MixtureKit: A General Framework for Composing, Training, and Visualizing Mixture-of-Experts Models

Ahmad Chamma[1*], **Omar El Herraoui**[1*], **Guokan Shang**[1†]
[1]MBZUAI

Abstract

We introduce `MixtureKit`, a modular open-source framework for constructing, training, and analyzing Mixture-of-Experts (MoE) models from arbitrary pre-trained or fine-tuned checkpoints. MixtureKit supports three complementary strategies: (i) *Traditional MoE*, using a single router per transformer block to select experts; (ii) *BTX* (Branch-Train-Mix), adding routers at user-specified sub-layers for fine-grained token routing; and (iii) *BTS* (Branch-Train-Stitch), preserving experts intact and introducing lightweight stitch layers for controlled hub–expert information exchange. Given a single configuration dictionary, MixtureKit automatically modifies model configuration, patches decoder and causal LM classes, and exports a unified `transformers`-compatible checkpoint ready for inference or further fine-tuning. We also provide a visualization interface to inspect token routing, expert weight distributions, and layer-wise contributions. Experiments on multilingual code-switched (Arabic–Latin) data show that BTX models built with MixtureKit can outperform dense baselines across multiple benchmarks. The library is accessible at: `https://github.com/MBZUAI-Paris/MixtureKit`.

1 Introduction

Large Language Models (LLMs) have rapidly become central tools in NLP research and deployment, achieving strong performance across a wide range of downstream tasks (Yang et al., 2024). This growth has coincided with a surge in open-source and open-weight releases (Castaño et al., 2023), including models tailored for domains such as healthcare (Sellergren et al., 2025; Luo et al., 2022), software development (Rozière et al., 2024; Hui et al., 2024), and legal/financial practice (Wu et al., 2023; Yang et al., 2023). Much of the performance progress has been driven by scaling (Kaplan et al., 2020), producing large dense models (Team et al., 2025; Grattafiori et al., 2024) that require substantial resources for training and inference (Cottier et al., 2025). Continual fine-tuning also exposes trade-offs: catastrophic forgetting can be more pronounced in larger models (Luo et al., 2025), while smaller models may retain learning capacity but often lag in overall quality (Haque, 2025).

Mixture-of-Experts (MoE) models (Shazeer et al., 2017) provide a practical alternative by routing tokens to a sparse subset of expert subnetworks, increasing total capacity while keeping the *activated* parameters per token small. Several recent MoE systems rank among strong open models while offering lower inference cost (DeepSeek-AI et al., 2025; OpenAI et al., 2025; Yang et al., 2025). However, pre-training MoEs from scratch remains expensive and typically offers limited control over the domain specialization of each expert (Sukhbaatar et al., 2024). In parallel, the ecosystem produces many capable checkpoints that become under-used once surpassed by newer releases (Osborne et al., 2024). This motivates *recycling* pre-trained and fine-tuned models into a unified MoE model that can combine domain expertise with controllable routing and efficient updates.

We present `MixtureKit`, an open-source Python library designed to make this recycling workflow accessible and reproducible. MixtureKit composes arbitrary HuggingFace checkpoints into a single MoE model and automatically patches the architecture so the resulting checkpoint is immediately usable via `transformers`. While prior efforts provide implementations for specific model families, MixtureKit targets broader generalization, including custom-code architectures, by automating configuration edits and code rewriting. Our main contributions are:

*Equal contribution.
†Correspondence: {ahmad.chamma, omar.el-herraoui, guokan.shang}@mbzuai.ac.ae

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), pages 148–156

- **Unified composer and pipeline**: a configuration-driven, one-call workflow that composes checkpoints, patches decoder/LM classes, and exports a unified checkpoint ready for inference or fine-tuning.
- **Multiple MoE strategies**: support for *Traditional MoE*, *BTX*, and *BTS*, enabling both router-based and stitch-based expert integration, plus load balancing to mitigate inactive experts (Fedus et al., 2022).
- **Routing visualization**: an interactive interface for analyzing token routing, expert usage, and layer-wise contributions, supporting diagnosis of collapse and specialization.

2 Related Work

Mixture-of-Experts (MoE). The Mixture-of-Experts paradigm originates from early gating-based formulations (Jacobs et al., 1991), where a gating network routes inputs to specialized subnetworks (experts). By restricting updates to a subset of parameters, MoE mitigates interference across tasks and improves learning efficiency compared to large dense models.

