HydraLLM: Adaptive Width Slicing for Elastic Text and Vision-Language Models

Oscar Kemme[1], Janek Haberer[2], Ali Hojjat[1,2], Olaf Landsiedel[1,2,3]
[1]Hamburg University of Technology (TUHH), Germany
[2]Kiel University, Germany
[3]UNU-INWEH, Germany
{oscar.kemme,ali.hojjat,olaf.landsiedel}@tuhh.de
janek.haberer@cs.uni-kiel.de

Abstract

LLMs and LVLMs are increasingly deployed across heterogeneous hardware, ranging from data-center GPUs to resource-constrained edge devices. Their fixed-size architectures limit the ability to reuse a single model across deployment settings with different compute, latency, and model-size targets. We introduce HydraLLM[1], an elastic MQA-based architecture. Its number of active attention heads determines the effective widths of the embeddings, attention projections, feed-forward layers, and the output head, forming a nested family of subnetworks within a single parameter set. We introduce a static-to-elastic conversion pipeline that, starting from a pre-trained model checkpoint, includes importance-driven weight reordering and stochastic subnetwork training with knowledge distillation. We extend this principle to multimodal inference through an elastic projection module whose output width follows the active text width. HydraLLM covers a wider range of active parameter configurations than FFN-only elastic architectures. With 75% of heads active, HydraLLM achieves 2.2 percentage points higher TriviaQA accuracy than a state-of-the-art nested baseline configuration which has 17.25% more parameters. With 50% of heads active, our multimodal model achieves a CLIPScore comparable to that of the 7× larger Gemma 3 4B model. With only 25% of heads active, HydraLLM increases token-generation throughput by 75.3% over the full-width configuration.

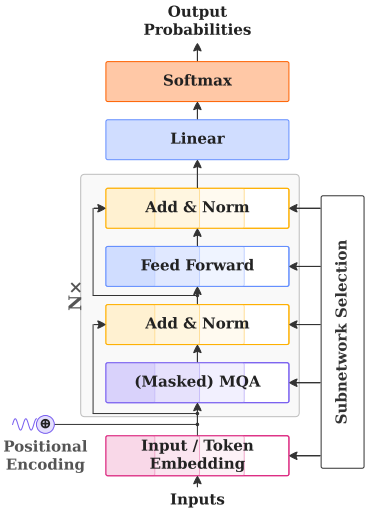

Figure 1: HydraLLM width slicing. The number of active heads determines the embedding dimension, the attention projection, and the feed-forward network width, producing nested subnetworks from a single parameter set.

1 Introduction

Large Language Models (LLMs) and Large Vision-Language Models (LVLMs) underpin applications such as document understanding (Deitke et al., 2024), conversational assistants (OpenAI, 2024), and agent systems (Google, 2024). Transformer-based architectures (Vaswani et al., 2017) exhibit robust scaling, driving the creation of larger foundation models. At the same time, their deployment spans heterogeneous hardware from data-center GPUs to resource-constrained edge devices with different latency, compute, and model-size requirements. Training separate models for each configuration is computationally prohibitive, as each variant must be trained independently from scratch. Additionally, conventional transformer-based models are inherently static at inference time, which means their parameters are fixed after training, limiting their ability to target heterogeneous systems. Recent elastic and nested methods address this by training multiple models as one, but most focus on a single component such as the feed-forward network (FFN) for size reduction or require a

[1]The source code is available at https://github.com/tuhh-ncps/hydrallm.

Proceedings of the Second BabyLM Workshop, pages 33–49
October 29, 2026 ©2026 Association for Computational Linguistics

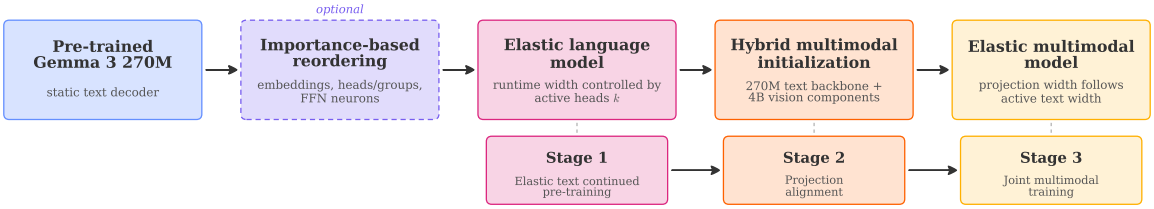

Figure 2: Pipeline overview. Starting from a static text checkpoint, such as Gemma 3 270M, the sliced axes can first be reordered to improve prefix quality. The resulting checkpoint is then trained as an elastic text model whose runtime width is controlled by the active number of query heads k. Finally, we combine the elastic text decoder with, for example, Gemma 3 4B vision components to form a multimodal model whose projection output follows the same active width.

multi-step extraction process of fixed model checkpoints (Haberer et al., 2024; Devvrit et al., 2024; Vaswani et al., 2017; Cai et al., 2020). In this paper, we introduce HydraLLM[1], a unified approach to adaptive slicing in LLMs and LVLMs by merging elastic architectural principles with efficient compression techniques while leveraging an already pre-trained static LLM.

We show that HydraViT-style (Haberer et al., 2024) end-to-end adaptive slicing transfers successfully to decoder-only LLMs, producing a single checkpoint that remains effective across several runtime width budgets. With 75% of heads active, HydraLLM improves Trivia-QA (Joshi et al., 2017) accuracy by 2.2 percentage points over a baseline configuration that has 17.25% more parameters. With 50% of heads active, the multimodal model reaches a CLIPScore of 66.9, surpassing the 65.3 score of the 7× larger Gemma 3 4B (Gemma Team et al., 2025) model. With only 25% of heads active, importance-based reordering improves post-training perplexity by 16.2%. This 25% configuration increases token-generation throughput by 75.29% compared to the full model. In summary, we contribute:

- **End-to-end slicing for decoder-only LLMs.** HydraLLM exposes a single head-count control that selects consistent prefixes of the embedding, attention, FFN, and output dimensions, producing nested decoder subnetworks from one checkpoint.
- **Unified training pipeline for elastic subnetworks.** We introduce a single training recipe to convert a static pre-trained LLM to an adaptively sliceable LLM while remaining competitive on text-only benchmarks.
- **Elastic multimodal projection for LVLMs.** We extend adaptive slicing to LVLMs by employing an elastic projection module that maps visual features into text embedding spaces of varying dimensionality.

2 Related Work

We discuss related work on nested representations and elastic architectures, transformer compression and submodel extraction, and projection-based multimodal models. We focus on how these approaches differ from HydraLLM's end-to-end runtime slicing of a pre-trained LLM and its extension to an elastic multimodal projection.

Kusupati et al. (2024) introduce multi-granularity, coarse-to-fine representations for embedding models. The same matryoshka nesting principle reappears in later work introducing elasticity within other transformer components. Cai et al. (2020) propose the Once-For-All (OFA) methodology, decoupling architecture search from training by progressively shrinking the convolutional neural network (CNN) kernel size, channel width, and model depth. OFA targets the search and extraction of specialized CNN architectures for different deployment constraints. In contrast, HydraLLM does not perform architecture search or progressive shrinking: it converts a pre-trained LLM into one elastic checkpoint whose embedding, attention, FFN, and output widths can be scaled jointly at runtime through the active head count. Devvrit et al. (2024) propose MatFormer, a natively elastic transformer that introduces a Matryoshka-like nested structure specifically *only* inside the FFN block. Haberer et al. (2024) target embeddings, attention heads, and FFN layers for their nested ViTs. Training employs stochastic subnetwork sampling by

drawing one head count k per step and updating only the corresponding subnetwork, which biases optimization toward earlier heads, implicitly ordering them by importance (Haberer et al., 2024). We apply the same training and architectural approach to LLMs and extend it to the vision domain. Cai et al. (2024) and Taghibakhshi et al. (2025) extend the nested approach in the attention and FFN components with routers in their FLEXTRON and Nemotron Elastic models. Both approaches are text-only. FLEXTRON slices only attention heads and FFN, and Nemotron Elastic targets an architecture distinct from our multi-query attention (MQA) (Shazeer, 2019) transformer. Muralidharan et al. (2024) design a static submodel extraction approach by applying iterative structured pruning on a pre-trained model, creating MINITRON. They obtain separate models that are not suitable for elastic inference, like ours.

ViTs (Dosovitskiy et al., 2021) adapt the transformer encoder to images by splitting an image into non-overlapping patches, flattening each patch, and projecting it into the model's embedding space to obtain patch embeddings. For multimodal models, Liu et al. (2023) connect a pre-trained CLIP (Radford et al., 2021) vision encoder to an open-source LLM via a projection layer that maps visual features into the LLM's text embedding space, producing so-called soft tokens. They merge the soft tokens with text tokens and process them jointly through self-attention. Deitke et al. (2024) build on these concepts with the Molmo family, while Marafioti et al. (2025) propose SmolVLM as a series of compact multimodal models. Gemma Team et al. (2025) release Gemma 3 with a tailored SigLIP (Zhai et al., 2023) vision encoder. We follow the training and architectural principles of these multimodal models and utilize insights from their training approaches. However, we employ the elastic LLM and multimodal projection.

