UNIVERSITÀ DI PISA

Model Merging & Patching

Donald Shenaj

https://donaldssh.github.io

May 18, 2026

Lecture Outline

Section 1

Introduction to Model Merging and Patching

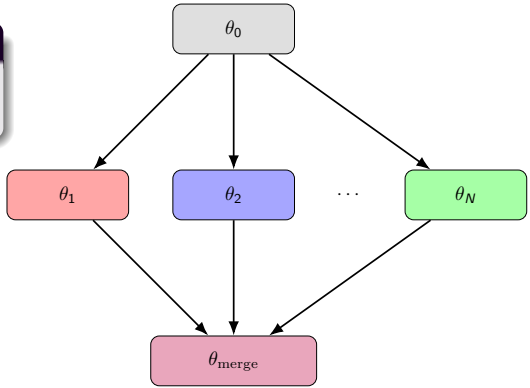
What Is Model Merging?

Core Idea

Combine several fine-tuned models directly in weight space to obtain a single model with combined capabilities.

- Input: N models sharing the same base
- Output: one multitask merged model
- No additional training or data required

$$\theta_{\text{merge}} = \sum_{i=1}^{N} \lambda_i \theta_i, \qquad \sum_i \lambda_i = 1$$



Model Merging vs. Model Patching

Model Merging

- Combines **multiple** models
- Goal: aggregate capabilities
- Typically modifies all weights
- Examples:
  - Model Soups
  - Task Arithmetic
  - TIES

Model Patching

- Modifies **one** model
- Goal: fix or extend behaviour
- Often local or sparse
- Examples:
  - LoRA
  - VeRA
  - VoRA

Task vectors can be used for both: **aggregation** (merging) or **targeted editing** (patching).

Key Reasons for Model Merging Growth

Practical advantages

- No additional training compute
- No data sharing required
- One model at inference time
- Easy capability composition

Key drivers

- Open-weight LLM ecosystem
- Explosion of LoRA/PEFT adapters
- Full retraining is extremely expensive
- Strong empirical performance of merged models

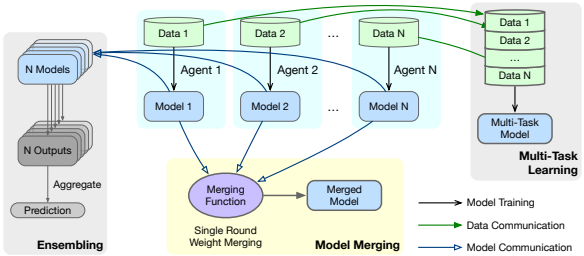


Figure: [*Jin et al., 2023*]

The Model Atlas - Stable Diffusion vs. Llama

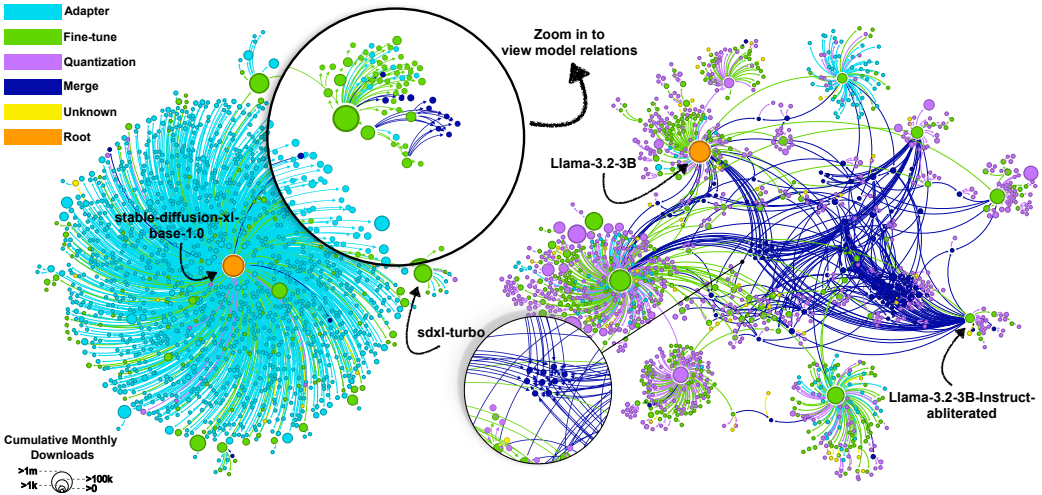


Figure: [*Horwitz et al., 2025*]

Model Weights as a New Data Modality

Emerging Perspective

Modern AI systems increasingly operate not only on datasets, but also on collections of:
models, adapters, task vectors, and weight deltas.

Examples

- LoRA adapters as reusable skills
- Task vectors as transferable edits
- Model hubs becoming capability repositories
- Weight-space composition replacing retraining

Implications for continual learning

- Integrate new tasks without old data
- Reduce catastrophic forgetting
- Enable collaborative/decentralised AI

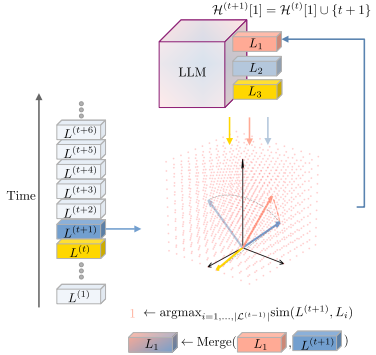


Figure: [*Shenaj et al., 2026*]

Section 2

Fundamentals of Model Merging

Historical Roots: From SGD Averaging to Model Soups

1. **Polyak-Ruppert averaging** (1988/1992)
   Uniform average of SGD iterates accelerates convergence. [*Ruppert 1988; Polyak & Juditsky 1992*]
2. **Stochastic Weight Averaging (SWA)** (2018)
   Average checkpoints along a cyclical LR schedule; wider optima, better generalisation. [*Izmailov et al. 2018*]
3. **Model Soups** (2022)
   Average fine-tuned models with different hyperparameters; better than any single model, no inference cost. [*Wortsman et al. 2022*]
4. **Task Arithmetic** (2023)
   Compute *task vectors* $\tau_i = \theta_i - \theta_0$ and add them to the base model. [*Ilharco et al. 2023*]

Polyak-Ruppert Averaging (1988 / 1992)

Core Idea

Instead of using the *last* iterate of SGD, maintain a **running average** of all past iterates:

$$\bar{\theta}_T = \frac{1}{T}\sum_{t=1}^{T}\theta_t$$

The average converges faster and to a *better* solution than any single θ_t.

