Making Your Action Policies Interpretable: Mixture of Action Queries

Suhyung Choi Youngseok Joo Hyundo Lee Kisung Shin Kyuhwan Shim
Chungwoo Lee Minjeong Gu Jun Ki Lee Byoung-Tak Zhang
Seoul National University, Seoul, Republic of Korea
{schoi,ysjoo,hdlee,ksshin,khshim,chlee,mjgu,jklee,btzhang}@bi.snu.ac.kr

Abstract

Recent advances in large vision language action models and chunk based or parallel decoding have enabled robust and precise low level robotic control from visual observations and language instructions. Yet most policies generate action chunks from a single fixed embedding or placeholder tokens, which forces one query to account for all behaviors and limits adaptivity across environments and tasks. Motivated by neuroscience on action selection, we model action chunks as mixtures of multiple primitive action queries, enabling adaptive and interpretable representations. We introduce the Mixture of Action Queries (MAQ), a lightweight and model agnostic module for existing parallel decoders. MAQ composes each chunk from a small set of learned action queries that represent reusable skills. A learnable router conditions on the current vision and language context and assigns mixture weights to form the chunk. MAQ integrates into existing decoders without changes to the backbone or latent dimensionality. We validate MAQ by integrating it into ACT and recent state of the art VLA models. Across real world and simulated settings, MAQ improves performance and provides chunk level interpretability that single query designs do not offer. In multi task training, the router assigns consistent queries to the same primitives across tasks, indicating reusable task agnostic skills and providing human interpretable insight into policy behavior.

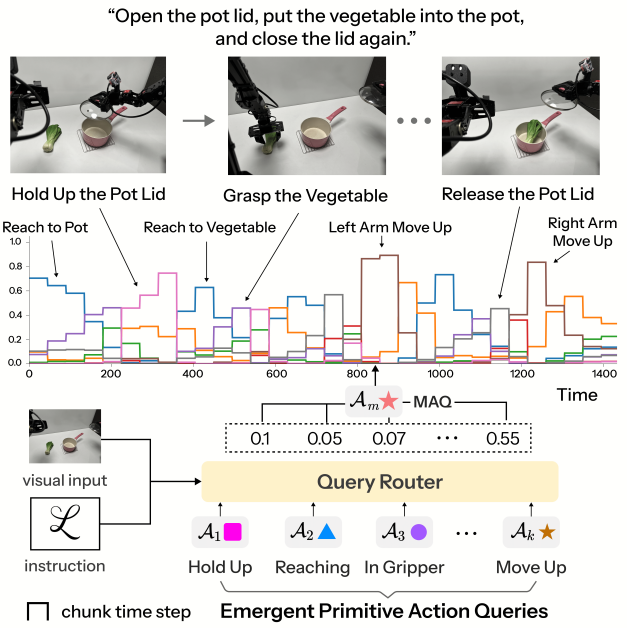

Figure 1. **Action Chunks from Mixture of Action Queries.** Given the task "Open the pot lid, put the vegetable into the pot, and close the lid again," MAQ forms each chunk by mixing action queries. The top row illustrates three phases of the task. The timeline shows mixture weights over queries A_1 to A_k under the current instruction and visual input. At the highlighted chunk time step the router assigns higher weight to queries that best support the scene and goal, and the resulting mixture produces the action chunk.

1. Introduction

Recent progress in vision and language instruction–based manipulation tasks has been driven by advances in large vision-language-action (VLA) models and large-scale datasets [2, 4, 11–13, 15, 19, 25]. These developments have enabled robust and fine-grained low-level robotic control across diverse tasks and embodiments [8, 19, 23, 24]. To achieve real-time inference with large VLA models, chunked or parallel decoding–predicting a short horizon of future controls in a single step–has become a common design choice [2, 13, 23]. This approach reduces the effective planning horizon, mitigates compounding errors in behavioral cloning, and stabilizes rollouts under latency.

Despite these advancements, the action embeddings used to generate action chunks are often naively designed, either as fixed vectors [1, 23] or as placeholder queries [11, 13].

Such designs force the model to learn how to map every required action chunk from a single action embedding, which in turn leads to overfitting to specific environments or tasks and limits generalization.

In this work, we address this limitation by replacing placeholder action embeddings with multiple action queries, where each query models primitive skills rather than the entire action space. Instead of relying on a single embedding, the model composes each action chunk as an output of a mixture of multiple queries. Neuroscience studies support this view; multiple candidate actions are prepared in parallel and compete during selection [6], and goals and context bias selection toward task-relevant behaviors [18]. Inspired by these findings, we propose modeling actions as combinations of multiple queries, as illustrated in Figure 1, where a single task emerges as a composition of primitive skills.

Building on this idea, we introduce **MAQ (Mixture of Action Queries)**: a lightweight module for chunk-based VLA models. Rather than emitting a chunk from a single fixed query, MAQ composes each chunk from a small set of primitive action queries that capture reusable skills. A learnable router conditions on the current language and vision context and dynamically mixes these queries into a single chunk. The method is model-agnostic and can be integrated to existing chunked decoders without modifying the backbone or latent dimensionality. Unlike fixed-query policies, which lack adaptivity, MAQ leverages mixtures whose weights vary with context and observation, making generated chunks both adaptive and interpretable.

We validate MAQ by integrating it as a drop in module into ACT [23] and recent state of the art VLA models, including OpenVLA-OFT [13] and CLIP-RT [11]. With MAQ, primitive action skills emerge, which leads to higher success rates, improved rollout stability, and greater interpretability of action chunks. Analyses across real world tasks, the ALOHA simulation [23], and the LIBERO benchmark [16] show that the router consistently assigns higher weights to the same queries for the same primitives across tasks, indicating that it autonomously discovers reusable task agnostic skills. To the best of our knowledge, this is the first study to interpret chunk level actions by modeling competition among action queries that self organize into primitive skills. Overall, mixtures of action queries enable precise control and provide human interpretable insights into policy behavior. Our contributions are threefold.

