – Text: Status of the release operation | | | |
| | – Image: View from robot camera after the place- | | | |
| | ment is executed | | | |
| | – Variables: <code>captured_image</code> (PIL Image) after | | | |
| | the placement is executed | | | |
| | <code>robot .</code> | | | |
| | <code>place_object_at_3d_location (point_3d)</code> | | | |
| | Move the robot to a 3D placement point and open the | | | |
| | gripper to place the object. | | | |
| | • Inputs: | | | |

- 652 – point_3d: 3D point in the robot’s camera frame
- 653 (list or numpy array) where the object should be
- 654 placed
- 655 • **Outputs:**
- 656 – **Raw value:** success (boolean),
- 657 execution_time_s (float), image (PIL
- 658 Image)
- 659 – **Text:** Status of the placement operation
- 660 – **Image:** View from robot camera after the place-
- 661 ment is executed
- 662 – **Variables:** captured_image (PIL Image) after
- 663 the placement is executed

664 C. Additional Method Details

665 C.1. Group Relative Policy Optimization

666 We employ Group Relative Policy Optimization
667 (GRPO) [5] as our RL training algorithm. We present
668 the details of GRPO below.

669 For each input \mathcal{I} , in total N rollout procedures are
670 launched asynchronously under the current policy π_θ . Each
671 rollout generates in total N multi-turn rollouts $O_1, O_2, \dots,$
672 O_N . Their rewards are calculated as r_1, r_2, \dots, r_N . Each
673 r_i is standardized into a relative advantage A_i via group
674 computation:

$$675 A_i = \frac{r_i - \text{mean}(\{r_1, r_2, \dots, r_N\})}{\text{std}(\{r_1, r_2, \dots, r_N\})}. \quad (1)$$

676 The policy is then optimized by minimizing the GRPO loss:

$$677 \mathcal{L}_{\text{GRPO}}(\theta) = \mathbb{E}_i \left[-\min(\rho_i A_i, \text{clip}(\rho_i, 1-\epsilon, 1+\epsilon) A_i) \right. \\ \left. + \beta \text{KL}(\pi_\theta \parallel \pi_{\text{ref}}) \right], \quad (2)$$

678 where $\rho_i = \frac{\pi_\theta(i)}{\pi_{\text{ref}}(i)}$, and π_{ref} denotes the reference policy
679 model, i.e., the VLM trained after stage 1. Here, ϵ and β
680 are tunable hyperparameters controlling the clipping range
681 and KL regularization strength. This formulation encour-
682 ages the policy to increase the probability of high-reward
683 responses while maintaining stability through KL regular-
684 ization.

685 C.2. Alternative Reward Design

686 **Other reward for the pointing questions.** Considering
687 that pointing is the first step of solving many spatial reason-
688 ing tasks or using other tools, we have experimented with
689 several different rewards for pointing before finalizing the
690 NNDC reward. We show results in Appendix E.2, empha-
691 sizing the importance the reward design for tasks requiring
692 explicit numerical estimation.

- 693 • **Binary:**

$$694 R_B = \begin{cases} 1.0, & \text{if the predicted point lies within} \\ & \text{the ground truth convex hull,} \\ 0, & \text{otherwise.} \end{cases}$$

System Prompt

You are an expert in 3D spatial reasoning for robotics.
Given a spatial reasoning task, **follow this process:**
1. First, think about the reasoning process as an internal monologue the first time you receive the question, and every time you receive new information. Your reasoning process MUST be enclosed within <think></think> tags.
2. After thinking, if you need additional information to answer the question, or conduct external control, call the appropriate tool.
3. When you receive a tool response, use that information to continue your analysis.
4. Once no further visual analysis or tool calls are needed, you **MUST** provide your final answer inside <answer></answer> tags without detailed illustrations.
Example answer format: <answer><Your final answer here></answer>
Tools
You may call one or more functions to assist with the user query.
You are provided with function signatures within <tools></tools> tags:
<tools> {tools} </tools>
For each function call, return a JSON object with function name and arguments within <tool_call></tool_call> tags:
<tool_call> {"name": <func-name>, "arguments": <args-json-object>} </tool_call>

Figure 3. **System prompt.** Instructional prompt guiding the model’s reasoning, tool-call, and answer process.

- 695 • **Normalized Signed Distance to Hull (NSDH):**

$$696 R_{\text{NSDH}} = \begin{cases} 0.5 + 0.5 \exp(s), & \text{if } s \leq 0, \\ 0.5 \exp(-s), & \text{if } s > 0, \end{cases}$$

697 where s is the signed distance from the predicted point to
698 the convex hull boundary (negative inside, positive out-
699 side).