Sparse MoEs were revitalized by Shazeer et al. (2017), who introduced conditional computation via a sparsely-gated top-k routing mechanism, activating only a subset of experts per token. Switch Transformers (Fedus et al., 2022) (top-1 routing) and GLaM (Du et al., 2022) (top-2 routing over 64 experts) demonstrated trillion-parameter scaling with improved compute efficiency relative to dense counterparts. However, these models were not released as open weights.

Mixtral (Jiang et al., 2024) marked a turning point as one of the first widely adopted open-weight sparse MoEs (8 experts, 2 active per token), matching or surpassing strong dense baselines such as LLaMA 2 70B and GPT-3.5 while achieving substantially faster inference due to sparsity.

Model Merging and Recycling. Model merging differs from traditional ensembles: ensembles combine predictions at inference with increased computational cost, whereas merging combines parameters into a single deployable model. Model Soups (Wortsman et al., 2022) demonstrated that simple weight averaging of fine-tuned checkpoints can retain collective strengths without additional inference overhead.

Li et al. (2022) introduced Branch-Train-Merge (BTM), where replicas of a base model are trained on data partitions and later combined. While effective, such approaches may limit continued expert specialization within a unified architecture. More recent recycling-based approaches (e.g., BTX-style integration (Sukhbaatar et al., 2024)) reinterpret trained replicas as experts under sparse routing, enabling reuse of existing checkpoints while preserving low activated compute.

Toolkits. `MergeKit` (Goddard et al., 2025) provides a YAML-based interface for checkpoint merging, including MoE-style merging with gate initialization strategies (e.g., hidden-state-based, embedding-based, or random). However, it primarily supports predefined MoE configurations (e.g., Mixtral, DeepSeek, Qwen) and does not offer open implementations of newer recycling techniques.

`Mergoo` (Leeroo AI, 2024) provides open-source implementations of BTX-style MoE merging with simplified configuration workflows, but relies on hand-crafted integrations for a limited set of model families (e.g., Phi3, Mistral, LLaMA), restricting portability. Our work complements these tools by enabling automatic patching of arbitrary checkpoints (including custom-code models), supporting routing- and stitch-based integration, and providing visualization utilities for interpretability. A detailed feature-level comparison across MoE strategies, architecture coverage, routing features, and interpretability is provided in Appendix A (Table 2).

3 MixtureKit: Design and Implementation

`MixtureKit` provides a method-agnostic pipeline that composes pretrained experts into a single checkpoint and patches the target architecture with either token routers (Traditional, BTX) or stitch layers (BTS). The system is driven by a configuration dictionary (Fig. 2) and exposes a uniform interface, so extending to new MoE variants largely reduces to registering a conversion rule for targeted submodules and a corresponding forward-pass hook.

3.1 Implemented Methods

Branch-Train-miX (BTX) Branch-Train-miX (BTX) (Sukhbaatar et al., 2024) is a recycling strategy that converts multiple trained replicas of a base model into a single sparse MoE (Fig. 1-B), avoiding the cost of training an MoE from scratch (Mu and Lin, 2025). Following the Branch-Train-Merge paradigm (Li et al., 2022), replicas are first trained

