BTS: Harmonizing Specialized Experts into a Generalist LLM

Qizhen Zhang[1,2,†] **Prajjwal Bhargava**[1,◇] **Chloe Bi**[1,◇] **Chris X. Cai**[†,◇]
Jakob Foerster[1,2,◇] **Jeremy Fu**[1,◇] **Punit Singh Koura**[1,◇] **Ruan Silva**[1,◇] **Sheng Shen**[†,◇]
Emily Dinan[1,*] **Suchin Gururangan**[†,*] **Mike Lewis**[1,*]

[1]Meta Superintelligence Labs [2]University of Oxford
qizhen.zhang@eng.ox.ac.uk, edinan@meta.com

Abstract

We present BRANCH-TRAIN-STITCH (BTS), an efficient and flexible training algorithm for combining independently trained large language model (LLM) experts into a single, capable generalist model. Following Li et al. [2022], we start with a single *seed* LLM which is branched into domain-specific (e.g., coding or math) experts with continual pretraining. BTS combines experts using lightweight stitch layers, which are inserted between frozen experts and the seed LLM, and trained on a small datamix of the expert domains. Stitch layers enable the seed LLM to integrate representations from any number of experts during the forward pass, allowing it to generalize to new domains, despite remaining frozen. Because BTS does not alter the constituent LLMs, BTS provides a modular and flexible approach: experts can be easily removed or added with only a small amount of training. Compared to alternative model merging or upcycling approaches, BTS yields the best generalist performance on a variety of downstream tasks, while retaining the specialized capabilities of each of the experts.

1 Introduction

To achieve strong performance across diverse domains, large language models (LLMs) are often densely trained on trillions of tokens using thousands of GPUs [Dubey et al., 2024]. However, this approach poses three challenges. **1) Infrastructure limitations**: Large-scale training requires massive synchronization across distant compute clusters. The necessary interconnection hardware is costly [Cottier et al., 2024], and the resulting inter-device communication overhead significantly slows down training [Gholami et al., 2024, Fernandez et al., 2024, Narayanan et al., 2021]. Moreover, the more devices involved, the greater the risk of hardware failure, where a single device failure can stop the entire training process [Kokolis et al., 2025, Zhang et al., 2022a]. **2) Inefficient reuse of existing models**: Many domain-specific expert models are already publicly available [Azerbayev et al., 2023, Roziere et al., 2023, Huang et al., 2023]. Pretraining from scratch is inefficient as it does not reuse existing models' expertise. **3) Data-mix tradeoffs** [Xie et al., 2023, Ye et al., 2024]: It can be challenging to improve performance on a new domain without forgetting the original data [McCloskey and Cohen, 1989, Aghajanyan et al., 2021] or correct unwanted behaviors without impacting others [Tuan et al., 2024].

"Branch-Train" methods [BTM; Li et al., 2022, BTX; Sukhbaatar et al., 2024, BAM; Zhang et al., 2024] address all three challenges by asynchronously training smaller, distinct expert models specialized to different domains in parallel. BTM [Li et al., 2022] efficiently combines the expert models by ensembling them at inference time, but is limited because there are no learned connections between expert layers; this restricts the model's overall expressivity, especially in distant test domains. On the other hand, approaches like BTX [Sukhbaatar et al., 2024] and BAM [Zhang et al., 2024], which upcycle expert models into a Mixture-of-Experts (MoE) model [Shazeer et al., 2017], show strong downstream task performance, but lose the flexibility and interpretability inherent in a modular approach where experts remain distinct and intact.

We present BRANCH-TRAIN-STITCH (BTS), a new algorithm for building a generalist LLM from a collection of smaller expert models which achieves the best generalist model performance. Like other "Branch-Train" methods, BTS begins with a training phase in which small experts are asynchronously created in parallel via independent continued pretraining on domains of interest. The experts are then adapted into a unified, general-

[*] Joint last author, ordered alphabetically
[†] Work done at Meta
[◇] Ordered alphabetically

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 6816–6834
November 4-9, 2025