- 700 • **Normalized Area Change (NAC):** Let ΔA be the
701 change in area after adding the predicted point to the con-
702 vex hull, and A_0 be the original area. Then:

$$703 R_{\text{NAC}} = \exp\left(-\frac{\Delta A}{A_0}\right).$$

704 Similar to NNDC, by default, we also apply clipping with
705 the binary accuracy term to emphasize precision for alter-
706 native non-binary rewards. (e.g., $R = \max(R_{\text{NSDH}}, R_B)$)

707 **Format score details.** In addition to task-specific re-
708 wards, we explored defining a *format score* to enforce
709 the structural correctness of model outputs (as defined in
710 the system prompt shown in Figure 3), but did not use
711 it in the final training. The format score verifies that
712 every <tool_call> tag is immediately preceded by a
713 <think> tag, and that the single final <answer> is also
714 directly preceded by a <think>. The output must contain
715 exactly one <answer> tag, positioned at the end of the
716 response. Additional optional constraints can be applied,
717 such as requiring at least one <tool_call> in the output.
718 Predictions failing to meet these structural requirements re-
719 ceive a format score of zero, while perfectly formatted pre-
720 dictions receive a score of one.

721 The final reward is computed as a weighted sum of the
722 accuracy-based reward and the format score:

$$723 R_{\text{final}} = R_{\text{acc}} + \lambda R_{\text{format}},$$

724 where R_{acc} denotes the task-specific accuracy reward,
725 $R_{\text{format}} \in \{0, 1\}$ is the format score (equal to 1.0 if all struc-
726 tural criteria described above are satisfied, and 0 otherwise),
727 and $\lambda \in [0, 1]$ is a scalar weight controlling the influence of
728 the format score. Following prior work, λ is often set to 0.3.

729 D. Additional Implementation Details

730 D.1. More Training Details

731 For Phase-1 SFT of the base model and Phase-2 IRL, we
732 use the system prompt shown in Figure 3. To improve the
733 effectiveness of Phase-1 IRL training for the teacher model,
734 we augment the prompt with two tool-use examples: one
735 demonstrating how to solve a spatial relationship problem
736 using the pointing tool, and another illustrating 2D bound-
737 ing box estimation with the same tool. Moreover, when
738 generating teaching data with the Claude Sonnet 4.5 model,
739 we include additional instructional prompts that encourage
740 careful interpretation of tool outputs and help understanding
741 of image coordinate systems.

742 Due to the substantial latency of robot-in-the-loop train-
743 ing and data collection, we construct grasp and place data
744 using the HOPE dataset and provide it to Claude Sonnet
745 4.5 alongside mock robot tools to generate the robotic por-
746 tion of the teaching dataset. The interactive learning stages
747 themselves do not incorporate this synthetic robot-tool com-
748 ponent; instead, they focus solely on spatial reasoning with
749 vision tools.

750 Another practical consideration is ensuring balanced answer
751 distributions for multiple-choice questions. For exam-
752 ple, the original RoboSpatial VQA dataset contains more
753 than 75% “no” answers, which biases the VLM toward pre-
754 dicting “no” during both SFT and IRL. We found rebalanc-
755 ing the data mitigates this issue and improves answer cali-
756 bration across tasks.

757 Finally, the major hyperparameters used during training
758 are summarized in Table 1. Through our experiments, we
759 find that the KL coefficient is a critical hyperparameter: a
760 relatively small KL value is necessary to encourage suf-
761 ficient exploration during RL. However, this introduces a
762 trade-off in training stability—specifically, we observe an
763 initial drop in rewards during Phase-1 IRL when using a
764 smaller KL coefficient. We experimented with format re-
765 wards, format penalties, alternative KL loss formulations,
766 and related variants, but were unable to eliminate this ef-
767 fect, suggesting that further investigation is needed.

768 D.2. MACE Metric for Grasp Affordances

769 The grasp estimation task requires the model to predict five
770 key points in normalized image coordinates: the grasp cen-
771 ter, left finger base, right finger base, left finger tip, and
772 right finger tip. From these points, we define four finger di-
773 rection vectors: grasp center \rightarrow left finger base, grasp center

Table 1. Training configurations for Phase-1 IRL, Phase-1 SFT, and Phase-2 IRL. A dash (–) indicates that the setting is not applicable to that phase.

| | Phase-1 IRL | Phase-1 SFT | Phase-2 IRL |
|---------------------|--------------------------------------------------------------|-------------------|-------------|
| <i>Data</i> | | | |
| Dataset | Direct VQA | Teaching tool-use | Direct VQA |
| #Samples | 4k | 8k | ≈8k |
| <i>Model</i> | | | |
| Trainable Part | Language model (LLM) only; vision encoder + projector frozen | | |
| #Tunable Parameters | 2.55B | | |
| <i>Training</i> | | | |
| Batch Size | 64 | 8 | 64 |
| Learning Rate | 1e-6 | 1e-5 | 1e-6 |
| Epoch | 5 | 2 | 2 |
| Warmup Ratio | 0.0 | 0.1 | 0.0 |
| LR Schedule | NA | cosine | NA |
| KL Coefficient | 1e-4 | – | 1e-4 |
| Entropy Coefficient | 0.0 | – | 0.0 |
| Temperature | 1.0 | – | 1.0 |
| Max Prompt Length | 8192 | 8192 | 8192 |
| Max Response Length | 8192 | 8192 | 8192 |
| Rollout Number | 5 | – | 5 |
| #GPU (VLM) | 8 | 8 | 8 |
| #GPU (Tools) | 8 | – | 8 |

774 \rightarrow right finger base, left finger base \rightarrow left finger tip, and
775 right finger base \rightarrow right finger tip.

