KEY-GRAM: EXTENSIBLE WORLD KNOWLEDGE FOR EMBODIED MANIPULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Language-conditioned manipulation requires both reusable task knowledge and state-dependent visual reasoning. Most vision-language-action policies represent both within a shared backbone or conditioning path, making targeted knowledge extension difficult. We introduce Key-Gram, a conditional-memory framework that extracts short, task-specific key-grams, maps them to learnable memory entries by deterministic multi-head hashing, and injects the retrieved entries into selected Transformer layers through token-wise gating and lightweight convolutional fusion. The memory is organized as independently addressable subtables, allowing capacity to grow without changing the backbone architecture and supporting constant-time lookup with respect to total table size. Across RoboTwin2.0, LIBERO/LIBERO-Plus, and real-world dual-arm manipulation, Key-Gram improves both π_0 and $\pi_{0.5}$. Relative gains are $29.5\%/9.9\%$ on the hard RoboTwin2.0 split, $35.8\%/4.5\%$ on LIBERO-Plus transfer without target-domain fine-tuning, and $15.4\%/8.1\%$ on real-world long-horizon tasks. In four-stage continual compositional assembly, Stage 4 terminal-node success rises from $20\%/52\%$ to $58\%/74\%$. These results support external linguistic memory as a useful architectural bias for compositional grounding and transfer under task expansion.

1 INTRODUCTION

Embodied manipulation requires reusable knowledge of objects, relations, constraints, and task structure, together with reasoning about scene dynamics. Vision-language-action (VLA) models incorporate language priors into control (Zitkovich et al., 2023; Kim et al., 2025b; Black et al., 2024; 2025; Kim et al., 2025a; Liu et al., 2025; NVIDIA et al., 2025; Cheang et al., 2025; Zheng et al., 2026); World Action Models (WAMs) add future-state prediction (Cheang et al., 2024; Huang et al., 2025; Liao et al., 2025; Lu et al., 2025; Li et al., 2026; Ye et al., 2026). Both typically represent instruction knowledge and physical reasoning within the same dense network.

Dense VLA fusion embeds language and vision in a shared token space (Zitkovich et al., 2023; Kim et al., 2025b; Black et al., 2024); diffusion policies and WAMs also use separate language conditioning (Chi et al., 2023; Ye et al., 2026). Both routes encode instruction knowledge in dense parameters, so adding concepts modifies weights supporting existing behaviors and risks interference during sequential adaptation (McCloskey & Cohen, 1989; French, 1999). FiLM, cross-attention, and AdaLN improve conditioning (Perez et al., 2018; Dumoulin et al., 2018; Peebles & Xie, 2023; Dasari et al., 2024), but provide no independently addressable instruction-level store for targeted knowledge expansion.

Motivated by conditional memory (Cheng et al., 2026), Key-Gram uses task-specific phrases to index a separate parameterized store and injects retrieved entries into selected visual-backbone layers. Its slot-head tables can grow independently of the backbone architecture. New entries can be trained with existing tables and backbone weights optionally frozen, while their grounding in action still requires learning.

We evaluate Key-Gram with π_0 and $\pi_{0.5}$ on RoboTwin2.0 (Chen et al., 2025), LIBERO (Liu et al., 2023), LIBERO-Plus (Fei et al., 2025), and real-world dual-arm tasks. These experiments assess hard-task performance, distribution transfer, unseen object–target compositions, and continual task

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

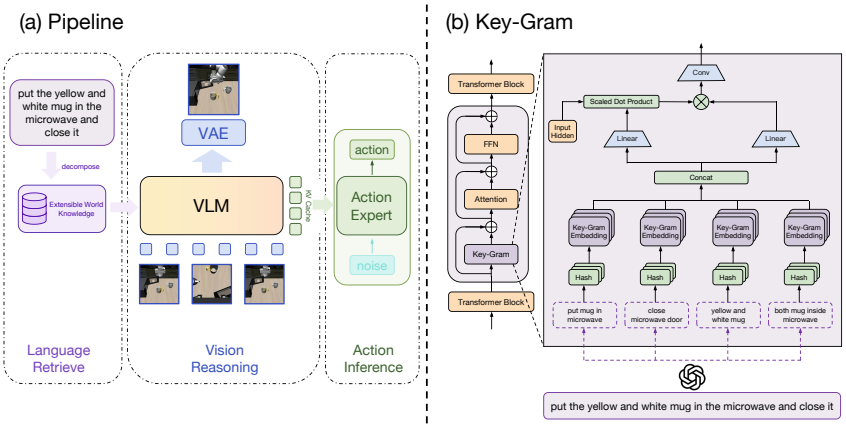


Figure 1: Overview of Key-Gram. (a) The framework separates language-derived knowledge retrieval, visual reasoning, and action inference. (b) The Key-Gram module retrieves external linguistic priors from decomposed key-grams via multi-head hashing, and injects them into selected Transformer layers through context-aware gated fusion.

expansion. Component and placement ablations favor semantic key-grams and early-layer memory access.

Our contributions are summarized as follows:

- **A functionally decoupled architecture.** Key-Gram adds memory without changing the backbone or action expert.
- **An extensible external memory for world knowledge.** Key-Gram uses deterministic hashed lookup over slot-head sub-tables whose capacity can grow without changing the backbone architecture.
- **Consistent gains in instruction-sensitive manipulation.** Experiments across simulated and real-world benchmarks show improvements on hard tasks, distribution transfer, unseen object-target compositions, and continual task expansion.

2 Related Work

2.1 Vision-Language-Action Models

VLA models transfer vision-language priors into robot control, improving semantic grounding, instruction following, cross-embodiment transfer, and long-horizon manipulation (Zitkovich et al., 2023; Kim et al., 2025b; Black et al., 2024; 2025; Zheng et al., 2026). Most use dense multimodal fusion, processing language and visual tokens through a shared backbone (Zitkovich et al., 2023; Kim et al., 2025b; Black et al., 2024). Short instructions consequently compete with dense visual tokens, potentially diluting instruction-specific information in cluttered or compositional scenes. Key-Gram separates instruction-side knowledge retrieval from scene-side visual reasoning to bypass this input-level competition.

2.2 World Action Models and Predictive Manipulation

Diffusion policies and WAMs predict future observations, latent states, or action trajectories to support manipulation (Chi et al., 2023; Lu et al., 2025; Li et al., 2026; Ye et al., 2026). DreamZero, for example, adapts a pretrained video diffusion model into a real-time closed-loop controller for long-horizon manipulation (Ye et al., 2026).

WAM-style methods strengthen dynamic scene prediction, while instruction knowledge generally remains represented in shared model parameters. Key-Gram complements predictive control by providing an independently addressable memory path for instruction-derived entries.