- **Problem formulation.** We identify limitations of fixed or placeholder action embeddings in chunked decoding and formalize the need for compositional action representations.
- **Methodology.** We present MAQ, a lightweight and model agnostic module that composes each action chunk from multiple primitive action queries via a learnable router.
- **Empirical validation.** We demonstrate that MAQ improves performance, yields interpretable skill composition, and discovers reusable primitives across real world and simulated environments.

2. Related Works

VLA Policies and Parallel Action Decoding. Vision-Language-Action (VLA) policies [2, 4, 11–13, 15, 19, 25] map visual observations and natural language instructions to action trajectories. Since ACT [23] introduced action chunking with parallel decoding for imitation learning, chunked actions have become a standard design choice. Subsequent work has focused on composing chunks to improve performance while reducing latency [13], studying the speed and efficiency of chunking [3], and building generalist policies [2]. Despite this advances, prior methods largely overlook the query that drives parallel decoding, the critical input that conditions each chunk. Many approaches either fix a single query embedding [1, 23, 24] or use dummy placeholder tokens [11, 13]. In contrast, we examine the design of the query itself and its impact on parallel decoding.

Mixture-of-Experts (MoE). The MoE framework was introduced in [9] and [10]. Its central idea is to allocate model capacity across specialized components, or experts, each trained to handle a different part of the problem. In sparse MoE, a learned router scores experts and routes each input to a small subset, typically the top k, which concentrates compute while keeping per example cost low [20]. Unlike conventional MoE, we do not add expert modules at the network level. Instead, we operate in the query embedding space that generates action chunks for the decoder. A lightweight router mixes a small set of input conditioned query embeddings, which makes the mechanism easy to integrate with diverse policies and requires only a modest number of additional parameters. We also employ dense routing rather than top k selection, so all query embeddings associated with primitives can contribute when needed.

Codebook-based Skill Policies. Another line of work [14, 17, 21, 22] discretizes actions or skills with vector quantization and then decodes the codes back to continuous control, which makes the invoked skill explicit at the token level. These approaches operate in a latent or intermediate space and leave the query used for parallel decoding unchanged. To our knowledge, no prior method provides an interpretable account of which skill or action is selected inside parallel action policies. In contrast, we provide an interpretable basis by using query embeddings that are directly tied to action chunks, which makes the method plug and play with any parallel action policy without modifying the backbone or introducing an extra latent space.

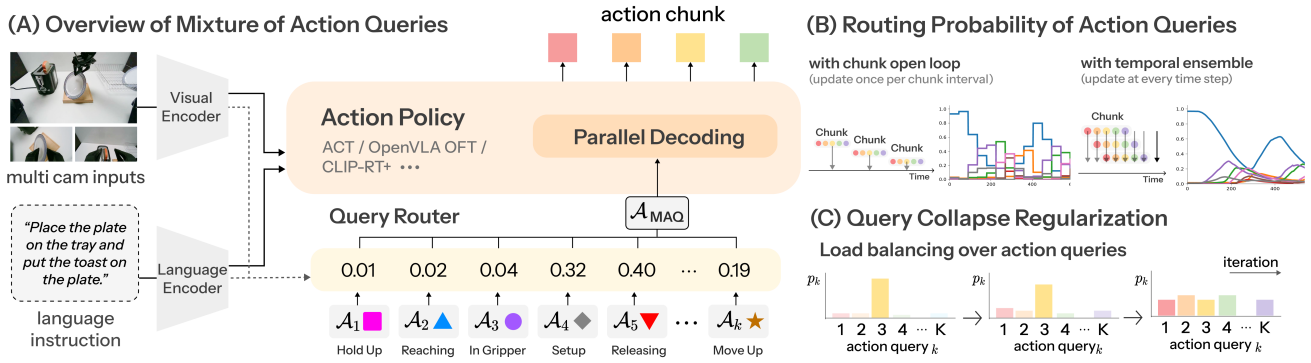

Figure 2. **Overview of MAQ and Query Routing.** (A) Visual and language encoders feed a parallel action decoder. A query router mixes action queries to form a chunk query, and the decoder turns it into an action chunk. (B) Routing probabilities under two update schemes. Chunk open loop updates once per chunk interval. Temporal ensemble updates at every step and averages within the chunk window. (C) Regularization to prevent query collapse. A load balancing term encourages uniform utilization across action queries.

3. Method

3.1. Preliminaries

We describe two broad designs for parallel action decoding policies. We begin with policies that use cross attention from queries to modality tokens [1, 23, 24], then cover decoder only policies that use self attention over queries and modality tokens [2, 11, 13]. **Cross Attention Based Policies.** To clarify the role of the query in parallel action policies, we start from DETR [5], the base architecture behind ACT [23]. In DETR, K *query slots* are decoded in parallel and each slot is refined into an object prediction. ACT adapts this decoder to output a sequence of K actions, which we call an action chunk. DETR uses a learnable *query embedding* with the same dimensionality as a slot. This embedding is added to the slot representation at every decoder layer and guides its refinement toward a specific target. In ACT, the same mechanism guides each action slot toward a coherent action sequence.

