Revisiting Model Stitching in the Foundation Model Era

Supplementary Material

We provide details omitted in the paper.

- Sec. A: Experiment and dataset details
- Sec. B: Detailed results

A. Experiment and Dataset Details

A.1. Vision Foundation Models (VFMs)

We use the following VFMs in our experiments.

- **DINOv2-L** [34]: A ViT-L/14 encoder trained with self-supervised DINOv2 on a curated collection of ∼142M internet images, using a teacher–student distillation objective that encourages consistency across multi-crop image views without labels. In our experiments, we feed 336×336 inputs with patch size 14.
- **CLIP-L** [36]: A ViT-L/14 vision encoder from CLIP, trained jointly with a text encoder on ∼400M image–text pairs from the web using a contrastive objective that pulls matched image–caption pairs together and pushes mismatched pairs apart in a shared embedding space. We use the vision tower with 336×336 inputs and patch size 14.
- **SigLIP2-L** [43]: A ViT-L/16 multilingual vision–language encoder trained on a diverse mixture of web image–text data with a sigmoid-based image–text matching loss, augmented with captioning-style pretraining, self-distillation, masked prediction, and online data curation to improve dense features and localization. We use 384×384 inputs and patch size 16.
- **DINOv3-L** [37]: A ViT-L/16 self-supervised encoder from the DINOv3 family, trained at large scale on a multi-domain image collection with a distillation-style objective and Gram anchoring to stabilize dense feature maps, yielding strong frozen representations for both global and dense tasks. In our experiments, we use 384×384 inputs and patch size 16.

A.2. Fine-Grained Classification

A.2.1. Dataset Details

fMoW [7] The Functional Map of the World (fMoW) dataset is a commonly used and challenging dataset for VFM evaluation. It is a collection of high-quality remote-sensing satellite images collected worldwide, annotated with 62 land-use and functional categories (e.g., airfield, port, hospital). We use the fMoW-RGB variant, which provides pan-sharpened RGB crops and associated metadata. Since the original dataset is highly imbalanced, we construct a class-balanced subset with 230 training and 26 test images per class (14,260 train / 1,612 test) so that our study can focus on model stitching itself rather than confounding effects from label imbalance, which would complicate the interpretation of stitching behavior.

iNaturalist-Subset [44] The iNaturalist 2021 dataset family has become a standard testbed for assessing VFM performance on fine-grained, real-world biodiversity recognition. It is a large-scale and heavily imbalanced species classification benchmark that reflects naturally occurring long-tailed distributions. To obtain a controlled yet challenging setting for analyzing stitching, we sample 106 species from three visually similar Lepidoptera families (Sphingidae, Pieridae, Pyralidae) and rebalance the data with 200 training and 50 test images per class (21,200 train / 5,300 test). This balanced construction removes imbalance as an additional variable, enabling a cleaner analysis of how stitching choices impact performance.

FGVC-Aircraft [33] FGVC-Aircraft is widely adopted as a canonical fine-grained classification benchmark for VFMs. The version we use contains 102 aircraft models and approximately 10k images, with each model appearing in around 100 images. We follow the standard split with 6.6k training and 3.3k test images, which requires distinguishing subtle variations in shape, livery, and viewpoint.

A.2.2. Training Details

For all fine-grained classification experiments, we adopt linear probing: a single linear classifier is trained on pooled image features while all VFM parameters remain frozen.

For *layer feature matching*, we first extract intermediate features from both source and target VFMs and train the stitch MLP purely on these features; no additional VFM forward passes are required during this phase. For *final feature matching*, we extract features from the target model and train only the stitch layer.

To isolate the effect of stitching, we do not apply any data augmentation. We use the AdamW optimizer in all experiments, train for up to 100 epochs with early stopping (patience of 5 epochs), and tune the learning rate in $\{0.001, 0.005, 0.01\}$. For layer feature matching, we use a batch size of 256; all other configurations use a batch size of 128. Training is performed with automatic mixed precision using `bfloat16`.

A.2.3. Semantic Segmentation

A.2.4. Dataset Details

ADE20K [47, 48] ADE20K is a scene-centric semantic segmentation dataset with pixel-level annotations for

150 object and stuff categories across diverse indoor and outdoor environments. We adopt the canonical split with 20,210 training images and 3,000 held-out test images. The dense annotations cover scenes, objects, and parts, making ADE20K a challenging benchmark for evaluating dense prediction performance.