2.3 World Memory, Layer-wise Memory Scaling, and Lifelong Expansion

Product-key memories and retrieval-augmented language models separate storage from computation to improve capacity and efficiency (Lample et al., 2019; Guu et al., 2020; Lewis et al., 2020; Borgeaud et al., 2022). Later approaches use hashed conditional memory, static embedding tables, and layer-wise memory or embedding modules (Wu et al., 2022; Wang et al., 2023; Cheng et al., 2026; Liu et al., 2026; Ding et al., 2026; Google, 2026). These methods mainly target language modeling; embodied adaptation must also address interference with previously learned objects, skills, and relations (McCloskey & Cohen, 1989; French, 1999). Key-Gram applies conditional memory to robot control, testing whether instruction-indexed entries improve transfer and composition.

3 Method

3.1 Overview

We introduce Key-Gram, an external conditional-memory module that augments visual reasoning models with structured linguistic entries. Given an instruction $\mathcal{I}$ and initial visual observations $\mathcal{V}_0$, the model predicts an action trajectory and a compact future visual state. As shown in Fig. 1, Key-Gram decomposes $\mathcal{I}$ into task-specific key-grams, maps them to learnable memory embeddings by deterministic lookup, and injects the retrieved embeddings into selected backbone layers through context-adaptive modulation and lightweight convolutional refinement.

The memory-guided backbone supports two downstream prediction pathways: final-layer hidden states are adapted into VAE-encodable future latents, while the backbone KV cache conditions an action expert for trajectory decoding. This design routes instruction-derived priors through an external memory path while retaining the backbone's visual and action computation. The following sections describe retrieval, fusion, prediction, and memory allocation.

3.2 Sparse Key-Gram Retrieval

The first phase maps instruction-level semantic units to memory entries through key-gram extraction and deterministic hashed retrieval.

Key-gram extraction. Rather than relying on tokenization or contiguous phrase segmentation, we use a lightweight language-model parser to extract a fixed number of short, knowledge-bearing key-grams from each instruction. The parser is prompted to compose reusable task-level units under a maximum word length. For example, "put the yellow and white mug in the microwave and close it" can be decomposed into {*put mug in microwave*, *close microwave door*, *yellow and white mug*, *mug inside microwave*}. The key-gram budget is set by environment-level instruction complexity. Appendix B gives the extraction prompt used in our experiments.

Hashed memory mapping. Let $\mathcal{G} = \{g_i\}_{i=1}^{K}$ be the extracted key-grams, where each g_i contains at most M words. We convert each word into an integer identifier and right-pad shorter key-grams with zeros to obtain a fixed-length key $\bar{g}_i = (a_{i,1}, \ldots, a_{i,M})$. Following multiplicative-XOR hashing, each key is mapped to H hash heads. For the h-th head at layer ℓ, the memory index is computed as

$$z_i^{(\ell,h)} = \left(\bigoplus_{j=1}^{M} a_{i,j} r_j^{(\ell,h)} \right) \bmod P^{(\ell,h)}, \tag{1}$$

where $\oplus$ denotes bitwise XOR, $r_j^{(\ell,h)}$ are deterministic odd multipliers, and $P^{(\ell,h)}$ is the sub-table's prime row count. Across all sub-tables, we choose mutually distinct primes as close as possible to the nominal capacity $V = 8192$. The corresponding embedding is retrieved as $e_i^{(\ell,h)} = E^{(\ell,h)}[z_i^{(\ell,h)}]$, and all head outputs are concatenated to form

$$e_i^{(\ell)} = \Big\Vert_{h=1}^{H} e_i^{(\ell,h)}. \tag{2}$$

The retrieved embeddings $\{e_i^{(\ell)}\}_{i=1}^{K}$ form the linguistic memory input to the subsequent fusion stage.

Address-localized writes. Independent slot–head hashes route each key-gram to a deterministic tuple of rows, and its memory gradient updates only those rows. Thus, absent a complete multi-head collision, a new key-gram cannot directly overwrite an existing entry.

3.3 Context-Adaptive Key-Gram Fusion

As illustrated in Fig. 1 (b), the fusion phase injects retrieved key-gram memories into the visual backbone through a residual module inserted before the attention operation of selected Transformer blocks.

Let $H^{(\ell)} \in \mathbb{R}^{B\times L\times d}$ denote the hidden states before attention at layer ℓ, where L is the number of visual tokens. The embeddings obtained by the hashed retrieval above are first concatenated into a memory vector $M^{(\ell)} \in \mathbb{R}^{B\times 1\times d_m}$, where d_m aggregates both the K extracted key-grams and their H hash heads. Specifically, if each hash head returns a d_h-dimensional vector, then $d_m = KHd_h$. We therefore project the memory into the backbone space through two learnable matrices, yielding $K_m^{(\ell)} = M^{(\ell)}W_K^{(\ell)}$ and $V_m^{(\ell)} = M^{(\ell)}W_V^{(\ell)}$, where $K_m^{(\ell)}, V_m^{(\ell)} \in \mathbb{R}^{B\times 1\times d}$. The incoming hidden states are directly used as visual queries. A token-wise gate is computed as

$$A^{(\ell)} = \sigma\Big(H^{(\ell)}{K_m^{(\ell)}}^{\top}/\sqrt{d}\Big), \qquad A^{(\ell)} \in \mathbb{R}^{B\times L\times 1}. \tag{3}$$

This gate measures the relevance of the retrieved linguistic memory to each visual token, enabling the model to inject knowledge priors only where they are useful.

The gated memory feature is then refined by a lightweight long-span convolution:

$$\Delta H^{(\ell)} = \mathrm{Conv}_{\mathrm{span}}\Big(A^{(\ell)} \odot V_m^{(\ell)}\Big). \tag{4}$$

Since $M^{(\ell)}$ is formed by concatenating multiple key-gram memories, the projected value $V_m^{(\ell)}$ contains heterogeneous pieces of instruction-level knowledge. The long-span convolution is used to mix these memory components over a wider receptive field, allowing relations among different key-grams to be jointly modeled before being injected into each visual token. This operation provides an explicit interaction mechanism among retrieved knowledge entries beyond independent gating.

The Key-Gram output is finally added to the original hidden states in residual form, $\widetilde{H}^{(\ell)} = H^{(\ell)} + \Delta H^{(\ell)}$, and $\widetilde{H}^{(\ell)}$ is passed to the standard attention computation. We instantiate this module only at selected backbone layers, following the placement strategy validated by the ablation study below.