A Transformer decoder maintains K query slots, where K is the action chunk length. At layer ℓ, each slot cross attends to the encoder context X, which concatenates tokens from the image, language, and proprioception encoders. A learned query embedding $Q \in \mathbb{R}^{K\times d}$ is added to the slots at each layer to guide refinement. Concretely, for $\ell = 1, \ldots, N$,

$$s^{(\ell+1)} = \text{CrossAttn}\Big(q = s^{(\ell)} + Q,\ k = X,\ v = X\Big), \tag{1}$$

where $s^{(\ell)} \in \mathbb{R}^{K\times d}$ is the slot matrix at layer ℓ, $Q \in \mathbb{R}^{K\times d}$ is the learned query embedding, and $X \in \mathbb{R}^{T\times d}$ is the encoder output. After the final layer, an action head maps the hidden states to the robot action space:

$$A = \text{ActionHead}\Big(s^{(N)}\Big) \in \mathbb{R}^{K\times D_{\text{act}}}. \tag{2}$$

Self Attention with Decoder Only Policies. Decoder only policies, including OpenVLA-OFT [13] and related models [2, 11], pair a decoder only Transformer with vision and language inputs and do not separate the query slot from the query embedding. Instead, they insert K fixed query placeholder tokens. These placeholders are concatenated with the language and vision tokens, and the Transformer processes the entire sequence. Self attention propagates context into the placeholder positions. After the final layer, the K hidden states at those positions are extracted, and an MLP maps them to an action chunk.

$$X = [X_{\text{lang}},\ X_{\text{vis}},\ X_{\text{placeholder}}], \quad H = \text{Decoder}(X),$$
$$S = H_{\text{placeholder}} \in \mathbb{R}^{K\times d}, \quad A = \text{MLP}(S) \in \mathbb{R}^{K\times D_{\text{act}}}. \tag{3}$$

$X_{\text{placeholder}}$ and $H_{\text{placeholder}}$ denote the K placeholder tokens and their hidden states.

3.2. Mixture of Action Queries

We introduce *Mixture of Action Queries* (MAQ), a plugin method for parallel action decoding policies that endows queries with primitive action level semantics, making chunk level behavior interpretable and reusable as skills. MAQ adaptively selects and weights a set of action queries based on the current observation, forming a context weighted mixture instead of committing to a single query. The mixture components capture reusable skills, so the resulting query carries task aligned semantics. We describe the core architecture, the training objective, and how to integrate it into state of the art parallel action policies.

Mixtures and Routing for Action Queries. In parallel action decoding policies, a query specifies what information to retrieve and attend to from vision, language, and proprioception to produce a chunk of K actions. Thus the query is a primary input on par with the modality tokens. Formally,

we write

$$\pi_\theta(a_{t:t+K-1} \mid l_t, v_t, p_t, Q)\,, \tag{4}$$

where l_t is the instruction at time t, v_t is the visual input, p_t is proprioception, and Q denotes the query input. The policy outputs an action sequence of length K from step t through $t+K-1$. Here Q is either a fixed query embedding or a collection of placeholder tokens as defined above.

Unlike a fixed query embedding or placeholder tokens, MAQ maintains a set of N action queries

$$\{\, Q_1, \ldots, Q_N \,\}, \qquad Q_i \in \mathbb{R}^{K\times D}, \tag{5}$$

where K is the chunk length and D is the hidden dimension. A router assigns context dependent weights to these queries. Given language, vision, and proprioception features (ℓ, v, p), the router produces logits

$$\mathbf{g}_t = h(\ell, v, p) \in \mathbb{R}^N, \qquad \mathbf{w}_t = \mathrm{softmax}(\mathbf{g}_t) \in \mathbb{R}^N, \tag{6}$$

where $\mathbf{w}_t$ is a probability vector whose components sum to one. Similar to mixture of experts [9, 10], the skill queries compete under the current context and the router outputs a distribution rather than selecting a single winner. The mixed chunk query is

$$Q_t \;=\; \sum_{i=1}^{N} w_{t,i}\, Q_i \;\in\; \mathbb{R}^{K\times D}. \tag{7}$$

We use Q_t in place of a fixed query within parallel action decoders.

3.3. Integrating MAQ Across Action Policy Architectures.

MAQ is architecture agnostic and can be integrated to any parallel action policy. Because the conditioning context differs across models, we configure the router for the available modalities and instantiate MAQ across representative architectures. We present three practical guidelines for this integration.

Perceiver Style Latent Pooling Router. Many action policies process a large number of modality tokens from vision [11, 13, 23] and in some cases from language [11, 13] and proprioception [13, 23]. A naive router that flattens all tokens $X \in \mathbb{R}^{T\times D}$ and maps them directly to N query weights incurs a large parameter cost. To reduce this cost, we summarize the context with a small set of latent tokens before routing. Let $X \in \mathbb{R}^{T\times D}$ be the concatenated encoder tokens. We introduce L learnable latent tokens $R \in \mathbb{R}^{L\times D}$ and first apply attention in the original dimension D:

$$\begin{aligned} R' &= \mathrm{SelfAttn}(R), \\ \tilde{U} &= \mathrm{CrossAttn}(q = R',\, k = X,\, v = X) \in \mathbb{R}^{L\times D}. \end{aligned} \tag{8}$$

We then project to a lower dimension for routing using a linear layer or a small MLP:

$$U = f_{\mathrm{proj}}(\tilde{U}) \in \mathbb{R}^{L\times D'}, \qquad D' < D. \tag{9}$$

Finally, we flatten U to $\mathbf{z} \in \mathbb{R}^{LD'}$ and use a small MLP head to produce router logits and weights. Because $LD' \ll TD$, the router remains compact while still conditioning on the full context. We use this latent pooling router when applying MAQ to ACT.

Token Preserving Router and Mean Pooling Router. As an alternative, when the number of modality tokens is small or when Perceiver style latent pooling is impractical, we use either a token preserving router or a mean pooling router. The token preserving router flattens the modality tokens into a single vector without pooling and maps it to mixture weights. The mean pooling router averages tokens within each modality and uses the pooled tokens as the router input.