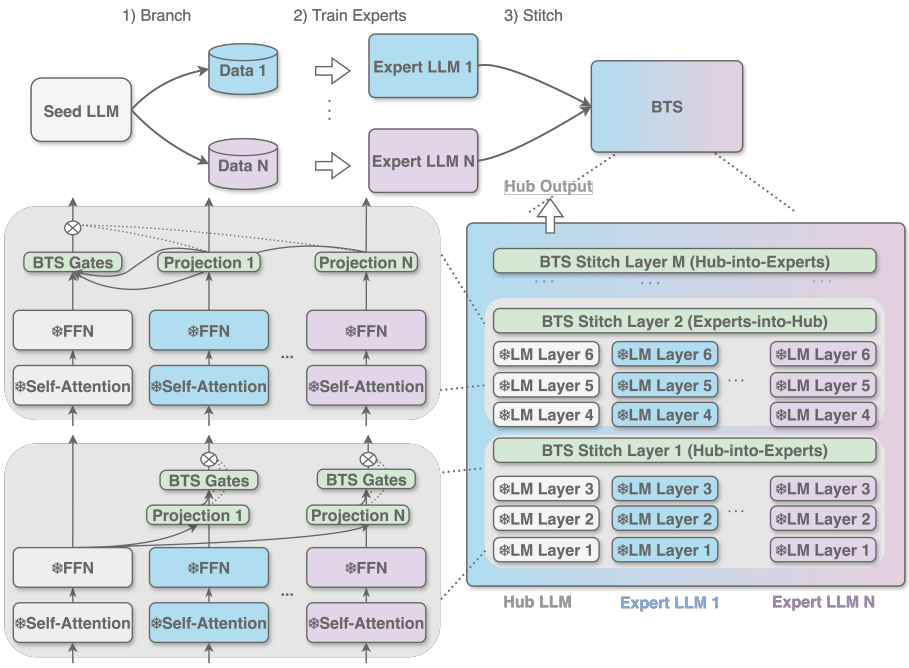

Figure 1: **Overview of the BTS algorithm.** BTS operates in three phases. Different colors correspond to different expert domains. **1) Branch & 2) Train**: Following Li et al. [2022], we begin with a pretrained seed LLM and create N copies. Each copy is independently pretrained on its respective data mixture, resulting in specialized expert models. **3) Stitch**: Stitch layers are inserted throughout the LLM layers, alternating between the *Experts-into-Hub* and the *Hub-into-Experts stitch layer*. Only the stitch layers are updated during this training phase. The BTS model always has a Experts-into-Hub stitch layer as the last layer, as the hub output is returned as the final BTS output.

ist model by inserting and training *stitch layers* between models, while keeping the experts themselves frozen.

This stitching architecture adds connections between experts via a gating mechanism on top of the language model layer outputs which determine how hidden states from one expert flow into another. One can imagine several ways to combine representations produced by different experts: all experts can directly connect to all other experts, only certain experts can connect to certain others, and everything in between. We opt for a hub-and-spoke model, in which a central "hub" model (the seed LLM) can update its own representations via the spokes (specialized experts), and vice versa, but the experts have no direct connection to each other. This design choice balances efficiency and performance. Since the seed model is trained on a variety of data, it is a natural choice for the hub, so all of our experiments adopt this set-up.

For each layer in the forward pass, the stitching architecture alternates between *hub-to-expert merging*, where the hidden representations of the experts are updated with a projected hub LLM representation, and *expert-to-hub merging*, where the hub's hidden representation is updated with a combined hidden representation of all experts. The final output provided by the merged LLM is the output of the hub (or, in our case, seed) model. These design choices are further motivated and validated empirically with ablations in §4.

In experiments (§3), we find that BTS achieves the best generalist model performance compared to both expert merging and expert upcycling baselines and can even perform better than some individual experts on their target tasks. Notably, this is achieved with training only the small set of stitching parameters. The modular design of BTS, in which individual experts remain unchanged in the merging process, offers flexibility and interpretability. Targeted performance improvements for specific domains can be achieved completely asynchronously. Furthermore, downstream behaviors can be easily understood by analyzing which experts are 'active' at any given token, providing transparency into the model's decision-making process.

Our contributions are summarized as follows:

- **BRANCH-TRAIN-STITCH, §2**: We propose BRANCH-TRAIN-STITCH, an efficient and flexible approach stitching distinct expert models into a more powerful, generalist LLM.

