


Politecnico di Torino


## Politecnico di Torino

Master's Degree in Computer Engineering

Artificial Intelligence and Data Analytics

Academic Year 2025/2026

Graduation Session September 2026


# Post-Training MoE-fication of a Vision-Language-Action Policy: A Staged Analysis of What Binds Conversion Quality


Supervisors:

Prof. Daniele Jahier Pagliari

Dr. Alessio Burrello

Matteo Risso

Luca Benfenati

Candidate:

Kevser Günaydın

# Abstract

Vision-Language-Action (VLA) policies inherit the parameter counts of the vision-language backbones they are built on, but must run on hardware far weaker than the machines they were trained on. Their feed-forward network (FFN) blocks dominate inference cost. Post-training MoE-fication cuts that cost without discarding fine-tuned weights: it analytically restructures a dense FFN into shared and routed experts, using a small calibration set and no gradient training. This thesis ports CMoE, an analytical FFN-to-MoE method built for text-only language models, to VLA-0, a 3B-parameter policy on a Qwen2.5-VL backbone, and evaluates it on a LIBERO-Goal manipulation task. The question is what determines conversion quality once success is judged by closed-loop task completion rather than next-token perplexity.

Five factors could plausibly determine this: calibration data, expert granularity, router quality, unconditional (always-on) capacity, and conversion depth. A staged pipeline of five diagnostics tests them one at a time, each stage fixing one design decision and stating what it cannot resolve.

Unconditional capacity is the binding factor within a fixed routing scheme. Moving capacity out of the always-on block while holding total activation fixed at 75% drops task success from 14/14 held-out episodes to 10/14; at 25% unconditional capacity it collapses to 2/14. At 75% activation with half the FFN always active, the converted policy matches the dense policy on every held-out episode, without any training. Conversion depth is a second, weaker factor: converting six more layers buys a few extra FLOP savings but costs four episodes of success. Calibration size, expert granularity, and router sophistication do not bind: fidelity is flat across a 47-fold calibration range, and the analytical router performs no better than random selection. The optional fine-tuning stage optimizes its own training objective without changing closed-loop behavior. Whether routing itself helps beyond capacity, though, is a separate, open question: a router-free static selection matches the same capacity trend on its own, yet the routed configuration still outperforms it at matched activation despite less capacity, a gap this thesis cannot attribute to router skill or rule out as sample noise.

Two independently measured noise floors, one from re-drawing the calibration

subset, one from fine-tuning's own non-reproducibility, land in the same resolution range, coarser than the differences this evaluation sets out to measure. Given the single-task, 14-episode dataset used here, this is a limitation of the present scenario, not a general claim that fine-tuning or a larger calibration set would not help; resolving it needs a larger and more varied benchmark.

These results concern one policy's language stack, calibrated on a single task. Extending the same pipeline to the vision tower and to multi-task calibration is the natural next step, and would test how far these findings generalize.

# Acknowledgements

I express my gratitude to my thesis advisors, Prof. Daniele Jahier Pagliari and Dr. Alessio Burrello, together with Matteo Risso and Luca Benfenati, for their guidance, close follow-up, and the freedom they gave me over the direction of this thesis. I am especially thankful to Prof. Jahier Pagliari for the effort he put into correcting both my thesis and my defense presentation.

I am grateful to Politecnico di Torino for the opportunity to continue my education and for welcoming me into its academic community, and to EDISU Piemonte for the financial support that let me live and study in Italy with greater peace of mind.

Finally, I thank my mother, Sonay Türkmen, who always supported my decisions and educational goals with a belief in me that I often struggled to find in myself. I continue to draw inspiration from her strength, decisiveness, and constant drive to grow.

Kevser Günaydın

# Table of Contents

**List of Figures** viii

**List of Tables** ix

**1 Introduction** 1
- 1.1 Edge Deployment Pressure for VLA Models . . . 1
- 1.2 Post-Training MoE-fication and CMoE . . . 2
- 1.3 The Central Question: What Binds Conversion Quality in a VLA? . 2
- 1.4 Contributions . . . 3
- 1.5 Thesis Structure . . . 4

**2 Background** 6
- 2.1 Vision-Language-Action Models . . . 6
  - 2.1.1 VLA-0 Architecture and Action-as-Text Decoding . . . 6
  - 2.1.2 LIBERO Benchmark . . . 7
- 2.2 Transformer FFN Architecture in Qwen2.5-VL . . . 8
- 2.3 Mixture-of-Experts Fundamentals . . . 9
- 2.4 CMoE: Analytical FFN-to-MoE Conversion . . . 9
  - 2.4.1 Activation Profiling with ATopK . . . 10
  - 2.4.2 Balanced Expert Construction . . . 10
  - 2.4.3 Analytical Router Construction . . . 11
  - 2.4.4 Fine-Tuning Enhancements . . . 11

**3 Related Work** 12
- 3.1 Post-Training MoE-fication and Alternative Compression . . . 12
- 3.2 VLA Efficiency . . . 13
- 3.3 Positioning . . . 14

**4 Method** 15
- 4.1 Problem Formulation . . . 15
- 4.2 Why a Staged Analysis Pipeline . . . 15

4.3 Stage 1: Activation Profiling . . . 16
4.4 Stage 2: Per-Layer Distortion . . . 17
4.5 Stage 3: Fidelity Over Calibration Size . . . 17
4.6 Stage 4: Routing . . . 18
4.7 Stage 5: Rollout Evaluation . . . 18
4.8 The Enhancement Stage . . . 19
4.9 Measurement Design . . . 19
4.9.1 Metrics and Determinism . . . 19
4.9.2 Noise-Floor Protocol . . . 19
4.9.3 Known Limitations of the Instrumentation . . . 20

**5 Experimental Setup** 22
5.1 Model and Task . . . 22
5.2 Calibration and Evaluation Data . . . 22
5.3 Configuration Space . . . 23
5.4 Implementation and Reproducibility . . . 23

**6 Results** 25
6.1 Layer Convertibility . . . 25
6.2 Per-Layer Distortion and Capacity Ranking . . . 26
6.3 Fidelity Across Calibration Size . . . 27
6.4 Routing . . . 29
6.5 Task Accuracy . . . 31
6.6 Compute and Latency . . . 33
6.6.1 Analytic FLOP Count . . . 33
6.6.2 Measured Wall-Clock Latency . . . 33
6.6.3 Carve Cost . . . 34
6.6.4 The Depth Trade-Off . . . 34
6.7 Synthesis . . . 34

**7 Results II: The Enhancement Stage** 36
7.1 What Is Being Tested, and How . . . 36
7.2 Findings Independent of the Noise Floor . . . 38
7.2.1 The Training Signal Is Weak . . . 38
7.2.2 Across Scope: the Objective Moves, Behavior Doesn't . . . 39
7.2.3 Cost . . . 39
7.3 The Training-Noise Floor . . . 40
7.4 Success-Rate Results: Reported, Not Concluded . . . 40
7.5 Why the Recipe Does Not Transfer . . . 42

**8 Discussion** 43
8.1 The Shared Expert as an Unconditional Capacity Floor . . . . . 43
8.2 Why the Router Performs at Chance . . . . . 43
8.3 What Transfers From CMoE, and What Does Not . . . . . 44
8.4 Measurement Resolution in Few-Episode Robot Evaluation . . . . . 45
8.5 Validity Boundaries . . . . . 45

**9 Future Work** 46
9.1 Statistical Power for Online Comparison . . . . . 46
9.2 Beyond a Single Task and Dataset . . . . . 46
9.3 Per-Layer Distortion and Capacity . . . . . 47
9.4 Token-Position Separation . . . . . 47
9.5 Vision Tower Conversion . . . . . 47
9.6 Closing the Runtime-Measurement Gaps . . . . . 47
9.7 Action-Chunk Ensembling . . . . . 48

**10 Conclusion** 49

**A Configuration Evidence Trail** 50

**B Full Per-Layer Tables** 52

**C Porting Activation Profiling to a Vision-Language Backbone** 55

**D Reproduction** 56

**Bibliography** 58

# List of Figures


2.1 **VLA-0's input/output structure.** Figure taken from [2]. . . . . 7
2.2 **A LIBERO-Goal episode.** Initial simulated scene for the task this thesis evaluates on (Chapter 5), paired with its instruction: *"put the wine bottle on top of the cabinet."* . . . . . 8

6.1 **Layer convertibility across the language stack** ($k_{act} = 50$, 374 observations / 78,914 prefill tokens). **(a)** Fraction of each layer's 11,008 FFN neurons that enter the top-$K$ at least once (the effective pool). **(b)** 95th-percentile per-neuron activation rate. . . . . 26
6.2 **Per-layer reconstruction error** (E8S4A2, $k_{act} = 50$, 374 observations; dense hidden states propagated between layers so each layer is measured in isolation). **(a)** Relative $L_2$. **(b)** Cosine distance. . . 27
6.3 **Fidelity is flat across a 47× range of calibration size**, across three sweeps varying depth and subset strategy. . . . . 29
6.4 **The analytical router performs at chance.** . . . . . 30
6.5 **The online noise floor.** . . . . . 32

7.1 **Per-episode outcomes, E8S4A2/13–28.** Rows are arms, columns are the 14 evaluation init states. . . . . 41
7.2 **Per-episode outcomes, E8S2A2/13–22, three seeds. ▲/▼** mark differences from the training-free baseline (2/14); at this success rate they reflect noise, not a real effect. . . . . 42

# List of Tables


5.1 **Configurations evaluated end-to-end on the wine-bottle task's 14 held-out states**, spanning the capacity, depth, and calibration-size axes. . . . . 24

5.2 **Static-selection control: rate-ranked pruning at $N_k = 0$, layers 13–22, $N$=132 phase-stratified.** Each row is matched to a routed configuration at the same activation %. . . . . 24

6.1 **Selecting $k_{act}$: worst-case degenerate/carve separation across candidate values, on a 20-observation pool.** . . . . . 26

6.2 **Unconditional capacity and expert granularity against reconstruction error and task success.** . . . . . 28

6.3 **Offline noise floor: metric movement from re-drawing the same-size calibration subset against sweeping calibration size.** 28

6.4 **Range compression by degree of freedom.** . . . . . 29

6.5 **Coverage and task success for the static-selection control** (layers 13–22, $N$=132 phase-stratified). . . . . 31

6.6 **Configuration sweep on the held-out initial states.** Paired step differences are computed only over states where both the configuration and the dense policy succeed. . . . . 31

6.7 **Measured latency against the analytic FLOP saving.** All figures are percent change relative to dense. Corrected disables the synchronization defect; not yet re-measured for the routed rows (n/a). 34

7.1 **Fine-tuning enhancement arms for E8S4A2 (layers 13–28) and the capacity-floor configuration E8S2A2 (layers 13–22): loss type, LoRA scope, and seed per arm.** . . . . . 37

7.2 **Training configuration, held constant across every fine-tuning arm.** . . . . . 38

7.3 **Task success and mean episode length per enhancement arm, evaluated against the training-free baseline, for both fine-tuned configurations.** Episode length is averaged over successful episodes only. . . . . 41

A.1 **Configuration evidence trail.** . . . . . 51

D.1 **Result-to-script reproduction map.** . . . . . 57

# Chapter 1

# Introduction

## 1.1 Edge Deployment Pressure for VLA Models

Vision-Language-Action (VLA) policies inherit the parameter counts of the vision-language backbones they are built on, while being deployed under hard latency budgets: a manipulation policy must emit an action chunk fast enough to close the control loop on hardware that is not a datacentre GPU. Autoregressive VLA inference commonly runs an order of magnitude below the control rate closed-loop manipulation needs, and a multi-billion-parameter backbone's memory footprint routinely exceeds what edge hardware provides [1]. VLA-0 itself, the policy this thesis converts, reports 4 Hz on standard PyTorch with no inference-time optimisation [2], the number this thesis's conversion targets. This latency gap makes inference-time efficiency directly relevant to deploying VLAs on robotic platforms, where the control loop sets the requirement.

Within a transformer block [3], inference cost is not spread evenly across sublayers. Each layer's feed-forward network (FFN) projects the hidden state up into a much wider intermediate dimension before projecting it back down, so its parameter and FLOP cost scales with that intermediate width. Attention's projections, by contrast, scale with the hidden dimension alone, while the rest of its cost scales with sequence length, not model width. In the Qwen2.5-VL-3B backbone this thesis converts, the intermediate width is 11,008 against a hidden dimension of 2,048, a 5.4× expansion, and the FFN accounts for 87.8% of all linear-layer FLOPs in a dense forward pass (Section 6.6.1). The FFN is therefore where a compression method has to act to move inference cost, and it is the target of the conversion this thesis studies.

## 1.2 Post-Training MoE-fication and CMoE

Mixture-of-Experts (MoE) architectures decouple model capacity from per-token computation by activating only a subset of parameters. Conventionally, obtaining an MoE requires pre-training from scratch or continual training on billions of tokens. *Post-training* MoE-fication instead restructures an existing dense FFN into shared and routed experts analytically, from a small calibration set and without gradient training.

Post-training MoE-fication is one of three ways to cut that FFN cost by touching fewer weights per token, and it is worth placing it against the other two. Pruning removes weights outright, driving individual connections to zero at the sparsity level a calibration set can tolerate [4]; because those weights are gone, the model has to compensate for their absence, so pruned models typically need a recovery stage to regain the accuracy the removed weights carried. Activation-sparsity methods instead exploit the fact that only a subset of neurons fire for any given input and skip the inactive ones at inference time, without altering a single weight [5]; this cuts compute but not the model's stored parameter count. Post-training MoE-fication sits between the two: like pruning, it restructures the model so fewer parameters are touched per token; like activation sparsity, it leaves every trained weight intact, only reorganizing which ones a given token uses. Quantization, which lowers the numeric precision of the stored weights rather than which weights are used [6], is orthogonal to all three and composes with any of them, including MoE-fication itself: a model can be MoE-fied and then quantized.

This distinction makes post-training MoE-fication the appropriate family for a VLA, whose weights are the product of expensive task-specific fine-tuning that should not be discarded. VLA pruning pipelines confirm the cost: reinforcement-learning recovery is needed just to reach the original success rate [7], which is why a weight-preserving method is worth testing as a separate approach.

Within this family, this thesis ports CMoE [8], an analytical conversion method whose construction rests on an empirical premise: FFN neuron activation frequencies are *bimodal*, with a subset of neurons active across nearly all inputs and a majority active only for specific inputs. The first group becomes an always-active shared expert; the second clusters by co-activation into conditionally routed experts, with a router derived analytically from representative neurons.

## 1.3 The Central Question: What Binds Conversion Quality in a VLA?

This thesis studies one specific setting: a 3B-parameter vision-language-action policy, one LIBERO-Goal manipulation task, and the language stack only, since the

vision tower is not converted. Conclusions are stated within that scope throughout; Chapter 8 states the full validity boundaries explicitly.

Porting CMoE to this VLA exposes five candidate constraints, and it is not obvious in advance which one will eventually be the limiting factor for the final quality of the converted model. Conversion quality could be limited by the *calibration data* (a robot policy sees narrow, repetitive demonstrations rather than diverse text), by *expert granularity* (how many routed experts the FFN's neurons are split into, and therefore how coarse or fine each expert's specialization can be), by the quality of the *analytical router* (the training-free mechanism that decides, per token, which experts to activate), by the *unconditional capacity* left always-on (the fraction of the FFN that runs on every token regardless of routing), or by the *depth* of the conversion (how many of the backbone's layers are converted from dense to MoE). These are not equivalent: they imply different remedies, different costs, and different claims about why the method works when it works.

This thesis answers the question empirically by constructing a staged analysis pipeline in which each stage fixes one design decision and is explicitly bounded by what it *cannot* decide. Within this scope, the conclusion is that unconditional capacity binds within a fixed routing scheme, while calibration size and strategy, expert granularity, and router sophistication do not. Granularity shows a smaller, secondary effect on task success that this thesis reports but does not fully resolve. Whether routing itself adds anything beyond capacity is a separate question; this thesis's evidence leaves it open rather than settling it.

## 1.4 Contributions

Below is a summary of the contributions of this thesis:

1. **The unconditional-capacity floor.** The always-on fraction of the FFN, not the total per-token activation ratio, determines whether a converted VLA policy works: holding the activation ratio fixed at 75% and moving capacity from the shared block into the routed block drops task success from 14/14 to 10/14, while holding the shared block fixed and varying the routed fraction changes nothing. The source method selects the shared block's neurons by activation rate because it expects them to form a distinct population that encodes "common knowledge" [8]. That population is largely absent in this model (Section 6.1), and simply discarding the remaining neurons instead of routing among them costs almost nothing (Section 6.4). So the shared block looks like a fixed capacity guarantee, not a store of specially informative neurons. Every configuration in this thesis still selects that block the same way CMoE does, by activation rate, so whether a differently chosen block of the same size would work just as well remains untested.