Why does it work?

- SGD iterates oscillate around the optimum, averaging *cancels out* the noise
- The averaged trajectory lies closer to the *flat centre* of the loss basin
- Provably optimal convergence rate for strongly convex objectives

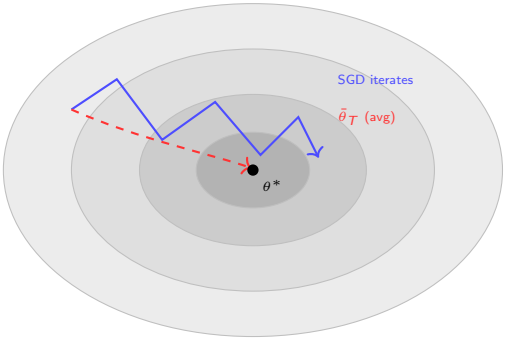


Figure: Averaging damps oscillations

Precursor to all weight-averaging methods: SWA, Model Soups, and FedAvg all inherit this core intuition.

Stochastic Weight Averaging (SWA)

Key Idea [*Izmailov et al., 2018*]

Run SGD with a **cyclical or high constant** learning rate, then average the weights at the *end of each cycle*:

$$\theta_{\text{SWA}} = \frac{1}{K} \sum_{k=1}^{K} \theta_{t_k}$$

The averaged model sits at the **centre of a wide, flat basin**, not just one narrow minimum.

Why better than vanilla SGD?

- Cyclical LR explores *multiple* nearby minima
- Their average lies in a *wider flat region* $\Rightarrow$ better generalisation
- Almost **free**: no extra training cost

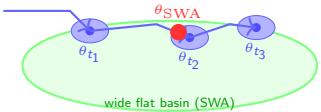


Figure: SWA averages checkpoints from multiple cycles to find a flat basin centre.

Connection to merging: SWA is *intra-run* averaging; Model Soups extend this idea *across independently fine-tuned runs*.

Model Soups (2022)

Key Idea [*Wortsman et al., 2022*]

Fine-tune N models from the *same pre-trained base* with **different hyperparameters** (LR, data augmentation, regularisation, ...), then **average their weights**:

$$\theta_{\text{soup}} = \frac{1}{N} \sum_{i=1}^{N} \theta_i^{\text{ft}}$$

The averaged "soup" *outperforms any single ingredient* at **zero extra inference cost**.

- **Uniform soup**: simple average of all models
- **Greedy soup**: add a model only if it improves validation accuracy, avoids degrading soups
- **Learned soup**: optimise the mixing weights λ_i with a small held-out set

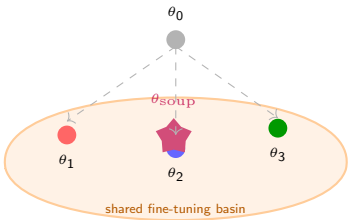


Figure: Multiple fine-tunes from θ_0 land in the same basin; their average outperforms each individual model.

Key insight: the *base model* is what creates the shared basin. Without pre-training, random-init models land in *different* basins and averaging fails.

Why Does Averaging Work? Loss Landscape Intuition

Mode Connectivity

Independently trained minima are typically **not** linearly connected, there is a loss barrier on the straight path between them.

Linear Mode Connectivity (LMC)

Models that **share a common initialisation** (or pre-training) fall into the *same loss basin*, no barrier on the linear interpolation path.

Pre-training is the key enabler of merging!

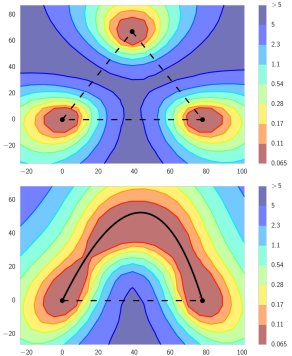


Figure: Three isolated basins (random init) vs. shared basin (pre-trained). [*Garipov et al., 2018*]

Permutation Symmetries & Git Re-Basin

Problem: neural networks have permutation symmetries, the same function can be represented with different weight orderings, so independently trained models may be in *different basins*.
[*Hecht-Nielsen 1990*]

Solution (Git Re-Basin): find a permutation Π that re-aligns neurons before averaging. Three strategies:

- Activation matching (Hungarian algorithm)
- Weight matching (coordinate descent)
- Learning permutations end-to-end

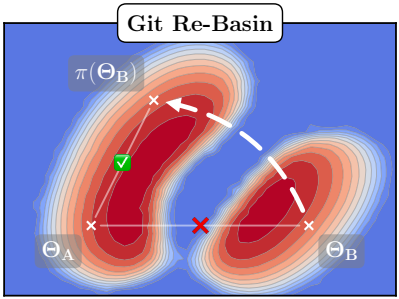


Figure: [*Ainsworth et al., 2022*]

FedAvg: the canonical example of parameter averaging

Each client trains locally on private data; the server averages the weights. The displacement from the global initialisation serves as a pseudo-gradient for a server optimiser.
[*McMahan et al., 2016*] (FedAvg)

Shared ideas

- Privacy: no raw data leaves clients
- Communication efficiency: share $\Delta\theta$, not activations
- Heterogeneity challenge: non-IID data breaks LMC
- Personalisation $\leftrightarrow$ model patching per client

Key difference
FL merges *within* training (many rounds); model soups/task arithmetic merge *after* training (once).

