FLEXUMM: FLEXIBLE REPRESENTATION LEARNING FOR UNIFIED MULTIMODAL MODELS

Anonymous authors
Paper under double-blind review

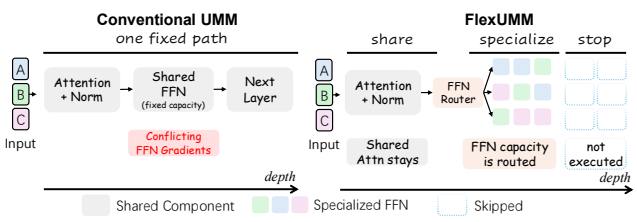
ABSTRACT

Unified multimodal models (UMMs) can integrate understanding and generation within a single backbone. However, it is commonly known that these two capabilities can cause conflicting representations, leading to a central question: how representation and computation should be shared across functionalities? Existing UMMs often rely on a largely fixed structure, where different inputs and functionalities follow the same depth and parameter-sharing pattern. We argue that this assumption is too restrictive: more effective unification requires sharing the computation that supports cross-functional communication while specializing the capacity that becomes task- or input-specific. We introduce FlexUMM, a flexible UMM framework that dynamically decides when to preserve shared computation, when to route specialized capacity, and when to stop further processing. Evidence shows that this flexible allocation improves UMMs along two complementary axes. It improves accuracy by reducing harmful competition between functionalities, and it improves efficiency by avoiding unnecessary computation once an input representation is sufficiently mature. Beyond simple architectural unification from the pre-trained models, FlexUMM works as a model-agnostic, *functional* plug-in to make existing UMMs more unified, stronger, and efficient. Code is available at: `https://anonymous.4open.science/r/flexumm`.

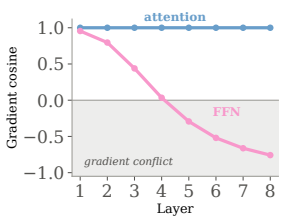
1 INTRODUCTION

Unified multimodal models (UMMs) are built around an appealing idea: understanding and generation can use one backbone instead of being served by two separate models (Wang et al., 2022; Lu et al., 2022a; Sun et al., 2024; Deng et al., 2025). Using the same shared architecture, one UMM can do both generation and understanding tasks with the same network. This unification, however, creates a fundamental challenge in knowledge representation and computation allocation (Wu et al., 2026c; Kil & Kim, 2026). A shared backbone must simultaneously support different capabilities, yet not every input requires the same computation, and not every component of the representation should be shared across capabilities (Han et al., 2026; Kang et al., 2026).

Existing UMMs largely rely on a fixed computational structure (Figure 1a), leaving both the allocation of capacity across capabilities and depth across inputs predetermined (Kil & Kim, 2026; Qiu et al., 2026). Such a rigid allocation can lead to two forms of mismatch: computation may be spent on inputs or capabilities that do not require it, while shared components may be overused even when their computation is not equally beneficial to different capabilities (He et al., 2025; Wu et al., 2026c; Ye et al., 2026). This limitation raises a central question: *how should a UMM adapt its representation and computation to both capabilities more adaptively?* Addressing this question requires resolving two closely related challenges.



(a) Shared communication versus routed FFN capacity.

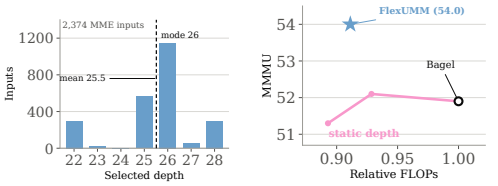


(b) Cross-functional gradient cosine.

Figure 1: FlexUMM adapts both who computes and how much computation an input receives.

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.

First, computation depth should adapt to the input and capability. Not every input requires the full depth of a deep network (Graves, 2016; Elbayad et al., 2019). In a UMM, however, the required depth depends not only on the input but also on the capability being executed (Kang et al., 2026; Su et al., 2026). The depth at which a representation becomes sufficient for understanding should not coincide with the depth required for image generation, even though both capabilities operate on the same backbone. A fixed execution depth therefore imposes the same computational budget on inputs with different requirements. It is evident even within a single functionality: using a learned policy on Bagel, all MME (Fu et al., 2026) inputs are distributed across six distinct depths (Figure 2a). Moreover, the quality–efficiency behavior in Figure 2b suggests that computation beyond the sufficient depth can introduce unnecessary interference and reduce accuracy, rather than simply incurring redundant FLOPs. These observations motivate input-adaptive depth allocation that determines whether further computation remains useful.



(a) Selected depths. (b) Quality–efficiency behavior.

Figure 2: Input-adaptive depth and quality–efficiency behavior.

Second, computational capacity must adapt to the capability. While depth determines how far computation proceeds, it does not determine which components should be shared or specialized. Understanding and generation rely on a common backbone, but their objectives can impose conflicting demands on the same parameters (Han et al., 2026; Wu et al., 2026c). Separating the capabilities entirely would reduce such interference, but would also discard the cross-capability communication that motivates unified modeling (Kang et al., 2026; Su et al., 2026). The challenge is therefore to preserve shared computation where the capabilities benefit from interaction while allowing specialization where their objectives diverge (Peng et al., 2026a; Qiu et al., 2026). Our controlled diagnostic, under the task construction described in the appendix, suggests a structured sharing pattern: attention gradients remain aligned across functionalities throughout the network, whereas FFN gradients increasingly diverge with depth (Figure 1b, Appendix B.1). This is a design hypothesis rather than a direct measurement of every released backbone, and it suggests that communication and transformation capacity should not be treated uniformly. Communication benefits from sharing while transformation capacity requires adaptive specialization. These interactions also have implications beyond accuracy and efficiency. If shared computation is adapted without accounting for capability-specific demands, improvements in one functionality may come with safety regressions in another.

We introduce **FlexUMM**, a flexible computation allocation method that adapts both capacity and depth on a shared UMM backbone. FlexUMM uses capability-adaptive allocation to preserve shared attention and selectively specialize FFN capacity through per-token routing, and depth-adaptive allocation to determine whether further computation is needed through per-sequence early exit. Together, these mechanisms form a progressive *share, specialize, and stop* policy that dynamically allocates computation across capabilities and inputs (Figure 1a). FlexUMM is implemented as a lightweight adaptation on top of the pretrained checkpoint, adding only routers, halting heads and low-rank residual banks, with only 0.381% increase in model size.

Our contributions are summarized as follows:

- We identify two complementary sources of conflicts in UMMs: input-dependent depth and capability-dependent capacity. Our analysis shows that attention provides a shared communication pathway, while FFN capacity exhibits increasing functional divergence across layers.
- We propose FlexUMM, which jointly enables token-level capacity allocation and sequence-level depth adaptation within a unified backbone, allowing computation to be shared, specialized, or terminated according to the input and capability.
- FlexUMM improves both multimodal understanding and generation while providing an adjustable accuracy-compute trade-off at inference time, with negligible extra computation.

2 Related Work

Unified multimodal models. UMMs combine multimodal understanding and generation within a single architecture, with different designs for shared representations, tokenizers, and computation paths (Xie et al., 2025; Chen et al., 2025; Deng et al., 2025; Peng et al., 2026a). Recent studies

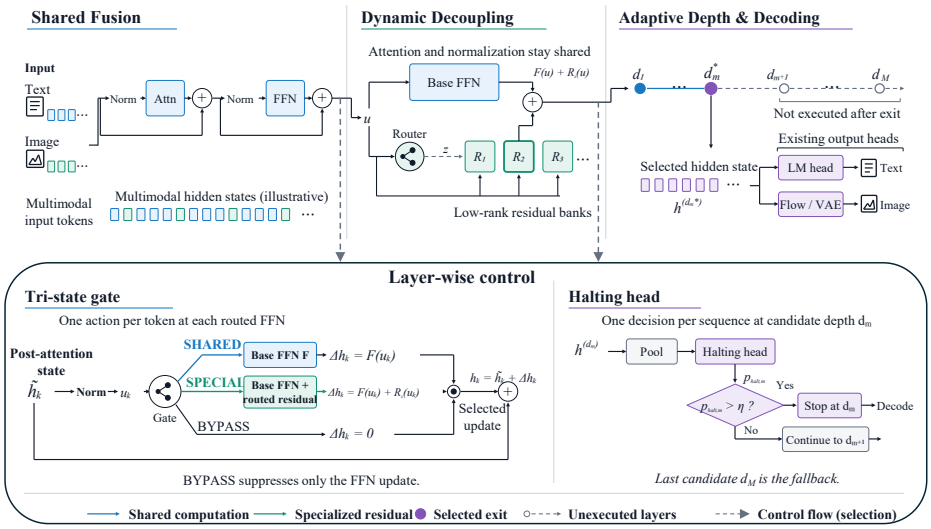


Figure 3: **FlexUMM separates three responsibilities in the implementation.** The shared decoder stack supplies attention and normalization for every route. A tri-state capacity gate selects SHARE, DECOUPLE, or EXIT for each token at each routed FFN. A separate candidate-depth router decides, at the sequence level, whether the remaining decoder layers are still needed.

further examine how architectural choices affect the interaction between functionalities (Kang et al., 2026; Qiu et al., 2026; Wu et al., 2026c). These models validate unification, but their computation paths remain largely fixed, with sharing patterns and depth predetermined before inference.

Adaptive sharing and specialization. Multi-task models mitigate task interference by mixing task-specific representations, routing through shared and private experts, or learning layerwise sharing policies (Misra et al., 2016; Ma et al., 2018; Tang et al., 2020; Sun et al., 2020; Rahimian et al., 2023). UMMs have also introduced function-specific computation to reduce conflicts: Janus decouples visual encoding for understanding and generation, while more recent work specializes conflicting visual computation while preserving semantic interaction (Wu et al., 2025a; 2026c). These approaches mainly target conventional multi-task learning or rely on predefined specialization, rather than dynamically allocating shared and specialized computation.

Adaptive computation. Adaptive computation methods reduce visual processing through token selection, pruning, or merging (Ryoo et al., 2021; Rao et al., 2021; Bolya et al., 2022). PruMerge, FastV, SparseVLM, DyVTE, and HiDrop (Shang et al., 2025; Chen et al., 2024a; Zhang et al., 2024; Wu et al., 2026d;b) exploit visual-token redundancy or vary token participation across layers, while MuE, Mixture-of-Depths, and MoR adapt executed depth or recursive computation (Tang et al., 2023; Raposo et al., 2024; Bae et al., 2026). These methods adapt token participation or executed depth; FlexUMM additionally adapts whether computation is shared or specialized.

3 METHOD

As illustrated in Figure 3, FlexUMM introduces an input-adaptive computation method for UMMs. The model preserves a shared backbone for multimodal communication while adapting both the amount of capacity allocated to individual tokens and the computation depth required by each input. It consists of three stages: 1) Shared Fusion: All tokens undergo the same attention, normalization, and FFN (feed-forward network) to establish a common multimodal representation. 2) Adaptive Specialization: The shared backbone remains intact, while a Tri-state Gate determines whether each token uses the shared FFN alone, receives an additional specialized residual transformation, or bypasses the FFN. 3) Adaptive Depth: A sequence-level Halting Head determines whether the current representation is sufficient, allowing different inputs to terminate computation at different depths. Thus, FlexUMM adapts computation along two complementary dimensions: capacity at the token level and depth at the sequence level, while retaining a shared multimodal communication pathway.

3.1 SHARED COMMUNICATION AND ADAPTIVE CAPACITY

Let $\boldsymbol{h}^{(\ell-1)} \in \mathbb{R}^{n\times d}$ denote the hidden states entering layer ℓ, where n is the sequence length and d is the hidden dimension. Each layer follows the standard Transformer backbone with multi-head

attention and normalization shared across modalities:

$$\tilde{\boldsymbol{h}}^{(\ell)} = \boldsymbol{h}^{(\ell-1)} + \mathrm{Attn}^{(\ell)}\left(\mathrm{Norm}_1^{(\ell)}\left(\boldsymbol{h}^{(\ell-1)}\right)\right), \qquad \boldsymbol{u}^{(\ell)} = \mathrm{Norm}_2^{(\ell)}\left(\tilde{\boldsymbol{h}}^{(\ell)}\right), \tag{1}$$

where $\boldsymbol{u}_k^{(\ell)} \in \mathbb{R}^d$ and $\tilde{\boldsymbol{h}}_k^{(\ell)} \in \mathbb{R}^d$ represent the normalized and residual state for token k, respectively.

Adaptive capacity allocation. At a given layer ℓ, a Tri-state Gate dynamically selects an action $a_k^{(\ell)} \in \mathcal{A} = \{\mathrm{SHARED}, \mathrm{SPECIALIZED}, \mathrm{BYPASS}\}$ to allocate FFN capacity for token k. Specifically, SHARED routes the token through the shared base FFN, SPECIALIZED augments the shared FFN with a learned decoupled residual subset, and BYPASS skips the current FFN update. The gate predicts the action based on the token representation and routing context, enabling input-adaptive capacity allocation without requiring manually annotated token attributes.