2. **A staged analysis pipeline with stated decision boundaries.** Five diagnostics: activation profiling (which layers have enough activation diversity to convert at all), per-layer distortion (how far a layer's output moves once converted), calibration-size fidelity (how much calibration data is enough), routing coverage (whether the router selects the experts a token actually needs), and rollout evaluation (whether the task succeeds end to end). Each fixes one hyperparameter and states explicitly what it cannot determine. The first two disagree with each other: the layers activation profiling excludes are exactly the layers per-layer distortion finds easiest to reconstruct. Neither diagnostic is wrong; they measure different properties of a layer, so the pipeline needs both rather than treating one as a stand-in for the other.

3. **A measured resolution limit for few-episode robot evaluation.** A noise floor obtained by perturbing only what should not matter (re-drawing the calibration subset at fixed size), and a second, independent floor from training non-reproducibility, both larger than the effects this thesis sets out to measure. Given the dataset used here (14 held-out episodes from a single task) and the specific conversion method tested, this is a limitation of the present scenario rather than a general property of VLAs: it means this evaluation cannot determine whether a larger or more varied calibration set, or fine-tuning, would help, not that they would not.

4. **An evaluation of CMoE's router and enhancement stage in a VLA.** The analytical router performs at chance against random expert selection, with a bounded oracle ceiling; the lightweight fine-tuning stage optimizes its own objective successfully while leaving closed-loop behavior unchanged.

5. **An analytic efficiency gain not yet realized in wall-clock latency.** The operating configuration reduces FFN FLOPs by 6.94% analytically, rising to 11.10% at the deeper conversion, but the current implementation does not realize either saving: measured latency is at or above dense across every configuration. The gap is attributed to a single, localized dispatch-path inefficiency rather than the conversion itself (Section 6.6.2), and closing it is left as future work (Section 9.6).

## 1.5 Thesis Structure

Chapter 2 discusses VLA architectures, MoE fundamentals, and the CMoE algorithm. Chapter 3 surveys related work. Chapter 4 presents the analysis pipeline and measurement design. Chapter 5 describes the experimental setup. Chapter 6 presents results in pipeline order. Chapter 7 evaluates the fine-tuning enhancement

stage. Chapter 8 discusses the findings and their boundaries. Chapter 9 outlines future directions, and Chapter 10 concludes.

# Chapter 2

# Background

## 2.1 Vision-Language-Action Models

VLA models combine a pretrained vision-language backbone with an action prediction mechanism, and the design choices that distinguish VLA families [1] follow from how this mechanism is attached. Text-token autoregressive generation (VLA-0 [2], RT-2 [9]) reuses the backbone's existing next-token machinery with no architectural addition, at the cost of the precision a fixed discretization bin size imposes and the latency of one forward pass per token. Diffusion and flow-matching action heads ($\pi_0$ [10], SmolVLA [11]) instead attach a separate generative module trained to output continuous action chunks directly, trading that per-token cost for extra parameters and a second training objective. Latent-action approaches [12] compress the action space before decoding it, at the cost of a more complex training pipeline. VLA-0's choice of the first family makes its FFN blocks the direct analog of a language model's, so the digit-token corruption and gripper-extremity failure modes below have no counterpart in the other two families.

### 2.1.1 VLA-0 Architecture and Action-as-Text Decoding

VLA-0 [2] is the model studied throughout. It is built on Qwen2.5-VL-3B-Instruct [13], a vision-language model that pairs a Vision Transformer image encoder with a language-model decoder, the image encoder's output features projected into the language model's embedding space so that image and text tokens are processed by the same transformer stack. VLA-0 makes no change to this backbone at all: it takes the same three inputs any such vision-language model would (a system prompt stating the task format, one or more camera images, and a natural-language instruction) and produces the same kind of output any such model would, plain text, as shown in Figure 2.1.

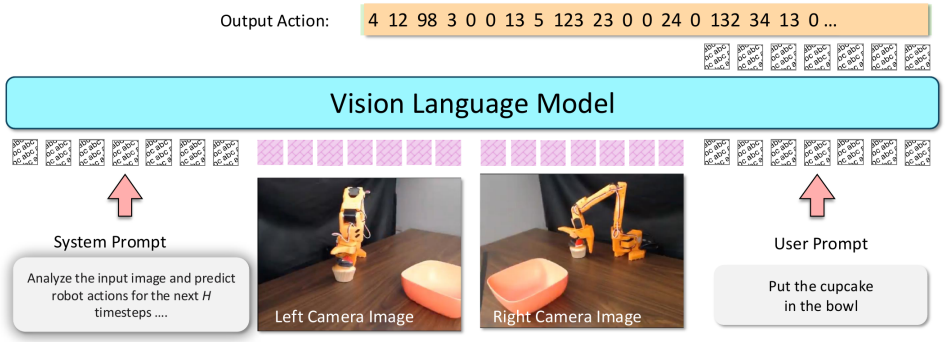



**Figure 2.1: VLA-0's input/output structure.** Figure taken from [2].

Concretely, that output text is a single flat sequence of space-separated integers. For each of the next $H$ timesteps, the model emits one integer for each of $D$ action dimensions, each obtained by normalizing the original continuous action value to a fixed integer range (for example 0 to 999) and rounding to the nearest bin. An *action chunk* is this full $H \times D$-integer sequence, decoded autoregressively one digit token at a time, exactly like any other language-model output; this thesis's setup uses $H = 8$ and $D = 7$ (six continuous end-effector degrees of freedom plus the gripper). Two consequences of this action-as-text scheme matter for this thesis. First, an action chunk is a sequence of digit tokens, so a single wrong bin index corrupts the remainder of the chunk. This failure mode has no analog in the source method's own evaluation: the source method, CMoE [8], is scored on multiple-choice benchmarks such as PIQA and ARC-Challenge, where the answer is decided by comparing the likelihood of a handful of candidate completions rather than by generating a multi-token sequence a single early error can corrupt. Second, the gripper is a binary decision, open or closed, encoded at the extreme of the bin range rather than as a discrete token of its own, so it decodes through the same continuous-valued digit sequence as the six analog degrees of freedom; the converted and dense policies disagree on which side of that decision they land on 4.0–4.9% of steps, a gap that does not shrink with more calibration data (Section 6.3).

### 2.1.2 LIBERO Benchmark

LIBERO [14] provides the manipulation benchmark: a set of task suites for evaluating knowledge transfer in robot manipulation, each suite varying a different axis (spatial layout, object identity, or task goal) while holding the rest fixed. LIBERO is simulated, not physical: each task places a robot arm and a handful of household objects on a tabletop and pairs the scene with a natural-language instruction, as in Figure 2.2; an episode succeeds if the arm brings the scene into



**Figure 2.2: A LIBERO-Goal episode.** Initial simulated scene for the task this thesis evaluates on (Chapter 5), paired with its instruction: *"put the wine bottle on top of the cabinet."*

the instructed goal configuration within a fixed number of steps. LIBERO-Goal fixes the objects and their spatial layout and varies the task goal, isolating whether a policy generalizes across instructions rather than across scenes or objects: the axis relevant to a language-conditioned VLA. Each task ships a fixed bank of initial robot and object poses; Chapter 5 states the specific task and the calibration/evaluation split drawn from that bank. Evaluation runs through the RoboVerse harness, which standardizes simulator stepping, observation formatting, and episode termination across the configurations compared in this thesis, so differences in outcomes reflect the policy rather than the evaluation scaffolding.

## 2.2 Transformer FFN Architecture in Qwen2.5-VL

The conversion target is the SwiGLU feed-forward block:

$$F(x) = W_{\text{down}}\, h, \qquad h = \text{Swish}(W_{\text{gate}}^{\top} x) \odot (W_{\text{up}}^{\top} x),$$

where $h \in \mathbb{R}^{d_h}$ is the intermediate activation vector, $h_i$ the scalar activation of intermediate neuron $i$ within it, and $\text{Swish}(x) = x \cdot \sigma(x)$ ($\sigma$ the logistic sigmoid) the smooth, non-monotonic gating nonlinearity Qwen2.5-VL's FFN uses [13]. With intermediate width $d_h = 11{,}008$ across 36 language layers, $W_{\text{gate}}$ and $W_{\text{up}}$ each project the hidden state into the $d_h$-dimensional intermediate space; their outputs are combined element-wise and projected back down by $W_{\text{down}}$. A neuron $i$ in this block is one column of $W_{\text{gate}}$ and $W_{\text{up}}$ together with the corresponding row of $W_{\text{down}}$: the unit that MoE-fication reassigns to shared or routed experts. Conversion

therefore operates on the intermediate dimension $d_h$, not the hidden dimension $d$: the object being partitioned is which of the $d_h$ intermediate neurons fire for a given token, and expert membership is a partition of that $d_h$-sized set, independent of the token representation's own width.

## 2.3 Mixture-of-Experts Fundamentals

A routed MoE layer replaces a single dense FFN with $z$ smaller expert FFNs and a router that scores each expert per token; only the top-$k$ scoring experts run, and their outputs are combined, so compute per token scales with $k$ rather than $z$ [15]. A router can collapse onto a small subset of experts, so MoE layers include a load-balancing mechanism (auxiliary losses in the original formulation, an adaptive bias in the enhancement stage described below).

Experts come in two kinds. A *shared* expert runs on every token regardless of routing; a *routed* expert runs only when the router selects it. DeepSeekMoE, which introduced the distinction, motivates the shared expert as a repository for knowledge common across inputs, freeing routed experts to specialize on what varies [16]. In a post-training conversion, both types are carved from the same dense FFN rather than added: a partition of its $d_h$ intermediate neurons assigns each one to the shared block or to exactly one routed expert. So the converted model keeps the same total parameter count as the dense one, redistributed rather than augmented.

Throughout this thesis, configurations are written in E$z$S$x$A$y$ notation: a model has $z$ total experts, of which $x$ are shared (always active) and $y$ additional routed experts are activated per token. E8S4A2, for example, has 8 total experts, 4 shared and 2 routed-active, so 6 of the 8 run on every token.

Shared and routed experts define two further quantities: the *activation ratio* and the *unconditional capacity*. The activation ratio is the fraction of experts active per token, $(N_s + N_k)/z$, where $N_s = x$ and $N_k = y$ in the notation above. The unconditional capacity is the fraction active for every token regardless of routing, $N_s/z$. From here on, "capacity" alone, without a qualifier, refers to this quantity. Two configurations can share an activation ratio while differing in unconditional capacity: E8S4A2 and E8S3A3 are both 75% active but hold 50% and 37.5% unconditionally. Chapter 6 shows that they behave differently despite the shared activation ratio.

## 2.4 CMoE: Analytical FFN-to-MoE Conversion

CMoE [8] is the post-training MoE-fication technique this thesis starts from. The source paper applies it to text-only language models; this thesis instead applies it

to a VLA, testing whether the same construction still works once the target is a closed-loop robot policy. The rest of this section describes CMoE's construction, in the stages this thesis reuses: activation profiling, expert construction, router construction, and the optional fine-tuning stage.

### 2.4.1 Activation Profiling with ATopK

CMoE uses a small set of *calibration tokens*, drawn from a handful of forward passes over held-out inputs and requiring no training, to decide how to partition each dense FFN layer's neurons into the shared and routed experts of the corresponding MoE layer (Chapter 5 states the specific calibration data this thesis uses). For each calibration token, the $K_a$ intermediate neurons with the largest activation magnitude $|h_i|$ are marked active (absolute top-$K$, ATopK), producing a binary activation matrix whose rows are tokens and whose columns are per-neuron activation patterns. A neuron's activation rate is its column mean: the fraction of calibration tokens for which it was among the $K_a$ most active. $K_a$ is the profiling stage's single free parameter: it sets how many neurons compete for the label "active" per token, and Section 6.1 reports how the layer-convertibility measurement depends on its value.

### 2.4.2 Balanced Expert Construction

The neurons with the highest activation rates form the shared expert. The remaining neurons are grouped into $N_r$ equal-sized routed experts by clustering: each neuron's activation pattern across the calibration tokens is a binary vector, one entry per token, and CMoE measures how similar two neurons are using the $L_2$ distance between their vectors. Because the vectors are binary, this $L_2$ distance simply counts how many tokens the two neurons disagree on, which is their Hamming distance; so two neurons that tend to fire on the same tokens end up close together, and two that rarely fire together end up far apart. Clustering then groups neurons into equal-sized clusters, minimizing the total distance within each cluster. CMoE solves this with the Jonker–Volgenant algorithm [17], which guarantees the best possible grouping given that constraint. That guarantee is bounded by the constraint itself: the algorithm finds the best grouping under the rule that every cluster must be the same size, not the best grouping if cluster sizes were left free. Equal-sized clusters are a design choice CMoE makes, not something the activation data itself requires. This refines the clustering step of the FFN-to-MoE decomposition that the seminal MoEfication paper [18] introduced, replacing its criterion with one derived from activation co-occurrence rather than weight similarity.

### 2.4.3 Analytical Router Construction

Each routed expert's cluster has one *representative neuron*: the member closest to the cluster's centroid in the activation space. The router reuses that neuron's own gate and up-projection columns (extracted directly from the original dense weights, not learned) to compute a score for the expert exactly as the SwiGLU block would compute that one neuron's contribution: $G(x) = \text{Swish}({W^R_{\text{gate}}}^\top x) \odot ({W^R_{\text{up}}}^\top x)$, where $W^R_{\text{gate}}, W^R_{\text{up}}$ holds only the representative columns. The router's construction rests on one assumption: that a single representative neuron's activation magnitude is a good enough stand-in for its whole cluster's contribution to rank experts by. This only holds if the clustering step actually grouped together neurons that behave similarly, which is the same sparsity and bimodality hypothesis Section 6.1 tests directly. Because of this, the router's quality cannot be judged on its own: it depends on whether the clustering step that feeds it produced good clusters in the first place.

### 2.4.4 Fine-Tuning Enhancements

The analytically constructed router's gates start binary: an expert is fully on or fully off. The enhancement stage relaxes this with one learnable scale $u_i$ per routed expert, initialized to zero, so that a selected expert's gate becomes $g_i = 1 + s'_i u_i$ with $s'_i$ its softmax-normalized router score; at initialization this recovers the analytical construction exactly, and training only moves gates away from it where the objective rewards doing so. Load balancing is adaptive rather than an auxiliary loss term [19]: a per-expert bias is added to router scores before top-$k$ selection, nudged down by a small fixed step when an expert is over-selected relative to its target load and up when it is under-selected, which shapes routing without adding a competing training signal. The source method fine-tunes both mechanisms with LoRA adapters over the backbone [20] on the calibration text itself. Chapter 7 evaluates this enhancement stage against a policy whose success metric is closed-loop rather than the single-step objective of the source method.

The next chapter surveys prior work against this background and positions the thesis within it.

# Chapter 3

# Related Work

## 3.1 Post-Training MoE-fication and Alternative Compression

Methods that convert a trained dense language model into an MoE without full retraining fall into two groups, depending on where the partition comes from: the model's weights, or its observed activations. MoEfication [18] was the first to convert FFNs into MoE layers this way, clustering neurons by weight similarity; CMoE [8] refines that same clustering step onto activation statistics instead of weights. MoEBERT [21] takes a third criterion, importance-guided rather than weight- or activation-based, and evaluates on encoder models rather than the decoder models this family otherwise targets. G-MoEfication [22] extends MoEfication to non-ReLU models, where the activation sparsity the original method relies on is no longer available. ExpertWeaver [23] and MLPMoE [24] are the two methods closest to CMoE within this activation-pattern branch: both convert dense GLU-based MLPs into experts without any training; MLPMoE remains an unreviewed preprint at the time of writing. ToMoE [25] takes a fourth route, structural pruning, to uncover latent experts already present in a dense model instead of clustering activations or weights directly. LLaMA-MoE and its successor [26, 27] take the opposite extreme on the retraining axis: after the initial split, both continue pretraining the whole model on billions of tokens. That full-retraining step is exactly what CMoE and this thesis both avoid. READ-ME [28] addresses a complementary problem: how to co-design the serving system around a router that is decoupled from its experts, rather than how to construct the partition itself. Constructing the partition is this thesis's focus.