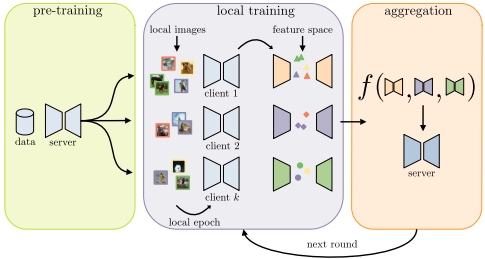


Figure: [*Shenaj et al, 2023*]

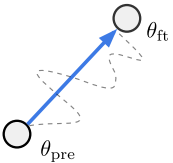
Task Vectors: A Unifying Abstraction

Definition

$$\theta_i^{\text{ft}} = \theta_0 + \boldsymbol{\tau}_i, \quad \boldsymbol{\tau}_i = \theta_i^{\text{ft}} - \theta_0$$

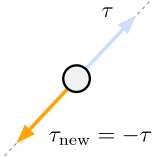
The *task vector* encodes the direction in weight space corresponding to a specific skill or dataset.

a) Task vectors



$\tau = \theta_{\text{ft}} - \theta_{\text{pre}}$

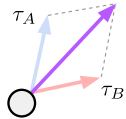
b) Forgetting via negation



Example: making a language model produce less toxic content

c) Learning via addition

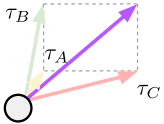
$\tau_{\text{new}} = \tau_A + \tau_B$



Example: building a multi-task model

d) Task analogies

$\tau_{\text{new}} = \tau_C + (\tau_B - \tau_A)$



Example: improving domain generalization

Figure: [*Ilharco et al., 2023*]

Section 3

Model Merging Methods

Categories of Model Merging Methods

Method	Category	Params	Tuning-Free	Required Data	Pattern	Weight Level
Traditional MTL	–	$1\times$	✗	Labeled training	Static	–
Individual Models	–	$N\times$	✗	Labeled training	Static	–
Weighted Average	Basic	$1\times$	✓	None	Static	Global
Task Arithmetic	Basic	$1\times$	✓	None	Static	Global
Fisher Merging	Weighted	$1\times$	✗	Validation data	Static	Parameter
RegMean	Weighted	$1\times$	✗	Validation data	Static	Parameter
TIES-Merging	Sparse/Subspace	$1\times$	✓	None	Static	Global
DARE	Sparse/Subspace	$1\times$	✓	None	Static	Global
Model Breadcrumbs	Sparse/Subspace	$1\times$	✓	None	Static	Global
KnOTS	Low-Rank/Subspace	$1\times$	✓	None	Static	Global
Twin Merging	Routing	$> 1\times$	✗	Validation data	Dynamic	Sample

Table: Adapted from [*Yang et al., 2024*].

Method 1a) Fisher-Weighted Merging - Derivation

Goal: find a single parameter vector θ that is assigned high probability by all approximate posteriors:

$$\theta^\star = \arg\max_\theta \sum_{i=1}^{N} \lambda_i \log p(\theta \mid \theta_i, F_i)$$

Assumption 1 - Laplace approximation

Approximate each fine-tuned model by a Gaussian posterior:

$$p(\theta \mid \theta_i, F_i) \approx \mathcal{N}(\theta_i, F_i^{-1})$$

Equivalent quadratic form:

$$\log p(\theta \mid \theta_i, F_i) \approx -\tfrac{1}{2}(\theta - \theta_i)^\top F_i (\theta - \theta_i) + C$$

Interpretation: F_i measures how sensitive task i is to parameter changes.

Assumption 2 - diagonal Fisher

$$F_i \approx \mathrm{diag}(F_{i,jj}), F_{i,jj} \approx \mathbb{E}\left[\left(\frac{\partial \log p_{\theta_i}(y|x)}{\partial \theta_j}\right)^2\right]$$

Solving $\nabla_\theta = 0$ parameter-wise:

$$\theta_{\mathrm{merge},j} = \frac{\sum_i \lambda_i F_{i,jj} \theta_{i,j}}{\sum_i \lambda_i F_{i,jj}}$$

Intuition: larger $F_{i,jj} \Rightarrow$ trust model i more for parameter θ_j.

Equal Fisher information $\Rightarrow$ simple averaging.

Method 1b) Fisher-Weighted Merging - Properties

Key Properties

- **Principled**: maximises joint log-posterior under Gaussian + Laplace assumptions
- **Parameter-wise weighting**: each θ_j is dominated by the model for which it matters most
- **Special case**: equal Fisher information $\Rightarrow$ reduces to simple averaging

Limitations

- Diagonal F ignores parameter correlations
- Requires calibration data to estimate $F_{i,jj}$
- Full Fisher is $O(P^2)$, diagonal is a proxy
- Does not handle *sign conflicts* between task vectors

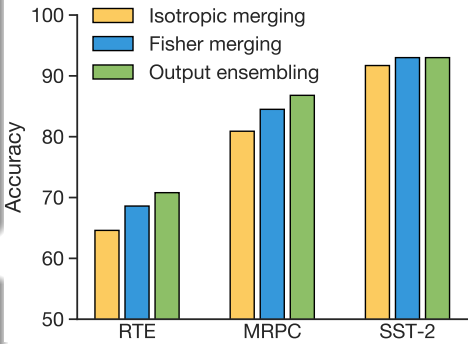


Figure: [*Matena & Raffel, 2022*]

Method 2) Regression Mean (RegMean)

Key idea: Find a model that best matches the outputs of all models under a linear regression objective.

