# FOLDUMM: UNLOCKING EFFICIENT INFERENCE IN UNIFIED MULTIMODAL MODELS VIA TASK-SPECIFIC LAYER FOLDING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Unified Multimodal Models (UMMs) couple image understanding and text-to-image generation within a shared Transformer backbone, but this shared design introduces substantial inference overhead. Through layer-wise redundancy profiling, we observe a structural divergence between the two tasks: understanding exhibits concentrated redundancy in deeper layers, whereas generation is more sensitive to depth reduction and lacks continuous redundant blocks. This task-dependent redundancy makes static layer pruning prone to cross-task degradation when transferred to UMMs. We propose FOLDUMM, a task-aware layer-folding framework that realizes a "One Backbone, Multiple Paths" inference paradigm. FOLDUMM formulates layer folding as a constrained shortest-path problem on a directed acyclic graph, and dynamic programming identifies task-specific folding paths under a target pruning budget. Lightweight STITCH LORA modules are further inserted at fold points to repair representation mismatch. Experiments on JANUS-PRO and OMNIGEN2 show that FOLDUMM achieves a more favorable performance–efficiency trade-off, preserving substantially higher understanding and generation performance under structured pruning while providing practical inference speedup. Our code is available at `https://anonymous.4open.science/r/FoldUMM/`.

## 1 INTRODUCTION

Historically, multimodal understanding and generation evolved as distinct fields relying on independent, specialized models. However, the field is rapidly converging toward Unified Multimodal Understanding and Generation Models (UMMs) (Zhang et al., 2025b). Representative works, such as BAGEL (Deng et al., 2025), OVIS-U1 (Wang et al., 2025), JANUS-PRO (Chen et al., 2025), and OMNIGEN2 (Wu et al., 2025), integrate diverse input/output interfaces and complex processing modules into a single system to handle "any-to-any" modality transformations. While this architectural integration enables powerful cross-modal capabilities, it inevitably leads to massive parameter counts (e.g., EMU3.5 (Cui et al., 2025) at 34B) and intricate computational graphs. Consequently, these models suffer from the inherent challenges of large-scale foundation models: prohibitive computational overhead and high inference latency, which severely constrain their practical deployment. For instance, JANUSFLOW (Ma et al., 2025b) incurs $\sim$7.4 TFLOPs per image.

Contemporary UMMs fall into two paradigms (Figure 2): **Decoupled** architectures (e.g., OMNIGEN2) that utilize external generators, and **Natively Unified** architectures (e.g., JANUS-PRO) that integrate all modalities into a single end-to-end Transformer. Despite this structural divergence, our empirical profiling reveals a critical consensus: the massive Transformer backbone remains the primary optimization target.[1] It accounts for over 85% of the total parameter count and contributes at least 88.8% of T2I latency in the profiled natively unified models. Therefore, the shared backbone is the central target for generalizable UMM optimization.

To determine *how* to optimize this shared backbone, we ground our approach in UMMs' internal mechanisms. We identify an intrinsic impedance mismatch in current designs: while architectural

[1] See Appendix E.1 for profiling details.



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

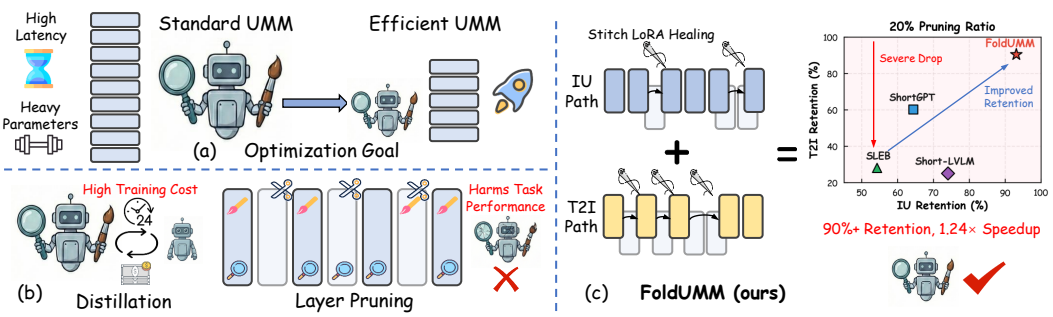



Figure 1: **FOLDUMM Paradigm.** (a) **Goal:** efficient UMM inference while preserving IU and T2I capabilities. (b) **Existing methods:** distillation incurs costly retraining, and static pruning uses one compressed path for all tasks, causing negative transfer. (c) **FOLDUMM:** one backbone supports task-specific folded paths with localized fold-point healing.

unification maximizes parameter efficiency via shared representations, it imposes a suboptimal, uniform inference topology on functionally distinct tasks (specifically, understanding versus generation). These tasks exhibit distinct boundaries categorized by their output modalities. We posit that the shared backbone naturally develops a functional differentiation similar to the Vision Function Layers (VFLs) observed in VLMs (Shi et al., 2025), but specialized for the distinct nature of understanding and generation. This functional divergence implies that a unified parameter space does not necessitate a uniform computational path. To align with this intrinsic structure, we propose a *"One Backbone, Multiple Paths"* inference paradigm, where the UMM's topology dynamically adapts to the specific functional requirements of each task.

However, conventional compression techniques fail to align with this dynamic nature. While knowledge distillation (Mansourian et al., 2025) and quantization (Li et al., 2024) involve prohibitive training costs or lack structural flexibility, *layer pruning* (Ashkboos et al., 2024) offers a promising avenue for immediate acceleration. Yet, standard pruning methods remain static, searching for a single path for all tasks. As shown in Figure 3, our empirical analysis reveals a critical redundancy divergence: while Image Understanding (IU) tasks tend to exhibit high redundancy in deeper layers, aligning with findings in LLMs (Gromov et al., 2025), Text-to-Image (T2I) generation displays a distinct, volatile redundancy pattern where removable parameters are distributed across intermediate or earlier layers rather than clustered at the end.[2] Consequently, enforcing a static pruning path inevitably leads to *negative transfer* (Zhang & Yang, 2022), where the topology optimized for one severely degrades the other.

To resolve this conflict, as illustrated in Figure 1, we introduce FOLDUMM, a *Task-Aware Layer Folding* framework that realizes the "One Backbone, Multiple Paths" paradigm. **To our knowledge, FOLDUMM is the first task-aware inference optimization framework for Unified Multimodal Models**, enabling one shared backbone to support multiple task-specific folded paths for understanding and generation. Unlike fixed-path static pruning, FOLDUMM treats layer selection as a constrained Shortest Path Problem on a Directed Acyclic Graph (DAG). By utilizing Dynamic Programming (DP), it identifies the globally optimal folding strategy for each task type under the defined additive representation-discrepancy objective. Furthermore, to mitigate the feature misalignment caused by discrete layer removal, we equip each task-specific path with STITCH LORA, a localized fold-point healing module that offers a lightweight post-hoc recovery mechanism with minimal parameter overhead.[3] Extensive experiments across both UMM paradigms demonstrate that FOLDUMM achieves a stronger performance–efficiency trade-off than the evaluated static pruning baselines, improving backbone token throughput by 1.22×–1.24× while retaining over 90% relative performance on both IU and T2I at 20% pruning.

[2]At a 10% pruning rate, we observe that the optimal folding paths for IU and T2I, derived via our post-hoc redundancy analysis, exhibit only 20% overlap.

[3]The healing sets are about 0.9% of JANUS-PRO-7B's SFT budget and 0.1% of OMNIGEN2's T2I data.

**Our main contributions are as follows:**

- We identify the shared backbone as the critical computational bottleneck in UMMs and reveal a structural conflict where IU and T2I tasks rely on divergent redundancy patterns, rendering static pruning ineffective.
- We propose FOLDUMM, which uses graph-based dynamic programming to identify task-specific paths globally optimal under an additive representation-discrepancy objective, together with STITCH LORA for lightweight post-hoc recovery.
- Experiments on both decoupled and natively unified UMM paradigms confirm that FOLDUMM consistently outperforms the evaluated static pruning baselines, achieving stronger performance–efficiency trade-offs while preserving performance across the evaluated IU and T2I tasks.

## 2 PRELIMINARIES

### 2.1 UNIFIED MULTIMODAL MODELS (UMMS): TAXONOMY AND EVOLUTION

Unified Multimodal Models (UMMs) represent a paradigm shift from specialized expert systems to general-purpose architectures capable of processing various input and output modalities within a single framework (Zhang et al., 2025b). Formally, given a multimodal input sequence $\mathbf{X} = \{x_1, x_2, \ldots, x_T\}$ containing interleaved text and visual tokens, a UMM models the conditional probability distribution of the target sequence $\mathbf{Y}$ as $P(\mathbf{Y} \mid \mathbf{X}) = \prod_{i=1}^{M} P(y_i \mid y_{<i}, \mathbf{X})$, where $M = |\mathbf{Y}|$ is the target sequence length and $y_{<i} = \{y_1, \ldots, y_{i-1}\}$ denotes all preceding tokens. Based on the depth of modality fusion and the degree of parameter sharing, we categorize contemporary UMMs into two primary architectural paradigms: **Decoupled** and **Natively Unified** (hereafter Unified). Despite their structural differences, both center computation in a massive Transformer backbone.

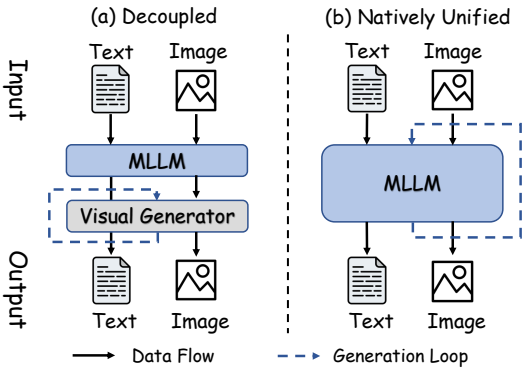



Figure 2: Taxonomy of UMM Architectures. (a) Decoupled models rely on external generators; (b) Unified models integrate all tasks into a single massive Transformer backbone.

**Decoupled Paradigm.** This approach bridges independent encoders and decoders via a central LLM, as depicted in Figure 2(a). Image understanding (IU) uses specialized encoders (e.g., SIGLIP (Zhai et al., 2023)), while text-to-image (T2I) generation relies on external decoders (e.g., VQ-VAE (Van Den Oord et al., 2017) or Diffusion Transformers (DiT) (Peebles & Xie, 2023)). Representative models include OMNIGEN2 (Wu et al., 2025) and OVIS-U1 (Wang et al., 2025). While modularity leverages specialized priors, it creates a **task-dependent bottleneck**: the backbone bears the primary computational load during IU tasks, whereas the external generator dominates latency during T2I tasks (e.g., executing heavy denoising). We utilize OMNIGEN2 to validate our proposal on this architecture, as it features a predominantly frozen MLLM backbone conditioning a separate DiT generator, serving as a quintessential example of this family.

**Unified Paradigm.** This approach achieves deeper integration by encoding all modalities into a shared representation space within a monolithic Transformer backbone (Figure 2(b)). Critically, the backbone directly participates in generation at every step, rather than delegating to an external decoder. Representative models include BAGEL (Deng et al., 2025), JANUS-PRO (Chen et al., 2025), and JANUSFLOW (Ma et al., 2025b). This fully shared parameter space creates an **integrated bottleneck**, where the massive backbone can dominate >70% of latency.[4] For this paradigm, we adopt JANUS-PRO as our primary testbed, as its fully autoregressive nature presents the ideal platform to address the integrated bottleneck.

[4]See Appendix E.1 for the latency decomposition.

**Efficiency Bottleneck.** Our profiling shows that the shared MLLM backbone is a key optimization target for improving TTFT and token throughput in UMM inference, although end-to-end T2I latency can also be dominated by external generators in decoupled architectures. This motivates our focus on task-aware backbone folding rather than unstructured sparsity alone.

### 2.2 Existing Efficiency Techniques and Critical Gap

Despite the clear bottleneck, generic efficiency techniques falter in the UMM context.[5] *Weight pruning* methods (e.g., SparseGPT (Frantar & Alistarh, 2023), Wanda (Sun et al., 2024)) produce unstructured sparsity that fails to yield practical speedups on standard GPUs (Wen et al., 2016; Gale et al., 2020). More critically for UMMs, pruning catastrophically degrades visual generation quality (Zhu et al., 2026; Kim et al., 2024), as fine-grained weights encode essential texture details. Mixture-of-Experts (MoE) architectures (e.g., Uni-MoE-2.0-Omni (Li et al., 2025), DeepSeek-VL2 (Wu et al., 2024)) reduce per-token computation but incur prohibitive training costs and 4–8$\times$ VRAM overhead (Cai et al., 2025), limiting their applicability.

Static layer pruning approaches (e.g., ShortGPT (Men et al., 2025), SLEB (Song et al., 2024), Short-LVLM (Ma et al., 2025a)) identify a single optimal path shared across all tasks during inference. This works for conventional LLMs where different tasks share similar computation patterns, but fails in UMMs due to **redundancy divergence**: image understanding and text-to-image generation exhibit fundamentally different layer-level redundancy patterns (empirically validated in Section 3.1 and Figure 3). A single static pruning path inevitably causes negative transfer (Zhang & Yang, 2022). To resolve this, we propose **Task-Aware Layer Folding**, which discovers task-specific optimal paths within the shared backbone.