A.2.5. Training Details

For semantic segmentation, we train a linear layer to predict per-pixel class logits for each patch token. The linear layer produces a low-resolution logit map (e.g., 24×24 for a model with patch size 14 and 336×336 input), which is then bilinearly upsampled to the original image resolution to obtain the final segmentation map.

For layer feature matching and final feature matching, we reuse the optimization and hyperparameter settings described for classification (optimizer, learning rates, batch sizes, early stopping, and mixed precision). All remaining experimental details for segmentation follow [22].

A.3. VFM Stitch Tree for Multimodal LLM

A.3.1. Dataset Details

We use the LLaVA-1.5 training data prepared by TinyLLaVA.[1] LLaVA-1.5-Pretrain (PT) consists of 558k image–caption pairs [31], while LLaVA-1.5-SFT comprises 665k visual instruction-tuning conversations that combine academic-style VQA [14, 18, 26, 38] samples with instruction-tuning data from LLaVA-Instruct [30]. In our preliminary study, we evaluate on VQA-v2 [14] and MME [12].

A.3.2. Training Details

We jointly use LLaVA-1.5-Pretrain and LLaVA-1.5-SFT to train the stitch layer for one epoch with a learning rate of 0.001 and batch size 64. All remaining hyperparameters follow the TinyLLaVA recipe [49]. We adopt Qwen2.5-3B [2] as the LLM and employ an interleaved mixture-of-features (MoF) [42] strategy when combining visual tokens from CLIP and DINOv2.

B. Detailed and Additional Results

B.1. Self-Stitch

To rigorously test whether gains only stem from added capacity, we introduce a Self-Stitch baseline: inserting the identical stitch layer into the source-only and target-only models at the same stitch positions. Fig. 10 illustrates the difference between model stitching and the self-stitching baseline.

[1] We use the data preparation pipeline from TinyLLaVA: https://tinyllava-factory.readthedocs.io/en/latest/Prepare%20Datasets.html

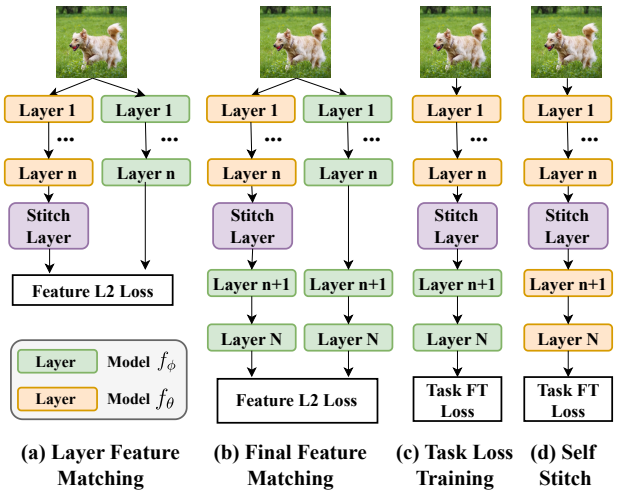

Figure 10. Model stitching training strategies: (a) Layer feature matching trains the stitch layer to match features between the source and target models at the stitch position. (b) Final feature matching trains the stitch layer so that the stitched model matches the target model's final features. (c) Task loss training optimizes the stitch layer with the downstream task objective. (d) Self-Stitch inserts the stitch layer into the source-only and target-only models at the same stitch position. Across all strategies, the stitch layer is the only trainable component; the source and target models are kept frozen.

B.2. Layer and Final Feature Matching

We present a detailed comparison between Layer and Final Feature Matching in Tab. 6. We observe that Final Feature Matching consistently outperforms Layer Feature Matching across all configurations. Notably, in the DINOv2 $\rightarrow$ SigLIP2 setting, the stitched model surpasses the performance of both constituent models. Our results further indicate that stitching a lower-performing source model to a higher-performing target is more effective than the reverse; DINOv2 $\rightarrow$ SigLIP2 consistently achieves higher accuracy than SigLIP2 $\rightarrow$ DINOv2, regardless of the matching strategy employed.