3 HydraLLM: Architecture and Pipeline

This section introduces HydraLLM's design and the pipeline for converting a static pre-trained LLM into an elastic text and multimodal model. Specifically, we describe the text architecture and width-control in Section 3.1, the static-to-elastic conversion procedure including reordering and the optimization objective details in Section 3.2, and the multimodal assembly in Section 3.3. Our pipeline starts from a static pre-trained text checkpoint and converts it into an elastic decoder whose active width is controlled at runtime; see Figure 2. We then extend the same elasticity principle to the multimodal setting with an elastic projection module.

3.1 Runtime Width-Control

HydraLLM adapts HydraViT's (Haberer et al., 2024) head-stacking principle to decoder-only language models by using the number of active query heads k as a single runtime width-control parameter; see Figure 1. A subnetwork keeps the first k query heads together with the corresponding prefixes of the embedding channels, output projection, and FFN neurons. Each sliced dimension scales linearly with the number of heads k in relation to the total number of query heads H_{q}: the active embedding width is $d_{E,\mathrm{active}} = k/H_{\mathrm{q}} \times d_E$, with the FFN dimensions following the same proportion. This produces a strict nesting of subnetworks that share parameters across configurations. At inference, selecting k computes only the corresponding prefix, so one checkpoint serves all width budgets. We target MQA (Shazeer, 2019) in our design, in which all query heads share a single key-value head.

3.2 Static-to-Elastic Conversion

A static pre-trained checkpoint is not immediately suitable for prefix slicing. In an unmodified checkpoint, units of a component are usually not sorted by their importance. Since prefix slicing always retains the first units and discards later units, naïve slicing can remove important parameters and severely degrade smaller subnetworks. HydraLLM employs a two-stage procedure to convert static models into elastic ones. First, the pipeline reorders the checkpoint offline, moving important units toward the beginning of each sliced dimension so that prefix slicing retains the most informative parameters. Second, it optimizes the checkpoint using stochastic subnetwork training: during each training step, the pipeline samples a single k and updates only the corresponding active subnetwork. Since earlier units are active for more sampled values of k, they receive proportionally more gradient updates, reinforcing the importance ordering and adapting the shared parameters to all configured widths (Taghibakhshi et al., 2025; Muralidharan et al., 2024; Cai et al., 2024; Haberer et al., 2024; Devvrit et al., 2024).

3.2.1 Importance Scoring and Reordering

The reordering step occurs before any training and requires the definition of importance. We compare three importance measures: Activation magnitude (Muralidharan et al., 2024), variance-weighted consumer strength (VxC) adapted from Weinstein and Belinkov (2022), and CETT (Zhang et al., 2024), which scores units by their normalized contribution to the component output and therefore aligns well with prefix slicing. We estimate unit importance from activation statistics collected during a forward pass over a calibration text corpus. After computing the importance scores, we permute the corresponding tensors so that important units appear earlier along the sliceable dimensions: For embedding-width reordering, we apply one global permutation to the token embedding channels and propagate it consistently through the RMSNorm weights, the input columns of the Q/K/V projections, the output rows of the attention output projection, the input columns of the FFN gate and up projections, the output rows of the FFN down projection, the final normalization layer, and, if untied, the language-model head. For attention reordering, we permute the query heads and the corresponding attention-output input blocks. For FFN reordering, we permute FFN neurons by moving the rows of the gate and up projections and the corresponding columns of the down projection. This reordering step preserves the function of the full model.

3.2.2 Optimization Objective

After reordering, we train the model. The training objective we employ is a combination of autoregressive language modeling, logit regularization, and teacher supervision. The primary task loss, $\mathcal{L}_{\text{CE}}$, is standard next-token cross-entropy. We additionally add a z-loss term $\mathcal{L}_z$, weighted by λ_z (Chowdhery et al., 2022), and logit-based knowledge distillation (Hinton et al., 2015). Let z_t and z_s denote the teacher and student logits, respectively, and let $p_t^\tau = \text{softmax}(z_t/\tau)$ and $p_s^\tau = \text{softmax}(z_s/\tau)$ denote their temperature-scaled output distributions. The training objective is:

$$\begin{aligned} \mathcal{L}_{\text{KD}} &= \tau^2 \text{KL}(p_t^\tau \,\|\, p_s^\tau)\,, \\ \mathcal{L} &= \mathcal{L}_{\text{CE}} + \lambda_z \mathcal{L}_z + \lambda_{\text{KD}} \mathcal{L}_{\text{KD}} \end{aligned} \tag{1}$$

We keep the teacher model frozen throughout training.

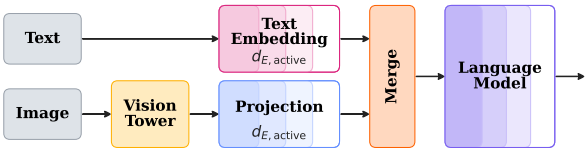

Figure 3: Elastic LVLM architecture. The projection output width follows the active text width.

3.3 Elastic Multimodal Assembly

We construct the multimodal variant of HydraLLM by combining the language decoder with a static pre-trained SigLIP-based vision encoder (Gemma Team et al., 2025; Zhai et al., 2023). The key difference from a conventional projection-based LVLM is that HydraLLM also makes the projection elastic. Let d_P denote the projection output width. We enforce $d_{P,\text{active}} = d_{E,\text{active}}$, so the same runtime parameter k controls both the active text decoder width and the projection output width; see Figure 3. This creates a single elasticity axis across modalities: selecting a smaller text subnetwork also selects a correspondingly smaller visual projection output.

4 Experiment Setup

We describe our training data and evaluation protocol in the following subsections 4.1 and 4.2, respectively.

4.1 Training Recipe

We base our experiments on Google Gemma 3 270M (Gemma Team et al., 2025), with $k \in \{1, 2, 3, 4\}$; see Appendix A.2. For text-only experiments, we use the frozen full-width Gemma 3 270M as the teacher model.

Most text experiments train on a subset of FineWeb-Edu 350BT (Lozhkov et al., 2024). We concatenate text samples with an <eos> separator and split the text into fixed-length token blocks. We employ non-uniform runtime width sampling of k, placing additional ($\times 3$) emphasis on the full-width model compared to the submodels; see Appendix A.5. In contrast to the packed text pretraining setup, the optional text fine-tuning stage uses text samples as they appear, without fully packing the sequence length; see Appendix C.1.

For multimodal training, we use the elastic HydraLLM Gemma 3 270m model combined with the vision encoder and projection module initialized from the Gemma 3 4B multimodal check-

Recipe	Perplexity ↓			
	H1	H2	H3	H4
no reordering	168.6M	1.1M	41.3k	49.0
mag-cett-cett	37.2k	2.43k	1.22k	49.0
mag-mag-mag	8.45M	378.9k	1.54k	49.0
vxc-vxc-vxc	38.0M	4.87M	60.4M	49.0

Table 1: Perplexity of **reordered** checkpoints measured on samples of plays by William Shakespeare. Magnitude for embeddings combined with CETT for heads and FFN yields the strongest subnetwork perplexities, while VxC consistently hurts.

Recipe	Perplexity ↓			
	H1	H2	H3	H4
no reordering	123.9	77.4	56.7	45.2
mag-cett-cett	106.6	71.6	54.4	44.3
mag-mag-mag	110.1	71.9	54.4	44.8
vxc-vxc-vxc	118.1	71.7	55.0	44.1

Table 2: Perplexity of **reordered and trained** (20k steps) checkpoints measured on samples of plays by William Shakespeare. Magnitude for embeddings combined with CETT for heads and FFN yields the strongest subnetwork perplexities.

point. The teacher for this training is the static and frozen Gemma 3 4B model. We employ uniform head sampling and separate learning rates for the text backbone, the projection module, and, optionally, the vision encoder. We configure separate peak learning rates and number of warmup steps for the text backbone, projection and vision encoder; see Appendix A.5. Multimodal training samples from PixMoCap, the COCO Recaptioned dataset, LLaVA Pretrain 558K (Liu, 2023), PixMo Ask Model Anything, PixMo Cap-QA, CoSyn-400K, and text-only data from Wikipedia (Wikimedia Foundation, 2023) and FineWeb-Edu. See Appendix A.3 for detailed composition. Projection alignment utilizes 128,000 captioning samples from the aforementioned captioning datasets.