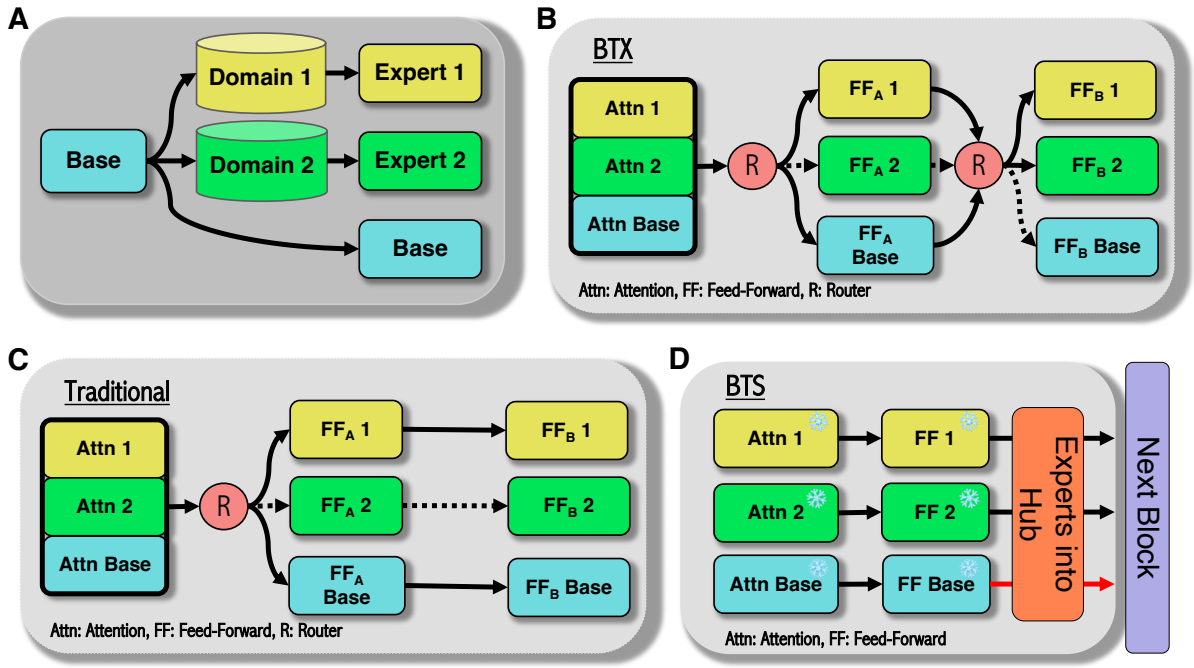

Figure 1: Workflow for building the unified MoE model. (**A**) Experts are obtained via continual pre-training or fine-tuning from HuggingFace checkpoints (Wolf et al., 2020), optionally including a base replica. (**B–C**) *BTX* and *Traditional MoE* routing. Solid lines indicate selected experts under top-2; dashed indicates an optional additional expert for top-k. FF_A and FF_B denote FFN internal projections (e.g., gate/up/down). (**D**) *BTS*: frozen hub and experts with trainable stitch layers; the red line shows the refined hub output.

independently on different data partitions (Fig. 1-A).

BTX then reinterprets selected feed-forward layers from each replica as experts within new MoE layers, while learning routing networks that assign each token to the top-k experts. Non-expert components such as attention and embeddings are typically merged via parameter averaging, yielding a unified backbone with expert-specialized FFNs. Routing is trained with load-balancing objectives to encourage effective expert utilization.

After composition, two adaptation strategies are commonly used: (i) router-only tuning, which is computationally efficient but may slightly reduce accuracy, and (ii) full fine-tuning of all parameters, which is more expensive yet often necessary to recover or improve performance after averaging. While effective, BTX requires decomposing and reintegrating model components into a shared MoE structure, limiting modular flexibility and interpretability compared to approaches that preserve experts as intact, separable units.

Traditional MoE BTX introduces separate routers at each internal FFN projection (gate, up, down), which increases routing flexibility but also adds routing complexity and parameters. To align with block-level routing in models such as Mixtral (Jiang et al., 2024), MixtureKit provides **Traditional MoE** (Fig. 1-C), where a single router selects one (or top-K) expert(s) per token and applies the same expert choice across all FFN projections. This reduces the number of routers and simplifies training, while preserving sparse activation at inference. As in prior work, Traditional routing can suffer from *dead experts*; MixtureKit includes a load-balancing loss with coefficient α to encourage more uniform expert utilization (Fedus et al., 2022), trading off balance against potential instability if α is too large.

Branch-Train-Stitch (BTS) Router-based recycling is effective but does not naturally yield a plug-in structure where intact experts can be appended and interpreted as separate modules. BTS (Zhang et al., 2025) addresses this by keeping experts *fully intact* in the final model (no parameter averaging) and adding lightweight *StitchLayer*s at selected transformer blocks (including the final block) at a user-defined frequency (Fig. 1-D). The hub (base) model and expert blocks remain frozen, while stitch parameters learn controlled information exchange. In the `experts-into-hub` direction, expert hidden states are projected into the hub space and com-

```
config = {
  "moe_method": "btx",
  "stitch_freq": 5,
  "model_type": "new_model_type",
  "num_experts_per_tok": 2,
  "experts": [
    {"expert_name": "base_expert", "model_id":
     "expert_base_checkpoint"},
    {"expert_name": "expert_1",    "model_id":
     "expert_1_checkpoint"},
    {"expert_name": "expert_2",    "model_id":
     "expert_2_checkpoint"},
  ],
  "router_layers": ["mlp.gate_proj",
    "mlp.up_proj", "mlp.down_proj"],
  "alpha": 0,
  "router_layers_index": []
}
```

Figure 2: Example configuration dictionary `config` for BTX with top-2 routing and load balancing disabled.

bined via a softmax gate conditioned on the hub state; in `hub-into-experts`, the hub state is projected into expert space and mixed via sigmoid gating. This yields efficient training and clearer modularity, at the cost of higher memory if many full experts must be loaded.