### 2.3 Problem Formulation: Task-Aware Layer Folding

We formulate *task-aware layer folding* as a constrained path optimization problem on the Transformer backbone. Let a backbone consist of $N$ layers indexed by $\mathcal{N} = \{0, 1, \ldots, N-1\}$. Each layer $\ell \in \mathcal{N}$ implements a transformation:

$$\mathbf{h}_\ell = f_\ell(\mathbf{h}_{\ell-1}; \boldsymbol{\theta}_\ell), \tag{1}$$

where $\mathbf{h}_\ell \in \mathbb{R}^d$ and $\boldsymbol{\theta}_\ell$ represents the hidden state and the trainable parameters at layer $\ell$, respectively. In this work, we define the task space as $\mathcal{T} = \{t_{\text{IU}}, t_{\text{T2I}}\}$, representing the two representative tasks in UMMs: Image Understanding and Text-to-Image generation. Each task $t \in \mathcal{T}$ operates on a dataset $\mathcal{D}_t$ with a specific objective $\mathcal{L}_t$. For our framework, we further partition the data into: (1) a *calibration set* $\mathcal{D}_t^{\text{cal}}$ for measuring layer redundancy, and (2) a *healing set* $\mathcal{D}_t^{\text{heal}}$ for training StitchLoRA modules.

Given a target pruning rate $\alpha \in [0, 1)$, we denote the number of retained layers as $K_\alpha = \lceil (1-\alpha)N \rceil$. For a target task $t \in \mathcal{T}$, we define a *folding path* as a layer-ordered subset:

$$\mathcal{P}_t = \{\ell_1, \ell_2, \ldots, \ell_{K_\alpha}\} \subset \mathcal{N}, \tag{2}$$

where $\ell_1 < \ell_2 < \cdots < \ell_{K_\alpha}$. During inference, data flows exclusively through the layers in $\mathcal{P}_t$, effectively "folding"(skipping) all layers not in $\mathcal{P}_t$ from the computational graph. Our goal is to identify the optimal task-specific path $\mathcal{P}_t^*$ that minimizes the expected loss for task $t$:

$$\mathcal{P}_t^* = \arg\min_{\mathcal{P}_t} \mathcal{L}_t(\mathcal{P}_t; \boldsymbol{\Theta}) \quad \text{subject to } |\mathcal{P}_t| = K_\alpha, \tag{3}$$

where $\boldsymbol{\Theta}$ represents the full model parameters, and $\mathcal{L}_t(\mathcal{P}_t; \boldsymbol{\Theta})$ denotes the task loss evaluated on dataset $\mathcal{D}_t$ when using folding path $\mathcal{P}_t$. This formulation allows $\mathcal{P}_{t_{\text{IU}}}$ and $\mathcal{P}_{t_{\text{T2I}}}$ to vary independently, enabling task-specific optimization within the same shared backbone.

## 3 Methodology

We propose FoldUMM, a novel framework that enables efficient inference for UMMs through task-aware layer folding. Moving beyond the static pruning paradigm, FoldUMM adopts an adaptive philosophy: allowing the model to dynamically determine where and how much to fold based

[5] Appendix I provides a broader review of related efficiency methods.

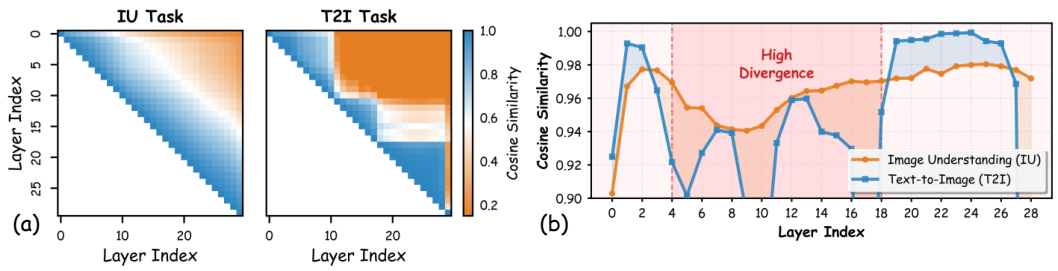



Figure 3: **Comparison of Redundancy Patterns.** (a) **Redundancy Heatmaps** visualize consistent block-wise redundancy for IU tasks versus fragmented patterns for T2I. (b) **Adjacent Similarity** further accentuates this divergence, highlighting the necessity for task-specific folding strategies.

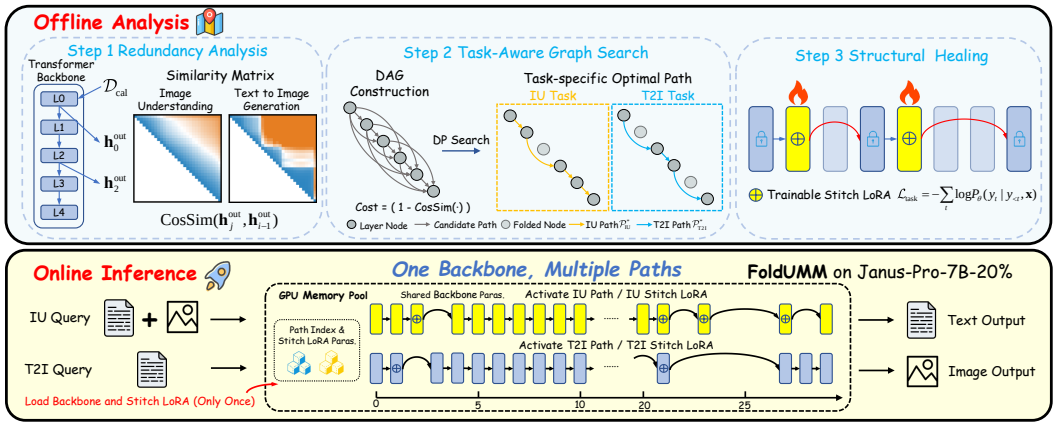



Figure 4: **Overview of the FOLDUMM Framework.** The pipeline comprises *Offline Analysis* and *Online Inference* phases. Offline, we perform (1) **Redundancy Profiling** and (2) **Graph Search** to identify optimal folding paths, followed by (3) **Structural Healing** via STITCH LORA, which re-aligns the feature manifold disrupted by layer folding. During inference, the system realizes the *"One Backbone, Multiple Paths"* paradigm, dynamically activating task-specific paths to improve backbone token throughput.

on task-specific costs rather than hard-coded rules. This enables a "One Backbone, Multiple Paths" architecture where different tasks traverse optimized, task-specific routes through the shared network. Consequently, this *simple yet effective* paradigm reconciles the trade-off between multimodal unification and efficiency, allowing FOLDUMM to achieve specialized model speedups while retaining the versatility of the shared backbone.

### 3.1 MOTIVATION: THE OPTIMIZATION DILEMMA

To investigate the limitations of static redundancy analysis in UMMs, we measure layer-wise cosine similarity on the JANUS-PRO backbone.[6] A high similarity score indicates that the layer performs a near-identity transformation and is potentially redundant. As clearly illustrated in Figure 3, this analysis reveals a fundamental structural *divergence* between task families:

- **IU Tasks:** Similar to findings in Large Language Models (LLMs), deeper layers (specifically the latter one-third) exhibit high redundancy. This suggests that once visual semantics are encoded, the final layers perform relatively shallow linguistic decoding.
- **T2I Tasks:** Conversely, T2I tasks (blue line) exhibit high *volatility*. Despite shallow redundancy and late plateaus, middle layers fluctuate drastically (e.g., drops at 4, 9, 16). This indicates generative tasks require specific feature transformations, lacking the continuous redundancy blocks observed in understanding tasks.

[6]Compared to metrics like activation magnitude, cosine similarity is a simple, effective proxy for redundancy, widely adopted in recent depth reduction studies (Gromov et al., 2025).

This observation is further validated by our folding strategy searching results (see Appendix G.2). A single static path optimized for one task becomes suboptimal for another, necessitating a **Task-Aware** strategy that derives task-specific folding paths.

### 3.2 FRAMEWORK OVERVIEW

FOLDUMM operates on the principle of **"One Backbone, Multiple Paths"**: we maintain a single physical backbone but derive task-specific optimal folding paths $\mathcal{P}_t^*$ for each task type. The framework consists of three stages. First, in **Offline Analysis**, we construct a Directed Acyclic Graph (DAG) representation of the backbone and measure the "folding difficulty" (edge cost) for every possible skip connection using calibration data, creating task-specific cost matrices. Second, in **Optimal Path Search**, we employ a constrained dynamic programming (DP) algorithm to discover the globally optimal folding path under the defined additive edge-cost objective for each task, balancing model compression and feature preservation. Third, in **Localized Fold-Point Healing**, we attach lightweight STITCH LORA modules at selected fold points to repair feature misalignment, providing a simple post-hoc recovery paradigm for folded any-to-any multimodal models.

A key advantage of this design is **low-overhead task switching with adaptive compression**: at inference time, the system routes inputs through the task-specific optimal path, activating only the necessary LoRA modules without reloading model weights. Crucially, different tasks can operate at *different pruning rates* to accommodate varying task complexity. For instance, more fragile generative tasks may require conservative folding while robust understanding tasks tolerate aggressive compression (see Section 4.3 for empirical analysis).

### 3.3 TASK-AWARE LAYER FOLDING VIA GRAPH SEARCH

To capture the sequential dependencies of layer selection and guarantee global optimality under the additive proxy objective, we formalize layer folding as a constrained shortest-path problem on a DAG, shifting from heuristic pruning rules to adaptive decision making. Unlike static approaches that impose hard rules (e.g., "prune top-$k$ redundant layers"), FOLDUMM evaluates feature distribution mismatch for every potential fold using a cosine-mismatch proxy. In regions of high redundancy (low cost), the algorithm naturally selects continuous folding to maximize efficiency; in critical regions (high cost), it automatically preserves intermediate layers to avoid excessive feature mismatch. Appendix D.1 evaluates angular and activation-based alternatives under the same search formulation.

**Graph Construction and Edge Cost.** We define a graph $G = (V, E)$ where nodes $V = \{0, 1, \ldots, N-1\}$ represent the $N$ Transformer layers. Edges $e_{i \to j}$ represent transitioning from layer $i$ to layer $j$. The cost of an edge is defined as the expected feature distribution mismatch:

$$\text{Cost}_{i,j}^{(t)} = \mathbb{E}_{\mathbf{x} \sim \mathcal{D}_t^{\text{cal}}} \left[ 1 - \text{CosSim}(\mathbf{h}_i^{\text{out}}, \mathbf{h}_{j-1}^{\text{out}}) \right]. \tag{4}$$

Here, a low cost implies that layer $i$'s output is already similar to what layer $j$ expects (safe to fold), while a high cost indicates significant transformation (requires healing). Crucially, we compute distinct cost matrices $C_{\text{IU}}$ and $C_{\text{T2I}}$ for each task.

**Dynamic Programming Algorithm.** We aim to find a path $\mathcal{P}^*$ that minimizes cumulative cost while retaining exactly $K_\alpha$ layers. Each non-adjacent transition in the selected path corresponds to a fold point that is later repaired by a localized STITCH LORA module.[7] We define the state $DP[j][k]$ as the minimum cost to reach layer $j$ having kept $k$ layers. The transition logic distinguishes two cases: (1) *Linear Edge* ($j = i + 1$), representing standard propagation with cost solely from edge mismatch (often near zero for adjacent layers); and (2) *Folding Edge* ($j > i + 1$), representing layer skipping that introduces a fold point for structural healing. The algorithm explores all valid predecessors $i$ to find the optimal substructure, guaranteeing global optimality for the specified additive edge cost and retained-layer budget.

[7]The complete pseudocode is provided in Appendix G.1.

### 3.4 LOCALIZED FOLD-POINT HEALING VIA STITCH LORA

Layer folding removes intermediate transformations, causing a distributional mismatch between the output of layer $i$ and the expected input of layer $j$. To bridge this gap, we use STITCH LORA as a localized fold-point healing module, inserted only at selected folding edges rather than uniformly across the backbone. For a folding edge $i \to j$, we attach a LoRA module to layer $i$, modifying its output as:

$$\tilde{\mathbf{h}}_i = f_i(\mathbf{h}_{i-1}; \boldsymbol{\theta}_i + \Delta\boldsymbol{\theta}_{\text{LoRA}}). \tag{5}$$

This adapted output $\tilde{\mathbf{h}}_i$ is then fed directly into layer $j$, bypassing the removed intermediate layers. The LoRA module learns to compensate for the missing transformations, effectively aligning the feature distributions across the fold. We train STITCH LORA using task-specific supervision: standard cross-entropy loss for IU tasks, and distillation objectives (KL divergence or MSE) for T2I tasks to align with the dense teacher.

**Advantages.** This design offers three distinct benefits: **(1)** ***Inference-Efficiency***, as the LORA weights can be merged into the backbone weights during inference, adding zero latency overhead compared to standard layer execution; **(2)** ***Parameter-Efficiency***, requiring the training of only a tiny fraction of total parameters; and **(3)** ***Task-Isolation***, as maintaining separate LORA module sets for IU and T2I prevents interference between task-specific adaptations.

## 4 EXPERIMENTS

### 4.1 EXPERIMENTAL SETUP

**Models and Implementation.** We evaluate FOLDUMM on two representative architectures: the unified JANUS-PRO-7B (Chen et al., 2025) and the decoupled OMNIGEN2 (Wu et al., 2025). For optimization, we utilize 256 calibration samples per task ($|\mathcal{D}_t^{\text{cal}}| = 256$) and healing datasets $\mathcal{D}_t^{\text{heal}}$ of 30K (IU, from The Cauldron (Laurençon et al., 2024)) and 15K (T2I, from ShareGPT4V (Chen et al., 2024b)) examples. STITCH LORA is configured with rank $r = 128$ and $\alpha = 256$. We apply STITCH LORA only at the fold points selected by the task-specific folding paths, keeping the healing module localized and parameter-efficient.[8] Appendix D.2 further analyzes healing-data composition.