- **Experiments, §3**: We validate this approach through experiments on seed language models of 2.7B parameters. Our results show that BTS outperforms competitive baselines in downstream task performance, achieving the best average performance across benchmarks.
- **Analysis, §4**: We motivate BTS architectural choices with ablations and investigate the impact on "cross capability" tasks, i.e. tasks at the intersection of expert domains, and show that, in certain settings, BTS can achieve cross capability performance greater than any expert. Finally, we provide a detailed analysis of the behavior of the stitch layer at inference time, demonstrating interpretable patterns where the specialized expert associated with the task is typically the most utilized.

2 BRANCH-TRAIN-STITCH

2.1 BTS algorithm overview[1]

The BTS algorithm (Figure 1) involves three stages, resulting in an efficiently-trained generalist dense model.

1. **Branch**: Following Li et al. 2022, given a pretrained Transformer seed model m_0, we create n copies of the model $m_1, ..., m_n$.

2. **Train**: Also following Li et al. 2022, each copy of the seed model m_i independently undergoes a continued pretraining phase on a specialized data mixture, $\mathcal{D}_i$, each tailored to different domains such as code, math, and multilingual [Gururangan et al., 2020]. This phase yields specialized models that have enhanced performance within their respective domains compared to the seed model m_0. However, these models might perform worse in domains outside of their specialization as they forget knowledge from the initial pretraining phase. We refer to these models m_i as *experts*, and note that this usage of the term "expert" differs in meaning from the feed forward network (FFN) experts in MoE models.

3. **Stitch**: We merge the seed (m_0) and expert models ($m_i, i > 0$) from the previous steps using our lightweight stitch layers Ψ, which are trained for a small number of steps on a mixture of data from expert domains. The stitch layer architecture is detailed in §2.2. Importantly, *only* the stitch layers are updated during this phase, while the parameters of the seed and expert models remain frozen. This ensures that BTS training is a flexible approach — experts can be added or removed after merging, only requiring retraining stitch layer parameters.

[1]Relevent backgrounds and notions on language model architectures are provided in Appendix B

2.2 Model architecture

We next provide more details on the BTS architecture (Figure 1). We introduce the *stitch layer*, which merges $n + 1$ Transformer models $m_0, ..., m_n$. We designate m_0 as the *hub* and $m_1, ..., m_n$ as the *experts*. The hub is usually the seed model, unless otherwise noted.

Suppose the expert m_i contains L Transformer layers, $\{\ell_i^j\}_{j=1}^L$. We insert K stitch layers – one each after every $\lfloor \frac{L}{K} \rfloor$ Transformer layers. We denote Ψ_j as the stitch layer inserted after Transformer layers $\{\ell_i^j\}_{i=0}^n$.

The stitch layer Ψ_j, takes as input the hidden states, or outputs, from the hub's j-th layer ℓ_0^j and the experts' j-th layers, $\{\ell_i^j\}_{i=1}^n$. We denote the hidden states respectively as h_0^j for the hub and $\{h_i^j\}_{i=1}^n$ for the experts. The outputs of the stitch layer, $\Psi_j(h_0^j, \ldots, h_n^j) = (\tilde{h}_0^j, \ldots, \tilde{h}_n^j)$, become the input to the corresponding experts m_i's $j+1$-th layer (ℓ_i^{j+1}). Each stitch layer Ψ introduces two sets of learnable parameters:

1. **Linear projections**, $\{w_{\text{proj}_1}, ..., w_{\text{proj}_n}\}$, where $w_{\text{proj}_i} \in \mathbb{R}^{\text{dim} \times \text{dim}}$ either projects the expert hidden states to the hub model's hidden state space or projects the hub model's hidden state into the expert's hidden state space.

2. **A linear gate** $w_{\text{gate}} \in \mathbb{R}^{\text{dim} \times \text{dim} \times n}$, which computes the contribution of each model's hidden state.