The same idea extends beyond language: Berisha et al. [29] cluster a pretrained Vision Transformer's MLP activations via density-based clustering to extract expert subnetworks post-training, recovering 98% of the original ImageNet-1k [30] accuracy

after minimal fine-tuning while cutting MACs by up to 36%. Their target is a classification head; the FFN this thesis converts instead sits inside a vision-language backbone whose output feeds an autoregressive action decoder, not a class label.

A different route to the same FFN cost skips expert structure altogether: pruning [4] removes weights statically, and activation-sparsity methods [5] skip inactive neurons at inference without restructuring the model at all. Both complement expert-level restructuring under the capacity-floor mechanism this thesis measures, rather than competing with it. Chapter 8 returns to this point: the always-on capacity result raises the question of what the routed experts contribute over a static selection.

## 3.2 VLA Efficiency

Efficiency work specific to vision-language-action policies spans three axes distinct from MoE-fication, each trading a different part of the pipeline.

TinyVLA [31] freezes a compact multimodal backbone (422M–1.3B parameters, well under OpenVLA's 7B) and attaches a diffusion policy head via LoRA [20], updating only around 5% of parameters; this avoids the OXE-scale [32] pretraining that OpenVLA [33] requires and its largest variant reports a 21.5% success-rate gain over Diffusion Policy [34] on MetaWorld [35] and a 25.7% gain over OpenVLA on real-world tasks. SmolVLA [11] instead skips the last layers of a compact VLM backbone at inference, aggressively reduces the visual token count, and attaches a generative action-prediction head, training on under 30,000 community-collected episodes. Its asynchronous inference stack decouples action-chunk prediction from execution. It is reported to roughly double task throughput without touching model latency: an orthogonal approach to MoE-fication, since the two reduce different halves of the control loop's wall-clock time rather than competing. Both redesign the architecture and retrain; this thesis converts an existing architecture and does not retrain it.

RLRC [7] takes the compression-then-recovery route directly: structured pruning of an OpenVLA-7B backbone followed by supervised fine-tuning and reinforcement-learning recovery, reporting up to 8× memory reduction and 2.3× inference speedup while matching or exceeding the original dense success rate. That recovery step matters for how this thesis is positioned: pruning needs a training stage to reach dense-level success, which is part of why a method that preserves every trained weight rather than removing them is worth testing as a separate approach from pruning.

AdaMoE [36] is the most conceptually similar prior work, and the sharpest point of contrast. It introduces MoE structure into a VLA's action-relevant layers with a decoupled router (an independent scale adapter that separates which expert is

selected from how much it contributes, avoiding winner-takes-all routing), reporting gains of 1.8% on LIBERO, 9.3% on RoboTwin [37], and 21.5% on real-world tasks over its dense baseline. Its experts are not a partition of the original FFN: AdaMoE copies the whole dense FFN once per expert, so parameter count grows before fine-tuning differentiates the copies. It inherits the dense VLA's pretrained weights rather than training from scratch, but still needs full fine-tuning to make those copies diverge and to train its new router and scale adapter. This thesis instead derives experts analytically from an already-trained dense model's activation statistics, splitting its existing neurons rather than copying them, and asks whether that training step can be skipped entirely.

None of these four lines of work performs post-training MoE conversion of an existing dense VLA backbone: TinyVLA and SmolVLA change the architecture and retrain it, RLRC removes parameters and recovers with reinforcement learning, and AdaMoE builds a MoE structure and trains it. A recent survey [1] situates all of the above within the broader VLA landscape but does not identify this gap directly.

## 3.3 Positioning

This thesis is among the first applications of analytical, calibration-based FFN-to-MoE restructuring to a vision-language-action policy: no prior MoE-fication work, vision-focused [29] or otherwise, has been evaluated on a policy whose metric is closed-loop task success rather than next-token perplexity or classification accuracy. It is also among the first to measure the source method's router and enhancement stage against controls rather than against its own ablations. Those controls are a random baseline, which selects experts by chance, and an oracle ceiling, which shows the best any router could possibly achieve.

The next chapter presents the analysis pipeline and measurement design this positioning motivates.

# Chapter 4

# Method

## 4.1 Problem Formulation

CMoE defines conversion as the reconstruction of the dense model's function:

$$\min \ \mathbb{E}_x \left[ \| F_{\text{MoE}}(x) - F(x) \|^2 \right].$$

In the source method's own words, "given the original FFN output $F(x)$, we seek an MoE architecture that minimizes reconstruction error" [8]: ground truth appears nowhere in that objective. This thesis therefore evaluates *paired fidelity to the dense model* rather than against demonstrations, matching CMoE's own objective and the standard practice in post-training compression [4, 6]. The practical goal is the same: reproduce dense behavior in a comparable number of steps to success.

## 4.2 Why a Staged Analysis Pipeline

A VLA has two properties that the source setting does not have to handle. First, actions are decoded autoregressively as digit tokens, so one wrong bin index corrupts the remaining chunk (Section 2.1.1). Second, reconstruction errors compound layer to layer during closed-loop inference; a diagnostic that feeds each layer a clean dense input cannot see this by construction [38]. No single summary statistic can therefore explain why a configuration succeeded or failed.

The pipeline uses five diagnostics, each fixing one decision and each answering a question the others structurally cannot. Stage 1, activation profiling, asks whether a layer has enough co-activation diversity to cluster at all, and fixes $k_{act}$ and which layers are hard-excluded; this pre-conversion check no post-conversion tool can see. Stage 2, distortion, asks how far a converted layer's output moves from dense, in magnitude or direction, and fixes the conversion range; profiling never converts, so it cannot measure approximation error. Stage 3, fidelity against calibration

size $N$, asks whether generated behavior converges to dense as calibration grows, and fixes calibration size and subset strategy; distortion is single-pass and never touches decoding. Stage 4, routing, asks whether the experts that fire are the right ones and whether any router could do better, and fixes the expert layout; distortion cannot separate "the partition lacks the neurons" from "the router failed to select them". Stage 5, rollout, asks whether the task actually succeeds, and fixes the final configuration and maximum safe conversion depth; it is the only stage where error compounds across layers, so nothing earlier can substitute for it. The order is cost-driven and logical. Stage 1 is cheapest, forward passes with no weight surgery, and prunes the layer search space before anything more expensive runs. Stage 2 carves each candidate layer, taking minutes of GPU time per configuration. Stage 3 is the most expensive single stage: full autoregressive generation through both dense and converted models at every calibration size, so it runs only once the conversion range is fixed rather than as an exploratory tool. Stage 4 is cheap to iterate and not optional: stage 2 cannot separate a missing partition from a router that fails to select what is there, so a routing-sensitive tool is needed regardless of cost. Stage 5 is slowest: real-environment rollouts, reserved for the one or two configurations that survive stages 1 through 4.

## 4.3 Stage 1: Activation Profiling

Scores are computed in the same space the conversion procedure uses, so profiling observes exactly what the conversion will later see: $h = \mathrm{act}(\hat{x}\hat{W}_{\mathrm{gate}}^{\top}) \odot (\hat{x}\hat{W}_{\mathrm{up}}^{\top})$, with $L_2$-normalised inputs and weights.

A layer is excluded primarily by its *effective pool*: the fraction of neurons that ever enter the top-$K$ at least once across all calibration tokens. A second signal is *tail richness*: the 95th-percentile per-neuron activation rate, which is zero whenever even a layer's most-active neurons barely fire. A layer is judged *degenerate* when it has both a small effective pool and zero tail richness: too few neurons ever fire with any regularity for clustering to have anything to work with. A separate, smaller group of *transition* layers passes both checks but is excluded anyway: their effective pool is large enough, but a single dominant neuron skews the clustering centroids expert construction relies on (Section 2.4.2). Layers that pass every check are the *candidate* layers this pipeline considers for conversion.

The premise under test is contextual activation sparsity: that a small, input-dependent neuron subset reproduces the dense output [5], the same premise the source method's own figures rest on.

$k_{act}$ is selected by *worst-case separation*: the effective pool of the least-covered candidate layer divided by the effective pool of the most-covered degenerate layer, evaluated at $k_{act} \in \{10, 50, 100, 1000\}$, spanning the source method's own value up

to a $k$ large enough to stop discriminating layer types at all. Using this worst pair, rather than averaging the ratio across all layers, matters because a few easy layers could otherwise hide how close the hardest candidate and degenerate layers really are.

Collecting the activations this stage scores requires intercepting each record's hidden state before it enters the first converted layer. Porting that collection to a vision-language backbone required two implementation adaptations beyond the reference text-only implementation, detailed in Appendix C.

## 4.4 Stage 2: Per-Layer Distortion

Each candidate layer is converted and its output compared against the dense output on identical inputs. *Dense* hidden states are propagated between layers, so a layer's error is measured in isolation rather than compounded from upstream. Two metrics are reported because one is insufficient: relative $L_2$ captures magnitude error and cosine distance captures direction error. A layer can have the right magnitude with the wrong direction, a failure that is more dangerous for autoregressive decoding and invisible to magnitude alone.

The scan covers all evaluated layers (5–35), including the degenerate and transition layers stage 1 rejects. These excluded layers provide the baseline for reading the conversion range's distortion. Section 6.2 shows they are informative in their own right.

What this stage cannot decide: expert granularity, the location of the capacity threshold, and whether to re-admit the layers stage 1 excluded as degenerate. Degenerate layers score *best* on this stage's distortion metric precisely because so little of them changes when converted, not because they are good conversion candidates, so stage 1's exclusion still stands regardless of this score.

## 4.5 Stage 3: Fidelity Over Calibration Size

For each calibration size $N$ and subset strategy, the converted model generates full action sequences autoregressively; the result is compared against the dense model's output using mean absolute error (Section 4.9.1).

Two deterministic strategies draw $N$ records from the full pool. Each fills its slots with evenly-spaced records rather than a random draw, so a given $N$ always returns the same records.

*Proportional* allocation groups records by episode and gives each episode a slot count proportional to its length: $n_e \approx N \cdot |E_e|/|E|$. A long episode gets more slots than a short one, but a brief motion phase within an episode gets few slots regardless of how important that phase is.

*Phase-stratified* allocation groups records by motion phase instead. Each record is assigned to one of four phases by gripper state (open or closed) and end-effector height, split at the median height within each gripper state: open and low is approach, closed and low is grasp, closed and high is transport, and open and high is release. $N$ is allocated across these four phases in proportion to their size in the pool, then across episodes within each phase, the same way proportional allocation works. This guarantees every phase is represented regardless of how long it lasts within an episode.

A useful side effect is exploited in Section 4.9.2: at low $N$, the two strategies select near-disjoint records at identical sample size, providing independent same-size draws without adding a seed parameter.

## 4.6 Stage 4: Routing

Per-expert activation rates are examined two ways: aggregated over an episode, and as a per-query time series within it. The aggregate view diagnoses whether routing collapses onto a few experts; the time-series view diagnoses whether routing behavior changes over the course of a manipulation. Both describe how *evenly* experts fire, not whether the ones that fire are the *right* ones.

Coverage of each token's needed neurons is defined in four ways: through the shared block alone, through the router's selection, through a random selection (chance), and through the best possible selection for that token (oracle): whichever experts actually held the neurons that token needed. No real router could reach the oracle at inference, since it does not know in advance what a token will need; the oracle instead marks a ceiling on how much any router could still gain. Gate scaling and the adaptive load-balancing bias (Section 2.4.4) are implemented but switched off for this base construction. The size of the oracle-actual gap predicts how much room Chapter 7's router-directed fine-tuning has to help.

## 4.7 Stage 5: Rollout Evaluation

Closed-loop evaluation runs on held-out initial states, each yielding one paired outcome: a converted-versus-dense comparison of success and steps on that exact state, rather than a single pooled statistic across the evaluation set. These are *online* metrics: each comparison comes from a live rollout rather than a pre-collected record (Section 4.9.3 contrasts this with Stage 3's offline metrics).

## 4.8 The Enhancement Stage

CMoE's analytical construction is training-free; the source method also specifies an optional lightweight fine-tuning stage on top of it, tuning two mechanisms introduced in Section 2.4.4 (a learnable per-expert gain and an adaptive load-balancing bias) via LoRA [20]. LoRA scope is swept from narrow (router only) to broad (full CMoE-scope adaptation), so the enhancement's effect is measured across how much of the network it touches. The pipeline tests both a hard cross-entropy target and a soft KL target against the dense teacher; the hard target is sequence-level distillation [39] and the soft target is word-level distillation [40], and the source method offers no reason to prefer one over the other.

The two configurations tested are the deeper conversion, which matches dense at a shallower depth but falls short once more layers are converted at the same capacity, and the capacity-floor configuration, where capacity itself is too low for fine-tuning to help.

Training-free rollout results are reported in Chapter 6; the enhancement stage's results are reported separately in Chapter 7.

## 4.9 Measurement Design

### 4.9.1 Metrics and Determinism

All metrics are paired (same observation, same horizon step, same degree of freedom) against the dense model. The graded measurement, mean absolute error (MAE) between the converted and dense action values, is placed at the *action* level rather than the token level, because in digit-token space a neighbouring digit and a transposed digit score identically, even though one is nearly correct and the other catastrophic. The token level is kept only as a coarser check: whether the two digits match exactly, with no notion of how close a mismatch is.

Every source of run-to-run variation the platform exposes deterministically is fixed, since the conversion is tie-sensitive enough that kernel non-determinism could change which neurons are selected, confounding any comparison between configurations. Conversion and evaluation are consequently bit-reproducible; training is not. Section 5.4 states the specific settings used to enforce this.

### 4.9.2 Noise-Floor Protocol

Determinism removes one source of variation; measuring the ones that remain, like the training variation just noted, requires a way to tell a real effect from noise. The approach is to perturb only what *should not* matter, and measure how much moves anyway. Re-drawing the calibration subset at fixed size holds every

actual design choice fixed, $N$, the layers converted, everything else, and varies only which specific records are included, so whatever it moves is the measurement's own instability. This yields an offline floor and, when run end-to-end, an online floor, each a minimum detectable effect: the smallest true difference between two configurations that this sample size could reliably tell apart from noise. Training non-determinism (Section 7.3) is measured the same way, repeating an identical training command rather than perturbing the calibration draw. Showing the effect is smaller than the measured floor supports a null result; a large $p$-value alone does not. The floor comes from a standard paired t-test power formula, a tool for setting sample-size resolution rather than for testing any specific pair; the comparisons it bounds use the non-parametric tests described next.

Two questions recur across this design, each answered with a different test. Whether a metric trends with calibration size at all is a monotonic-trend question, answered with Spearman rank correlation. Whether a paired configuration succeeds or fails relative to dense is answered with McNemar's test, the paired test for a binary outcome measured once per system (Section 7.4). Step-to-success differences are reported descriptively rather than tested formally, since the online floor above already shows they sit below what this sample size can reliably resolve.

### 4.9.3 Known Limitations of the Instrumentation

Three further properties constrain interpretation. Per-degree-of-freedom drift is averaged across many observations and cannot isolate a single instance, so it never decides anything on its own. Generation is free-running, so position-wise distributional comparison would require teacher forcing, which is not implemented. Profiling, meanwhile, observes only the prefill token population, never the decode-side action tokens.

A fourth property is specific to Stage 3: its metrics are *offline*, computed from the pre-collected trajectory records (Section 5.2) and pooled across observations with equal weight although they derive from few episodes, so the effective sample size is well below the record count. Stage 5's *online* metrics (Section 4.7), computed from live rollouts and paired per initial state, do not share this problem, so online comparisons are trustworthy in a way the pooled offline ones are not.

The prefill-only limitation is inherited from the source method rather than introduced by this implementation. A VLA's inference has two token populations: prefill (image patches and instruction text) and decode (generated action digits). The calibration procedure this thesis ports captures only the first, because the source method was built for a single forward pass over natural-language text, where no such split exists.

The reconstruction objective in Section 4.1 is an expectation over the calibration distribution. If that distribution structurally excludes half of what inference

actually sees, the expectation being minimized is not obviously the one that governs deployment. Whether this matters is unresolved rather than negative. The partition is optimized on visual and instruction tokens only, but decode-time hidden states are themselves conditioned on the prefill, so decode-relevant neurons may largely overlap with the ones the partition was optimized on. Chapter 6 shows one configuration matching dense; that is evidence the mismatch is not fatal *at that configuration*, not evidence the question is closed in general.

The next chapter fixes the specific model, task, and configuration space to which these methodological choices apply.

# Chapter 5

# Experimental Setup

## 5.1 Model and Task

This thesis converts VLA-0 [2], built on a Qwen2.5-VL-3B-Instruct [13] backbone with 36 language MLP layers of intermediate width 11,008.