3.2 End-to-End Workflow

User-centric configuration. MixtureKit is controlled by a single configuration dictionary (Fig. 2), designed to make experiments repeatable and shareable. The field `moe_method` selects the strategy (`btx`, `traditional`, `bts`), while `model_type` defines the identifier for the exported checkpoint. The `experts` list specifies Hugging Face model IDs for the base and domain-specialized checkpoints. Sparsity is set by `num_experts_per_tok` (top-k). The fields `router_layers` and `router_layers_index` determine which submodules and which transformer blocks are converted. Method-specific knobs such as `stitch_freq` and `alpha` control stitch insertion and load balancing. A single call `build_moe(config)` exports a complete `transformers`-compatible checkpoint.

Expert composition. The pipeline begins with `compose()`, which loads each expert and integrates parameters into a unified state dictionary. For each parameter, MixtureKit decides whether it is shared across experts or stored as expert-specific. Shared parameters are averaged with shape-aware alignment (so small dimension mismatches can be handled when possible), while expert parameters are stored under a structured namespace (e.g., `experts.expert_i.weight`). This produces a coherent parameter layout expected by the patched modules.

Architecture patching and checkpoint export. After composition, `save_checkpoint()` copies the original `modeling_<base>.py` and `configuration_<base>.py` files, renames them for the new `model_type`, and updates imports/class names via `_replace_type_dependency`. Targeted edits then inject MoE logic: `_replace_script` rewrites `nn.Linear` and `Conv1D` layers in `Attention` and `MLP`, while `_modify_decoder` and `_modify_model` adjust forward passes to incorporate routing and (optionally) the load-balancing loss. For router-based methods, MixtureKit replaces selected layers with MoE-aware modules (`convert_linear_to_moe`) implementing top-k selection; for stitch-based methods, `_patch_stitches` introduces parallel expert streams and inserts `StitchLayers` at depth controlled by `stitch_freq`. Importantly, the same composition/export pipeline is reused across methods; only the forward computation and parameter organization differ.

Extensibility. Adding a new variant requires implementing a small adapter that (i) defines which layers are converted or stitched, and (ii) registers the corresponding patch rule. The new method then automatically benefits from MixtureKit's loading, saving, and compatibility pipeline.

3.3 Visualization and Interpretability

An important aspect of `MixtureKit` is the visualization interface, which provides real-time routing information for routing-based methods (*Traditional*, *BTX*). The tool is implemented as a Streamlit application and exposes both qualitative inspection and quantitative routing statistics.

Routing mechanism. Let $h_t^{(\ell,p)} \in \mathbb{R}^d$ denote token t's hidden state at block ℓ and projection $p \in \{\text{gate}, \text{up}, \text{down}\}$. Each projection has a gating matrix $W^{(\ell,p)} \in \mathbb{R}^{d \times E}$ producing expert logits:

$$g_{t,e}^{(\ell,p)} = h_t^{(\ell,p)} W^{(\ell,p)}, \qquad e \in \{1, \ldots, E\}.$$

The router selects the top-k experts

$$S_t^{(\ell,p)} = \text{TopK}\big(g_t^{(\ell,p)}, k\big),$$

and assigns normalized weights via softmax over the selected set

$$w_{t,e}^{(\ell,p)} = \frac{\exp(g_{t,e}^{(\ell,p)})}{\sum_{j \in S_t^{(\ell,p)}} \exp(g_{t,j}^{(\ell,p)})}, \qquad e \in S_t^{(\ell,p)}.$$