To apply these gates, we alternate between two types of stitch layers (refer to Figure 1 for the illustration and Appendix F for the pseudo code):

The Experts-into-Hub Stitch Layer In this layer, the expert models' hidden states are first projected into the hub model's hidden state space. The hub combines its own hidden state with the projected experts' hidden states, weighted by the outputs of a softmax-based gating mechanism.

$$
\begin{aligned}
g &= \text{softmax}(\text{dropout}(w_{\text{gate}}(h_0))) \\
\tilde{h}_i &= w_{\text{proj}_i}(h_i),\ i \in \{1, \ldots, n\} \\
\tilde{h}_0 &= h_0 * g_0 + \sum_{i=1}^{n} g_i * \tilde{h}_i
\end{aligned}
\tag{1}
$$

The Hub-into-Experts Stitch Layer In this layer, the hub hidden state is projected into each of the expert model's hidden state space. Each expert combines its own hidden state with a gated projection of the hub hidden state using a sigmoid-based gating mechanism:

$$
\begin{aligned}
g &= \text{Sigmoid}(\text{dropout}(w_{\text{gate}}(h_0))) \\
\tilde{h}_0 &= h_0 \\
\tilde{h}_i &= (1-g_i)*h_i + g_i*w_{\text{proj}_i}(h_0),\ i \in \{1,\ldots,n\}
\end{aligned} \tag{2}
$$

As we demonstrate in §4, this alternating architecture is essential for enabling cross capabilities without degrading generalist performance.

3 Results: Building a generalist model

We validate the BTS approach through experiments with a seed language model of 2.7B parameters.

3.1 Model details[2]

Seed model We pretrain a 2.7B parameter language model, following the same text recipe used in Llama 3 [Dubey et al., 2024]. See Table 6 in Appendix A for architecture details. The seed model is trained for 15T tokens over 2.2M steps.

Expert models We create three copies of the seed model, each of which is continually trained for 96k training steps over a 200B token specialized data mixture to produce expert models for code, math, and multilingual tasks. We use a batch size of 2M tokens. In practice, one can leverage existing open-sourced experts such as [Rozière et al., 2023, Azerbayev et al., 2023]. We pretrained our own experts to have full transparency into the data mixtures for controlled experiments.[3]

BTS model We use four stitch layers to combine the seed model together with the three expert models. The four stitch layers are inserted after every five layers in the seed and expert models. We refer to the resulting model as the BTS model. As described in §2, the four stitch layers alternate between a *Hub-into-Expert layer* and *Experts-into-Hub stitch layer*. Upon initialization, the BTS model is further trained for 15B tokens over 7000 steps using a batch size of 2M tokens. The optimization objective is to minimize the next-token prediction loss from the hub model's output.

[2]See Appendix C for additional model details.

[3]This also follows the same experiment setup from [Li et al., 2022, Sukhbaatar et al., 2024, Zhang et al., 2024]

	Training	Total	Active
Expert upcycling			
BTX Sample	7.2B	7.2B	2.9B
BTX Soft	7.2B	7.2B	7.2B
BAM	8.4B	8.4B	8.4B
Expert merging			
Model Soup	N/A	2.7B	2.7B
BTM	N/A	10.8B	10.8B
Expert Routing	15k	10.8B	2.7B
BAM Adapters	1.5B	9.9B	9.9B
BTS	264M	11B	11B

Table 1: **Training, total, and active parameter count** for BTS and baselines. Unlike expert upcycling methods, expert merging methods require minimal number of training parameters, making them more modular, interpretable, and easy to train.

During the BTS training phase, only the stitch layers are updated while all the parameters of the seed model and the expert models are frozen.

3.2 Data details

Seed model We adopt the same text pretraining mixture as Llama 3 [Dubey et al., 2024].

Expert models Each dense expert is continued pretrained on a specialized data mixture for 200B tokens. The **Code expert** uses a recipe similar to that of CodeLlama [Rozière et al., 2023] with $> 85\%$ code tokens, utilizing the code data subset of the seed model mixture. The **Math expert** trains on OpenWebMath [Paster et al., 2023]. The **Multilingual expert** trains on 90% non-English and 10% English data from the seed mixture, following [Dubey et al., 2024].

BTS model The data mixture for the BTS training phrase contains 15% each from the code, math, and multilingual expert domains. The remaining 55% contains the seed model's pretraining data outside of these domains.

