ONE MODEL, MULTIPLE COMPUTE BUDGETS: MULTI-RATIO CONCEPTMOE FOR FLEXIBLE COMPUTE ACROSS TURNS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models typically support different inference budgets through separate model sizes or fixed compute configurations. In multi-turn interactions, however, compute demand can vary substantially across turns. Using a high-compute model for every turn wastes computation, while routing among independent models may require rebuilding model-specific history states after switches. We therefore ask whether a single shared model can support multiple compute budgets that can be flexibly assigned across turns. We introduce **Multi-Ratio ConceptMoE**, which jointly supports token-to-concept compression ratios of 2, 4, and 8 (CR2/CR4/CR8) within a shared backbone. Lightweight full-token modules reduce compression-independent cost, ratio-specific boundary routers support distinct compression levels, and cross-ratio training enables mixed-ratio processing within the same context. Across PTBench and SFTBench, the three fixed modes outperform their compute-matched MoE baselines in all **6** comparisons while remaining competitive with independently trained fixed-ratio specialists. On CodeChat and CoSQL, mixed-ratio inference improves the accuracy–compute trade-off, outperforming the Large MoE by **5.87** and **2.68** percentage points while reducing FLOPs by **41.1%** and **12.3%**, respectively. Under the same turn-wise budget schedule, Multi-Ratio ConceptMoE further reduces FLOPs by **78.4%** and **46.9%** relative to routing among independent models. These results demonstrate that a single shared model can provide multiple compute budgets for flexible compute allocation across turns.

1 INTRODUCTION

Large language models exhibit a natural trade-off between prediction quality and inference cost Chen et al. (2023). In real-world multi-turn interactions, however, the computational workload is intrinsically non-uniform across conversational turns. An initial request often demands intense computation for complex reasoning or code generation, whereas subsequent turns frequently involve lightweight edits, local refinements, or follow-up clarifications Schuster et al. (2022); Raposo et al. (2024). This dynamic nature of human-AI interaction calls for a paradigm shift from static inference to turn-wise adaptive compute allocation.

A natural strategy is to route requests among separately deployed Large, Medium, and Small models Ong et al. (2025). However, independent models maintain model-specific key-value (KV) states that cannot be directly reused across models. Switching models during a stateful interaction can therefore require rebuilding the target model's history state from the preceding context (Fig. 1(a)). As the dialogue context grows, this additional history processing can offset part of the compute saved by routing to a smaller model. This motivates a compelling question: *Can we build a single shared model that supports multiple compute budgets that can be flexibly assigned across turns?*

In this paper, we answer this question with token-to-concept compression, providing a natural basis for exposing different compute budgets within a single model. Recent concept models such as DLCM Qu et al. (2025) and ConceptMoE Huang et al. (2026) compress consecutive token representations into a shorter sequence of semantic concepts, allowing the main backbone to process fewer representations. In ConceptMoE, the target compression ratio directly controls the overall compute

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

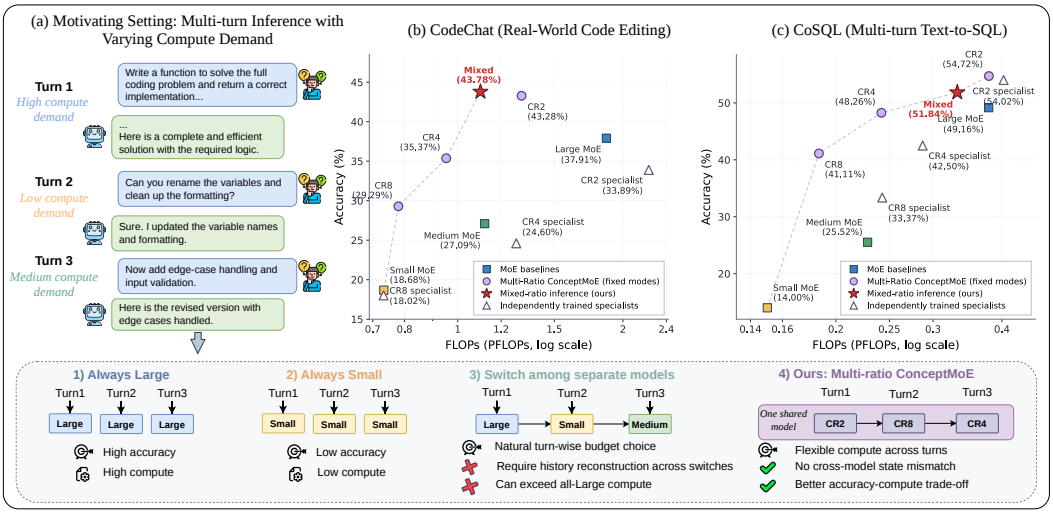

Figure 1: **Motivation and accuracy–compute trade-offs of Multi-Ratio ConceptMoE. (a)** Multi-turn interactions naturally exhibit varying compute demands across turns. Using a fixed Large or Small model leads to an unfavorable accuracy–compute trade-off, while switching among separately deployed models may require rebuilding model-specific history states. Multi-Ratio ConceptMoE instead supports different compute budgets within a single shared model. **(b,c)** On CodeChat and CoSQL, mixed-ratio inference lies on the accuracy–compute Pareto frontier of the compared fixed-ratio ConceptMoE modes, independently trained fixed-ratio specialists, and MoE baselines.

budget by changing the number of concept representations processed by the backbone. Existing fixed-ratio models, however, support only one such budget after training. We extend this setting to multiple compression-based budgets that can be used across interaction turns.

We introduce **Multi-Ratio ConceptMoE**, which supports multiple compute budgets within a single shared model. The design addresses two challenges. First, the token-level encoder and decoder operate at full token resolution, so their computation does not decrease proportionally as the compression ratio increases and can create a substantial compression-independent compute floor. We therefore make the full-token modules lightweight and concentrate more model capacity in the compression-scaled concept backbone, widening the effective compute range across compression ratios. Second, strong performance at each individual compression ratio does not necessarily translate to effective mixed-ratio inference. Multi-Ratio ConceptMoE uses lightweight ratio-specific boundary routers to jointly support CR2, CR4, and CR8 within the shared backbone. We further introduce **cross-ratio training**, which applies different compression ratios within the same training sequence so that the shared model learns to handle mixed-ratio representations. Together, these designs allow one checkpoint to provide multiple fixed operating points and mixed-ratio inference across turns.

Our experiments show that Multi-Ratio ConceptMoE provides both strong fixed operating points and improved accuracy–compute trade-offs. Across PTBench and SFTBench evaluation suites (see Appendix B.1 for benchmark composition), its CR2, CR4, and CR8 modes outperform the corresponding compute-matched Large, Medium, and Small MoE baselines in all 6 comparisons, while remaining competitive with independently trained fixed-ratio specialists.

More importantly, mixed-ratio inference improves the accuracy–compute trade-off on both multi-turn tasks (Fig. 1(b,c)). Compared with the Large MoE, it improves accuracy by **5.87** and **2.68** percentage points while reducing FLOPs by **41.1%** and **12.3%** on CodeChat and CoSQL, respectively; on CodeChat, it also strictly dominates the fixed CR2 mode. Furthermore, under the same turn-wise compute-budget schedule, the shared Multi-Ratio model reduces FLOPs by **78.4%** on CodeChat and **46.9%** on CoSQL compared with routing among independent models, largely by avoiding repeated history re-prefill after model switches.

Contributions. Our main contributions are summarized as follows:

- **Single-model flexible compute for multi-turn inference.** We study a shared-model approach to turn-wise compute allocation, where a single model supports multiple compute budgets that can be flexibly used across turns.
- **Multi-Ratio ConceptMoE.** We extend token-to-concept compression to multiple compute budgets within a shared model, combining lightweight full-token processing, ratio-specific boundary routers, and cross-ratio training to support mixed-ratio inference.
- **Improved accuracy–compute trade-offs.** Multi-Ratio ConceptMoE provides strong fixed operating points across compute budgets, while mixed-ratio inference further improves the accuracy–compute trade-off on CodeChat and CoSQL. Under the same budget schedule, the shared model also avoids substantial history re-prefill overhead incurred by routing among independent models.

2 Related Work

2.1 Adaptive and Multi-Budget Computation

Prior work has explored several ways to reduce or adapt inference computation, including model-level routing or cascading across models of different sizes Chen et al. (2023); Ong et al. (2025), as well as single-model approaches based on early exiting, dynamic depth, elastic subnetworks, or adaptive MoE activation Elbayad et al. (2019); Raposo et al. (2024); Devvrit et al. (2024); Cai et al. (2024); Zeng et al. (2024); Gu et al. (2026); Zhou et al. (2026). These approaches adapt computation by changing executed layers, subnetworks, or expert activation.