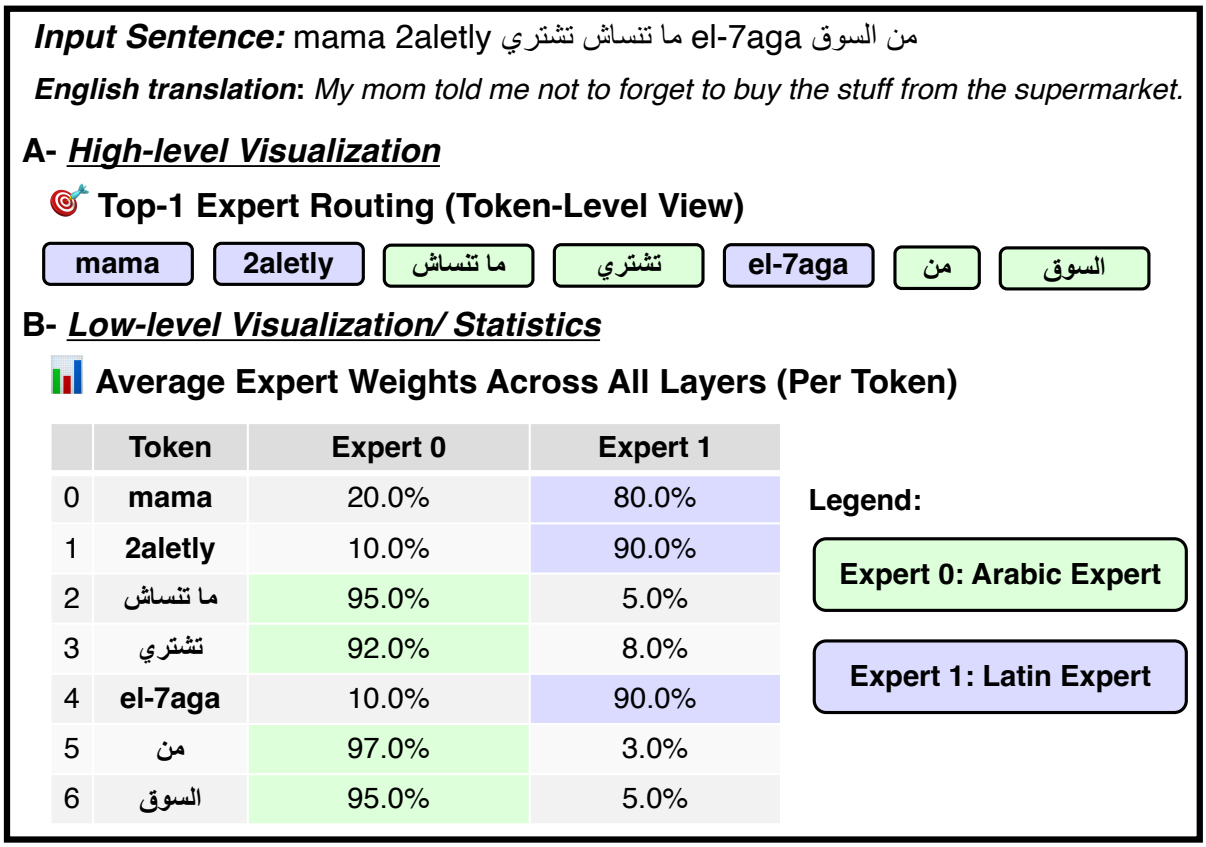

Figure 3: MixtureKit Token-Routing Visualizer: (**A**) high-level color-coded token assignments (dominant expert), and (**B**) average expert weights across selected layers for each token.

In *BTX*, each projection has its own router; in *Traditional MoE*, a single router is shared across projections in the block.

To provide an interpretable summary, MixtureKit aggregates normalized weights across depth. For token t, the aggregated contribution of expert e is

$$\overline{w}_{t,e} = \frac{1}{|\mathcal{L}_t|} \sum_{(\ell,p)\in\mathcal{L}_t} w_{t,e}^{(\ell,p)}, \qquad (1)$$
$$\mathcal{L}_t = \{(\ell,p) : t \text{ at block } \ell,\ \text{proj. } p\}.$$

Figure 3 shows (A) dominant expert assignments based on $\arg\max_e \overline{w}_{t,e}$ and (B) the full $\overline{w}_{t,e}$ distribution. The interface can also restrict aggregation to specific blocks or projections to study how routing evolves with depth. In practice, this helps diagnose under-utilization, detect routing collapse (one expert dominating), and analyze specialization behavior in multilingual and code-switched settings.

4 Practical Example: Sript-specialized experts

Egyptian Arabic, commonly referred to as *Masri*, is the most widely spoken Arabic dialect, with over 100 million native speakers in Egypt and broad intelligibility across the Arab world. It differs substantially from Modern Standard Arabic (MSA) in phonology, vocabulary, and grammar. A defining characteristic of Masri is its dual-script usage: speakers frequently write in both Arabic script and a Latin-based transliteration known as Arabizi or Franco-Arabic (Elnagar et al., 2021). This multi-script setting naturally motivates a modular approach, where script-specialized experts—one trained on Arabic-script data and another on Latin-script data—are integrated into a unified MoE model that dynamically routes tokens to the appropriate expert.