3.4 Coupled Visual and Action Prediction

As illustrated in Fig. 1 (a), the memory-guided backbone is followed by two prediction pathways for visual foresight and action decoding.

Vision reasoning. Let $H^{(L)}$ denote the final-layer hidden states of the backbone. We introduce a short sequence of learnable queries Q_v and use a cross-attention block to extract compact future-vision representations from $H^{(L)}$. The resulting query features are projected into the visual latent space and passed through two lightweight upsampling stages to match the spatial resolution of the VAE latent. The visual pathway therefore predicts visual foresight in a compact latent space.

Action inference. For action prediction, we reuse the KV cache accumulated by the preceding backbone as conditional context for a lightweight action expert. The action expert is architecturally isomorphic to the backbone but smaller in scale, and performs a flow-matching denoising process over action trajectories. This design allows action decoding to remain tightly coupled with the same memory-guided visual reasoning process that produces future visual latents.

3.5 Extensible Memory Allocation

As illustrated in Fig. 2, Key-Gram organizes memory as a logical table composed of multiple sub-tables rather than a single monolithic table. Each key-gram slot and hash head owns an independent sub-table $E_{s,h} \in \mathbb{R}^{P_{s,h}\times d_e}$, where $P_{s,h}$ is its actual prime row count and d_e is the embedding width.

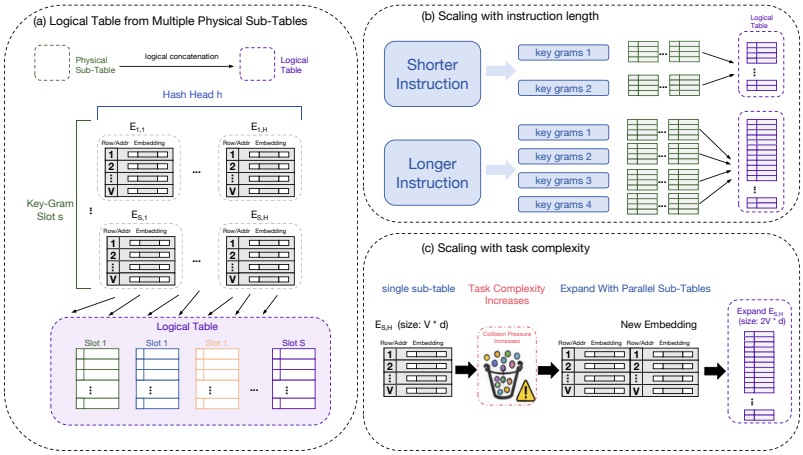


Figure 2: Extensible memory allocation of Key-Gram. The memory is a logical table composed of slot-head physical sub-tables rather than a single monolithic table. Here, V denotes nominal row capacity; actual sub-table sizes are distinct nearby primes. Longer instructions are handled by adding key-gram slots, while more complex task distributions are supported by appending new sub-tables to enlarge the effective capacity and reduce hash collisions.

Retrieved entries from all slot-head sub-tables are concatenated before fusion, and sub-tables can be partitioned across devices during training. At inference, the instruction and fixed hash functions directly determine the indices, giving $O(1)$ lookup with respect to total table size for a fixed number of active slots and heads.

Scaling with instruction length. Longer instructions typically require more key-grams to represent their world knowledge. Key-Gram handles this by increasing the number of key-gram slots, where each added slot introduces a group of H hash-head sub-tables. For example, a short instruction may activate key-grams such as $\{$*put mug in microwave*, *close microwave door*, *yellow and white mug*$\}$, whereas a longer instruction may require additional entries such as $\{$*pick hamburger and fries*, *French fries in red packaging*, *smooth glossy orange tray*$\}$. Thus, longer tasks can retrieve richer linguistic priors while keeping the visual backbone unchanged.

Scaling with task complexity. More complex environments introduce additional entities, relations, and affordances, increasing hash-collision pressure under a fixed capacity. Key-Gram can enlarge V or append parallel sub-tables while retaining all previously allocated rows and hash functions. New key-grams are consequently assigned localized parameter support without reparameterizing existing addresses, and the visual-backbone architecture remains unchanged. The continual compositional-assembly protocol evaluates this extensible memory path under the same full-parameter update regime used by the conventional backbone baseline.

4 Experiment

4.1 Experiments Setup

Benchmarks. We evaluate both Key-Gram backbones on RoboTwin2.0, LIBERO, and LIBERO-Plus (Chen et al., 2025; Liu et al., 2023; Fei et al., 2025). RoboTwin2.0 covers 50 tasks under clean and randomized settings, with 100 trials per configuration in unseen environments. We use LIBERO's four standard suites and evaluate LIBERO-Plus distribution expansion through zero-shot transfer from LIBERO-trained models and target-domain fine-tuning.

Model Configuration. We instantiate Key-Gram on two open-source VLA backbones, π_0 and $\pi_{0.5}$ (Black et al., 2024; 2025). Both baselines and KG variants use matched full-parameter training, isolating the added memory path rather than a frozen-backbone advantage. Layers $(1, 8, 13)$ are

selected with independently trained checkpoints on a separate four-task dataset and then fixed for all main experiments. Simulation uses eight slots, four heads, 32-dimensional entries, and nominal subtable capacity $V = 8192$; real-world experiments use 16 slots and two heads at the same retrieved width (1024).

Real-world robot. We use a Piper dual-arm platform for three long-horizon tasks, five object–target recombination settings, and four-stage continual assembly (Appendix D). Long-horizon training uses 300 trajectories per task. Recombination uses two 50-trajectory training sets and evaluates seen execution, two unseen pairings, and sequential re-evaluation after the second update.

Continual assembly uses the same jointly trained task as Table 3, but decomposes it into four stages. Each stage continues from the previous checkpoint and updates all parameters. The task grows from one to seven tableware, food, and cookware objects; actions grow from picking/placing to include pulling/pouring. Each stage has 50 evaluation trials, and Node j is the unconditional fraction reaching that milestone, so later nodes certify earlier ones. An external LLM first diversifies the instructions shared by all variants; each KG variant then associates every diversified instruction with a task-specific Key-Gram, controlling language augmentation across models.

Table 1: RoboTwin2.0 results (%). Gains in parentheses for Key-Gram variants are relative improvements over their corresponding base backbones. $\mathrm{H} = 1, 2, 3$ denote the benchmark's conventional short-, medium-, and long-horizon task-length groups.