Multi-Ratio ConceptMoE controls computation through representation granularity: the shared backbone processes a variable number of concept representations determined by the token-to-concept compression ratio. Different ratios provide distinct accuracy–compute operating points while sharing the same model components. We further study how these compression levels can be used within the same context and changed across interaction turns.

2.2 Latent Concept Models and Token-to-Concept Compression

The most closely related work focuses on latent concept models. **Dynamic Large Concept Models (DLCM)** learns latent semantic boundaries and compresses variable-length token spans into concepts, shifting more computation from token space to a compact concept space Qu et al. (2025). Its compression-aware scaling analysis further studies how computation can be allocated among compression ratio, token-level capacity, and concept-level reasoning capacity.

Subsequently, **ConceptMoE** Huang et al. (2026) introduces a learnable chunking module that predicts content-adaptive concept boundaries under a target compression ratio. ConceptMoE further combines token-to-concept compression with a Mixture-of-Experts backbone to control total parameters and activated FLOPs, enabling a more direct comparison between concept-level and conventional token-level computation.

Our work builds directly on this token-to-concept compression paradigm and further considers the joint support and composition of multiple compression budgets. Existing concept models typically operate under a predefined overall compression budget, whereas Multi-Ratio ConceptMoE exposes multiple compression-based compute operating points within a single shared model. We jointly train multiple target compression ratios and use cross-ratio training to expose the shared backbone to mixed-granularity concept representations. In this way, concept models are extended to support multiple compute operating points, allowing us to study how these modes can be composed within the same context and across interaction turns, as well as the resulting accuracy–compute trade-offs.

2.3 Efficient Multi-Turn Inference

An important challenge in multi-turn inference is efficiently processing an ever-growing interaction history. Prior work has addressed this problem through context compression, KV-cache compression and quantization, cache selection or eviction, and more efficient long-sequence attention Rae et al. (2019); Beltagy et al. (2020); Zhang et al. (2023); Li et al. (2024b); Liu et al. (2024); Kwon et al. (2023). These approaches mainly reduce the cost of representing, storing, or attending to history.

Another line of work adapts inference compute through model routing or cascading, selecting among models of different capacities according to the input or task Chen et al. (2023); Ong et al. (2025). Such approaches provide multiple compute levels through separate model instances. In stateful multi-turn serving, however, cached representations are model-specific, so switching between models may require reconstructing the history state for the newly selected model.

Our work complements these directions by realizing multiple compression-based compute budgets within a single shared model, allowing different budgets to be used across turns without switching between separate model instances.

3 Approach

Our design extends ConceptMoE along two dimensions: widening the effective compute range across compression ratios and enabling mixed-ratio processing. The first requires reducing computation that remains at full token resolution, while the second requires the shared backbone to handle representations produced under different compression ratios. We address these issues with lightweight full-token modules, ratio-specific boundary routing, and cross-ratio training. Figure 2 provides an overview of the resulting architecture and training scheme.

3.1 Preliminaries: Token-to-Concept Compression

We briefly review the token-to-concept compression mechanism of ConceptMoE Huang et al. (2026). Given a sequence of N token representations, a token-level encoder $\mathcal{E}$ produces contextualized representations while preserving the sequence length:

$$X = [x_1, \ldots, x_N] \in \mathbb{R}^{N \times d_e}, \qquad H = \mathcal{E}(X) = [h_1, \ldots, h_N] \in \mathbb{R}^{N \times d_e}. \tag{1}$$

The Chunk module then predicts content-adaptive concept boundaries from H. Let $b_i \in \{0, 1\}$ indicate whether position i starts a new concept. These boundary decisions partition the original sequence into M contiguous token chunks, denoted by $S_1, \ldots, S_M$. Each chunk is compressed into one concept representation, producing

$$C = [c_1, \ldots, c_M] \in \mathbb{R}^{M \times d_c}, \qquad M \leq N. \tag{2}$$

When $d_e \neq d_c$, learned linear projections map the chunked representations to width d_c before concept-level processing and back to width d_e after DeChunk. The concept backbone $\mathcal{C}$ processes the compressed sequence, producing $\hat{C} = \mathcal{C}(C) \in \mathbb{R}^{M \times d_c}$. DeChunk then restores token resolution, and the token-level decoder $\mathcal{D}$ processes the resulting length-N representation. The overall computation is summarized as

$$X \xrightarrow{\mathcal{E}} H \xrightarrow{\text{Chunk}} C \xrightarrow{\mathcal{C}} \hat{C} \xrightarrow{\text{DeChunk}} Z \xrightarrow{\mathcal{D}} \hat{Z}. \tag{3}$$

ConceptMoE controls the average compression level through a target ratio r, encouraging $M \approx N/r$. Since chunk boundaries are content-adaptive, the realized ratio $\tilde{r} = N/M$ varies across inputs. Ignoring the relatively small Chunk/DeChunk overhead and attention-map computation, the total computation can be approximated as

$$F(r) \approx F_{\text{full}} + \frac{F_{\text{concept}}}{r}, \tag{4}$$

where F_{full} denotes full-token computation in the encoder and decoder, and F_{concept} denotes the uncompressed computation of the concept backbone. As r increases, concept-level computation decreases approximately as $1/r$, while F_{full} remains unchanged.

3.2 Multi-Ratio ConceptMoE Architecture

To widen the compute range controlled by compression, we first reduce the compression-independent cost of full-token processing. We keep the token-level encoder and decoder lightweight and concentrate most of the model's capacity in the compression-scaled concept backbone.

Specifically, the encoder and decoder retain moderate depth but use a hidden width of approximately two-thirds of the concept-backbone width. This reduces F_{full} in Eq. 4, allowing different compression ratios to produce more clearly separated compute budgets.

Multi-Ratio ConceptMoE jointly supports three target compression ratios. For each ratio, the Chunk module contains a lightweight, ratio-specific boundary router g_r:

$$\mathcal{R} = \{2, 4, 8\},$$
$$\left(P^{(r)}, B^{(r)}\right) = g_r(H), \qquad r \in \mathcal{R}. \tag{5}$$

where $P^{(r)} \in [0,1]^N$ contains the boundary probabilities and $B^{(r)} \in \{0,1\}^N$ the corresponding boundary decisions. All routers operate on the same shared encoder representation H, but they are trained toward different target compression ratios. Only the boundary routers are ratio-specific; the encoder, concept backbone, DeChunk module, and decoder are shared across all ratios.

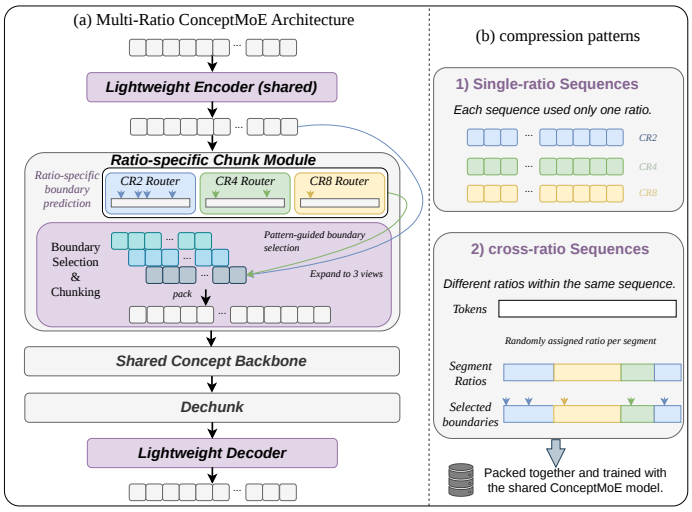

Figure 2: **Overview of Multi-Ratio ConceptMoE. (a)** A shared encoder supports ratio-specific boundary prediction and multiple compression views, which are packed for shared concept-level processing. **(b)** Cross-ratio patterns assign different compression ratios to contiguous segments, with segment starts forced as chunk boundaries.

During training, each input is encoded once, and the three routers produce candidate boundary sequences from the same representation H. We then construct three compression views. For view v, its per-token ratio assignment is represented as

$$A^{(v)} = \left(a_1^{(v)}, \ldots, a_N^{(v)}\right) \in \mathcal{R}^N, \tag{6}$$

where $a_i^{(v)}$ specifies the compression ratio assigned to position i. At each position, we select the candidate boundary decision from the router corresponding to the assigned ratio:

$$\bar{B}_i^{(v)} = B_i^{\left(a_i^{(v)}\right)}. \tag{7}$$

These candidate boundaries are combined with the sampled segment boundaries to obtain the final chunk boundaries, as described in Section 3.3. Figure 2(a) summarizes the shared architecture and ratio-specific boundary selection.

3.3 Cross-Ratio Compression Training

We train Multi-Ratio ConceptMoE with cross-ratio compression patterns that use different compression ratios within the same sequence.