776 The Mean Angular Coordinate Error (MACE) metric is
777 defined as follows. Given the predicted and ground-truth
778 grasp centers \hat{c} and c , and the set of four corresponding fin-
779 ger direction vectors $\{\hat{r}_k\}_{k=1}^4$ and $\{r_k\}_{k=1}^4$, we define:

$$780 \text{MACE} = 1 - \frac{1}{2} \left(\frac{\|\hat{c} - c\|_2}{w} + \frac{1}{4} \sum_{k=1}^4 \frac{1 - \cos(\hat{r}_k, r_k)}{2} \right), \quad (3)$$

781 where w denotes the gripper width used for spatial normal-
782 ization, and $\cos(\hat{r}_k, r_k) = \frac{\hat{r}_k \cdot r_k}{\|\hat{r}_k\| \|r_k\|}$ represents the cosine
783 similarity between the predicted and ground-truth orienta-
784 tions of the k -th finger-related vector.

785 D.3. Robot Manipulation Setup

786 **Robot System** We conduct robot experiments on a
787 Kinova Jaco arm equipped with the CuRobo [6] mo-
788 tion planner and a ZED2 RGB-D camera. We ex-
789 pose the robot as a tool and make it available to the
790 VLM. The tool has functions: `capture_image` retrieves
791 the current RGB image from the camera, `get_depth`
792 and `get_depth_with_pointcloud` retrieve the cur-
793 rent depth image, optionally with a pointcloud in the
794 robot frame, `execute_grasp` moves the end-effector
795 to a specified grasp pose via a pre-grasp point and
796 closes the gripper, `place_object_at_2d_location`
797 and `place_object_at_3d_location` offer two ways
798 to parameterize a place operation that moves the robot hand
799 holding an object over a location in the scene and opens the
800 gripper. All motions are executed with the motion planner.

| Task | Ours | Claude | GPT |
|----------------------------------------------------------------------------|------|--------|-----|
| Pick | | | |
| Pick up the dark blue object | ✓ | ✓ | ✓ |
| Pick up the soft toy | × | × | × |
| Pick up the solid toy | ✓ | ✓ | × |
| Pick up the tall cylindrical tennis ball container | ✓ | ✓ | ✓ |
| Pick up the coconut water | ✓ | ✓ | ✓ |
| Pick up the plastic bottle | ✓ | ✓ | ✓ |
| Pick up the red box | ✓ | ✓ | ✓ |
| Relational Pick | | | |
| Pick up the far coconut water | ✓ | × | × |
| Pick up the coconut water that is closer to the camera | ✓ | ✓ | × |
| Pick up the left pineapple juice can | ✓ | × | × |
| Pick up the right pineapple juice can | ✓ | ✓ | × |
| Pick up the further purple drink | × | ✓ | ✓ |
| Pick up the near purple bottle | ✓ | × | ✓ |
| Pick & Place | | | |
| Pick up the hot dog and place it in the black bin | 0 | 2 | 2 |
| Pick up the tall cylindrical container and place it in the transparent bin | 2 | 2 | 2 |
| Pick up the leftmost condiment and place it in the transparent bin | 2 | 2 | 2 |
| Pick up the cinnamon and place it in front of the rice box | 2 | 2 | 0 |
| Pick up the rice box and place it next to the hot sauce | 2 | 1 | 0 |
| Pick up the plushie and place it left of the coconut water | 2 | 1 | 1 |
| Pick up the pony and place it left of the two plushies | 2 | 1 | 2 |

Table 2. Per-task breakdown of the real-world manipulation results, comparing **Ours** (SpaceTools), Claude Sonnet 4.5 and GPT-5.

Robot Experiments Tasks and Results We design a suite of tasks across three categories. *Pick*, *Relational Pick*, and *Pick & Place*. We score both *Pick* tasks based on the success rate, and *Pick & Place* based on partial success rate, awarding 1 point each for a correct pick and place operation.

The full results at individual task level are in Table 2, omitting methods that fail to achieve any points. In *Pick up the soft toy* task, all models failed due to a common failure in pointing tool not being able to differentiate the soft toy from a rigid toy. In *Relational Pick* and *Pick & Place* tasks, SpaceTools shows superior ability than Claude and GPT-5 in correctly using the pointing tool to resolve spatial relations, reflecting an understanding of its strengths and limitations likely attributable to the interactive training with the tool.

Additional Details on Robot SFT Data Collection In order to collect SFT data of from the Universal Teacher (Claude Sonnet 4.5) on using the robot tool, we design a “mock robot” that has the same API as the robot tool, but it always simulates successful actions provided the API was called with valid arguments. This allows collecting a small number (~500) examples of valid robot API calls without requiring the physical robot in the loop and ensuring that our robot is unseen during training.