**Baselines.** We compare FOLDUMM with representative structured layer pruning methods, including SHORTGPT (Men et al., 2025), SLEB (Song et al., 2024), and SHORT-LVLM (Ma et al., 2025a), as well as unstructured weight pruning methods, including WANDA (Sun et al., 2024) and SPARSEGPT (Frantar & Alistarh, 2023).[9] All methods are evaluated at a 20% pruning rate and adapted to both IU and T2I using the same calibration data as FOLDUMM; implementation details are in Appendix H. Matched SHORTGPT and SLEB use the same STITCH LORA healing as FOLDUMM; SHORT-LVLM retains its original SVD compensation.

**Evaluation Metrics.** For Image Understanding, we evaluate on MME (Chaoyou et al., 2025), POPE (Li et al., 2023), GQA (Hudson & Manning, 2019), MM-Vet-V2 (Yu et al., 2024), and MMMU-Pro (Yue et al., 2025). Text-to-Image, we evaluate on DPG-Bench (Hu et al., 2024) and GenEval (Ghosh et al., 2023). Across independent runs, we observe 95% confidence intervals within $\pm 3\%$ for IU tasks and $\pm 1\%$ for T2I tasks. Relative Performance (RP) is computed by normalizing each metric to the Dense model and averaging within each task group. For **Efficiency**, we report Time-To-First-Token (TTFT), Gen.(s) (per-image wall-clock time), and SpeedUp measured by backbone token-generation throughput relative to the dense model.

### 4.2 MAIN RESULTS

**Performance on JANUS-PRO.** Table 1 shows that FOLDUMM achieves the strongest performance–efficiency trade-off among structured pruning methods on JANUS-PRO-7B. At 20% pruning, it is the only evaluated structured method retaining over 90% relative performance on both IU and

[8] Training details and healing-data scaling are provided in Appendix G.3 and Appendix E.4.

[9] Unstructured pruning methods preserve strong quality but provide little practical speedup on standard GPUs. They are included mainly as quality-oriented references.

Table 1: **Main Results at 20% pruning. Recovery** marks post-pruning recovery; ✓for SHORTGPT/SLEB denotes matched STITCH LORA, while SHORT-LVLM uses its original compensation. **SpeedUp** is backbone token throughput, **Gen.(s)** is per-image wall-clock generation time, and **RP** is relative performance normalized to Dense. †Unstructured pruning is included for quality comparison.

| Method | Recovery | Image Understanding (IU) | | | | | | Text-to-Image (T2I) | | | Efficiency | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | MME | POPE | GQA | MM-Vet-V2 | MMMU-Pro | RP | DPG-Bench | GenEval | RP | TTFT(ms)↓ | Gen.(s)↓ | SpeedUp↑ |
| **JANUS-PRO-7B** *(Unified)* | | | | | | | | | | | | | |
| Dense (Original) | – | 1567 | 85.2 | 60.4 | 47.1 | 35.3 | 100% | 84.5 | 0.79 | 100% | 104.2 | 3.94 | 1.00× |
| WANDA† | | 1514 | 86.0 | 59.6 | 49.4 | 34.3 | 99.7% | 84.0 | 0.74 | 96.5% | 105.2 | 3.92 | ∼1.00× |
| SPARSEGPT† | | 1551 | 86.8 | 60.3 | 38.3 | 35.8 | 96.7% | 83.3 | 0.75 | 96.8% | 106.6 | 3.93 | ∼0.99× |
| SHORTGPT | | 468 | 79.8 | 41.1 | 19.1 | 31.7 | 64.4% | 57.9 | 0.41 | 60.2% | 85.9 | 3.17 | ∼1.22× |
| + *Stitch LoRA* | ✓ | 1496 | 86.9 | 55.2 | 27.1 | 29.8 | 86.2% | 79.3 | 0.61 | 85.5% | – | – | – |
| SLEB | | 779 | 46.4 | 37.7 | 12.4 | 27.6 | 54.2% | 29.5 | 0.17 | 28.2% | 86.4 | 3.17 | ∼1.21× |
| + *Stitch LoRA* | ✓ | 912 | 83.1 | 47.9 | 20.2 | 32.5 | 74.0% | 58.5 | 0.39 | 59.3% | – | – | – |
| SHORT-LVLM | ✓ | 1430 | 89.0 | 32.9 | 16.8 | 29.7 | 74.0% | 25.4 | 0.16 | 25.2% | 87.1 | **3.16** | ∼1.22× |
| ↪ **FOLDUMM** | ✓ | **1505** | **90.2** | **56.4** | **36.4** | **32.8** | **93.1%** | **82.3** | **0.66** | **90.5%** | **83.3** | **3.16** | **∼1.24×** |
| **OMNIGEN2** *(Decoupled)* | | | | | | | | | | | | | |
| Dense (Original) | – | 1941 | 90.9 | 59.5 | 48.1 | 35.7 | 100% | 83.7 | 0.79 | 100% | 34.2 | 30.10 | 1.00× |
| WANDA† | | 2000 | 90.6 | 59.3 | 45.0 | 35.2 | 98.9% | 83.6 | 0.78 | 99.3% | 43.5 | 30.19 | ∼0.99× |
| SPARSEGPT† | | 1930 | 90.7 | 59.9 | 50.1 | 34.5 | 100.1% | 83.1 | 0.79 | 99.6% | 38.1 | 30.11 | ∼1.00× |
| SHORTGPT | | 565 | 83.6 | 46.5 | 20.2 | 16.9 | 57.7% | 70.2 | 0.67 | 84.3% | 28.2 | **29.97** | ∼1.21× |
| + *Stitch LoRA* | ✓ | 1131 | 87.3 | 49.1 | 31.6 | 26.3 | 75.2% | 77.4 | 0.72 | 91.8% | – | – | – |
| SLEB | | 1557 | 89.9 | 36.6 | 32.3 | 23.6 | 74.8% | 73.6 | 0.69 | 87.6% | **27.8** | 30.12 | ∼1.22× |
| + *Stitch LoRA* | ✓ | **1958** | 89.4 | 52.5 | 30.9 | 32.1 | 88.3% | 81.9 | 0.74 | 95.8% | – | – | – |
| SHORT-LVLM | ✓ | 676 | 80.4 | 46.2 | 19.5 | 17.8 | 58.3% | 76.5 | 0.68 | 88.7% | 27.9 | 30.10 | ∼1.22× |
| ↪ **FOLDUMM** | ✓ | 1942 | **90.7** | **54.9** | **36.9** | **33.6** | **92.6%** | **82.1** | **0.75** | **96.5%** | 28.1 | 30.10 | **∼1.22×** |

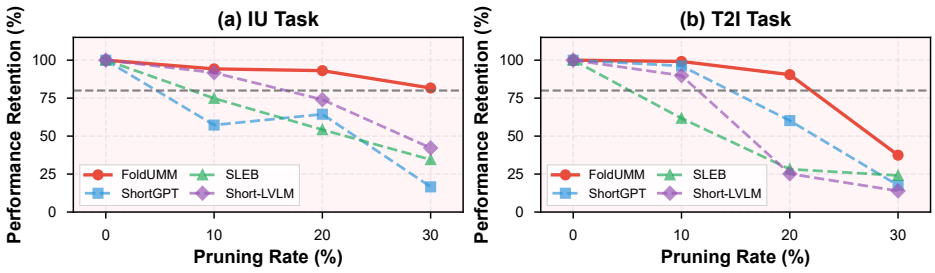



Figure 5: **Performance–pruning trade-off on JANUS-PRO-7B.** We report relative performance retention under 10%, 20%, and 30% pruning rates for Image Understanding (IU) and Text-to-Image generation (T2I). IU retention is averaged over five understanding benchmarks, while T2I retention is averaged over DPG-Bench and GenEval.

T2I, reaching 93.1%/90.5% RP while improving backbone throughput by 1.24×, close to the ideal Transformer-layer compute limit of 1.25× at this pruning rate. Complete image-generation latency decreases from 3.94 s to 3.16 s. Together, the throughput and latency results show that structural layer reduction translates into both backbone-level and complete-pipeline acceleration. Matched STITCH LORA improves SHORTGPT and SLEB, while FOLDUMM retains the strongest joint IU/T2I performance under the same pruning budget. MM-Vet-V2 remains the most pruning-sensitive IU benchmark; Appendices D.2 and E.2 analyze its healing-data sensitivity and pruning-rate behavior.

**Generalization to OMNIGEN2.** The lower block of Table 1 evaluates FOLDUMM on the architecturally different, decoupled OMNIGEN2. At 20% pruning, FOLDUMM retains 92.6% IU RP and 96.5% T2I RP while improving backbone throughput by 1.22×. As expected for a decoupled architecture dominated by an external generator, the structural efficiency gain is concentrated in the shared Transformer backbone. Together with JANUS-PRO, this result shows that task-specific folding transfers across integrated and decoupled UMM designs. Although matched-recovery SLEB is competitive on T2I, FOLDUMM provides the strongest combined IU/T2I retention.[10] Appendix C further examines cross-backbone generalization on BAGEL and EMU3.5.

### 4.3 PERFORMANCE-EFFICIENCY TRADE-OFF

Figure 5 shows a favorable retention region for FOLDUMM across pruning rates. At 20% pruning, it is the only structured method retaining over 90% performance on both tasks; at 30%, it still retains

[10]Additional performance analysis and baseline details are discussed in Appendix E.2.

Table 2: **Task-aware path ablation.** Cross-task transfer degrades the target task, confirming distinct IU and T2I dependencies on JANUS-PRO. ↓ marks large drops.

| Model | Path | MM-Vet-V2 | IU RP | GenEval | T2I RP |
|---|---|---|---|---|---|
| JANUS-PRO | IU Path | **36.4** | **93.1%** | 0.49↓ | 67.2% |
| | T2I Path | 32.1↓ | 82.1% | **0.66** | **90.5%** |
| OMNIGEN2 | IU Path | **36.9** | **92.6%** | 0.74 | 95.2% |
| | T2I Path | 35.7 | 91.3% | **0.75** | **96.5%** |

Table 3: **Ablation on Search Strategy (JANUS-PRO).** All variants use task-specific paths and STITCH LORA healing at 20% pruning; they differ only in path selection. **Rel.(%)** reports the performance of other strategies relative to FOLDUMM on the MM-Vet-V2 and GenEval.

| Search Strategy | Selection Mechanism | MM-Vet-V2 | GenEval | Rel.(%) |
|---|---|---|---|---|
| Uniform | Deterministic (every $k$-th) | 13.6 | 0.24 | 37.4 / 36.4 |
| Random | Stochastic Sampling | $15.4_{\pm 3.4}$ | $0.18_{\pm 0.13}$ | 42.3 / 27.3 |
| Task-specific Greedy | Local BI Minimization | 29.7 | 0.62 | 72.6 / 93.9 |
| DAG-based (Ours) | Global DP Optimization | 36.4 | 0.66 | 100 / 100 |

81.7% IU RP. The dashed 80% line marks severely degraded configurations. T2I is more sensitive to aggressive pruning, motivating task-specific compression rates rather than one shared schedule.

IU and T2I paths can also use different pruning rates; Appendix F reports mixed schedules, switching time, and memory cost.

### 4.4 ABLATION STUDIES

We ablate task-aware paths, global search, and localized healing at 20% pruning.

Table 4: **Stitch LoRA vs Full LoRA** on JANUS-PRO.

| Strategy | Params | MM-Vet-V2 | GenEval |
|---|---|---|---|
| No LoRA | 0 | 23.2 | 0.42 |
| Full LoRA | 480M | 37.3 | 0.69 |
| STITCH LORA | 60M | 36.4 | 0.66 |

Table 2 confirms the value of task-aware paths on JANUS-PRO. Table 4 further shows that localized STITCH LORA approaches Full LoRA performance with 60M versus 480M trainable parameters. To isolate each design choice under the same 20% pruning budget, we add one component at a time and report MM-Vet-V2/GenEval: SHORTGPT gives 19.1/0.41; adding STITCH LORA yields 27.1/0.61; additionally selecting separate IU/T2I paths yields 29.7/0.62; and replacing local BI selection with global DAG-DP yields 36.4/0.66. This sequence attributes the successive gains to healing, task-specific paths, and global folding-path optimization. The final step holds the Block-Influence criterion, pruning budget, and healing setup fixed. Appendix D.3 provides the complete controlled analysis.

## 5 CONCLUSION

We proposed FOLDUMM, a task-aware layer folding framework that addresses the efficiency bottleneck in Unified Multimodal Models. Driven by our discovery of the redundancy divergence between understanding and generation, FOLDUMM implements a "One Backbone, Multiple Paths" architecture that formalizes layer selection as a DAG-based shortest-path problem. Coupled with STITCH LORA for structural healing, our framework achieves practical speedups while preserving strong performance across representative UMM architectures. By systematically characterizing task-specific redundancy and translating it into a deployable layer-pruning algorithm, this work contributes both a practical efficiency method and empirical evidence of functional differentiation between understanding and generation within shared UMM backbones. Future work includes exploring finer-grained redundancy patterns, integrating with orthogonal techniques, and designing dynamic serving algorithms.[11]

[11] Limitations and future deployment directions are discussed in Appendix B and Appendix F.

### Reproducibility Statement

The method, calibration protocol, search algorithm, task-specific folding paths, healing objectives, optimization settings, and baseline adaptations are documented in the main paper and Appendix A. Code and configuration files are provided through the anonymous repository linked in the abstract. The appendix also reports additional cost-metric analyses, generalization evidence, and qualitative comparisons that support independent verification of the main claims.