Fixed-ratio training exposes the shared backbone only to concept sequences formed under a uniform compression level. Mixed-ratio inference creates a different setting: concepts with different local compression granularities can coexist within the same context. The backbone must therefore process heterogeneous concept densities while preserving coherent contextual representations across ratio transitions. Cross-ratio training explicitly exposes the model to this setting during training, rather than relying on independently learned fixed-ratio behaviors to combine at inference time.

A single-ratio pattern assigns one compression ratio throughout the sequence, i.e., $a_i^{(v)} = r$ for all positions. In contrast, a cross-ratio pattern partitions the sequence into contiguous segments $\mathcal{I}_1, \ldots, \mathcal{I}_K$ and independently samples a ratio for each segment:

$$r_k \sim \text{Uniform}(\mathcal{R}),$$
$$a_i^{(v)} = r_k, \qquad i \in \mathcal{I}_k. \tag{8}$$

The resulting assignment is piecewise constant, as illustrated in Figure 2(b).

To prevent concepts from spanning sampled segments, we force a chunk boundary at every segment start. Let $s_i^{(v)} \in \{0, 1\}$ indicate whether position i starts a segment. The final boundary decisions are

$$B_i^{(v)} = \bar{B}_i^{(v)} \vee s_i^{(v)}, \qquad i = 1, \ldots, N. \tag{9}$$

All other boundaries within each segment remain content-adaptive and are predicted by the corresponding ratio-specific router. The final boundaries are used to construct $C^{(v)} = \mathrm{Chunk}(H, B^{(v)})$. The resulting concept sequences are packed for joint processing by the shared concept backbone while preserving their sequence boundaries. After concept-level processing, the outputs are separated by view, mapped back to token resolution through DeChunk, and processed by the shared decoder.

In our final configuration, all three compression views, including their segmentations and ratio assignments, are independently sampled from the cross-ratio pattern distribution:

$$A^{(1)}, A^{(2)}, A^{(3)} \overset{\text{i.i.d.}}{\sim} \mathcal{P}_{\text{cross}}. \tag{10}$$

We introduce no explicit cross-ratio distillation, representation alignment, or consistency objective. Section 4.5 studies alternative mixtures of single-ratio and cross-ratio training views. The sampled training segments are not intended to reproduce a particular turn-level routing distribution. Instead, they expose the shared backbone to ratio transitions and heterogeneous concept densities within the same context, matching the setting encountered during mixed-ratio inference.

At inference time, an external policy selects a ratio-specific router for each segment, enabling fixed- or mixed-ratio operation. In our multi-turn setting, each newly appended turn is treated as a segment and assigned one compression ratio by the external policy. Previously processed turns retain their original compression ratios, so a ratio change affects only the newly appended context. Each new segment starts at a forced chunk boundary, so changing its ratio does not repartition previously constructed concepts. Since all ratios share the same concept backbone, previously computed history states remain valid across ratio changes. This allows the compute budget to change at turn boundaries while preserving the representation state accumulated from earlier turns. The same mechanism allows different compression levels to coexist within a single growing context while preserving previously constructed representations.

4 Experiments

4.1 Experimental Setup

Models and evaluation. We evaluate Multi-Ratio ConceptMoE at three fixed operating points, CR2, CR4, and CR8, against compute-matched Large, Medium, and Small MoE baselines, as well as independently trained SingleCR specialists. General capability is evaluated with PTBench on the PT checkpoints and SFTBench after supervised fine-tuning. We further evaluate mixed-ratio inference on two multi-turn tasks, CodeChat Guo et al. (2026) and CoSQL Yu et al. (2019). For Pareto analysis, we use Combined Joint accuracy for CodeChat and SQLite execution accuracy for CoSQL. Dataset composition, evaluation splits, prompting details, and full evaluation protocols are provided in Appendix B. For mixed-ratio evaluation, turn-wise CR assignments are generated by task-specific routing prompts and frozen before model evaluation. The routing policy does not observe model outputs, correctness signals, or future turns. The same frozen schedules are used for the routed independent-model baseline and the cross-ratio training ablations; full routing protocols are provided in Appendix B.5. The reported FLOPs exclude the cost of the external routing policy and characterize model execution under the frozen turn-wise schedules. Our evaluation therefore focuses on model execution under a fixed external compute-allocation schedule.

Compute accounting. Following the compute-matching setup of ConceptMoE Huang et al. (2026), the Large, Medium, and Small MoE baselines are configured to approximately match the activated FLOPs of CR2, CR4, and CR8, respectively, excluding attention-map computation, whose cost depends on sequence length and therefore varies with the compression ratio. For multi-turn evaluations, we report workload-realized FLOPs under a shared terminal-turn workload, including

Table 1: **Against compute-matched MoE baselines.** Each cell reports MoE / Multi-Ratio accuracy (%). Multi-Ratio ConceptMoE wins all 6 comparisons.

Budget	PTBench	SFTBench
High	31.7 / **32.1**	19.0 / **21.5**
Medium	28.2 / **29.3**	15.5 / **19.1**
Low	25.4 / **27.6**	13.1 / **17.2**

Table 2: **Against independently trained specialists.** Each cell reports SingleCR / Multi-Ratio accuracy (%).

Ratio	PTBench	SFTBench
CR2	**33.2** / 32.1	21.1 / **21.5**
CR4	**30.0** / 29.3	18.8 / **19.1**
CR8	25.8 / **27.6**	16.4 / **17.2**
Avg.	29.7 / 29.7	18.8 / **19.3**

Table 3: **Accuracy–compute trade-offs on CodeChat and CoSQL.** CodeChat reports Combined Joint accuracy, while CoSQL reports SQLite execution accuracy. FLOPs include attention-map computation and use the terminal-turn workload protocol.

	CodeChat		CoSQL	
Model	Acc. (%)	FLOPs (PF)	Acc. (%)	FLOPs (PF)
MoE Large	37.91	1.8662	49.16	0.3785
MoE Medium	27.09	1.1198	25.52	0.2283
MoE Small	18.68	0.7334	14.00	0.1502
Multi-Ratio CR2	43.28	1.3072	**54.72**	0.3788
Multi-Ratio CR4	35.37	0.9529	48.26	0.2418
Multi-Ratio CR8	29.29	0.7795	41.11	0.1863
Multi-Ratio Mixed	**43.78**	1.0999	51.84	0.3319
SingleCR2	33.89	2.2290	54.02	0.4027
SingleCR4	24.60	1.2784	42.50	0.2873
SingleCR8	18.02	0.7326	33.37	0.2427

attention-map and sliding-window-attention computation. The same prompt and output-token workload is used across models, while ConceptMoE FLOPs are computed using the realized prefill and decode compression ratios. Full accounting details are provided in Appendix C.

4.2 Strong Fixed-Ratio Operating Points

We first examine whether a single Multi-Ratio checkpoint can provide strong standalone operating points across compute budgets. As shown in Table 1, CR2, CR4, and CR8 all outperform their corresponding Large, Medium, and Small MoE baselines on both PTBench and SFTBench, yielding **6 wins out of 6 comparisons**. The gains span all three evaluated compute budgets, showing that the shared checkpoint retains strong fixed-ratio performance across the evaluated compute range.

We further compare Multi-Ratio ConceptMoE with independently trained SingleCR specialists. Although the specialists retain small advantages on several PT evaluations, the shared model remains competitive overall. Averaged across CR2/CR4/CR8, Multi-Ratio matches the independently trained specialists on PTBench at 29.7 and improves SFTBench from 18.8 to 19.3. The advantage is particularly clear at CR8, where Multi-Ratio outperforms the corresponding specialist on both evaluation suites. These results indicate that supporting multiple compression modes in one shared model does not introduce a substantial overall specialization penalty.

4.3 Mixed-Ratio Inference Improves the Accuracy–Compute Trade-off

We next evaluate whether composing different compression modes across turns can improve the accuracy–compute trade-off. Figure 1(b,c) and Table 3 show that Mixed lies on the accuracy–compute Pareto frontier of the configurations on both CodeChat and CoSQL. The comparison includes MoE baselines, fixed CR2/CR4/CR8 modes, and independently trained SingleCR specialists.

On CodeChat, Mixed achieves **43.78%** accuracy at **1.0999 PFLOPs**, outperforming the Large MoE by 5.87 points while using 41.1% fewer FLOPs, and strictly dominating fixed CR2. On CoSQL,

Table 4: **Switching overhead under the same turn-wise compute-budget schedule.** For the routed independent-model baseline, total FLOPs include the additional history re-prefill required after model switches. The comparison quantifies the algorithmic compute overhead of switching among independent models.

Dataset	Switches	Re-prefilled Tokens	Routed FLOPs (PF)	Multi-Ratio FLOPs (PF)	Reduction
CodeChat	1,925	2,882,580	5.0895	1.0999	78.4%
CoSQL	512	320,108	0.6250	0.3319	46.9%

Table 5: **Effect of cross-ratio training exposure.** We compare homogeneous-only training (3 single-ratio views), mixed training (1 single-ratio + 2 cross-ratio views), and fully cross-ratio training (3 cross-ratio views). PTBench and SFTBench report average fixed-ratio accuracy across CR2/CR4/CR8, while CodeChat and CoSQL report mixed-ratio inference accuracy.