All evaluation runs on the LIBERO-Goal [14] task `put_the_wine_bottle_on_top_of_the_cabinet`, using the fine-tuned checkpoint distributed with the source policy. This checkpoint is never retrained for this thesis: every result concerns conversion of an already-trained policy, not a policy trained alongside its conversion.

Actions are decoded greedily at zero temperature in chunks of 8. Ensembling across sampled predictions, part of the source method's own recipe, is disabled throughout. This single-chunk setting has the highest variance, and it is the one under which the noise floors in Chapter 6 and Chapter 7 are measured, rather than an average across several. Section 9.7 returns to what enabling it would change.

## 5.2 Calibration and Evaluation Data

Calibration draws from 33 initial states (374 records); evaluation uses 14 held-out initial states (163 records). Both are drawn from the same task's fixed bank of initial poses that LIBERO-Goal ships (Section 2.1.2). A record pairs one such initial state with the trajectory the dense policy produces from it; this trajectory is used to profile activation statistics, not as a training target.

The two sets are disjoint by construction: calibration and evaluation states are partitioned before either is touched, and the partition is checked programmatically rather than assumed. States 7, 42, and 48 are deliberately excluded from both sets, so every reported evaluation number is measured on states the conversion never observed.

Three measurements depart from the full pools: the routing-coverage measurement (Section 6.4) uses a 24-record evaluation subsample, since coverage is computed per token and does not need the full set to stabilize; the teacher-forcing diagnostics (Section 7.2.1) use a 40-record evaluation subsample; and the $k_{act}$-selection sweep (Section 6.1) uses a 20-observation calibration subsample, to keep its four-way comparison cheap. Every other rollout and fidelity metric uses the complete pools.

## 5.3 Configuration Space

All conversions use eight experts, matching the source method's own choice and dividing the intermediate width exactly, except for a dedicated granularity control that instead varies the total expert count $z$ while holding unconditional and total activation fixed at the operating point's values; $k_{act} = 50$ throughout. The swept axes are the shared-expert count $N_s$ (which sets unconditional capacity), the routed-active count $N_k$, total expert count $z$, conversion depth, calibration size $N$, and subset strategy. In the routed configurations, $N_s \in \{1,2,3,4\}$ and $N_k \in \{2,3\}$; in the router-free static family described below, $N_s \in \{4,5,6\}$; in the granularity control, $z \in \{4,8,16,32\}$ with unconditional capacity $N_s/z = 50\%$ and total activation $(N_s + N_k)/z = 75\%$ held fixed. Depth is either layers 13–22 or 13–28, and $N \in \{8,33,66,132,264,374\}$.

Table 5.1 enumerates the routed configurations evaluated end-to-end. Appendix A maps every one of these values back to the stage that fixed it and the evidence behind it.

A second, static family removes the router entirely: neurons are rate-ranked by activation frequency and the top fraction is kept, with no routing decision at inference ($N_k = 0$). This is not an alternative conversion method under test; it is a control that isolates whether unconditional capacity alone explains task success, independent of any routing algorithm. Table 5.2 enumerates it.

Across both tables, the dense policy is the reference; the conversion arms span the capacity sweep at fixed depth, the depth comparison at fixed capacity, the source method's own shared/routed split, a routed-capacity control holding unconditional capacity fixed while varying routed-active count $N_k$, and a granularity control holding capacity fixed while varying total expert count $z$.

## 5.4 Implementation and Reproducibility

Conversion is made bit-reproducible by fixing the sources of run-to-run variation the platform exposes deterministically: PyTorch's deterministic-algorithms mode,

**Table 5.1: Configurations evaluated end-to-end on the wine-bottle task's 14 held-out states**, spanning the capacity, depth, and calibration-size axes.

| configuration | layers | uncond. % | active % | calibration | role |
|---|---|---|---|---|---|
| dense | n/a | 100 | 100 | n/a | reference |
| E8S1A2 | 13–22 | 12.5 | 37.5 | $N$=132 ps | below-floor control |
| E8S2A2 | 13–22 | 25 | 50 | $N$=132 ps | capacity floor |
| E8S3A2 | 13–22 | 37.5 | 62.5 | $N$=132 ps | capacity sweep |
| E8S3A3 | 13–22 | 37.5 | 75 | $N$=132 ps | source-method split |
| **E8S4A2** | 13–22 | 50 | 75 | $N$=132 ps | **operating point** |
| E4S2A1 | 13–22 | 50 | 75 | $N$=132 ps | granularity control |
| E16S8A4 | 13–22 | 50 | 75 | $N$=132 ps | granularity control |
| E32S16A8 | 13–22 | 50 | 75 | $N$=132 ps | granularity control |
| E8S4A2 | 13–28 | 50 | 75 | $N$=132 ps | depth comparison |
| E8S4A2 | 13–22 | 50 | 75 | $N$=8/374 ps | calibration size |
| E8S4A2 | 13–22 | 50 | 75 | $N$=132 prop. | same-size control |

**Table 5.2: Static-selection control: rate-ranked pruning at $N_k = 0$, layers 13–22, $N$=132 phase-stratified.** Each row is matched to a routed configuration at the same activation %.

| configuration | uncond. % | active % | routed config. |
|---|---|---|---|
| E8S4A0 | 50 | 50 | E8S2A2 |
| E8S5A0 | 62.5 | 62.5 | E8S3A2 |
| E8S6A0 | 75 | 75 | E8S4A2 |

a fixed `CUBLAS_WORKSPACE_CONFIG`, deterministic cuDNN kernels, disabled ensembling (Section 5.1), and `torch.compile` turned off, since a compiled graph is not guaranteed to reproduce an eager one bit-for-bit. The deterministic mode runs with `warn_only`, which lets the few operations with no deterministic CUDA kernel pass through rather than aborting the run. Under these flags, two independent carves of the same configuration from the same calibration pool agree exactly: identical success on every held-out episode and identical per-episode step counts. The training-stage variation reported in Section 7.3 originates in fine-tuning; the calibration-draw variation reported in Chapter 6 is a separate mechanism. Neither arises in carving or evaluation. Appendix D maps each reported result to the run directory and script that produced it.

The next chapter reports the training-free results this setup produces, in the same pipeline order as Chapter 4: each section states the decision its stage fixes, reports the measurement, and states what the measurement does not settle.

# Chapter 6

# Results

## 6.1 Layer Convertibility

*Decision fixed: which layers are convertible, and $k_{act}$.*

Figure 6.1 gives the per-layer profile across all 36 language layers, reproduced in full in Appendix B. The shaded band spanning layers 1–9 contains two distinct populations: *hard-degenerate* layers (1–3, 8, 9), where only 0.9–1.6% of neurons ever fire and $p_{95}$ is exactly zero, and *semi-degenerate* layers (4–7), which reach 3.9–17.5% and are excluded for the same reason at lower severity. Transition layers 10–12 are excluded despite a healthy effective pool because they pair a high active fraction with a single dominant neuron that biases centroid initialization. The remaining candidate layers are labeled in the figure as *carve* (13–28) and *action zone* (29–35); layer 0, despite comparable activity, is excluded from both and left unshaded. Tail richness is exactly zero for every hard-degenerate layer, and it is the sharpest single discriminator between the two groups.

The effective-pool statistic grows with token count by construction, so it is only comparable across runs at matched token counts or via its growth ratio. $k_{act}$ is selected by the same worst-case separation defined in Chapter 4: Table 6.1 shows it peaks at $k_{act} = 50$ and collapses to unity at $k_{act} = 1000$, where the metric no longer distinguishes layer types. A mean-of-zones column is also reported, since it orders candidates differently from the worst-case definition. This sweep uses a smaller 20-observation pool to keep the four-way comparison cheap.

The source method's own premise [8] is that text-model activations separate into two groups by an order of magnitude, a bimodal, contextually-sparse pattern. This same profiling shows a much weaker separation in this VLA, measured at the source method's own $K_a$: only about twice-chance tail concentration. Where bimodality *does* exist in this model, it coincides with the layers that are already excluded as degenerate. This is a negative result about the premise, not about the

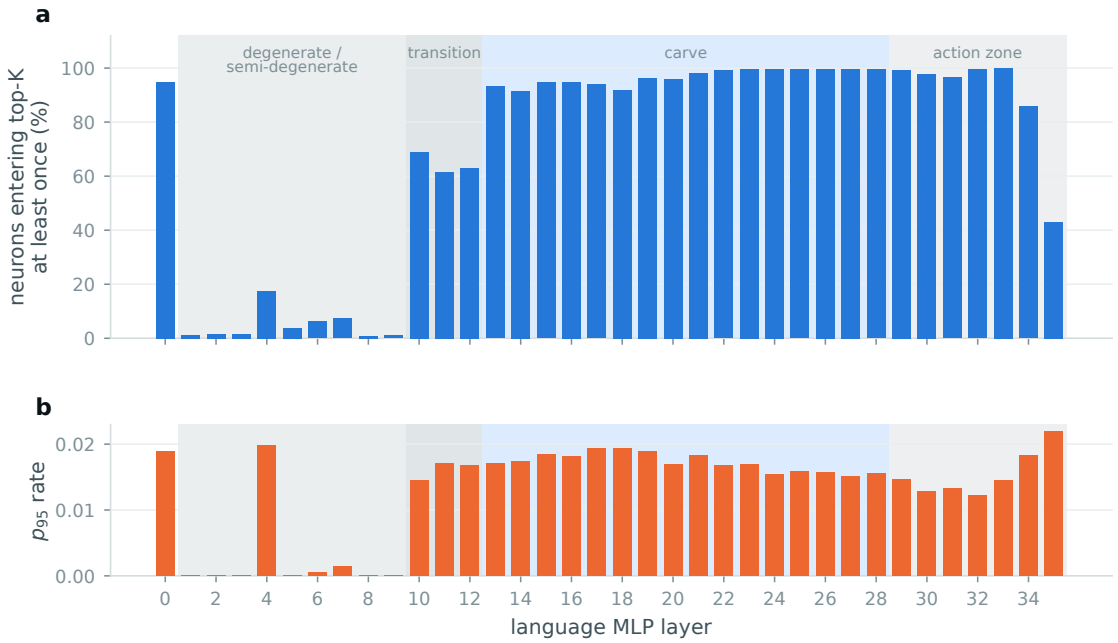



**Figure 6.1: Layer convertibility across the language stack** ($k_{act} = 50$, 374 observations / 78,914 prefill tokens). **(a)** Fraction of each layer's 11,008 FFN neurons that enter the top-$K$ at least once (the effective pool). **(b)** 95th-percentile per-neuron activation rate.

**Table 6.1: Selecting $k_{act}$: worst-case degenerate/carve separation across candidate values, on a 20-observation pool.**

| $k_{act}$ | carve act. % | degen. act. % | separation worst case | separation mean | carve $p_{95}$ |
|---|---|---|---|---|---|
| 10 | 23.4 | 0.34 | 37× | 69× | 0.0026–0.0047 |
| **50** | **70.1** | **1.20** | **42×** | **59×** | **0.0171–0.0196** |
| 100 | 88.9 | 2.08 | 27× | 43× | 0.0320–0.0372 |
| 1000 | 100.0 | 51.89 | 1× | 2× | 0.2106–0.2258 |

method: Section 6.5 shows the conversion works regardless.

## 6.2 Per-Layer Distortion and Capacity Ranking

*Decision fixed: the conversion range. Secondary: a ranking over capacity settings.*

Figure 6.2 reports both metrics across the full layer range. The figure labels layers 13–22 as the *operating range* and 23–28 as *extended*, the two conversion

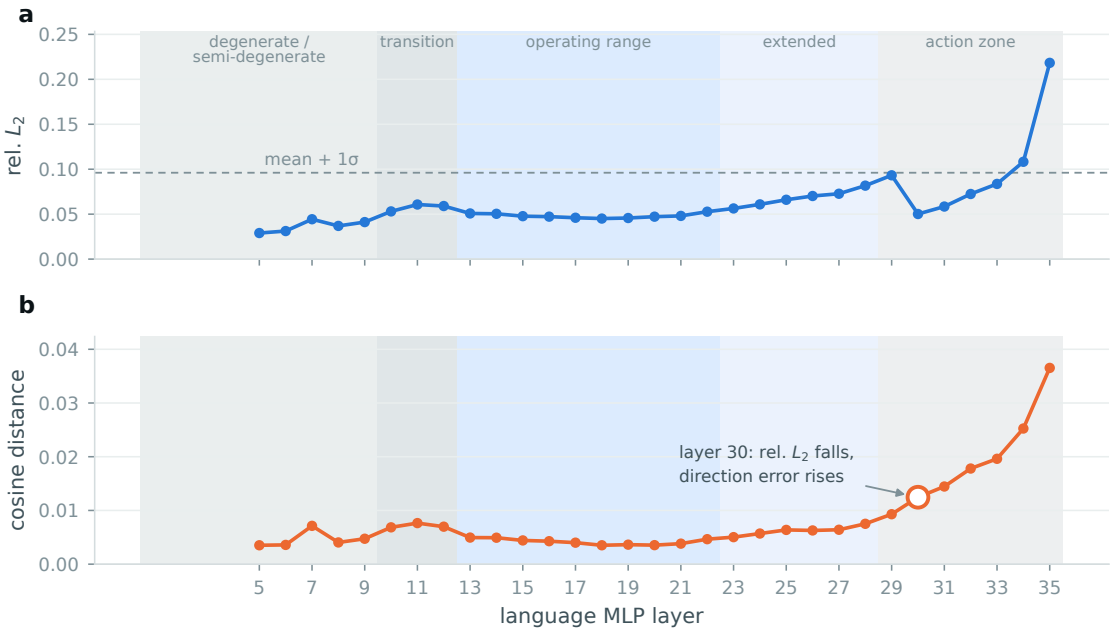



**Figure 6.2: Per-layer reconstruction error** (E8S4A2, $k_{act} = 50$, 374 observations; dense hidden states propagated between layers so each layer is measured in isolation). **(a)** Relative $L_2$. **(b)** Cosine distance.

depths compared later in this thesis (Section 6.6.4).

Panel (a)'s automatic screen, flagging any layer more than one standard deviation above the mean, is a first-pass alert rather than a deliberate cutoff. Panel (b) shows why two metrics are needed: layer 30's relative $L_2$ *falls* to 0.63× its 26–29 neighborhood while its direction error rises to 1.69× the same reference. This is the right magnitude but the wrong direction. The degenerate zone (5–9) has the *lowest* distortion of any zone, confirming that this stage cannot exclude the layers stage 1 rejects (Section 4.2).

Table 6.2 sets the capacity ranking against the task outcome it predicts: reconstruction error declines smoothly, but task success drops sharply at a threshold this stage cannot locate. E8S1A2 confirms this at the floor: distortion-measured at 0.0889, it fails every held-out episode (0/14). The table's granularity control isolates the other axis: holding capacity fixed and varying total expert count $z$ from 4 to 32 leaves reconstruction error flat (0.0481–0.0482); granularity does not affect how closely the conversion approximates the dense model.

## 6.3 Fidelity Across Calibration Size

*Decision fixed: calibration size and subset strategy.*

**Table 6.2: Unconditional capacity and expert granularity against reconstruction error and task success.**

| config | uncond. % | active % | mean rel. $L_2$ | task success |
|---|---|---|---|---|
| E8S1A2 | 12.5 | 37.5 | 0.0889 | 0/14 |
| E8S2A2 | 25 | 50 | 0.0750 | 2/14 |
| E8S3A2 | 37.5 | 62.5 | 0.0606 | 10/14 |
| **E8S4A2** | **50** | **75** | **0.0481** | **14/14** |
| *Granularity control* (fixed: uncond. 50%, active 75%) | | | | |
| E4S2A1 | 50 | 75 | 0.0482 | 10/14 |
| E16S8A4 | 50 | 75 | 0.0481 | 12/14 |
| E32S16A8 | 50 | 75 | 0.0481 | 12/14 |

**Table 6.3: Offline noise floor: metric movement from re-drawing the same-size calibration subset against sweeping calibration size.**

| metric | $\sigma$ same-$N$ (re-drawn subset) | $\sigma$ across $N$ ($N$=8 – 374) | ratio |
|---|---|---|---|
| Action MAE | 0.00178 | 0.00136 | **1.31** |
| Token agreement | 0.00353 | 0.00313 | **1.13** |
| Gripper agreement | 0.00385 | 0.00514 | 0.75 |

Figure 6.3 tracks mean absolute error (MAE, Section 4.9.1) between converted and dense actions. Fidelity is flat across a 47× range of calibration size in three sweeps, each changing one dimension from a base run (layers 13–22, phase-stratified): a depth variant (layers 13–28, same strategy) and a strategy variant (layers 13–22, proportional). $N = 8$ (the calibration size Pei et al. [8] use) is indistinguishable from $N = 374$.