Table 3. Comparison of Qwen2.5-VL-3B, its inference variants, and fine-tuned models on RoboSpatial and RefSpatial. *Direct Inference* refers to answering the question without intermediate reasoning or tool use. *CoT* denotes chain-of-thought inference. *+Toolshed* indicates tool-augmented inference without any additional training. Among all variants, Tool IRL achieves the highest overall accuracy on RoboSpatial and the strongest generalization to RefSpatial.

| Model | RoboSpatial | | | RefSpatial |
|--------------------------------------------|--------------|--------------|--------------|--------------|
| | VQA | Vacant | Overall | |
| <i>Inference Baseline (no fine-tuning)</i> | | | | |
| Direct Inference | 53.07 | 0.00 | 35.71 | 0.00 |
| CoT | 66.67 | 0.00 | 43.71 | 0.00 |
| +Toolshed | 47.37 | 9.02 | 34.00 | 17.69 |
| <i>Fine-tuned on RoboSpatial</i> | | | | |
| Tool-free SFT | 75.88 | 13.11 | 54.00 | 0.00 |
| Tool-free RL | 72.37 | 20.49 | 54.28 | 0.00 |
| Tool IRL | 77.64 | 62.30 | 72.30 | 34.30 |

D.4. Details of Non-interactive RL Baseline

We present the detailed description of the Tool NIRL baseline referenced in the ablation section of the main paper, as space limitations prevented us from including all details there.

We follow the conventional tool-learning setup used in LLMs [8] to perform reinforcement learning of tool usage *without* executing tools during training. The core idea is to compute the reward solely from the correctness of the predicted tool name and its arguments, which requires access to ground-truth tool call traces for supervision. After obtaining tool-augmented reasoning traces from Claude, each multi-turn trace with T turns is decomposed into T single-turn training instances: the i^{th} instance contains the conversation history up to turn i as input, and the corresponding ground-truth output for turn i as the target.

During training, for tool-call turns, we adopt the binary reward used in [8]. A reward of 1.0 is given only when both the tool call format and the tool call content match the ground truth:

$$r = \begin{cases} 1, & \text{if FormatCorrect} \wedge \text{ToolCallMatch} \\ 0, & \text{otherwise,} \end{cases} \quad (4)$$

where `FormatCorrect` verifies that the model output is wrapped in the required tags, and `ToolCallMatch` checks that both the tool name and its arguments exactly match the ground-truth tool call. For final-answer turns (i.e., non-tool turns), we reuse the same task-specific normalized rewards introduced in this paper.

852 **E. Additional Experimental Results**853 **E.1. A Closer Look at Tool IRL Alone.**

854 While direct IRL over diverse tools poses challenges due
 855 to the vast action space, we demonstrate its effectiveness
 856 within a constrained setup using the RoboSpatial dataset
 857 and pointing tools. As shown in Table 3, this approach sub-
 858 stantially improves spatial reasoning compared with both
 859 direct tool-free SFT and vanilla tool-free GRPO baselines
 860 (e.g., classic reasoning models like R1 [2]), as well as other
 861 inference approaches [7]. On RoboSpatial, the IRL with
 862 Tools model achieves 72.3% overall accuracy, outperform-
 863 ing SFT and vanilla GRPO. Notably, IRL with Tools is
 864 the only method that generalizes to unseen tasks: achiev-
 865 ing 34.3% on RefSpatial, whereas other fine-tuning strate-
 866 gies yield zero accuracy. These results show that interactive
 867 tool use during RL enables the model to internalize trans-
 868 ferable geometric reasoning skills beyond the training do-
 869 main. Even without fine-tuning, connecting Toolshed to the
 870 pretrained model yields measurable improvements on Ref-
 871 Spatial, highlighting the intrinsic generalization benefit of
 872 tool-augmented spatial reasoning.

873 **E.2. Other Ablations**

874 **Direct IRL on all tasks with all tools.** As mentioned
 875 in the main paper, directly reinforcement learning with all
 876 tools on all tasks result in a large search space and is hard
 877 to learn effectively. We provide qualitative performance in
 878 Table 4, supporting this argument.

Table 4. Direct IRL on all tasks (*Direct IRL All.*) with all tools compared with our method.

| Variants | IRL-T | Univ-T | S2-IRL | RoboSpatial | RefSpatial | Pose | Mean |
|-------------------|-------|--------|--------|-------------|------------|-------|-------|
| SpaceTools (Ours) | ✓ | ✓ | ✓ | 70.00 | 53.07 | 34.37 | 52.48 |
| Direct IRL All. | ✗ | ✗ | ✓ | 52.86 | 3.25 | 3.26 | 19.79 |

879 **Reward and prompt for IRL.** Due to computational
 880 constraints that prevent running full-scale IRL ablations,
 881 we evaluate different pointing rewards (e.g., NSDH, NAC,
 882 Binary) and prompt design choices introduced in Ap-
 883 pendix C.2 on a subset of 1k vacant-space localization ques-
 884 tions from RoboSpatial using the Molmo pointing tool. The
 885 results in Table 5 show that NNDC without an additional
 886 format reward yields the most stable and reliable learning
 887 behavior. Accordingly, we adopt NNDC (without a format
 888 reward) for all subsequent training stages. These experi-
 889 ments also highlight the importance of normalizing rewards
 890 to the $[0, 1]$ range, a practice we apply consistently across
 891 all tasks. More broadly, this study underscores the richness
 892 of the reward-design space for spatial reasoning tasks, es-
 893 pecially those requiring explicit numerical estimation.