Training composition	Fixed-Ratio Evaluation		Mixed-Ratio Inference	
	PTBench	SFTBench	CodeChat	CoSQL
3 Single + 0 Cross	**29.97**	18.83	37.65	43.00
1 Single + 2 Cross	**29.97**	18.73	37.57	45.88
0 Single + 3 Cross	29.67	**19.27**	**43.78**	**51.84**

Mixed reaches **51.84%** execution accuracy at **0.3319 PFLOPs**, improving over the Large MoE by 2.68 points with 12.3% fewer FLOPs. Compared with fixed CR2, it sacrifices only 2.88 points of accuracy while reducing compute by 12.4%. Thus, mixed-ratio inference yields accuracy–compute operating points beyond the evaluated fixed-budget alternatives on both tasks.

4.4 Switching Overhead under the Same Budget Schedule

A natural alternative to mixed-ratio inference is to route each turn among separately deployed Large, Medium, and Small models. To quantify the compute overhead associated with model switching, we apply exactly the same frozen turn-wise budget schedule used by Multi-Ratio ConceptMoE (Appendix B.5), mapping CR2/CR4/CR8 to Large/Medium/Small, respectively.

Both approaches use the same terminal-turn workload convention. Multi-Ratio ConceptMoE changes compression modes within one shared model, whereas independently deployed models maintain model-specific history states. We therefore account for the history re-prefill required to reconstruct the state of the newly selected independent model after each switch. This accounting assumes on-demand state reconstruction for the selected independent model. Alternatively, retaining or continuously updating separate history states for all candidate models would trade re-prefill cost for additional memory or computation.

Figure 3 visualizes the resulting overhead, and Table 4 reports the corresponding FLOPs. On CodeChat, the frozen schedule contains 1,925 model switches and requires approximately 2.88M history tokens to be re-prefilled. This adds 3.5365 PFLOPs of history re-prefill to the routed baseline, increasing its total computation from 1.5530 to 5.0895 PFLOPs. Multi-Ratio Mixed uses 1.0999 PFLOPs in total, 78.4% lower than the resulting routed total. On CoSQL, 512 switches require approximately 320K re-prefilled history tokens, raising the routed baseline from 0.2830 to 0.6250 PFLOPs, while Multi-Ratio Mixed uses 0.3319 PFLOPs, corresponding to a 46.9% reduction.

This comparison is intended to quantify the algorithmic compute overhead introduced by switching among independent models.

4.5 Effect of Cross-Ratio Training Exposure

Finally, we study whether cross-ratio training exposure is important for composing different compression modes. We compare three training compositions: three single-ratio views, one single-ratio plus two cross-ratio views, and the fully cross-ratio configuration with three cross-ratio views. Fig-

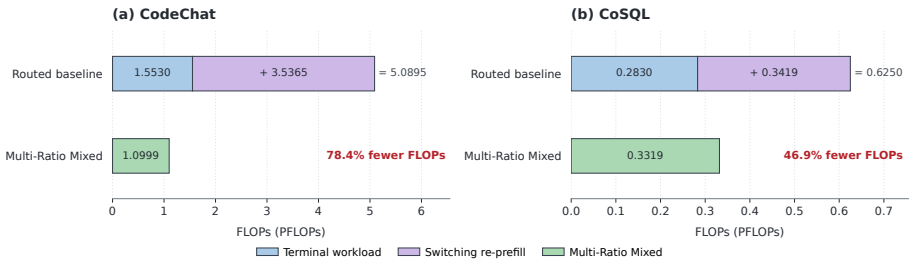

Figure 3: **FLOPs breakdown for turn-wise compute allocation.** Both approaches follow the same turn-wise compute-budget schedule. Routing among independent models incurs additional history re-prefill whenever the selected model changes, whereas Multi-Ratio ConceptMoE changes compression modes within a shared model.

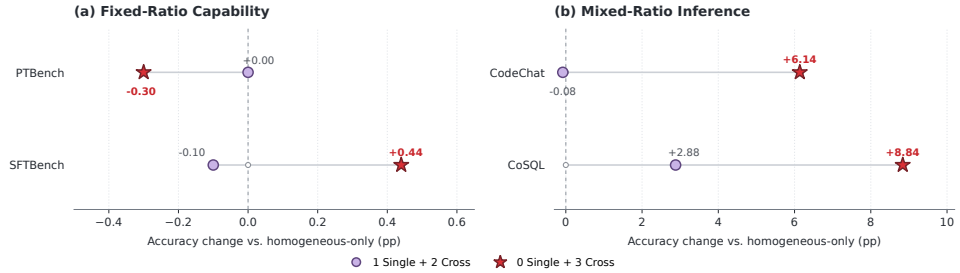

Figure 4: **Effect of cross-ratio training exposure.** Accuracy changes are measured relative to homogeneous-only training (3 Single + 0 Cross). Cross-ratio exposure largely preserves fixed-ratio capability while fully cross-ratio training substantially improves mixed-ratio inference on both tasks.

ure 4 visualizes the accuracy changes relative to homogeneous-only training, while Table 5 reports the corresponding absolute scores.

For fixed-ratio general capability, the effect of cross-ratio exposure is relatively small. Compared with homogeneous-only training, the fully cross-ratio configuration changes PTBench by -0.30 points, while improving SFTBench by 0.44 points. The intermediate configuration with one single-ratio and two cross-ratio views likewise remains close to the homogeneous baseline. Thus, exposing all training views to cross-ratio patterns does not cause a substantial overall loss in the standalone utility of the fixed operating points.

The effect is much larger for mixed-ratio inference. Relative to homogeneous-only training, fully cross-ratio training improves CodeChat by **6.14** points and CoSQL by **8.84** points, while the corresponding inference FLOPs remain comparable. Overall, fully cross-ratio training largely preserves fixed-ratio capability and slightly improves it after SFT, while substantially strengthening the ability to compose different compression modes. We therefore adopt the fully cross-ratio configuration.

5 Conclusion

We study flexible compute allocation across turns without switching among separately deployed models. To this end, we introduce Multi-Ratio ConceptMoE, which exposes multiple compression-based operating points within a shared model. By keeping full-token processing lightweight, using ratio-specific boundary routing, and training with cross-ratio compression patterns, a single checkpoint can operate at fixed CR2/CR4/CR8 budgets or compose these modes across turns. Across general evaluations, the fixed modes consistently outperform their compute-matched MoE baselines, while remaining competitive with independently trained fixed-ratio specialists. On CodeChat and CoSQL, mixed-ratio inference improves the accuracy–compute trade-off and avoids the cross-model state reconstruction overhead accounted for in our routed baseline. These results show that a shared model can flexibly allocate compute across turns without cross-model switching.

AI USE STATEMENT

Generative AI tools were used only to assist with language editing and polishing of the manuscript. The technical content, methodology, experiments, analyses, and conclusions were developed and verified by the authors.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Ruisi Cai, Saurav Muralidharan, Greg Heinrich, Hongxu Yin, Zhangyang Wang, Jan Kautz, and Pavlo Molchanov. Flextron: Many-in-one flexible large language model. *arXiv preprint arXiv:2406.10260*, 2024.

Linzheng Chai, Shukai Liu, Jian Yang, Yuwei Yin, Jiaheng Liu, Tao Sun, Ge Zhang, Changyu Ren, Hongcheng Guo, Zekun Wang, et al. Mceval: Massively multilingual code evaluation. In *International Conference on Learning Representations*, volume 2025, pp. 87704–87730, 2025.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

François Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

Kaustubh Deshpande, Ved Sirdeshmukh, Johannes Baptist Mols, Lifeng Jin, Ed-Yeremai Hernandez-Cardona, Dean Lee, Jeremy Kritz, Willow E Primack, Summer Yue, and Chen Xing. Multichallenge: A realistic multi-turn conversation evaluation benchmark challenging to frontier llms. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 18632–18702, 2025.

Fnu Devvrit, Sneha Kudugunta, Aditya Kusupati, Tim Dettmers, Kaifeng Chen, Inderjit S Dhillon, Yulia Tsvetkov, Hannaneh Hajishirzi, Sham M Kakade, Ali Farhadi, et al. Matformer: Nested transformer for elastic inference. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Xeron Du, Yifan Yao, Kaijing Ma, Bingli Wang, Tianyu Zheng, Minghao Liu, Yiming Liang, Xiaolong Jin, Zhenlin Wei, Chujie Zheng, et al. Supergpqa: Scaling llm evaluation across 285 graduate disciplines. *Advances in Neural Information Processing Systems*, 38, 2026.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2368–2378, 2019.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. *arXiv preprint arXiv:1910.10073*, 2019.