Flatness alone is not enough: it could mean there is no effect, or that the sweep is too weak to detect one. The noise floor in Table 6.3 rules out the second option: re-drawing a different calibration subset of the same size moves the metric more than changing the size does. Rank correlations between MAE and $N$ are weak in all three sweeps (Spearman $p = 0.072$ base, 0.787 strategy variant, 0.329 depth variant). The base run trends in the opposite direction from the other two (positive vs. negative), consistent with a chance fluctuation rather than a real trend.

The vertical offset of the 13–28 sweep is the cost of conversion *depth*, not of calibration size.

Each degree of freedom's error is measured as a fraction of that dimension's own range, since the six joints do not share a scale. This normalized error grows across the horizon, but this thesis cannot tell whether that is because earlier

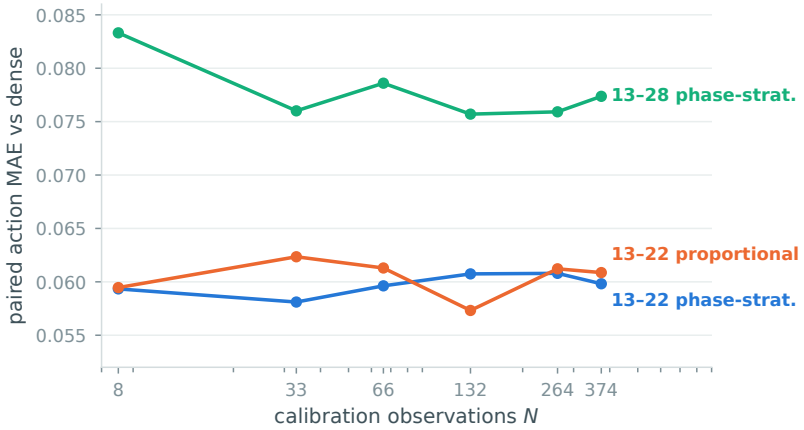



**Figure 6.3: Fidelity is flat across a 47× range of calibration size**, across three sweeps varying depth and subset strategy.

**Table 6.4: Range compression by degree of freedom.**

| sweep | DOF0 | DOF1 | DOF2 | DOF3 | DOF4 | DOF5 | median |
|---|---|---|---|---|---|---|---|
| 13–22 phase-strat. | 0.979 | 0.971 | 0.944 | 0.965 | 0.851 | 0.810 | 0.954 |
| 13–22 proportional | 0.992 | 0.949 | 0.933 | 0.891 | 0.865 | 0.868 | 0.912 |
| 13–28 phase-strat. | 0.921 | 0.870 | 0.900 | 0.984 | 0.774 | **1.011** | 0.911 |

errors compound or later steps are simply harder, since teacher forcing is not implemented (Section 4.9.3). Range compression is the ratio of converted to dense output standard deviation for each degree of freedom, taken as the median across calibration sizes; a value below one means the converted policy under-produces extreme actions. Table 6.4 shows this holds on all six channels at the operating depth (13–22), and on five of six at the deeper conversion (13–28), where one channel slightly exceeds one.

The gripper is excluded because its target distribution is bimodal, open or closed, rather than continuous. It is reported instead by paired sign agreement: whether the converted and dense policies choose the same side of the decision at each step. Converted and dense disagree on the gripper's sign 4.0–4.9% of the time. Like the continuous degrees of freedom, this is flat across calibration size but slightly worse at the deeper conversion (13–28 against 13–22).

## 6.4 Routing

*Decision fixed: expert layout, and whether router-side work can help at all.*

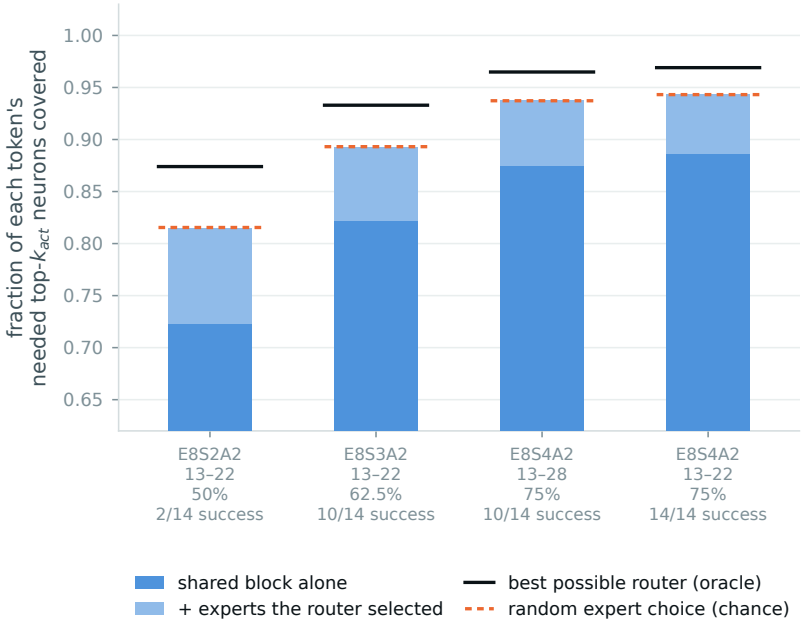



**Figure 6.4: The analytical router performs at chance.**

Figure 6.4 reports coverage for four converted configurations at $k_{act} = 50$: the fraction of every token's needed top-$k_{act}$ neurons that the carve actually delivers, measured on held-out episodes (24 records each). The router's coverage matches random selection in each case (actual − chance between −0.0008 and −0.0001). The oracle bound shows that even a perfect router recovers only a few percentage points (2.6–5.9 pp). The oracle itself falls short of full coverage because it is still limited to selecting $N_k$ of the $N_r$ routed experts per token. When a token's needed neurons are spread across more experts than that budget allows, even the best possible choice misses some of them. Shared-block coverage alone orders the four configurations the same way task success does, so it could rank configurations without running the expensive rollout evaluation. It only ranks correctly, though: two configurations with different coverage values both reach 10/14 success, so a coverage value alone does not predict the exact success rate. Expert utilization shows no collapse: the least-used routed expert anywhere still fires for 27% of tokens against a 50% uniform target.

Shared-block coverage's role in task success is confirmed independently by the static-selection control (Table 5.2): these configurations have $N_k = 0$, so no routing decision is possible, and neurons are kept by rate-ranked pruning rather than assigned to routed experts. Table 6.5 reports all three points. Coverage and success rise together monotonically within this family, confirming that capacity is a binding variable on its own, independent of any router. Section 6.7 compares this

**Table 6.5: Coverage and task success for the static-selection control** (layers 13–22, $N$=132 phase-stratified).

| configuration | active % | coverage (actual) | task success |
|---|---|---|---|
| E8S4A0 | 50 | 0.879 | 4/14 |
| E8S5A0 | 62.5 | 0.925 | 6/14 |
| E8S6A0 | 75 | 0.957 | 12/14 |

**Table 6.6: Configuration sweep on the held-out initial states.** Paired step differences are computed only over states where both the configuration and the dense policy succeed.

| config | layers | act. % | success | mean steps | Δ vs dense | fails on |
|---|---|---|---|---|---|---|
| Dense | n/a | 100 | 12/14 | 85.4 | n/a | 38, 40 |
| E8S2A2 | 13–22 | 50 | 2/14 | 182.5 | +39 | near-total collapse |
| E8S3A2 | 13–22 | 62.5 | 10/14 | 94.4 | +7.5 | 39, 41, 43, 46 |
| E8S3A3 | 13–22 | 75 | 10/14 | 93.0 | +8.0 | 35, 39, 43, 49 |
| E8S4A2 | 13–22 | 75 | **14/14** | 88.1 | +2.9 | none |
| E8S4A2 | 13–28 | 75 | 10/14 | 115.2 | +34.9 | 19, 35, 39, 47 |

family against the routed one directly at matched activation, though their coverage numbers are not directly comparable: coverage means which neurons remain in the static family, and which are selected from those that remain in the routed family.

## 6.5 Task Accuracy

*Ground truth. Every prior stage is a cheaper proxy for this one.*

Table 6.6 reports the configuration sweep. The headline result: a converted policy **matches the dense policy** on the held-out set while activating only three quarters of the FFN. This configuration uses 75% activation, 50% unconditional capacity, and converts ten layers. It additionally solves two episodes that the dense policy fails. This difference is not resolvable at this sample size; the source method reports a similar case, a converted model slightly outperforming dense on one benchmark [8].

Figure 6.5 establishes this resolution directly. Two carves at identical calibration size ($N = 132$, E8S4A2, layers 13–22) differ only in which records were drawn (33% overlap): one is the phase-stratified operating point, the other the proportional same-size control (Table 5.1). Both solve all 14 held-out states. Nothing that should affect the outcome differs, yet the paired per-state difference has mean

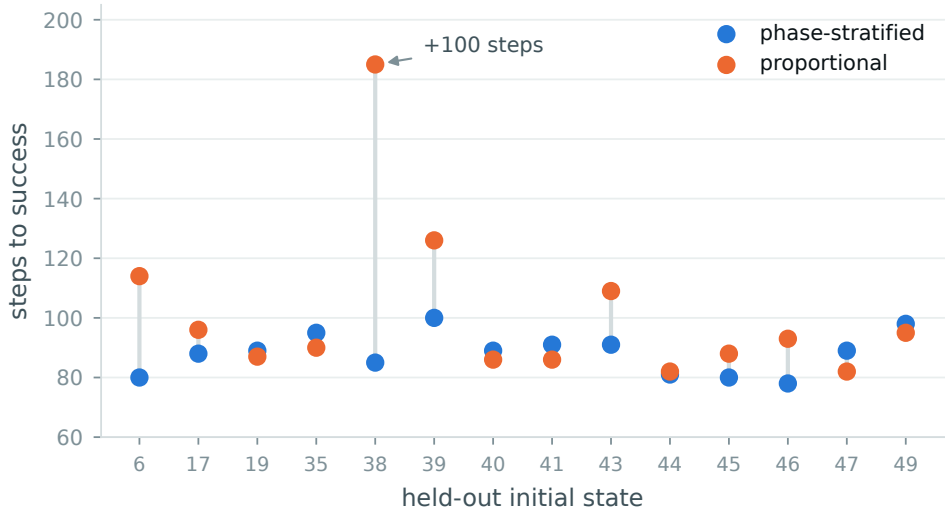



**Figure 6.5: The online noise floor.**

+13.2 steps and sd = 28.0. One episode swings +100 (state 38, which sits on a decision boundary and is also one of the two the dense model fails). At $n = 14$, a paired t-test at 80% power ($\alpha = 0.05$) puts the minimum detectable step effect at $\approx 23$, so every configuration difference reported in this work is below detection, and success rate, not steps, carries the claims.

Unconditional capacity orders the sweep exactly, while the routed fraction contributes nothing: E8S3A2 and E8S3A3 differ by 12.5 points of activation ratio (62.5% vs. 75%) but share the same 37.5% unconditional capacity, and tie in both success (10/14 each) and speed (153.1 vs. 152.1 mean steps). E8S3A3, the source method's own shared/routed split, matches E8S4A2's expert count and 75% activation ratio but not its outcome: it solves ten of the fourteen states against the operating point's fourteen. Across the ten states both solve, E8S3A3 is 8.0 steps slower on average and slower on eight of the ten. The static-selection control of Table 6.5 reproduces the same capacity ordering without any router at all.

The granularity control also shows a smaller, secondary pattern in task success: 14/14 at the operating point's $z = 8$, against 10/14 at $z = 4$ and 12/14 at $z = 16$ and $z = 32$ (Table 6.2), even though reconstruction error stays flat across the same range (Section 6.2). The pattern is non-monotonic, peaking at $z = 8$ rather than trending with granularity, so this thesis does not further resolve whether it reflects granularity itself or this task's small episode count.

## 6.6 Compute and Latency

*Decision fixed: the depth trade-off between task accuracy and FLOP reduction, and the current implementation's bottleneck.*

The configuration with the highest task accuracy (50% unconditional capacity, 10 layers) also converts the fewest layers, so accuracy and savings move in opposite directions along the depth axis. The trade-off is quantified through three measurements: analytic FLOPs, measured wall-clock latency under the current implementation, and the one-off cost of carving itself.

### 6.6.1 Analytic FLOP Count

FLOPs are counted as multiply-accumulate operations per linear projection, comparing converted against dense at matched sequence length. The model configuration is $d_{\text{model}} = 2048$, $d_{\text{ff}} = 11008$, and 36 language layers. In the dense forward pass, the FFN accounts for 87.8% of all linear operations. Two converted configurations are evaluated: E8S4A2 over ten layers (13–22) and over sixteen (13–28), both at 50% unconditional capacity.

At 75% activation ratio, E8S4A2/13–22 reduces FFN FLOPs by 6.94% and linear FLOPs by 6.09%. At the same capacity over six further layers (13–28), the savings rise to 11.10% FFN and 9.74% linear.

### 6.6.2 Measured Wall-Clock Latency

Dense FFN-stack latency is 19.10 ms per batch in prefill and 8.34 ms/token in decode. Latency is measured via `torch.cuda.Event` timing, isolating the FFN stack from the rest of the forward pass and comparing dense against each carve at matched sequence length.[1]

Table 6.7 summarizes the result. Under the current implementation, no configuration realizes the analytic saving: routed E8S4A2 runs 21.6–33.6% slower than dense across the two depths, and static-pruned configurations at matched capacity are flat to slightly slower despite an identical FLOP reduction.[2]

---

[1] Measured with `scripts/benchmark_runtime.py -measure`, which requires CUDA and was run once; unlike `-analytic` and `-carve-cost`, it is not re-verified alongside the rest of this thesis's results, and the routed dispatch-overhead figures (21.6%, 33.6%) have no independent cross-check. Appendix D gives the steps to re-run it.

[2] The cause is a single, localized synchronization defect in the expert-dispatch path, not a flaw in the carve itself (Section 9.6). Disabling it and re-measuring the static-pruned configurations brings corrected latency within a point of the analytic prediction (−7.1%, −10.4%; Table 6.7), validating the FLOP model.

**Table 6.7: Measured latency against the analytic FLOP saving.** All figures are percent change relative to dense. Corrected disables the synchronization defect; not yet re-measured for the routed rows (n/a).

| config | layers | analytic | measured | corrected |
|---|---|---|---|---|
| E8S4A2 (routed) | 13–22 | 6.94% | +21.6% | n/a |
| E8S4A2 (routed) | 13–28 | 11.10% | +33.6% | n/a |
| static-75 | 13–22 | 6.94% | +0.7% | −7.1% |
| static-75 | 13–28 | 11.11% | +4.2% | −10.4% |

### 6.6.3 Carve Cost

Carving the expert partition costs 2.5 minutes on a single GPU, or 3.8% of a single rollout evaluation. This is a one-time cost, paid once before deployment rather than repeated at every inference. This carve-time estimate is measured on a smaller configuration (E8S2A2) than the headline result, and is presumed representative since carve time is dominated by calibration forward passes and clustering rather than by the shared/routed split.

### 6.6.4 The Depth Trade-Off

Comparing the FLOP axis (Table 6.7) against task success (Table 6.6): 13–22 reduces FFN FLOPs by 6.94% at 14/14 success, while 13–28 reduces them by 11.10% at 10/14. Six further layers add about four points of saving and cost four episodes, the shallower conversion having both the highest accuracy and the smallest saving. This is consistent with unconditional capacity being the binding variable: the added layers compound reconstruction error while leaving capacity unchanged. This four-episode gap should not be treated as precise: it is close to the noise floor established in Section 6.5, where re-drawing the calibration subset alone moved outcomes by a comparable margin.

## 6.7 Synthesis

- **Unconditional capacity is the binding variable within a fixed routing scheme.** Holding depth and expert count fixed, success rises with unconditional capacity in both the routed and static families considered on their own, while routed-fraction size does not matter once capacity is fixed (Section 6.5).
- **Routing still beats static selection, for a reason this thesis cannot pin down.** Compared directly at matched activation, the routed configuration beats the static one at two of the three levels tested, despite less unconditional

capacity in both cases. At 50% active, the lowest level tested, static wins: E8S4A0 (50% capacity) reaches 4/14 against E8S2A2's 2/14. At 62.5% active, E8S3A2 (37.5% capacity) reaches 10/14 against E8S5A0's 6/14 (62.5% capacity). At 75% active, E8S4A2 (50% capacity) reaches 14/14 against E8S6A0's 12/14 (75% capacity). Two explanations remain open: per-token variation in which neurons fire may help even with an unskilled router, or the gap may simply be noise (Section 9.1).