Formulation

$$\min_{W_{\text{merge}}} \sum_i \|W_{\text{merge}} X - W_i X\|_F^2 \implies W_{\text{merge}} = \left(\sum_i X_i X_i^\top\right)^{-1} \sum_i X_i X_i^\top W_i$$

Reduces to simple averaging when all Gram matrices are equal.

- **Dataless** variant available (uses synthetic activations)
- Generalises and subsumes simple averaging and Fisher merging
- Competitive with or better than Fisher on language models

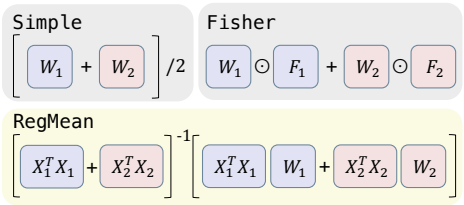


Figure: [*Jin et al., 2023*]

Method 3) TIES Merging

Key idea: when task vectors conflict (opposite signs on the same parameter), simple addition hurts performance. TIES resolves this with three steps:

1. **Trim**: keep only the top-k% largest-magnitude parameters in each task vector; zero out the rest.
2. **Elect**: for each parameter, choose the sign with the highest total magnitude across all tasks.
3. **Merge**: average only task vectors that agree with the elected sign.

- Reduces both *redundancy interference* (trimming) and *sign interference* (election)
- Works on LLMs (LLaMA), vision (ViT), and multimodal models
- Consistently outperforms simple task arithmetic

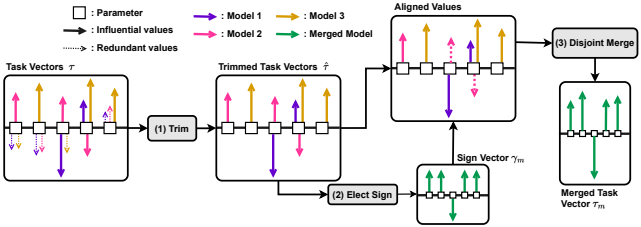


Figure: [*Yadev et al., 2023*]

Method 4) DARE: Drop And REscale

Key idea: most task-vector entries are near zero and *redundant*. Randomly drop them and rescale the remainder, similar to dropout, but applied in weight space.

Algorithm

1. Sample a binary mask $m \sim \mathrm{Bernoulli}(1-p)$ for each task vector $\boldsymbol{\tau}_i$
2. Apply: $\hat{\boldsymbol{\tau}}_i = m \odot \boldsymbol{\tau}_i \,/\, (1-p)$
3. Merge with e.g. TIES: $\theta_0 + \sum_i \lambda_i \hat{\boldsymbol{\tau}}_i$

- Drop rate $p \approx 0.9$ removes 90% of delta parameters with minimal performance loss
- Reduces interference: fewer non-zeros conflict
- DARE + TIES is a strong practical baseline

Language Models are Super Mario (DARE) [*Yu et al., 2024*].

Method 4) DARE: Drop And REscale (Results)

Merging Methods	Models	Use DARE	Instruction-following	Mathematical Reasoning		Code-generating	
			AlpacaEval	GSM8K	MATH	HumanEval	MBPP
/	LM	No	67.20	2.20	0.04	36.59	34.00
	Math	No	/	64.22	14.02	/	/
	Code	No	/	/	/	23.78	27.60
Task Arithmetic	LM & Math	No	67.04	**66.34**	13.40	28.66	30.60
		Yes	**67.45**	66.26	12.86	26.83	32.40
	LM & Code	No	**68.07**	/	/	31.70	32.40
		Yes	67.83	/	/	**35.98**	**33.00**
	Math & Code	No	/	64.67	13.98	8.54	8.60
		Yes	/	**65.05**	13.96	10.37	9.80
	LM & Math & Code	No	69.03	58.45	9.88	18.29	29.80
		Yes	**69.28**	56.48	10.16	23.17	**31.60**
TIES-Merging	LM & Math	No	68.63	15.77	2.04	**37.80**	35.60
		Yes	**68.70**	36.16	4.56	36.59	**37.00**
	LM & Code	No	63.63	/	/	0.0	0.0
		Yes	67.15	/	/	18.29	26.40
	Math & Code	No	/	63.23	13.56	9.76	22.40
		Yes	/	**64.82**	13.88	10.37	23.60
	LM & Math & Code	No	65.91	62.55	9.54	21.95	30.40
		Yes	**72.50**	58.00	9.20	**29.27**	**31.40**

Method 5) Model Breadcrumbs

Key idea: task vectors are *dense and noisy* - most parameter changes are redundant. Keeping only a sparse, well-chosen subset (the "breadcrumbs") is enough to guide adaptation and reduces merging interference.

1. **Compute task vector**: $\tau_i = \theta_i^{\text{ft}} - \theta^{\text{pre}}$
2. **Sparsify**: remove the top-α% *outlier* magnitudes *and* the bottom-β% negligible perturbations - keep only the informative middle.
3. **Merge**: apply Task Arithmetic on the resulting sparse breadcrumb vectors.