4.2 Metrics and Evaluation Datasets

We employ a mix of metrics and benchmarks to evaluate models and derive conclusions. For text evaluation, we report ARC-Easy (Clark et al., 2018) and TriviaQA (Joshi et al., 2017). We evaluate ARC-Easy with both raw accuracy and answer-length-normalized accuracy. We evaluate TriviaQA with five-shot prompting and exact-match scoring. We measure perplexity (Jurafsky and Martin, 2026) on samples of plays by William Shakespeare (Balbay, 2024) as a general-purpose indicator of language modeling quality.

For multimodal evaluation, we report CLIPScore (Hessel et al., 2021; Radford et al., 2021) on COCO validation images and multiply the score by 100 for readability. We additionally use qualitative captioning examples to inspect generation quality across different head configurations. The standard error on text- and CLIPScore-benchmark results denotes standard errors over evaluation examples.

We measure throughput with `llama-bench` from `llama.cpp` (Gerganov and ggml-org contributors, 2025), with prompt-processing of 2048 tokens and autoregressive generation of 512 tokens on an NVIDIA RTX PRO 4000. This setup allows measuring prompt-processing and token generation speed improvements.

5 Evaluation

We evaluate the static-to-elastic conversion pipeline in terms of reordering in Section 5.1, compare HydraLLM with MatFormer-style (Devvrit et al., 2024) FFN slicing in terms of accuracy and throughput in Section 5.2, and assess the quality of multimodal extensions in Section 5.3.

5.1 Effect of Importance Reordering

To determine the effect of reordering, we measure perplexity for each head configuration and compare it to the unmodified pre-trained model in Table 1. We denote the k-head configurations as H1-H4. Reordering clearly improves the perplexity of submodels when utilizing the magnitude metric for embeddings combined with CETT for heads and FFN, or the magnitude metric for all components. Employing only VxC provides an improvement solely for the H1 configuration.

After reordering, we train the reordered models for 20,000 steps ($\sim$1B tokens) and measure their perplexities again. After training, perplexity improves for all heads, regardless of the starting point; see Table 2. However, the CETT importance metric for heads and FFN with magnitude importance for the embeddings provides the best results for the submodels. VxC slightly edges out CETT on the four-head configuration. For H4 of the not reordered model, continued training reduces perplexity from 49.0 to 45.2. However, this improvement does not consistently transfer to downstream benchmarks. Figure 4 and Figure 5 from the next Section 5.2 also show the ARC-Easy and TriviaQA

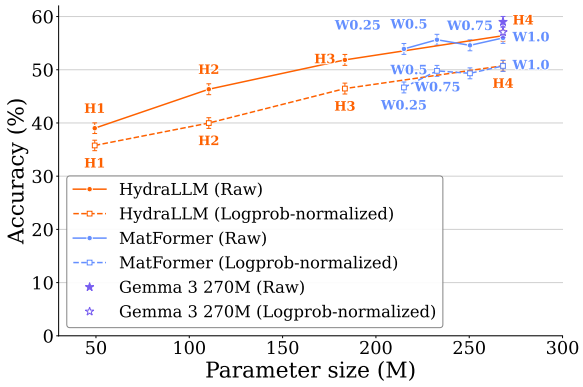

Figure 4: ARC-Easy accuracy as a function of active parameter size for HydraLLM and MatFormer configurations. HydraLLM covers a wider active parameter range than MatFormer while offering competitive accuracies.

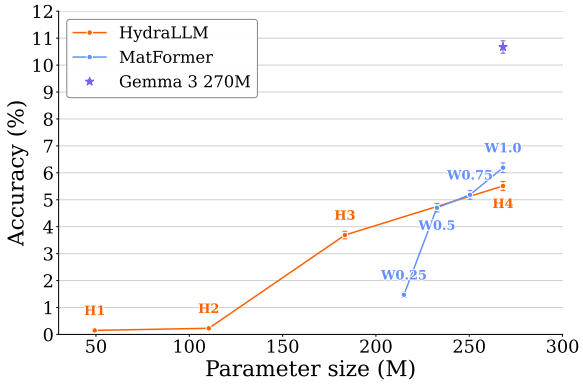

Figure 5: TriviaQA accuracy as a function of active parameter size for HydraLLM and MatFormer configurations. HydraLLM is stronger at H3 clearly outperforming the similarly sized MatFormer W0.25, while MatFormer surpasses HydraLLM at the full configuration.

accuracy of the pre-trained and the no-reordering model trained for 20,000 steps. The benchmarks actually show a lower accuracy compared to the static baseline. We provide more benchmark results in Appendix B.2.4.

5.2 HydraLLM vs. MatFormer

We compare HydraLLM to MatFormer-style FFN slicing, which we reimplement using the same Gemma 3 270M checkpoint, in terms of accuracy and runtime throughput. The hyperparameters for both models are the same, apart from the width selection mechanism. In Figures 4 and 5, we label the MatFormer FFN configurations as W0.25-W1.0, corresponding to the fraction of FFN neurons active. Because MatFormer only alters the FFN component, the smallest MatFormer configuration, W0.25, still exceeds the second-largest HydraLLM configuration, H3, by 17.25% in parameter count. The raw ARC-Easy accuracy of W0.25 is slightly higher than that of H3, though not statistically significant. The log-probability-normalized accuracies are virtually the same. The HydraLLM approach has a clear advantage with similar accuracies, but a wider parameter size coverage. The ten-shot prompting HellaSwag (Zellers et al., 2019) results lead to the same conclusion, therefore we provide them in Appendix B.1 only.

Config	HydraLLM		MatFormer	
	Prefill	Decode	Prefill	Decode
H1 / W0.25	55,891.3	1,057.0	51,549.1	669.6
H2 / W0.50	52,851.7	785.9	51,353.8	648.5
H3 / W0.75	51,165.4	594.2	49,945.2	623.4
H4 / W1.00	50,487.9	603.0	50,509.8	602.3

Table 3: Throughput in tokens per second for HydraLLM and MatFormer variants. Prompt-processing of 2048 tokens. Decode 512 tokens. Prompt-processing throughput gains are modest for both methods, while token-generation scaling is much larger for HydraLLM than for MatFormer.

The results on TriviaQA differ. While the W0.5 and W0.75 MatFormer configurations interpolate smoothly between the H4 and H3 HydraLLM configurations in Figure 5, H3 achieves 2.21 percentage points ($\times$2.51) higher accuracy than W0.25, although they are similar in size. Conversely, W1.0 outperforms H4 by 0.68 percentage points. The HydraLLM approach has a broader parameter spread, starting from 49.34M parameters, compared to MatFormer, starting from 215.01M parameters, while achieving competitive results with similar submodel sizes. However, submodels H1 and H2 only achieve close to 0% accuracy on this benchmark.

The prompt-processing improvements from smaller model configurations displayed in Table 3 are very modest. The improvement from the H4 to H1 configuration ranges from 9.78% to 11.61%, and the improvement from W1.0 to W0.25 ranges from 1.03% to 3.1%. The Gemma 3 270M model comprises $\sim$170 million embedding parameters and $\sim$100 million transformer block parameters. Since the embeddings are mainly a lookup operation, they do not contribute much to throughput, but they account for a large portion of the parameter savings in the HydraLLM approach (Gemma Team et al., 2025; Vaswani et al., 2017).

For token generation, MatFormer shows consistent speed-ups with shrinking parameter count. HydraLLM produces fewer tokens per second with the H3 configuration compared to H4, which we hypothesize stems from kernel-shape optimizations in llama.cpp. The improvement in token generation speed from the largest to the smallest configuration is 75.29% for HydraLLM and 11.17% for MatFormer. The main takeaway is that prompt-processing gains are minor, while token-generation scaling is much greater, but does not scale one-to-one with parameter count. The reported throughput gains therefore stem mostly from smaller embeddings and query, output, and FFN projections, because MQA (Shazeer, 2019) prohibits key-value cache savings from width reduction.

5.3 Multimodal Pipeline Results

The first stage in the multimodal training pipeline is aligning the projection with the text embedding space. The projection-only block of Table 4 shows that the untrained projection already produces a non-trivial CLIPScore (Hessel et al., 2021; Radford et al., 2021) across all head configurations, indicating that it inherits useful structure from the Gemma 3 4B vision tower. Training the projection in isolation yields only modest improvements over the untrained baseline. During training, we observe a plateau in the loss after 300 to 350 steps, depending on the head configuration; see more details in Appendix C.2. The rapid plateau indicates a lack of capacity in the projection layer, caused by the much smaller projection due to the lower text embedding dimension of the Gemma 3 270M model compared to Gemma 3 4B. The projection has 25% of the parameters of the Gemma 3 4B model and therefore possesses much less representational power (Gemma Team et al., 2025).

Joint training of the projection with the text model substantially improves the CLIPScore; see the projection-plus-text-model block of Table 4. The models trained from an already projection-pre-aligned checkpoint perform best across all head configurations, especially at H2, implying that aligning the projection before joint training of the projection and text model is beneficial. When training all model parameters together, the all-parameters block shows that results are similar across different starting points, indicating that joint optimization can compensate for an inferior starting projection.