### AI Use Statement

Generative AI tools assisted with experiment planning, result interpretation, code review, evidence organization, and editing the manuscript for structure and readability. The authors reviewed the AI-assisted work, checked all reported claims and numerical results against the underlying experiments, and take responsibility for the final content of this submission.

## References

Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. SliceGPT: Compress large language models by deleting rows and columns. In *The Twelfth International Conference on Learning Representations*, 2024.

Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. A survey on mixture of experts in large language models. *IEEE Trans. Knowl. Data Eng.*, 37(7):3896–3915, 2025. URL https://doi.org/10.1109/TKDE.2025.3554028.

Fu Chaoyou, Chen Peixian, Shen Yunhang, Qin Yulei, Zhang Mengdan, Lin Xu, Yang Jinrui, Zheng Xiawu, Li Ke, Sun Xing, et al. Mme: A comprehensive evaluation benchmark for multimodal large language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

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 *Computer Vision - ECCV 2024 - 18th European Conference*, volume 15139 of *Lecture Notes in Computer Science*, pp. 19–35. Springer, 2024a. URL https://doi.org/10.1007/978-3-031-73004-7_2.

Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Conghui He, Jiaqi Wang, Feng Zhao, and Dahua Lin. Sharegpt4v: Improving large multi-modal models with better captions. In *Computer Vision - ECCV 2024 - 18th European Conference*, volume 15075, pp. 370–387. Springer, 2024b. URL https://doi.org/10.1007/978-3-031-72643-9_22.

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.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642, 2024.

Elias Frantar and Dan Alistarh. SparseGPT: Massive language models can be accurately pruned in one-shot. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10323–10337. PMLR, 2023.

Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. Sparse GPU kernels for deep learning. In Christine Cuicchi, Irene Qualters, and William T. Kramer (eds.), *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 17. IEEE/ACM, 2020. URL https://doi.org/10.1109/SC41405.2020.00021.

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

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Dan Roberts. The unreasonable ineffectiveness of the deeper layers. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=ngmEcEer8a.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

Xiwei Hu, Rui Wang, Yixiao Fang, Bin Fu, Pei Cheng, and Gang Yu. Ella: Equip diffusion models with llm for enhanced semantic alignment, 2024. URL https://arxiv.org/abs/2403.05135.

Drew A Hudson and Christopher D Manning. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 6700–6709, 2019.

Bo-Kyeong Kim, Hyoung-Kyu Song, Thibault Castells, and Shinkook Choi. BK-SDM: A lightweight, fast, and cheap version of stable diffusion. In *Computer Vision - ECCV 2024 - 18th European Conference*, volume 15112 of *Lecture Notes in Computer Science*, pp. 381–399. Springer, 2024. URL https://doi.org/10.1007/978-3-031-72949-2_22.

Hugo Laurençon, Léo Tronchon, Matthieu Cord, and Victor Sanh. What matters when building vision-language models? In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems*, 2024.

Min Li, Zihao Huang, Lin Chen, Junxing Ren, Miao Jiang, Fengfa Li, Jitao Fu, and Chenghua Gao. Contemporary advances in neural network quantization: A survey. In *International Joint Conference on Neural Networks*, pp. 1–10. IEEE, 2024.

Yifan Li, Yifan Du, Kun Zhou, Jinpeng Wang, Wayne Xin Zhao, and Ji-Rong Wen. Evaluating object hallucination in large vision-language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=xozJw0kZXF.

Yunxin Li, Xinyu Chen, Shenyuan Jiang, Haoyuan Shi, Zhenyu Liu, Xuanyu Zhang, Nanhao Deng, Zhenran Xu, Yicheng Ma, Meishan Zhang, Baotian Hu, and Min Zhang. Uni-moe-2.0-omni: Scaling language-centric omnimodal large model with advanced moe, training and data, 2025. URL https://arxiv.org/abs/2511.12609.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Yanzuo Lu, Xin Xia, Manlin Zhang, Huafeng Kuang, Jianbin Zheng, Yuxi Ren, and Xuefeng Xiao. Hyper-bagel: A unified acceleration framework for multimodal understanding and generation, 2025. URL https://arxiv.org/abs/2509.18824.

Xuan Luo, Weizhi Wang, and Xifeng Yan. Adaptive layer-skipping in pre-trained llms, 2025. URL https://arxiv.org/abs/2503.23798.

Ji Ma, Wei Suo, Peng Wang, and Yanning Zhang. Short-lvlm: Compressing and accelerating large vision-language models by pruning redundant layers. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 3575–3584, 2025a.

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 *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 7739–7751, 2025b.

Amir M Mansourian, Rozhan Ahmadi, Masoud Ghafouri, Amir Mohammad Babaei, Elaheh Badali Golezani, Zeynab Yasamani Ghamchi, Vida Ramezanian, Alireza Taherian, Kimia Dinashi, Amirali Miri, et al. A comprehensive survey on knowledge distillation. *arXiv preprint arXiv:2503.12067*, 2025.

Weijia Mao, Zhenheng Yang, and Mike Zheng Shou. Unimod: Efficient unified multimodal transformers with mixture-of-depths, 2025. URL https://arxiv.org/abs/2502.06474.

Xin Men, Mingyu Xu, Qingyu Zhang, Qianhao Yuan, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect. In *Findings of the Association for Computational Linguistics*, pp. 20192–20204. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.findings-acl.1035/.

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

Cheng Shi, Yizhou Yu, and Sibei Yang. Vision function layer in multimodal llms. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Jiwon Song, Kyungseok Oh, Taesu Kim, Hyungjun Kim, Yulhwa Kim, and Jae-Joon Kim. SLEB: streamlining llms through redundancy verification and elimination of transformer blocks. In *Forty-first International Conference on Machine Learning*. OpenReview.net, 2024. URL https://openreview.net/forum?id=fuX4hyLPmO.

Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations*. OpenReview.net, 2024.

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Guo-Hua Wang, Shanshan Zhao, Xinjie Zhang, Liangfu Cao, Pengxin Zhan, Lunhao Duan, Shiyin Lu, Minghao Fu, Xiaohao Chen, Jianshan Zhao, et al. Ovis-u1 technical report. *arXiv preprint arXiv:2506.23044*, 2025.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems*, pp. 2074–2082, 2016.

Chenyuan Wu, Pengfei Zheng, Ruiran Yan, Shitao Xiao, Xin Luo, Yueze Wang, Wanli Li, Xiyan Jiang, Yexin Liu, Junjie Zhou, et al. Omnigen2: Exploration to advanced multimodal generation. *arXiv preprint arXiv:2506.18871*, 2025.

Zhiyu Wu, Xiaokang Chen, Zizheng Pan, Xingchao Liu, Wen Liu, Damai Dai, Huazuo Gao, Yiyang Ma, Chengyue Wu, Bingxuan Wang, Zhenda Xie, Yu Wu, Kai Hu, Jiawei Wang, Yaofeng Sun, Yukun Li, Yishi Piao, Kang Guan, Aixin Liu, Xin Xie, Yuxiang You, Kai Dong, Xingkai Yu, Haowei Zhang, Liang Zhao, Yisong Wang, and Chong Ruan. Deepseek-vl2: Mixture-of-experts vision-language models for advanced multimodal understanding, 2024. URL https://arxiv.org/abs/2412.10302.

Guowei Xu, Peng Jin, Hao Li, Yibing Song, Lichao Sun, and Li Yuan. Llava-cot: Let vision language models reason step-by-step, 2024. URL https://arxiv.org/abs/2411.10440.

Weihao Yu, Zhengyuan Yang, Lingfeng Ren, Linjie Li, Jianfeng Wang, Kevin Lin, Chung-Ching Lin, Zicheng Liu, Lijuan Wang, and Xinchao Wang. Mm-vet v2: A challenging benchmark to evaluate large multimodal models for integrated capabilities, 2024. URL https://doi.org/10.21227/pvmd-s489.

Qianhao Yuan, Qingyu Zhang, Yanjiang Liu, Jiawei Chen, Yaojie Lu, Hongyu Lin, Jia Zheng, Xianpei Han, and Le Sun. Shortv: Efficient multimodal large language models by freezing visual tokens in ineffective layers, 2025. URL https://arxiv.org/abs/2504.00502.

Xiang Yue, Tianyu Zheng, Yuansheng Ni, Yubo Wang, Kai Zhang, Shengbang Tong, Yuxuan Sun, Botao Yu, Ge Zhang, Huan Sun, Yu Su, Wenhu Chen, and Graham Neubig. Mmmu-pro: A more robust multi-discipline multimodal understanding benchmark. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15134–15186. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.736/.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 11975–11986. IEEE, 2023.

Chen Zhang, Meizhi Zhong, Qimeng Wang, Xuantao Lu, Zheyu Ye, Chengqiang Lu, Yan Gao, Yao Hu, Kehai Chen, Min Zhang, and Dawei Song. MoDification: Mixture of depths made easy. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5137–5149. Association for Computational Linguistics, 2025a. URL https://aclanthology.org/2025.naacl-long.265/.

Xinjie Zhang, Jintao Guo, Shanshan Zhao, Minghao Fu, Lunhao Duan, Jiakui Hu, Yong Xien Chng, Guo-Hua Wang, Qing-Guo Chen, Zhao Xu, et al. Unified multimodal understanding and generation models: Advances, challenges, and opportunities. *arXiv preprint arXiv:2505.02567*, 2025b.

Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Trans. Knowl. Data Eng.*, 34(12): 5586–5609, 2022. URL https://doi.org/10.1109/TKDE.2021.3070203.

Yuan Zhang, Chun-Kai Fan, Junpeng Ma, Wenzhao Zheng, Tao Huang, Kuan Cheng, Denis A. Gudovskiy, Tomoyuki Okuno, Yohei Nakata, Kurt Keutzer, and Shanghang Zhang. Sparsevlm: Visual token sparsification for efficient vision-language model inference. In *Forty-second International Conference on Machine Learning*. OpenReview.net, 2025c. URL https://openreview.net/forum?id=80faIPZ67S.

Junhan Zhu, Hesong Wang, Mingluo Su, Zefang Wang, and Huan Wang. Obs-diff: Accurate pruning for diffusion models in one-shot, 2026. URL https://arxiv.org/abs/2510.06751.

# A APPENDIX ROADMAP

Appendices B–I organize the supporting material from scope and additional evidence to implementation and related work. Table 5 provides a section-level guide using the exact appendix titles.

Table 5: Organization and contents of the appendices.

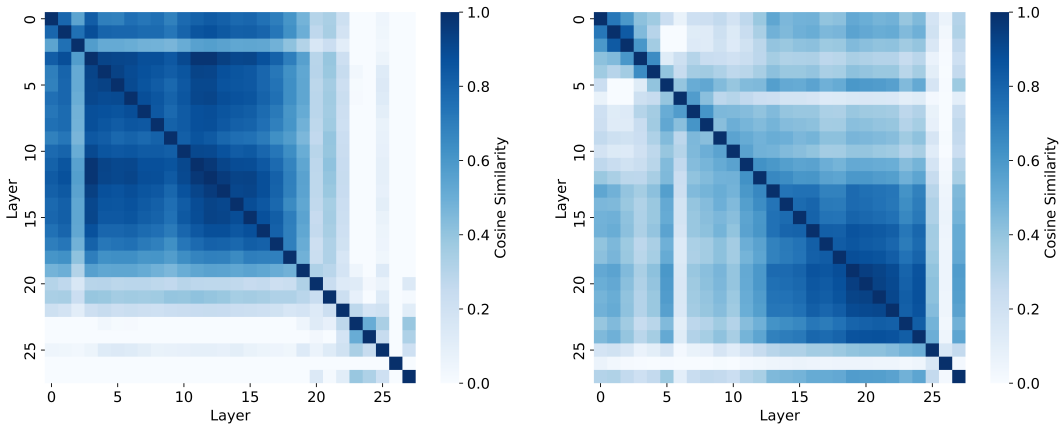
| Appendix section | Contents |
|---|---|
| B. **Limitations** | Generalization scope, pruning sensitivity, memory trade-offs, healing-protocol comparisons, and search-space extensions. |
| C. **Additional Evidence for Task-Dependent Redundancy** | Task-dependent similarity structures in BAGEL and folding paths in EMU3.5. |
| D. **Cost-Metric and Healing-Data Analyses** | Alternative representation-discrepancy costs, healing-data composition, the matched local baseline, and qualitative generation examples. |
| E. **Additional Experimental Analysis** | Efficiency bottlenecks, detailed results, pruning-rate trade-offs, healing-data scaling, and additional ablations. |
| F. **Deployment Cost and Discussion** | Timing protocol, path switching, weight memory, serving compatibility, and cost-model extensions. |
| G. **Implementation Details** | Dynamic programming, calibration stability, folding-path statistics, training objectives, and compute resources. |
| H. **Baseline Implementation Details** | Adaptations of the structured layer-pruning and unstructured weight-pruning baselines. |
| I. **Extended Related Work** | Dynamic inference, width and depth acceleration, layer redundancy, structural healing, and execution acceleration. |

# B LIMITATIONS

**Generalization Beyond UMMs.** FOLDUMM is designed to address the task-conflict problem in Unified Multimodal Models, where IU and T2I share a large backbone but exhibit different redundancy patterns. In this work, our DAG-based path search is empirically validated in the UMM setting. From a methodological perspective, the same formulation can naturally extend to other Transformer-based architectures, such as LLMs and VLMs, by treating layer selection as a constrained path-search problem over the backbone. Therefore, DAG-based global path optimization may provide a promising direction for broader structured pruning beyond UMMs. A systematic evaluation of its generality on LLMs and VLMs is left as future work. Moreover, the healing training recipe must be adapted to model-specific generation mechanisms; consequently, we have not yet validated the full training pipeline across a wider range of UMM backbones.