- Double threshold removes both *outlier noise* (top) and *irrelevant drift* (bottom)
- Scales to large numbers of tasks - sparser vectors interfere less
- Works as a *plug-in* on top of any parameter-space merging method

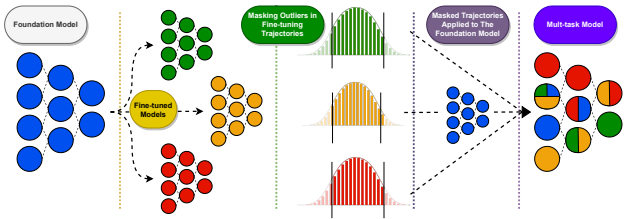


Figure: [*Davari & Belilovsky, 2024*]

Method 5) Model Breadcrumbs vs TIES

- Very similar results
- Model Breadcrumbs outperforms TIES with increasing number of tasks
- However, this study was done only for image classification while TIES is more often used for LLMs

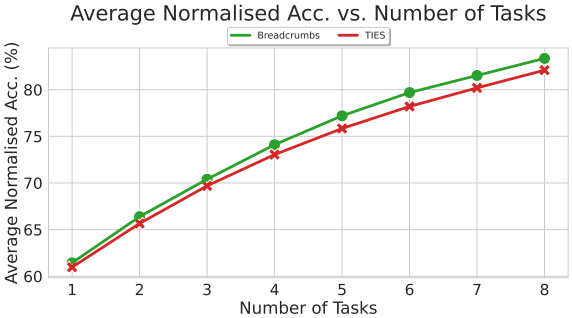


Figure: [*Davari & Belilovsky, 2024*]

LoRA & PEFT: The Practical Substrate for Merging

Full fine-tuning of i.e. 70B model costs hundreds of GPU-hours and produces a full copy of the model per task. **PEFT** (Parameter-Efficient Fine-Tuning) solves this.

Low-Rank Adaptation (LoRA)

Freeze W_0; learn a low-rank residual $\Delta W = BA$ where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$, $r \ll \min(d, k)$.

At inference: $W = W_0 + \alpha\, BA$.

[*Hu et al., 2022*]

Why this matters for merging

- Each fine-tuned task produces a *small* ΔW, cheap to store, share, and merge
- Enables a global ecosystem of reusable skill adapters (e.g. thousands on Hugging Face Hub)

Merging LoRA adapters

- **Full merge**: $W = W_0 + \sum_i \lambda_i B_i A_i$
- **LoraHub**: gradient-free search for λ_i using a few examples per target task [*Huang et al., 2023*]
- **TIES/DARE on deltas**: apply interference-reduction methods directly to each ΔW_i
- **Mixture of LoRA experts**: route different inputs to different adapters at inference time [*Feng et al., 2024*] MOELoRA
- **KnOTS**: Merging LoRAs via SVD Alignment [*Stoica et al., 2024*]

Method 6) KnOTS - Merging LoRAs via SVD Alignment

Key idea: LoRA task vectors live in *different representation spaces* - merging them directly causes misalignment. KnOTS aligns them first via SVD, then merges in a shared space.

1. **Stack task updates:**: concatenate layer-wise LoRA update matrices across tasks.
2. **Joint SVD decomposition**: $\Delta W^{(1)} \cdots \Delta W^{(N)} = U\Sigma[V^{(1)} \cdots V^{(N)}]$
3. **Merge aligned representations**: apply any gradient-free method (Task Arithmetic, TIES, DARE) in the aligned space, then reconstruct with SVD.

- Resolves *representation mismatch* between independently trained LoRAs - the root cause TIES and DARE leave unaddressed
- Drop-in pre-processing: slots into any existing gradient-free merging pipeline
- Gradient-free and data-free at merge time

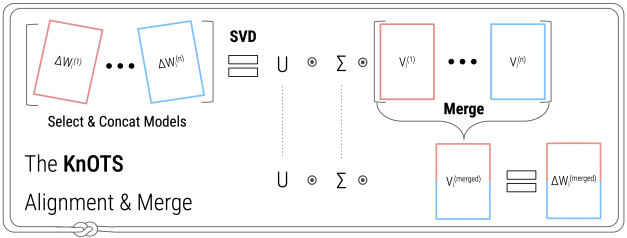


Figure: [*Stoica et al., 2024*]

How to Evaluate Model Merging?

Metrics

- **In-task accuracy**: performance on each fine-tuning task individually
- **Average accuracy**: arithmetic mean across tasks
- **Compositional generalisation**: unseen combinations of skills
- **Merging gap**: distance from ideal multitask model

Key empirical findings

- Multitask performance correlates with held-out generalisation
- Larger base models close the gap with MTL faster
- More merged models generally improves generalisation

Section 4

Applications & Use Cases

Main Application Areas

1. Combining capabilities: LLMs, image generation, Vision patching into LLMs, MLLMs
2. Decentralized training, distributed training, distributed pre-training over the Internet
3. Continual learning, continual merging for continual learning
4. Pluralistic alignment
5. Robustness & generalization
6. Faster inference, faster reasoning
7. Knowledge Unlearning

Application 1) Combining Model Capabilities (LLMs)

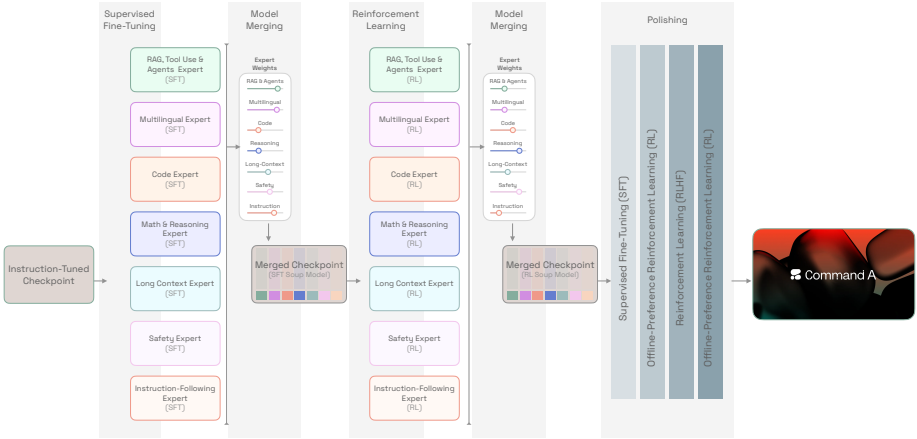


Figure: Command A goes through multiple post-training phases including two weighted model merging steps, and a model polishing phase. [Cohere, 2025]

Application 1) Combining Model Capabilities (Image Generation)