Let $s_{k,a}^{(\ell)}$ denote the gate logit for action a. The action probability and inference decision are:

$$\pi_{k,a}^{(\ell)} = \frac{\exp\left(s_{k,a}^{(\ell)}/\tau\right)}{\sum_{a' \in \mathcal{A}} \exp\left(s_{k,a'}^{(\ell)}/\tau\right)}, \qquad a_k^{(\ell)} = \arg\max_{a \in \mathcal{A}} s_{k,a}^{(\ell)}. \tag{2}$$

For the SPECIALIZED action, a learned routing head selects a residual-bank index $z_k^{(\ell)}$ from the token representation and routing context. The index identifies a learned residual subset rather than a manually assigned token label. The overall output state is $\boldsymbol{h}_k = \tilde{\boldsymbol{h}}_k + \Delta\boldsymbol{h}_k$, where the token-level FFN update $\Delta\boldsymbol{h}_k$ is defined as

$$\Delta\boldsymbol{h}_k = \begin{cases} F\left(\boldsymbol{u}_k\right), & a_k^{(\ell)} = \mathrm{SHARED}, \\ F\left(\boldsymbol{u}_k\right) + R_{z_k^{(\ell)}}\left(\boldsymbol{u}_k\right), & a_k^{(\ell)} = \mathrm{SPECIALIZED}, \\ \mathbf{0}, & a_k^{(\ell)} = \mathrm{BYPASS}. \end{cases} \tag{3}$$

Here, F denotes the shared base FFN, while $R_{z_k^{(\ell)}}$ denotes the selected learned residual subset. The residual subsets are optimized jointly with the routing mechanism during post-training, providing additional input-adaptive capacity while preserving the shared FFN as the common foundation.

3.2 Adaptive Computation Depth

While the capacity gate dynamically allocates computation within each layer, the effective computation depth may vary across inputs. We therefore introduce a depth router that selects from a predefined set of candidate exit depths $\mathcal{D} = \{d_1, \ldots, d_M\}$, where $d_1 < \cdots < d_M$. Here, m corresponds to a configured candidate depth rather than a continuously learned computation depth.

At candidate depth d_m, the depth router receives the current shared representation and produces a halting logit r_m. The corresponding halting probability is defined as $p_{\mathrm{halt},m} = \sigma\left(\frac{r_m}{\tau_{\mathrm{depth}}}\right)$, where τ_{depth} denotes the temperature of the depth router and $\sigma(\cdot)$ is the sigmoid function. We distinguish τ_{depth} from the temperature used by the capacity gate, as the two temperatures govern different routing decisions. In our implementation, $\tau_{\mathrm{depth}} = 1.0$.

Sequential Exit Distribution. To jointly optimize the candidate exits, we define a sequential exit distribution based on the halting probabilities:

$$q_m = \begin{cases} p_{\mathrm{halt},m} \displaystyle\prod_{r=1}^{m-1} (1 - p_{\mathrm{halt},r}), & m < M, \\ \displaystyle\prod_{r=1}^{M-1} (1 - p_{\mathrm{halt},r}), & m = M. \end{cases} \tag{4}$$

The final candidate serves as a mandatory fallback, ensuring that $\sum_{m=1}^{M} q_m = 1$. The resulting expected computation depth is $\mathbb{E}[\mathcal{D}] = \sum_{m=1}^{M} q_m d_m$. During training, the full-depth candidate provides the reference computation for assessing the quality of earlier exits. Each candidate is evaluated according to its task objective, and candidates satisfying a predefined quality constraint relative to the

full-depth reference are considered eligible for early termination. The depth router is optimized to favor earlier eligible candidates while retaining the full-depth candidate as a fallback. This training procedure encourages input-adaptive computation allocation without requiring the model to learn an unrestricted continuous exit depth.

Inference-Time Exit Policy. At inference, the model evaluates the configured candidate depths in ascending order and terminates at the first candidate whose halting probability exceeds a threshold:

$$m^\star = \min \{m \geq m_{\min} \mid p_{\text{halt},m} \geq \eta_\beta\}, \tag{5}$$

where $m_{\min}$ specifies the earliest eligible exit and η_β is an inference-time threshold controlling the computation-quality trade-off. If no earlier candidate satisfies the criterion, the model falls back to the final candidate $m = M$. In our experiments, we set the inference threshold to $\eta_\beta = 0.50$.

3.3 Training and Inference

FlexUMM jointly optimizes task performance and computational efficiency through token-level capacity allocation and sequence-level depth adaptation. The overall objective is defined as

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda_{\text{cap}}\mathcal{L}_{\text{cap}} + \lambda_{\text{depth}}\mathcal{L}_{\text{depth}}, \tag{6}$$

where λ_{cap} and λ_{depth} are hyperparameters.

Task Objective. To preserve the native task interfaces of the unified model, we reuse the original output heads at each candidate depth rather than introducing task-specific prediction heads. Specifically, the language-model head is used during the autoregressive decoding process for understanding, whereas the flow-matching head is used during the generation decoding process. The task loss at candidate depth d_m is defined as

$$\ell_m^{\text{task}} = \begin{cases} \mathcal{L}_{\text{CE}}\left(\text{LMHead}(\boldsymbol{g}_m), y\right), & \text{understanding}, \\ \|\text{FlowHead}(\boldsymbol{g}_m) - \mathbf{v}\|_2^2, & \text{generation}, \end{cases} \tag{7}$$

where $\mathbf{v}$ denotes the target flow-matching velocity and $\mathcal{L}_{\text{CE}}$ denotes the cross-entropy loss for the understanding task. The expected task loss across candidate exits is then formulated as

$$\mathcal{L}_{\text{task}} = \sum_{m=1}^{M} q_m \ell_m^{\text{task}}, \tag{8}$$

where q_m is the sequential exit probability defined in Eq. (4). This formulation allows the depth router to account for task quality across candidate computation depths while retaining the native prediction interfaces of the underlying model.

Capacity Regularization. The capacity regularizer encourages efficient allocation of token-level computation across the adaptive layers. Let $\mathcal{A} = \{\text{SHARED}, \text{SPECIALIZED}, \text{BYPASS}\}$ denote the available actions, and let c_a represent the normalized relative computational cost associated with action a. We define

$$\mathcal{L}_{\text{cap}} = \frac{1}{Ln} \sum_{\ell=1}^{L} \sum_{k=1}^{n} \sum_{a \in \mathcal{A}} c_a \, \pi_{k,a}^{(\ell)}, \tag{9}$$

where $\pi_{k,a}^{(\ell)}$ denotes the action probability at layer ℓ for token k, as defined in Eq. (2). The relative costs are determined according to the computation associated with each action, with $c_{\text{BYPASS}} \leq c_{\text{SHARED}} \leq c_{\text{SPECIALIZED}}$. The cost values are normalized with respect to a common reference to ensure that the regularizer reflects the relative computational burden of the available routes.

Depth Regularization. To promote input-adaptive early termination, we penalize the expected computation depth normalized by the maximum candidate depth:

$$\mathcal{L}_{\text{depth}} = \frac{\mathbb{E}[\mathcal{D}]}{d_M} = \frac{1}{d_M} \sum_{m=1}^{M} q_m d_m. \tag{10}$$

This regularization encourages the router to favor earlier candidate exits when they provide sufficient task quality, while retaining the full-depth candidate as a fallback.

Inference and Computational Efficiency. During inference, FlexUMM evaluates candidate depths in ascending order and terminates execution at the first eligible candidate according to the halting policy in Eq. (5). Unlike uniformly deep execution, this adaptive policy allows samples to exit at earlier depths, reducing unnecessary decoder computation and improving inference efficiency.

By jointly adapting token-level capacity and sequence-level depth, FlexUMM reduces redundant computation while preserving the shared multimodal processing pathway. Token-level capacity allocation adjusts the computational route within each layer, while sequence-level early exiting reduces the number of executed layers, enabling more efficient inference.

4 EXPERIMENTS

We perform extensive experiments on effectiveness, efficiency, and explanations of FlexUMM.

4.1 EXPERIMENTAL SETUP

Backbone models. We evaluate FlexUMM on three UMM backbones with different sharing structures: Bagel (Deng et al., 2025), Janus-Pro (Chen et al., 2025), and Harmon (Wu et al., 2025b), with Bagel serving as the primary backbone.

Post-training data. FlexUMM is trained applied to each released checkpoint using ScienceQA (Lu et al., 2022b), AI2D (Kembhavi et al., 2016), TextVQA (Singh et al., 2019), MathV-360K (Shi et al., 2024), and InstructPix2Pix (Brooks et al., 2023).

Evaluation benchmarks and metrics. For understanding, we use MMMU (Yue et al., 2024), MathVista (Lu et al., 2024), MMStar (Chen et al., 2024b), and MMBench (Liu et al., 2024). For generation, GenEval (Ghosh et al., 2023) and WISE (Niu et al., 2025). For unified evaluation, Unison (Liu et al., 2026) and Unified-Bench (Yan et al., 2026).

Implementation details. Only the routers, halting heads, and low-rank residual banks are trained; the backbone configurations and trainable parameter counts are summarized in Appendix Table 7. All experiments are conducted using TorchUMM (Luo et al., 2026) for fair comparison. More details in Appendix A.

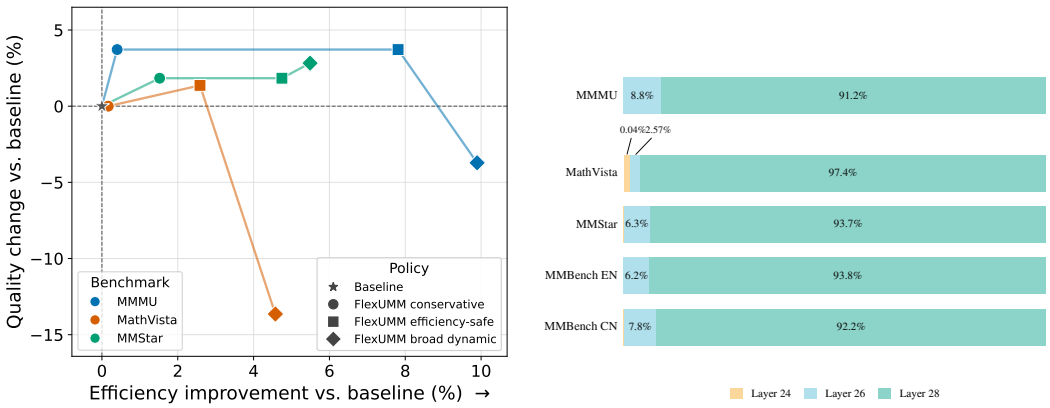
4.2 MAIN RESULTS

FlexUMM improves understanding accuracy over existing UMMs. Table 1 compares FlexUMM with existing unified models across understanding and generation benchmarks. FlexUMM achieves the best results on all five understanding benchmarks and consistently improves both generation metrics over its Bagel backbone, while remaining behind generation-specialized models. These gains demonstrate the benefit of adaptive computation allocation without increasing model scale.

Table 1: Main results. FlexUMM is applied on top of the Bagel backbone; all other rows are released unified models evaluated under the same protocol. Best results are in **bold**.

Model	Params	Understanding					Generation	
		MMMU	MMBench-CN	MMBench-EN	MathVista	MMStar	GenEval	WISE
Bagel (Deng et al., 2025)	14B	51.9	80.94	82.65	71.3	63.20	78.81	0.4044
Janus-Pro (Chen et al., 2025)	7B	40.9	69.93	64.84	43.8	47.93	79.02	0.3900
Janus (Wu et al., 2025a)	1.3B	27.3	37.19	27.69	26.6	26.67	40.04	0.2222
Janus Flow (Ma et al., 2025)	1.3B	29.0	60.31	64.16	34.8	39.13	49.99	0.2964
Show-O2 (Xie et al., 2026)	7B	47.9	76.68	77.25	51.5	54.53	59.87	0.3595
Show-O2 (Xie et al., 2026)	1.5B	37.1	61.04	65.09	37.9	43.87	55.49	0.3349
Show-O (Xie et al., 2025)	1.3B	26.1	11.01	43.48	29.0	27.87	65.06	0.3037
Emu3 (Wang et al., 2024)	8B	31.4	45.66	59.53	44.9	43.60	45.76	0.3373
Emu3.5 (Cui et al., 2025)	34B	29.2	18.13	17.56	41.7	30.27	81.83	**0.6331**
Omnigen2 (Wu et al., 2026a)	7B	46.0	75.43	76.36	63.5	52.47	78.53	0.4029
Harmon (Wu et al., 2025b)	1.5B	35.0	39.43	48.94	22.0	36.53	67.81	0.3917
Bagel+SFT	–	52.6	80.88	82.65	73.1	68.27	78.03	0.2274
Bagel+UniCot (Qin et al., 2026)	–	53.1	80.99	83.12	83.0	70.00	77.91	0.4037
Bagel+IRG (Huang et al., 2026)	–	48.0	57.35	60.47	68.0	61.53	72.06	0.3842
Bagel+FlexUMM	–	**54.0**	**85.87**	**86.75**	**73.4**	**70.93**	**82.03**	0.4292

One model serves multiple budgets. The inference-time threshold η_β traces an accuracy-compute curve without retraining. Table 2 shows that adaptive halting consistently outperforms static trun-



(a) Quality–efficiency trade-off.