Scaling the joint projection and text model training, as well as the full-parameter training, from 2,500 to 16,000 steps further closes the gap to the Gemma 3 4B reference; see the bottom block of Table 4. The full-parameter trainable run matches or exceeds Gemma 3 4B at H2, H3, and H4 and reaches 62.6 at H1. The results suggest that, on CLIPScore, the elastic model matches the vision-language capabilities of Gemma 3 4B across most subnetworks. The cause for this lies in the response style of the Gemma 3 4B model: it often responds to captioning prompts with question-answer-style continuations rather than direct descriptive sentences, which reduces its CLIP similarity score. CLIPScore is therefore sensitive to response style and does not, by itself, guarantee stable caption quality (Hessel et al., 2021).

Given the strong CLIPScore results of HydraLLM compared to Gemma 3 4B, we evaluate our model with MMMU (Yue et al., 2024). Our models consistently achieve random accuracy of around 25%, whereas the Gemma 3 4B model reaches about 39%. We illustrate the qualitative

Table 4: CLIPScore ($\times 100$) across head configurations for each multimodal training recipe. Bold marks the best score per head configuration. *non-elastic*: starting from pre-trained Gemma 3 270m and only head configuration k=4 is trained; *elastic*: starting from elastic text model and all head configurations are trained stochastically; *fine-tune*: an additional text-only fine-tuning step is applied before multimodal training; *pre-aligned*: the projection is pre-aligned on captioning data before joint training; *Gemma 3 4B*: the original frozen Gemma 3 4B multimodal model provided as a baseline.

Training Recipe	**H1**	**H2**	**H3**	**H4**
No training				
elastic	32.4	31.5	32.0	32.7
Gemma 3 4B	–	–	–	65.3
Projection only (500 steps)				
non-elastic	–	–	–	28.9
elastic	**38.2**	31.4	**31.1**	**33.5**
elastic, fine-tune	34.8	**34.2**	31.0	32.1
Projection + text model (2,500 steps)				
elastic	46.3	53.9	58.5	58.8
pre-aligned, non-elastic	–	–	–	62.5
pre-aligned	**47.4**	56.9	**61.3**	61.0
pre-aligned, fine-tune	47.1	**57.4**	59.1	**61.1**
All parameters (2,500 steps)				
elastic	50.2	59.4	**63.1**	**63.9**
pre-aligned	51.1	**60.7**	61.3	62.5
pre-aligned, fine-tune	**51.2**	60.4	61.5	63.6
From text fine-tune + pre-aligned (16,000 steps)				
Projection + text model	59.1	65.4	65.1	68.0
All parameters	**62.6**	**66.9**	**68.6**	**69.6**

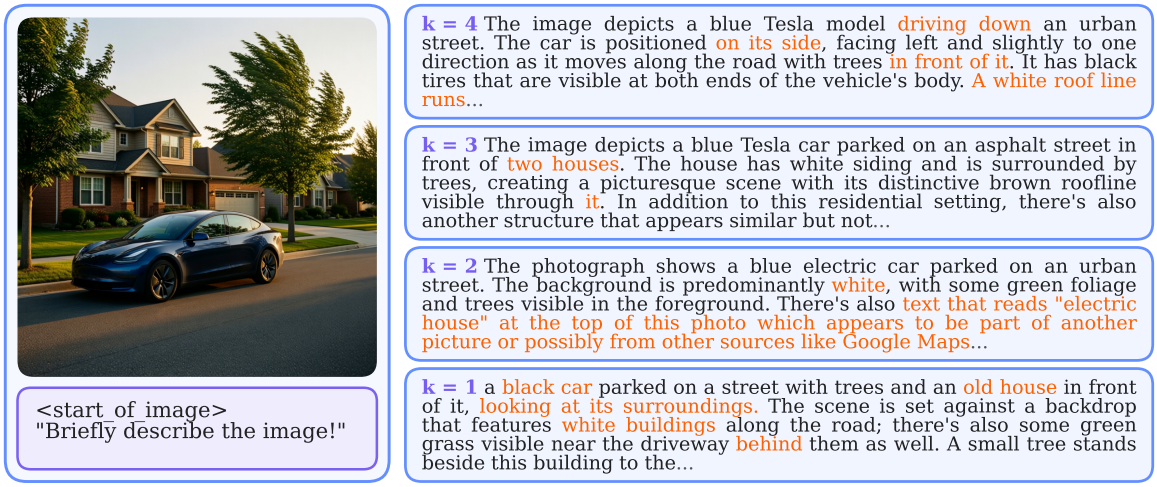

Figure 6: Qualitative captioning example. The elastic LVLM is prompted to describe an image of a car across the four head configurations ($k = 1$ to $k = 4$). Hallucinated or incorrect statements are highlighted in orange, illustrating the qualitative gap that persists alongside the encouraging CLIPScore trends.

behavior with the captioning example in Figure 6. We prompt the model to describe an image of a car. Across all head configurations, the captions include hallucinated or incorrect statements about the scene. Subject resolution, spatial relations, and object identity are unreliable: the four-head configuration hallucinates motion and orientation, the three-head and two-head configurations misidentify surrounding objects, and the single-head configuration misidentifies the car's color and introduces incorrect scene elements. Hallucinations, weak grounding, and unstable object descriptions therefore persist across all head configurations of HydraLLM. The elastic projection establishes feasibility, but the vision-to-language interface remains a suspected bottleneck and does not yet yield robust caption quality.

6 Conclusion

We introduce HydraLLM, an MQA-based (Shazeer, 2019) elastic architecture that employs stochastic head stacking and end-to-end width slicing. A single runtime parameter k controls the active width of the decoder end-to-end, from token embeddings through attention and FFN projections to the output head, producing nested subnetworks from a single parameter set. We extended the elasticity axis to LVLMs through an elastic projection module whose output width follows the active text width.

On text, magnitude-based reordering of embeddings combined with CETT-based (Zhang et al., 2024) reordering of heads and FFN units yields the strongest subnetworks, reducing perplexity by up to a factor of approximately 4,500 compared to no reordering. However, the effect is reduced to at most 14% after training. Knowledge distillation (Hinton et al., 2015) is crucial during training and improves TriviaQA exact-match accuracy at 75% of active heads by 97.3%. HydraLLM covers a substantially wider parameter range at competitive accuracy compared to MatFormer-style (Devvrit et al., 2024) FFN slicing. It achieves larger token-generation speed-ups of up to 75.3%, but only modest prompt-processing improvements.

On the multimodal side, scaled training matches or exceeds the Gemma 3 4B CLIPScore (Hessel et al., 2021; Radford et al., 2021) of 65.3 by 2.5%, 5.1% and 6.6% with two, three and four active heads respectively, confirming that the elastic text backbone transfers to the vision-language setting. However, the model remains near random accuracy on MMMU, and qualitative analysis reveals hallucinations and weak object grounding. These limitations point to clear avenues for future research: (i) replacing the constrained small single-layer projector with a larger or a multi-layer FFN projection, which would better align vision features with the elastic text backbone; (ii) longer and more diverse multimodal training schedules to improve grounding beyond captioning tasks; and (iii) combining HydraLLM's end-to-end slicing with MatFormer-style per-layer granularity for finer accuracy-compute control. We leave these directions for future work.

Acknowledgments

We thank our anonymous reviewers for their insights and suggestions for improvement. This work has received funding from the Federal Ministry of Research, Technology and Space under the REAKT ISA project (grant no. 03DPC1415B) and from the Federal Ministry for Economic Affairs and Energy under the CAPTN X-FERRY project (grant no. 03SX612A).

References

Allen Institute for AI. 2024a. PixMo-AskModelAnything. https://huggingface.co/datasets/allenai/pixmo-ask-model-anything. Hugging Face dataset, 161,737 rows, ODC-BY-1.0. Accessed: 2026-03-11.

Allen Institute for AI. 2024b. PixMo-Cap. https://huggingface.co/datasets/allenai/pixmo-cap. Hugging Face dataset. Image-to-text dataset, 717,042 rows. Accessed: 2026-03-11.

Allen Institute for AI. 2024c. PixMo-CapQA. https://huggingface.co/datasets/allenai/pixmo-cap-qa. Hugging Face dataset. Visual Question Answering dataset, 271,714 rows. Accessed: 2026-03-11.

Allen Institute for AI. 2025. CoSyn-400K. https://huggingface.co/datasets/allenai/CoSyn-400K. Hugging Face dataset. Visual Question Answering dataset, 408,227 rows. Accessed: 2026-03-11.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, and 45 others. 2025. Qwen3-vl technical report. *Preprint*, arXiv:2511.21631.