	X-VLA		π_0		$\pi_{0.5}$		π_0-KG (ours)		$\pi_{0.5}$-KG (ours)	
Metric	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard
Avg. $_{\mathrm{H=1}}$	81.6	82.5	66.5	61.6	85.1	80.2	80.4 (+20.9%)	77.9 (+26.4%)	90.8 (+6.6%)	87.0 (+8.5%)
Avg. $_{\mathrm{H=2}}$	59.3	55.9	66.1	54.7	79.3	73.0	80.5 (+21.7%)	72.7 (+33.0%)	86.3 (+8.7%)	80.3 (+10.0%)
Avg. $_{\mathrm{H=3}}$	61.2	66.0	61.6	50.2	78.6	67.4	79.4 (+28.9%)	70.2 (+39.8%)	86.8 (+10.4%)	80.8 (+19.9%)
Avg. $_{\mathrm{50\ Tasks}}$	72.9	72.8	65.9	58.4	82.7	76.8	80.3 (+21.9%)	75.6 (+29.5%)	89.0 (+7.6%)	84.4 (+9.9%)

Table 2: LIBERO and LIBERO-Plus results (%). Gains in parentheses for Key-Gram variants are relative improvements over their corresponding base backbones.

Model	LIBERO fine-tuned	LIBERO-Plus from LIBERO	LIBERO-Plus fine-tuned
OpenVLA-OFT	95.3	69.6	79.6
π_0	94.2	53.6	84.0
$\pi_{0.5}$	96.9	83.9	90.4
π_0-KG (ours)	94.6 (+0.4%)	72.8 (+35.8%)	88.5 (+5.4%)
$\pi_{0.5}$-KG (ours)	96.7 (-0.2%)	87.7 (+4.5%)	92.6 (+2.4%)

4.2 Results on Simulation

Table 1 shows consistent RoboTwin2.0 gains. Easy/Hard averages rise from $65.9/58.4$ to $80.3/75.6$ for π_0 and from $82.7/76.8$ to $89.0/84.4$ for $\pi_{0.5}$. Gains generally increase with horizon, consistent with retrieved priors helping maintain object relations across multi-step manipulation. Appendix A reports per-task results for all 50 RoboTwin2.0 tasks.

Table 2 shows near-saturated LIBERO performance and stronger LIBERO-Plus generalization. Direct transfer improves from 53.6 to 72.8 ($+35.8\%$) for π_0 and from 83.9 to 87.7 ($+4.5\%$) for $\pi_{0.5}$; after target fine-tuning, KG reaches $88.5/92.6$. This supports the memory path under distribution expansion.

4.3 Results on Real-world Robot

Long-horizon Tasks. Table 3 shows gains on every long-horizon task, led by Assembly at $+26.9\%/+17.1\%$ for π_0/$\pi_{0.5}$. It trains the complete Assembly task jointly, whereas Table 5 incrementally trains the same task and evaluates prefix retention. Figure 3 shows representative rollouts.

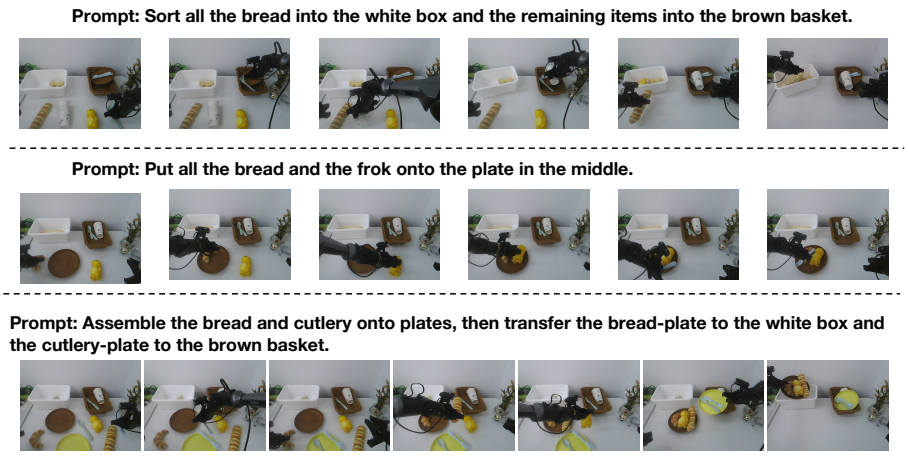


Figure 3: Representative $\pi_{0.5}$-KG rollouts for the three real-world long-horizon tasks: sorting, picking, and compositional assembly. See Table 3 for aggregate success rates.

Table 3: Real-world long-horizon task results (%). Gains in parentheses for Key-Gram variants are relative improvements over their corresponding base backbones. The complete Assembly task is trained jointly here; Table 5 evaluates incremental training on the same underlying task.

Model	Picking	Sorting	Assembly	Avg.
π_0	82.0	74.0	52.0	69.3
$\pi_{0.5}$	90.0	86.0	70.0	82.0
π_0-KG (ours)	88.0 (+7.3%)	86.0 (+16.2%)	66.0 (+26.9%)	80.0 (+15.4%)
$\pi_{0.5}$-KG (ours)	94.0 (+4.4%)	90.0 (+4.7%)	82.0 (+17.1%)	88.7 (+8.1%)

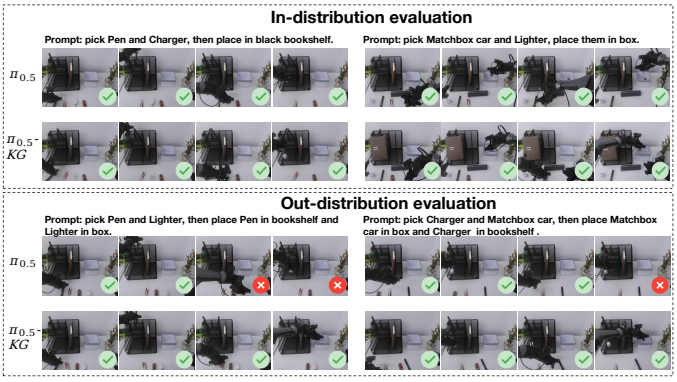


Figure 4: Qualitative real-world expansion tasks. Both models handle the seen pairings (top); on unseen object–target recombinations (bottom), the vanilla $\pi_{0.5}$ policy fails in the illustrated trials while $\pi_{0.5}$-KG succeeds. See Table 4 for aggregate success rates.

Expansion via compositional recombination. Table 4 shows small changes on strong in-distribution tasks but improved unseen pairings: averages rise from 72.4 to 81.6 for π_0 and from 80.0 to 86.8 for $\pi_{0.5}$, with the largest gains on recombined Tasks 3–4. Figure 4 shows both settings.