(b) Exit-layer distribution.

Figure 4: Quality-efficiency trade-off and adaptive exit distribution.

cation across the operating range, with the optimal depth varying across benchmarks. This supports input- and benchmark-dependent computation allocation rather than a single fixed depth.

Table 2: Effect of the halting threshold η_β on Bagel.

Setting	η_β	$\bar{d}$	Rel. compute	MMMU	MMStar	MathVista
Aggressive	0.30	26.072	0.930	52.9	71.2	69.1
Balanced	0.50	27.841	0.993	54.0	70.9	73.4
Conservative	0.80	28.000	1.000	53.6	70.8	73.1

Figure 4a further shows the accuracy-efficiency trade-off: the efficiency-safe policy preserves full-depth quality while reducing the layer-token compute proxy, whereas more aggressive early exit yields larger savings at some quality cost.

Low adaptation overhead. The added routers, halting heads, and low-rank residual banks increase the checkpoint size by only 0.381% (Appendix Table 7). Early exit reduces the layer-token compute proxy, while Decouple activates only a selected low-rank residual branch. We report these as compute proxy savings rather than direct wall-clock speedups, since the current dense implementation still evaluates the base FFN (Section 4.7).

4.3 Ablation Study

Our ablations are run on Bagel in Table 3.

First, FlexUMM specializes FFN capacity while keeping attention shared, following the gradient-alignment analysis in Section B.1. The first block of Table 3 confirms this choice: routing attention alone underperforms the backbone on most understanding benchmarks, while routing both sublayers recovers only part of the gain. Specialization is therefore most effective when applied to the FFN while preserving shared attention.

Second, FlexUMM uses input-adaptive depth rather than a fixed truncation. The second block of Table 3 shows that fixed depths around FlexUMM's mean exit depth are consistently suboptimal: changing the fixed depth shifts which benchmarks degrade without eliminating the loss. This supports the need for input-dependent depth allocation.

Third, we test whether gains come from learned routing rather than added parameters. The last block of Table 3 keeps the architecture and parameter count fixed but replaces learned routing with random decisions. Random routing loses FlexUMM's gains and falls below the backbone on several benchmarks, confirming that performance depends on effective computation allocation rather than parameter addition alone. The remaining design-choice ablations, including the capacity and stopping variants, are reported in Appendix C.1.

Table 3: Ablation study on Bagel.

Variant	Understanding					Generation	
	MMMU	MMBench-CN	MMBench-EN	MathVista	MMStar	GenEval	WISE
Bagel	51.9	80.94	82.65	71.3	63.20	78.81	0.4044
Bagel+FlexUMM	**53.6**	**85.87**	**86.81**	**73.1**	**70.80**	**82.03**	**0.4292**
Capacity axis: which components are routed							
Routed attention, shared FFN	52.1	82.26	81.95	71.2	62.30	81.21	0.4028
Routed attention, routed FFN	53.2	84.81	84.24	71.9	68.90	80.65	0.3995
Depth axis: fixed instead of input-dependent							
Fixed depth $d = 25$	51.3	81.61	83.58	48.2	68.80	77.19	0.3974
Fixed depth $d = 26$	52.1	81.01	84.13	67.2	67.90	76.87	0.4039
Routing policy							
Random router	52.4	80.48	74.34	71.0	64.53	76.38	0.3849

Table 4: Unification test on Unison (Liu et al., 2026) and Unified-Bench (Yan et al., 2026).

Method	Unison					Unified-Bench				
	Int. Cons.	U.-G. Gen.	G.-G. Und.	Mut. Enh.	Overall	CLIP	DINOv2	DINOv3	LongCLIP	Overall
Bagel	80.72	72.09	33.68	35.87	55.59	89.47	78.77	72.40	93.21	83.46
+FlexUMM	83.21 ↑3.1%	72.96 ↑1.2%	37.08 ↑10.1%	46.08 ↑28.5%	59.83 ↑7.6%	90.91 ↑1.6%	79.88 ↑1.4%	74.53 ↑2.9%	94.87 ↑1.8%	85.05 ↑1.9%

4.4 Evaluation on UMM Unification

We evaluate FlexUMM on two unified-model benchmarks, Unison (Liu et al., 2026) and Unified-Bench (Yan et al., 2026). Effective unification requires a model to support both understanding and generation while maintaining consistent performance across these complementary capabilities. Table 4 shows consistent improvements over the baseline on both unification benchmarks, with gains across all evaluated dimensions and overall scores. These results suggest that adaptive computation and input-dependent depth selection help allocate computation more flexibly across different capabilities, improving the balance between understanding and generation. Overall, FlexUMM enhances the unified performance of the model across both evaluated benchmarks.

4.5 Evaluation on Safety

We further assess safety using Uni-SafeBench (Peng et al., 2026b), which is important for UMMs that combine multimodal understanding with generation, requiring both accurate safety recognition and appropriate response behavior. As shown in Table 5, FlexUMM lowers both unsafe rates while increasing the rejection rate and the rate of outputs that are safe under both criteria. These improvements suggest that adaptive depth selection provides more flexible computation for processing safety-relevant cues and making appropriate response decisions.

Table 5: Uni-SafeBench safety results.

Model	C-Unsafe ↓	C-Reject ↑	I-Unsafe ↓	Both-Safe ↑
Bagel	43.72%	5.07%	49.20%	43.96%
+FlexUMM	**42.35%**	**5.23%**	**49.06%**	**44.86%**

4.6 Generalization Across Backbones

FlexUMM is agnostic to UMM architectures. Table 6 evaluates FlexUMM on two additional backbones, Janus-Pro and Harmon. Generalization across backbones is important for assessing whether the benefits of adaptive computation extend beyond a specific architecture or sharing structure. FlexUMM consistently improves understanding and generation performance across all three backbones while reducing computation, demonstrating robust generalization across unified-model architectures with different sharing structures. The results suggest that the proposed share-specialize-stop formulation can adapt to diverse backbone designs without relying on architecture-specific modifications.

4.7 Effectiveness Analysis

We analyze the computation structure underlying FlexUMM: attention gradients remain aligned across functionalities, while FFN gradients become increasingly competitive with depth (Section B.1). Depth requirements also vary substantially across inputs (Section B.2), supporting FlexUMM's capability-adaptive capacity allocation and input-adaptive computation depth.

Table 6: Generalization across backbones with different sharing structures.

Backbone	Method	Understanding					Generation		Efficiency
		MMMU	MMBench-CN	MMBench-EN	MathVista	MMStar	GenEval	WISE	Rel. compute
Bagel	Full backbone	51.9	80.94	82.65	71.3	63.20	78.81	0.4044	1.00
	FlexUMM	54.0 ↑4.0%	85.87 ↑6.1%	86.75 ↑5.0%	73.4 ↑2.9%	70.93 ↑12.2%	82.03 ↑4.1%	0.4292 ↑6.1%	0.96
Janus-Pro	Full backbone	40.89	64.84	69.93	43.8	47.93	79.02	0.3900	1.00
	FlexUMM	46.11 ↑12.8%	68.47 ↑5.6%	73.11 ↑4.5%	51.0 ↑16.4%	52.33 ↑9.2%	81.03 ↑2.5%	0.4095 ↑5.0%	0.96
Harmon	Full backbone	35.0	39.43	48.94	22.0	36.53	67.81	0.3917	1.00
	FlexUMM	38.6 ↑10.3%	42.6 ↑8.0%	53.09 ↑8.5%	35.1 ↑59.5%	43.07 ↑17.9%	71.71 ↑5.8%	0.4197 ↑7.1%	0.89

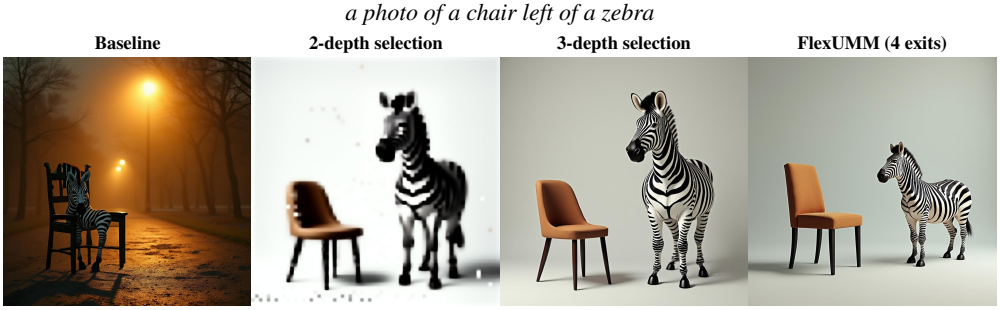


Figure 5: Qualitative depth-selection progression for one generation prompt.

Exit-layer distribution is consistent across benchmarks. Figure 4b shows that across benchmarks, the router systematically exits at earlier candidate depths. Consequently, the sequence-depth component contributes a modest reduction in relative computation across benchmarks. This consistent behavior indicates that adaptive depth selectively allocates less computation to easier inputs, rather than relying heavily on early exit for only specific benchmarks.

4.8 Case Study

We inspect representative outputs under the same generation prompt and seed.

Understanding. The details are provided in Appendix Section D which cover medical terminology, geometric reasoning, person identification, counting, and color recognition.

Generation. We use this case study to illustrate how input-adaptive depth selection affects generation quality under matched evaluation conditions. All outputs are produced with the same checkpoint, prompt, random seed, and decoding settings; only the available candidate-depth configurations differ. The Bagel baseline executes the full backbone, whereas the 2-, 3-, and 4-exit configurations allow the policy to select from progressively finer-grained computation depths. As shown in Figure 5, the baseline fails to satisfy the requested spatial relation, while adaptive depth selection progressively recovers the relation and improves visual consistency. The example is illustrative rather than a standalone benchmark result: Appendix Section D reports ten strict paired understanding wins and six detector-verified generation wins, while the aggregate benchmark results quantify the overall accuracy-efficient benefit.

5 Conclusion

We study adaptive computation in unified multimodal models and propose FlexUMM, a unified framework that dynamically allocates computation according to both capability and input complexity. By adapting computation within and across layers while preserving a shared multimodal backbone, FlexUMM enables the model to allocate computation where it is most useful rather than applying uniform computation to all tokens and inputs. Across three unified multimodal backbones, FlexUMM consistently improves overall performance, achieving an average improvement of **10.17%** while reducing FLOPs by **6.33%**. These results show that adaptive computation provides an effective principle for improving both the capability and efficiency of unified multimodal models. We discuss limitations in Appendix E.

AI USE STATEMENT

We used generative AI tools to assist with manuscript revision, literature search, and code development. The experimental design and methodological choices were made by the authors.

REPRODUCIBILITY STATEMENT

Dataset protocols, preprocessing, model pools, and training hyperparameters are provided in Appendix A.

REFERENCES

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, et al. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *Advances in Neural Information Processing Systems*, 38:96572–96617, 2026.

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your vit but faster. *arXiv preprint arXiv:2210.09461*, 2022.

Tim Brooks, Aleksander Holynski, and Alexei A Efros. Instructpix2pix: Learning to follow image editing instructions. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18392–18402. IEEE, 2023.

Liang Chen, Haozhe Zhao, Tianyu Liu, Shuai Bai, Junyang Lin, Chang Zhou, and Baobao Chang. An image is worth 1/2 tokens after layer 2: Plug-and-play inference acceleration for large vision-language models. In *European Conference on Computer Vision*, pp. 19–35. Springer, 2024a.

Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, et al. Are we on the right way for evaluating large vision-language models? *Advances in Neural Information Processing Systems*, 37:27056–27087, 2024b.

Xiaokang Chen, Zhiyu Wu, Xingchao Liu, Zizheng Pan, Wen Liu, Zhenda Xie, Xingkai Yu, and Chong Ruan. Janus-pro: Unified multimodal understanding and generation with data and model scaling. *arXiv preprint arXiv:2501.17811*, 2025.

Yufeng Cui, Honghao Chen, Haoge Deng, Xu Huang, Xinghang Li, Jirong Liu, Yang Liu, Zhuoyan Luo, Jinsheng Wang, Wenxuan Wang, et al. Emu3. 5: Native multimodal models are world learners. *arXiv preprint arXiv:2510.26583*, 2025.