Table 5. Ablation on reward and prompt design for the pointing task as introduced in Appendix C.2. *Norm.* indicates whether normalization to range $[0, 1]$ is applied to the reward function. *Clip.* indicates whether binary clipping is applied. *Format* indicates whether the format reward is applied. *Example in Prompt* indicates whether two tool-use examples are added in the prompt. Checkmarks indicate which components are included for each variant.

| Reward Variant | Norm. | Clip. | Format | Example in Prompt | Acc. |
|--------------------|-------|-------|--------|-------------------|--------------|
| NNDC (Ours) | ✓ | ✓ | ✗ | ✓ | 35.25 |
| w/o Clip. | ✓ | ✗ | ✗ | ✓ | 14.8 |
| w/o Norm. | ✗ | ✓ | ✗ | ✓ | 0.00 |
| w Format. | ✓ | ✓ | ✓ | ✓ | 33.61 |
| w/o Example. | ✓ | ✓ | ✗ | ✗ | 17.21 |
| NSDH | ✓ | ✓ | ✗ | ✓ | 21.31 |
| w/o Clip. | ✓ | ✗ | ✗ | ✓ | 22.31 |
| w/o Norm. | ✗ | ✓ | ✗ | ✓ | 0.00 |
| NAC | ✓ | ✓ | ✗ | ✓ | 22.95 |
| w/o Clip. | ✓ | ✗ | ✗ | ✓ | 22.95 |
| w/o Norm. | ✗ | ✓ | ✗ | ✓ | 0.00 |
| Binary | ✓ | ✓ | ✗ | ✓ | 15.57 |

894 **Dataset size and type for IRL.** We conduct preliminary
 895 experiments on how dataset size and data-type composition
 896 affect IRL performance using the RoboSpatial dataset and
 897 the Roborefer pointing tool. RoboSpatial contains four data
 898 types: configuration, compatibility, grounding, and vacant-
 899 space localization. Configuration and compatibility are bi-
 900 nary yes/no questions, grounding requires predicting 2D
 901 bounding boxes, and vacant-space localization involves pre-
 902 dicting a free-space location. We vary the mixture of these
 903 four types and evaluate performance on RoboSpatial-Home,
 904 with results summarized in Table 6. Notably, for configura-
 905 tion and compatibility, we ensure a balanced distribution of
 906 yes/no answers, as discussed in Appendix D. Interestingly,
 907 although grounding data are not present in RoboSpatial-
 908 Home, including grounding during training improves per-
 909 formance on the other tasks. In contrast, increasing the
 910 overall dataset size beyond a moderate scale yields limited
 911 gains, suggesting that data diversity and label balance con-
 912 tribute more to IRL effectiveness than raw quantity alone.

913 **E.3. Additional Demonstrations**

914 **Visualizations of success cases.** Apart from the cases
 915 where the grasp tool fails to find a collision-free pose or the
 916 pointing tool fails to localize points accurately for relative
 917 depth questions, we also present detailed examples of suc-
 918 cessful tool executions containing all special format-related
 919 tokens in Figure 6. In this example, the grasp tool success-
 920 fully finds an accurate pose in a cluttered scene. Figure 7
 921 provides examples for relative depth questions. One exam-
 922 ple shows the tool accurately predicts the required point lo-
 923 cations, enabling the VLM to produce the correct answer.
 924 The other example shows multiple chairs are present in the

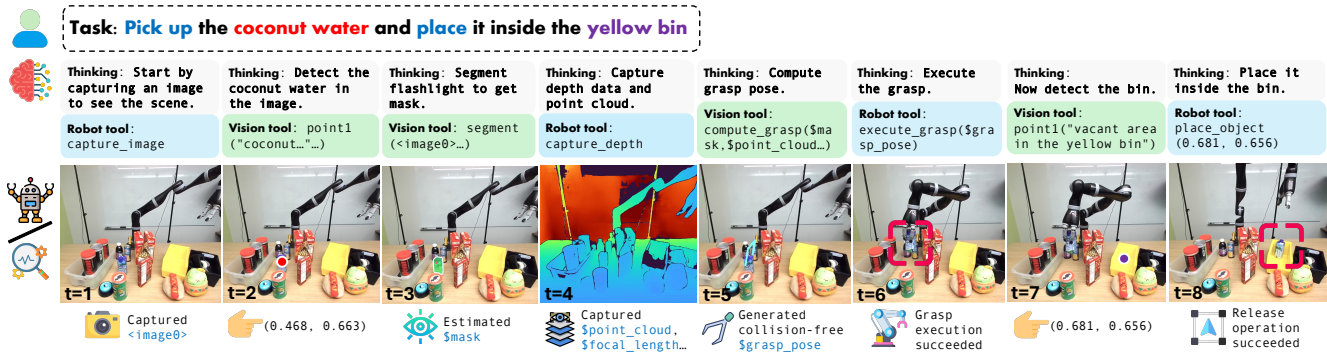


Figure 4. A hard real-world robot manipulation example with SpaceTools. The model successfully identifies the target object and completes the manipulation task in a cluttered and visually complex scene