**Performance Degradation Under Aggressive Pruning.** Although FOLDUMM improves the performance–pruning trade-off over structured pruning baselines, pruning remains a lossy compression process. At higher pruning rates, especially for T2I, performance drops noticeably, suggesting that generation-critical layers are less redundant and should be pruned more conservatively. This motivates the mixed pruning-rate setting discussed in Appendix E.3, where IU and T2I can adopt different task-specific pruning schedules.

**Memory Trade-Off in Multi-Task Deployment.** The "one backbone, multiple paths" design improves task-specific performance retention and enables fast task switching, but it does not always maximize weight-memory reduction. When multiple task paths are kept resident on GPU, FOLDUMM may use more weight memory than single-path structured pruning methods. Thus,




(a) Image understanding. (b) Image generation.

Figure 6: Task-specific layer-similarity structures on BAGEL. The matrices provide profiling-level evidence; the complete pruning and healing pipeline remains evaluated on Janus-Pro and OmniGen2.

FOLDUMM should be viewed as a practical performance–efficiency trade-off method rather than a method that solely optimizes memory compression. Detailed switching-time and memory measurements are provided in Appendix F.

**Healing Protocol Comparisons.** FOLDUMM uses localized STITCH LORA healing rather than modifying the full backbone. There is currently no standard, broadly adopted protocol for comparing this lightweight recipe against a complete multimodal healing pipeline with matched objectives, data construction, and implementation details. We therefore focus on controlled comparisons among localized healing, full LoRA, and no-healing variants, while reporting the training resources and data scales used by our recipe in Appendix G.3 and Appendix E.4.

**Search-Space Extensions.** The current DAG formulation uses a simple feature-mismatch edge cost and fixes the retained-layer budget. The same framework can naturally support additional constraints, such as bounding the number of fold points, limiting the maximum folding span, or enriching the cost model with task-conditioned sensitivity and healing difficulty. We leave these constrained search variants to future work and discuss possible extensions in Appendix F.5.

# C ADDITIONAL EVIDENCE FOR TASK-DEPENDENT REDUNDANCY

The complete pruning and healing evaluation in the main paper covers the natively unified JANUS-PRO and the architecturally different decoupled OMNIGEN2. We further test the prerequisite underlying task-specific folding on two newer natively unified backbones.

## C.1 BAGEL SIMILARITY STRUCTURE

Figure 6 shows separate layer-similarity matrices for understanding and generation on the 28-layer BAGEL backbone. Understanding exhibits a broad high-similarity region in the early and middle layers, whereas generation exhibits a stronger late-layer region and different structural boundaries. The qualitative difference reproduces task-dependent redundancy on another natively unified model.

## C.2 EMU3.5 FOLDING PATHS

Using the same cosine-mismatch graph construction and a 20% pruning budget, the Emu3.5 searches select the removed-layer sets in Table 6. Their intersection is empty, yielding an IU/T2I Jaccard similarity of 0.000. This provides mechanism-level evidence for FOLDUMM's central prerequisite: different tasks expose distinct redundancy structures, which directly motivates task-specific folding.

Table 6: Emu3.5 removed layers at 20% pruning.

| Task | Removed layers |
|---|---|
| IU | 4, 5, 6, 8, 10, 11, 13, 14, 16, 18, 20, 22 |
| T2I | 24, 25, 27, 40, 42, 43, 44, 46, 48, 51, 55, 56 |

A complete end-to-end transfer to these architectures is listed as future evaluation in the revision plan. The evidence here targets the cross-model generality of task-dependent redundancy, while the main experiments establish the complete method on two representative UMM designs.

# D Cost-Metric and Healing-Data Analyses

## D.1 Alternative Representation-Discrepancy Costs

Cosine mismatch is a simple, scale-insensitive edge cost that follows common practice in depth-reduction work. We additionally evaluate angular distance and relative-$\ell_2$ activation discrepancy under the same calibration data, graph constraints, and 20% pruning budget. Table 7 shows that the metric affects the precise topology while every metric retains task-dependent IU and T2I paths.

Table 7: Removed layers selected by three edge-cost definitions on Janus-Pro.

| Task | Cosine | Angular | Activation relative-$\ell_2$ |
|---|---|---|---|
| IU | 3, 22, 24, 25, 26, 28 | 21, 22, 23, 24, 25, 26 | 24, 25, 26, 27, 28, 29 |
| T2I | 20, 22, 23, 24, 25, 26 | 22, 23, 24, 25, 26, 27 | 23, 24, 25, 26, 27, 28 |

Within each task, pairwise Jaccard similarities range from 0.333 to 0.714. Across tasks, the cosine paths have a Jaccard similarity of 0.500; angular and activation paths each have 0.714. Applying the same healing protocol to the angular paths yields 33.1 on MM-Vet-V2 and 0.62 on GenEval, compared with 36.4 and 0.66 for cosine mismatch. Thus, cosine performs best among these tested costs, while the framework remains effective with another established representation discrepancy. Activation relative-$\ell_2$ further shows that the DAG search accepts magnitude-sensitive costs directly once a cost matrix is supplied.

## D.2 Healing-Data Composition

The submitted IU recipe uniformly samples 30K examples from The Cauldron to keep data construction simple and reproducible. In a matched variant, we keep the total 30K budget fixed and replace 5K examples with 5K multimodal reasoning examples sampled across the constituent sources of LLaVA-CoT-100K (Xu et al., 2024). The folding path, initialization, and optimization settings remain fixed. MM-Vet-V2 improves by 2.2, 2.7, and 4.1 percentage points across three random seeds, for a mean improvement of 3.0 points.

On the same 50-example diagnostic subset, the mean response length increases from 127.5 to 358.8 characters, and the median increases from 46 to 136. The result supports deliberate healing-data composition as a useful extension. It also preserves the main conclusion obtained with the original simple sampling strategy, since the comparison changes only the composition of a fixed-size healing budget.

## D.3 Matched Task-Specific Local Baseline

The Greedy row in Table 3 is the task-specific local ShortGPT/Block-Influence control: it separately chooses IU and T2I removals under the same 20% budget and applies the same Stitch LoRA healing. The one-block edge cost of FoldUMM reduces to this local cosine Block Influence criterion. The complete controlled progression on MM-Vet-V2/GenEval is 19.1/0.41 for ShortGPT, 27.1/0.61 after adding Stitch LoRA, 29.7/0.62 after selecting separate IU and T2I paths, and 36.4/0.66 with the global DAG-DP search. Thus, the first transition measures localized healing, the second measures

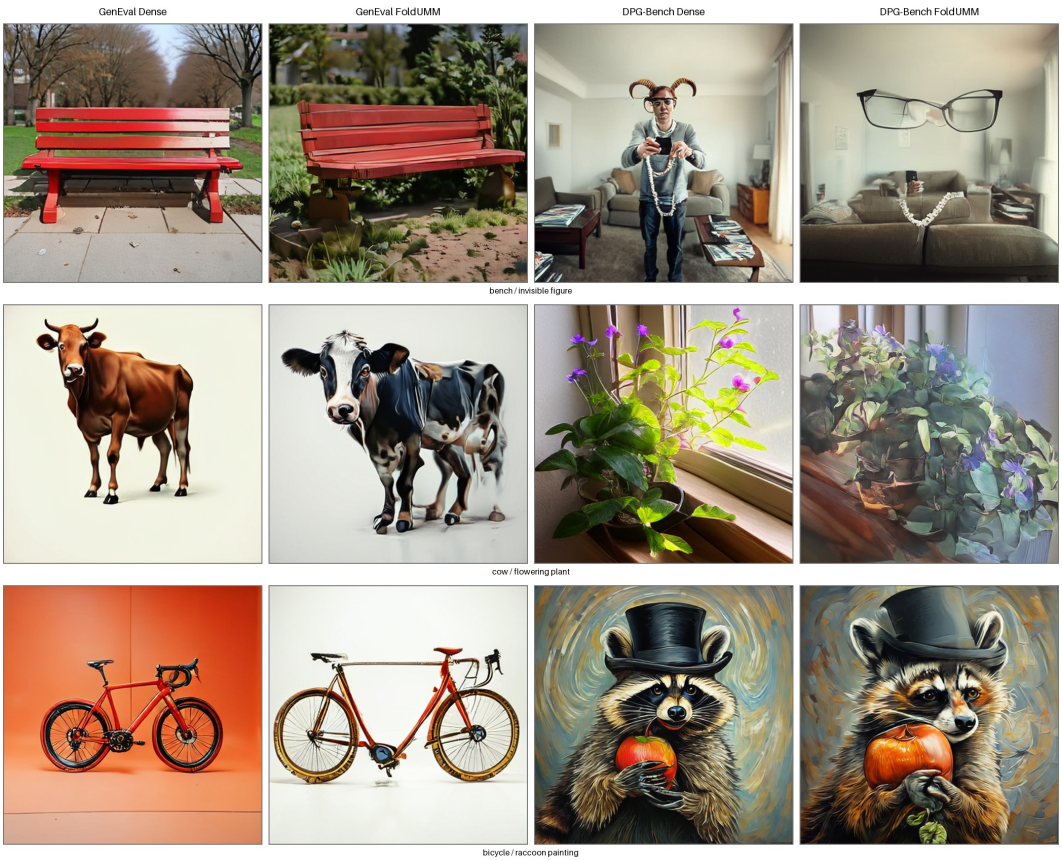



Figure 7: Dense and 20% FOLDUMM generations. The three GenEval prompts are "a photo of a bench," "a photo of a cow," and "a photo of a bicycle." The DPG-Bench examples cover an invisible figure with floating accessories, a purple-flower potted plant, and a stylized elderly raccoon portrait.

task-specific paths, and the final transition isolates the gain from globally optimizing the full folding path. SHORTGPT is the closest control in both selection principle and search cost; the other baselines introduce additional method-specific selection or compensation components.

### D.4 QUALITATIVE GENERATION EXAMPLES

Figure 7 shows selected examples from GenEval and DPG-Bench under the same official Janus-Pro generation settings. These images are illustrative; the quantitative GenEval and DPG-Bench metrics in the main paper provide the primary evaluation.

## E ADDITIONAL EXPERIMENTAL ANALYSIS

### E.1 EFFICIENCY BOTTLENECKS ANALYSIS

We prioritize the **Unified** paradigm for optimization. While the Decoupled paradigm offers modularity, "natively unified" represents the frontier of UMM research: their shared parameter space unlocks complex cross-modal reasoning and instruction-following capabilities unattainable in modular systems (Deng et al., 2025; Chen et al., 2025). However, this design centralizes computation in a massive Transformer backbone that becomes the primary efficiency bottleneck.

**Empirical Bottleneck Profiling.** As profiled in Table 8, the backbone accounts for $> 85\%$ of total parameters in Unified architectures, reflecting its role as a shared multimodal knowledge repository. Computationally, this dominance is exacerbated during autoregressive generation: the backbone

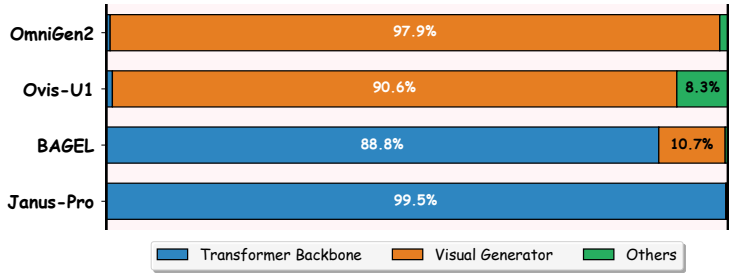



Figure 8: T2I latency decomposition reveals a critical bottleneck divergence: Decoupled models (OMNIGEN2, OVIS-U1) are dominated by the Visual Generator ($> 90\%$), while Unified models (BAGEL, JANUS-PRO) are dominated by the Backbone ($> 88\%$).

consumes 88.8% of T2I latency in BAGEL and 99.5% in JANUS-PRO (Figure 8). FLOPs analysis of JANUSFLOW confirms this pattern, with the core LLM consuming 246.3 GFLOPs per step (90.8% of total). This computational concentration implies that compressing the single backbone component simultaneously accelerates all tasks.

### E.2 DETAILED MAIN RESULTS

**Structured Pruning Baselines.** The main results in Table 1 show that structured pruning baselines can obtain non-trivial speedups, but often suffer substantial degradation on either IU or T2I. On JANUS-PRO-7B, SHORTGPT achieves a similar token-throughput speedup to FOLDUMM, but its IU and T2I relative performance are much lower. SHORTGPT and SLEB benefit from matched STITCH LORA recovery, but their recovered variants still retain only 86.2%/85.5% and 74.0%/59.3% IU/T2I RP at 20% pruning. This suggests that redundancy-based block elimination alone is insufficient for unified multimodal compression. SHORT-LVLM retains relatively stronger IU performance than unrecovered baselines, but its T2I performance drops severely, indicating that preserving visual understanding does not necessarily preserve image generation.

**Search Cost.** Matched recovery does not remove the offline search-cost gap among structured pruning methods. SLEB performs iterative greedy layer elimination and requires repeated model evaluation during redundancy analysis. In our JANUS-PRO-7B 20% pruning setting, the full SLEB redundancy analysis takes approximately 8 hours on a single GPU, which is comparable to one STITCH LORA healing run in our setup. By contrast, after FOLDUMM collects layer-wise redundancy statistics, the DP path search completes in under one second on CPU. Thus, the matched-recovery SLEB baseline is substantially more expensive offline than the main table alone suggests.