Ippei Fujisawa, Sensho Nobe, Hiroki Seto, Rina Onda, Yoshiaki Uchida, Hiroki Ikoma, Pei-Chun Chien, and Ryota Kanai. Procbench: Benchmark for multi-step reasoning and following procedure. *arXiv preprint arXiv:2410.03117*, 2024.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Zhengyang Tang, et al. Omni-math: A universal olympiad level mathematic benchmark for large language models. In *International Conference on Learning Representations*, volume 2025, pp. 100540–100569, 2025.

Naibin Gu, Zhenyu Zhang, Yuchen Feng, Yilong Chen, Peng Fu, Zheng Lin, Shuohuan Wang, Yu Sun, Hua Wu, Weiping Wang, and Haifeng Wang. Elastic moe: Unlocking the inference-time scalability of mixture-of-experts, 2026. URL https://arxiv.org/abs/2509.21892.

Guoxiang Guo, Kla Tantithamthavorn, Neelofar Neelofar, Yuanyuan Qi, and Aldeida Aleti. Codechat-eval: Evaluating large language models in multi-turn code refinement dialogues. *arXiv preprint arXiv:2606.25747*, 2026.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, 2024.

Yancheng He, Shilong Li, Jiaheng Liu, Yingshui Tan, Weixun Wang, Hui Huang, Xingyuan Bu, Hangyu Guo, Chengwei Hu, Boren Zheng, et al. Chinese simpleqa: A chinese factuality evaluation for large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19182–19208, 2025.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Yao Fu, et al. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *Advances in neural information processing systems*, 36:62991–63010, 2023.

Zihao Huang, Jundong Zhou, Xingwei Qu, Qiyang Min, and Ge Zhang. Conceptmoe: Adaptive token-to-concept compression for implicit compute allocation. *CoRR*, abs/2601.21420, 2026. doi: 10.48550/ARXIV.2601.21420. URL https://doi.org/10.48550/arXiv.2601.21420.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, 2017.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E Gonzalez, and Ion Stoica. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. *arXiv preprint arXiv:2406.11939*, 2024a.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024b.

Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. Zebralogic: On the scaling limits of llms for logical reasoning. *arXiv preprint arXiv:2502.01100*, 2025.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in neural information processing systems*, 36:21558–21572, 2023.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*, 2024.

Kaijing Ma, Xeron Du, Yunran Wang, Haoran Zhang, Zhoufutu Wen, Xingwei Qu, Jian Yang, Jiaheng Liu, Minghao Liu, Xiang Yue, et al. Kor-bench: Benchmarking language models on knowledge-orthogonal reasoning tasks. In *International Conference on Learning Representations*, volume 2025, pp. 80062–80161, 2025.

Jinjie Ni, Fuzhao Xue, Xiang Yue, Yuntian Deng, Mahir Shah, Kabir Jain, Graham Neubig, and Yang You. Mixeval: Deriving wisdom of the crowd from llm benchmark mixtures. *Advances in Neural Information Processing Systems*, 37:98180–98212, 2024.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, Mohammed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *International Conference on Learning Representations*, volume 2025, pp. 34433–34448, 2025.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

Xingwei Qu, Shaowen Wang, Zihao Huang, Kai Hua, Fan Yin, Rui-Jie Zhu, Jundong Zhou, Qiyang Min, Zihao Wang, Yizhi Li, Tianyu Zhang, He Xing, Zheng Zhang, Yuxuan Song, Tianyu Zheng, Zhiyuan Zeng, Chenghua Lin, Ge Zhang, and Wenhao Huang. Dynamic large concept models: Latent reasoning in an adaptive semantic space. *CoRR*, abs/2512.24617, 2025. doi: 10.48550/ARXIV.2512.24617. URL https://doi.org/10.48550/arXiv.2512.24617.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472, 2022.

Shivalika Singh, Angelika Romanou, Clémentine Fourrier, David Ifeoluwa Adelani, Jian Gang Ngui, Daniel Vila-Suero, Peerat Limkonchotiwat, Kelly Marchisio, Wei Qi Leong, Yosephine Susanto, et al. Global mmlu: Understanding and addressing cultural and linguistic biases in multilingual evaluation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 18761–18799, 2025.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, 2023.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.

Jason Wei, Nguyen Karina, Hyung Won Chung, Yunxin Joy Jiao, Spencer Papay, Amelia Glaese, John Schulman, and William Fedus. Measuring short-form factuality in large language models. *arXiv preprint arXiv:2411.04368*, 2024.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, et al. Livebench: A challenging, contamination-limited llm benchmark. In *International Conference on Learning Representations*, volume 2025, pp. 91595–91631, 2025.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, et al. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 1962–1979, 2019.

Albert S Yue, Lovish Madaan, Ted Moskovitz, DJ Strouse, and Aaditya K Singh. Harp: A challenging human-annotated math reasoning benchmark. *arXiv preprint arXiv:2412.08819*, 2024.

Zihao Zeng, Yibo Miao, Hongcheng Gao, Hao Zhang, and Zhijie Deng. Adamoe: Token-adaptive routing with null experts for mixture-of-experts language models, 2024. URL https://arxiv.org/abs/2406.13233.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in neural information processing systems*, 36:34661–34710, 2023.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. Agieval: A human-centric benchmark for evaluating foundation models. In *Findings of the association for computational linguistics: NAACL 2024*, pp. 2299–2314, 2024.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Yuhua Zhou, Shaoqi Yu, Shichao Weng, Changhai Zhou, Mingze Yin, Fei Yang, and Aimin Pan. Buddy: Budget-driven dynamic depth routing for adaptive large language model inference, 2026. URL https://arxiv.org/abs/2606.09514.

A Detailed Model and Training Setup

A.1 Multi-Ratio Configuration

Multi-Ratio ConceptMoE supports three target compression ratios,

$$\mathcal{R} = \{2, 4, 8\}.$$

Each ratio uses an independent lightweight boundary router, while the encoder, concept backbone, DeChunk module, and decoder are shared across all ratios. The encoder and decoder use approximately two-thirds of the hidden width of the concept backbone to reduce full-token computation.

During training, each input is encoded once and expanded into three compression views from the same encoder representation. The views are chunked independently according to their sampled compression patterns, packed for shared concept-level processing, and separated again before DeChunk and decoding. Unless otherwise specified, our final model uses three independently sampled cross-ratio views.

A.2 Baselines

We use Large, Medium, and Small MoE baselines for the high-, medium-, and low-compute operating points, respectively. Their non-attention activated FLOPs are approximately matched to CR2, CR4, and CR8:

$$\text{Large} \leftrightarrow \text{CR2}, \qquad \text{Medium} \leftrightarrow \text{CR4}, \qquad \text{Small} \leftrightarrow \text{CR8}.$$

We additionally train three independently optimized fixed-ratio ConceptMoE specialists, denoted as SingleCR2, SingleCR4, and SingleCR8. For the cross-ratio training ablation, we compare

$$3 \text{ single} + 0 \text{ cross}, \qquad 1 \text{ single} + 2 \text{ cross}, \qquad 0 \text{ single} + 3 \text{ cross}.$$

A.3 Cross-Ratio Training Details

For each cross-ratio view, a training sequence is partitioned into contiguous token segments. Interior segment lengths are sampled uniformly from 64 to 960 tokens, with an expected length of 512 tokens; a residual suffix shorter than 64 tokens is absorbed into the preceding segment. The segmentation is performed at the token level and is not constrained to align with semantic or turn boundaries. Each segment independently samples a compression ratio,

$$r_k \sim \text{Uniform}\{2, 4, 8\}.$$

The assigned ratio remains constant within a segment. At every segment start, we force a chunk boundary to keep concept formation aligned with the sampled segmentation, while all other boundaries are predicted adaptively by the corresponding ratio-specific router. This procedure defines the cross-ratio pattern distribution $\mathcal{P}_{\text{cross}}$ used in Section 3.3.

The three compression views are sampled independently, so the same encoder representation can be associated with different segmentations and ratio assignments within the same training step. After pattern-guided boundary selection and chunking, the resulting concept sequences are packed and processed jointly by the shared concept backbone.

Boundary prediction, hard boundary construction, and the compression-control objective follow ConceptMoE Huang et al. (2026) without modification. For cross-ratio training, the compression-control statistics of each ratio-specific router are computed only over tokens assigned to that ratio. Training otherwise uses the standard token-level language-modeling objective together with the inherited compression-control auxiliary objective. We introduce no additional cross-ratio distillation, representation alignment, or consistency objective.

The same Multi-Ratio architecture is used for both PT and SFT. Unless otherwise specified, all final results use three independently sampled cross-ratio compression views per input.