Continual compositional expansion. Table 5 reports cumulative node-reaching success. Across Stages 1–4, terminal success changes from $82/68/42/20$ to $88/88/74/58$ for π_0 (gains of $6/20/32/38$ points) and from $90/82/72/52$ to $94/90/84/74$ for $\pi_{0.5}$ (gains of $4/8/12/22$ points). The advantage grows as objects expand from tableware to food and cookware and actions from picking/placing to pulling/pouring. At Stage 4, KG improves every prefix from $38/26/22/20$ to

Table 4: Real-world expansion-task results (%). Gains in parentheses for KG variants are relative improvements over their corresponding base backbones. Task 1–5 correspond to two in-distribution settings, two unseen compositional pairings, and one sequential-adaptation setting.

Model	Task 1	Task 2	Task 3	Task 4	Task 5	Avg.
π_0	92.0	90.0	52.0	48.0	80.0	72.4
$\pi_{0.5}$	92.0	92.0	64.0	66.0	86.0	80.0
π_0-KG (ours)	90.0 (-2.2%)	92.0 (+2.2%)	70.0 (+34.6%)	68.0 (+41.7%)	88.0 (+10.0%)	81.6 (+12.7%)
$\pi_{0.5}$-KG (ours)	96.0 (+4.3%)	92.0 (+0.0%)	76.0 (+18.8%)	80.0 (+21.2%)	90.0 (+4.7%)	86.8 (+8.5%)

Table 5: Continual compositional-assembly success rates (%). Each stage continues from the preceding checkpoint with all parameters trainable and is evaluated over 50 trials. Node j is the unconditional fraction of trials reaching that ordered milestone; reaching it entails completing all preceding nodes. Across Stages 1–4, the task expands from one to seven objects and from two to four action modes. Bold values mark the KG result within each backbone pair.

	Stage 1	Stage 2		Stage 3			Stage 4			
Model	N1	N1	N2	N1	N2	N3	N1	N2	N3	N4
π_0	82	80	68	54	46	42	38	26	22	20
$\pi_{0.5}$	90	90	82	76	72	72	66	60	54	52
π_0-KG (ours)	**88**	**92**	**88**	**82**	**76**	**74**	**74**	**66**	**62**	**58**
$\pi_{0.5}$-KG (ours)	**94**	**94**	**90**	**88**	**84**	**84**	**80**	**80**	**76**	**74**

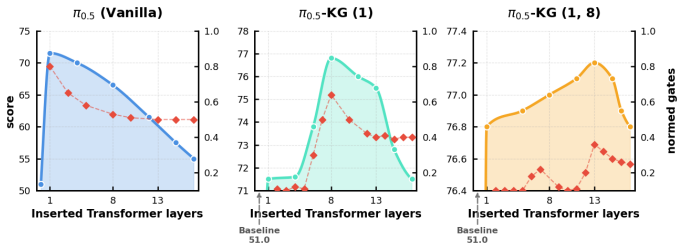


Figure 5: Layer-placement ablation on RoboTwin2.0. Shaded curves denote the weighted task score, while diamond markers denote the normalized gate activation of the probed Key-Gram module. The left and right axes correspond to score and normalized gates, respectively.

74/66/62/58 and from 66/60/54/52 to 80/80/76/74, respectively. All nodes use the same 50 complete rollouts, and later-node success entails earlier-node success; prefix rates therefore measure capability retention while the terminal rate measures acquisition under fully trainable incremental updates.

4.4 Ablation Study

We ablate Key-Gram on RoboTwin2.0 with the $\pi_{0.5}$ backbone using *hanging mug*, *move stapler pad*, *pick dual bottles*, and *stack blocks three*. Every configuration is trained independently on this isolated dataset with checkpoints disjoint from the main experiment; the score weights Easy and Hard by 1:9 (Appendix C).

Table 6 first isolates the components. The trained, parameter-matched +VAE adapter/MLP control raises the vanilla score from 51.000 to 61.950; it is not an extra-loss-only baseline. Untrained random memory falls to 30.225, while shuffling the trained instruction–Key-Gram associations reduces 77.175 to 58.825. CRFsuite phrase indices reach 68.075, 9.100 points below semantic key-grams. Fixing the gate to one gives 75.325, while removing future-latent prediction gives 68.925. Thus, performance depends on content-matched semantic retrieval as well as adaptive gating and visual foresight.

Table 6: Component and layer-placement ablations on RoboTwin2.0 (%) with the $\pi_{0.5}$ backbone. Parentheses give inserted Transformer layers. Score averages the four tasks after weighting Easy and Hard by 1:9. The +VAE control includes the parameter-matched VAE-bound residual adapter/MLP and future-latent loss. Random memory retrieves untrained table entries; shuffled KG breaks the instruction–Key-Gram correspondence while retaining trained entries; CRF phrases replace semantic extraction.

	Hanging Mug		Move Stapler Pad		Pick Dual Bottles		Stack Blocks Three		
Configuration	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard	Score
Vanilla	18	17	56	42	93	63	91	76	51.000
+ VAE	24	25	70	64	95	71	93	84	61.950
+ VAE + random memory	12	5	32	26	47	35	56	52	30.225
+ VAE + KG (1)	30	32	83	79	97	86	94	87	71.500
+ VAE + KG (1, 8)	35	36	88	85	99	95	96	90	76.800
+ VAE + KG (1, 8, 13)	**38**	35	**90**	**85**	98	**98**	89	**90**	**77.175**
+ VAE + shuffled KG (1, 8, 13)	13	27	39	57	67	84	56	74	58.825
+ VAE + CRF phrases (1, 8, 13)	29	25	81	77	98	85	94	82	68.075
+ VAE + KG (1, 8, 13), gate = 1	36	**36**	87	80	95	94	**95**	**90**	75.325
+ KG (1, 8, 13), no VAE	22	21	84	81	**99**	87	**95**	84	68.925

The two corrupted-retrieval controls probe different failure modes. Random memory tests whether arbitrary untrained vectors can serve as an additional conditioning channel; its 30.225 score rules out this explanation. Shuffled KG preserves the trained table and the same layer, gate, and fusion pathway, but changes which learned entries are paired with each instruction. It decreases six of eight Easy/Hard task cells and scores 18.350 points below correct retrieval and 3.125 below the +VAE control. Trained memory vectors alone are therefore insufficient: their semantic alignment with the instruction is essential. The 28.600-point gap between shuffled and random memory also suggests that trained entries retain some reusable structure when misassigned, although that structure is unreliable without correct addressing.

The placement sweep in Fig. 5 and Table 6 raises the VAE baseline from 61.950 to 71.500 with a Layer 1 KG insertion. Adding Layer 8 reaches 76.800, and Layer 13 gives 77.175. The small increment from $(1, 8)$ to $(1, 8, 13)$ indicates diminishing returns in this four-task ablation and is consistent with early memory access being most useful.