**Unstructured Pruning Baselines.** WANDA and SPARSEGPT preserve strong quality at 20% sparsity, but this quality retention does not translate into practical acceleration on standard GPUs. In our setting, their measured token throughput remains close to the dense model. Therefore, we include them as quality-oriented references rather than practical acceleration baselines. This distinction is important because FOLDUMM targets structured path-level acceleration, whereas unstructured sparsity typically needs higher sparsity and specialized hardware for wall-clock gains.

Table 8: **Empirical bottleneck profiling across representative UMMs.** Backbone Ratio denotes the fraction of backbone parameters in the full model. Bottleneck identifies the primary latency source: Task-dependent (shifts between components) vs. Integrated (dominated by the shared backbone).

| Model | Paradigm | Params | Backbone Ratio | Bottleneck |
|---|---|---|---|---|
| OMNIGEN2 | Decoupled | 7B | 48.1% | Task-dependent |
| OVIS-U1 | Decoupled | 3.6B | 49.0% | Task-dependent |
| BAGEL | Unified | 14B | 96.1% | Integrated |
| JANUS-PRO | Unified | 7B | 86.7% | Integrated |

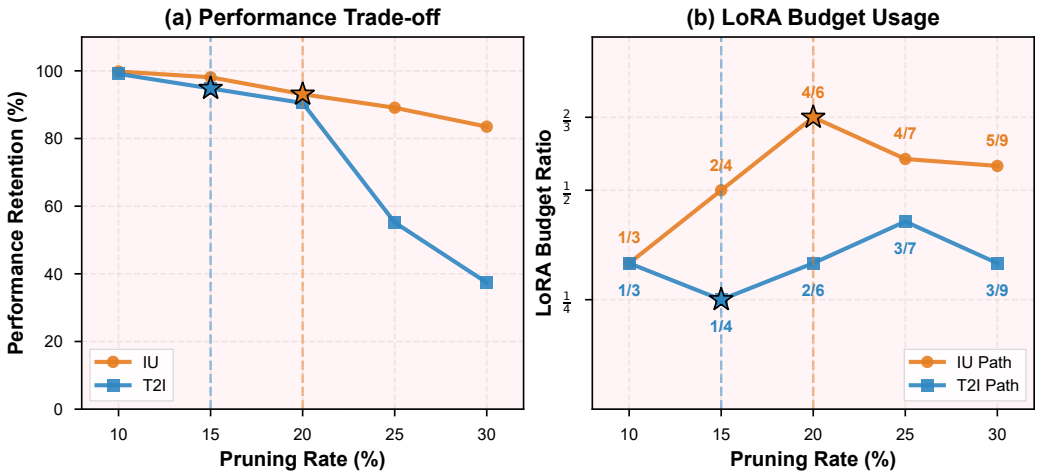



Figure 9: **Detailed pruning-rate and LoRA budget analysis on JANUS-PRO-7B.** We show task-specific performance retention and STITCH LORA budget usage across pruning rates. IU and T2I prefer different pruning schedules, supporting task-aware path selection.

**Interpretation.** These results support the main motivation of FOLDUMM: UMM compression should preserve both understanding and generation, rather than optimizing only one task family. The comparison also shows that task-aware structured pruning improves the trade-off over static layer pruning without requiring specialized sparse hardware.

**MM-Vet-V2 Analysis.** Among the IU benchmarks, MM-Vet-V2 exhibits the most pronounced sensitivity to pruning and structured compression. This benchmark requires multiple skills, including OCR, long-horizon reasoning, and knowledge integration, making it inherently challenging. Structured pruning introduces biases that affect both FOLDUMM and baseline models; for instance, ShortGPT and Short-LVLM tend to produce shorter or affirmative ("yes") answers, leading to larger declines. FOLDUMM experiences moderate performance drops on MM-Vet-V2, reflecting dataset complexity rather than methodological weakness. Importantly, the FOLDUMM fine-tuning data was uniformly sampled across tasks, ensuring that no benchmark, including MM-Vet-V2, was overrepresented. Table 9 summarizes IU performance across different pruning rates for FOLDUMM and structured baselines, supporting this analysis.

Table 9: MM-Vet-V2 accuracy under 10%, 20%, and 30% pruning rates. FOLDUMM shows moderate drops, while baselines degrade more severely.

| **Method** | 10% Prune | 20% Prune | 30% Prune |
|---|---|---|---|
| ShortGPT | 35.0% | 19.1% | 12.0% |
| Short-LVLM | 38.2% | 16.8% | 10.5% |
| SLEB | 17.9% | 12.4% | 9.0% |
| FOLDUMM + STITCH LORA | 39.3% | 36.4% | 25.7% |

### E.3 PRUNING-RATE TRADE-OFF ANALYSIS

**Task-Specific Optimal Pruning Rates.** The main paper shows that FOLDUMM remains more stable than structured pruning baselines across pruning rates. Here, we provide additional analysis of how different tasks prefer different pruning schedules. For IU tasks, a 20% pruning rate provides the best balance between efficiency and performance retention. For T2I tasks, however, the best trade-off appears at a more conservative 15% pruning rate. This difference suggests that IU and T2I do not share the same redundancy structure even though they are served by the same unified backbone.

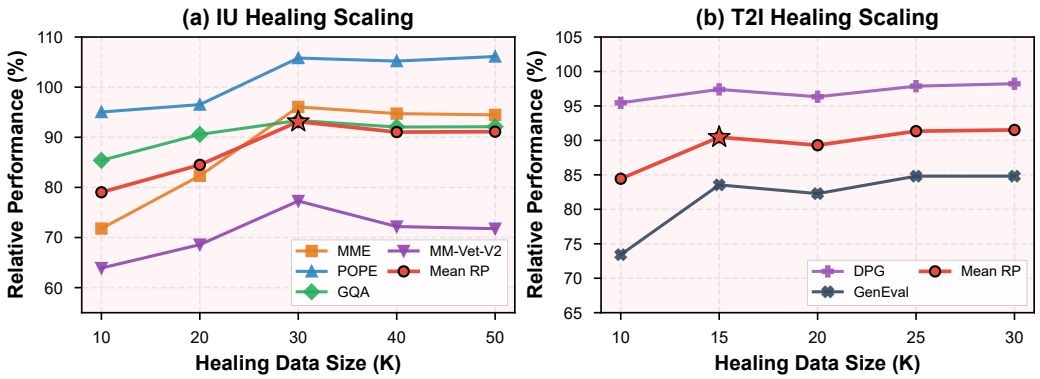



Figure 10: **Healing data scaling on JANUS-PRO-7B at 20% pruning.** Stars mark the healing-data budgets used in the main experiments. IU RP is computed over MME, POPE, GQA, and MM-Vet-V2.

**Mixed Pruning Schedules.** A central benefit of FOLDUMM is that task-specific paths can coexist and be activated dynamically. Therefore, deployment does not need to choose a single pruning rate for all tasks. Instead, the system can use the 20% IU path for understanding requests and the 15% T2I path for generation requests. This mixed configuration better matches the redundancy pattern of each task: IU can tolerate more aggressive layer folding, while T2I benefits from a more conservative path that avoids removing generation-critical layers.

**Fold-Point Healing Analysis.** We further analyze the STITCH LORA budget used by each task-specific path. At 15% pruning, IU uses $2/4$ LoRA modules, while T2I uses only $1/4$. This indicates that T2I is not only more sensitive to aggressive pruning, but also benefits from a path that requires fewer fold-point healing modules at moderate pruning rates. In contrast, IU can tolerate more structural folding and achieves its best trade-off at a higher pruning rate. These results support the design choice of allocating healing modules locally at fold points rather than applying LoRA uniformly to all retained layers.

**Implication.** The above observations reinforce the main motivation of FOLDUMM: UMM compression should not be treated as a single static layer-removal problem. Because IU and T2I exhibit different redundancy and healing requirements, task-aware path selection provides a more flexible compression strategy. This enables different tasks to operate at different compression levels while sharing the same overall model framework.

### E.4 HEALING DATA SCALING AND DATA DESIGN

STITCH LORA repairs feature mismatch only at fold points; consequently, its recovery quality depends on whether the healing data covers the capabilities affected by folding. We therefore treat healing-data scale and task coverage as compression-design choices.

**Image Understanding.** Figure 10(a) evaluates IU healing from 10K to 50K examples. Overall recovery improves sharply from 10K to 30K, supporting our choice of 30K IU healing examples as a cost-efficient setting that already obtains strong recovery. MM-Vet-V2 is an exception to the monotonic trend: larger healing sets do not always improve it because MM-Vet-V2 often requires long chain-of-thought style answers, whereas our current IU healing data are dominated by short-answer instruction examples. Increasing such data can therefore bias the folded model toward shorter responses and reduce MM-Vet-V2 performance.

**Text-to-Image Generation.** Figure 10(b) shows the corresponding T2I trend from 10K to 30K examples. Increasing the healing data from 10K to 15K substantially improves generation recovery, and larger budgets can further improve some generation metrics. We use 15K T2I examples in the main experiments to limit healing cost while retaining a strong recovery level; the scaling results indicate that additional data can still be useful when maximizing generation quality is prioritized.

**Data Design.** These scaling trends suggest that data quantity is only one part of the healing recipe; the composition of healing data also matters. For IU, data should cover OCR, recognition, spatial reasoning, and knowledge-heavy instructions. Reasoning-heavy benchmarks further require long-form answer patterns rather than only short factual responses. For T2I, prompts should cover object attributes, counting, spatial relations, and compositional generation, because these categories are among the first to degrade under aggressive folding.

### E.5 Additional Ablation Analysis

**Task-Aware Path Selection.** To verify whether IU and T2I require different execution paths, we conduct cross-task path transfer by applying the IU-optimized path to T2I and the T2I-optimized path to IU. As shown in Table 2, the two tasks exhibit clear path mismatch on JANUS-PRO. Using the T2I-optimized path for IU causes a substantial drop on the IU benchmark, indicating that the T2I path removes layers that remain important for language-grounded visual understanding. Conversely, applying the IU path to T2I also affects generation quality, although the degradation is less severe in our current setting. This asymmetric transfer pattern suggests that IU and T2I have partially overlapping but non-identical layer requirements.

The behavior is different on OMNIGEN2. Since OMNIGEN2 adopts a decoupled design, its MLLM backbone mainly acts as a condition encoder for the external generation module. Therefore, the role of the backbone in T2I is closer to its role in IU, leading to higher path overlap and smaller cross-path degradation. This contrast explains why task-aware path selection is especially important for natively unified architectures such as JANUS-PRO, where the same backbone directly supports both understanding and autoregressive image generation.

**DAG-Based Global Search.** Table 3 compares our DAG-based dynamic programming search with uniform, random, and greedy layer selection. Uniform pruning removes layers according to a fixed interval and ignores the non-uniform redundancy distribution across the backbone. Random selection is unstable because the retained path can vary substantially across trials. Greedy selection improves over these simple baselines, but it makes local removal decisions and can miss globally better layer paths. In contrast, our DAG formulation evaluates the cumulative cost of a complete folding path, allowing the dynamic programming search to identify more reliable task-specific configurations.

**Localized Fold-Point Healing.** Layer folding directly connects non-adjacent layers, which can introduce feature misalignment at fold points. As shown in Table 4, direct folding without LoRA causes severe degradation, confirming that some form of structural healing is necessary. Full LoRA provides strong recovery by adapting all retained layers, but it introduces substantially more trainable parameters. STITCH LORA instead applies lightweight adaptation only at selected fold points, where the mismatch caused by layer folding is most concentrated. This localized design recovers most of the performance of Full LoRA while using far fewer trainable parameters, supporting our treatment of STITCH LORA as a localized fold-point healing mechanism rather than a full fine-tuning strategy.

**Summary.** Overall, the ablation results validate the three main components of FOLDUMM. Task-aware path selection is necessary because IU and T2I do not share identical redundancy patterns. DAG-based global search is more reliable than local or heuristic layer-removal strategies. Finally, localized fold-point healing is sufficient to repair most of the feature mismatch introduced by layer folding, avoiding the cost of applying LoRA uniformly to all retained layers.

## F Deployment Cost and Discussion

**Overview.** FOLDUMM maintains task-specific execution paths for IU and T2I. This design improves task-specific performance retention, but it also introduces two deployment questions: whether switching between task paths incurs non-negligible latency, and whether maintaining multiple paths increases resident memory. We analyze these costs below and then discuss practical deployment scenarios and future improvements.

Table 10: **Path-switching overhead on JANUS-PRO-7B.** We report the minimum switching time between IU and T2I paths under different deployment strategies. Dual-path resident VRAM only switches lightweight path-specific references and therefore incurs negligible overhead.

| **Strategy** | **10%** | **20%** | **30%** |
|---|---|---|---|
| Dual-path resident VRAM | 0.02 ms | 0.01 ms | 0.01 ms |
| Smart offloading | 1070 ms | 664 ms | 965 ms |
| Cold reloading | 11128 ms | 9117 ms | 7488 ms |

Table 11: **Weight-memory footprint on JANUS-PRO-7B.** FOLDUMM reduces memory compared with the dense model, but uses more resident weight memory than single-path structured pruning methods because it maintains task-specific paths.

| **Method** | Dense | Wanda | SLEB | ShortGPT | FOLDUMM |
|---|---|---|---|---|---|
| Weight memory (GB) | 14.9 | 14.9 | 11.6 | 11.6 | 13.2 |

### F.1 INFERENCE AND TIMING PROTOCOL

The main results use the standard Hugging Face/PyTorch inference path on an NVIDIA A6000 with batch size 1 and each model's official default generation configuration. For Janus-Pro, image generation uses 576 autoregressive visual tokens at $384 \times 384$ resolution, classifier-free guidance 5.0, and temperature 1.0. Backbone speedup measures token-generation throughput; Gen.(s) includes prompt processing, autoregressive visual-token generation, and VQ decoding. The submitted paired measurement reduces complete Janus-Pro image generation from 3.94 s to 3.16 s.