Table 6: **Evaluation sets included in PTBench across different domains.**

Comprehensive Evaluation	Reasoning	Math	Code	Knowledge	Specialized Knowledge
MMLU	BBH	MATH	HumanEval	TriviaQA	GPQA
C-Eval	DROP	AIME 2024	MBPP+	SimpleQA	
MMLU-Pro	ARC-AGI	AIME 2025	LiveCodeBench	ChineseSimpleQA	
AGIEval	ProcBench	HARP	McEval		
HLE	ZebraLogic	Omni-MATH			
SuperGPQA	KOR-Bench	OlympiadBench			
LiveBench					

References: MMLU (Hendrycks et al., 2020); C-Eval (Huang et al., 2023); MMLU-Pro (Wang et al., 2024); AGIEval (Zhong et al., 2024); HLE (Phan et al., 2025); SuperGPQA (Du et al., 2026); LiveBench (White et al., 2025); BBH (Suzgun et al., 2023); DROP (Dua et al., 2019); ARC-AGI (Chollet, 2019); ProcBench (Fujisawa et al., 2024); ZebraLogic (Lin et al., 2025); KOR-Bench (Ma et al., 2025); MATH (Hendrycks et al., 2021); HARP (Yue et al., 2024); Omni-MATH (Gao et al., 2025); OlympiadBench (He et al., 2024); HumanEval (Chen et al., 2021); MBPP+ (Austin et al., 2021; Liu et al., 2023); LiveCodeBench (Jain et al., 2025); McEval (Chai et al., 2025); TriviaQA (Joshi et al., 2017); SimpleQA (Wei et al., 2024); Chinese SimpleQA (He et al., 2025); GPQA (Rein et al., 2023).

Table 7: **Evaluation sets included in SFTBench across different domains.**

Comprehensive Evaluation	Reasoning	Math	Code	Instruction Following	Knowledge / Hallucination	Multilingual
MMLU-Pro	ARC-AGI	AIME 2025	Aider	Multi Challenge	Chinese SimpleQA	MMMLU
MMLU	ProcBench	AIME 2024	LiveCode Bench	IFEval	SimpleQA	
HLE	KOR-Bench	HARP		Collie Hard		
GPQA Diamond	ZebraLogic	Omni-MATH				
SuperGPQA		OlympiadBench				
LiveBench						
MixEvalHard						
ArenaHard						

References: MMLU-Pro (Wang et al., 2024); MMLU (Hendrycks et al., 2020); HLE (Phan et al., 2025); GPQA (Rein et al., 2023); SuperGPQA (Du et al., 2026); LiveBench (White et al., 2025); MixEvalHard (Ni et al., 2024); ArenaHard (Li et al., 2024a); ARC-AGI (Chollet, 2019); ProcBench (Fujisawa et al., 2024); KOR-Bench (Ma et al., 2025); ZebraLogic (Lin et al., 2025); HARP (Yue et al., 2024); Omni-MATH (Gao et al., 2025); OlympiadBench (He et al., 2024); LiveCodeBench (Jain et al., 2025); MultiChallenge (Deshpande et al., 2025); IFEval (Zhou et al., 2023); Chinese SimpleQA (He et al., 2025); SimpleQA (Wei et al., 2024); MMMLU (Singh et al., 2025).

B Evaluation Protocols

B.1 General Benchmarks

We use two aggregate evaluation suites for different training stages: PTBench for PT checkpoints and SFTBench for supervised fine-tuned checkpoints. Their benchmark compositions are summarized in Tables 6 and 7, respectively.

PTBench and SFTBench aggregate their category-level scores using fixed category weights. For PTBench, the weights for Comprehensive, Reasoning, Math, Code, Specialized Knowledge, and Knowledge are $0.30/0.15/0.15/0.15/0.10/0.15$, respectively. For SFTBench, the weights for Comprehensive, Reasoning, Math, Code, Instruction Following, Knowledge, and Multilingual are $0.35/0.10/0.15/0.15/0.10/0.10/0.05$, respectively.

B.2 CODECHAT-EVAL

We use the MBPP+-based subset of CodeChat-Eval Guo et al. (2026) consisting of 378 multi-turn code-refinement trajectories. Each trajectory contains one initial programming turn followed by nine refinement turns, yielding 3,780 turns in total, including 3,402 refinement turns.

We report three complementary metrics.

Functional. For every turn, we use EvalPlus+ functional tests to determine whether the generated program is functionally correct.

Refinement Joint. For turns 1–9, a prediction is considered correct only when it satisfies both functional correctness and the corresponding refinement instruction:

$$\text{Joint} = \text{Functional} \wedge \text{Instruction Adherence}.$$

Functional correctness is determined by EvalPlus+ tests, while instruction adherence is judged by Terra based on GPT-5.6. Refinement Joint is computed over all 3,402 refinement turns and measures whether a model can perform the requested edit while preserving program functionality.

Combined Joint. To obtain a single metric over complete trajectories, turn 0 is evaluated using functional correctness, while turns 1–9 use the Joint criterion:

$$\text{CombinedJoint} = \frac{N_{\text{functional},\, t=0} + N_{\text{joint},\, t=1:9}}{3780}.$$

We use Combined Joint for the accuracy–compute Pareto analysis in the main paper, while Refinement Joint is additionally reported as a more direct measure of code-editing capability.

B.3 COSQL

We evaluate conversational text-to-SQL on the development split of CoSQL Yu et al. (2019). Our primary metric is turn-level SQLite execution accuracy.

For each turn, we execute both the predicted SQL and the gold SQL against the corresponding database and compare their query results. If the gold query explicitly contains an `ORDER BY` clause, result order is preserved during comparison; otherwise, row order is ignored. A prediction is counted as incorrect if the SQL cannot be parsed or executed, if generation terminates before a complete query is produced, or if its execution result differs from that of the gold query.

We additionally compute Exact Match as an auxiliary diagnostic metric. However, syntactically different SQL queries can be semantically equivalent and yield identical execution results. We therefore use execution accuracy, rather than Exact Match, as the primary CoSQL metric throughout the main paper.

B.4 PROMPTING AND GENERATION PROTOCOL

Different checkpoints may require different prompting or prefill formats for stable and well-formed generation, particularly when they are sensitive to special reasoning prefixes such as `<think>`. We therefore use checkpoint-specific compatible configurations to avoid formatting or truncation artifacts, while keeping the task instructions and evaluation criteria unchanged within each benchmark.

B.5 MIXED-RATIO ROUTING PROTOCOL

For mixed-ratio evaluation, we construct task-specific turn-wise compute schedules using `gpt-5.6-terra` as the routing judge. Each decision is generated from a physically prefix-isolated record containing only information available through the current turn. The resulting schedules are frozen before evaluating the reported Multi-Ratio checkpoint. The router does not observe model outputs, correctness judgments, evaluation results, or future turns. The same frozen routing schedule is used when evaluating Multi-Ratio ConceptMoE and the corresponding routed independent-model baseline.

CodeChat-Eval. For each code-refinement turn, the router receives a separate record containing only the original programming problem, the refinement instructions up to and including the current turn, and the current turn index. It does not observe model-generated code, functional-test outcomes, instruction-adherence judgments, fixed-ratio results, future refinement instructions, or model identity. The router is instructed to select the lowest-cost compression mode that maintains an acceptable risk of preserving the behavioral constraints visible so far.

The exact routing prompt is:

```
You are a serving-time routing judge for one turn of a cumulative code-editing
conversation.

Produce three separate judgments:

1. a raw inference tier: CR2, CR4, or CR8;
2. a risk class;
3. confidence in the interpretation of the current request.

The raw tier is an initial routing judgment. A separate deterministic quality
guard will later promote turn 0, all `behavioral` turns, and all low-confidence
turns to CR2. Do not perform that later promotion yourself merely because a
turn is behavioral or low-confidence.

The input is physically prefix-isolated. Use only the original problem,
`visible_instructions` ending at the current turn, `current_instruction`, and
`turn_index`. No future instruction, generated code, test, EvalPlus result,
instruction-following judgment, fixed-CR result, catalog category, or prior
router label is available.

Risk class:

- `mechanical_local`: a clearly local edit, such as a rename, comment,
  formatting/presentation change, or narrowly specified syntactic rewrite.
- `stateful_refactor`: a bounded implementation/refactor that must preserve the
  visible accumulated requirements but does not clearly request changed
  behavior.
- `behavioral`: an edit that changes or may change behavior, invariants,
  algorithm, data structures, error/boundary handling, concurrency, or resource
  semantics.

For turn 0, classify an ordinary bounded initial implementation as
`stateful_refactor`; use `behavioral` only when the original problem itself has
substantial algorithmic, invariant, boundary-condition, or ambiguity risk. Do
not call every initial implementation behavioral merely because it creates
program behavior.

Confidence describes how unambiguously the visible prefix determines the
intended edit. An underspecified instruction can be `low` confidence even when
its risk class is `stateful_refactor`. Do not change the risk class merely to
encode confidence.

Raw CR policy:

- CR8 only for `mechanical_local` with high confidence.
- CR4 is the default for an ordinary bounded implementation or edit, including
  many stateful refactors, underspecified edits, and behavioral edits whose
  raw need for exact context is not clearly critical.
- CR2 only when exact high-fidelity context is already clearly critical before
  the later quality guard: extensive accumulated constraints, fragile
  long-distance state, several interacting edits, or a particularly complex
  behavior/invariant change.
- Turn 0 normally uses CR4; use CR2 only for an unusually complex or fragile
  initial implementation. Never use CR8 for turn 0.
```

```
Do not force CR2 solely because `risk_class` is `behavioral` or confidence is
`low`; the later quality guard handles those conservative promotions.

Return exactly one JSON object per input record:

{
  "route_key": "Mbpp/2::turn-3",
  "task_id": "Mbpp/2",
  "turn_index": 3,
  "input_sha256": "...",
  "route_cr": 4,
  "risk_class": "stateful_refactor",
  "confidence": "low",
  "reason": "one short concrete reason"
}

Allowed risk classes are `behavioral`, `stateful_refactor`, and
`mechanical_local`. Confidence must be `low`, `medium`, or `high`. Preserve
identity and hash fields exactly. Return JSONL only, in input order.
```