Chaorui Deng, Deyao Zhu, Kunchang Li, Chenhui Gou, Feng Li, Zeyu Wang, Shu Zhong, Weihao Yu, Xiaonan Nie, Ziang Song, et al. Emerging properties in unified multimodal pretraining. *arXiv preprint arXiv:2505.14683*, 2025.

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

Chaoyou Fu, Peixian Chen, Yunhang Shen, Yulei Qin, Mengdan Zhang, Xu Lin, Jinrui Yang, Xiawu Zheng, Ke Li, Xing Sun, et al. Mme: A comprehensive evaluation benchmark for multimodal large language models. *Advances in Neural Information Processing Systems*, 38, 2026.

Dhruba Ghosh, Hannaneh Hajishirzi, and Ludwig Schmidt. Geneval: An object-focused framework for evaluating text-to-image alignment. *Advances in Neural Information Processing Systems*, 36: 52132–52152, 2023.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Yujin Han, Hao Chen, Andi Han, Zhiheng Wang, Xinyu Liu, Shiwei Zhang, Difan Zou, et al. Turning internal gap into self-improvement: Promoting the generation-understanding unification in mllms. In *International Conference on Learning Representations*, volume 2026, pp. 136501–136534, 2026.

Shwai He, Chaorui Deng, Ang Li, and Shen Yan. Understanding and harnessing sparsity in unified multimodal models. *arXiv preprint arXiv:2512.02351*, 2025.

Wenxuan Huang, Shuang Chen, Zheyong Xie, Shaosheng Cao, Shixiang Tang, Yufan Shen, Qingyu Yin, Wenbo Hu, Xiaoman Wang, Yuntian Tang, et al. Interleaving reasoning for better text-to-image generation. In *International Conference on Learning Representations*, volume 2026, pp. 106153–106182, 2026.

Jiwon Kang, Heeji Yoon, Jaewoo Jung, Jaewon Min, Minkyeong Jeon, Biyeon Hwang, Sangwon Jung, and Seungryong Kim. Transferability between understanding and generation in unified multimodal models. In *European Conference on Computer Vision*, pp. 626–647. Springer, 2026.

Aniruddha Kembhavi, Mike Salvato, Eric Kolve, Minjoon Seo, Hannaneh Hajishirzi, and Ali Farhadi. A diagram is worth a dozen images. In *European conference on computer vision*, pp. 235–251. Springer, 2016.

Dayoung Kil and Seong-heum Kim. Efficient unified multimodal understanding (eumu): Winning solution for the mumu track at the 8th lsvos challenge. *arXiv preprint arXiv:2609.19451*, 2026.

Jinyu Liu, Xincheng Shuai, Henghui Ding, and Yu-Gang Jiang. Unison: Benchmarking unified multimodal models via synergistic understanding and generation. *arXiv preprint arXiv:2606.26984*, 2026.

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, et al. Mmbench: Is your multi-modal model an all-around player? In *European conference on computer vision*, pp. 216–233. Springer, 2024.

Jiasen Lu, Christopher Clark, Rowan Zellers, Roozbeh Mottaghi, and Aniruddha Kembhavi. Unified-io: A unified model for vision, language, and multi-modal tasks. *arXiv preprint arXiv:2206.08916*, 2022a.

Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. *Advances in neural information processing systems*, 35:2507–2521, 2022b.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. In *International Conference on Learning Representations*, volume 2024, pp. 23439–23554, 2024.

Yinyi Luo, Wenwen Wang, Hayes Bai, Hongyu Zhu, Hao Chen, Pan He, Marios Savvides, Sharon Li, and Jindong Wang. Torchumm: A unified multimodal model codebase for evaluation, analysis, and post-training. *arXiv preprint arXiv:2604.10784*, 2026.

Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1930–1939, 2018.

Yiyang Ma, Xingchao Liu, Xiaokang Chen, Wen Liu, Chengyue Wu, Zhiyu Wu, Zizheng Pan, Zhenda Xie, Haowei Zhang, Xingkai Yu, et al. Janusflow: Harmonizing autoregression and rectified flow for unified multimodal understanding and generation. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7739–7751. IEEE, 2025.

Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3994–4003, 2016.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *2019 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 11256–11264. IEEE, 2019.

Yuwei Niu, Munan Ning, Mengren Zheng, Weiyang Jin, Bin Lin, Peng Jin, Jiaqi Liao, Chaoran Feng, Fanqing Meng, Kunpeng Ning, et al. Wise: A world knowledge-informed semantic evaluation for text-to-image generation. *arXiv preprint arXiv:2503.07265*, 2025.

Wujian Peng, Lingchen Meng, Yuxuan Cai, Xianwei Zhuang, Yuhuan Yang, Rongyao Fang, Chenfei Wu, Junyang Lin, Zuxuan Wu, and Shuai Bai. Unified multimodal autoregressive modeling with shared context-visual tokenizer is key to unification. *arXiv preprint arXiv:2606.18249*, 2026a.

Zixiang Peng, Yongxiu Xu, Qin-Yi Zhang, Jiexun Shen, Yi-Fan Zhang, Hongbo Xu, Yubin Wang, and Gaopeng Gou. Does unification come at a cost? uni-safebench: A safety benchmark for unified multimodal large models. *arXiv preprint arXiv:2604.00547*, 2026b.

Luozheng Qin, GONG JIA, Yuqing Sun, Tianjiao Li, Haoyu Pan, Mengping Yang, Xiaomeng Yang, Chao Qu, Zhiyu Tan, and Hao Li. Uni-cot: Towards unified chain-of-thought reasoning across text and vision. In *International Conference on Learning Representations*, volume 2026, pp. 15999–16028, 2026.

Zongyang Qiu, Yihan Wu, Kaixuan Fan, Bo Li, and Hui Xiong. Where a new concept must enter: Entry point gates cross-task usability in unified multimodal models. *arXiv preprint arXiv:2608.17564*, 2026.

Elahe Rahimian, Golara Javadi, Frederick Tung, and Gabriel Oliveira. Dynashare: Task and instance conditioned parameter sharing for multi-task learning. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 4535–4543. IEEE, 2023.

Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. *Advances in neural information processing systems*, 34:13937–13949, 2021.

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.

Michael S Ryoo, AJ Piergiovanni, Anurag Arnab, Mostafa Dehghani, and Anelia Angelova. Tokenlearner: What can 8 learned tokens do for images and videos? *arXiv preprint arXiv:2106.11297*, 2021.

Yuzhang Shang, Mu Cai, Bingxin Xu, Yong Jae Lee, and Yan Yan. Llava-prumerge: Adaptive token reduction for efficient large multimodal models. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 22857–22867. IEEE, 2025.

Wenhao Shi, Zhiqiang Hu, Yi Bin, Junhua Liu, Yang Yang, See Kiong Ng, Lidong Bing, and Roy Ka-Wei Lee. Math-llava: Bootstrapping mathematical reasoning for multimodal large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 4663–4680, 2024.

Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. Towards vqa models that can read. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8309–8318. IEEE, 2019.

Zihan Su, Hongyang Wei, Kangrui Cen, Yong Wang, Guanhua Chen, Chun Yuan, and Xiangxiang Chu. Generation enhances understanding in unified multimodal models via multi-representation generation. *arXiv preprint arXiv:2601.21406*, 2026.

Quan Sun, Yufeng Cui, Xiaosong Zhang, Fan Zhang, Qiying Yu, Yueze Wang, Yongming Rao, Jingjing Liu, Tiejun Huang, and Xinlong Wang. Generative multimodal models are in-context learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14398–14409, 2024.

Ximeng Sun, Rameswar Panda, Rogerio Feris, and Kate Saenko. Adashare: Learning what to share for efficient deep multi-task learning. *Advances in Neural Information Processing Systems*, 33: 8728–8740, 2020.

Hongyan Tang, Junning Liu, Ming Zhao, and Xudong Gong. Progressive layered extraction (ple): A novel multi-task learning (mtl) model for personalized recommendations. In *Proceedings of the 14th ACM conference on recommender systems*, pp. 269–278, 2020.

Shengkun Tang, Yaqing Wang, Zhenglun Kong, Tianchi Zhang, Yao Li, Caiwen Ding, Yanzhi Wang, Yi Liang, and Dongkuan Xu. You need multiple exiting: Dynamic early exiting for accelerating unified vision language model. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10781–10791. IEEE, 2023.

Peng Wang, An Yang, Rui Men, Junyang Lin, Shuai Bai, Zhikang Li, Jianxin Ma, Chang Zhou, Jingren Zhou, and Hongxia Yang. Ofa: Unifying architectures, tasks, and modalities through a simple sequence-to-sequence learning framework. In *International conference on machine learning*, pp. 23318–23340. PMLR, 2022.

Xinlong Wang, Xiaosong Zhang, Zhengxiong Luo, Quan Sun, Yufeng Cui, Jinsheng Wang, Fan Zhang, Yueze Wang, Zhen Li, Qiying Yu, et al. Emu3: Next-token prediction is all you need. *arXiv preprint arXiv:2409.18869*, 2024.

Chengyue Wu, Xiaokang Chen, Zhiyu Wu, Yiyang Ma, Xingchao Liu, Zizheng Pan, Wen Liu, Zhenda Xie, Xingkai Yu, Chong Ruan, et al. Janus: Decoupling visual encoding for unified multimodal understanding and generation. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12966–12977. IEEE, 2025a.

Chenyuan Wu, Jiahao Wang, Pengfei Zheng, Ruiran Yan, Shitao Xiao, Xin Luo, Yueze Wang, Wanli Li, Xiyan Jiang, Yexin Liu, et al. Omnigen2: Towards instruction-aligned multimodal generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 21964–21975, 2026a.

Hao Wu, Yingqi Fan, Dai Jinyang, Junlong Tong, Yunpu Ma, and Xiaoyu Shen. Hidrop: Hierarchical vision token reduction in mllms via late injection, concave pyramid pruning, and early exit. In *International Conference on Learning Representations*, volume 2026, pp. 97296–97316, 2026b.

Penghao Wu, Haiwen Diao, Weichen Fan, Lewei Lu, Dahua Lin, and Ziwei Liu. Uncovering understanding-generation synergy in native unified multimodal models: From representation, task to system. *arXiv preprint arXiv:2609.01607*, 2026c.

Qiong Wu, Wenhao Lin, Yiyi Zhou, Weihao Ye, Zhanpeng Zeng, Xiaoshuai Sun, and Rongrong Ji. Accelerating multimodal large language models via dynamic visual-token exit and the empirical findings. *Advances in Neural Information Processing Systems*, 38:168378–168403, 2026d.

Size Wu, Wenwei Zhang, Lumin Xu, Sheng Jin, Zhonghua Wu, Qingyi Tao, Wentao Liu, Wei Li, and Chen Change Loy. Harmonizing visual representations for unified multimodal understanding and generation. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 17739–17750. IEEE, 2025b.

Jinheng Xie, Weijia Mao, Zechen Bai, David Junhao Zhang, Weihao Wang, Kevin Qinghong Lin, Yuchao Gu, Zhijie Chen, Zhenheng Yang, and Mike Zheng Shou. Show-o: One single transformer to unify multimodal understanding and generation. In *International Conference on Learning Representations*, volume 2025, pp. 28240–28264, 2025.

Jinheng Xie, Zhenheng Yang, and Mike Zheng Shou. Show-o2: Improved native unified multimodal models. *Advances in Neural Information Processing Systems*, 38:47490–47518, 2026.

Zhiyuan Yan, Kaiqing Lin, Zongjian Li, Junyan Ye, Hui Han, Haochen Wang, Zhendong Wang, Bin Lin, Hao Li, Xinyan Xiao, et al. Unified multimodal models as auto-encoders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 41903–41912, 2026.

Sen Ye, Mengde Xu, Shuyang Gu, Di He, Liwei Wang, and Winston Hu. Understanding vs. generation: Navigating optimization dilemma in multimodal models. In *International Conference on Learning Representations*, volume 2026, pp. 25118–25141, 2026.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9556–9567, 2024.

Yuan Zhang, Chun-Kai Fan, Junpeng Ma, Wenzhao Zheng, Tao Huang, Kuan Cheng, Denis Gudovskiy, Tomoyuki Okuno, Yohei Nakata, Kurt Keutzer, et al. Sparsevlm: Visual token sparsification for efficient vision-language model inference. *arXiv preprint arXiv:2410.04417*, 2024.

Appendix
FlexUMM: FlexUMM

CONTENTS

A Configuration **16**
A.1 Backbone configurations 16
A.2 Insertion into the shared decoder 16
A.3 TAFE controller 16
A.4 Low-rank residual banks 17
A.5 Candidate-depth execution 17
A.6 Forward-pass sequence 18
A.7 Metrics 18

B Findings **19**
B.1 Sharing Is Beneficial Early, But Not Uniformly Throughout Depth 19
B.2 Computation Depth Should Be Input-Dependent 23

C Additional Ablation and Policy Analyses **25**
C.1 Remaining Design Choices 25
C.2 Early-Exit Subset Audit 27

D Extended Case Study **29**
D.1 Understanding cases 29
D.2 Generation cases 30

E Limitations **30**

A CONFIGURATION

This section describes the runtime configuration of FlexUMM. It focuses on how the gates, residual banks, and depth controller are inserted into the shared decoder and how they interact during a forward pass. It intentionally does not describe parameter selection, tuning schedules, or the post-training procedure.