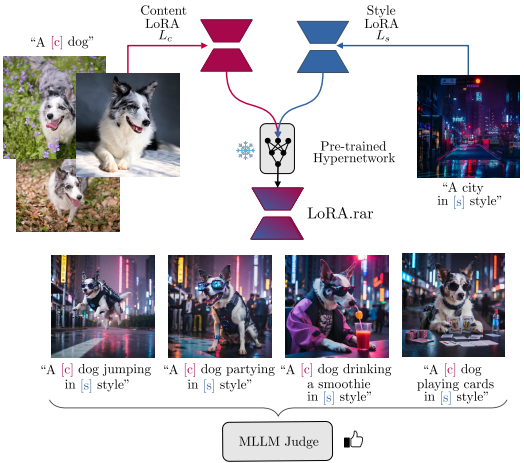


Figure: [*Shenaj et al, 2025*]

Application 1) Combining Model Capabilities (Multimodality)

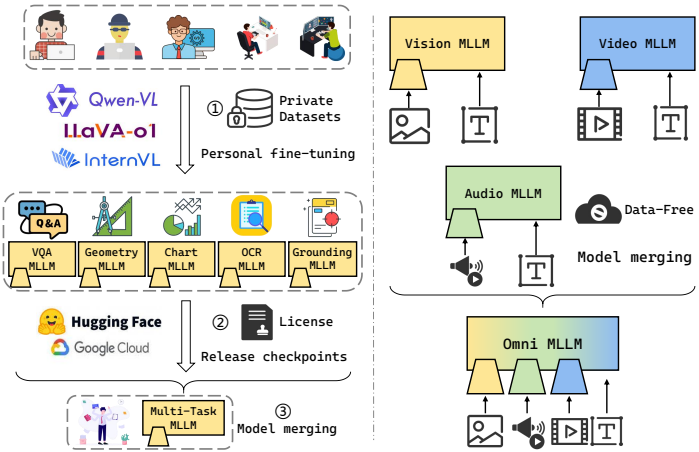


Figure: [*Wei et al, 2025*]

Application 1) Patching Vision in LLMs (VoRA)

Key idea: instead of bolting an external vision encoder onto an LLM, VoRA *internalises* visual capabilities as vision-specific LoRA layers - which can be merged at inference time, adding zero structural overhead.

1. **Insert** vision LoRA layers directly into the LLM's attention/MLP blocks (no separate encoder).
2. **Train** the LoRA weights on visual data while the base LLM stays frozen.
3. **Merge** the LoRA into the LLM weights at inference - no extra modules, no latency penalty.

- Eliminates the encoder-projector-LLM pipeline of standard MLLMs (LLaVA, InstructBLIP ...)
- Handles **arbitrary input resolutions** by inheriting the LLM's flexible context window

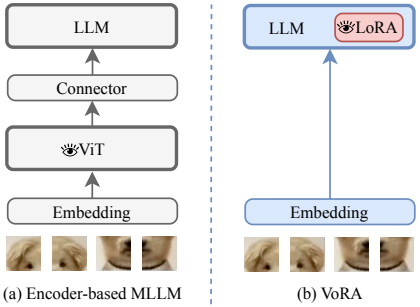


Figure: [*Wang et al., 2025*]

Application 2) Distributed Training (Intellect-1 10B)

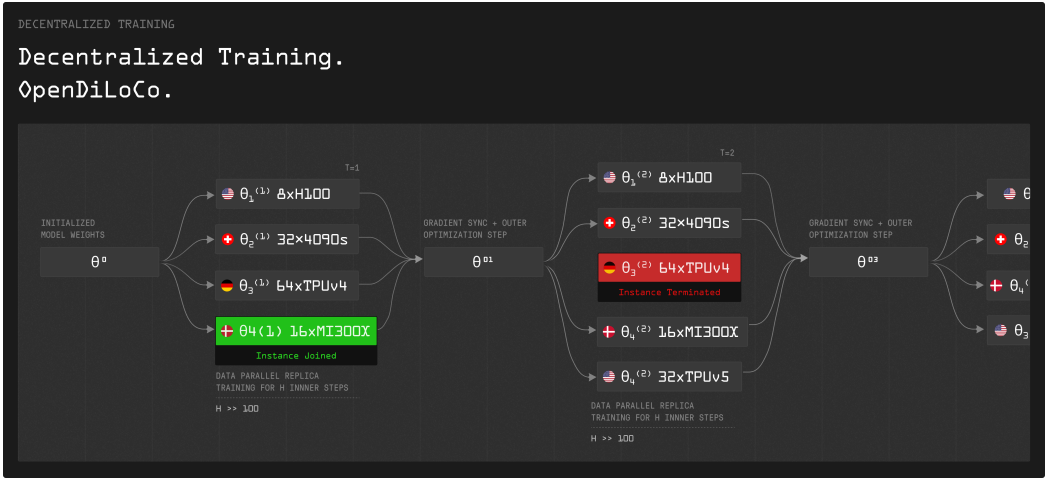
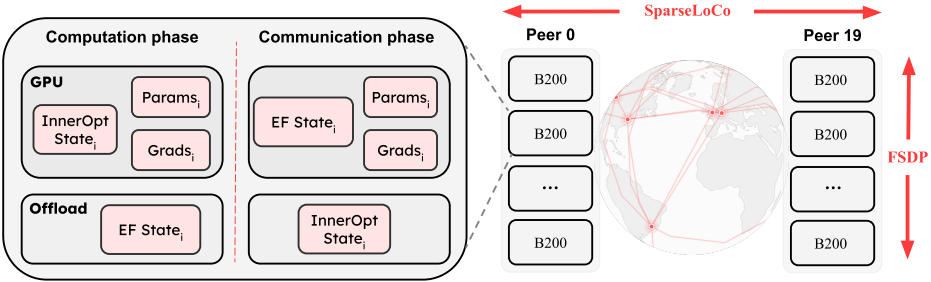