- **The second constraint is compounding depth.** Converting additional layers saves more FLOPs but compounds reconstruction error, trading off against task accuracy: the extra layers cost more in accuracy than they save in FLOPs (Section 6.6.2), favoring the shallow conversion despite its smaller saving.

- **Router sophistication and calibration-subset composition do not matter.** The router's coverage matches random selection almost exactly, and even a perfect router (the oracle) recovers only a few percentage points more (Section 6.4). Different calibration draws, whether by size or by subset strategy, yield different neuron-to-expert clusterings, yet task outcomes are indistinguishable across them (Section 6.3).

- **Expert granularity leaves reconstruction error unaffected; its effect on task success is inconclusive.** Holding capacity fixed and varying total expert count from 4 to 32 leaves reconstruction error flat, but task success is non-monotonic and peaks at the operating point (Section 6.5); this thesis does not resolve whether that pattern reflects granularity or the small episode count.

- **The shared expert implements an unconditional capacity floor, not a repository of bimodal knowledge.** The activation profile shows only a weak, roughly twice-chance bimodal separation in this model, and where it does appear, it sits in layers already excluded as non-convertible (Section 6.1).

- **The FLOP saving is real but not yet realized in latency.** Ten converted layers cut FFN compute by roughly 7%, sixteen by roughly 11% (Section 6.6.1), yet no configuration runs faster than dense under the present implementation (Section 6.6.2). A single synchronization defect, not the conversion itself, accounts for the gap.

The training-free results above establish the capacity floor, the depth trade-off, and the current implementation's compute bottleneck. The next chapter evaluates whether the enhancement stage's fine-tuning can move either picture.

# Chapter 7

# Results II: The Enhancement Stage

CMoE's construction is training-free by default; the method also includes an optional fine-tuning stage on top of it. Chapter 6 evaluated the training-free construction; this chapter evaluates the fine-tuning stage, asking a question the source setting never had to answer: does a single-step training objective still help once the policy is judged by whether it succeeds in closed loop, not by how well it predicts the next token?

Section 7.2 reports findings that do not depend on noisy rollouts: diagnostics computed directly from the trained weights, and timing measurements. Section 7.3 measures a second noise floor, this time from training itself rather than from calibration. Only Section 7.4 reports success rates, and reports them as unresolved.

## 7.1 What Is Being Tested, and How

This chapter fine-tunes two configurations, chosen to test different failure modes. The deeper conversion (E8S4A2, layers 13–28) reaches 14/14 at the shallower depth with the same capacity (Section 6.5) but drops to 10/14 once six more layers are converted, a real, measured gap fine-tuning might close. The capacity-floor configuration (E8S2A2) tests the opposite case: there, unconditional capacity itself is the bottleneck (Section 6.7), not a routing decision made within it.

Section 6.4 already predicts a limited outcome for the deeper conversion: the router performs at chance, and even a perfect router would gain only 2.8 percentage points of coverage, so router-directed training has little room to help. The deeper conversion was originally chosen because a trained router might fix its reconstruction-distortion deficit (Section 4.8); this result rules that out, but it still remains the right target as the accuracy gap is real. For the capacity floor, the same logic

**Table 7.1: Fine-tuning enhancement arms for E8S4A2 (layers 13–28) and the capacity-floor configuration E8S2A2 (layers 13–22): loss type, LoRA scope, and seed per arm.**

| arm | configuration | loss | LoRA scope | seed |
|---|---|---|---|---|
| training-free baseline | E8S4A2 13–28 | n/a | n/a | n/a |
| ft_bias_ce | E8S4A2 13–28 | ce | none | 0 |
| ft_scale_ce | E8S4A2 13–28 | ce | none | 0 |
| ft_scale_kl | E8S4A2 13–28 | kl | none | 0 |
| ft_lora_router | E8S4A2 13–28 | kl | router | 0 |
| ft_lora_router_ce | E8S4A2 13–28 | ce | router | 0 |
| ft_lora_cmoe | E8S4A2 13–28 | kl | cmoe | 0 |
| ft_lora_cmoe_ce | E8S4A2 13–28 | ce | cmoe | 0 |
| ft_lora_cmoe_router_ce | E8S4A2 13–28 | ce | cmoe_router | 0 |
| ft_lora_router_ce_repeat1 | E8S4A2 13–28 | ce | router | 0 |
| ft_lora_router_ce_repeat2 | E8S4A2 13–28 | ce | router | 0 |
| ft_lora_router_ce_seed1 | E8S4A2 13–28 | ce | router | 1 |
| training-free baseline | E8S2A2 13–22 | n/a | n/a | n/a |
| ft_s2a2_scale_ce_s0 | E8S2A2 13–22 | ce | none | 0 |
| ft_s2a2_scale_ce_s1 | E8S2A2 13–22 | ce | none | 1 |
| ft_s2a2_scale_ce_s2 | E8S2A2 13–22 | ce | none | 2 |
| ft_s2a2_scale_kl_s0 | E8S2A2 13–22 | kl | none | 0 |
| ft_s2a2_scale_kl_s1 | E8S2A2 13–22 | kl | none | 1 |
| ft_s2a2_scale_kl_s2 | E8S2A2 13–22 | kl | none | 2 |
| ft_s2a2_lora_router_ce_s0 | E8S2A2 13–22 | ce | router | 0 |
| ft_s2a2_lora_router_ce_s1 | E8S2A2 13–22 | ce | router | 1 |
| ft_s2a2_lora_router_ce_s2 | E8S2A2 13–22 | ce | router | 2 |
| ft_s2a2_cmoe_full_s0 | E8S2A2 13–22 | ce | cmoe | 0 |
| ft_s2a2_cmoe_full_s1 | E8S2A2 13–22 | ce | cmoe | 1 |
| ft_s2a2_cmoe_full_s2 | E8S2A2 13–22 | ce | cmoe | 2 |

predicts something more direct: training can only reweight the capacity that exists, not add more. This chapter tests both predictions independently.

Eight fine-tuned arms are run on the deeper conversion, measured against the training-free baseline they share, spanning both self-distillation losses and three adapter scopes, narrow to broad: `router` touches only the router's own weights, `cmoe` reproduces the source method's own target-module list (attention projections plus every expert's gate, up and down projections), and `cmoe_router` is their union. Four arms at three seeds each are run on the capacity-floor configuration, since fine-tuning is not bit-reproducible (Section 7.3) and a single seed is not a point estimate this thesis is willing to report from. Table 7.1 enumerates them with the frozen conversion they share.

Table 7.2 states the training configuration shared by every arm: LoRA rank and target modules, the optimizer preset held constant throughout, and the step

**Table 7.2: Training configuration, held constant across every fine-tuning arm.**

| parameter | value |
|---|---|
| LoRA rank / $\alpha$ | 8 / 32 |
| LoRA target modules | `q_proj, k_proj, v_proj, o_proj` (attention); `gate_proj, up_proj, down_proj` of every expert, shared and routed (`cmoe` scope) |
| LoRA precision | fp32 adapters, bf16 backbone |
| Optimizer | Adam, two parameter groups (LoRA weights; router `extra_scale`) |
| LoRA learning rate | $5.65\times10^{-5}$, $\beta = (0.9, 0.999)$, weight decay 0.01 |
| `extra_scale` learning rate | $1\times10^{-3}$ |
| Load-balancing bias step $\gamma$ | $1\times10^{-3}$, heuristic (no gradient, no optimizer) |
| Gradient accumulation | 8 steps |
| Epochs (bias pass / scale+LoRA) | 1 / 2 |
| Optimizer steps (E8S4A2/13–28, $N$=132) | ≈34 total (2 epochs × ≈17 steps/epoch) |
| Seeds | 3 per stochastic arm (E8S2A2 study), reported as a range |

count, which Section 7.2.2 reasons from directly. Every arm here uses CMoE's [8] code preset.

## 7.2 Findings Independent of the Noise Floor

### 7.2.1 The Training Signal Is Weak

The converted policy agrees with dense on only 57.9% of generated action tokens (E8S4A2/13–22) but matches it exactly on task success (14/14 vs. dense's 12/14). In action space the same model is close: gripper sign agreement is 95.7%, and per-DOF error is 4.3–8.6% of each dimension's range. Actions are multi-digit numbers, so a neighboring digit is a full token-level disagreement and a negligible action difference. This already bounds what any distillation-style objective can achieve before training starts.

The objective becomes even weaker once training begins. Under teacher forcing, the converted model's argmax departs from the dense token sequence at only 0.3% of the way through the average 222-token record, the first or second token, so almost the entire loss is computed on prefixes the model would never generate in closed loop: loss is roughly tenfold higher after that divergence point than before it. This is exposure bias [41]: a model trained only on ground-truth prefixes never learns to handle the imperfect prefixes its own free-running generation produces. Here it is measured rather than argued.

### 7.2.2 Across Scope: the Objective Moves, Behavior Doesn't

Router-only adaptation, the narrowest scope, changes nothing: digit-level loss and token agreement are unchanged, and the adapter direction has cosine 0.645 to an identical-seed repeat but −0.04 across seeds (statistically orthogonal), indicating no reproducible signal at the 34 optimizer steps available (Table 7.2).

Broad scope drives perplexity from 5.07 to 1.86, comparable in direction to the source method's own reported gain (7.32 to 5.92) [8]. The exposure-bias divergence point, however, stays fixed at 0.3%: the model predicts better under teacher forcing, but its free-running behavior does not change. Hard cross-entropy against the teacher's stored tokens is sequence-level distillation [39] and per-position soft KL is word-level [40]. Sequence-level distillation is meant to work by training the student on its own free-running output, mistakes included, but the loss measured here is computed under teacher forcing: the student never sees its own output, so that training signal never happens.

Broad scope also breaks a property that should stay fixed. The distillation teacher is the sum of all experts, so adapting the projection matrices means the converted model is no longer an exact re-parameterization of the validated dense weights.

This is measured as token agreement between the dense model and the carved model with every expert forced active. In exact arithmetic this agreement is 1.0, since summing every expert reproduces the dense computation exactly. Measured, it sits at 0.973–0.975 before any training: bf16 rounding, not a training effect. Narrow scopes leave this baseline unchanged; every broad scope degrades it substantially, to 0.71–0.81, because the distillation target is itself assembled from the weights the adapter is editing, and drifts as training proceeds.

### 7.2.3 Cost

Fine-tuning itself is cheap relative to a single evaluation. Broad-scope adaptation, however, adds 116% to evaluation wall-clock per environment step (1008 to 2181 ms): for a method whose premise is a speedup, adapting every layer erases the saving several times over. Two qualifications matter here. First, this figure includes simulator-stepping time, which is constant across arms; the increase attributable to inference alone is therefore larger than +116%. Second, the LoRA adapters were kept separate from the base weights during evaluation rather than folded into them, an extra computation a standard deployment step would remove.

Routing stays healthy under fine-tuning: zero dead experts in every converted layer of every arm, with normalized routing entropy in the range 0.968–0.997 throughout. The source method's union regularisation, an auxiliary loss term that penalizes collapsed routing, is not needed here, since collapse never occurs.

## 7.3 The Training-Noise Floor

Repeated runs of a single arm under identical commands produce different weights. The backward pass is not bit-reproducible, and a zero-initialized parameter with near-noise-level gradients has its update direction set by that non-determinism rather than by a real signal. Four runs of one configuration (E8S4A2/13–28, `lora_router_ce`): three with byte-identical commands at the same seed score 12/14, 6/14, and 9/14; a fourth at a different seed scores 10/14. The worst pair disagrees on 13 of the 14 episodes. The closed-loop metric then amplifies a marginal logit into a different trajectory. The floor is not a constant: it scales with how many episodes sit near the success boundary. The mid-range configuration's success count ranges over six episodes across repeated runs (from 6/14 to 12/14); the capacity-floor configuration, where almost every episode is already decided, ranges over only two. This is why differences between arms are trustworthy at 50% activation but not at 75%.

Section 6.5 measured a noise floor from calibration-subset composition before any fine-tuning was run. That floor was large enough that no configuration difference reported anywhere in this thesis can be told apart from noise. The floor measured here comes from a different source, gradient non-determinism during training rather than calibration-subset composition. It is just as large: a training-noise range of six episodes out of fourteen is enough, on its own, to swallow the differences this thesis reports. Two unrelated sources of noise producing floors this large is evidence that the floor is a property of this evaluation's dataset and conversion method, not of fine-tuning. It is not evidence that the floor is a property of VLAs in general. A larger or more varied dataset, or a different conversion method, is untested here and may not hit the same limit.

## 7.4 Success-Rate Results: Reported, Not Concluded

Table 7.3 reports every arm at the deeper conversion. The differences are small next to the noise floor from Section 7.3, and McNemar's test agrees: no arm's win/loss split against the training-free baseline is significant at $p < 0.05$ (e.g. $p = 0.219$ for the two LoRA-CMoE-scope arms, CE and KL). The table is not a ranking; it is included because its per-episode detail feeds that noise-floor analysis. The CE and KL arms illustrate why: both score 6/14, yet fail on almost disjoint episodes, sharing only one rescued case. Figure 7.1 shows this directly.

At the capacity-floor configuration, solved counts vary little enough to compare directly, and every arm sits at or below the training-free baseline, consistent with capacity itself being the bottleneck: router calibration cannot add what is missing.

**Table 7.3: Task success and mean episode length per enhancement arm, evaluated against the training-free baseline, for both fine-tuned configurations.** Episode length is averaged over successful episodes only.

| arm (E8S4A2, layers 13–28) | solved / 14 | mean steps |
|---|---|---|
| **training-free baseline** | **10** | **115** |
| bias ce | 9 | 99 |
| scale ce | 8 | 97 |
| scale kl | 12 | 137 |
| lora router | 10 | 108 |
| lora router ce | 12 | 113 |
| lora cmoe | 6 | 90 |
| lora cmoe ce | 6 | 132 |
| lora cmoe router ce | 10 | 110 |
| *E8S2A2, layers 13–22, mean over three seeds* | | |
| scale-only, CE | 1.0 | 178 |
| scale-only, KL | 2.0 | 146 |
| LoRA router, CE | 1.3 | 188 |
| full recipe | 2.0 | 114 |
| **training-free baseline** | **2** | **183** |

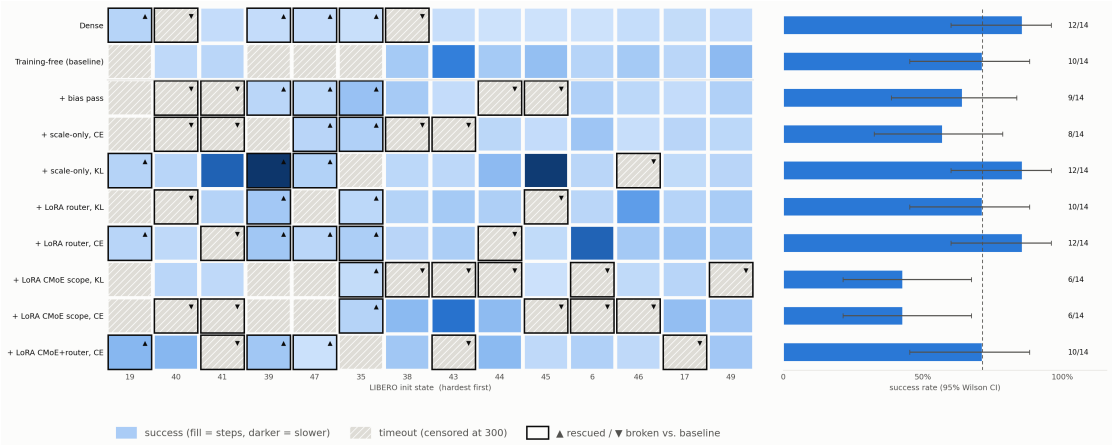



**Figure 7.1: Per-episode outcomes, E8S4A2/13–28.** Rows are arms, columns are the 14 evaluation init states.

Mean episode length is noisier, since each mean comes from as few as one success; the full recipe (114 steps) and LoRA router CE (188) fall on opposite sides of the baseline (183), but the sample is too small to trust that gap. Six of the fourteen episodes are never solved by any arm, against a dense model that solves twelve of

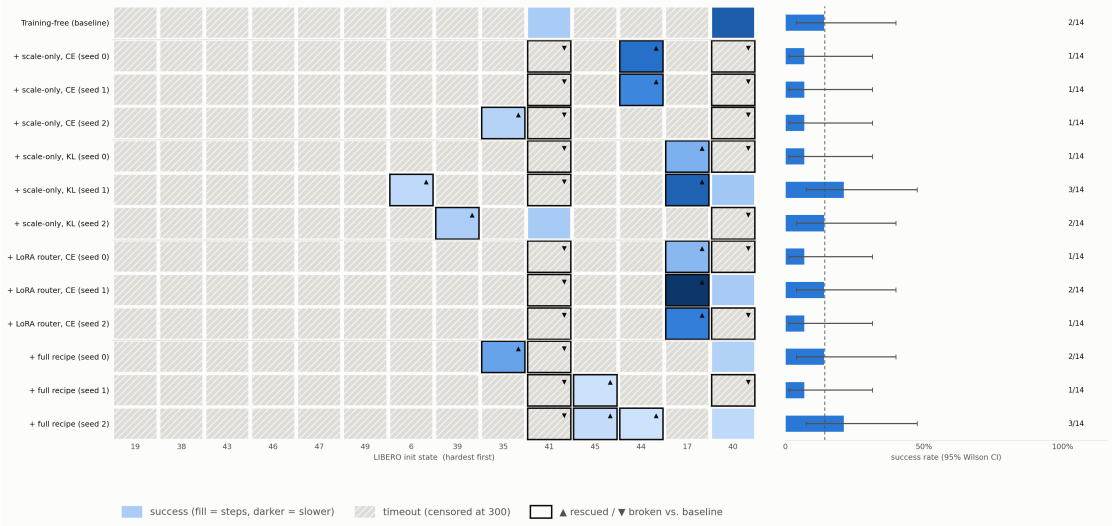



**Figure 7.2: Per-episode outcomes, E8S2A2/13–22, three seeds. ▲/▼** mark differences from the training-free baseline (2/14); at this success rate they reflect noise, not a real effect.

the fourteen in 79–98 steps; the episodes that are solved are scattered across seeds (Figure 7.2) and slower than dense throughout (96–292 steps).