To demonstrate MixtureKit in a realistic and high-impact setting, we follow the experimental design of Shang et al. (2025) and reproduce the Nile-Chat setup. Starting from `Gemma3-4B-pt`, we continually pre-train two replicas on Arabic-script and Latin-script corpora to obtain script-specific

Model	Average Arabic	Average Latin	Translation Long (chrF)	Translation Short (chrF)	Transliteration (chrF)
Nile-Chat-4B-Arabic-Expert	53.21	44.63	58.81	52.70	26.21
Nile-Chat-4B-Latin-Expert	48.85	48.06	37.09	31.27	80.59
Nile-Chat-4B	53.01	49.07	58.40	52.01	80.44
Nile-Chat-2x4B-A6B	55.87	52.32	61.59	53.71	83.89
Nile-Chat-3x4B-A6B	55.74	51.23	**61.90**	**55.37**	**83.97**
Nile-Chat-12B	**60.00**	**52.61**	60.61	53.53	80.97

Table 1: Performance comparison of Arabic/Latin experts, Nile-Chat dense models (4B and 12B), and *BTX*-based counterparts across Arabic, Latin, and generation benchmarks. The highest scores are shown in **bold**, and the second-highest in underline.

experts. Using BTX, we compose these experts together with the original base model, forming a three-expert MoE where two experts are activated per token (6B activated parameters), resulting in *Nile-Chat-3x4B-A6B*. Incorporating the base checkpoint as a third expert preserves general knowledge and strengthens English capabilities beyond the script-specialized components. For comparison, we also construct *Nile-Chat-2x4B-A6B*, merging only the two script experts.

Training proceeds in two stages. First, we apply Supervised Fine-Tuning (SFT) with LoRA (alpha 512, learning rate $1e^{-4}$, effective batch size 256), mixing in English instructions from Wild-Chat to maintain the English-centric strengths of the base expert. Second, we perform Direct Preference Optimization (DPO) for alignment.

Evaluation follows Shang et al. (2025), reporting averages over Egyptian benchmarks in both scripts (accuracy/normalized accuracy) and generation benchmarks for translation and transliteration (chrF). As shown in Table 1, both BTX variants strike a balance between dense 4B and 12B baselines on Arabic-script discriminative tasks ($53.01 < \underline{55.87} <$ **60.0**), remain competitive on Latin script ($\underline{52.32} \approx$ **52.61**), and outperform dense models on generation-heavy translation and transliteration tasks, highlighting the benefit of routing heterogeneous script expertise under sparse activation.

5 Conclusion

We introduced `MixtureKit`, an open-source framework that lowers the barrier to building recycled MoE models by composing existing pretrained and fine-tuned checkpoints into a unified `transformers`-compatible model. MixtureKit supports multiple expert-integration strategies (Traditional, BTX, BTS), automates architecture patching and checkpoint export, and includes a visualization interface for interpreting token routing and diagnosing imbalance. Through a script-specialized Egyptian Arabic case study, we show that BTX models built with MixtureKit reproduce and improve over dense baselines on multiple benchmarks. Overall, MixtureKit provides a practical foundation for MoE research and deployment across multilingual and domain-specialized settings.

6 Future Work

MixtureKit currently relies on regex-based submodule matching and the HuggingFace versioning of configuration/modeling files; we aim to broaden version coverage and make conversion rules more robust. The exported configurations also do not yet integrate recent inference optimizations such as MoE kernels in *vLLM*[1]. Finally, while most work assumes identical architectures across experts, an interesting direction is cross-architecture composition.

References

Joel Castaño, Silverio Martínez-Fernández, Xavier Franch, and Justus Bogner. 2023. Exploring the carbon footprint of hugging face's ml models: A repository mining study. In *2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, page 1–12. IEEE.

Ben Cottier, Robi Rahman, Loredana Fattorini, Nestor Maslej, Tamay Besiroglu, and David Owen. 2025. The rising costs of training frontier ai models. *Preprint*, arXiv:2405.21015.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang

[1]https://github.com/vllm-project/vllm

Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, and 181 others. 2025. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, and 8 others. 2022. Glam: Efficient scaling of language models with mixture-of-experts. *Preprint*, arXiv:2112.06905.

Ashraf Elnagar, Sane M. Yagi, Ali Bou Nassif, Ismail Shahin, and Said A. Salloum. 2021. Systematic literature review of dialectal arabic: Identification and detection. *IEEE Access*, 9:31010–31042.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Preprint*, arXiv:2101.03961.