CLIP-RT+. Following Section 3.1, CLIP-RT+ [11] concatenates one vision token and one language token with K placeholder tokens. The router input has shape $\mathbb{R}^{2\times d}$. We flatten this input and apply a linear projection to obtain mixture weights $\mathbf{w} \in \mathbb{R}^N$. Using $\mathbf{w}$, we mix the action queries $\{Q_i\}_{i=1}^N$ and replace the K placeholder tokens with the mixed chunk query $Q_t \in \mathbb{R}^{K\times D}$ in the decoder input.

OpenVLA-OFT. OpenVLA-OFT [13] also follows the decoder only design in Section 3.1, but unlike CLIP-RT+, it exposes long sequences of language and vision tokens. Although Perceiver style pooling could summarize these tokens, we found it unstable when fine-tuning the pretrained action model with low rank adaptation. Instead, we apply mean pooling within each modality to obtain one token for each modality. We pool vision together with proprioception into one token, concatenate this token with the pooled language token, and feed the result to the router to produce $\mathbf{w} \in \mathbb{R}^N$. This yields a stable and compact router for large models while preserving conditioning on both modalities.

3.4. Query Collapse Regularization

Mixture and codebook methods often collapse to a single expert or code, which harms coverage and interpretability [7, 20]. We avoid collapse with two lightweight mechanisms: a temperature schedule on the router softmax and a load balancing term that encourages even query utilization [20, 26].

Temperature scheduling at the router. Let $\mathbf{g}_t \in \mathbb{R}^N$ be router logits at step t and let $\tau > 0$ be a temperature. We form gates with a temperature scaled softmax

$$\mathbf{p}_t = \mathrm{softmax}\big(\mathbf{g}_t/\tau\big), \qquad p_{t,m} = \frac{\exp(g_{t,m}/\tau)}{\sum_{j=1}^{N} \exp(g_{t,j}/\tau)}. \tag{10}$$

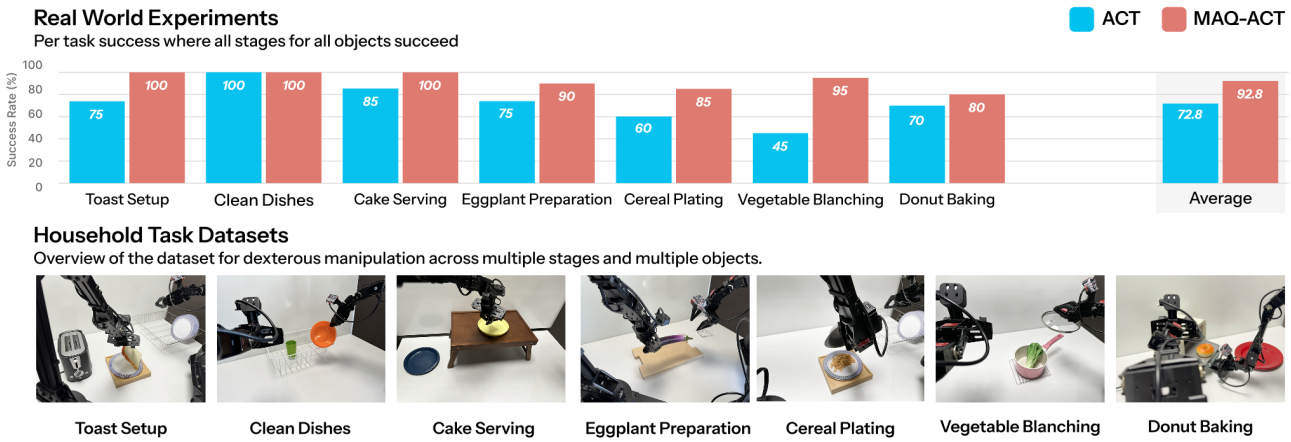

Figure 3. **Real world experiments for seven household tasks & Mobile ALOHA dataset.** Bars show the success rate per task over 20 trials, with the overall average on the right. The images below illustrate our mobile ALOHA dataset. We collected 490 teleoperated episodes on a dual arm mobile ALOHA platform with 14 DoF control. Each episode contains 1,250 time steps sampled at 50 Hz, yielding 612k action steps in total. The tasks cover seven household activities with a multi stage structure, including reach, align, grasp, and place. We use this dataset to evaluate long horizon control and to test whether chunk mixtures align with reusable skills. We will release the dataset and the evaluation scripts upon publication.

Lower τ concentrates mass on fewer queries, while higher τ spreads mass more evenly. During training we anneal τ with a cosine schedule to shift from exploration to specialization.

Load balancing loss. To avoid persistent preference for a few queries, we match the average gate to a uniform prior. Let $\bar{\mathbf{p}} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_t$ be the gate averaged over a batch or sequence and let $\mathbf{u} = [\frac{1}{N}, \ldots, \frac{1}{N}]$ be the uniform distribution over N queries. We add a small KL penalty as below.

$$\mathcal{L}_{\mathrm{LB}} = \lambda_{\mathrm{lb}}\, \mathrm{KL}(\bar{\mathbf{p}} \,\|\, \mathbf{u}) = \lambda_{\mathrm{lb}} \sum_{m=1}^{N} \bar{p}_m \log\big(N\, \bar{p}_m\big) \qquad (11)$$

Minimizing this term spreads usage across queries by encouraging high-entropy average gates, which reduces collapse and makes routing more stable. The objective serves the same role as auxiliary balancing losses in sparse MoE layers that control expert importance and load [7, 20, 26].