For the reported CodeChat-Eval results, we additionally apply a deterministic quality guard to the raw router outputs. The initial implementation is always assigned CR2; behavioral-risk edits and low-confidence decisions are also promoted to CR2:

$$r_t = \begin{cases} 2, & t = 0, \\ 2, & \text{risk}_t = \texttt{behavioral}, \\ 2, & \text{confidence}_t = \texttt{low}, \\ r_t^{\text{Terra}}, & \text{otherwise.} \end{cases} \tag{11}$$

Thus, CR8 is reserved for high-confidence mechanical edits, while CR4 is used primarily for bounded local refactors.

CoSQL. For each user turn, the router receives a separate record containing only the database schema, user utterances up to and including the current turn, and the current turn index. It does not observe gold SQL, model-generated SQL, execution results, exact-match scores, fixed-ratio correctness, or future utterances. The routing objective is to select the lowest-cost compression mode with acceptable risk of corrupting the accumulated SQL state.

The exact user-turn routing prompt is:

```
You are a routing judge for one turn of a multi-turn text-to-SQL conversation.

Choose exactly one inference tier for the current user turn: CR2, CR4, or CR8.
CR2 is strongest and most expensive; CR8 is cheapest and most compressed.

The input is physically prefix-isolated. You may use only:

1. the database schema;
2. `visible_utterances`, which ends at the current turn;
3. `current_utterance` and `turn_index`.

No future utterance, gold SQL, model-generated SQL, execution result, test
outcome, fixed-CR correctness, or hidden label is available. Do not infer or
invent unseen future requests.

Choose the lowest-cost tier with acceptable risk of corrupting the accumulated
SQL state:

- CR8: only for a clearly local, mechanical, high-confidence operation, such as
  changing an explicit constant, sort direction, limit, or projected field,
  where the referent and prior constraints are unambiguous and no join,
  grouping, query-plan, nesting, or set-logic restructuring is needed.
- CR4: the default for a bounded follow-up that requires the visible dialogue
```

```
  state but makes one clear change without substantially restructuring joins,
  grouping, nesting, or set logic.
- CR2: use for the initial query; ambiguous or long-distance references;
  multiple accumulated constraints; corrections or replacement of earlier
  constraints; joins with non-obvious paths; nesting; set operations; negation;
  grouping or HAVING; derived aggregates; or any low-confidence decision where
  an error can corrupt later turns.

The initial user turn must use CR2. Any low-confidence decision must use CR2.
CR8 is allowed only for `mechanical_local` risk with high confidence. If
uncertain between CR4 and CR8, choose CR4. If uncertain whether established SQL
state will be preserved, choose CR2.

For each input JSON object, return exactly one JSON object with:

{
  "route_key": "cosql_dev_0000::turn-0",
  "interaction_id": "cosql_dev_0000",
  "dev_index": 0,
  "turn_index": 0,
  "route_cr": 2,
  "risk_class": "state_establishment",
  "confidence": "high",
  "reason": "one short concrete reason"
}

Allowed risk classes are `state_establishment`, `semantic_state_change`,
`bounded_state_update`, and `mechanical_local`. Preserve the input identity
fields exactly. Return JSONL only, in input order.
```

The unsplit system message, which contains the task instruction and complete database schema, is routed separately using a lightweight heuristic. We assign CR2 when the schema contains at least four tables or when a classifier trained only on the CoSQL training split predicts the initial SQL query to have medium-or-higher hardness; otherwise, we assign CR4:

$$r_{\text{sys}} = \begin{cases} 2, & \text{complex schema or medium-or-harder predicted query,} \\ 4, & \text{otherwise.} \end{cases} \tag{12}$$

The classifier uses only schema statistics and features derived from the visible user-utterance prefix, and is used only for this system-message decision. It does not participate in user-turn routing. CoSQL development-set gold SQL and evaluation outcomes are not used.

Within each benchmark, fixed-ratio and mixed-ratio runs use the same generation prompt and decoding parameters; only the compression-ratio assignment changes.

For completeness, the frozen CodeChat schedule assigns 2,125 (56.2%), 1,062 (28.1%), and 593 (15.7%) turns to CR2, CR4, and CR8, respectively, and contains 1,925 transitions between compression modes.

For CoSQL, the 293 system messages are assigned to CR2/CR4 in 193/100 cases, while the 1,007 user turns are assigned to CR2/CR4/CR8 in 539/334/134 cases, respectively. The resulting CoSQL schedules contain 512 compression-mode transitions, including transitions from the system message to the first user turn.

The same frozen routing schedules are used across the cross-ratio training ablations, so the differences in mixed-ratio accuracy arise under an unchanged external compute-allocation policy.

C FLOPs Accounting

C.1 Terminal-Turn Workload

For CodeChat and CoSQL, we use a standardized *terminal-turn workload* for multi-turn FLOPs evaluation, decomposed into prefill and decode costs:

$$F_{\text{terminal}} = F_{\text{prefill}} + F_{\text{decode}}.$$

Table 8: **Workload-realized compression ratios.** r_{prefill} and r_{decode} are the measured compression ratios used for FLOPs accounting.

Dataset	Model	r_{prefill}	r_{decode}
CodeChat	Multi-Ratio CR2	2.91	3.82
	Multi-Ratio CR4	4.70	5.40
	Multi-Ratio CR8	6.81	8.09
	Multi-Ratio Mixed	3.69	5.69
	SingleCR2	1.54	1.60
	SingleCR4	3.03	3.24
	SingleCR8	7.67	12.27
CoSQL	Multi-Ratio CR2	1.95	2.38
	Multi-Ratio CR4	3.56	4.66
	Multi-Ratio CR8	5.53	6.19
	Multi-Ratio Mixed	2.28	3.36
	SingleCR2	1.88	1.31
	SingleCR4	2.82	2.82
	SingleCR8	3.62	3.32

Within each benchmark, all models use matched prompt and output token workloads, enabling direct comparison of computational cost across model architectures and compression modes under the same input and generation scale. Specifically, we use the Baseline Large token workload as the common reference, avoiding confounding effects from model-dependent generation lengths.

For Multi-Ratio ConceptMoE, prefill and decode computation are scaled using the compression ratios measured from evaluation telemetry, rather than assuming the nominal CR2, CR4, or CR8 values.

The FLOPs reported in the main paper include full attention computation, including $QK^\top$, attention–value multiplication, and sliding-window attention. For prefill, we follow an inference-serving LM-head convention and do not apply the LM head at every prompt position; during decoding, the LM head is computed normally for each newly generated token.

The reported FLOPs measure algorithmic computation under a standardized workload and should not be interpreted as wall-clock latency, throughput, or energy consumption. Actual serving performance may additionally depend on kernel implementations, memory access, communication, and batching strategies.

C.2 Realized Compression Ratios

CR2, CR4, and CR8 denote target average compression ratios rather than fixed-length chunking rules. Because boundary prediction is learned and content-adaptive, independently trained checkpoints can deviate from their nominal target ratios under downstream workloads. For a token sequence of length N, the effective concept-sequence length can be approximated as

$$N_{\text{concept}} \approx \frac{N}{r_{\text{realized}}}.$$

For FLOPs accounting, we therefore use the measured prefill compression ratio r_{prefill} and decode compression ratio r_{decode} for each checkpoint and evaluation workload. The reported ratios are token-weighted averages over the terminal-turn workloads defined in Appendix C.1. Table 8 reports the values used in the CodeChat and CoSQL FLOPs calculations.

These workload-realized ratios, rather than the nominal CR labels, are used when computing the FLOPs reported in the main paper. Consequently, checkpoints with the same nominal CR can have different workload-realized FLOPs.