Alper Balbay. 2024. Calibration split of the tiny shakespeare dataset. GitHub repository file. Commit 5a58873, accessed 21 July 2026.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once-for-all: Train one network and specialize it for efficient deployment. *Preprint*, arXiv:1908.09791.

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

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, and 48 others. 2022. Palm: Scaling language modeling with pathways. *Preprint*, arXiv:2204.02311.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the AI2 reasoning challenge. *CoRR*, abs/1803.05457.

Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *Preprint*, arXiv:2307.08691.

Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, Jiasen Lu, Taira Anderson, Erin Bransom, Kiana Ehsani, Huong Ngo, YenSung Chen, Ajay Patel, Mark Yatskar, Chris Callison-Burch, and 31 others. 2024. Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models. *Preprint*, arXiv:2409.17146.

Devvrit, Sneha Kudugunta, Aditya Kusupati, Tim Dettmers, Kaifeng Chen, Inderjit Dhillon, Yulia Tsvetkov, Hannaneh Hajishirzi, Sham Kakade, Ali Farhadi, and Prateek Jain. 2024. Matformer: Nested transformer for elastic inference. *Preprint*, arXiv:2310.07707.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. *Preprint*, arXiv:2010.11929.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, and 197 others. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

Georgi Gerganov and ggml-org contributors. 2025. llama.cpp: LLM inference in C/C++. Build: 4df6e859e (7349).

Google. 2024. Gemini: Your personal ai assistant. https://gemini.google/assistant/. Accessed: 2026-02-03.

Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, Sourab Mangrulkar, Marc Sun, and Benjamin Bossan. 2022. Accelerate: Training and inference at scale made simple, efficient and adaptable. https://github.com/huggingface/accelerate.

Janek Haberer, Ali Hojjat, and Olaf Landsiedel. 2024. Hydravit: Stacking heads for a scalable vit. *Preprint*, arXiv:2409.17978.

Nathan Habib, Clémentine Fourrier, Hynek Kydlíček, Thomas Wolf, and Lewis Tunstall. 2023. Lighteval: A lightweight framework for llm evaluation.

Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. 2021. Clipscore: A reference-free evaluation metric for image captioning. *CoRR*, abs/2104.08718.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *Preprint*, arXiv:1503.02531.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *Preprint*, arXiv:1705.03551.

Daniel Jurafsky and James H. Martin. 2026. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, with Language Models*, 3rd edition. Online manuscript released January 6, 2026.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, and Ali Farhadi. 2024. Matryoshka representation learning. *Preprint*, arXiv:2205.13147.

Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. 2015. Microsoft coco: Common objects in context. *Preprint*, arXiv:1405.0312.

Haotian Liu. 2023. LLaVA-Pretrain. https://huggingface.co/datasets/liuhaotian/LLaVA-Pretrain. Hugging Face dataset; LLaVA Visual Instruct Pretrain dataset created in May 2023. Accessed: 2026-03-11.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning. *Preprint*, arXiv:2304.08485.

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and 1 others. 2024. FineWeb-Edu: The Finest Collection of Educational Web Data on Hugging Face.

Andrés Marafioti, Orr Zohar, Miquel Farré, Merve Noyan, Elie Bakouch, Pedro Cuenca, Cyril Zakka, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, Vaibhav Srivastav, Joshua Lochner, Hugo Larcher, Mathieu Morlon, Lewis Tunstall, Leandro von Werra, and Thomas Wolf. 2025. Smolvlm: Redefining small and efficient multimodal models. *Preprint*, arXiv:2504.05299.

Saurav Muralidharan, Sharath Turuvekere Sreenivas, Raviraj Joshi, Marcin Chochowski, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, Jan Kautz, and Pavlo Molchanov. 2024. Compact language models via pruning and knowledge distillation. *Preprint*, arXiv:2407.14679.

OpenAI. 2024. Chatgpt overview. https://chatgpt.com/overview?openaicom_referred=true. Accessed: 2026-02-28.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, and 2 others. 2019. Pytorch: An imperative style, high-performance deep learning library. *Preprint*, arXiv:1912.01703.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning transferable visual models from natural language supervision. *Preprint*, arXiv:2103.00020.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 3505–3506, New York, NY, USA. Association for Computing Machinery.

Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need. *Preprint*, arXiv:1911.02150.

Ali Taghibakhshi, Sharath Turuvekere Sreenivas, Saurav Muralidharan, Ruisi Cai, Marcin Chochowski, Ameya Sunil Mahabaleshwarkar, Yoshi Suhara, Oluwatobi Olabiyi, Daniel Korzekwa, Mostofa Patwary, Mohammad Shoeybi, Jan Kautz, Bryan Catanzaro, Ashwath Aithal, Nima Tajbakhsh, and Pavlo Molchanov. 2025. Nemotron elastic: Towards efficient many-in-one reasoning llms. *Preprint*, arXiv:2511.16664.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

Berry Weinstein and Yonatan Belinkov. 2022. Variance pruning: Pruning language models via temporal neuron variance.

Wikimedia Foundation. 2023. Wikimedia Downloads. Referenced via the Hugging Face wikimedia/wikipedia dataset card. Accessed: 2026-03-11.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Huggingface's transformers: State-of-the-art natural language processing. *Preprint*, arXiv:1910.03771.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, Cong Wei, Botao Yu, Ruibin Yuan, Renliang Sun, Ming Yin, Boyuan Zheng, Zhenzhu Yang, Yibo Liu, Wenhao Huang, and 3 others. 2024. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. *Preprint*, arXiv:2311.16502.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *Preprint*, arXiv:1905.07830.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. 2023. Sigmoid loss for language image pre-training. *Preprint*, arXiv:2303.15343.

Zhengyan Zhang, Yixin Song, Guanghui Yu, Xu Han, Yankai Lin, Chaojun Xiao, Chenyang Song, Zhiyuan Liu, Zeyu Mi, and Maosong Sun. 2024. Relu^2 wins: Discovering efficient activation functions for sparse llms. *Preprint*, arXiv:2402.03804.

Table 5: Key Python package versions used in the implementation.

Package	Version
torch	2.8.0
torchvision	0.23.0
transformers	4.56.1
accelerate	1.10.1
datasets	4.2.0
lighteval	0.11.0

A Reproducibility Details

This section provides implementation and hardware details, runtime configurations, multimodal dataset composition, CLIPScore evaluation details, and training hyperparameters.

A.1 Implementation and Hardware

We build our implementation around PyTorch (Paszke et al., 2019) and Hugging Face Transformers (Wolf et al., 2020). Accelerate (Gugger et al., 2022) and DeepSpeed (Rasley et al., 2020) support distributed training, and LightEval (Habib et al., 2023) integrates the text benchmarks. CLIPScore follows Hessel et al. (2021). When available, we use optional acceleration libraries such as FlashAttention 2 (Dao, 2023), while ensuring a transparent fallback to the standard PyTorch and Transformers backends. Table 5 lists exact package versions.

Text-only experiments run on two NVIDIA RTX PRO 4000 GPUs with at least 24 GB of VRAM each. If we disable the teacher model, text-only training fits on a single GPU. Multimodal training is more resource-demanding: the unified multimodal pipeline requires at least 48 GB of VRAM, and joint training of all parameters requires roughly 60 GB. The multimodal experiments therefore run on two NVIDIA Tesla L40 GPUs or two NVIDIA A100 80 GB GPUs. Throughput measurements use `llama.cpp` build `4df6e859e` (7349) (Gerganov and ggml-org contributors, 2025).

A.2 Runtime Configurations

Table 6 lists the active component dimensions for each head configuration $k \in \{1, 2, 3, 4\}$.

	H1	H2	H3	H4
H_{kv}	1	1	1	1
query heads k	1	2	3	4
$d_{E,\text{active}}$	160	320	480	640
$d_{\text{ff},\text{active}}$	512	1024	1536	2048
$d_{Q,\text{active}}$	256	512	768	1024

Table 6: HydraLLM's Gemma 3 270M runtime configurations and active component dimensions. $d_{E,\text{active}}$ is the active embedding width, $d_{\text{ff},\text{active}}$ the active FFN neuron count, and $d_{Q,\text{active}}$ the total query projection dimension. MatFormer's W0.25-W1.0 have the same $d_{\text{ff},\text{active}}$ as H1-H4, but otherwise keep the dimensions of H4.

A.3 Multimodal Dataset Composition

Tables 7 and 8 report the dataset mixtures for the main 2,500-step multimodal runs and the scaled 16,000-step runs, respectively. The scaled setting preserves the same broad mixture of captioning, question answering, code generation, and text-only data, but increases the number of samples. The mixtures comprise PixMo-Cap (Allen Institute for AI, 2024b), our recaptioned version of COCO (Lin et al., 2015), LLaVA Pretrain (Liu, 2023), PixMo Ask Model Anything (Allen Institute for AI, 2024a), PixMo Cap-QA (Allen Institute for AI, 2024c), the VQA and code subsets of CoSyn-400K (Allen Institute for AI, 2025), and text-only samples from Wikipedia (Wikimedia Foundation, 2023) and FineWeb-Edu (Lozhkov et al., 2024).