3.5. Objective.

Ultimately, our training loss is the sum of an L1 task loss and a balancing term.

$$\mathcal{L} = \mathcal{L}_{\mathrm{task}} + \mathcal{L}_{\mathrm{LB}} \qquad (12)$$

Here, $\mathcal{L}_{\mathrm{task}}$ is the L1 loss between the predicted action chunks and the target. The balancing term $\mathcal{L}_{\mathrm{LB}}$ encourages balanced usage of queries and prevents collapse, leading each query to specialize into a primitive behavior that is semantically meaningful. We analyze the emergent action queries learned by this objective in the experiments section.

4. Experiments

We evaluate MAQ along three dimensions: effectiveness, interpretability, and compatibility across architectures. We measure success rates for various tasks and find that MAQ yields context appropriate action chunks, with improvements that track the learned mixture over action queries. For interpretability, we visualize router gating weights and observe that queries specialize into primitive skills and that mixtures compose new behaviors within the chunk window. For compatibility across architectures, we integrate MAQ into ACT [23], CLIP-RT+ [11], and OpenVLA-OFT [13] and evaluate across multiple environments, demonstrating broad compatibility and enabling detailed analysis of its effects.

4.1. Real-World Experiments

In this section, we integrate MAQ into ACT and evaluate it in real-world scenarios on bimanual manipulation platforms.

Dataset and Experimental Setup. Following recent VLA work [2, 8, 13], we collect a dataset using Mobile ALOHA [8] and evaluate on the same platform. The dataset covers seven household tasks in a bimanual tabletop setting. Each task comprises a sequence of primitive skills, including approaching, grasping, lifting, placing, pouring, uncapping, and raising or lowering the arms. All tasks involve manipulating two or more objects. Examples are shown in Fig. 3. Each episode contains 1,250 time steps sampled at 50 Hz, yielding 25 s per episode. In total, the dataset contains 612,500 frames with aligned actions. Public Mobile ALOHA datasets are limited, so we will release ours

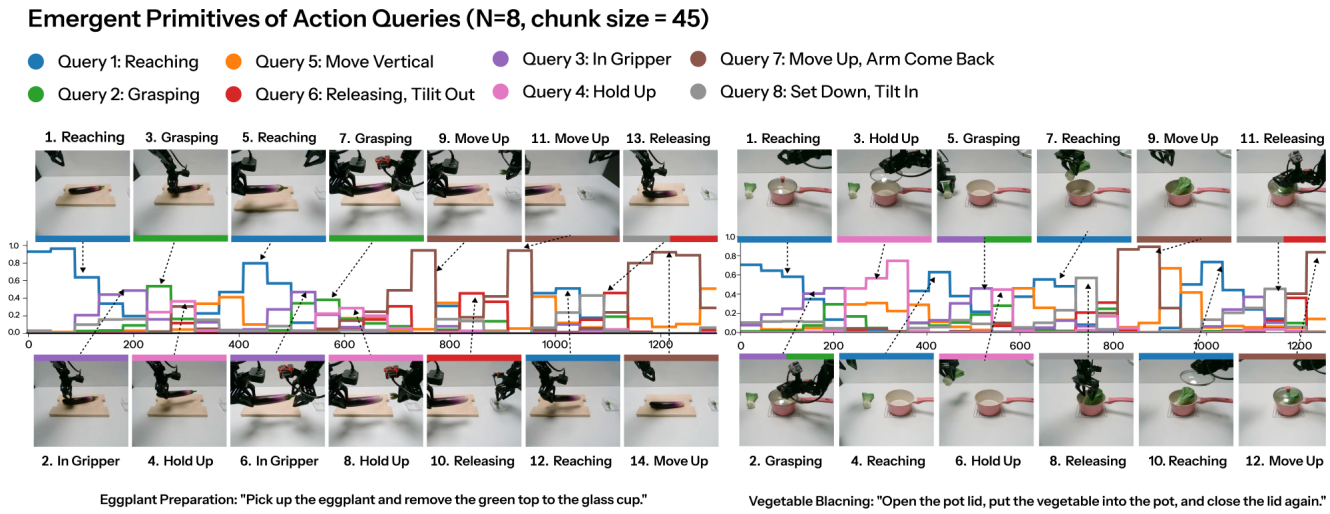

Figure 4. **Emergent primitives of action queries in real world rollouts.** The legend maps the $N = 8$ queries to primitive skills. Colored plot router mixture weights over time while decoding with chunk size $K = 45$; thumbnails mark representative steps. Left, Eggplant Preparation; right, Vegetable Blanching. Peaks identify the dominant primitive for each chunk, and overlapping peaks within a chunk reveal composition, for example *In Gripper* with *Grasping* and *Set Down* with *Releasing*. The same primitives recur across tasks and arms, indicating reusable skills.

upon publication to support research on primitive skills and robotic manipulation.

MAQ-ACT and Baseline. For real world experiments, we adopt ACT [23], a widely used imitation learning baseline, as our starting point. As noted in Section 3.1, ACT is a parallel action decoder with a single query embedding. On the Mobile ALOHA platform without the mobile base, it outputs 14 DoF action chunks by predicting seven joints per arm, and we train it with an ℓ_1 loss. We configure MAQ-ACT by adding an action query set and a query router, and we train with the temperature scheduling and load balancing losses described in Section 3.4. We set the number of action queries to eight, which we treat as a hyperparameter. For a fair comparison, ACT uses three Transformer decoder layers and MAQ-ACT uses two, so the overall parameter counts are comparable. Both models use an action chunk length of 45 following the real world configuration in [8]. We train ACT and MAQ-ACT jointly on all seven tasks.