## 7.5 Why the Recipe Does Not Transfer

Four properties of this setting explain why, none of which the source method's own evaluation has to deal with. First, the distillation target does not stay fixed: under broad-scope training it is assembled from the same weights the adapter is editing, so it drifts as training proceeds (Section 7.2.2), and training is not guaranteed to move the model toward something better than where it started. Second, the training objective and the evaluation metric measure different things: training scores next-token prediction under teacher forcing, while evaluation runs the model on its own outputs for hundreds of steps [38], the exposure-bias gap measured in Section 7.2.1. Third, token-level loss is a poor proxy for action error: 42% token disagreement corresponds to roughly 6% action error and 96% gripper agreement (Section 7.2.1). Finally, the available supervision is far smaller and narrower than the source method's own: roughly 70× fewer tokens, 59k action tokens from a single task and instruction against $2{,}048 \times 2{,}048 \approx 4.2$M tokens of general text.

The next chapter (Chapter 8) returns to the mechanism these two chapters establish: why unconditional capacity, not routing or fine-tuning, is what the shared expert actually buys.

# Chapter 8

# Discussion

## 8.1 The Shared Expert as an Unconditional Capacity Floor

The source method describes the shared expert as a repository of commonly needed knowledge, identified by the bimodal structure of activation frequencies [8]. Chapter 6 establishes that this structure is absent where conversion occurs, yet the shared expert remains essential: its size, not its contents, is the binding variable (Section 6.7). This raises a question: why would a robot-manipulation policy depend more on unconditional capacity than the text models the source method was built for?

A plausible answer is a difference in how failures play out. A text model's per-token loss is graded, so one routing miss costs a small, local increase in perplexity. An action chunk is different: it is a sequence of digit tokens, and one bad token can corrupt the rest of the chunk (Section 2.1.1), then compound across a rollout into a failed episode. A smaller always-on block does more than raise average error: it raises the chance that some step needs a neuron the routed half does not have. In closed-loop execution, that single miss can fail the whole episode, a risk a single-step benchmark never faces. In practice, this means the unconditional fraction is the variable to tune: it can be adjusted from a conversion checkpoint in minutes (Section 6.6.3), without touching the router at all.

## 8.2 Why the Router Performs at Chance

Chapter 6 found that the router earns nothing over random selection; even the oracle bound shows there was little to earn (Section 6.4). The mechanism follows directly from the bimodality result. The router scores each expert using one representative neuron's activation as a stand-in for the whole cluster. That stand-in only works if

the clustering step grouped genuinely similar neurons together. Section 6.1 shows that this activation structure is largely absent here. The router is therefore not failing on its own terms: it inherited a partition where that condition was never met.

This finding is narrow: one model, one task family. The router may still work as intended wherever that bimodal premise holds. Routing itself is not worthless, either: at two of three matched activation levels, the routed configuration beats a static one with more unconditional capacity (Section 6.7). The router's own coverage cannot explain that gap, and this thesis does not resolve it.

## 8.3 What Transfers From CMoE, and What Does Not

The analytical restructuring transfers and produces a policy that matches dense while activating three quarters of the FFN. Needing very little calibration data also transfers, in a stronger form: calibration size's effect is smaller than the measurement's own noise floor. The bimodality premise does not hold. The router performs at chance against random selection, though whether routing itself helps beyond capacity remains open (Section 6.7). The enhancement stage optimizes its objective without reaching the task.

The staged pipeline itself transfers independently of any of these specific findings. Each stage fixes one decision and states what it cannot decide. Two of the five even disagree by construction: profiling excludes exactly the layers distortion finds easiest to reconstruct (Section 4.2), which shows the diagnostics measure different properties of a layer rather than duplicating each other. The pipeline's structure, not the specific numbers it produced here, is the more portable contribution: that structure should carry over to any post-training MoE-fication method applied to any autoregressive policy, whether or not CMoE's own premises hold there.

The analytic FLOP saving (6.94% at the operating point, rising to 11.10% at the deeper conversion, Section 6.6.1) is a property of the conversion itself. Section 6.6.2 attributes the fact that no configuration runs faster than dense to a single, localized synchronization defect in the dispatch path, not to the conversion. Until the dispatch-path defect is fixed, this conversion's deployment case rests on accuracy alone: the compute saving is real on paper but not yet in wall-clock terms.

Section 3.2 compares the capacity-floor fix (Section 8.1) against related work, and the practical distinction is that it needs no training step at all. RLRC's pruning of a VLA backbone needs reinforcement-learning recovery to reach the original success rate; AdaMoE copies the dense FFN into each expert and still needs full fine-tuning to make those copies diverge. That claim is narrower than either prior result, though: it fixes one specific failure, insufficient unconditional capacity, rather

than recovering from compression in general. Whether a training-free fix exists once the bimodality premise fails entirely, as it does here (Section 8.2), is left open.

## 8.4 Measurement Resolution in Few-Episode Robot Evaluation

Two independently obtained noise floors agree on a resolution limit large enough to swallow every configuration difference this thesis reports. This floor is a property of the present scenario, not a general limit on VLA evaluation: it reflects the size and single-task diversity of the dataset used here, and the specific conversion method tested. Whether a larger or more varied benchmark would shrink this floor is untested (Chapter 9).

The floor also varies: its width depends on how many episodes sit near their own decision boundary, so it resolves a comparison at the capacity floor while leaving one at the operating point open. At 75% activation (Chapter 7), the floor shows the apparent differences between arms are noise, not a real effect. That is a limitation of this evaluation, not evidence that fine-tuning or a larger calibration set would not help.

## 8.5 Validity Boundaries

To keep the claims above from being over-read, the scope is stated explicitly: one policy, one task, one instruction, fourteen held-out episodes, language stack only. Offline observations are correlated within episodes. The Stage-3 fidelity metrics (MAE against dense, Section 6.3) were never checked against task success, and at this evaluation scale they cannot be: the sample is too small to tell a real correlation from noise. Profiling observes only the prefill token population. The efficiency benefit is reported in Section 6.6; conclusions about deployment value are bounded by it, and the routed-configuration dispatch-overhead figures specifically remain a first-iteration measurement pending GPU re-validation (Section 6.6.2).

The next chapter turns these boundaries into a research agenda.

# Chapter 9

# Future Work

## 9.1 Statistical Power for Online Comparison

The binding constraint on every online comparison is resolution, not missing analysis. The fix is to treat the conversion draw as a random effect: three conversions per condition at fixed calibration size, pooled. This is the only change that separates a real treatment effect from subset noise, and every further claim about the enhancement stage depends on it. It would also settle the question Section 8.2 leaves open: whether the routed configuration's edge over static pruning (Section 6.7) is a real effect of per-token variability, or just the 14-episode sample size.

Once online comparisons have that power, joining per-episode offline metrics to per-episode outcomes would show whether any offline metric predicts task success (Section 8.5).

## 9.2 Beyond a Single Task and Dataset

Every noise floor reported in this thesis (Sections 6.3, 7.3) comes from a single LIBERO-Goal task: 374 calibration records and 14 held-out episodes. Whether these floors are a real limit on what more calibration data or fine-tuning can achieve, or just an artifact of this one dataset and of CMoE as the conversion method, is untested. Repeating the calibration-size and fine-tuning sweeps on a larger, multi-task dataset, and against at least one alternative post-training MoE-fication method, would separate a dataset limitation from a method limitation.

All results in this thesis also use single-task calibration for the conversion itself. The first step: profile shared-expert selection separately for each LIBERO-Goal task, then compare the overlap between tasks. This is the same measurement the source method uses to argue domain invariance across text datasets [8]. High overlap would extend the calibration-insensitivity result from *size* to *content*; low

overlap would bound it to this task. The measurement is offline, so it needs no rollouts and is not subject to Section 6.5's resolution limit. Beyond that, the same question extends to other LIBERO suites and other VLA backbones.

## 9.3 Per-Layer Distortion and Capacity

Section 6.2 measures each layer in isolation; how error compounds across the converted block is not measured. Feeding the converted model dense hidden states up to a chosen cutoff layer, then switching to its own outputs, would show where that compounding becomes critical and could motivate hybrid architectures that convert only part of the range.

The distortion profile also varies by layer, while capacity is applied uniformly here. Giving harder-to-reconstruct layers more capacity would show whether the capacity floor holds per layer.

## 9.4 Token-Position Separation

Profiling observes prefill tokens (image patches and instruction text) and never the decode-side action tokens, although both populations are processed at inference. Profiling prefill and decode tokens separately would measure how much this omission matters, and show whether a conversion calibrated on both behaves differently once action decoding starts.

## 9.5 Vision Tower Conversion

All conversions in this thesis are language-stack only. Applying Stages 1 and 2 (Sections 4.3–4.4) to the vision tower's SwiGLU blocks would function as a gate: if the vision FFNs are clustering-degenerate under the same criterion, that is establishable in a single profiling run and is itself reportable, since the source method scopes its claims to decoder-only text transformers. If they are convertible, the capacity-floor result (Section 6.7) could be tested against a different token distribution, the strongest available test of whether that result is a property of the method or of the language stack.

## 9.6 Closing the Runtime-Measurement Gaps

Section 6.6.2 isolates a device-to-host synchronization that erases the analytic FLOP saving for routed configurations; the corrected static-configuration measurements confirm the diagnosis without fixing it. Rewriting the dispatch path as a fused

kernel, then re-measuring routed-configuration latency on GPU hardware, would close the gap between what the method saves in principle and what the current implementation delivers. The routed-configuration figures reported in this thesis are a first-iteration measurement, pending that re-validation.

The carve-cost figure (Section 6.6.3) is just as provisional. It comes from a single `e8s2a2` carve timed against a single baseline evaluation, not the `e8s4a2` configuration behind the headline result. It also rests on an untested assumption: that conversion time is set by the calibration and clustering stages, not by the shared/routed split. Re-running the carve-cost benchmark (Appendix D) across multiple seeds and expert configurations, including `e8s4a2`, would test that assumption directly.

## 9.7 Action-Chunk Ensembling

Every online result in this thesis uses single-chunk inference, the highest-variance decoding setting available and the one under which the training-noise floor (Section 7.3) is measured. The source method reports that sparse models gain far more from self-consistency ensembling than dense ones (+4.72 percentage points vs. +0.57), which makes ensembling a plausible fix for that floor rather than a routine robustness check, and it is currently untested here.

The next chapter concludes.

# Chapter 10

# Conclusion

This thesis asked what binds the quality of analytical, calibration-based FFN-to-MoE conversion when the target is a vision-language-action policy rather than a text model.

The answer is unconditional capacity, within a fixed routing scheme. Holding the per-token activation ratio fixed and moving capacity out of the always-on block degrades task performance; holding the always-on block fixed and varying the routed fraction does not. At three-quarters FFN activity, with half unconditional, a converted policy matches the dense policy on held-out episodes without any gradient training. Whether routing itself adds anything beyond capacity remains open (Section 9.1).

The candidates that did not bind are just as informative. Calibration size and calibration strategy make no difference: performance stays flat across a forty-seven-fold range of calibration size, and that flatness sits below the measurement's own noise. Expert granularity leaves reconstruction error flat across a fourfold range of expert count as well, though task success is non-monotonic, a pattern this thesis leaves unresolved. The analytical router adds nothing beyond random expert selection, and even a perfect router would gain only a few points more. Fine-tuning is no different: closed-loop success stays the same, even though training does move the objective it was optimizing.

Two contributions extend beyond the specific configuration: the staged pipeline (Section 8.3), argued there to carry over to any post-training MoE-fication method, and the noise-floor protocol (Section 4.9.2), a measurement technique not tied to this dataset or task. Whether either actually helps on other datasets, tasks, or methods is future work, not an established fact.

The findings are bounded to one policy, one manipulation task, and the language stack. Chapter 9 sets out what would extend them: more statistical power, a broader dataset, vision-tower conversion, and closing the runtime-measurement gaps.

# Appendix A

# Configuration Evidence Trail

This table lists every hyperparameter used in this thesis: the stage that fixed it, and the evidence behind it. Chapters 4 and 5 give the full reasoning; this table is a quick reference for tracing any configuration choice back to its evidence.

**Table A.1: Configuration evidence trail.**

| parameter | value | fixed by | evidence |
|---|---|---|---|
| $k_{act}$ | 50 | Stage 1 | worst-case degenerate/carve separation peaks here (Table 6.1); distortion insensitive to it over 50–100 (Section 6.2) |
| Excluded layers | 1–9 (degenerate), 10–12 (transition) | Stage 1 | effective-pool and $p_{95}$ criteria (Figure 6.1) |
| Total experts $z$ | 8 (swept 4–32) | source method; granularity control | divides $d_h$ = 11,008 exactly; reconstruction error flat, task success non-monotonic across $z \in$ {4,8,16,32} (Table 6.2) |
| Shared experts $N_s$ | 1–4 in the routed family, 4–6 in the static family | Stage 2 capacity sweep | Table 6.2, Table 5.2 |
| Routed active $N_k$ | 2 (operating point), 3 (source-method split) | source-method comparison | Table 6.6 |
| Conversion depth | layers 13–22 (validated), 13–28 (fine-tuning target) | Stage 2 / Stage 5 | Table 6.6, Section 6.6.4 |
| Calibration size $N$ | 8, 33, 66, 132, 264, 374 | Stage 3 | Figure 6.3; flat across the full range |
| Calibration subset strategy | phase-stratified (default), proportional (control) | Stage 3 | Section 4.5 |
| Operating configuration | E8S4A2, layers 13–22, $N$=132 phase-stratified | Stages 1–5 jointly | matches dense on held-out task success (Table 6.6) |
| LoRA rank / $\alpha$ | 8 / 32 | source method (`cmoe` scope) | Table 7.2 |
| LoRA scope | router, cmoe, cmoe_router | experimental variable | Section 7.1 |
| Optimizer preset | code preset (throughout) | held constant to avoid confounding | Table 7.2 |
| Seeds (stochastic arms) | 3 | fine-tuning is not bit-reproducible | Section 7.3 |

# Appendix B

# Full Per-Layer Tables

The tables below give activation profiling across all 36 language layers at $k_{act} = 50$ (374 observations, the reference run cited throughout Chapter 6), and per-layer distortion across the full candidate range (layers 5–35, E8S4A2, 374 observations). They are reproduced in full so the zone boundaries used in Chapter 6 can be checked against the underlying measurements, not just the summary figures.