Table 7: Composition of the multimodal dataset of the main multimodal experiments.

Dataset	Num Samples
Captioning & Pretraining	
PixMo-Cap	50,000
COCO Recaptioned	12,000
LLaVA Pretrain	50,000
Question Answering	
PixMo Ask Model Anything	30,000
PixMo Cap-QA	30,000
CoSyn 400K VQA	100,000
Code Generation	
CoSyn 400K Code	3,200
Text Only	
Wikipedia EN	14,800
FineWeb-Edu 10BT	30,000
Total	320,000

Projection alignment trains on a separate mixture of 64,000 samples from COCO Recaptioned and 64,000 samples from PixMo-Cap, for a total of 128,000 samples. We generate COCO Recaptioned locally by recaptioning COCO training images with Qwen 3 VL 4B Instruct (Bai et al., 2025) utilizing the fixed descriptive instruction "Describe the image in 3-5 sentences!". The resulting captions are denser and more uniform than the original short COCO annotations.

Table 8: Composition of the multimodal dataset of the scaled 16,000-step multimodal training runs.

Dataset	Num Samples
Captioning & Pretraining	
PixMo-Cap	500,000
COCO Recaptioned	100,000
LLaVA Pretrain	500,000
Question Answering	
PixMo Ask Model Anything	125,000
PixMo Cap-QA	125,000
CoSyn 400K VQA	1,000,000
Code Generation	
CoSyn 400K Code	50,000
Text Only	
Wikipedia EN	200,000
FineWeb-Edu 10BT	200,000
Total	2,800,000

A.4 CLIPScore Evaluation

The CLIPScore describes how to measure the caption quality of a model, but it does not define the underlying dataset or the CLIP-based (Radford et al., 2021) model (Hessel et al., 2021). We use the entire COCO validation set to calculate the similarity score with clip-vit-large-patch14 (Radford et al., 2021).

A.5 Training Hyperparameters

Table 9 summarizes the hyperparameters for the text-only and default multimodal training runs. The scaled multimodal runs retain the settings of the corresponding 2,500-step runs, but train for 16,000 steps on the larger dataset mixture in Table 8, optionally train the vision encoder at a learning rate of 6×10^{-6} and reduce the text model warmup from 125 to 75 steps, while, if enabled, adding 75 warmup steps for the vision encoder. Compared with text-only training, we employ a lower text-backbone learning rate, a separate projection learning rate, a larger effective batch size, a larger knowledge-distillation weight when distilling from Gemma 3 4B, and uniform head sampling instead of the non-uniform text-only sampling weights for both multimodal runs. The sampling weight array assigns a weight to each head for random sampling.

B Additional Text Experiments

This section presents the text training ablations, followed by the importance metrics and additional reordering results.

B.1 Training Recipe Ablation

Table 10 reports the full text-training ablation across ARC-Easy, HellaSwag, and TriviaQA. We evaluate ARC-Easy and HellaSwag with both raw and answer-length-normalized accuracy with zero- and ten-shot prompting respectively, and TriviaQA with five-shot exact-match accuracy. Relative to the default recipe, the ablations disable the teacher, use uniform head sampling, restrict training to head configurations $\{2, 3, 4\}$, increase the knowledge-distillation weight to 0.3, use the FineWeb-Edu 10BT subset, apply an EMA decay of 0.999, increase the effective batch size to 128, or reduce the learning rate to 2×10^{-5}. The long-run variant instead extends training to one million steps and uses a larger FineWeb-Edu 350BT subset.

B.2 Importance Reordering

This section describes the three importance metrics we implement to reorder embedding channels, attention heads, and FFN neurons before stochastic subnetwork training. We compute importance scores from activation statistics collected during a forward pass over a calibration text corpus (Balbay, 2024) using forward hooks (Taghibakhshi et al., 2025). Each metric ranks units independently for each component. We then apply the resulting orders to the embedding dimension, the query-head axis, and the FFN intermediate axis, as described in Section 3.2.1.

B.2.1 Magnitude-Based Importance

The simplest criterion is activation magnitude (Taghibakhshi et al., 2025; Muralidharan et al., 2024). It assumes that units with consistently large activations contribute more strongly than units with small activations.

For FFN neurons, let $g^{(\ell)}_{b,t,j}$ denote the j-th channel of the gate-projection output in layer ℓ, for batch element b and sequence position t. The magnitude score is:

$$s^{\mathrm{mag}}_{j,\ell} = \sum_{b,t} |g^{(\ell)}_{b,t,j}| \tag{2}$$

For attention heads, we score the query-side projection slice. If $q^{(\ell)}_{b,t,h} \in \mathbb{R}^{d_h}$ denotes the query activation of head h, then:

$$s^{\mathrm{mag}}_{h,\ell} = \sum_{b,t} \|q^{(\ell)}_{b,t,h}\|_2 \tag{3}$$

For embedding dimensions, we compute magnitude from the normalized hidden states at the output of the pre-layer normalization. Let $x^{(\ell)}_{b,t,c}$ denote

Table 9: Training hyperparameters for the text-only and default multimodal runs. The full multimodal run where all components are trained differs only in vision encoder learning rate (6×10^{-6}) and warmup steps (text: 75, vision: 75). Where not listed, the multimodal configuration inherits the text default.

Hyperparameter	**Text**	**Multimodal**
Sequence length	3072	3072
Learning rate (text backbone)	3×10^{-4}	2×10^{-5}
Learning rate (projection)	–	2×10^{-4}
LR schedule	cosine	cosine
Weight decay	0.1	0.1
Optimizer	AdamW	AdamW
Adam betas	[0.9, 0.999]	[0.9, 0.999]
Adam epsilon	1×10^{-8}	1×10^{-8}
Warmup ratio (text)	0.01	–
Warmup steps (text model)	–	125
Warmup steps (projection)	–	25
Training steps	20,000	2,500
Effective batch size	16	128
Gradient clipping (norm)	1.0	1.0
Head selection	[1, 2, 3, 4]	[1, 2, 3, 4]
Sampling weights	[1, 1, 1, 3] (non-uniform)	[1,1,1,1] (uniform)
Loss	CE + z-loss + KD	CE + z-loss + KD
z-loss alpha	1×10^{-5}	1×10^{-5}
KD weight	0.1	0.2
KD temperature	2	2
Teacher model	Gemma 3 270M	Gemma 3 4B
Precision	bf16	bf16
Random seed	42	42

embedding channel c after normalization in layer ℓ. The layer-local score is:

$$s^{\text{mag}}_{c,\ell} = \sum_{b,t} |x^{(\ell)}_{b,t,c}| \tag{4}$$

Because we consistently slice embedding dimensions across the entire decoder, we sum layer-local scores into a single global ranking:

$$s^{\text{mag}}_{c} = \sum_{\ell} s^{\text{mag}}_{c,\ell} \tag{5}$$

Magnitude scoring is simple and inexpensive, but it does not account for how strongly downstream weights consume a unit's output.

B.2.2 Variance × Consumer Norms

The second metric, VxC, combines activation variance (Weinstein and Belinkov, 2022) with downstream consumer strength. The intuition is that a unit should rank highly if it varies across inputs and if strong outgoing weights amplify its output. Given activations z_i, the empirical variance is $\text{Var}(z_i) = \mathbb{E}[z_i^2] - \mathbb{E}[z_i]^2$, where the expectation is taken over the calibration data.

For FFN neurons, VxC multiplies the gate-projection activation variance by the squared norm of the corresponding down-projection column:

$$s^{\text{vxc}}_{j,\ell} = \text{Var}_{b,t}\left(g^{(\ell)}_{b,t,j}\right) \cdot \|W_{\text{down},\ell}[:, j]\|_2^2 \tag{6}$$

For attention heads, we first sum the channel-wise variance within each query head:

$$v_{h,\ell} = \sum_{d=1}^{d_h} \text{Var}_{b,t}\left(q^{(\ell)}_{b,t,h,d}\right) \tag{7}$$

Then we weight this term by the squared Frobenius norm of the corresponding output-projection slice $W^O_{\ell,h}$:

$$s^{\text{vxc}}_{h,\ell} = v_{h,\ell} \cdot \|W^O_{\ell,h}\|_F^2 \tag{8}$$

Several projections inside a transformer block consume the embedding channels. Let $\mathcal{C}_\ell = \{W^Q_\ell, W^K_\ell, W^V_\ell, W^{\text{gate}}_\ell, W^{\text{up}}_\ell\}$ denote these consumers. The per-layer embedding score is:

$$s^{\text{vxc}}_{c,\ell} = \text{Var}_{b,t}\left(x^{(\ell)}_{b,t,c}\right) \cdot \sum_{W \in \mathcal{C}_\ell} \|W[:, c]\|_2^2 \tag{9}$$

To obtain one global ranking, we sum the embedding scores across layers, just like we do with magnitude scoring. In comparison to magnitude, VxC directly captures a unit's role in the computation graph. Units that fluctuate significantly but have weak consumption receive lower rankings than those whose variation is amplified by large outgoing weights.