Charles Goddard, Shamane Siriwardhana, Malikeh Ehghaghi, Luke Meyers, Vlad Karpukhin, Brian Benedict, Mark McQuade, and Jacob Solawetz. 2025. Arcee's mergekit: A toolkit for merging large language models. *Preprint*, arXiv:2403.13257.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Naimul Haque. 2025. Catastrophic forgetting in llms: A comparative analysis across language tasks. *Preprint*, arXiv:2504.01241.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, and 5 others. 2024. Qwen2.5-coder technical report. *Preprint*, arXiv:2409.12186.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. 1991. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, and 7 others. 2024. Mixtral of experts. *Preprint*, arXiv:2401.04088.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *Preprint*, arXiv:2001.08361.

Leeroo AI. 2024. Mergoo: A library for easily merging multiple LLM experts. `https://github.com/Leeroo-AI/mergoo`. Accessed: 2026-05-15.

Margaret Li, Suchin Gururangan, Tim Dettmers, Mike Lewis, Tim Althoff, Noah A. Smith, and Luke Zettlemoyer. 2022. Branch-train-merge: Embarrassingly parallel training of expert language models. *Preprint*, arXiv:2208.03306.

Renqian Luo, Liai Sun, Yingce Xia, Tao Qin, Sheng Zhang, Hoifung Poon, and Tie-Yan Liu. 2022. Biogpt: generative pre-trained transformer for biomedical text generation and mining. *Briefings in Bioinformatics*, 23(6).

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2025. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *IEEE Transactions on Audio, Speech and Language Processing*, 33:3776–3786.

Siyuan Mu and Sen Lin. 2025. A comprehensive survey of mixture-of-experts: Algorithms, theory, and applications. *Preprint*, arXiv:2503.07137.

OpenAI, :, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, and 108 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.

Cailean Osborne, Jennifer Ding, and Hannah Rose Kirk. 2024. The ai community building the future? a quantitative analysis of development activity on hugging face hub. *Journal of Computational Social Science*, 7(2):2067–2105.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, and 7 others. 2024. Code llama: Open foundation models for code. *Preprint*, arXiv:2308.12950.

Andrew Sellergren, Sahar Kazemzadeh, Tiam Jaroensri, Atilla Kiraly, Madeleine Traverse, Timo Kohlberger, Shawn Xu, Fayaz Jamil, Cían Hughes, Charles Lau, Justin Chen, Fereshteh Mahvar, Liron Yatziv, Tiffany Chen, Bram Sterling, Stefanie Anna Baby, Susanna Maria Baby, Jeremy Lai, Samuel Schmidgall, and 62 others. 2025. Medgemma technical report. *Preprint*, arXiv:2507.05201.

Guokan Shang, Hadi Abdine, Ahmad Chamma, Amr Mohamed, Mohamed Anwar, Abdelaziz Bounhar,

Omar El Herraoui, Preslav Nakov, Michalis Vazirgiannis, and Eric Xing. 2025. Nile-chat: Egyptian language models for arabic and latin scripts. *Preprint*, arXiv:2507.04569.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *Preprint*, arXiv:1701.06538.

Sainbayar Sukhbaatar, Olga Golovneva, Vasu Sharma, Hu Xu, Xi Victoria Lin, Baptiste Rozière, Jacob Kahn, Daniel Li, Wen tau Yih, Jason Weston, and Xian Li. 2024. Branch-train-mix: Mixing expert llms into a mixture-of-experts llm. *Preprint*, arXiv:2403.07816.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, and 150 others. 2025. Kimi k2: Open agentic intelligence. *Preprint*, arXiv:2507.20534.

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.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. 2022. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. *Preprint*, arXiv:2203.05482.

Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. 2023. Bloomberggpt: A large language model for finance. *Preprint*, arXiv:2303.17564.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Hongyang Yang, Xiao-Yang Liu, and Christina Dan Wang. 2023. Fingpt: Open-source financial large language models. *Preprint*, arXiv:2306.06031.

Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Shaochen Zhong, Bing Yin, and Xia Hu. 2024. Harnessing the power of llms in practice: A survey on chatgpt and beyond. *ACM Trans. Knowl. Discov. Data*, 18(6).

Qizhen Zhang, Prajjwal Bhargava, Chloe Bi, Chris X. Cai, Jakob Foerster, Jeremy Fu, Punit Singh Koura, Ruan Silva, Sheng Shen, Emily Dinan, Suchin Gururangan, and Mike Lewis. 2025. Bts: Harmonizing specialized experts into a generalist llm. *Preprint*, arXiv:2502.00075.