Gate probing in Fig. 5 supports this trend. In the vanilla layer sweep, performance peaks at shallow layers and decreases for deeper insertions. After one layer is selected, nearby subsequent layers are strongly suppressed in the next probing stage, suggesting that the inserted module has already absorbed much of the useful linguistic prior for downstream blocks. The greedy search therefore selects increasingly separated layers and finally adopts the $(1, 8, 13)$ configuration.

5 Conclusion

We presented Key-Gram, an external conditional-memory framework for language-conditioned manipulation. It retrieves task-specific key-gram entries and injects them at selected Transformer layers. Independent addresses localize new writes, and full-parameter baselines isolate the added memory path. Across RoboTwin2.0, LIBERO/LIBERO-Plus, and real-world dual-arm tasks, Key-Gram improves both π_0 and $\pi_{0.5}$, especially on hard, transfer, compositional, and continually expanded settings. Ablations support semantic phrase selection, adaptive gating, future-latent prediction, and early memory access. Broader validation remains future work.

AI USE STATEMENT

We used generative AI to assist writing and language polishing, retrieve and discover related literature, and generate synthetic language data. Manuscript preparation also included AI-assisted LaTeX formatting. Specifically, DeepSeek (`deepseek-reasoner`) diversified task instructions and extracted key-grams offline, as detailed in Appendix B. The generated instructions were fixed before training and shared by all model variants; instruction–key-gram pairs were cached before optimization.

REFERENCES

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. In *Proceedings of the 9th Conference on Robot Learning*, pp. 17–40, 2025.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, et al. Improving language models by retrieving from trillions of tokens. In *Proceedings of the International Conference on Machine Learning*, pp. 2206–2240, 2022.

Chi-Lam Cheang, Guangzeng Chen, Ya Jing, Tao Kong, Hang Li, Yifeng Li, Yuxiao Liu, et al. GR-2: A generative video-language-action model with web-scale knowledge for robot manipulation. *arXiv preprint arXiv:2410.06158*, 2024.

Chilam Cheang, Sijin Chen, Zhongren Cui, Yingdong Hu, Liqun Huang, Tao Kong, Hang Li, et al. GR-3 technical report. *arXiv preprint arXiv:2507.15493*, 2025.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, et al. RoboTwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025.

Xin Cheng, Rui Tian, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, et al. Conditional memory via scalable lookup: A new axis of sparsity for large language models. *arXiv preprint arXiv:2601.07372*, 2026.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Robotics: Science and Systems*, 2023.

Sudeep Dasari, Oier Mees, Sebastian Zhao, Mohan Kumar Srirama, and Sergey Levine. The ingredients for robotic diffusion transformers. *arXiv preprint arXiv:2410.10088*, 2024.

Ning Ding, Fangcheng Liu, Kyungrae Kim, Linji Hao, Kyeng-Hun Lee, Hyeonmok Ko, and Yehui Tang. MeKi: Memory-based expert knowledge injection for efficient LLM scaling. *arXiv preprint arXiv:2602.03359*, 2026.

Vincent Dumoulin, Ethan Perez, Nathan Schucher, Florian Strub, Harm de Vries, Aaron Courville, and Yoshua Bengio. Feature-wise transformations. *Distill*, 2018. doi: 10.23915/distill.00011.

Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, et al. LIBERO-plus: In-depth robustness analysis of vision-language-action models. *arXiv preprint arXiv:2510.13626*, 2025.

Robert M. French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999.

Google. Gemma 4 model overview, 2026. Google AI for Developers documentation, `https://ai.google.dev/gemma/docs/core`, accessed May 7, 2026.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. REALM: Retrieval-augmented language model pre-training. In *Proceedings of the International Conference on Machine Learning*, pp. 3929–3938, 2020.

Siyuan Huang, Liliang Chen, Pengfei Zhou, Shengcong Chen, Zhengkai Jiang, Yue Hu, Yue Liao, et al. EnerVerse: Envisioning embodied future space for robotics manipulation. *arXiv preprint arXiv:2501.01895*, 2025.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025a.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, et al. OpenVLA: An open-source vision-language-action model. In *Proceedings of the Conference on Robot Learning*, pp. 2679–2713, 2025b.

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474, 2020.

Lin Li, Qihang Zhang, Yiming Luo, Shuai Yang, Ruilin Wang, Fei Han, Mingrui Yu, et al. Causal world modeling for robot control. *arXiv preprint arXiv:2601.21998*, 2026.

Yue Liao, Pengfei Zhou, Siyuan Huang, Donglin Yang, Shengcong Chen, Yuxin Jiang, Yue Hu, et al. Genie envisioner: A unified world foundation platform for robotic manipulation. *arXiv preprint arXiv:2508.05635*, 2025.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 44776–44791, 2023.

Hong Liu, Jiaqi Zhang, Chao Wang, Xing Hu, Linkun Lyu, Jiaqi Sun, Xurui Yang, et al. Scaling embeddings outperforms scaling experts in language models. *arXiv preprint arXiv:2601.21204*, 2026.

Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. RDT-1b: A diffusion foundation model for bimanual manipulation. In *International Conference on Learning Representations*, 2025.

Guanxing Lu, Baoxiong Jia, Puhao Li, Yixin Chen, Ziwei Wang, Yansong Tang, and Siyuan Huang. GWM: Towards scalable gaussian world models for robotic manipulation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9263–9274, 2025.

Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, volume 24, pp. 109–165. Academic Press, 1989.

NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, et al. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4195–4205, 2023.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting language models with long-term memory. In *Advances in Neural Information Processing Systems*, volume 36, pp. 74530–74543, 2023.

Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. In *International Conference on Learning Representations*, 2022.

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, et al. X-VLA: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. In *International Conference on Learning Representations*, 2026.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Proceedings of the Conference on Robot Learning*, pp. 2165–2183, 2023.

APPENDIX

A FULL ROBOTWIN2.0 RESULTS

Table 7: Full RoboTwin2.0 results (%). Gains in parentheses for KG variants are relative improvements over their corresponding base backbones.