3.3 Baselines

In addition to the seed and expert models, we also compare BTS with *expert upcycling* and *expert merging* baselines. We use *expert upcycling* to describe methods where the seed and expert models initialize an MoE model, which is further trained.

The entire MoE is updated during training and as such the experts and seed model themselves do not remain intact. This approach loses the flexibility and interpretability inherent in a more modular approach, and any model change requires updating a large number of parameters. In contrast, *expert merging* methods like BTS keep seed and expert weights frozen during merging, preserving flexibility and interpretability.

Expert upcycling baselines:

- **BTX [Sukhbaatar et al., 2024]:** We upcycle the seed and three expert models into an MoE[4]. Our baselines include two BTX variants, where the Feedforward Network (FFN) experts employ one of two routing strategies: 1) *sample top-1 routing* [Sukhbaatar et al., 2024], where we use a Gumbel-Softmax [Jang et al., 2016] for the routing function, and 2) *soft-routing*, where all four experts are activated at all times. We use the same experiment setup as BTS runs.
- **BAM [Zhang et al., 2024]:** We upcycle the seed model and the three expert models into an MoE with both attention experts [MoA; Zhang et al., 2022b] and FFN experts[5]. We employ soft-routing for both sets of experts, ensuring that, like BTS, all FFN and attention parameters of the seed and expert models are activated during training and inference. We use the same experiment setup as BTS runs.

Expert merging baselines:

- **Model soup [Wortsman et al., 2022]:** We average the weights of the seed and expert models. No further training is required upon initialization.
- **BTM [Li et al., 2022]:** We ensemble the output logits of the seed and expert models. The ensemble weights are estimated using Bayes' rule with a uniform prior [Li et al., 2022, Gururangan et al., 2023]. No further training is required upon initialization.
- **Expert routing:** We train a linear router with cross-entropy loss to select the model with the lowest next-token loss for a given prompt, routing the entire sequence to that model. It is trained for 1B tokens with a constant learning rate of $5e-4$ and batch size of 1M, as further training showed no benefit.
- **BAM with adapters [Zhang et al., 2024]:** We train an expert-intact variant of BAM with soft-routing, adding a linear adapter $W_{\text{proj}i} \in \mathbb{R}^{dim \times dim}$ to each attention and FFN expert output. Formally, we replace Equation 4 and Equation 5 by the following:

$$y_{\text{MoE}} = \sum_i p_i(x) W_{\text{ffn}_i}(\text{FFN}_i(x))$$
$$y_{\text{MoA}} = \sum_i q_i(x) W_{\text{attn}_i}(\text{Attn}_i(x)) \quad (3)$$

  where p and q are the normalized router probabilities. Only the router and adapters are trained; all other parameters remain frozen. We use the same experiment setup as BTS.

We report training, active, and total parameters in Table 1. While BTS has the highest total parameter count, it requires far fewer training parameters than expert upcycling methods. Since BTS inserts a stitch layer every five layers, we compare its efficiency per five layers to the strongest baseline, BAM, in Appendix E. We find that both models achieve similar inference FLOPs per five layers, but BTS has substantially lower training costs.

3.4 Evaluation

We evaluate zero- and few-shot performance on tasks relevant to the expert domains.

General Knowledge and Reasoning We report MMLU [5-shot; Hendrycks et al., 2021a] and Big-Bench Hard (BBH) [3-shot; Suzgun et al., 2022].

Code We evaluate on MBPP [3-shot; Austin et al., 2021] and HumanEval (HE) [0-shot; Chen et al., 2021] benchmarks.

Multilingual We use machine translation tasks in Flores [1-shot; Goyal et al., 2022] on seven languages: Dutch, Spanish, Portuguese, Vietnamese, Indonesian, Hindi, and French. We report two categories: (S) English as the source translation language and (T) English as target translation language.

Math We report the performance on GSM8K [8-shot; Cobbe et al., 2021] and MATH [4-shot; Hendrycks et al., 2021b].

3.5 Results

Results on general knowledge, code, multilingual, and math benchmarks for the seed model, expert

[4]See Appendix B for details on the MoE architecture

[5]See Appendix B for a description of the attention experts architecture.