Evaluation Protocol. We report success rates for seven household tasks, as shown in Fig. 3. A trial is counted as a success only if the policy completes all manipulation stages for all relevant objects. We run 20 trials per task. For example, the toast setup task requires lifting the bowl and placing it on the tray, then removing the toast from the toaster and placing it on the plate.

4.2. Real-World ALOHA Task Performance Results

Figure 3 reports performance across all seven real world tasks compared with the ACT baseline, and Figure 5 illustrates how integrating MAQ into ACT improves precise manipulation by activating the appropriate primitive action queries. As shown in Figure 3, attaching MAQ to ACT improves performance across all seven tasks. Gains are especially pronounced on Eggplant Preparation and Vegetable Blanching, which require precise manipulation and tight spatial alignment. For tasks with looser spatial tolerance, such as Clean Dishes, the baseline ACT already performs well, and the margin is smaller.

4.3. Interpretability and Emergent Skills.

Figure 4 summarizes query activations across tasks. During evaluation we record the router probabilities for every chunk. These are the mixture weights over action queries. We aggregate them into a per query activation map over the full trajectory. The map shows which queries are active at each chunk step and how their usage changes over time.

We consistently find eight primitives with stable roles across tasks. *Reach to Object* approaches a grasp or moves toward a placement goal. *In Gripper* marks contact. *Grasping* closes and secures. *Hold Up* lifts after grasp. *Move Up or Arm Return* raises to relocate or retreats after placement. *Set Down or Tilt Inward* makes contact before release or resets the wrist after pouring. *Release or Tilt Outward* opens the gripper or rotates for pouring. *Vertical Adjustment* makes small vertical motions while holding.

These queries act as task agnostic and arm agnostic primitives. For example, when either arm reaches toward an object, the *Reach to Object* query tends to dominate. Across all seven real world tasks we observe query activations that respond to specific actions. Some primitives are standalone,

such as *Reach to Object*, *Hold Up*, and *Move Up*. Others often appear as sequences, for example *In Gripper* with *Grasping*, and *Set Down* with *Release*. Across our real world tasks, a common pattern emerges. Most involve two objects, so the typical flow is to reach the first object, manipulate it, then reach the second object, manipulate it, and return to a neutral pose. The activation maps in Figure 4 trace this flow. Together these patterns provide direct evidence of emergent skills and motivate the analyses that follow. Beyond these aggregate activation patterns, we provide additional representative failure case analysis in Appendix A.1 to understand how MAQ changes the behavior of ACT. The analysis shows that MAQ-ACT improves not only the final predicted actions, but also the timing of transitions between primitive skills. In particular, MAQ-ACT tends to shift from *In Gripper* to *Grasping* near contact and then to *Hold Up*, which helps reduce premature gripper closure and stabilize rollouts.

4.4. Simulation Experiments

In addition to real-world results, we use simulation to further substantiate the effectiveness of MAQ. In the ALOHA simulation [23], we assess whether MAQ selects contextually appropriate queries, increases representational capacity compared to fixed query designs, and improves model performance. We also examine compatibility with parallel action decoding models by integrating MAQ into several models in the LIBERO benchmark [16]. Across these settings, mixtures of queries match or improve success rates and policy metrics, and yield chunk-level primitive actions that are interpretable.

4.5. ALOHA Simulation Results and Analysis

We compare ACT and MAQ-ACT in ALOHA simulation on *Peg Insertion* and *Transfer Cube*. Both models are trained jointly on the two tasks. We report success rates with and without a *temporal ensemble*. In the ensembled setting the policy decodes a chunk at each time step and aggregates overlapping predictions to produce the control. With MAQ, the router forms a query mixture at every time step.

Results. Table 1 summarizes success and reward with temporal ensemble. On *Transfer Cube*, all methods reach near-perfect success. On *Peg Insertion*, MAQ-ACT with $N = 8$ achieves 0.73 ± 0.01, outperforming ACT at 0.50 ± 0.02. MAQ-ACT with $N = 4$ and $N = 16$ also improves over ACT, achieving 0.58 ± 0.03 and 0.57 ± 0.01, respectively. These results show that MAQ improves performance on the precision-sensitive *Peg Insertion* task while maintaining strong performance on the easier *Transfer Cube* task.

Analysis. *Peg Insertion* is precision sensitive, requiring alignment of about $5\,\mathrm{mm}$. Figure 5(b) shows that MAQ-ACT often recruits an insertion focused query that becomes dominant near contact and remains active through insertion, while ACT relies on one fixed query to handle both grasping and insertion. Predicting a chunk and router weights at every time step reduces sensitivity to chunk phase and aligns control with contact evidence, which is consistent with the improved scores on *Peg Insertion*.

4.6. LIBERO Simulation and Baselines and Results.

To demonstrate integration with parallel action policies on LIBERO, we apply MAQ to the state of the art models CLIP-RT+ [11] and OpenVLA-OFT [13]. We evaluate on LIBERO Long, which contains ten diverse long horizon tasks. Both models follow the decoder only design described in Section 3.1. We set the number of action queries to 16 for CLIP-RT+ and 10 for OpenVLA-OFT. Table 2 reports the results. Attaching MAQ to CLIP-RT+ increases the average success rate by 0.8 percentage points with a parameter overhead of about 10M, which is 0.77% of the CLIP-RT+ model size. MAQ also exposes clear primitive skill queries that the original placeholder tokens could not reveal. For OpenVLA-OFT, MAQ adds $655\,\mathrm{k}$ parameters, which is 0.0087% relative to the 7.5B backbone, and improves performance by 0.3 percentage points over the baseline. Even in this case, MAQ provides interpretability across the ten long horizon tasks in LIBERO-Long. These results indicate that MAQ can be attached to large models and diverse parallel action policies with minimal overhead, yielding gains or parity while offering a transparent account of policy behavior.