B.3. Stitching Between Different VFMs

To verify the generality of our findings, we extend our analysis beyond DINOv2 and SigLIP2 to include the widely adopted CLIP and the recently released DINOv3. Detailed comparisons between Stitched models, Self-Stitch baselines, and Linear Probing are provided in Tabs. 7 to 11. Consistent with our previous results, stitched models generally outperform their corresponding self-stitch baselines across most layers. The primary exception arises when CLIP is utilized as the source model; we discuss the potential causes for this phenomenon in the main paper.

	DINOv2			SigLIP2		
Linear Probing	46.7			53.5		
Layer	2	6	10	14	18	22
Layer Feature Matching						
DINOv2 → SigLIP2	49.4	48.2	51.0	48.5	45.3	48.2
SigLIP2 → DINOv2	29.9	32.4	26.9	42.4	43.7	47.2
Final Feature Matching						
DINOv2 → SigLIP2	51.4	53.1	56.1	51.0	55.7	54.7
SigLIP2 → DINOv2	46.4	47.1	43.3	47.3	48.8	50.8

Table 6. **Comparison of Layer and Final Feature Matching.** We report linear probing accuracy for different stitching strategies. Orange indicates performance surpassing the SigLIP2 baseline, while Blue indicates performance surpassing the DINOv2 baseline.

	DINOv2		CLIP	
Linear Probing	46.7		46.4	
Layer	6	14	18	22
DINOv2 → DINOv2	41.5	59.7	68.2	69.9
CLIP → CLIP	48.5	63.1	59.6	60.7
CLIP → DINOv2	53.1	65.8	63.3	64.0
DINOv2 → CLIP	**54.1**	**68.1**	**72.0**	**71.9**

Table 7. Stitched Model vs Self-Stitch vs Linear Probing: DINOv2 and CLIP.

	DINOv2		SigLIP2	
Linear Probing	46.7		53.5	
Layer	6	14	18	22
DINOv2 → DINOv2	41.5	59.7	68.2	69.9
SigLIP2 → SigLIP2	50.5	62.0	69.4	68.9
SigLIP2 → DINOv2	53.8	**69.6**	70.4	**72.2**
DINOv2 → SigLIP2	**55.8**	68.0	**72.0**	71.8

Table 8. Stitched Model vs Self-Stitch vs Linear Probing: DINOv2 and SigLIP2.

B.4. Prediction Analysis

To better understand how knowledge fusion affects prediction behavior, we compare each stitched model (DINOv2→SigLIP2 and SigLIP2→DINOv2) against the two self-stitched baselines (DINOv2→DINOv2 and SigLIP2→SigLIP2) on fMoW and iNaturalist. We discard trivial cases where all three models are correct or all three are wrong, and partition the remaining examples into the scenarios in Tab. 12:

1. **Preserve (Cols. 1–2).** At least one self-stitched model is correct and the stitched model is also correct, preserving the correct prediction.
2. **Rescue (Col. 3).** Both self-stitched models are wrong but the stitched model is correct.
3. **Interference (Cols. 4–6).** At least one self-stitched model is correct but the stitched model becomes wrong.

Across both datasets and stitch directions, the total count of *preserve* and *rescue* scenarios clearly exceeds that of *interference* scenarios, indicating that the stitched models benefit more from fusing complementary signals than they suffer from conflicts between the two backbones.

We further ask whose behavior the stitched model tends to follow when the two self-stitched models disagree. On fMoW, where the two self-stitched accuracies are similar, the stitched models are more likely to match the *source* model: for SigLIP2→DINOv2, the number of cases where the stitched prediction agrees with SigLIP2→SigLIP2 substantially exceeds the cases where it agrees with DINOv2→DINOv2, and an analogous trend

	SigLIP2		CLIP	
Linear Probing	53.5		46.4	
Layer	6	14	18	22
SigLIP2 → SigLIP2	50.5	62.0	69.4	68.9
CLIP → CLIP	48.5	63.1	59.6	60.7
CLIP → SigLIP2	48.3	68.9	65.4	62.9
SigLIP2 → CLIP	**59.8**	**70.7**	**73.2**	**71.9**

Table 9. Stitched Model vs Self-Stitch vs Linear Probing: SigLIP2 and CLIP.