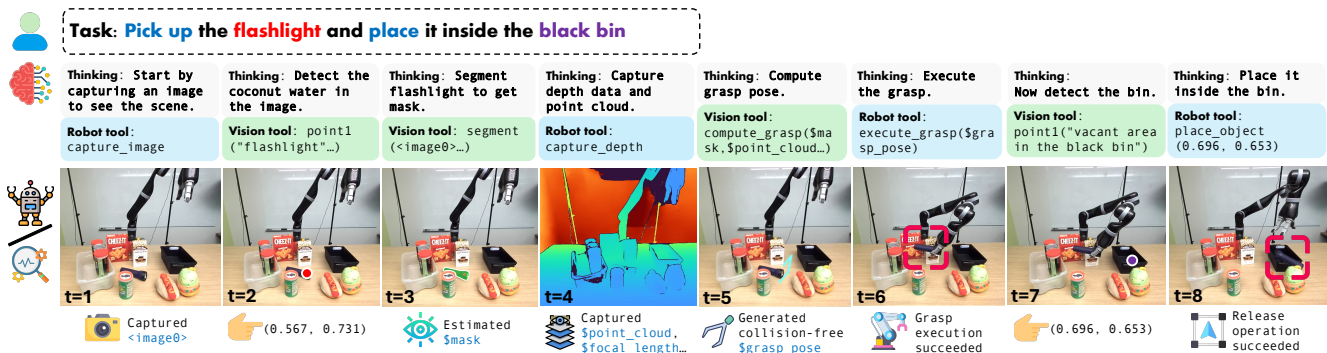


Figure 5. A failure case in real-world robot manipulation with SpaceTools. The model localizes a valid vacant area but selects a point too close to the boundary, resulting in a failed placement of the flashlight on the box boundary.

Table 6. Evaluation on RoboSpatial-Home using models trained with Tool IRL under different data compositions drawn from the four RoboSpatial data types. *Config.* refers to configuration data, *Compat.* to compatibility data, *Ground.* to grounding (2D bounding box) data, and *Vacant* to vacant-space localization data. Each entry in the middle columns indicates the number of samples included for that data category. *Overall Acc.* reports the final accuracy on RoboSpatial-Home.

| Variant | Config. | Compat. | Ground. | Vacant | Overall Acc. |
|-------------|---------|---------|---------|--------|--------------|
| All-v1 | 0.5k | 0.5k | 0.5k | 0.5k | 69.70 |
| All-v2 | 1.0k | 1.0k | 1.0k | 1.0k | 69.70 |
| All-v3 | 2.0k | 2.0k | 1.0k | 1.0k | 69.10 |
| w/o Ground. | 2.0k | 2.0k | 0.0k | 2.0k | 56.90 |

925 image, and the VLM must identify the specific one high-
 926 lighted by the provided bounding box. Moreover, as illus-
 927 trated in Figure 4, SpaceTools can reliably identify target
 928 objects and execute the required manipulation steps even in
 929 cluttered, visually complex real-world environments.

930 **Visualizations of failure cases.** Although the model per-
 931 forms well across many scenarios, it is not universally reli-

able. We therefore examine representative failure cases to 932
 highlight remaining challenges and motivate future explora- 933
 tion. Grasp estimation currently yields the lowest accu- 934
 racy, largely due to difficulties in detecting the target object 935
 in cluttered scenes and in predicting accurate yet collision- 936
 free grasp poses, as illustrated in Figure 8. Object detec- 937
 tion in complex, visually crowded environments remains a 938
 major bottleneck for both grasp and pose estimation, leav- 939
 ing substantial room for improvement. These issues point 940
 not only to opportunities for improving individual tools but 941
 also to the potential of modular enhancements to spatial rea- 942
 soning by strengthening specific tool components instead of 943
 retraining or finetuning the VLM. 944

Another failure example of real-world robot manipula- 945
 tion is shown in Figure 5. This failure case highlights a 946
 subtle but important challenge in real-world manipulation. 947
 Although the model correctly identifies a vacant region, it 948
 selects a point near the boundary, and thus, the robot fi- 949
 nally performs the placement on the boundary. Such cases 950
 underscore the need for highly precise geometric reason- 951
 ing and tighter coupling between 2D point predictions and 952
 physical feasibility on the robot. They also reveal a cur- 953
 rent limitation of the model’s tool coordination and point- 954

Grasp

What are the five normalized (x, y) points between 0 and 1 in the image that outline the grasp plane for the box of oatmeal-raisin cookies? Your answer should be as Grasp center: [], Left finger base: [], Right finger base: [], Left finger tip: [], Right finger tip: [].



Thinking: <think> I need to find the grasp plane points for the box of oatmeal-raisin cookies. To do this, I should:

1. First detect the oatmeal-raisin cookies box in the image
2. Segment it to get a mask
3. Estimate depth to get a point cloud
4. Compute a grasp pose
5. Extract the five key points that define the grasp plane

Let me start by detecting the oatmeal-raisin cookies box. </think>

Tool calls:

```
<tool_call>{"name": "roborefer.detect_one", "arguments": {"obj_name": "oatmeal-raisin cookies", "image_index": 0}}</tool_call>
```

<tool_response>Detected 1 instance(s) of 'oatmeal-raisin cookies': [(0.479, 0.417)]</tool_response>

Thinking: <think> Good, I found the oatmeal-raisin cookies at coordinates (0.479, 0.417). Now I need to segment it to get a mask of the object. </think>

```
<tool_call>{"name": "sam2.segment_from_point", "arguments": {"x": 0.479, "y": 0.417, "image_index": 0}}</tool_call>
```