	General		Code		Multilingual		Math		
	MMLU	BBH	MBPP	HE	Flores(S)	Flores(T)	GSM8K	MATH	Avg.
2.7B Dense models									
Seed Model	28.4	35.6	27.0	20.7	29.5	35.7	10.5	4.82	24.0
Code Expert	30.3	35.2	**32.0**	*__25.0__	29.0	35.5	11.4	4.40	25.4
Multiling. Expert	26.6	34.7	26.2	18.3	*__31.9__	*__37.1__	10.8	4.16	23.7
Math Expert	*__36.3__	*__37.2__	26.2	16.5	23.6	32.7	*__20.5__	**10.1**	**25.4**
Expert upcycling									
BTX Sample	30.4	36.6	30.0	21.3	30.5	36.0	13.9	6.58	25.7
BTX Soft	34.7	36.8	29.6	23.2	31.0	36.0	19.2	9.10	27.4
BAM	35.2	**37.1**	29.8	22.6	31.0	36.1	**20.3**	10.1	27.8
Expert merging									
Model Soup	30.7	37.0	29.6	22.6	29.5	36.2	13.6	6.46	25.7
BTM	30.6	37.0	31.8	**23.8**	**31.8**	**37.0**	12.7	10.1	26.9
Expert Routing	28.4	35.6	27.0	**23.8**	30.8	**37.0**	10.5	5.04	24.8
BAM Adapters	34.0	37.0	28.8	22.6	31.0	36.1	18.8	10.0	27.3
BTS	**35.8**	36.9	*__32.2__	22.0	30.9	36.2	20.2	*__10.6__	*__28.1__

Table 2: **Performance of BTS against expert merging and upcycling methods, seed and expert models** measured on popular benchmarks across several capabilities. **Bolded** numbers indicate the best performance among dense models or merged models, while an asterisk (*) denotes the best performance across all models. Among all models, BTS achieves the best average performance. Notably, BTS is the only method that outperforms domain-specific experts on their own tasks, surpassing the coding expert on MBPP and the math expert on MATH.

models, and all expert merging and expert upcycling baselines are reported in Table 2. We make the following observations:

- **Expert models highlight datamix tradeoffs:** While the dense expert models achieve the best results in their domains, they significantly underperform on other domains. This highlights that improving performance in one domain may come at the cost of regressing in others. For example, the Math expert outperforms all models in GSM8K, but lags behind the seed model substantially in coding tasks.
- **Learned connections are important for expressive merging:** Methods like BAM with adapters and BTS outperform expert merging methods without learned connections between experts, such as Model Soup, BTM, and Expert Routing. This demonstrates the importance of adding intermediate learned connections between experts.
- **BTS achieves the best generalist performance:** Among all models – seed, expert, expert merging, and experts upcycling – BTS achieves the highest average performance across tasks. Notably, BTS achieves similar or better performance to the expert upcycling baselines at only a fraction of the training parameters.
- **BTS can outperform individual experts in their specialized tasks:** BTS emerges not only as the most well-rounded generalist model, but is also the *only* model which achieves *better* performance than any individual expert in some tasks. BTS outperforms the Code expert in MBPP and the Math expert in the MATH task.

4 Ablations and analysis

4.1 Enabling cross capabilities

In addition to evaluating merged models on the union of the expert capabilities, we also explore whether merged models can demonstrate entirely new capabilities at the *intersection* of expert specialties [Zhong et al., 2024]. For example – can a Russian-language expert and a Math expert be combined in such a way that the merged model performs better than either expert at Russian math tasks? We refer to these as cross capabilities.

Cross capabilities experimental set-up In order to evaluate cross capabilities, we train an additional Russian-language expert specifically on Russian data, and all merged models are created with *only*

	General		Code		Multilingual		Math		
	MMLU	BBH	MBPP	HE	Flores(S)	Flores(T)	GSM8K	MATH	Avg.
Seed Hub	**35.8**	36.9	**32.2**	**22.0**	**30.9**	**36.2**	**20.2**	**10.6**	**28.1**
Math Hub	33.9	**37.8**	30.7	20.1	29.8	36.0	15.6	5.73	26.2

Table 3: **Comparison of using the seed model as the hub versus an expert.** We ablate BTS with a variant where we use the Math expert model as the hub. Using the seed model as the hub significantly outperforms using an expert model as the hub across most tasks.