4.7. Discussion of Query Mixtures and Primitive Skills

Even without labels or hand crafted supervision, the proposed architecture leads learned queries to self organize into primitive action skills. This is especially useful for chunk based decoders, where each output chunk can be explained as a mixture of a few primitives.

Effect of query count on composition. In the case of MAQ-ACT in the real world setting, a moderate number of queries $(N = 8)$ gives each query a clear meaning, and chunks are formed by mixing a small set of recognizable primitives. For example, when a chunk must grasp and lift, *Grasping* and *Hold Up* co-activate in the same chunk. In CLIP-RT+, we find that using more action queries than the $N = 10$ setting can further decompose behavior into finer-grained primitives and enable richer compositions. For example, a move to the upper left can emerge from the joint activation of *move up* and *move left*. However, very large N may fragment query usage, so we consider $N = 8$ to $N = 16$ a practical range.

Router design and mixture behavior. MAQ-ACT uses the Perceiver style latent pooling router in Section 3.3, while CLIP-RT+ uses a token preserving concatenation fed to an

Table 1. **ALOHA simulation results on *Peg Insertion* and *Transfer Cube* with temporal ensemble.** We report success rate and reward for ACT and MAQ-ACT with N action queries. MAQ-ACT with $N = 8$ achieves the best success rate and reward on *Peg Insertion*, while all methods reach near-perfect success on *Transfer Cube*. Values are mean $\pm$ deviation.

	Peg Insertion		***Transfer Cube***	
Model	Success	Reward	Success	Reward
ACT	0.50±0.02	392.35±1.54	1.00±0.00	632.40±3.79
MAQ-ACT ($N = 4$)	0.58±0.03	353.29±15.83	1.00±0.00	645.18±14.45
MAQ-ACT ($N = 8$)	**0.73**±0.01	**456.94**±1.73	0.99±0.01	645.48±13.46
MAQ-ACT ($N = 16$)	0.57±0.01	413.07±10.28	1.00±0.00	**693.88**±12.87

Table 2. **LIBERO-Long results for CLIP-RT+ and OpenVLA-OFT with and without MAQ.** We report average success over ten long-horizon tasks and the relative parameter increase. MAQ improves CLIP-RT+ by 0.8 percentage points with 0.77% additional parameters, and improves OpenVLA-OFT by 0.3 percentage points with only 0.0087% overhead.

Model	**LIBERO-Long (%)**	**Param ↑ (%)**
CLIP-RT+	83.8	-
CLIP-RT+ (+MAQ)	**84.6**	0.77
OpenVLA-OFT	94.5	-
OpenVLA-OFT (+MAQ)	**94.8**	0.0087

MLP. Both yield clear specialization into primitive skills. OpenVLA-OFT uses a mean token router for stability during LoRA fine tuning. This produces sparser activations with less fine detail, which is expected when hundreds of vision, language, and proprioception tokens are averaged into one. We adopt this design for stability on the 7B backbone and leave applying a Perceiver style router to OpenVLA-OFT as future work.

Interpreting mixture weights. The executed motion is shaped by the whole mixture rather than the largest component alone. Lower weight queries still influence direction and timing, and their joint activation produces richer motions within a single chunk. These observations suggest that policy behavior can be described by a small set of basis queries, with chunks expressed as linear combinations of those elements. Exploring low dimensional structure and principal component like bases is a promising direction for future work.

5. Conclusion

In this work, we revisited the naive design of action embedding, which is often treated as fixed vectors or placeholder tokens, and proposed a new perspective that models them as mixtures of primitive action queries. By introducing MAQ (Mixture of Action Queries), we showed that parallel VLA decoders can generate adaptive and interpretable control sequences through mixtures of action queries rather than fixed embeddings. Our experiments across real-world manipulation tasks, ALOHA simulation, and the LIBERO benchmark confirmed that MAQ improves success rates and rollout stability while uncovering task-agnostic primitives that recur across environments for various VLA models. These findings highlight that chunk-level actions can be decomposed into compositional skills, offering both practical performance gains and human-interpretable insights into policy behavior.

Acknowledgments

This work was partly supported by grants from the IITP (RS-2021-II211343-GSAI/10%, RS-2022-II220951-LBA/15%, RS-2022-II220953-PICA/15%), the NRF (RS-2024-00353991-SPARC/15%, RS-2023-00274280-HEI/15%), the KEIT (RS-2025-25453780/15%), and the KIAT (RS-2025-25460896/15%), funded by the Korean government.

References

[1] Homanga Bharadhwaj, Jay Vakil, Mohit Sharma, Abhinav Gupta, Shubham Tulsiani, and Vikash Kumar. Roboagent: Generalization and efficiency in robot manipulation via semantic augmentations and action chunking. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4788–4795. IEEE, 2024. 1, 2, 3

[2] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024. 1, 2, 3, 5

[3] Kevin Black, Manuel Y Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. *arXiv preprint arXiv:2506.07339*, 2025. 2

[4] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al.

Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022. 1, 2

[5] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020. 3

[6] Paul Cisek. Cortical mechanisms of action selection: the affordance competition hypothesis. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 362(1485): 1585–1599, 2007. 2

[7] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022. 4, 5

[8] Zipeng Fu, Tony Z Zhao, and Chelsea Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. *arXiv preprint arXiv:2401.02117*, 2024. 1, 5, 6

[9] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991. 2, 4

[10] Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2): 181–214, 1994. 2, 4

[11] Gi-Cheon Kang, Junghyun Kim, Kyuhwan Shim, Jun Ki Lee, and Byoung-Tak Zhang. Clip-rt: Learning language-conditioned robotic policies from natural language supervision. *arXiv preprint arXiv:2411.00508*, 2024. 1, 2, 3, 4, 5, 7

[12] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[13] Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025. 1, 2, 3, 4, 5, 7

[14] Seungjae Lee, Yibin Wang, Haritheja Etukuru, H Jin Kim, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Behavior generation with latent actions. *arXiv preprint arXiv:2403.03181*, 2024. 2

[15] Xinghang Li, Minghuan Liu, Hanbo Zhang, Cunjun Yu, Jie Xu, Hongtao Wu, Chilam Cheang, Ya Jing, Weinan Zhang, Huaping Liu, et al. Vision-language foundation models as effective robot imitators. *arXiv preprint arXiv:2311.01378*, 2023. 1, 2

[16] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023. 2, 7

[17] Atharva Mete, Haotian Xue, Albert Wilcox, Yongxin Chen, and Animesh Garg. Quest: Self-supervised skill abstractions for learning continuous control. *Advances in Neural Information Processing Systems*, 37:4062–4089, 2024. 2

[18] Earl K Miller and Jonathan D Cohen. An integrative theory of prefrontal cortex function. *Annual review of neuroscience*, 24(1):167–202, 2001. 2

[19] Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903. IEEE, 2024. 1, 2

[20] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017. 2, 4, 5

[21] Yating Wang, Haoyi Zhu, Mingyu Liu, Jiange Yang, Hao-Shu Fang, and Tong He. Vq-vla: Improving vision-language-action models via scaling vector-quantized action tokenizers. *arXiv preprint arXiv:2507.01016*, 2025. 2

[22] Kun Wu, Yichen Zhu, Jinming Li, Junjie Wen, Ning Liu, Zhiyuan Xu, and Jian Tang. Discrete policy: Learning disentangled action space for multi-task robotic manipulation. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8811–8818. IEEE, 2025. 2

[23] Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023. 1, 2, 3, 4, 5, 6, 7

[24] Tony Z Zhao, Jonathan Tompson, Danny Driess, Pete Florence, Kamyar Ghasemipour, Chelsea Finn, and Ayzaan Wahid. Aloha unleashed: A simple recipe for robot dexterity. *arXiv preprint arXiv:2410.13126*, 2024. 1, 2, 3

[25] Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183. PMLR, 2023. 1, 2

[26] Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. St-moe: Designing stable and transferable sparse expert models. *arXiv preprint arXiv:2202.08906*, 2022. 4, 5

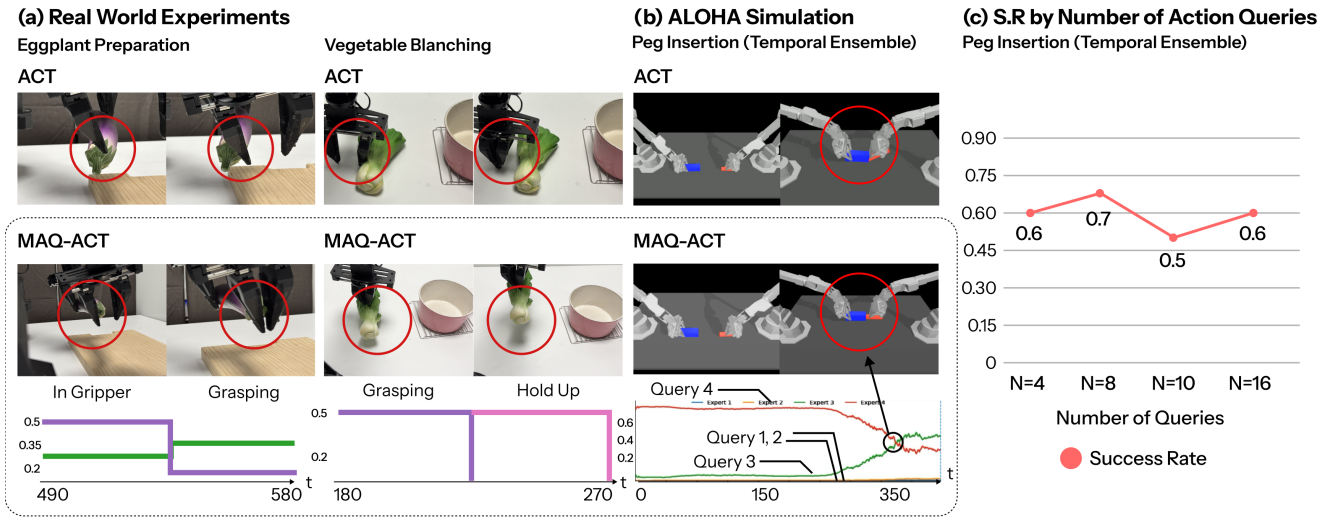

Figure 5. **Comparison of ACT and MAQ-ACT in real world and simulation.** (a) Real world sequences for Vegetable Blanching and Eggplant Preparation. ACT often closes early or without contact. With MAQ-ACT, closure more often aligns with contact. *In Gripper* tends to rise before *Grasping*, and *Hold Up* follows. (b) ALOHA simulation for Peg Insertion. Frames show approach and insertion, and the plot on the right shows router probabilities with a temporal ensemble. A query specialized for insertion becomes the highest weight near contact and remains active through insertion, showing how MAQ-ACT selects primitives that correct ACT failures.

A. Appendix

A.1. Failure Case Analysis

Figure 5 compares ACT and MAQ-ACT on representative failure cases. ACT often closes before contact, leading to retries and unstable motion. With MAQ-ACT, weights shift from *In Gripper* to *Grasping* only after contact and then to *Hold Up*. MAQ-ACT more often closes at contact rather than at a preset step, and its rollouts are more stable.