<tool_response>Segmented object at normalized coords (0.479, 0.417) with IoU score: 0.940. Use \$segmentation_mask (boolean array, 289x511) to reference it.</tool_response>

Thinking: <think> Great! I have the segmentation mask. Now I need to estimate depth and generate a point cloud so I can compute the grasp pose. </think>

```
<tool_call>{"name": "depth_estimator.estimate_depth_with_pointcloud", "arguments": {"image_index": 0}}</tool_call>
```

<tool_response>Generated 511x289 depth map and 147,679 3D points. Depth range in point cloud: 1.01m to 1.44m (mean: 1.18m). Use \$depth_map (numpy array, 289x511), \$point_cloud (numpy array, 147679x3) and \$focal_length_px (float) to reference the results.</tool_response>

Thinking: <think> Perfect! Now I have:- \$segmentation_mask: the mask of the oatmeal-raisin cookies box- \$point_cloud: the 3D point cloud- \$focal_length_px: the focal length. Now I can compute the grasp pose for the object. </think>

```
<tool_call> {"name": "grasp_generator.compute_grasp", "arguments": {"point_cloud": "$point_cloud", "mask": "$segmentation_mask", "focal_length_px": "$focal_length_px", "image_index": 0}} </tool_call>
```

<tool_response>Generated collision-free grasp with confidence 0.978 from 1980/147679 masked points. Filtered 100 → 12 collision-free grasps. Projected 2D gripper points: [(0.501, 0.213), (0.460, 0.168), (0.546, 0.261), (0.461, 0.262), (0.543, 0.359)].</tool_response>

Thinking: <think> Excellent! The grasp generator has computed a grasp pose and provided the five projected 2D gripper points in normalized coordinates. These points represent: 1. Grasp center: (0.501, 0.213) 2. Left finger base: (0.460, 0.168) 3. Right finger base: (0.546, 0.261) 4. Left finger tip: (0.461, 0.262) 5. Right finger tip: (0.543, 0.359) These five points outline the grasp plane for the oatmeal-raisin cookies box. </think>

Answer: <answer>

```
Grasp center: [0.501, 0.213],
Left finger base: [0.460, 0.168],
Right finger base: [0.546,
0.261], Left finger tip: [0.461,
0.262], Right finger tip: [0.543,
0.359]</answer>
```




Figure 6. A detailed example of tool-augmented reasoning of a grasp estimation question.

955 selection strategy, suggesting promising directions for fu-
956 ture improvements, including integrating real or simulated

robot feedback into the training process.

957

Relative depth



Which point is closer to the camera?

Thinking: <think> Looking at the image, I can see a person wearing a headscarf with two red dots labeled A and B. Point A appears to be on the left side of the headscarf, while point B is on the right side. Both points seem to be on the same person, so I need to use depth estimation to determine which is closer to the camera. Let me analyze this systematically: </think>

Tool calls: <tool_call>{"name": "depth_estimator.estimate_depth", "arguments": {"image_index": 0}}</tool_call><tool_call>{"name": "roborefer.detect_one", "arguments": {"obj_name": "red circle under label A", "image_index": 0}}</tool_call><tool_call>{"name": "roborefer.detect_one", "arguments": {"obj_name": "red circle under label B", "image_index": 0}}</tool_call>

<tool_response>Estimated depth map for 504x476 image. Focal length: 1198.4px. Depth range: 0.52m to 1.34m (mean: 0.74m). Use \$depth_map (numpy array, 476x504) to reference the depth data and \$focal_length_px (float) to reference the focal length.</tool_response><tool_response>Detected 1 instance(s) of 'red circle under label A': [(0.327, 0.439)]</tool_response><tool_response>Detected 1 instance(s) of 'red circle under label B': [(0.672, 0.437)]</tool_response>

Thinking: <think> Now let me get the depth values at these two points: </think>


Tool calls: <tool_call>{"name": "vision_ops.index_at", "arguments": {"data": "\$depth_map", "u": 0.327, "v": 0.439}}</tool_call><tool_call>{"name": "vision_ops.index_at", "arguments": {"data": "\$depth_map", "u": 0.672, "v": 0.437}}</tool_call>

<tool_response>Pixel value at (0.327, 0.439) is 0.6470662355422974</tool_response><tool_response>Pixel value at (0.672, 0.437) is 0.5613858699798584</tool_response>

Thinking: <think> Based on the depth analysis:
- Point A is at depth: 0.647m
- Point B is at depth: 0.561m
The depth difference is approximately 0.086m, which is relatively small. However, point B has a smaller depth value, meaning it is closer to the camera. Looking at the image again, this makes sense as point B appears to be on the right side of the headscarf which might be slightly forward, or the depth estimation indicates it's closer. </think>

Answer: <answer>B</answer> ✓

Relative depth



Which object is closer to the camera taking this photo, the books (highlighted by a red box) or the chair (highlighted by a blue box)?