| layer | active neurons (%) | $p_{95}$ rate | max rate |
|---|---|---|---|
| 0 | 94.9 | 0.0190 | 0.245 |
| 1 | 1.1 | 0.0000 | 1.000 |
| 2 | 1.6 | 0.0000 | 1.000 |
| 3 | 1.5 | 0.0000 | 0.995 |
| 4 | 17.5 | 0.0197 | 0.691 |
| 5 | 3.9 | 0.0000 | 0.924 |
| 6 | 6.2 | 0.0005 | 0.818 |
| 7 | 7.4 | 0.0015 | 0.923 |
| 8 | 0.9 | 0.0000 | 1.000 |
| 9 | 1.1 | 0.0000 | 1.000 |
| 10 | 68.9 | 0.0145 | 0.995 |
| 11 | 61.6 | 0.0171 | 0.986 |
| 12 | 63.0 | 0.0168 | 0.947 |
| 13 | 93.2 | 0.0171 | 0.723 |
| 14 | 91.6 | 0.0174 | 0.621 |
| 15 | 94.7 | 0.0184 | 0.625 |
| 16 | 94.8 | 0.0182 | 0.565 |
| 17 | 93.9 | 0.0193 | 0.651 |
| 18 | 92.0 | 0.0193 | 0.505 |
| 19 | 96.3 | 0.0190 | 0.373 |
| 20 | 96.1 | 0.0169 | 0.618 |
| 21 | 98.2 | 0.0184 | 0.395 |

| layer | active neurons (%) | $p_{95}$ rate | max rate |
|---|---|---|---|
| 22 | 99.1 | 0.0168 | 0.653 |
| 23 | 99.7 | 0.0170 | 0.612 |
| 24 | 99.6 | 0.0154 | 0.571 |
| 25 | 99.5 | 0.0159 | 0.756 |
| 26 | 99.6 | 0.0157 | 0.654 |
| 27 | 99.6 | 0.0151 | 0.764 |
| 28 | 99.5 | 0.0156 | 0.834 |
| 29 | 99.3 | 0.0146 | 0.766 |
| 30 | 97.9 | 0.0129 | 0.716 |
| 31 | 96.5 | 0.0133 | 0.710 |
| 32 | 99.4 | 0.0122 | 0.997 |
| 33 | 100.0 | 0.0146 | 0.829 |
| 34 | 85.8 | 0.0183 | 0.509 |
| 35 | 43.0 | 0.0219 | 0.674 |

| layer | rel. $L_2$ | cosine distance |
|---|---|---|
| 5 | 0.0290 | 0.0035 |
| 6 | 0.0312 | 0.0036 |
| 7 | 0.0444 | 0.0071 |
| 8 | 0.0369 | 0.0040 |
| 9 | 0.0412 | 0.0047 |
| 10 | 0.0531 | 0.0069 |
| 11 | 0.0607 | 0.0076 |
| 12 | 0.0590 | 0.0070 |
| 13 | 0.0508 | 0.0050 |
| 14 | 0.0504 | 0.0049 |
| 15 | 0.0478 | 0.0044 |
| 16 | 0.0472 | 0.0043 |
| 17 | 0.0460 | 0.0040 |
| 18 | 0.0451 | 0.0035 |
| 19 | 0.0457 | 0.0036 |
| 20 | 0.0472 | 0.0035 |
| 21 | 0.0481 | 0.0038 |
| 22 | 0.0527 | 0.0047 |
| 23 | 0.0564 | 0.0050 |
| 24 | 0.0609 | 0.0057 |
| 25 | 0.0660 | 0.0064 |
| 26 | 0.0702 | 0.0063 |
| 27 | 0.0727 | 0.0064 |
| 28 | 0.0817 | 0.0075 |
| 29 | 0.0932 | 0.0093 |

| layer | rel. $L_2$ | cosine distance |
|---|---|---|
| 30 | 0.0502 | 0.0125 |
| 31 | 0.0585 | 0.0145 |
| 32 | 0.0724 | 0.0178 |
| 33 | 0.0836 | 0.0196 |
| 34 | 0.1083 | 0.0253 |
| 35 | 0.2183 | 0.0365 |

# Appendix C

# Porting Activation Profiling to a Vision-Language Backbone

Stage 1 (Section 4.3) scores activations by intercepting each record's hidden state before it enters the first converted layer. Porting that collection to VLA-0's vision-language backbone, for the language-layer conversion this thesis performs, required two adaptations beyond the reference text-only implementation.

First, positional information in Qwen2.5-VL [13] is a fixed three-dimensional encoding tied to image and text layout, not the batch-indexed array the reference implementation assumes. The calibration collection therefore had to detect and preserve that structure rather than naively broadcasting it across records.

Second, calibration metadata (attention mask, position encoding) is captured from the *first* calibration record rather than the last. Same-task LIBERO [14] records share identical positional structure, so this is a correctness fix relative to the reference behavior, not a design choice.

# Appendix D

# Reproduction

This appendix maps each reported result to the script that produced it. Every conversion and evaluation result is bit-reproducible under the determinism flags of Section 5.4; fine-tuning is not. Repeated runs of one deeper-conversion arm first exposed this (Section 7.3), which is why every capacity-floor arm is run at three seeds and reported as a range rather than a single number. Per-run fine-tuning configuration (loss, LoRA scope, seed) is recorded in the conversion checkpoint each arm was trained from, not re-derived from this table.

**Table D.1: Result-to-script reproduction map.**

| result | script | notes |
|---|---|---|
| Layer convertibility (Figure 6.1, Table 6.1) | `scripts/analyze_qwen_activations.py` | swept $k_{act} \in \{10,50,100,1000\}$; deterministic given the calibration set |
| Per-layer distortion (Figure 6.2) | `scripts/analyze_moe_distortion.py` | full layer range 5–35 scanned in one run |
| Fidelity vs. calibration size (Figure 6.3, Table 6.3) | `scripts/analyze_fidelity.py` | one run per calibration size per subset strategy |
| Routing ceiling (Figure 6.4) | `scripts/analyze_routing_ceiling.py` | 24-record held-out subsample (Section 5.2) |
| Rollout evaluation (Table 6.6, Figure 6.5) | `eval/eval_libero_moe.py` | bit-reproducible under the determinism flags of Section 5.4 |
| Analytic FLOPs, measured latency, carve cost (Section 6.6) | `scripts/benchmark_runtime.py` | `-analytic`/`-carve-cost` need no GPU; `-measure` requires CUDA (Section 6.6.2) |
| Fine-tuning arms (Table 7.1, Table 7.2) | `scripts/run_s2a2_seed_study.sh`, per-arm `moefication/finetune.py` | per-run fine-tuning configuration is recorded in the conversion checkpoint, not re-derived |
| Per-episode outcome figures (Chapter 7) | `scripts/collect_results.py`, `scripts/figures/plot_outcome_matrix.py` | reads `finetune_analysis/episodes.csv`, aggregated from every run's `results.json` |

# Bibliography


[1] Kento Kawaharazuka, Jihoon Oh, Jun Yamada, Ingmar Posner, and Yuke Zhu. *Vision-Language-Action Models for Robotics: A Review Towards Real-World Applications*. arXiv:2510.07077. Preprint. 2025 (cit. on pp. 1, 6, 14).

[2] Ankit Goyal, Hugo Hadfield, Xuning Yang, Valts Blukis, and Fabio Ramos. *VLA-0: Building State-of-the-Art VLAs with Zero Modification*. arXiv:2510.13054 NVIDIA. Code: `https://github.com/NVlabs/vla0`. 2025 (cit. on pp. 1, 6, 7, 22).

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. «Attention Is All You Need». In: *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*. arXiv:1706.03762. 2017 (cit. on p. 1).

[4] Elias Frantar and Dan Alistarh. «SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot». In: *Proceedings of the 40th International Conference on Machine Learning (ICML 2023)*. arXiv:2301.00774. 2023, pp. 10323–10337 (cit. on pp. 2, 13, 15).

[5] Zichang Liu, Jue Wang, Tri Dao, et al. «Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time». In: *Proceedings of the 40th International Conference on Machine Learning (ICML 2023)*. Oral. arXiv:2310.17157. 2023, pp. 22137–22176 (cit. on pp. 2, 13, 16).

[6] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. «GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers». In: *Proceedings of the 11th International Conference on Learning Representations (ICLR 2023)*. arXiv:2210.17323. 2023 (cit. on pp. 2, 15).

[7] Yuxuan Chen, Yixin Han, Yize Huang, and Xiao Li. *RLRC: Reinforcement Learning-based Recovery for Compressed Vision-Language-Action Models*. arXiv:2506.17639. Preprint. Project: `https://rlrc-vla.github.io`. 2025 (cit. on pp. 2, 13).

[8] Zehua Pei, Hui-Ling Zhen, Lancheng Zou, et al. «Analytical FFN-to-MoE Restructuring via Activation Pattern Analysis». In: *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL 2026)*. arXiv:2502.04416. 2026, pp. 4777–4789 (cit. on pp. 2, 3, 7, 9, 12, 15, 25, 28, 31, 38, 39, 43, 46).

[9] Anthony Brohan, Noah Brown, Justice Carbajal, et al. «RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control». In: *Proceedings of The 7th Conference on Robot Learning (CoRL 2023)*. Vol. 229. Proceedings of Machine Learning Research. arXiv:2307.15818. 2023, pp. 2165–2183 (cit. on p. 6).

[10] Kevin Black, Noah Brown, Danny Driess, et al. *$\pi_0$: A Vision-Language-Action Flow Model for General Robot Control*. arXiv:2410.24164. Physical Intelligence. Preprint. 2024 (cit. on p. 6).

[11] Mustafa Shukor, Dana Aubakirova, Francesco Capuano, et al. *SmolVLA: A Vision-Language-Action Model for Affordable and Efficient Robotics*. arXiv:2506.01844. HuggingFace. Preprint. 2025 (cit. on pp. 6, 13).

[12] Seonghyeon Ye, Joel Jang, Byeongguk Jeon, et al. «Latent Action Pretraining from Videos». In: *International Conference on Learning Representations (ICLR 2025)*. arXiv:2410.11758. 2025 (cit. on p. 6).

[13] An Yang, Baosong Yang, Beichen Zhang, et al. *Qwen2.5 Technical Report*. arXiv:2412.15115. Preprint. 2024 (cit. on pp. 6, 8, 22, 55).

[14] Bo Liu, Yifeng Zhu, Chongkai Gao, et al. «LIBERO: Benchmarking Knowledge Transfer for Lifelong Robot Learning». In: *Advances in Neural Information Processing Systems (NeurIPS 2023)*. arXiv:2306.03310. 2023 (cit. on pp. 7, 22, 55).

[15] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, et al. «Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer». In: *5th International Conference on Learning Representations (ICLR 2017)*. arXiv:1701.06538. 2017 (cit. on p. 9).

[16] Damai Dai, Chengqi Deng, Chenggang Zhao, et al. *DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models*. arXiv:2401.06066. Preprint. 2024 (cit. on p. 9).

[17] Roy Jonker and Ton Volgenant. «A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems». In: *Computing* 38 (1987), pp. 325–340 (cit. on p. 10).

[18] Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. «MoEfication: Transformer Feed-Forward Layers are Mixtures of Experts». In: *Findings of the Association for Computational Linguistics: ACL 2022*. arXiv:2110.01786. 2022, pp. 877–890 (cit. on pp. 10, 12).

[19] Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. *Auxiliary-Loss-Free Load Balancing Strategy for Mixture-of-Experts*. arXiv:2408.15664. Preprint. 2024 (cit. on p. 11).

[20] Edward Hu, Yelong Shen, Phillip Wallis, et al. *LoRA: Low-Rank Adaptation of Large Language Models*. arXiv:2106.09685. ICLR 2022. 2021 (cit. on pp. 11, 13, 19).

[21] Simiao Zuo, Qingru Zhang, Chen Liang, Pengcheng He, Tuo Zhao, and Weizhu Chen. «MoEBERT: From BERT to Mixture-of-Experts via Importance-Guided Adaptation». In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2022)*. arXiv:2204.07675. 2022 (cit. on p. 12).

[22] Jaeseong Lee, Seung-won Hwang, Wonpyo Park, and Mingi Ji. «Breaking ReLU Barrier: Generalized MoEfication for Dense Pretrained Models». In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP 2024)*. 2024, pp. 10097–10107 (cit. on p. 12).

[23] Ziyu Zhao, Tong Zhu, Zhi Zhang, et al. *ExpertWeaver: Unlocking the Inherent MoE in Dense LLMs with GLU Activation Patterns*. arXiv:2602.15521. Preprint. 2026 (cit. on p. 12).

[24] Ivan Novikov. *MLPMoE: Zero-Shot Architectural Metamorphosis of Dense LLM MLPs into Static Mixture-of-Experts*. arXiv:2511.21089. Unreviewed preprint; no peer-reviewed venue at time of writing. 2025 (cit. on p. 12).

[25] Shangqian Gao, Hua Ting, Reza Shirkavand, et al. «ToMoE: Converting Dense Large Language Models to Mixture-of-Experts through Dynamic Structural Pruning». In: *Transactions on Machine Learning Research* (Jan. 2026). arXiv:2501.15316. URL: `https://openreview.net/forum?id=RFHq46pjb6` (cit. on p. 12).

[26] Tong Zhu, Xiaoye Qu, Daize Dong, et al. «LLaMA-MoE: Building Mixture-of-Experts from LLaMA with Continual Pre-training». In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP 2024)*. arXiv:2406.16554. 2024, pp. 15913–15923 (cit. on p. 12).

[27] Xiaoye Qu, Daize Dong, Xuyang Hu, Tong Zhu, Weigao Sun, and Yu Cheng. *LLaMA-MoE v2: Exploring Sparsity of LLaMA from Perspective of Mixture-of-Experts with Post-training*. arXiv:2411.15708. Preprint. 2024 (cit. on p. 12).

[28] Ruisi Cai, Yeonju Ro, Geon-Woo Kim, et al. «READ-ME: Refactorizing LLMs as Router-Decoupled Mixture of Experts with System Co-design». In: *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*. arXiv:2410.19123. 2024, pp. 116126–116148 (cit. on p. 12).

[29] Uranik Berisha, Jens Mehnert, and Alexandru Paul Condurache. *Efficient Data Driven Mixture-of-Expert Extraction from Trained Networks*. arXiv:2505.15414. Preprint. 2025 (cit. on pp. 12, 14).

[30] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. «ImageNet: A Large-Scale Hierarchical Image Database». In: *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009, pp. 248–255 (cit. on p. 12).

[31] Junjie Wen, Yichen Zhu, Jinming Li, et al. *TinyVLA: Towards Fast, Data-Efficient Vision-Language-Action Models for Robotic Manipulation*. arXiv:2409.12514. Preprint; presented at IROS 2025 / IEEE RA-L. 2024 (cit. on p. 13).

[32] Abby O'Neill, Abdul Rehman, Abhinav Gupta, et al. *Open X-Embodiment: Robotic Learning Datasets and RT-X Models*. arXiv:2310.08864. ICRA 2024. 2023 (cit. on p. 13).

[33] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, et al. «OpenVLA: An Open-Source Vision-Language-Action Model». In: *Proceedings of The 8th Conference on Robot Learning (CoRL 2024)*. Vol. 270. Proceedings of Machine Learning Research. arXiv:2406.09246. 2024, pp. 2679–2713 (cit. on p. 13).

[34] Cheng Chi, Zhenjia Xu, Siyuan Feng, et al. *Diffusion Policy: Visuomotor Policy Learning via Action Diffusion*. arXiv:2303.04137. RSS 2023. 2023 (cit. on p. 13).

[35] Tianhe Yu, Deirdre Quillen, Zhanpeng He, et al. «Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning». In: *Conference on Robot Learning (CoRL 2019)*. arXiv:1910.10897. 2019 (cit. on p. 13).

[36] Weijie Shen, Yitian Liu, Yuhao Wu, et al. *Expertise Need Not Monopolize: Action-Specialized Mixture of Experts for Vision-Language-Action Learning*. arXiv:2510.14300. Preprint. 2025 (cit. on p. 13).

[37] Yao Mu, Tianxing Chen, Zanxin Chen, et al. *RoboTwin: Dual-Arm Robot Benchmark with Generative Digital Twins*. arXiv:2504.13059. CVPR 2025 Highlight. 2025 (cit. on p. 14).

[38] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. «A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning». In: *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*. Vol. 15. PMLR. arXiv:1011.0686. 2011, pp. 627–635 (cit. on pp. 15, 42).

[39] Yoon Kim and Alexander M. Rush. «Sequence-Level Knowledge Distillation». In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP 2016)*. arXiv:1606.07947. 2016, pp. 1317–1327 (cit. on pp. 19, 39).

[40] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. arXiv:1503.02531. NIPS 2014 Deep Learning Workshop. 2015 (cit. on pp. 19, 39).

[41] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. «Sequence Level Training with Recurrent Neural Networks». In: *4th International Conference on Learning Representations (ICLR 2016)*. arXiv:1511.06732 Coins the term *exposure bias*. 2016 (cit. on p. 38).