A.1 BACKBONE CONFIGURATIONS

The backbone settings and the number of trained parameters used in our experiments are summarized in Table 7.

Table 7: Backbone configurations for FlexUMM.

Backbone	Params	Lang. layers	Candidate depths $\mathcal{D}$	Experts E	Trained params
Bagel	14.607B	28	$\{22, 24, 26, 28\}$	2	0.056B (+0.381%)
Janus-Pro	7.420B	30	$\{21, 24, 27, 30\}$	2	0.035B (+0.472%)
Harmon	1.5B	28	$\{22, 24, 26, 28\}$	2	0.013B (+0.837%)

A.2 INSERTION INTO THE SHARED DECODER

The runtime configuration attaches FlexUMM to the language-decoder stack of the UMM rather than constructing a second model. Each decoder layer keeps its pretrained attention, normalization, and base FFN modules. For a layer ℓ, the shared part of the computation is therefore the ordinary decoder operation

$$\tilde{\boldsymbol{h}}^{(\ell)} = \boldsymbol{h}^{(\ell-1)} + \mathrm{Attn}^{(\ell)}\left(\mathrm{Norm}_1^{(\ell)}\left(\boldsymbol{h}^{(\ell-1)}\right)\right), \qquad \boldsymbol{u}^{(\ell)} = \mathrm{Norm}_2^{(\ell)}\left(\tilde{\boldsymbol{h}}^{(\ell)}\right). \tag{11}$$

The TAFE module is placed at the FFN call. Thus, attention remains the shared communication channel, while the FFN residual is the component whose capacity can be shared, specialized, or suppressed. Understanding and generation have separate TAFE controllers and residual banks, even when both token streams occur in one packed transformer forward.

The runtime preserves the original names and parameter objects of the base FFN projections. A checkpoint can therefore load the pretrained attention, normalization, and FFN weights directly; FlexUMM adds only the controller modules and residual banks around those weights. Packed batches carry token indexes, sample lengths, and task metadata. Before a gate is called, the runtime maps each routed token back to its packed sample so that the correct task embedding and modality-specific controller are used.

A.3 TAFE CONTROLLER

For token k at layer ℓ, the gate forms a route feature by combining the current hidden state with task, layer, route-kind, and remaining-budget information:

$$\boldsymbol{z}_k^{(\ell)} = W_s\ \mathrm{LN}(\boldsymbol{u}_k^{(\ell)}) + \boldsymbol{e}_{\mathrm{task}(k)} + \boldsymbol{e}_{\mathrm{layer}(\ell)} + \boldsymbol{e}_{\mathrm{kind}} + W_b b_k^{(\ell)}. \tag{12}$$

Two small MLP heads read this feature. The utility head produces three values, one for each action in $\{\textsc{Share}, \textsc{Decouple}, \textsc{Exit}\}$, and the subset head produces one logit for each residual bank. The action decision uses the cost-adjusted utility in Eq. (2); the subset decision is the argmax bank index (or the top-k bank indexes when multiple residuals are enabled). The subset index is predicted from the hidden state and metadata; it is not a human-supplied task label.

The hard inference path is implemented tokenwise. For a SHARE token, the base FFN output is retained. For a DECOUPLE token, one selected residual bank is evaluated and added to the base output. For an EXIT token, the FFN output is replaced by zero, so the decoder's outer residual connection carries the post-attention state forward unchanged by the FFN. If top-k routing is used, the selected residual outputs are averaged before being added. The resulting FFN update is exactly the piecewise operation in Eq. (3).

During differentiable routing, the hard action is replaced by the soft mixture π from Eq. (2). With $\rho_{k,e}$ denoting the soft bank weights, the routed FFN output is

$$\Delta \boldsymbol{h}_k^{(\ell)} = \left(\pi_{k,\textsc{Share}}^{(\ell)} + \pi_{k,\textsc{Decouple}}^{(\ell)}\right) F(\boldsymbol{u}_k^{(\ell)}) + \pi_{k,\textsc{Decouple}}^{(\ell)} \sum_{e \in \mathcal{T}_k} \rho_{k,e}^{(\ell)} R_e(\boldsymbol{u}_k^{(\ell)}), \tag{13}$$

where $\mathcal{T}_k$ is the selected top-k bank set. The EXIT probability contributes neither the shared FFN output nor a specialized residual. Evaluation switches back to the discrete argmax route.

A.4 LOW-RANK RESIDUAL BANKS

Each configured specialized bank is a residual adapter rather than a copied FFN block. For bank e, the configured residual block computes

$$R_e(\boldsymbol{u}) = U_e \ \mathrm{SiLU}(D_e \ \mathrm{RMSNorm}(\boldsymbol{u})), \qquad D_e \in \mathbb{R}^{r \times d}, \quad U_e \in \mathbb{R}^{d \times r}. \tag{14}$$

The down projection is initialized with a small random matrix and the up projection is initialized to zero. Consequently, inserting the banks does not change the released backbone's initial forward function, while each bank can learn a different functionality- or modality-conditioned residual. The dense wrapper first evaluates the pretrained base FFN $F(\boldsymbol{u})$ for every routed token. The selected action is then applied to that result: a SHARE token retains $F(\boldsymbol{u})$, a DECOUPLE token adds one selected low-rank residual $R_e(\boldsymbol{u})$, and an EXIT token masks the FFN update after the base result has been formed. Thus DECOUPLE adds only a low-rank residual bank (roughly the two rank-r projections) rather than a second full FFN, but SHARE or EXIT should not be interpreted as removing the dense base-FFN kernel in this runtime. The action-level compute proxy credits the low-rank branch only to selected tokens; direct execution savings come from terminating the sequence before later decoder layers.

A.5 CANDIDATE-DEPTH EXECUTION

The depth controller is separate from the tokenwise TAFE gate. It is a small MLP shared across the configured candidate exits. At candidate depth d_m, the controller pools the current shared hidden state for the routed sequence and combines that pooled state with the candidate-depth embedding, route-kind embedding, task embedding, and generation timestep when one is available. An optional previous-candidate state supplies a representation-change feature. The controller returns a halt logit and, when enabled, a predicted harmful regret and uncertainty scale.

The runtime evaluates layers only up to the largest configured candidate. At each candidate it computes $p_{\mathrm{halt},m}$ and applies the eligibility threshold from Eq. (5). The first eligible candidate is selected; the last candidate is always the fallback. If all active token routes take EXIT at an earlier layer, the decoder loop can terminate immediately, before the next candidate-depth check. Thus the two controls have different granularities: TAFE acts on token–layer pairs, while the depth controller terminates the remaining sequence-level computation.

A.5.1 QUALITY-CONSTRAINED ROUTER TRAINING

The depth router is trained from all configured candidate depths in one forward pass. If the candidate depths are $d_1 < \cdots < d_M$, the runtime stores the hidden state $h_{i,m}$ for sample i at every d_m while executing the backbone to d_M. It then applies the existing task head to every candidate: the language-model head produces a teacher-forced cross-entropy loss for understanding, while the flow-to-VAE head produces a flow-matching MSE against the same generation target used at the final depth. Hence the router is not trained from a single early-exit head or from a sampled candidate; every candidate contributes a task-cost value $c_{i,m}$ on every training example.

The quality constraint is defined relative to the final candidate. Let δ be the configured quality margin and let $a_{i,m}$ denote any task-specific validity mask (for example, the label-correct target used by the understanding router). The candidate is eligible when

$$s_{i,m} = \mathbf{1}[c_{i,m} \le c_{i,M} + \delta]\, a_{i,m} = 1, \qquad s_{i,M} = 1. \tag{15}$$

The final-candidate override guarantees a valid target even when no earlier candidate satisfies the quality constraint. The default supervised target is the first eligible candidate,

$$t_i = \min\{m : s_{i,m} = 1\}, \tag{16}$$

although the implementation also supports selecting the lowest-cost or best candidate target when that strategy is requested. The candidate costs, eligibility masks, and teacher-regret labels are detached before they supervise the router; they are labels for allocation, not an additional gradient path through the router into the backbone.

The sequential probabilities $q_{i,m}$ in Eq. (4) are the differentiable training surrogate for the first-halt policy. The router-side objective combines the expected detached candidate cost, a normalized expected depth, entropy regularization, and supervised/calibration terms:

$$\begin{aligned}\mathcal{L}_{\text{router}} = \lambda_{\text{task}} \sum_m q_{i,m}\ \text{sg}(c_{i,m}) + \lambda_{\text{depth}} \sum_m q_{i,m} \frac{d_m}{d_M} - \lambda_{\text{ent}}\, H(q_i) \\ + \lambda_{\text{sup}} \mathcal{L}_{\text{sup}}(q_i, t_i) + \lambda_{\text{cal}} \mathcal{L}_{\text{cal}}(q_i, s_i) + \lambda_{\text{risk}} \mathcal{L}_{\text{risk}}.\end{aligned} \tag{17}$$

Here sg denotes stop-gradient. The supervised term is a masked binary-cross-entropy loss on the sequential halt hazards: candidates before t_i are trained to continue, candidate t_i to halt, and later candidates are not treated as competing positive exits. The calibration term aligns the predicted halt behavior with the eligible-candidate mask. The optional risk head predicts both a harmful-regret probability $\hat{r}_{i,m}$ and a positive uncertainty scale $\hat{u}_{i,m}$. Its loss is a binary cross-entropy term for the task-specific risk target plus a weighted smooth-L_1 term that calibrates the uncertainty scale. For understanding, the harmful- regret target marks a candidate only when it is wrong while the final-candidate teacher is correct, so correcting a teacher mistake is not penalized. For generation, the risk target is built from normalized excess flow-MSE relative to the final candidate (together with the configured candidate-cost strategy), rather than from label correctness.

For clarity, the complete training sequence is: (1) run to the deepest candidate and collect every candidate hidden state; (2) evaluate every native task head and form the per-candidate task costs; (3) construct quality, label-correctness, and harmful-regret targets from detached candidate and final candidate predictions; (4) form $q_{i,m}$ and the router objective above; and (5) backpropagate the native task loss, multi-exit task losses, final-depth hidden-state distillation, router loss, and TAFE loss through the configured trainable adaptation modules. The router input features are detached as well, so the router cannot reshape the backbone representation merely to satisfy its allocation labels.

At inference, the router scans candidates in increasing depth. It halts at the first candidate satisfying the minimum-depth condition, the halt probability threshold, and when the risk head is enabled, the upper confidence bound

$$\hat{r}_{i,m} + \kappa \hat{u}_{i,m} \le \tau_{\text{risk}}. \tag{18}$$

The final candidate remains an unconditional fallback. Thus $q_{i,m}$ is a smooth surrogate used during training, whereas deployment uses the first eligible candidate with fixed validation-time thresholds; it does not inspect test labels or recompute the quality constraint at inference.

A.6 Forward-pass sequence

For one packed forward pass, the configured computation proceeds as follows:

1. The runtime identifies the understanding and generation token streams, records their packed positions, and associates each sample with its route kind and task metadata.
2. At each decoder layer, the shared attention and normalization run on the packed sequence. The corresponding TAFE gate is then called on each active stream, and the selected FFN residual is written back to the same packed positions.
3. After the layer update, an all-route EXIT condition may stop the decoder loop. Otherwise, at a configured candidate depth, the depth controller decides whether the sequence continues.
4. The selected hidden state is passed to the backbone's existing output head: the language-model head for understanding or the flow-to-VAE head for generation. No independent prediction head is introduced for an intermediate depth.

A.7 Metrics

We report three efficiency quantities. *Relative compute* is a normalized compute proxy, $\mathbb{E}[\mathcal{C}_{\text{proxy}}]/\mathcal{C}_{\max}$, where the reference is the same backbone run fully shared to full depth. The

proxy has two components: (i) layer–token work, which decreases when the depth policy stops a sequence and skips the remaining layers, and (ii) the action-conditioned FFN term, which charges a full shared-FFN unit to SHARE and only the selected low-rank residual-bank work to DECOUPLE. In the routing telemetry this action term uses nominal relative costs of 1.0, 0.75, and 0.0 for SHARE, DECOUPLE, and EXIT, respectively; these weights describe allocation, not hardware-measured FLOPs. The latter is important because the dense base FFN is evaluated before the tokenwise action mask in the current wrapper. Thus the direct and robust runtime saving is the sequence-level skip; the realized latency benefit of the low-rank term depends on sparse or fused kernel support and is not measured by this paper's proxy. Consequently, the relative values in Table 6 should not be read as early-exit percentages or as wall-clock speedups: they combine the depth and action-level accounting terms, while launch, packing, and memory effects are outside the proxy. *Mean exit depth* $\bar{d} = \sum_{\ell \in \mathcal{D}} q^{(\ell)} \ell$ summarizes when the model stops computing. *Specialization rate* $\bar{\alpha} = \frac{1}{NL} \sum_{i=1}^{N} \sum_{\ell=1}^{L} \mathbf{1}\big[a_i^{(\ell)} = \text{DECOUPLE}\big]$ is the fraction of the $N \times L$ token–layer pairs that take the DECOUPLE action of Equation (3), and summarizes how much capacity specializes.