Table 9: **Switching workload for the routed independent-model baseline.** Base FLOPs follow the same terminal-turn workload used throughout the multi-turn evaluation, while re-prefill FLOPs account for history-state reconstruction after model switches.

Dataset	Prompt Tokens	Output Tokens	Switches	Re-prefilled Tokens	Base FLOPs	Re-prefill FLOPs
CodeChat	923,286	59,910	1,925	2,882,580	1.5530	3.5365
CoSQL	214,740	15,092	512	320,108	0.2830	0.3419

C.3 Routed Independent-Model Switching

Section 4.4 compares Multi-Ratio ConceptMoE with a turn-wise routing baseline based on independent models. Both methods follow the same compute-budget schedule, with

$$\mathrm{CR2} \rightarrow \mathrm{Large}, \qquad \mathrm{CR4} \rightarrow \mathrm{Medium}, \qquad \mathrm{CR8} \rightarrow \mathrm{Small}.$$

Both methods use the terminal-turn workload defined in Appendix C.1 as their base computation. Since CR2, CR4, and CR8 are operating modes of the same Multi-Ratio model, changing the compression mode does not introduce cross-model state incompatibility. In contrast, Large, Medium, and Small are independent models with model-specific history states. For the routed baseline, we account for the history re-prefill required to reconstruct the target model state after each model switch:

$$F_{\mathrm{routed}} = F_{\mathrm{terminal}} + \sum_{j \in \mathcal{S}} F^{(j)}_{\text{re-prefill}},$$

where $\mathcal{S}$ denotes the set of model-switch events.

These additional re-prefill operations are used only to reconstruct the KV state of the target model, and therefore do not apply the LM head at every re-prefilled position. This keeps the same base FLOPs convention for both methods while accounting only for the additional history-state reconstruction cost introduced by independent-model switching. This analysis isolates the additional algorithmic FLOPs associated with state reconstruction under model switching.

The resulting routed totals are 5.0895 PFLOPs on CodeChat and 0.6250 PFLOPs on CoSQL, compared with 1.0999 and 0.3319 PFLOPs for Multi-Ratio Mixed, respectively. This comparison quantifies the additional switching overhead avoided by integrating multiple compute operating points within one shared model.

D Full General-Benchmark Results

D.1 Aggregate Results

Table 10 reports the aggregate results of the three fixed compute operating points on PTBench and SFTBench. We compare the MoE baselines, independently trained SingleCR specialists, and the fixed modes of Multi-Ratio ConceptMoE.

D.2 Category-Level Breakdown

Tables 11 and 12 report the category-level results underlying the aggregate PTBench and SFTBench scores.

Table 10: **Full general-benchmark results across compute budgets.** For the MoE baselines, High/Medium/Low correspond to the operating points matched to CR2/CR4/CR8, respectively. The Avg. rows average over the three operating points.

Model	Operating Point	PTBench	SFTBench
MoE Baseline	High	31.7	19.0
	Medium	28.2	15.5
	Low	25.4	13.1
	Avg.	28.4	15.9
SingleCR Specialist	CR2	**33.2**	21.1
	CR4	**30.0**	18.8
	CR8	25.8	16.4
	Avg.	29.7	18.8
Multi-Ratio	CR2	32.1	**21.5**
	CR4	29.3	**19.1**
	CR8	**27.6**	**17.2**
	Avg.	29.7	**19.3**

Table 11: **Category-level PTBench results.** All entries report accuracy (%). Overall is the weighted aggregate used in the main paper.

Model	Comp.	Reason.	Math	Code	Spec. Know.	Know.	Overall
MoE Large	41.5	25.6	12.1	46.9	26.3	26.6	31.7
Multi-Ratio CR2	42.2	26.3	11.6	46.8	29.8	24.9	32.1
SingleCR2	42.6	27.7	12.6	47.7	29.2	29.1	33.2
MoE Medium	36.5	20.8	9.3	42.5	26.1	25.2	28.2
Multi-Ratio CR4	39.2	22.4	10.6	43.5	27.9	21.9	29.3
SingleCR4	39.8	22.0	11.1	45.2	25.4	25.6	30.0
MoE Small	33.3	16.4	7.5	39.8	25.1	22.3	25.4
Multi-Ratio CR8	35.8	19.5	9.3	41.1	27.6	23.9	27.6
SingleCR8	33.2	17.9	8.8	42.6	25.3	19.4	25.8

Table 12: **Category-level SFTBench results.** All entries report accuracy (%). Overall is the weighted aggregate used in the main paper.

Model	Comp.	Reason.	Math	Code	Inst. Follow.	Know.	Multi.	Overall
MoE Large	28.0	5.5	5.9	4.0	28.9	18.3	49.3	19.0
Multi-Ratio CR2	30.4	8.2	7.3	8.1	33.6	15.7	56.3	21.5
SingleCR2	29.8	8.5	7.0	4.2	35.9	18.5	53.0	21.1
MoE Medium	24.6	3.2	2.6	4.2	21.3	14.0	40.3	15.5
Multi-Ratio CR4	28.0	5.5	4.4	5.3	33.6	15.2	48.0	19.1
SingleCR4	27.8	7.1	4.6	5.2	30.1	14.2	49.0	18.8
MoE Small	21.7	1.9	2.0	2.1	19.9	11.5	32.0	13.1
Multi-Ratio CR8	25.3	4.2	4.6	4.8	32.6	14.2	37.0	17.2
SingleCR8	24.0	6.2	3.3	4.4	30.5	10.7	41.3	16.4

Table 13: **Per-ratio breakdown of cross-ratio training exposure.** All entries report accuracy (%).

Evaluation	Mode	3 Single 0 Cross	1 Single 2 Cross	0 Single 3 Cross
PTBench	CR2	32.4	**32.5**	32.1
	CR4	**29.9**	29.8	29.3
	CR8	**27.6**	**27.6**	**27.6**
SFTBench	CR2	21.2	20.7	**21.5**
	CR4	18.7	18.4	**19.1**
	CR8	16.6	17.1	**17.2**

Table 14: **Mixed-ratio inference under different cross-ratio training exposure.** CodeChat reports Combined Joint accuracy and CoSQL reports SQLite execution accuracy. FLOPs include attention-map computation.

Training Composition	CodeChat Acc. (%)	CodeChat FLOPs (PF)	CoSQL Acc. (%)	CoSQL FLOPs (PF)
3 Single + 0 Cross	37.65	1.1719	43.00	0.3559
1 Single + 2 Cross	37.57	1.1601	45.88	0.3395
0 Single + 3 Cross	**43.78**	**1.0999**	**51.84**	**0.3319**

E Additional Cross-Ratio Training Ablations

E.1 Per-Ratio Fixed-Mode Results

Table 13 expands the aggregate results reported in the main paper into CR2, CR4, and CR8. Increasing cross-ratio exposure has mixed effects at the PT stage, while the fully cross-ratio configuration achieves the strongest SFTBench result at all three fixed operating points.

E.2 Mixed-Inference Compute

Table 14 reports the corresponding mixed-ratio accuracy and workload-realized FLOPs. The gains from fully cross-ratio training are not explained by increased inference computation: on both tasks, the final configuration achieves the highest accuracy with FLOPs comparable to or lower than the alternatives.

F Full-Token Architecture Design

F.1 Motivation

The encoder and decoder in ConceptMoE operate at full token resolution, so their computation does not decrease as the compression ratio increases. If these modules account for a substantial fraction of the total compute, aggressive compression in the concept backbone yields only limited end-to-end savings. We therefore aim to reduce this compression-independent compute floor while preserving sufficient token-level processing capacity.

Two natural design choices are to reduce the depth or the hidden width of the full-token encoder and decoder. Because full-scale pretraining is expensive, we use early-stage optimization behavior to screen candidate architectures before committing to complete training runs.

F.2 Early-Stage Architecture Screening

We compare three representative allocations under a constrained full-token compute budget. The depth of the final encoder/decoder design is normalized to $1\times$, and the concept-backbone hidden width is normalized to $1\times$. Table 15 summarizes the three configurations and their early-stage optimization behavior.

Table 15: **Early-stage screening of full-token encoder/decoder designs.** Alternative configurations with substantially worse early optimization were terminated before full training.

Configuration	E/D Depth	E/D Width	Early Training
Shallow-wide	$0.5\times$	$1.0\times$	Worse convergence
Deep-narrow	$2.0\times$	$0.5\times$	Worse convergence
Balanced (ours)	$1.0\times$	$\sim 0.67\times$	Stable

Both aggressive depth reduction and aggressive width reduction led to noticeably worse optimization during early training and were therefore terminated early. Our final architecture adopts the balanced configuration, retaining moderate token-level depth while reducing hidden width. This design lowers the full-token compute floor while maintaining stable optimization, and is used throughout the main experiments.