Thinking: <think> I need to determine which object is closer to the camera by analyzing the depth information. Let me follow the systematic process: First, I'll estimate the depth map of the image, then locate both objects and compare their depth values. </think>

Tool calls: <tool_call>{"name": "depth_estimator.estimate_depth", "arguments": {"image_index": 0}}</tool_call><tool_call>{"name": "roborefer.detect_one", "arguments": {"obj_name": "books (highlighted by a red box)", "image_index": 0}}</tool_call><tool_call>{"name": "roborefer.detect_one", "arguments": {"obj_name": "chair (highlighted by a blue box)", "image_index": 0}}</tool_call>

<tool_response>Estimated depth map for 512x373 image. Focal length: 473.6px. Depth range: 1.84m to 8.38m (mean: 4.02m). Use \$depth_map (numpy array, 373x512) to reference the depth data and \$focal_length_px (float) to reference the focal length.</tool_response><tool_response>Detected 1 instance(s) of 'books (highlighted by a red box)': [(0.861, 0.672)]</tool_response><tool_response>Detected 1 instance(s) of 'chair (highlighted by a blue box)': [(0.443, 0.733)]</tool_response>

Thinking: <think> Now I'll get the depth values at both detected locations: </think>

Tool calls: <tool_call>{"name": "vision_ops.index_at", "arguments": {"data": "\$depth_map", "u": 0.861, "v": 0.672}}</tool_call><tool_call>{"name": "vision_ops.index_at", "arguments": {"data": "\$depth_map", "u": 0.443, "v": 0.733}}</tool_call>

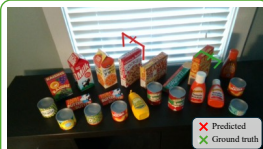
<tool_response>Pixel value at (0.861, 0.672) is 2.70392107963562</tool_response><tool_response>Pixel value at (0.443, 0.733) is 5.636236667633057</tool_response>

Thinking: <think> Based on the depth analysis:
- Books (highlighted by red box): depth = 2.70 meters
- Chair (highlighted by blue box): depth = 5.64 meters
The books have a significantly smaller depth value (2.70m) compared to the chair (5.64m), meaning the books are closer to the camera </think>

Answer: <answer>A</answer> ✓


Figure 7. Detailed examples of tool-augmented reasoning of relative depth questions.

What are the five normalized (x, y) points between 0 and 1 in the image that outline the grasp plane for the box of macaroni and cheese labeled 'SO CHEESY'? Your answer should be as Grasp center: [], Left finger base: [], Right finger base: [], Left finger tip: [], Right finger tip: [].



(a) Wrong object localization

What are the five normalized (x, y) points between 0 and 1 in the image that outline the grasp plane for the can of tuna fish? Your answer should be as Grasp center: [], Left finger base: [], Right finger base: [], Left finger tip: [], Right finger tip: [].



(b) Inaccurate pose estimation

Figure 8. Failure cases for the grasp estimation task. Intermediate tool-augmented reasoning steps are omitted for clarity.

958

References

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

981

982

983

984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

1001

1002

1003

1004

1005

1006

1007

1008

1009

1010

1011

1012

1013

- [1] Aleksei Bochkovskii, Amaël Delaunoy, Hugo Germain, Marcel Santos, Yichao Zhou, Stephan R. Richter, and Vladlen Koltun. Depth pro: Sharp monocular metric depth in less than a second. In *International Conference on Learning Representations*, 2025. 2
- [2] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. 11
- [3] Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, Jiasen Lu, Taira Anderson, Erin Bransom, Kiana Ehsani, Huong Ngo, Yen-Sung Chen, Ajay Patel, Mark Yatskar, Chris Callison-Burch, Andrew Head, Rose Hendrix, Favyen Bastani, Eli Vanderbilt, Nathan Lambert, Yvonne Chou, Arnavi Chheda, Jenna Sparks, Sam Skjonsberg, Michael Schmitz, Aaron Sarnat, Byron Bischoff, Pete Walsh, Chris Newell, Piper Wolters, Tanmay Gupta, Kuo-Hao Zeng, Jon Borchardt, Dirk Groeneveld, Crystal Nam, Sophie Lebrecht, Caitlin Wittlif, Carissa Schoenick, Oscar Michel, Ranjay Krishna, Luca Weihs, Noah A. Smith, Hannaneh Hajishirzi, Ross Girshick, Ali Farhadi, and Aniruddha Kembhavi. Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025. 2
- [4] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*, 2024. 2
- [5] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. 8
- [6] Balakumar Sundaralingam, Siva Kumar Sastry Hari, Adam Fishman, Caelan Garrett, Karl Van Wyk, Valts Blukis, Alexander Millane, Helen Oleynikova, Ankur Handa, Fabio Ramos, Nathan Ratliff, and Dieter Fox. curobo: Parallelized collision-free minimum-jerk robot motion generation, 2023. 9
- [7] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2022. Curran Associates Inc. 11
- [8] Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz, Bryan Catanzaro, Andrew Tao, Qingyun Wu, Zhiding Yu, and Guilin Liu. Nemotron-research-tool-n1: Exploring tool-using language models with reinforced reasoning, 2025. 10
- [9] Enshen Zhou, Jingkun An, Cheng Chi, Yi Han, Shanyu Rong, Chi Zhang, Pengwei Wang, Zhongyuan Wang, Tiejun Huang, Lu Sheng, et al. Roborefer: Towards spatial referring with reasoning in vision-language models for robotics. *arXiv preprint arXiv:2506.04308*, 2025. 2