B FINDINGS

B.1 SHARING IS BENEFICIAL EARLY, BUT NOT UNIFORMLY THROUGHOUT DEPTH

In early layers, different functionalities can benefit from processing a common representation. Shared attention provides a mechanism for information exchange across modalities and functionalities, allowing different pathways to use complementary contextual information. However, as representations become increasingly task-specific, forcing all functionalities through the same representational capacity can induce competition. This is particularly problematic for the FFN, which provides much of the transformer's representational capacity.

B.1.1 CONTROLLED (SYNTHETIC) LAYERWISE COMPETITION DIAGNOSTIC

This study is designed to answer one mechanistic question: when two functionalities share the same transformer component, do their learning signals update that component in compatible or conflicting directions? We use a synthetic two-task setting so that the amount of shared structure and task-private pressure can be controlled directly. The shared target, private targets, and depth-increasing private weight are deliberately constructed interventions. They are useful for isolating a mechanism, but they are not measurements of a released Bagel, Janus-Pro, Harmon, or Show-o backbone.

Each input sequence is associated with two tasks, denoted task A and task B. Both tasks depend on a shared target y^{sh}, but each task also has its own private target, y^A or y^B. The shared target creates a reason for the two tasks to benefit from common computation, while the private targets create task-specific pressure. For layer ℓ, we write the task losses as

$$\mathcal{L}_\ell^A = \mathcal{L}_\ell^{\text{sh}} + \lambda_\ell \mathcal{L}_\ell^{A,\text{priv}}, \tag{19}$$

$$\mathcal{L}_\ell^B = \mathcal{L}_\ell^{\text{sh}} + \lambda_\ell \mathcal{L}_\ell^{B,\text{priv}}. \tag{20}$$

Here, $\mathcal{L}_\ell^{\text{sh}}$ is the shared objective at layer ℓ, $\mathcal{L}_\ell^{A,\text{priv}}$ and $\mathcal{L}_\ell^{B,\text{priv}}$ are task-private objectives, and λ_ℓ controls the strength of the private objective at depth ℓ. Increasing λ_ℓ with depth lets us isolate how growing task-specific demands affect shared computation.

For a transformer component c at layer ℓ, such as attention, normalization, or FFN, let θ_ℓ^c denote its parameters. We compute the task-specific gradients

$$g_\ell^{A,c} = \nabla_{\theta_\ell^c} \mathcal{L}_\ell^A, \tag{21}$$

$$g_\ell^{B,c} = \nabla_{\theta_\ell^c} \mathcal{L}_\ell^B. \tag{22}$$

The alignment between the two tasks for component c at layer ℓ is measured by gradient cosine similarity:

$$\cos_\ell^c = \frac{\langle g_\ell^{A,c}, g_\ell^{B,c} \rangle}{\|g_\ell^{A,c}\|_2 \|g_\ell^{B,c}\|_2}. \tag{23}$$

Positive cosine means the two tasks update the shared component in compatible directions. Negative cosine means the tasks compete: improving one task pushes the shared component in a direction that hurts the other. We report FFN competition as

$$\mathrm{comp}_\ell^{\mathrm{FFN}} = \max(0, -\cos_\ell^{\mathrm{FFN}}). \tag{24}$$

Thus, a competition value of 0 means no negative alignment, while larger values indicate stronger conflict.

For a model with L layers, the depth-dependent private weight is

$$\lambda_\ell = 0.15 + 2.5 \left(\frac{\ell - 1}{L - 1} \right)^{1.35}, \qquad \ell \in \{1, \ldots, L\}. \tag{25}$$

Thus, $\lambda_1 = 0.15$ in the first layer and $\lambda_L = 2.65$ in the final layer for every model depth. The three regimes use the same λ_ℓ schedule and differ by the private-target amplitude α, which scales the opposing task-private target before it enters $\mathcal{L}_\ell^{A,\mathrm{priv}}$ and $\mathcal{L}_\ell^{B,\mathrm{priv}}$. For readability, Table 8 also reports the endpoint range of the amplitude-weighted private pressure, $\alpha\lambda_1 \to \alpha\lambda_L$.

Table 8: Private-pressure settings for the controlled layerwise competition study.

Regime	α	$\alpha\lambda_1 \to \alpha\lambda_L$	Description
`weak-private`	0.5	$0.075 \to 1.325$	Half-amplitude opposing private target
`medium-private`	1.0	$0.150 \to 2.650$	Unit-amplitude opposing private target
`strong-private`	1.5	$0.225 \to 3.975$	1.5×-amplitude opposing private target

We evaluate three model configurations (4, 8, and 12 layers), these three private-objective regimes, eight random seeds per condition, and 16,384 gradient samples per condition. The `weak-private` regime keeps private pressure small, the `strong-private` regime makes private pressure dominant, and `medium-private` uses an intermediate setting where the tasks still share context but develop increasingly task-specific objectives with depth.

Table 9: Layerwise gradient alignment for the 8-layer model under the `medium-private` regime.

Layer	Attention cosine	FFN cosine	FFN competition
1	1.000	0.955	0.000
2	1.000	0.796	0.000
3	1.000	0.439	0.000
4	1.000	0.034	0.004
5	1.000	-0.293	0.293
6	1.000	-0.520	0.520
7	1.000	-0.663	0.663
8	1.000	-0.758	0.758

Within this controlled family of synthetic conditions, we observe a consistent asymmetric pattern: shared attention maintains strong alignment across depth, while FFN alignment progressively deteriorates and becomes increasingly competitive in deeper layers. This conditional result is a design hypothesis and stress test, not evidence that the same gradient trajectory must occur in every production backbone. For example, in the 8-layer `medium-private` setting, FFN cosine drops from 0.955 in layer 1 to -0.758 in layer 8, while attention cosine remains 1.000 throughout.

B.1.2 Shared Normalization Ablation

We further isolate the role of shared normalization using the same controlled two-task setting. This ablation separates three possible sources of alignment: shared attention, shared normalization, and shared FFN capacity. The objective is to determine whether positive cross-task alignment comes only from attention, or whether normalization can also provide a stabilizing shared pathway.

The notation is the same as in the layerwise study. "Shared Attn." indicates whether task A and task B use the same attention parameters $\theta_\ell^{\mathrm{attn}}$. "Shared Norm" indicates whether the tasks use the same

Table 10: Aggregate layerwise trends from the first to the final layer.

Model	Regime	Attention cosine	FFN cosine / competition
4L-8H	medium-private	$1.000 \rightarrow 1.000$	$0.954 \rightarrow -0.757$ / $0.000 \rightarrow 0.757$
4L-8H	weak-private	$1.000 \rightarrow 1.000$	$0.988 \rightarrow -0.316$ / $0.000 \rightarrow 0.316$
4L-8H	strong-private	$1.000 \rightarrow 1.000$	$0.900 \rightarrow -0.883$ / $0.000 \rightarrow 0.883$
8L-8H	medium-private	$1.000 \rightarrow 1.000$	$0.955 \rightarrow -0.758$ / $0.000 \rightarrow 0.758$
8L-8H	weak-private	$1.000 \rightarrow 1.000$	$0.989 \rightarrow -0.249$ / $0.000 \rightarrow 0.249$
8L-8H	strong-private	$1.000 \rightarrow 1.000$	$0.906 \rightarrow -0.878$ / $0.000 \rightarrow 0.878$
12L-16H	medium-private	$1.000 \rightarrow 1.000$	$0.956 \rightarrow -0.751$ / $0.000 \rightarrow 0.751$
12L-16H	weak-private	$1.000 \rightarrow 1.000$	$0.989 \rightarrow -0.287$ / $0.000 \rightarrow 0.287$
12L-16H	strong-private	$1.000 \rightarrow 1.000$	$0.903 \rightarrow -0.883$ / $0.000 \rightarrow 0.883$

normalization parameters $\theta_\ell^{\text{norm}}$. When a component is private, each task has its own copy of the parameters, so there is no shared component gradient cosine to report for that component. This is marked as N/A in the table.

The FFN objective in this ablation is deliberately constructed to oppose the two tasks, so the FFN cosine is expected to be negative. This controlled design lets us ask whether shared attention and shared normalization can remain aligned even when the capacity-bearing FFN objective is maximally competitive.

Table 11: Shared normalization ablation in the controlled two-task setting.

Condition	Shared Attn.	Shared Norm	Attn. cosine	Norm cosine	FFN cosine
Shared Attn. + Shared Norm	✓	✓	$0.989 \rightarrow 0.989$	$1.000 \rightarrow 1.000$	$-1.000 \rightarrow -1.000$
Shared Attn. + Private Norm	✓	×	$0.989 \rightarrow 0.989$	N/A	$-1.000 \rightarrow -1.000$
Private Attn. + Shared Norm	×	✓	N/A	$1.000 \rightarrow 1.000$	$-1.000 \rightarrow -1.000$
Private Attn. + Private Norm	×	×	N/A	N/A	$-1.000 \rightarrow -1.000$

The results reveal an important asymmetry in unified computation: communication and capacity do not have the same optimal sharing pattern. The key observation is not simply that competition increases with depth, but that different components exhibit fundamentally different sharing profiles. Attention remains aligned while FFN capacity becomes increasingly task-specific. This motivates the first design principle of FlexUMM: preserve shared communication while selectively specializing task-specific capacity.

Evidence boundary. The paper therefore separates the mechanistic diagnostic from the real-model evidence. The gradient cosines above are synthetic and human-designed; they show that the proposed sharing asymmetry can arise from increasing private pressure, but they do not establish its magnitude in a production backbone. The empirical tests on Bagel and Janus-Pro (and the additional Harmon benchmark) measure task quality, routing behavior, and compute proxies. The separate Show-o2 token-importance experiment is a real Show-o-family analysis of functionality-specific importance, not a direct layerwise gradient-cosine measurement. These real-model results test whether the resulting flexible allocation policy is useful; they should not be read as a direct validation of the exact synthetic gradient values. Direct component-gradient measurements on each released backbone would be a separate experiment.

B.1.3 Layerwise Token-Importance Sharing Across Functionalities

We further examine whether a fixed degree of sharing between functionalities is appropriate throughout computation in a real UMM. We use Show-o2 and estimate token importance separately for understanding and generation, then compare the resulting functionality-specific importance patterns on the same held-out inputs.

Token-importance estimation. For token i at layer ℓ, we define the layer-induced representation update as

$$\Delta h_{\ell,i} = h_{\ell,i}^{\text{out}} - h_{\ell,i}^{\text{in}}. \tag{26}$$

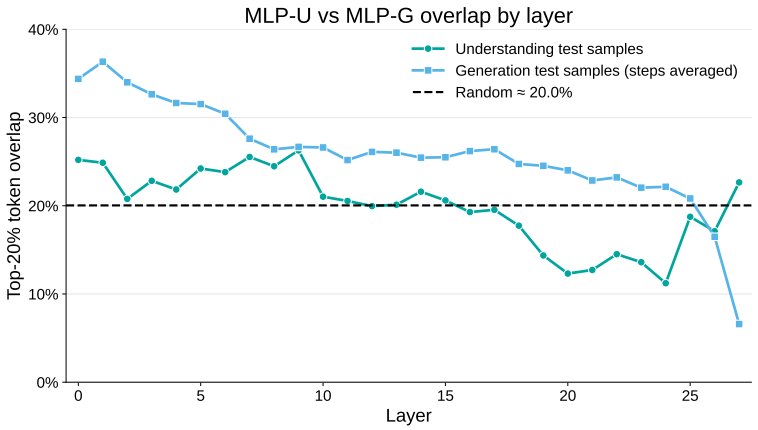


Figure 6: Layerwise top-20% token overlap between MLP-U and MLP-G predictions on held-out Show-o2 understanding and generation samples. Generation results are averaged over generation steps.

Following first-order Taylor importance estimation (Molchanov et al., 2019), we assign each token the score

$$s_{\ell,i} = \left| \nabla_{h_{\ell,i}^{\text{out}}} \mathcal{L}^\top \Delta h_{\ell,i} \right|, \tag{27}$$

which measures the alignment between the layer update and the gradient of the task loss.

For understanding, $\mathcal{L}$ is the cross-entropy loss on the expected answer tokens under teacher forcing. For generation, $\mathcal{L}$ is the flow-matching MSE loss computed along the generation trajectory using a Show-o2-generated reference latent. Show-o2 remains frozen when collecting these scores.