Table 10: Training recipe ablation across text benchmarks after conversion of the static Gemma 3 270M model, with MatFormer as a width-pruning comparison. Values are accuracies in percent, with standard error, for the four runtime head configurations H1–H4 and MatFormer widths W0.25–W1.0. ARC-Easy Norm is answer-length-normalized accuracy. TriviaQA is five-shot exact-match accuracy.

Variant	**H1/W0.25**	**H2/W0.5**	**H3/W0.75**	**H4/W1.0**
ARC-Easy (Raw)				
MatFormer	53.9 ± 1.02	55.6 ± 1.02	54.6 ± 1.02	56.0 ± 1.02
default	39.0 ± 1.00	46.3 ± 1.02	51.9 ± 1.03	56.4 ± 1.02
1M steps	43.6 ± 1.02	51.1 ± 1.03	56.9 ± 1.02	60.3 ± 1.00
no teacher	37.2 ± 1.00	46.2 ± 1.03	51.9 ± 1.02	56.1 ± 1.02
uniform sampling	38.8 ± 0.99	48.2 ± 1.02	52.2 ± 1.02	55.4 ± 1.02
heads 2, 3, 4	26.8 ± 0.91	46.2 ± 1.02	52.3 ± 1.03	57.8 ± 1.01
KD=0.3	37.9 ± 1.00	46.3 ± 1.02	52.9 ± 1.02	58.8 ± 1.01
FW-Edu 10BT	38.3 ± 1.00	45.7 ± 1.02	52.6 ± 1.02	56.7 ± 1.02
EMA=0.999	26.8 ± 0.91	28.3 ± 0.92	29.3 ± 0.93	54.2 ± 1.02
batch size=128	31.5 ± 0.95	39.2 ± 1.00	47.3 ± 1.02	57.3 ± 1.02
lower LR=2e-5	30.4 ± 0.94	34.4 ± 0.97	42.1 ± 1.01	57.4 ± 1.01
ARC-Easy (Norm)				
MatFormer	46.7 ± 1.02	49.8 ± 1.03	49.3 ± 1.03	50.7 ± 1.03
default	35.8 ± 0.98	40.0 ± 1.01	46.5 ± 1.02	50.8 ± 1.03
1M steps	38.6 ± 1.00	44.5 ± 1.02	49.5 ± 1.03	52.7 ± 1.02
no teacher	34.4 ± 0.99	40.5 ± 1.01	46.5 ± 1.02	49.7 ± 1.03
uniform sampling	36.3 ± 0.97	41.5 ± 1.01	46.8 ± 1.02	49.0 ± 1.03
heads 2, 3, 4	28.5 ± 0.92	41.2 ± 1.01	47.0 ± 1.02	51.0 ± 1.03
KD=0.3	34.7 ± 0.98	40.2 ± 1.01	48.4 ± 1.03	51.9 ± 1.03
FW-Edu 10BT	36.1 ± 0.99	39.9 ± 1.00	48.2 ± 1.03	51.8 ± 1.03
EMA=0.999	27.6 ± 0.92	27.7 ± 0.92	28.8 ± 0.93	51.4 ± 1.03
batch size=128	30.9 ± 0.95	34.8 ± 0.98	42.2 ± 1.01	50.2 ± 1.03
lower LR=2e-5	30.8 ± 0.95	32.2 ± 0.96	37.8 ± 1.00	52.8 ± 1.02
HellaSwag (Raw)				
MatFormer	29.2 ± 0.45	30.1 ± 0.46	30.7 ± 0.46	31.3 ± 0.46
default	26.2 ± 0.44	26.9 ± 0.44	29.0 ± 0.45	31.1 ± 0.46
1M steps	26.3 ± 0.44	28.2 ± 0.45	30.7 ± 0.46	32.5 ± 0.47
no teacher	26.0 ± 0.44	27.1 ± 0.44	28.9 ± 0.45	31.0 ± 0.46
uniform sampling	26.2 ± 0.44	27.1 ± 0.44	28.9 ± 0.45	29.9 ± 0.46
heads 2, 3, 4	25.7 ± 0.44	27.1 ± 0.44	29.1 ± 0.45	31.2 ± 0.46
KD=0.3	25.7 ± 0.44	26.9 ± 0.44	28.8 ± 0.45	31.1 ± 0.46
FW-Edu 10BT	25.9 ± 0.44	26.9 ± 0.44	28.6 ± 0.45	31.2 ± 0.46
EMA=0.999	25.7 ± 0.44	25.6 ± 0.44	26.1 ± 0.44	31.6 ± 0.46
batch size=128	25.7 ± 0.44	26.3 ± 0.44	27.6 ± 0.45	32.1 ± 0.47
lower LR=2e-5	25.5 ± 0.43	26.3 ± 0.44	27.0 ± 0.44	33.7 ± 0.47
HellaSwag (Norm)				
MatFormer	31.6 ± 0.46	33.6 ± 0.47	34.6 ± 0.47	36.1 ± 0.48
default	26.4 ± 0.44	28.3 ± 0.45	31.2 ± 0.46	35.6 ± 0.48
1M steps	27.4 ± 0.44	30.3 ± 0.46	34.4 ± 0.47	38.4 ± 0.49
no teacher	26.2 ± 0.44	27.9 ± 0.45	30.7 ± 0.46	35.2 ± 0.48
uniform sampling	26.5 ± 0.44	28.2 ± 0.45	31.0 ± 0.46	33.5 ± 0.47
heads 2, 3, 4	26.0 ± 0.44	28.3 ± 0.45	32.2 ± 0.46	36.4 ± 0.48
KD=0.3	26.0 ± 0.44	27.8 ± 0.45	31.3 ± 0.46	35.8 ± 0.48
FW-Edu 10BT	26.3 ± 0.44	27.7 ± 0.45	31.3 ± 0.46	35.7 ± 0.48
EMA=0.999	26.0 ± 0.44	26.1 ± 0.44	26.3 ± 0.44	37.2 ± 0.48
batch size=128	24.8 ± 0.43	27.1 ± 0.44	29.7 ± 0.46	38.0 ± 0.48
lower LR=2e-5	25.5 ± 0.43	26.3 ± 0.44	29.1 ± 0.45	40.8 ± 0.49
TriviaQA				
MatFormer	1.47 ± 0.09	4.70 ± 0.16	5.18 ± 0.17	6.19 ± 0.18
default	0.15 ± 0.03	0.23 ± 0.04	3.69 ± 0.14	5.51 ± 0.17
1M steps	0.13 ± 0.03	0.66 ± 0.06	2.35 ± 0.11	3.55 ± 0.14
no teacher	0.01 ± 0.01	0.02 ± 0.01	1.87 ± 0.10	2.16 ± 0.11
uniform sampling	0.14 ± 0.03	0.33 ± 0.04	3.38 ± 0.13	5.48 ± 0.17
heads 2, 3, 4	0.00 ± 0.00	0.25 ± 0.04	3.81 ± 0.14	6.50 ± 0.18
KD=0.3	0.14 ± 0.03	0.18 ± 0.03	3.72 ± 0.14	6.88 ± 0.19
FW-Edu 10BT	0.11 ± 0.02	0.22 ± 0.04	2.89 ± 0.12	6.12 ± 0.18
EMA=0.999	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.01	3.90 ± 0.14
batch size=128	0.00 ± 0.00	0.17 ± 0.03	2.67 ± 0.12	7.17 ± 0.19
lower LR=2e-5	0.00 ± 0.00	0.02 ± 0.01	1.33 ± 0.09	8.00 ± 0.20

B.2.3 CETT-Based Importance

Finally, the metric based on CETT (Zhang et al., 2024) measures the extent to which tail truncation affects the component output. We adapt this idea into per-unit scores based on the norm of each unit's direct contribution to the output.