		Flores		Ru-
	GSM8K	En/Ru	Ru/En	MGSM
Dense models				
Seed Model	10.5	22.8	32.8	12.8
Math Expert	*__20.5__	10.2	28.9	10.8
Russian Expert	9.48	*__32.3__	__34.6__	9.60
Seed DM	12.6	24.8	32.8	__14.0__
Expert upcycling				
BTX Sample	15.6	29.9	34.3	17.6
BTX Soft	17.6	30.6	34.5	17.6
BAM	__19.3__	30.9	34.5	*__18.4__
Expert merging				
Model Soup	17.5	14.7	32.3	13.2
BTM	20.5	*__32.3__	34.6	9.60
Expert Routing	9.48	*__32.3__	34.6	9.60
BAM Adapters	15.2	31.0	34.3	15.6
BTS	13.3	31.9	*__34.7__	16.0

Table 4: **Cross capability performance.** We evaluate the seed model, Russian-language, and Math experts on the Russian subset of MGSM. We compare their performance with expert merging and expert upcyling baselines trained with small amounts of in-domain data on Russian math. We also continued pretraining the strongest dense model, the seed model, on the same in-domain data, referred to as Seed Model (DM).

the Russian and Math experts[6]. We make these choices in order to study cross capability emergence in a controlled setting:

- **Reducing cross capability expert contamination:** Our coding data contained significant portions of non-English natural language, affecting the Code expert's ability in multilingual reasoning tasks, so we remove this model from this mix [Blevins and Zettlemoyer, 2022]. We further remove the seed model which contains both multilingual and math data.
- **Prevalance of cross capability training and evaluation data:** We limit our study to languages in which we have cross capability data to both train and evaluate the models on — for this reason, we focused on Russian and Math.

During the expert merging or upcycling training phase, we train on 2B tokens of Russian math data extracted from web data using a combination of language identification (LID) and math classifiers. We found this additional cross capability in-domain training data was essential for any variants to achieve strong performance (see experiments in §D.1).

We introduce an additional baseline via continued pretraining the strongest dense model, the seed model, in a data-matched manner on the Russian math data. This is to evaluate the impact of training on in-domain data without increasing the overall model capacity. Additional details of the experimental set-up are provided in §D.1. All models are evaluated on the Russian subset of MGSM (8-shot; Shi et al. 2022), which are Russian translations of examples from GSM8K [Cobbe et al., 2021].

Cross capabilities results Results are reported in Table 4. Notably, BTS can effectively leverage both experts to excel at a new task, surpassing the data-matched seed model baseline, even though the experts themselves remain unchanged: by adding connections between them, the resulting model exceeds the sum of its individual parts. Among all expert-merging baselines, BTS achieves the best cross capability performance. BTX and BAM variants also show strong performance, outperforming BTS, likely due to their significantly greater training capacity on in-domain data.

4.2 BTS architecture design

Below, we ablate the impact of choosing the seed model as the hub, as well as the importance of the

[6]Note that when merging only two experts, there is no notion of "hub" model: the stitch layers alternate between merging Russian-into-Math and Math-into-Russian.

alternating stitch layer architecture. For additional ablations on the impact of varying the number of stitch layers, see §D.3.

Impact of hub model selection We default to using the seed model as the hub in BTS, since all experts are initialized from it, making its representations more aligned with the experts than the experts are with each other. We hypothesized that allows a more effective merging of representations via the BTS stitch layers. To test this, we use an expert model as the hub instead. We select the Math expert for this experiment, as it has the best average performance among all expert models, and treat the seed model as a spoke. As shown in Table 3, using the seed model as the hub consistently yields better downstream performance.

Importance of the alternating stitch layer architecture BTS alternates between Experts-into-Hub and Hub-into-Experts stitch layers. We ablate this design by comparing to a non-alternating variant with only all Experts-into-Hub layers. As shown in Table 5, the alternating design significantly improves cross capability performance. Both variants performe similarly on generalist tasks (see Table 8). These results demonstrate that an alternating architecture is essential for achieving cross capability performance while maintaining strong generalist performance.