Functionality-specific predictors. We train two lightweight predictors, denoted MLP-U and MLP-G, to estimate the log Gradient $\times$ Update score from the token hidden state and layer identity. MLP-U is trained only on understanding samples, while MLP-G is trained only on generation samples.

The understanding set contains four task categories: counting, spatial relations, attribute combinations, and visual-state recognition, with six samples per category. For each category, four samples are used for training, one for validation, and one for testing. The generation set contains 12 samples, split into 8 for training, 2 for validation, and 2 for testing, with all 49 generation steps recorded. Each predictor is trained with two random seeds, and the evaluation results are averaged across seeds.

As a basic check that the predictors capture their corresponding importance signals, we compare their predicted top-20% tokens with those selected directly by the Gradient $\times$ Update scores. MLP-U reaches 64.8% overlap on understanding test samples, while MLP-G reaches 34.3% on generation test samples. Cross-functionality transfer is lower, with 23.9% for MLP-U on generation and 17.9% for MLP-G on understanding. For independent random top-20% selections, the expected normalized overlap is approximately 20%, since each selected token has a 20% chance of appearing in the other set.

Layerwise sharing. To compare the two functionality-specific importance patterns under the same input, we apply MLP-U and MLP-G to identical held-out samples. At each layer, each predictor selects its top K tokens, where $K = \lceil 0.2N \rceil$ and N is the number of valid tokens. We compute

$$\text{Overlap}_\ell = \frac{\left| \mathcal{T}_\ell^U \cap \mathcal{T}_\ell^G \right|}{K}, \tag{28}$$

where $\mathcal{T}_\ell^U$ and $\mathcal{T}_\ell^G$ are the top-K sets predicted by MLP-U and MLP-G, respectively.

Figure 6 shows a depth-dependent pattern across both functionalities. On understanding inputs, the overlap is generally above the random baseline in earlier layers and becomes lower in deeper layers. On generation inputs, the trend is more pronounced: overlap is substantially above random

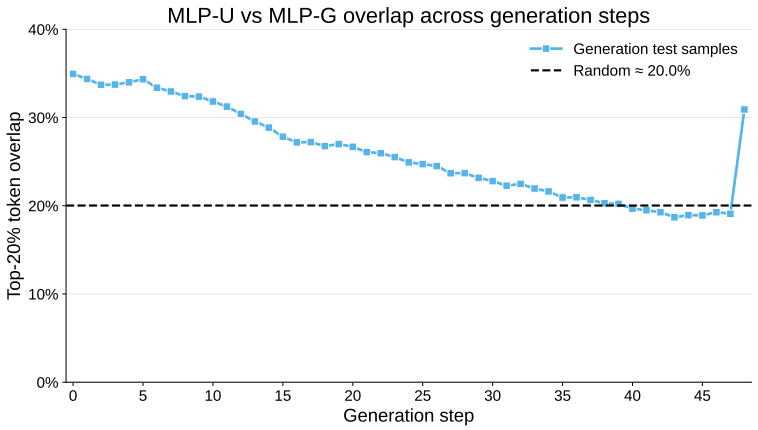


Figure 7: Top-20% token overlap between MLP-U and MLP-G predictions across generation steps on held-out Show-o2 generation samples, averaged over layers.

in early layers and progressively decreases with depth, falling below the random baseline in the final layers. These results are consistent with stronger sharing in earlier computation and increasing functionality-specific specialization at greater depth.

Variation across generation steps. We further examine whether the relationship between the two functionality-specific predictors changes over the generation trajectory. At each generation step, we average their top-20% token overlap across layers.

As shown in Figure 7, the overlap generally decreases over the generation trajectory and approaches the random baseline at later steps. The final step is an exception to this overall trend. Together, the layerwise and generation-step analyses suggest that a fixed sharing pattern may be too restrictive, as functionality-specific token-importance patterns vary across both model depth and generation progress.

B.2 Computation Depth Should Be Input-Dependent

Even after accounting for parameter competition, a second limitation remains: UMMs generally require every input to traverse the same number of layers. Different inputs can reach a sufficiently mature representation at different depths. A simple understanding query may require relatively little computation, whereas a difficult reasoning problem may benefit from deeper processing. Generation inputs can also differ substantially in the amount of refinement they require.

B.2.1 Bagel Selected-Depth Distribution on MME

We first investigate this question using Bagel with a learned FlexUMM routing policy on the full 2,374-question MME benchmark. The goal is to test whether inputs within the same model and the same functionality prefer different amounts of computation.

Let x_i denote the i-th MME input. Bagel contains 28 language layers, and the routing policy evaluates a candidate exit-depth set

$$\mathcal{D} = \{22, 23, 25, 26, 27, 28\}. \tag{29}$$

For each input x_i, the router selects one depth

$$d_i = r_\phi(x_i), \qquad d_i \in \mathcal{D}, \tag{30}$$

where r_ϕ is the learned routing policy and d_i is the selected number of language layers used for that input. A smaller d_i means the model exits earlier and saves computation; a larger d_i means the model continues deeper for additional refinement.

Table 12 reports how often each depth is selected. "MME samples" is the number of benchmark examples assigned to a given depth, and "Percentage" is this count divided by the full 2,374-question benchmark size.

Table 12: Selected-depth distribution for Bagel on MME.

Selected depth	MME samples	Percentage
22	296	12.47%
23	18	0.76%
25	564	23.76%
26	1,146	48.27%
27	52	2.19%
28	298	12.55%
Total	2,374	100.00%

The selected depth ranges from 22 to 28 layers, with a mean selected depth of 25.514, a minimum of 22, a maximum of 28, and six distinct selected depths. Nearly half of the examples select depth 26, but substantial mass remains at both extremes: 12.47% exit at depth 22 and 12.55% run the full 28 layers. This variation occurs within a single model and a single benchmark, rather than simply reflecting differences between understanding and generation tasks; it is genuine input-level variation in how much computation an example requires. The result provides real-model evidence that computation depth should not necessarily be treated as a fixed architectural constant.

B.2.2 Controlled Progressive-Depth Study

To isolate the underlying phenomenon, we construct a controlled progressive-depth setting. The objective is to test whether variable optimal depth can arise even when the data-generating process is known and controlled, rather than being an artifact of a particular real benchmark.

Each sample x_i has a binary label $y_i \in \{0, 1\}$ and a hidden required depth L_i^*. The required depth represents the first layer at which the sample contains enough evidence for a reliable prediction. For a model with L layers, each intermediate layer $\ell \in \{1, \ldots, L\}$ produces a prediction probability

$$p_{i,\ell} = f_\ell(x_i), \tag{31}$$

where f_ℓ denotes the model's prediction head applied at depth ℓ.

The synthetic evidence is organized in three phases:

- Before the required depth, $\ell < L_i^*$, the sample provides only weak evidence, so early predictions are uncertain.
- At the required depth, $\ell = L_i^*$, the sample provides a strong identifying signal for the label.
- After the required depth, $\ell > L_i^*$, additional layers receive zero-mean refinement noise, so deeper computation is not guaranteed to improve the prediction.

For each intermediate prediction, we compute binary cross-entropy

$$\mathrm{BCE}(y_i, p_{i,\ell}) = -y_i \log p_{i,\ell} - (1 - y_i) \log(1 - p_{i,\ell}). \tag{32}$$

An oracle chooses the best exit depth for each sample by minimizing this loss:

$$\hat{d}_i = \arg \min_{\ell \in \{1,\ldots,L\}} \mathrm{BCE}(y_i, p_{i,\ell}). \tag{33}$$

Here, $\hat{d}_i$ is the empirically best depth for sample i, which may differ from the hidden required depth L_i^* because the model's intermediate predictions can be affected by noise and imperfect learning.

We compare this oracle variable-depth policy with fixed full-depth computation. Let N be the number of samples. The full-depth loss is

$$\mathcal{L}_{\text{full}} = \frac{1}{N} \sum_{i=1}^{N} \mathrm{BCE}(y_i, p_{i,L}), \tag{34}$$

and the oracle variable-depth loss is

$$\mathcal{L}_{\text{oracle}} = \frac{1}{N} \sum_{i=1}^{N} \mathrm{BCE}(y_i, p_{i,\hat{d}_i}). \tag{35}$$

We report oracle gain as

$$\text{Gain} = \mathcal{L}_{\text{full}} - \mathcal{L}_{\text{oracle}}. \tag{36}$$

A positive gain means that choosing the best depth per sample improves prediction quality relative to always using the final layer.

We evaluate 4-, 8-, and 12-layer models; three difficulty distributions; 32,768 samples per condition; and nine total model/difficulty conditions. The uniform difficulty distribution spreads required depths across the full model, the early-heavy distribution places more mass on shallower required depths, and the late-heavy distribution places more mass on deeper required depths.

In Table 13, "Unique best depths" counts how many distinct values appear among the oracle-selected depths $\{\hat{d}_i\}_{i=1}^{N}$. "Best-depth mean $\pm$ std" reports the mean and standard deviation of the oracle-selected depth distribution. "Oracle gain" is the BCE improvement of the oracle variable-depth policy over fixed full-depth computation.

Table 13: Controlled progressive-depth results. Oracle gain is measured relative to fixed full-depth computation in BCE.

Model	Difficulty	Unique best depths	Best-depth mean $\pm$ std	Oracle gain
4L	Uniform	4	3.24 ± 0.98	0.055
4L	Early-heavy	4	2.97 ± 1.12	0.054
4L	Late-heavy	4	3.53 ± 0.82	0.045
8L	Uniform	8	6.23 ± 2.00	0.106
8L	Early-heavy	8	5.64 ± 2.34	0.102
8L	Late-heavy	8	6.89 ± 1.70	0.088
12L	Uniform	12	9.24 ± 2.99	0.143
12L	Early-heavy	12	8.33 ± 3.54	0.141
12L	Late-heavy	12	10.23 ± 2.58	0.121

Every condition contains multiple distinct optimal depths. For the 12-layer uniform-difficulty model, all 12 candidate depths are selected by at least some samples and $\text{Std}(L^*) = 2.99$. The oracle variable-depth policy consistently improves over fixed full-depth computation, with gains ranging from 0.045 to 0.143 BCE.

Across model depths and difficulty regimes, different samples have different optimal computation depths, and adapting depth can improve both computation efficiency and prediction quality. This motivates the second design principle of FlexUMM: computation depth should be determined by the maturity of each input rather than fixed globally.

C Additional Ablation and Policy Analyses

C.1 Remaining Design Choices

Discrete versus graded specialization. Equation (2) switches discretely between shared and specialized computation rather than interpolating between them. Table 14 compares this discrete gate with a graded alternative that mixes the shared and specialized outputs by a continuous weight, and also ablates the action-mixture target inside $\mathcal{L}_{\text{gate}}$, the expert count E, and the routing width k. Removing the action-mixture target tests the failure mode it is meant to prevent: gate saturation, where the shared expert is abandoned and the model degenerates into a fully specialized backbone that has lost cross-functional transfer.

Expert count and routing width. Widening the pool to $E = 6$ experts edges out the deployed $E = 4, k = 2$ configuration on every benchmark in Table 14, but FlexUMM is designed to be a lightweight adapter, and $E = 6$ adds proportionally more trained parameters than $E = 4$'s 0.381% overhead (Table 7) for a marginal accuracy gain. We keep $E = 4$ as the deployed setting and read the $E = 6$ result as headroom rather than a reason to grow the added parameter budget. Shrinking the pool to $E = 2, k = 2$ instead removes the shared-versus-specialized choice at each routed layer, which costs more than widening does, confirming that a nontrivial expert pool is needed for the router to specialize meaningfully.

Table 14: Capacity-axis design ablations on Bagel. "Rel. compute" is the normalized route/action proxy defined in Section A.7.

Variant	Rel. compute	MMMU	MMBench-CN	MMBench-EN	MathVista	MMStar
Graded mixing instead of discrete gate	0.987	54.8	82.4	85.2	70.2	69.6
No action-mixture target in $\mathcal{L}_{\text{gate}}$	0.988	55.2	82.0	84.0	72.2	70.4
$E = 4, k = 1$	0.958	54.0	85.87	86.75	73.4	70.93
$E = 2, k = 2$	0.978	54.4	82.0	84.8	72.4	70.2
$E = 6, k = 2$	0.951	55.2	86.6	87.1	73.6	74.4

C.2 Early-Exit Subset Audit

The main paper already reports the overall selected-depth distribution, so we do not repeat that plot here. Instead, we analyze the rare early-exit cases in the same four-GPU lightweight screen. The screen produced 16 raw routes with a selected depth below 28. Four-way circular choice evaluation creates duplicate MMBench routes, leaving 10 unique inputs after deduplication: three MMMU inputs, five MathVista inputs, and two MMBench-English inputs.

Every unique early case selected depth 26; no case selected depth 22 or 24. This is a targeted audit of the early-exit tail, not a representative estimate of the full benchmark distribution. The evaluators do not provide a common per-example correctness field across all three suites, so the analysis focuses on the input and router signals rather than claiming early-exit accuracy.