	π_0		$\pi_{0.5}$		π_0-KG (ours)		$\pi_{0.5}$-KG (ours)	
Task	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard
adjust_bottle	99	95	100	99	99	98	100	99
beat_block_hammer	79	84	96	93	92	90	100	94
blocks_ranking_rgb	80	63	92	85	87	81	94	89
blocks_ranking_size	14	5	49	26	55	36	71	60
click_alarmclock	77	68	98	89	75	78	99	92
click_bell	71	48	99	66	88	80	98	87
dump_bin_bigbin	88	83	92	97	90	85	94	89
grab_roller	98	94	100	100	100	98	100	98
handover_block	47	31	66	57	70	58	75	61
handover_mic	97	97	98	97	97	90	98	96
hanging_mug	14	11	18	17	31	29	38	36
lift_pot	80	72	96	85	92	91	95	97
move_can_pot	68	48	51	55	71	67	88	74
move_pillbottle_pad	67	46	84	61	75	63	96	81
move_playingcard_away	74	65	96	84	88	80	94	86
move_stapler_pad	41	24	56	42	73	67	91	83
open_laptop	71	81	90	96	90	93	92	96
open_microwave	4	32	34	77	69	78	80	84
pick_diverse_bottles	69	31	81	71	85	66	92	79
pick_dual_bottles	59	37	93	63	91	83	100	97
place_a2b_left	43	47	87	82	79	64	85	85
place_a2b_right	39	34	87	84	74	57	86	87
place_bread_basket	62	46	77	64	70	65	83	79
place_bread_skillet	66	49	85	66	76	72	80	77
place_burger_fries	81	76	94	87	94	81	92	88
place_can_basket	55	46	62	62	86	73	92	77
place_cans_plasticbox	63	45	94	84	72	63	94	85
place_container_plate	97	92	99	95	95	94	96	95
place_dual_shoes	59	51	75	75	78	79	86	83
place_empty_cup	91	85	100	99	90	94	96	98
place_fan	66	71	87	85	78	86	93	88
place_mouse_pad	20	20	60	39	52	41	75	70
place_object_basket	67	70	80	76	77	70	80	78
place_object_scale	57	52	86	80	73	68	87	82
place_object_stand	82	68	91	85	88	83	91	87
place_phone_stand	49	53	81	81	77	72	84	83
place_shoe	76	76	92	93	89	91	99	95
press_stapler	44	37	87	83	64	75	90	85
put_bottles_dustbin	65	56	84	79	82	76	87	85
put_object_cabinet	73	60	80	79	82	75	89	80
rotate_qrcode	74	70	89	87	88	83	86	84
scan_object	55	42	72	65	71	66	80	70
shake_bottle_horizontally	98	92	99	99	92	93	97	98
shake_bottle	94	91	99	97	95	92	99	96
stack_blocks_three	72	52	91	76	91	83	95	91
stack_blocks_two	93	79	97	100	98	91	100	99
stack_bowls_three	77	75	77	71	82	75	87	79
stack_bowls_two	94	95	95	96	99	95	98	98
stamp_seal	46	33	79	55	67	70	89	83
turn_switch	41	42	62	54	40	42	60	56

	π_0		$\pi_{0.5}$		π_0-KG (ours)		$\pi_{0.5}$-KG (ours)	
Task	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard
Avg. $_{\text{H=1}}$	66.5	61.6	85.1	80.2	80.4 (+20.9%)	77.9 (+26.4%)	90.8 (+6.6%)	87.0 (+8.5%)
Avg. $_{\text{H=2}}$	66.1	54.7	79.3	73.0	80.5 (+21.7%)	72.7 (+33.0%)	86.3 (+8.7%)	80.3 (+10.0%)
Avg. $_{\text{H=3}}$	61.6	50.2	78.6	67.4	79.4 (+28.9%)	70.2 (+39.8%)	86.8 (+10.4%)	80.8 (+19.9%)
Avg. $_{\text{50 Tasks}}$	65.9	58.4	82.7	76.8	80.3 (+21.9%)	75.6 (+29.5%)	89.0 (+7.6%)	84.4 (+9.9%)

B KEY-GRAM EXTRACTION PROMPT

We use the same DeepSeek API model, identified as `deepseek-reasoner`, for both language diversification and key-gram extraction, following the provider's official client invocation without custom decoding overrides. Generation is entirely offline. Before training, each demonstration trajectory is assigned a distinct diversified version of its original task instruction; hence, a training set with N trajectories contains N diversified instructions. The generated instructions are then fixed for the complete experiment. Every model variant, including the non-Key-Gram baselines, receives exactly the same diversified instructions, so language augmentation is controlled across comparisons.

For each diversified instruction, we separately extract its key-grams and cache the resulting instruction–Key-Gram pair before optimization. Neither training nor evaluation invokes the API, and no generated instruction or key-gram is manually edited. All API responses in our data-generation run satisfied the requested format and key-gram count, so no retry or repair rule was invoked. For simulation, the parser extracts exactly eight key-grams per instruction; for real-world experiments, it extracts 16. Both settings use the same extraction prompt, changing only the requested output count. Since RoboTwin2.0 contains diverse object-centric and long-horizon manipulation commands, the extraction prompt encourages short, compositional, and action-centered phrases that preserve task-relevant semantic roles, such as actions, objects, attributes, sources, targets, and spatial relations. The eight-key-gram simulation prompt used in our experiments is shown below.

```
SYSTEM_PROMPT = """
You are an expert parser for embodied manipulation instructions.

Your task is to convert one instruction into exactly 8 composed keywords.

These keywords must be short, natural, semantically meaningful phrases
↪  for embodied actions, not fragmented n-grams.

Rules:
1. Output exactly 8 keywords.
2. Each keyword must contain 2 to 4 words.
3. Prefer high-information phrases that combine multiple semantic roles
↪  in one phrase.
4. Prefer action-centered phrases over static descriptive phrases
↪  whenever possible.
5. At least 3 of the 8 keywords must explicitly contain an action verb.
6. Prefer these phrase types, in this priority order:
   a. verb + object + relation/target/source
   b. verb + particle + object
   c. verb + prep + object
   d. object + prep + object
   e. attribute + object
7. A good keyword should ideally compress 2 or more semantic elements,
↪  such as:
   - action + object
   - action + object + source
   - action + object + target
   - object + attribute
   - object + location
8. Use standalone static noun phrases only when they add important
↪  information that is not already covered elsewhere.
```

```
9. Use at most 5 standalone noun phrases.
10. If a static phrase can be replaced by a more informative action
↪  phrase, prefer the action phrase.
11. Prefer phrases like:
   - "pick up"
   - "pick bowl from drawer"
   - "pick up bowl"
   - "place bowl on plate"
   - "bowl in top drawer"
   - "black bowl"
12. Avoid:
   - fragmented phrases
   - fake combinations across unrelated spans
   - pronoun-centered phrases like "place it on"
   - low-information phrases
   - too many static environment phrases
   - duplicated semantics across multiple keywords
   - more than 4 words in a keyword
   - less or more than 8 keywords
13. Do not explain anything.
14. Return valid JSON only.

Example:
Instruction:
pick up the green sponge from the sink and wipe the wooden table near
↪  the window

Output:
{
  "keywords": [
    "pick and wipe",
    "pick sponge from sink",
    "pick up sponge",
    "green sponge",
    "wipe wooden table",
    "wipe table near window",
    "table near window",
    "wooden table"
  ]
}

MUST FOLLOW:
- Do NOT less or more than 8 keywords.
- Do NOT use more than 4 words in a keyword.
Must check the keywords strictly follow the rules above.

"""
```