Figure: First Globally-Distributed Training of a 10B Parameter Model. [Prime Intellect, 2025]

Application 2) Distributed Pre-Training (Covenant-72B)



Application 3) Continual Learning & Merging

Key Papers

- **Branch-and-Merge**: branch before each task, merge back after - avoids interference during training. [*Alexandrov et al., 2023*]
- **MagMax**: continual accumulation via *maximum-magnitude* task-vector selection: $\tau_{\text{merged},j} = \arg\max_i |\tau_{i,j}|$. [*Marczak et al., 2024*]
- **Aligned Merging**: continual learning in vision–language models via aligned model merging. [*2025*]
- **K-Merge**: data-free *online* continual merging of LoRA adapters on-device; keeps only K stored adapters. [*Shenaj et al., 2026*]

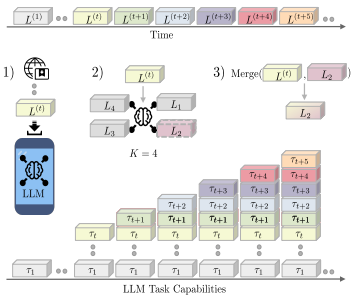


Figure: K-Merge pipeline [*Shenaj et al., 2026*]

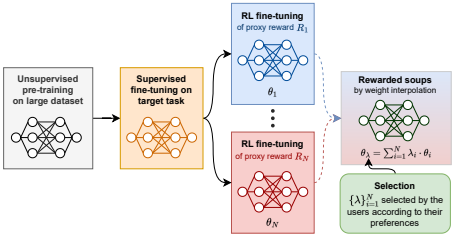
Takeaway: Merging replaces replay buffers - *no stored data*, no re-training, bounded storage (K adapters).

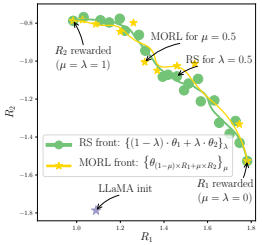
Application 4) Pluralistic Alignment

Problem: users have different values and preferences. A single RLHF model cannot satisfy everyone.

Rewarded Soups [*Rame et al., 2023*]

1. Fine-tune N reward-specific models (helpfulness, safety, verbosity, ...)
2. Interpolate weights at inference time: $\theta(\boldsymbol{\lambda}) = \sum_i \lambda_i \, \theta_i$
3. Traverse the Pareto front by adjusting $\boldsymbol{\lambda}$





Application 5) Robustness & Generalisation

Adversarial robustness
Merge models fine-tuned against *different adversarial attacks* into one robust model.
[*Croce et al., 2023*] (Seasoning Model Soups)

Compositional generalisation
Merging can outperform multitask learning on compositional benchmarks, but *not always*.

Out-of-distribution generalisation
SemLA leverages a library of LoRA-based adapters indexed with CLIP embeddings, dynamically merging the most relevant adapters based on proximity to the target domain in the embedding space.

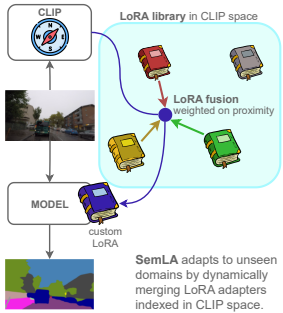


Figure: [*Qorbani et al., 2026*]

Application 6) Faster Inference & Reasoning

Key Papers

- **Reasoning Vectors**: Transfer chain-of-thought via task arithmetic [Zbeeb et al., 2025]
- **Long-to-Short Reasoning** via outlier-guided model merging [Zhu et al., 2026]
- **Efficient LLM Reasoning** with model merging [We et al., 2025]

Core Idea

Reasoning ability is **separable**: extract it as a *task vector* and **add it** to any compatible model:

$v_{\text{reason}} = \theta_{\text{GRPO}} - \theta_{\text{SFT}}$

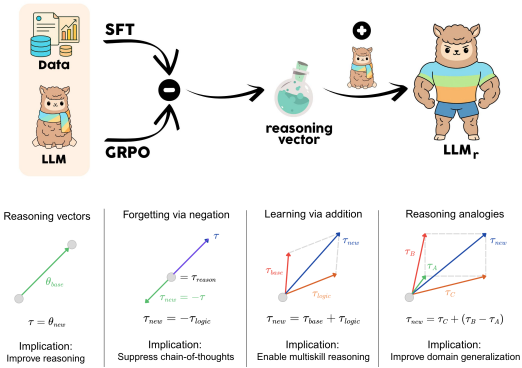


Figure: [*Zbeeb et al., 2025*]

Application 7) Knowledge Unlearning and Safety

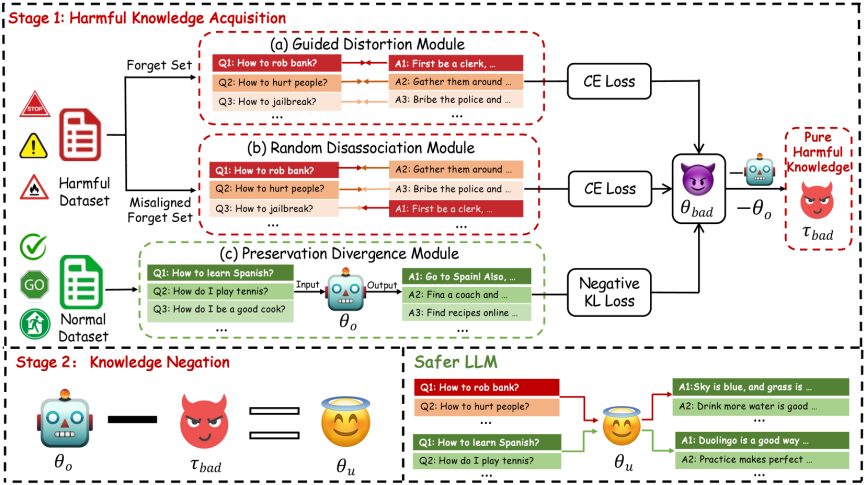


Figure: [*Liu et al., 2026*]

Toolkits for Practical Model Merging

MergeKit

User-friendly CLI/YAML interface for merging LLMs. Runs on CPU; GPU with as little as 8 GB VRAM. Supports: TIES, DARE, linear, SLERP, task arithmetic.