	DINOv3		SigLIP2	
Linear Probing	50.0		53.5	
Layer	6	14	18	22
DINOv3 → DINOv3	44.7	63.9	66.9	69.0
SigLIP2 → SigLIP2	50.5	62.0	69.4	68.9
SigLIP2 → DINOv3	**58.6**	67.0	69.4	69.3
DINOv3 → SigLIP2	45.0	**69.4**	**70.5**	**72.4**

Table 10. Stitched Model vs Self-Stitch vs Linear Probing: DINOv3 and SigLIP2.

	DINOv2		DINOv3	
Linear Probing	46.7		50.0	
Layer	6	14	18	22
DINOv2 → DINOv2	41.5	59.7	68.2	69.9
DINOv3 → DINOv3	44.7	63.9	66.9	69.0
DINOv3 → DINOv2	43.8	**67.2**	67.9	**72.0**
DINOv2 → DINOv3	**57.8**	65.0	**69.2**	70.3

Table 11. Stitched Model vs Self-Stitch vs Linear Probing: DINOv2 and DINOv3.

fMoW

	Scenario						Acc.
DINOv2→DINOv2	W	R	W	R	R	W	69.9
SigLIP2→SigLIP2	R	W	W	R	W	R	68.9
SigLIP2→DINOv2	R	R	R	W	W	W	72.2
Count	116	78	47	40	86	33	

	Scenario						Acc.
DINOv2→DINOv2	W	R	W	R	R	W	69.9
SigLIP2→SigLIP2	R	W	W	R	W	R	68.9
DINOv2→SigLIP2	R	R	R	W	W	W	71.8
Count	77	116	54	52	48	72	

iNaturalist

	Scenario						Acc.
DINOv2→DINOv2	W	R	W	R	R	W	91.2
SigLIP2→SigLIP2	R	W	W	R	W	R	87.3
SigLIP2→DINOv2	R	R	R	W	W	W	91.9
Count	138	292	75	68	116	53	

	Scenario						Acc.
DINOv2→DINOv2	W	R	W	R	R	W	91.2
SigLIP2→SigLIP2	R	W	W	R	W	R	87.3
DINOv2→SigLIP2	R	R	R	W	W	W	92.8
Count	127	331	80	49	77	64	

Table 12. Analysis of Predictions: R and W denote Right and Wrong predictions for each model, with counts in the last row and accuracies in the last column for fMoW and iNaturalist. Stitched models fall into complementarity scenarios (stitched is correct while at least one self-stitched baseline is wrong) far more often than interference ones (stitched is wrong while at least one self-stitched baseline is correct). When the two self-stitched models disagree, the stitched model tends to follow the source model on fMoW but follows the stronger model DINOv2 on iNaturalist.

holds for DINOv2→SigLIP2. In contrast, on iNaturalist, where DINOv2→DINOv2 is the stronger self-stitched model, both stitch directions tend to align with the stronger model's predictions regardless of whether it is used as source or target.

Overall, these analyses support our interpretation that stitching primarily acts as a knowledge-fusion mechanism: it preserves or rescues correct predictions from at least one backbone much more often than it disrupts them, and when there is a strong/weak asymmetry, the stitched model gravitates toward the stronger expert's decisions.

B.5. VFM Stitch Tree (VST) for Multimodal LLM (MLLM)

B.5.1. Computation Comparison

Table 13 compares the computational cost of running all VFMs independently (Full) versus our VST method at various stitch positions. We illustrate the efficiency gains using Cambrian-1 with **4** VFMs [41]. Assuming a standard 24-layer ViT-L architecture [43], and noting that MLLMs typically extract features from the second last layer (layer 23) [31], we base our calculations on a 23-layer depth. While the "Full" setting incurs a 300% computational overhead compared to a single VFM, VST-14 significantly reduces this burden. By sharing the first 14 layers and maintaining specialized branches only from layer 15 onwards, VST-14 requires processing only $3 \times (23 - 14) = 27$ additional layers. This results in an overhead of just $27/23 \approx 117\%$, demonstrating a substantial efficiency improvement over the independent baseline.

Number of VFMs	Full	VST-6	VST-14	VST-22
2	100%	74%	39%	4%
3	200%	148%	78%	9%
4	300%	222%	117%	13%
5	400%	296%	156%	17%

Table 13. **Comparison of Additional Computation Cost.** We report the additional increase in computation compared to a single VFM. "Full" denotes running all VFMs independently. "VST-n" denotes a shared backbone up to the stitch position n, where layers 1 through n are shared and subsequent layers are executed independently. For example, VST-14 shares the first 14 layers and maintains specialized branches from layer 15 onwards.