C Ablation Protocol and Control Definitions

All component and layer-placement ablations use a dedicated four-task RoboTwin2.0 dataset and independently trained checkpoints. These data and checkpoints are disjoint from the main 50-task training run; the selected layer tuple $(1, 8, 13)$ is subsequently fixed for the reported main experiments. The $+$VAE configuration includes the parameter-matched residual adapter/MLP bound to future-latent prediction as well as the VAE loss, and therefore controls for trained auxiliary capacity rather than adding only an objective term. The random-memory row is a negative control that retrieves randomly initialized, untrained entries from this VAE checkpoint. The shuffled-KG control instead retains trained memory entries but breaks their instruction–Key-Gram correspondence, testing whether gains require content-matched retrieval rather than merely an additional memory pathway.

Key-Gram decomposition is part of the address definition rather than an interchangeable text preprocessor. Hashing an entire instruction would assign paraphrases and recombinations to example-level

sentence addresses, eliminating the reusable action–object–relation units that define the knowledge store. We therefore use CRFsuite phrases as the structurally matched non-semantic segmentation control: it preserves multiple phrase-level addresses and the same downstream memory architecture while changing how the addressable units are constructed.

D Real-World Protocol Details

Platform, control, and data collection. We use the unmodified Cobot Magic/Mobile ALOHA-style AgileX PiPER configuration and official master–slave data-collection and control procedure. Four six-degree-of-freedom PiPER arms comprise two leaders for demonstration and two followers for execution. Three synchronized RGB views provide one overhead image and one wrist image per follower; frames are recorded at 640×480 pixels and 50 Hz, then resized to 224×224 by the standard $\pi_0/\pi_{0.5}$ preprocessing pipeline. Each action is a 14-dimensional joint-delta vector with six joint increments and one gripper command per arm, and the policy predicts 50-step chunks. Different operators independently arrange training and evaluation layouts, and neither is shown the other's initial object placements.

Long-horizon tasks and scoring (Table 3). Picking moves all tableware items into the designated organizer. Sorting places food and tableware objects into their respective category-specific boxes. Assembly executes the complete four-part continual task: tidy the knife, fork, and plate; place the selected breads onto the plate; pull the serving-tray prop into position; and pour the breads onto the tray. Each task is evaluated with 50 physical-robot rollouts, and success requires every operation. A missed grasp, dropped object, or imminent collision terminates the rollout as failure with no retry. We impose no additional timeout or maximum-step cutoff. At inference, the controller executes all 50 predicted steps before replanning.

Object–target recombination (Table 4). Tasks 1 and 2 are two 50-trajectory in-distribution training settings: (1) pick the pen and charger and place them in the black bookshelf, and (2) pick the matchbox car and lighter and place them in the box. For Tasks 1–4, one checkpoint is jointly trained on the union of these 100 trajectories. Tasks 1 and 2 evaluate the two seen pairings. Task 3 recombines the pen with the lighter and sends them to their corresponding bookshelf and box targets; Task 4 recombines the charger with the matchbox car and likewise swaps their target assignment. Tasks 3 and 4 are evaluated zero-shot, with no trajectories from either recombination used for training or fine-tuning. Task 5 uses a separate sequential protocol: the model is first trained only on Task 1, then all parameters are updated using only the 50 Task 2 trajectories, without replaying Task 1, after which Task 1 is re-evaluated. Each of the five settings uses 50 physical-robot rollouts and the same success and termination criteria defined above. Thus, the columns distinguish seen execution, unseen entity–target composition, and post-update retention.

Continual compositional assembly (Table 5). The four stages form one incrementally lengthened task rather than four unrelated tasks. The complete object inventory contains seven physical objects: three tableware items (a knife, a fork, and a plate), three food items (a croissant, a baguette, and a round cake), and one cookware-like prop (a serving tray). The serving tray is a lightweight experimental prop representing a large tray rather than a real metal oven tray.

Stage 1 introduces the three tableware objects and trains the policy to tidy the knife, fork, and plate using picking and placing. Stage 2 introduces the three food objects and adds the milestone of selecting the breads and placing them onto the plate. Stage 3 introduces the serving-tray prop and adds a pulling action that brings the tray into the workspace. Stage 4 introduces no additional object but adds pouring, requiring the policy to pour the breads from the plate onto the serving tray. The ordered evaluation milestones are therefore: N_1, tableware tidied; N_2, selected breads placed on the plate; N_3, serving tray pulled into position; and N_4, breads poured onto the tray.

Training at Stage t starts from the complete Stage $t-1$ model checkpoint and keeps all model parameters trainable. Each stage contributes exactly 200 newly collected trajectories and is trained for two epochs. Crucially, the update uses only the new demonstrations for the single action step introduced at that stage; it does not replay trajectories from earlier stages. Evaluation nevertheless starts from the beginning of the cumulative task and uses 50 complete rollouts per stage, terminating at that stage's prescribed depth. Node j is the unconditional fraction of those same rollouts that

reaches milestone N_j, so reaching a later node certifies completion of all earlier nodes rather than defining a separately conditioned trial set. All stages use the same optimization configuration as the main experiments. Each model at each stage is trained once; because of the cost of full-parameter and physical-robot training, we do not repeat training with different random seeds or initializations.

E Limitations and Future Work

This work has several limitations. First, we instantiate Key-Gram only on π_0 and $\pi_{0.5}$; results on RoboTwin2.0, LIBERO, LIBERO-Plus, and one dual-arm platform do not establish generality across other VLA or world-model policies. Second, an external language-model parser extracts the key-grams, so accuracy, latency, and sensitivity to parser choice remain potential confounds. The present evaluation compares semantic extraction with CRFsuite phrase construction but does not vary the parser model, prompt, key-gram count, or fusion operator. Finally, results are reported as success-rate point estimates without repeated-training uncertainty or statistical tests. Broader backbones, parser-sensitivity studies, and replicated training runs are important next steps.

F Computational Resources

For full-parameter training, we use a compute node with eight NVIDIA H200 GPUs. The batch size is 16 per GPU (128 globally), with approximately 135 GB of GPU memory used per device. Training processes approximately 128 samples per second and typically takes 1–1.5 days per epoch on the complete RoboTwin2.0 dataset.