### F.2 PATH SWITCHING OVERHEAD

**Switching Strategies.** We consider three representative deployment strategies. **Dual-path resident VRAM** keeps both task-specific paths resident on GPU and only switches lightweight path-specific references at inference time. **Smart offloading** keeps shared layers resident and migrates only path-specific layers when switching tasks. **Cold reloading** reloads the full task path for each switch. Table 10 reports the minimum IU–T2I switching time on JANUS-PRO-7B.

**Analysis.** The results show that keeping both paths resident on GPU makes task switching almost free. At 20% pruning, for example, the dual-path resident strategy requires only 0.01 ms for switching, while smart offloading requires 664 ms and cold reloading requires 9117 ms. This indicates that the practical deployment bottleneck is not the logical path switch itself, but whether the system needs to migrate or reload path-specific weights. When GPU memory permits, dual-path residency is therefore the preferred deployment mode for mixed IU–T2I workloads.

### F.3 WEIGHT-MEMORY FOOTPRINT

The main trade-off of dual-path residency is memory. Table 11 compares the weight-memory footprint of FOLDUMM with dense and pruning baselines. Since FOLDUMM maintains task-specific paths, its memory footprint is lower than the dense model but higher than single-path structured pruning baselines.

These results clarify the deployment trade-off of FOLDUMM. Compared with single-path structured pruning, FOLDUMM sacrifices part of the maximum possible memory saving in exchange for task-specific execution paths and negligible task-switching overhead. Compared with cold reloading or CPU–GPU migration, dual-path residency avoids second-level switching latency, making it more suitable for UMM applications where IU and T2I requests appear in the same serving workload. Thus, FOLDUMM should be viewed as a performance–efficiency trade-off method rather than a method that solely maximizes weight-memory compression.

### F.4 DEPLOYMENT FLEXIBILITY

The "one backbone, multiple paths" design supports flexible single-task and mixed-task deployment.

**Single-Task Deployment.** For applications that predominantly serve one task type, such as a dedicated image captioning service or a text-to-image generation API, only the corresponding task path needs to remain resident. Folded layers that are irrelevant to the active task can be offloaded from GPU memory, enabling memory savings similar to structural pruning while retaining the option to support other tasks when needed.

**Mixed-Task Online Serving.** For online serving with mixed task requests, such as a unified multimodal assistant, dual-path residency enables fast switching between IU and T2I paths. vLLM is extending UMM serving through vLLM-Omni on a model-specific basis. BAGEL is supported, while Janus-Pro support remains an unmerged pull request. FOLDUMM reduces the executed Transformer depth, so its structural speedup is orthogonal to serving-side scheduling, batching, KV-cache, and kernel optimizations. Path-specific STITCH LORA adapters can reuse vLLM's Multi-LoRA mechanism when kept separate. Compatible vLLM-Omni support therefore allows both gains to be combined. The current unmerged Janus-Pro support precludes matched serving-engine benchmarking.

**Stage-Composed Multimodal Workflows.** Additional task types can be supported by calibrating and adding a corresponding task-specific path. For example, image editing can use an image-and-instruction-to-image path with its own calibration data and fold-point healing modules. Reflection-style generation composes image generation, textual reflection, and revision as successive stages. FOLDUMM supports this stage-composed workflow by keeping a fixed path within each autoregressive stage and selecting the appropriate path at an output-modality boundary. KV caching proceeds normally within a fixed path, while a stage-boundary path change starts with fresh encoding or prefill. Fine-grained token-level path switching and cross-path KV reuse introduce additional cache dependencies and remain useful directions for dedicated study.

**Permanent Layer Pruning.** Layers folded across all task-specific paths can be permanently pruned from the model. These universally redundant layers provide guaranteed memory reduction without sacrificing any task-specific path. On JANUS-PRO, task-specific folding exposes 2–3 layers that are redundant across all task paths at moderate pruning rates.

**Future Directions.** More sophisticated scheduling strategies for dynamic path switching remain open. Important directions include proactive path activation, cross-path batching, and latency–throughput optimization for mixed workloads.

### F.5 COST MODEL AND LORA BUDGET ALLOCATION

The relationship between healing capacity and performance recovery warrants further investigation. More aggressive folding generally requires stronger local healing, and the preferred LoRA allocation can differ across tasks. For example, IU and T2I may select different pruning schedules and different numbers of fold-point healing modules, consistent with their different redundancy patterns.

This observation suggests that STITCH LORA's potential could be further improved by designing more expressive cost models. The current edge cost in the main text focuses on feature distribution mismatch. Future extensions could incorporate:

- **Fold-count constraint:** Bounding the number of fold points can directly control the number of required STITCH LORA modules and simplify deployment.
- **Folding span penalty:** Longer folds may require stronger healing; therefore, a span-dependent cost could better capture the difficulty of skipping multiple layers.
- **Maximum folding span:** Explicitly limiting the largest allowed layer jump could avoid unstable long-range transitions under aggressive pruning.
- **Layer criticality scores:** Gradient-based importance scores or attention-pattern analysis could help identify layers critical for specific capabilities.

- **Task-specific sensitivity:** Different tasks may tolerate folding differently at different depths, which could be modeled by task-conditioned edge costs.
- **Healing-aware cost:** Edge costs could include an estimate of the healing difficulty or expected LoRA capacity required for each fold.
- **Data-aware cost:** The path search could weight transitions by capability coverage or task-data diversity, aligning the selected path with available healing data.
- **Joint optimization:** Joint path and budget optimization may improve the trade-off.

These extensions represent promising directions for improving the efficiency–performance trade-off in task-aware model compression.

# G Implementation Details

## G.1 Dynamic Programming Algorithm

Algorithm 1 presents the complete dynamic programming procedure for finding the optimal folding path. The algorithm explores all valid layer transitions under the retained-layer budget.

**Algorithm 1** Task-Aware Optimal Folding Path Search

**Input:** Backbone with $N$ layers, task-specific cost matrix $C \in \mathbb{R}^{N \times N}$, pruning rate $\alpha$
**Output:** Optimal folding path $\mathcal{P}^*$

*// Initialize parameters*
$M \leftarrow \lceil (1 - \alpha) \cdot N \rceil$ ▷ Target retained layers
$DP[j][k] \leftarrow \infty$ for all $j, k$
$DP[0][1] \leftarrow 0$ ▷ Start from layer 0

*// Forward pass: fill DP table*
**for** $j = 1$ **to** $N - 1$ **do**
  **for** $k = 1$ **to** $\min(j + 1, M)$ **do**
    **for** $i = 0$ **to** $j - 1$ **do**
      $c \leftarrow C[i][j]$
      $DP[j][k] \leftarrow \min(DP[j][k],\ DP[i][k - 1] + c)$
    **end for**
  **end for**
**end for**

*// Backtrack to recover optimal path*
$\mathcal{P}^* \leftarrow \textsc{Backtrack}(DP, N - 1, M)$
**return** $\mathcal{P}^*$

**Complexity Analysis.** The time complexity is $O(N^2 \cdot M)$, where $N$ is the number of layers and $M$ is the target number of retained layers. For a 30-layer backbone with 20% pruning, the search completes in under 1 second on CPU.

**Calibration Stability.** We use 256 calibration samples per task for all reported path-search experiments and for all adapted baselines. In preliminary checks on Janus-Pro-7B, reducing the calibration set to 64 or 128 samples produced the same qualitative redundancy pattern and similar selected folding regions, while 256 samples provided slightly more stable cost estimates across tasks. We therefore use 256 samples as a conservative default across methods.

## G.2 Folding Strategy Statistics

Table 12 summarizes the discovered folding paths for IU and T2I tasks on Janus-Pro-7B (30 layers) at different pruning rates. The removed layer overlap quantifies the Jaccard similarity between the sets of layers removed by each task, revealing the degree of redundancy divergence.

Table 12: **Folding Strategy Statistics** on JANUS-PRO-7B. Removed Overlap denotes the Jaccard similarity between the sets of removed layers for IU and T2I. #LoRA indicates the number of STITCH LORA modules required for each task.

| Pruning Rate | Removed Layers | | #LoRA | | Removed Overlap |
|---|---|---|---|---|---|
| | IU | T2I | IU | T2I | |
| 10% | 25–27 | 23–25 | 1 | 1 | 20.0% |
| 15% | 22, 25–27 | 23–26 | 2 | 1 | 33.3% |
| 20% | 3, 22, 24–26, 28 | 20, 22–26 | 4 | 2 | 50.0% |
| 25% | 3–4, 22, 24–26, 28 | 2, 20, 22–26 | 4 | 3 | 40.0% |
| 30% | 3–4, 19–20, 22, 24–26, 28 | 2–3, 20, 22–27 | 5 | 3 | 50.0% |

**Observations.** The results reveal several patterns: (1) The removed layer overlap is consistently low (20%–50%), confirming that IU and T2I tasks identify different layers as redundant—a static pruning approach would inevitably sacrifice one task's performance. (2) Both tasks predominantly target the latter half of the backbone, but with distinct preferences—IU removes layers around 24–28 while T2I concentrates on 22–26. (3) IU consistently requires more LoRA modules at the same pruning rate, reflecting a more distributed folding pattern than T2I.

### G.3 TRAINING DETAILS

We describe the STITCH LORA training procedure for both Image Understanding (IU) and Text-to-Image (T2I) tasks. Due to the distinct nature of these tasks, we employ different training objectives: standard supervised fine-tuning for IU and offline knowledge distillation for T2I.

Table 13 consolidates the data, objectives, and shared optimization settings used by the reported healing experiments. The following subsections provide the corresponding objective definitions and architecture-specific implementation details.

Table 13: **Consolidated STITCH LORA training protocol.** All backbone parameters are frozen, and only fold-point modules are optimized.

| Setting | IU | Janus-Pro T2I | OmniGen2 T2I |
|---|---|---|---|
| Calibration | 256 Cauldron samples | 256 ShareGPT4V samples | 256 ShareGPT4V samples |
| Healing data | 30K Cauldron samples | 15K ShareGPT4V samples | 15K ShareGPT4V samples |
| Supervision | Causal-LM cross-entropy | Offline logit distillation, KL at $T = 2.0$ | Offline final-hidden-state distillation, MSE |
| Adaptation | One STITCH LORA module per non-adjacent folding edge; $r = 128$, $\alpha = 256$ | | |
| Optimization | AdamW, lr $10^{-4}$, weight decay 0.01, cosine schedule, 10% warm-up | | |
| Batch and seeds | Effective batch size 32; three random seeds | | |
| Precision | bfloat16 | bfloat16 | bfloat16 weights and cached targets; float16 DiT autocast |
| Hardware | NVIDIA A6000; approximately 32 GB peak memory and less than 12 GPU-hours per method | | |

#### G.3.1 DATA SOURCES

For calibration data $\mathcal{D}_t^{\text{cal}}$, we hold out 256 samples each from The Cauldron (Laurençon et al., 2024) for IU and ShareGPT4V (Chen et al., 2024b) for T2I, constituting subsets of the training data. For healing data $\mathcal{D}_t^{\text{heal}}$, we sample 30K instances from The Cauldron for IU healing and 15K instances from ShareGPT4V for T2I healing.

#### G.3.2 IMAGE UNDERSTANDING TRAINING

For IU tasks, we employ standard causal language modeling with cross-entropy loss:

$$\mathcal{L}_{\text{IU}} = -\sum_{t=1}^{T} \log P_\theta(y_t | y_{<t}, \mathbf{x}), \tag{6}$$

where $\mathbf{x}$ is the multimodal input (image + text) and $y_t$ is the target token at position $t$. This objective supports understanding recovery after folding.

#### G.3.3 TEXT-TO-IMAGE TRAINING

T2I training requires more sophisticated techniques due to the complexity of visual generation. We adopt **offline knowledge distillation**, which decouples teacher inference from student training, reducing memory requirements by avoiding simultaneous loading of both models. Crucially, the choice of distillation objective is determined by each architecture's generation mechanism:

**Janus-Pro: Logit Distillation.** JANUS-PRO generates images by **autoregressively predicting VQ tokens** from a discrete visual vocabulary. At each generation step, the model outputs a probability distribution over the VQ codebook, and the next token is sampled from this distribution. Since the generation quality directly depends on the accuracy of these token-level predictions, we employ **KL divergence-based logit distillation** to align the student's output distribution with the teacher's:

**Step 1: Teacher Logit Extraction.** For each prompt $\mathbf{x}$, we first generate image tokens $\mathbf{v} = (v_1, \ldots, v_L)$ using the teacher model with Classifier-Free Guidance (CFG=5.0). We then extract teacher logits via teacher forcing:

$$\mathbf{Z}^{(t)} = \text{GenHead}\left(\text{LM}^{(t)}(\text{Embed}(\mathbf{x}) \oplus \text{Embed}(\mathbf{v}))\right). \tag{7}$$

To reduce storage overhead (from 112 MB to 1.1 MB per sample), we apply **Top-K sparse storage** ($K = 500$), retaining only the highest-probability logits at each position.