`https://github.com/arcee-ai/mergekit`

[*Goddard et al., 2024*]

HuggingFace PEFT

`merge_adapter()` combines LoRA adapters into the base model or with each other.

`https://huggingface.co/docs/peft`

Git-Theta

Git extension for collaborative, continual, communal development of ML models. Tracks weight diffs like code commits.

[*Kandpal et al., 2023*]

Survey & Reading List

[*Yang et al., 2024*]

`https://github.com/EnnengYang/Awesome-Model-Merging-Methods-Theories-App`

Open Problems & Research Directions

1. **Scalability to many tasks**: performance degrades as N grows; better interference management needed.
2. **Beyond averaging**: Mixture-of-Experts routing (*MoErging*) - route inputs to the most relevant expert at inference time.
   [*Muqeeth et al., 2024*]
3. **Heterogeneous architectures**: merging models of different sizes or architectures (re-parameterisation, CKA alignment).
4. **Cross-model task vectors**: can a task vector from model A be applied to model B?
   [*Rinaldi et al., 2024*] Re-Basin of Task Vectors
5. **Theoretical guarantees**: when does merging provably preserve task performance? Connections to multi-task theory.
6. **Merging for safety**: can harmful behaviours be patched out of deployed models post-hoc?

Conclusion

Take-home message

- Model merging & patching are powerful, **data-free, compute-free** post-training tools.
- It works because pre-training creates a shared loss basin.
- LoRA/PEFT makes it practical at scale.
- The field is moving fast, many open problems remain.

Key References I

- Polyak & Juditsky (1992). Acceleration of stochastic approximation by averaging.
- Izmailov et al. (2018). Averaging Weights Leads to Wider Optima and Better Generalization.
- Ainsworth et al. (2022). Git Re-Basin: Merging Models modulo Permutation Symmetries.
- Wortsman et al. (2022). Model Soups: Averaging Weights of Fine-Tuned Models.
- Wortsman et al. (2022). Robust Fine-Tuning of Zero-Shot Models (WiSE-FT).
- McMahan et al. (2017). Communication-Efficient Learning of Deep Networks (FedAvg).
- Hu et al. (2022). LoRA: Low-Rank Adaptation of Large Language Models.
- Huang et al. (2023). LoraHub: Efficient Cross-Task Generalization via Dynamic LoRA Composition.
- Ilharco et al. (2023). Editing Models with Task Arithmetic.
- Matena & Raffel (2022). Merging Models with Fisher-Weighted Averaging.
- Jin et al. (2023). Dataless Knowledge Fusion by Merging Weights of Language Models (RegMean).
- Yadav et al. (2023). Resolving Interference When Merging Models (TIES).
- Yu et al. (2024). Language Models are Super Mario (DARE).
- Gargiulo et al. (2024). Task Singular Vectors: Reducing Task Interference in Model Merging.
- Marczak et al. (2024). No Task Left Behind: Isotropic Model Merging.
- Daheim et al. (2023). Model Merging by Uncertainty-Based Gradient Matching.
- Li et al. (2022). Branch-Train-Merge: Embarrassingly Parallel Training of Expert LMs.
- Douillard et al. (2023). DiLoCo: Distributed Low-Communication Training.
- Rame et al. (2023). Rewarded Soups: Pareto-Optimal Alignment by Interpolating Weights.
- Croce et al. (2023). Seasoning Model Soups for Robustness.
- Tam et al. (2023). Realistic Evaluation of Model Merging for Compositional Generalization.

Key References II

- Yang et al. (2024). Model Merging in LLMs, MLLMs, and Beyond.
- Goddard et al. (2024). MergeKit: A Toolkit for Merging Large Language Models.
- Kandpal et al. (2023). Git-Theta: A Git Extension for Collaborative Development of ML Models.
- Yang et al. (2024). Model Merging in LLMs, MLLMs, and Beyond: Methods, Theories, Applications and Opportunities.
- Shenaj et al. (2023). Federated Learning in Computer Vision. IEEE Access 2023.
- Shenaj et al. (2025). LoRA.rar: Learning to Merge LoRAs via Hypernetworks for Subject-Style Conditioned Image Generation. ICCV 2025.
- Shenaj et al. (2026). K-Merge: Online Continual Merging of Adapters for On-device Large Language Models. ACL 2026.
- Lidin et al. (2026). Covenant-72B: Pre-Training a 72B LLM with Trustless Peers Over-the-Internet.
- Wang et al. (2025). Vision as LoRA.
- Wei et al. (2025). OptMerge: Unifying Multimodal LLM Capabilities and Modalities via Model Merging.
- Zhu et al. (2026). Outlier Matters: Efficient Long-to-Short Reasoning via Outlier-Guided Model Merging.
- We et al. (2025). Unlocking Efficient Long-to-Short LLM Reasoning with Model Merging.
- Liu et al. (2024). Towards Safer Large Language Models through Machine Unlearning. ACL 2024 (Findings).
- Qorbani et al. (2025). Semantic Library Adaptation: LoRA Retrieval and Fusion for Open-Vocabulary Semantic Segmentation. CVPR 2025.
- Zbeeb et al. (2025). Reasoning Vectors: Transferring Chain-of-Thought Capabilities via Task Arithmetic.