B.6. VST as an Accuracy-Efficiency Knob

We introduce the concept of "Normalized Gain" to quantify the effectiveness of VST relative to running all VFMs independently. Table 14 provides the step-by-step derivation of these values for both VST-22 and VST-14.

We define the Normalized Gain as the ratio between the performance improvement achieved by VST (Δ_{VST}) and the maximum improvement achieved by running all VFMs independently (Δ_{max}):

$$\text{Normalized Gain (\%)} = \frac{\Delta_{\text{VST}}}{\Delta_{\text{max}}} = \frac{\text{Perf}_{\text{VST}} - \text{Perf}_{\text{CLIP}}}{\text{Perf}_{\text{Full}} - \text{Perf}_{\text{CLIP}}} \quad (2)$$

As illustrated in Tab. 14:

- The Orange Row represents the **Denominator** (Δ_{max}): the total performance gap between the single-model baseline and the computationally expensive naive full running

Model Configuration	VQAv2			MME		Efficiency Metrics	
	Yes/No	Number	Other	Percep.	Cogn.	Avg Gain %	Add. Cost
1. Define the Upper Bound (Denominator)							
(A) CLIP Baseline	91.75	58.74	69.00	1418.5	277.1	-	0%
(B) Full (CLIP + DINOv2)	92.72	61.64	70.30	1460.3	311.8	-	100%
(C) Max Gain ($\Delta_{\text{max}} = B - A$)	**0.97**	**2.90**	**1.30**	**41.8**	**34.7**	**(Denom.)**	**-**
2. VST-22: The Lightweight Knob (High Efficiency)							
(D) VST-22 (Ours)	92.12	59.21	69.15	1451.6	305.7	-	**4.3%**
(E) VST Gain ($\Delta_{\text{VST}} = D - A$)	**0.37**	**0.47**	**0.15**	**33.1**	**28.6**	**(Num.)**	**-**
Normalized Gain ($\% = E/C$)	38.1%	16.2%	11.5%	79.1%	82.5%	**45.5%**	**4.3%**
3. VST-14: The Balanced Knob (High Performance)							
(F) VST-14 (Ours)	92.54	60.69	69.88	1474.4	301.8	-	39.0%
(G) VST Gain ($\Delta_{\text{VST}} = F - A$)	**0.79**	**1.95**	**0.88**	**55.9**	**24.7**	**(Num.)**	**-**
Normalized Gain ($\% = G/C$)	81.4%	67.2%	67.7%	133.7%	71.1%	**84.2%**	39.0%

Table 14. **Detailed Calculation of Normalized Gain.** Comparison of the "Lightweight" VST-22 and the "Balanced" VST-14. The Orange Row represents the gain (Δ_{max}) achieved by the computationally expensive Naive ensemble (100% Cost). The Pink Rows represent the actual gain (Δ_{VST}) achieved by our method. The **Normalized Gain** is calculated as $\Delta_{\text{VST}}/\Delta_{\text{max}}$. Note that VST-22 achieves nearly half the total gain (45.5%) with negligible cost (4.3%).

(100% additional cost).

- The Pink Rows represent the **Numerator** (Δ_{VST}): the actual gain realized by our VST method.

The results highlight the versatility of VST as a compute-aware knob. **VST-22** serves as an ultra-lightweight option, recovering **45.5%** of the total possible gain while incurring a negligible **4.3%** increase in backbone cost. Conversely, **VST-14** acts as a balanced high-performance option, recovering **84.2%** of the total gain for only **39%** of the additional cost. This granular control allows practitioners to maximize model utility within specific computational budgets.

Consequently, we view VST as a vital *accuracy–efficiency knob* for architecture design. Without VST, practitioners face a rigid binary choice: either deploy an entire additional VFM (yielding higher performance but incurring **100%** extra backbone cost) or deploy none (yielding no gain). VST transforms this discrete step function into a continuous spectrum. By serving as a compute-aware knob, our method effectively interpolates between these extremes, enabling controllable performance–efficiency trade-offs that can be dynamically tuned to meet strict real-world deployment constraints.