**Step 2: KL Divergence Training.** The student model is trained to match the teacher's softened distribution using temperature-scaled KL divergence:

$$\mathcal{L}_{\text{T2I}}^{\text{Janus}} = T^2 \cdot \frac{1}{L} \sum_{l=1}^{L} D_{\text{KL}}\left(P_t^{(l)} \| P_s^{(l)}\right), \tag{8}$$

where $P_t^{(l)}$ and $P_s^{(l)}$ are the temperature-softened distributions ($T = 2.0$) of teacher and student at position $l$. The $T^2$ factor maintains gradient magnitude consistency (Hinton et al., 2015).

**OmniGen2: Feature Distillation.** OMNIGEN2 adopts a **decoupled architecture** where the MLLM backbone serves as a condition encoder, and an external Diffusion Transformer (DiT) performs the actual image generation. The MLLM does not predict discrete tokens; instead, it **passes its final hidden states to the DiT** as conditioning signals. Since the DiT receives hidden state representations rather than token probabilities, aligning the student's hidden states with the teacher's is the most direct and effective approach. We therefore employ **MSE-based feature distillation**:

**Step 1: Teacher Hidden State Extraction.** For each prompt $\mathbf{x}$, we extract the teacher model's last-layer hidden states:

$$\mathbf{H}^{(t)} = \text{LM}_N^{(t)}(\text{Embed}(\mathbf{x})) \in \mathbb{R}^{L_x \times d}, \tag{9}$$

where $L_x$ is the text sequence length and $d$ is the hidden dimension. Hidden states are stored in compressed format ($\sim$1 MB per sample), significantly smaller than logits.

**Step 2: MSE Training.** The student model is trained to match the teacher's hidden state distribution using mean squared error:

$$\mathcal{L}_{\text{T2I}}^{\text{OmniGen}} = \frac{1}{L_x \cdot d} \sum_{l=1}^{L_x} \sum_{k=1}^{d} \left(H_{l,k}^{(s)} - H_{l,k}^{(t)}\right)^2, \tag{10}$$

where $\mathbf{H}^{(s)} \in \mathbb{R}^{L_x \times d}$ is the student hidden states. This approach converges faster than KL divergence due to the simpler optimization landscape.

#### G.3.4 LORA CONFIGURATION

STITCH LORA modules are applied only at fold points, where layer $i$ connects directly to layer $j$ ($j > i + 1$). The LoRA update is formulated as:

$$\mathbf{W}' = \mathbf{W} + \frac{\alpha}{r}\mathbf{B}\mathbf{A}, \tag{11}$$

where $r$ is the rank and $\alpha$ is the scaling factor. We use $r = 128$ and $\alpha = 256$ for all experiments. For a selected path $\mathcal{P}$, define its folding edges as $\mathcal{F}(\mathcal{P}) = \{(i, j) \in \mathcal{P} : j > i + 1\}$. We instantiate one healing module at each folding edge, so the module count is $|\mathcal{F}(\mathcal{P})|$. This definition produces the path-specific counts reported in Table 12. Because FOLDUMM freezes the backbone and repairs only fold-point transitions, the LoRA modules must absorb the feature mismatch introduced by non-adjacent layer connections. In preliminary rank sweeps, ranks no larger than 64 were insufficient for stable recovery under moderate pruning, while rank 128 provided more reliable performance gains. We therefore use rank 128 as a conservative default for the reported experiments.

#### G.3.5 OPTIMIZATION

During training, we freeze all backbone parameters and optimize only STITCH LORA weights. We use AdamW (Loshchilov & Hutter, 2019) with learning rate $1 \times 10^{-4}$, weight decay $0.01$, and cosine learning rate schedule with 10% warmup. All experiments are conducted on NVIDIA A6000 GPUs with effective batch size 32. To ensure robustness, we initialize LoRA weights with 3 different random seeds and report averaged results.

**Compute Resources.** Model weights and the primary Janus-Pro healing and inference paths use bfloat16 on NVIDIA A6000 GPUs. The OmniGen2 T2I distillation path retains bfloat16 model weights and cached targets while using float16 autocast for the DiT forward pass. Each method requires less than 12 GPU-hours in total for training, and the peak GPU memory usage is approximately 32GB. Profiling and path-search overhead is run offline before inference and is lightweight compared with healing training.

## H BASELINE IMPLEMENTATION DETAILS

For fair comparison, we adapt all baseline pruning methods to support both Image Understanding (IU) and Text-to-Image (T2I) tasks. All methods use the same calibration data $\mathcal{D}_t^{\text{cal}}$ as FOLDUMM (see Section G.3) for redundancy/importance analysis, and receive equivalent fine-tuning budgets.

### H.1 STRUCTURED LAYER PRUNING

**SHORTGPT (Men et al., 2025).** SHORTGPT identifies redundant layers using cosine Block Influence (BI), which measures the directional change introduced by a block. For layer $\ell$, the BI score is computed as:

$$\text{BI}_\ell = 1 - \mathbb{E}_{x,t} \frac{\mathbf{X}_{\ell,t}^\top \mathbf{X}_{\ell+1,t}}{\|\mathbf{X}_{\ell,t}\|_2 \|\mathbf{X}_{\ell+1,t}\|_2}, \tag{12}$$

where $\mathbf{X}_{\ell,t}$ and $\mathbf{X}_{\ell+1,t}$ are the token representations before and after block $\ell$. Layers with lower BI scores are considered more redundant and are removed first.

**UMM Adaptation:** Unlike the original method which uses pure text data, we compute BI scores using mixed calibration data from both IU and T2I tasks: $\mathcal{D}^{\text{cal}} = \mathcal{D}_{\text{IU}}^{\text{cal}} \cup \mathcal{D}_{\text{T2I}}^{\text{cal}}$. This joint computation identifies layers redundant for *both* task types, avoiding task-critical removals.

**SLEB (Song et al., 2024).** SLEB (Shortening LLMs by Eliminating Blocks) employs an iterative greedy search that removes one layer at a time based on task loss. At each iteration $t$, it selects the layer $\ell^*$ whose removal minimizes the loss:

$$\ell_t^* = \arg\min_{\ell \in \mathcal{S}_t} \; \mathcal{L}(\mathcal{M}^{\setminus \ell}; \mathcal{D}^{\text{cal}}), \tag{13}$$

where $\mathcal{S}_t$ is the set of remaining layers and $\mathcal{M}^{\setminus \ell}$ denotes the model with layer $\ell$ removed.

**UMM Adaptation:** We replace the single-task loss with a mixed objective:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{IU}}(\mathcal{D}_{\text{IU}}^{\text{cal}}) + \lambda \cdot \mathcal{L}_{\text{T2I}}(\mathcal{D}_{\text{T2I}}^{\text{cal}}), \tag{14}$$

with $\lambda = 1.0$ to equally weight both tasks. This ensures the greedy selection accounts for both understanding and generation capabilities.

**SHORT-LVLM (Ma et al., 2025a).** SHORT-LVLM extends layer pruning to vision-language models by combining layer similarity analysis with Subspace-based Component Projection (SCP) for feature compensation. The method uses the following layer-wise cosine-similarity score:

$$R(\ell) = \text{CosSim}(\mathbf{h}_{\ell-1}, \mathbf{h}_\ell), \tag{15}$$

where higher similarity indicates higher redundancy.

**UMM Adaptation:** We extend the method by applying a Min-Similarity rule across tasks. A layer is considered redundant only if it shows high redundancy in *both* IU and T2I tasks:

$$R_{\text{joint}}(\ell) = \min\left(R_{\text{IU}}(\ell),\ R_{\text{T2I}}(\ell)\right). \tag{16}$$

This conservative criterion prevents removing layers that are critical to either task.

### H.2 UNSTRUCTURED WEIGHT PRUNING

**WANDA (Sun et al., 2024).** WANDA (Weights And activations) performs unstructured weight pruning based on the product of weight magnitude and input activation norm. For weight $w_{ij}$ connecting input $i$ to output $j$, the importance score is:

$$S_{ij} = |w_{ij}| \cdot \|\mathbf{X}_{:,i}\|_2, \tag{17}$$

where $\mathbf{X}_{:,i}$ denotes the $i$-th column of the input activation matrix. Lower-scored weights are pruned.

**UMM Adaptation:** We compute activation norms using mixed calibration data. Specifically, we forward both IU and T2I samples through the model and average the activation statistics:

$$\|\mathbf{X}_{:,i}\|_2^{\text{mixed}} = \frac{1}{2}\left(\|\mathbf{X}_{:,i}^{\text{IU}}\|_2 + \|\mathbf{X}_{:,i}^{\text{T2I}}\|_2\right). \tag{18}$$

**SPARSEGPT (Frantar & Alistarh, 2023).** SPARSEGPT formulates pruning as a layer-wise reconstruction problem, minimizing the squared error between the original and pruned outputs. For each layer, it solves:

$$\min_{\hat{\mathbf{W}}} \|\mathbf{W}\mathbf{X} - \hat{\mathbf{W}}\mathbf{X}\|_2^2, \tag{19}$$

where $\mathbf{W}$ is the original weight matrix and $\hat{\mathbf{W}}$ is the pruned weight with remaining weights updated via approximate Hessian inverse to compensate for pruning errors.

**UMM Adaptation:** Unlike the original method which uses pure text (WikiText-2), we compute the Hessian approximation using mixed calibration data. The activation matrix $\mathbf{X}$ is constructed by streaming both IU image-text pairs and T2I prompts:

$$\mathbf{X} = [\mathbf{X}^{\text{IU}}; \mathbf{X}^{\text{T2I}}], \tag{20}$$

ensuring the pruning decisions account for both modalities.

## I EXTENDED RELATED WORK

This section provides a comprehensive review of efficiency optimization techniques for large multimodal models, contextualizing FOLDUMM within the broader research landscape.

**Dynamic Inference vs. Task-Aware Layer Folding.** A prominent direction in model acceleration is dynamic depth optimization, where the model decides per-token or per-sample which layers to execute at runtime. **LayerSkip** (Elhoushi et al., 2024) introduces self-speculative decoding with early exits, while **UniMoD** (Mao et al., 2025) and **MoDification** (Zhang et al., 2025a) extend the Mixture-of-Depths (MoD) mechanism to multimodal contexts using routers to skip blocks dynamically. While these methods achieve theoretical FLOPs reduction, they introduce significant system complexity: dynamic routing requires runtime verification loops, KV-cache management for variable paths, and router computation overhead, making them difficult to deploy on standard hardware. In contrast, FOLDUMM realizes the "One Backbone, Multiple Paths" paradigm through task-specific folding paths determined via offline analysis. Instead of making granular decisions for every token, we exploit the macro-level redundancy differences between tasks (IU vs. T2I). This shifts the optimization burden from runtime to offline preparation, resulting in deterministic computational graphs that require no specialized kernels or cache management, ensuring that theoretical FLOPs reductions translate directly into wall-clock speedups.

**Orthogonal Acceleration: Width vs. Depth.** While FOLDUMM reduces the *depth* of the model (layer count $N$), an orthogonal line of research focuses on reducing the *width* (token count $L$). Methods like **FastV** (Chen et al., 2024a) and **SparseVLM** (Zhang et al., 2025c) operate on the premise that visual tokens are redundant in deeper layers. FastV's "Visual Echo" hypothesis—that visual tokens are critical in shallow layers but ignored in deep layers—strongly aligns with our empirical finding that Image Understanding tasks exhibit high redundancy in the latter stages of the backbone. These token reduction techniques are orthogonal and complementary to FOLDUMM. A combined approach could theoretically achieve multiplicative efficiency gains (Speedup $\approx \text{Reduction}_{\text{width}} \times \text{Reduction}_{\text{depth}}$), offering a promising direction for future ultra-efficient UMMs.

**Theoretical Foundations: Layer Redundancy.** Our work is grounded in recent empirical studies that validate the non-uniform importance of transformer layers. Shi et al. (2025) identified "Vision Function Layers" (VFLs), showing that different visual capabilities (e.g., OCR vs. Recognition) activate at specific depths. Similarly, **ShortV** (Yuan et al., 2025) utilized a Layer Contribution metric to reveal that nearly 60% of MLLM layers are ineffective for visual processing. FOLDUMM operationalizes these observations by extending the concept of redundancy from a general property to a task-dependent divergence. By explicitly modeling the conflict between the redundancy patterns of Understanding and Generation, we formalize layer selection as a global optimization problem rather than heuristic removal.

**Structural Healing and Adaptation.** Directly removing layers creates feature distribution shifts that can degrade performance. Gromov et al. (2025) and **FlexiDepth** (Luo et al., 2025) demonstrated that lightweight fine-tuning (e.g., QLoRA) can effectively "heal" these breaks. **Short-LVLM** (Ma et al., 2025a) further proposed subspace projection to compensate for removed components. FOLDUMM integrates this insight through STITCH LORA. Unlike general fine-tuning, our approach is surgically targeted: we place LoRA modules specifically at the fold points identified by our DAG search. This ensures that the limited parameter budget is spent exactly where the feature mismatch is highest, improving the trade-off between parameter efficiency and performance restoration compared to uniform adaptation.

**Comparison with Execution Acceleration.** We distinguish structural reduction from execution acceleration. **Hyper-Bagel** (Lu et al., 2025) accelerates UMMs using speculative decoding and multi-stage diffusion distillation. These methods optimize *how* the full model is executed or sampled, whereas FOLDUMM optimizes *what* part of the model is executed. These strategies are mutually compatible; for instance, speculative decoding could be applied on top of a FOLDUMM-pruned backbone to achieve compound speedups, leveraging both structural sparsity and algorithmic efficiency.