For FFN neurons, let $h^{(\ell)}_{b,t,j}$ denote the j-th component of the input to the down projection in layer ℓ, and let $W_{\text{down},\ell}[:, j]$ denote the corresponding output weight vector. The contribution norm of neuron j is:

$$\begin{aligned} c^{(\ell)}_{b,t,j} &= \|h^{(\ell)}_{b,t,j} W_{\text{down},\ell}[:, j]\|_2 \\ &= |h^{(\ell)}_{b,t,j}| \cdot \|W_{\text{down},\ell}[:, j]\|_2 \end{aligned} \tag{10}$$

For attention heads, let $u^{(\ell)}_{b,t,h} \in \mathbb{R}^{d_h}$ denote the output of head h before the output projection, and let $W^O_{\ell,h} \in \mathbb{R}^{d_{\text{model}} \times d_h}$ be the corresponding output-projection slice. The head contribution norm is:

$$c^{(\ell)}_{b,t,h} = \|W^O_{\ell,h} u^{(\ell)}_{b,t,h}\|_2 \tag{11}$$

The unnormalized CETT score is the empirical mean contribution norm. The normalized variant divides the contribution norm by the norm of the full component output at the same token position. For FFN neurons, this gives:

$$s^{\text{cett}}_{j,\ell} = \mathbb{E}_{b,t}\left[\frac{c^{(\ell)}_{b,t,j}}{\|y^{\text{FFN},(\ell)}_{b,t}\|_2}\right] \tag{12}$$

For attention heads, this gives:

$$s^{\text{cett}}_{h,\ell} = \mathbb{E}_{b,t}\left[\frac{c^{(\ell)}_{b,t,h}}{\|y^{\text{Attn},(\ell)}_{b,t}\|_2}\right] \tag{13}$$

The normalized form is the most relevant variant for prefix slicing because it estimates a unit's contribution relative to the total component output. Since adaptive slicing removes the tail of a dimension, this ranking directly targets the expected relative output error incurred when we discard low-ranked units.

B.2.4 Effect of Importance Reordering

Table 11 complements the perplexity comparison in Section 5.1 with downstream accuracy after 20,000 training steps. Each row corresponds to a different combination of importance metrics for embeddings, attention heads, and FFN units. Mag-CETT-CETT improves TriviaQA exact-match accuracy across all head configurations, while ARC-Easy and HellaSwag remain close to the no-reordering baseline.

C Additional Multimodal Results

This section provides additional results on text recovery after multimodal training and the optimization behavior of the elastic projection during alignment.

C.1 Text Recovery after Multimodal Training

Table 12 lists TriviaQA accuracies after text-only continued pre-training and after subsequent multimodal or text-only fine-tuning stages. The comparison illustrates how teacher choice and continued fine-tuning affect recovery of factual recall after the multimodal stage. The text-only baseline is trained for one million steps. We compare it with the default multimodal run and with two text fine-tuning variants that use either Gemma 3 270M or Gemma 3 4B as the teacher.

The Gemma 3 4B teacher consistently outperforms the 270M teacher at the four-head configuration, suggesting that the larger teacher transfers more factual knowledge to the student. The improved accuracy relative to the long-run text baseline is likely due to the shift from packed context-window training to training on individual padded samples, together with the lower learning rate and larger batch size used in this stage.

C.2 Projection Alignment Loss Curves

Figure 7 shows the training loss for each head configuration during projection alignment. The one-head configuration converges to a substantially higher loss than the wider configurations, indicating that the single-head subnetwork reaches its capacity limit earlier. Configurations with two or more heads quickly reduce their loss and stabilize around steps 50-100, consistent with the rapid plateau observed in the main text.

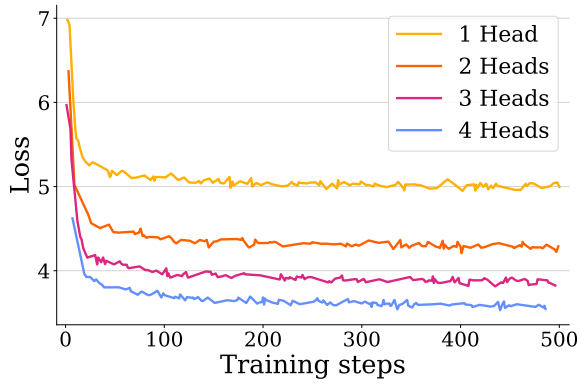

Figure 7: Training loss per-head configuration during projection alignment. The one-head model converges to a higher loss, while configurations with two or more heads reduce their loss rapidly and stabilize within the first 100 steps.

Table 11: Effect of importance reordering on the four nested runtime head configurations after 20,000 training steps. Values are accuracies in percent with standard error. ARC-Easy Norm is answer-length-normalized accuracy. TriviaQA is five-shot exact-match accuracy. The no-reordering variant is the default baseline from Table 10.

Variant	**H1**	**H2**	**H3**	**H4**
ARC-Easy (Raw)				
no reordering	39.0 ± 1.00	46.3 ± 1.02	51.9 ± 1.03	56.4 ± 1.02
Mag-CETT-CETT	38.4 ± 1.00	46.3 ± 1.02	52.2 ± 1.02	56.6 ± 1.02
Mag-Mag-Mag	37.6 ± 0.99	46.8 ± 1.02	52.9 ± 1.02	57.1 ± 1.02
VxC-VxC-VxC	36.9 ± 0.99	46.8 ± 1.02	51.9 ± 1.03	56.7 ± 1.02
Mag-no-Mag	38.3 ± 1.00	46.3 ± 1.02	53.2 ± 1.02	56.5 ± 1.02
Mag-VxC-Mag	37.7 ± 0.99	45.4 ± 1.02	52.1 ± 1.02	56.0 ± 1.02
ARC-Easy (Norm)				
no reordering	35.8 ± 0.98	40.0 ± 1.01	46.5 ± 1.02	50.8 ± 1.03
Mag-CETT-CETT	34.7 ± 0.98	41.6 ± 1.01	46.9 ± 1.02	50.5 ± 1.03
Mag-Mag-Mag	33.9 ± 0.97	42.0 ± 1.01	46.9 ± 1.02	51.1 ± 1.03
VxC-VxC-VxC	34.2 ± 0.97	41.4 ± 1.01	47.3 ± 1.02	51.1 ± 1.03
Mag-no-Mag	36.2 ± 0.99	41.1 ± 1.01	47.3 ± 1.02	49.4 ± 1.03
Mag-VxC-Mag	34.9 ± 0.98	40.7 ± 1.01	46.3 ± 1.02	51.3 ± 1.03
HellaSwag (Raw)				
no reordering	26.2 ± 0.44	26.9 ± 0.44	29.0 ± 0.45	31.1 ± 0.46
Mag-CETT-CETT	26.2 ± 0.44	26.8 ± 0.44	28.9 ± 0.45	30.9 ± 0.46
Mag-Mag-Mag	26.1 ± 0.44	27.0 ± 0.44	28.5 ± 0.45	31.2 ± 0.46
VxC-VxC-VxC	26.1 ± 0.44	26.9 ± 0.44	28.9 ± 0.45	31.1 ± 0.46
Mag-no-Mag	26.0 ± 0.44	26.8 ± 0.44	28.7 ± 0.45	31.2 ± 0.46
Mag-VxC-Mag	26.1 ± 0.44	26.8 ± 0.44	28.8 ± 0.45	31.1 ± 0.46
HellaSwag (Norm)				
no reordering	26.4 ± 0.44	28.3 ± 0.45	31.2 ± 0.46	35.6 ± 0.48
Mag-CETT-CETT	26.3 ± 0.44	28.0 ± 0.45	31.2 ± 0.46	36.0 ± 0.48
Mag-Mag-Mag	26.3 ± 0.44	28.7 ± 0.45	31.4 ± 0.46	35.9 ± 0.48
VxC-VxC-VxC	25.6 ± 0.44	28.1 ± 0.45	31.3 ± 0.46	35.7 ± 0.48
Mag-no-Mag	26.2 ± 0.44	28.0 ± 0.45	31.8 ± 0.46	35.9 ± 0.48
Mag-VxC-Mag	25.8 ± 0.44	28.4 ± 0.45	31.8 ± 0.46	35.6 ± 0.48
TriviaQA				
no reordering	0.15 ± 0.03	0.23 ± 0.04	3.69 ± 0.14	5.51 ± 0.17
Mag-CETT-CETT	0.21 ± 0.03	0.82 ± 0.07	4.84 ± 0.16	6.63 ± 0.19
Mag-Mag-Mag	0.14 ± 0.03	0.08 ± 0.02	3.08 ± 0.13	5.59 ± 0.17
VxC-VxC-VxC	0.00 ± 0.00	0.77 ± 0.07	3.73 ± 0.14	5.43 ± 0.17
Mag-no-Mag	0.11 ± 0.02	0.45 ± 0.05	4.26 ± 0.15	5.57 ± 0.17
Mag-VxC-Mag	0.03 ± 0.01	0.79 ± 0.07	2.99 ± 0.13	4.60 ± 0.16

Table 12: TriviaQA accuracy (exact match, %) across head configurations for text-only continued pre-training and subsequent multimodal or text-only fine-tuning variants.

Variant	**H1**	**H2**	**H3**	**H4**
text, 1M steps	0.13	0.66	2.35	3.55
multimodal, default	0.80	2.16	4.08	6.48
text fine-tune, 270M teacher	0.12	1.10	3.64	5.07
text fine-tune, 4B teacher	0.08	2.13	4.56	7.32