C.2.1 What receives an early exit?

Table 15 lists every unique early case. The subset is concentrated in a few visually grounded or short-answer task families: MathVista icon-style questions account for four of the five MathVista cases, while the MMMU cases come from biology and chemistry. Question length is not uniform: the subset contains very short 21-character questions as well as a 726-character biology question. Thus the observed early exits cannot be explained by a simple short-question rule.

Table 15: Unique early-exit cases in the lightweight screen. MMBench circular-choice copies are deduplicated. The router selected depth 26 in every case; the signals are recorded at the selected depth.

Suite	Case	Task	Chars.	Depth	Halt@26	Risk@26	Regret@26
MMBench-EN	493	MMBench-English	66	26	0.581	0.091	0.038
MMBench-EN	630	MMBench-English	41	26	0.640	0.081	0.030
MMMU	Biology_21	biology	218	26	0.511	0.098	0.045
MMMU	Biology_8	biology	726	26	0.506	0.088	0.035
MMMU	Chemistry_30	chemistry	257	26	0.505	0.090	0.037
MathVista	129	iconqa_integer	104	26	0.523	0.087	0.034
MathVista	157	iconqa_integer	21	26	0.524	0.097	0.044
MathVista	163	iconqa_integer	21	26	0.535	0.096	0.043
MathVista	187	iconqa_integer	21	26	0.535	0.097	0.043
MathVista	92	vqa2_0_text	60	26	0.602	0.093	0.040

Figure 8 summarizes the subset composition and plots question length against the halt probability at the selected depth. All points lie at or above the router's 0.5 threshold, while their marker size encodes the residual-risk upper bound. The small sample and repeated task families make this an audit of behavior, not a statistical generalization.

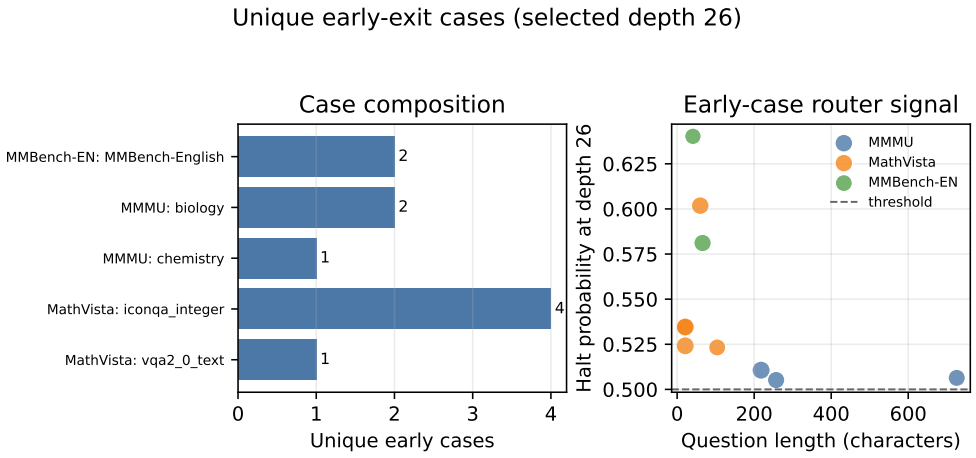


Figure 8: Early-exit subset composition and router signal. Each point is a unique input selected at depth 26; marker size is proportional to the residual-risk upper bound.

C.2.2 Router traces around the early decision

Figure 9 follows the complete router trace for each unique early case. The individual traces show a consistent pattern: the risk upper bound falls from depths 22 to 26, while halt probability rises and crosses the 0.5 decision threshold at depth 26. The router therefore exits when its own risk signal has decreased sufficiently, rather than because the input was assigned a fixed shallow depth in advance. Since these traces stop once the policy exits, there is no depth-28 point for an early case.

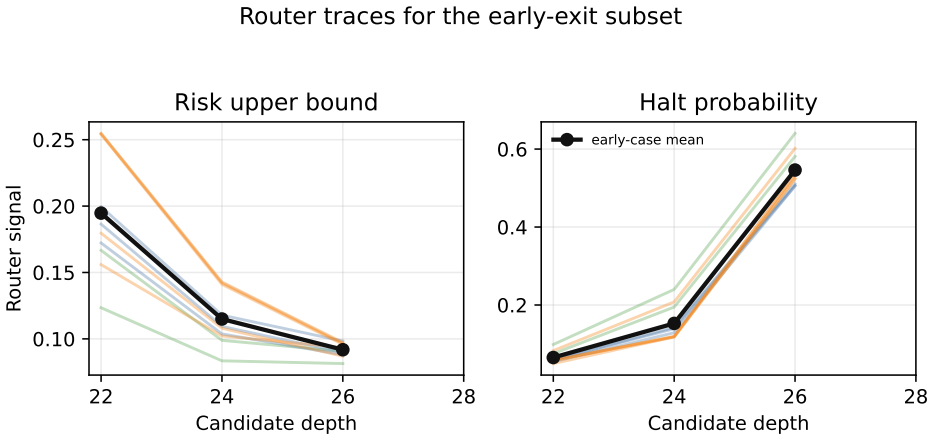


Figure 9: Candidate-depth router traces for the unique early-exit subset. Faint lines are individual cases and the black line is their mean at each evaluated candidate depth.

C.2.3 Why the gate passes for early cases

The implementation requires both a halt probability of at least 0.5 and a risk upper bound of at most 0.1 before exiting. We compare these two gate signals at depth 26 for every deduplicated input in the screen. As shown in Figure 10 and Table 16, all 10 early cases satisfy both conditions, whereas none of the 740 full-depth cases does. Only 5.0% of full-depth cases cross the halt threshold alone, and their risk signal remains too high. This provides a more mechanistic explanation for the rare early exits than question length or task identity by itself.

Table 16: Depth-26 gate-condition comparison. "Halt pass" uses the configured halt threshold 0.5; "Joint pass" requires both halt probability ≥ 0.5 and risk upper bound ≤ 0.1.

Group	N	Mean halt@26	Mean risk@26	Halt pass	Joint pass
early	10	0.546	0.092	100.0%	100.0%
full	740	0.424	0.132	5.0%	0.0%

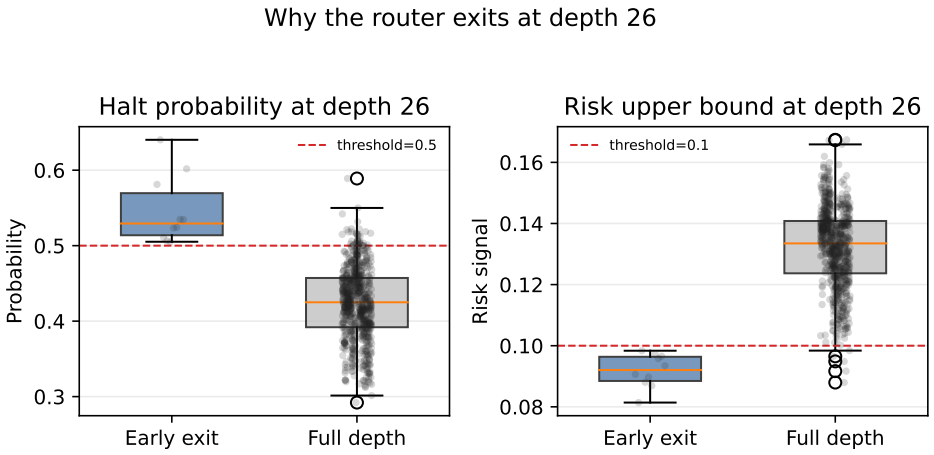


Figure 10: Depth-26 gate signals for early-exit and full-depth inputs. The dashed lines show the configured halt-probability and risk thresholds.

Interpretation. The early-exit subset provides concrete evidence that the learned policy is active on individual understanding inputs, but it also clarifies the current checkpoint's operating point: early exits are rare and all occur at depth 26. The result supports input-conditioned routing as a mechanism while motivating future calibration or threshold studies if broader compute savings are desired.

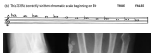
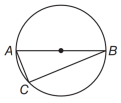
D Extended Case Study

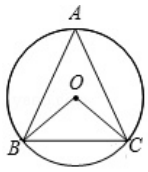
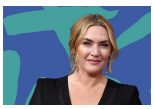
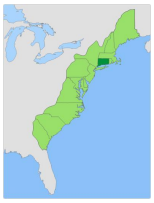
This appendix gives the full selected qualitative set referenced in Section 4.8. The examples are strict paired wins: the FlexUMM answer or image is correct under the benchmark evaluator and the baseline comparator is not. They are selected for auditability and diversity, and are not intended to replace aggregate benchmark results.

D.1 Understanding cases

Table 17 lists all ten paired understanding examples. The MMBench entries are representative displayed rows from the four-way circular-group evaluation used in the paired output analysis.

Table 17: Complete selected understanding set. Each row includes the benchmark input image and the paired answers. The FlexUMM answer agrees with the ground truth and the baseline answer does not.

#	Input image	FlexUMM	Baseline	GT
1		A	B	A
2		D	B	D
3		8.5	7.5	8.5

#	Input image	FlexUMM	Baseline	GT
4		100°	130°	100°
5		C	B	C
6		A	B	A
7		B	A	B
8		B	C	B
9		D	C	D
10		B	C	B

D.2 Generation cases

Figure 11 shows six distinct selected generation wins spanning color, object identity, and spatial-relation constraints. The set includes one sample for each of the repeated red-dog and white-scissors prompts, while omitting the purple-bear samples. Within each tile, the FlexUMM image is on the left and the baseline comparator is on the right. The pairs were generated with identical prompts and evaluation settings, and the detector outcome is summarized in Table 18.

E Limitations

Our evaluation considers a finite set of candidate depths for each backbone, which limits the granularity of the available quality-compute operating points. In addition, our experiments focus on a limited set of unified multimodal backbones and model scales. Extending the evaluation to finer-grained depth configurations and a broader range of architectures could further characterize the flexibility and applicability of the proposed framework.

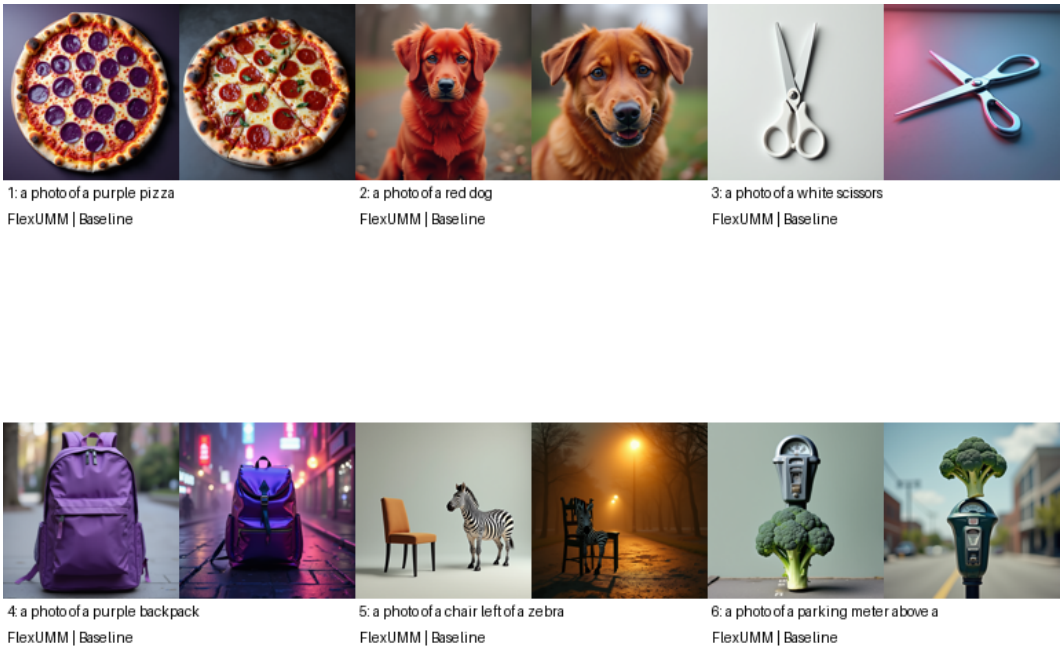


Figure 11: Selected generation set. Each tile contains the FlexUMM image followed by the paired baseline image.

Table 18: Detector outcomes for the selected generation set. The rows are individual image-level wins from distinct prompts.

#	Prompt	FlexUMM result	Baseline failure mode
1	purple pizza	Pass	No purple pizza; red pizza detected
2	red dog	Pass	Brown dog rather than red
3	white scissors	Pass	Purple scissors rather than white
4	purple backpack	Pass	Suitcase/handbag detected, not backpack
5	chair left of a zebra	Pass	Bench detected; no chair
6	parking meter above a broccoli	Pass	Parking meter below the broccoli