A Comparison with Existing Frameworks

To situate MixtureKit within the broader ecosystem of model-merging and MoE-composition toolkits, Table 2 provides a detailed feature-level comparison against the two most closely related libraries: MergeKit (Goddard et al., 2025) and Mergoo (Leeroo AI, 2024). The comparison is organized along six dimensions: (i) the MoE integration strategies each toolkit implements, (ii) the weight-space merging algorithms available, (iii) coverage of model architectures and HuggingFace checkpoints, (iv) routing and training features, (v) the end-user workflow, and (vi) interpretability tooling. MergeKit excels as a general-purpose weight-space merging library with a wide algorithmic catalog (LERP, SLERP, TIES, DARE, Task Arithmetic, passthrough), but its MoE export is restricted to three predefined architectures (Mixtral, DeepSeek-MoE, Qwen-MoE). Mergoo focuses on BTX-style routing and Mixture-of-Adapters with LoRA experts, but supports only a small set of base families (Llama, Mistral, Phi3, BERT). MixtureKit complements both by automating architecture patching for arbitrary (including custom-code) checkpoints, supporting three complementary MoE strategies (Traditional, BTX, BTS), and shipping an interactive routing visualizer for interpretability.

Dimension	MergeKit	Mergoo	MixtureKit (Ours)
MoE Integration Strategies			
Traditional block-level MoE (single router)	✓	✗	✓
BTX (Branch-Train-Mix, per-projection routing)	✗	✓	✓
BTS (Branch-Train-Stitch, intact experts + stitch layers)	✗	✗	✓
Mixture-of-Adapters (LoRA experts)	✗	✓	✗
Weight-Space Merging Methods			
Linear / SLERP averaging	✓	∼	∼
Task Arithmetic / TIES / DARE	✓	✗	✗
Passthrough / depth up-scaling (FrankenMerging)	✓	✗	✗
Model / Architecture Coverage			
Architecture-agnostic MoE export (beyond a fixed set)	✗[†]	✗[‡]	✓
Supports arbitrary HuggingFace checkpoints for MoE export	✗	✗	✓
Supports custom-code (`trust_remote_code`) models for MoE export	✗	✗	✓
Automatic decoder / causal-LM class patching	✗	∼	✓
Unified `transformers`-compatible export	✓	✓	✓
Routing & Training Features			
Top-k sparse routing	✓	✓	✓
Per-projection routers (gate / up / down)	✗	✓	✓
User-selectable router-layer indices	✗	✓	✓
Gate initialization heuristics (hidden / embed / random)	✓	✗	✗
Router-only or full fine-tuning	∼	✓	✓
Load-balancing loss (coefficient α)	✗	∼[§]	✓
Frozen hub + trainable stitch layers	✗	✗	✓
Configurable stitch insertion frequency	✗	✗	✓
User Workflow			
Single-call build pipeline	∼	✓	✓
Declarative configuration (YAML / dict)	✓	✓	✓
Graphical user interface	✓	✗	✗
Lazy tensor loading / out-of-core merging	✓	∼[§]	✗
Extensible via adapter + patch-rule registration	∼	✗	✓
Interpretability			
Token-routing visualization interface	✗	✗	✓
Per-layer / per-projection routing statistics	✗	✗	✓
Dead-expert / routing-collapse diagnosis	✗	✗	✓

Table 2: Comparison of MixtureKit with existing model-merging and MoE-composition toolkits. ✓ full support, ∼ partial / planned / indirect support, ✗ no support. [†] MergeKit's `mergekit-moe` can only output Mixtral, DeepSeek-MoE, or Qwen-MoE architectures, requiring input experts to share a common base family (Llama / Mistral / Qwen2). [‡] Mergoo's BTX-style MoE export only supports Llama, Mistral, Phi3, and BERT base families. [§] Router load-balancing loss and lazy tensor loading appear on Mergoo's public roadmap but are not yet implemented in released versions. MergeKit's strength is a broad catalog of weight-space merging algorithms (LERP, SLERP, TIES, DARE, Task Arithmetic, passthrough); Mergoo's contribution is BTX-style routing with LoRA-adapter mixtures; MixtureKit complements both by automating architecture patching for arbitrary (including custom-code) checkpoints, supporting three complementary MoE strategies (Traditional, BTX, BTS), and shipping an interactive routing visualizer for interpretability.