	GSM8K	Flores En/Ru	Flores Ru/En	Ru-MGSM
BTS Alternate	13.3	31.9	34.7	**16.0**
Non-Alternate	**15.2**	**32.0**	**35.0**	11.6

Table 5: **Comparison of alternating and non-alternating BTS variants on cross capabilities tasks** with additional in-domain Russian math training data.

4.3 Interpreting the BTS stitch layers

The gate values of the Experts-into-Hub stitch layer determine the weight of each expert in the combined representation. Intuitively, the higher the expert or seed model's gate values, the more important this model is for the task. We inspect these values to get insight into the model's decision-making progress on various tasks.

Visualizing gate values on expert specialty tasks Figure 2 visualizes how the gate values of the last stitch layer, an Experts-into-Hub stitch layer, vary when generating a sequence during inference on various expert specialty tasks. The first row plots the gate values for prompt tokens, while the second row plots the gate values for the generated tokens. Each column corresponds to a different prompt, sampled from the corresponding benchmark task.

This visualization shows that the gate values align closely with the task requirements. The specialized expert associated with the task typically dominates the gate values, while effectively mixing representations from different models over the course of the sequence. For the math task GSM8K, the math expert has the highest gate values throughout the generation, while the other models' gate values are near zero. For the language translation task Flores, the multilingual expert and the seed model dominate, with each model being relied on more heavily at different parts of the prompt or generation. For the coding task HumanEval, the coding expert and the seed model dominates.

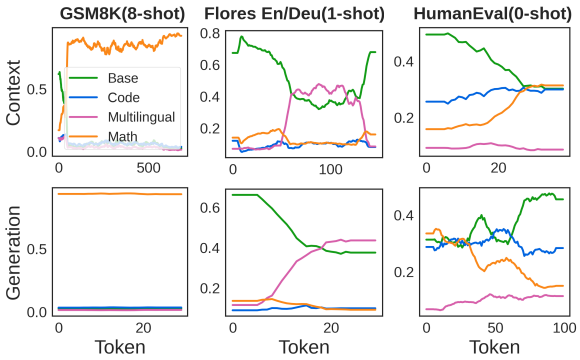

Figure 2: **Visualization of BTS gate values in the final stitch layer during inference.** For each task, the corresponding specialized expert typically has the highest gate values, showing that BTS learns to rely on the relevant experts.

Visualizing gate value transitions on context-switching tasks Unlike merge methods which make sequence-level choices about which expert to use, BTS can effectively context switch over the course of the sequence, seamlessly transitioning between different tasks. Figure 3 illustrates the gate values of BTS's final stitch layer when processing context-switching prompts. These prompts are constructed by concatenating examples from Flores (3-shot), GSM8K (2-shot), and TriviaQA [2-shot; Joshi et al., 2017], in that order, with vertical dotted lines indicating where a new task begins. Each column corresponds to a different context-switching prompt, created from distinct sampled inputs. In both examples, BTS demonstrates its ability to dy-

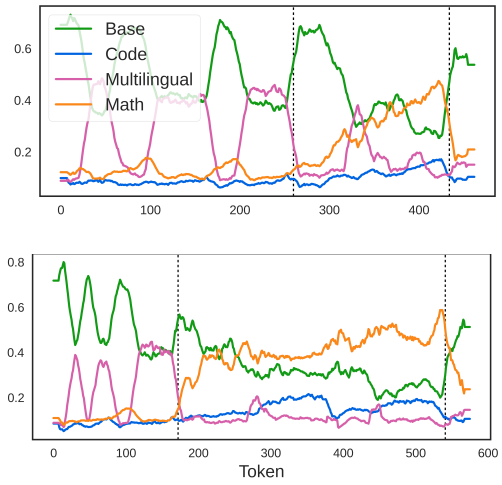

Figure 3: **Visualization of the gate values of BTS's final stitch layer for context-switching sequences at inference time.** These sequences are constructed by concatenating question-answer examples from Flores (3-shot), GSM8K (2-shot), and TriviaQA (2-shot), in that order, with dotted lines indicating task transitions. Each plot corresponds to a different randomly sampled prompt.

namically adjust expert utilization. During the Flores prompt, the seed model and multilingual expert are predominantly active. During the GSM8K prompt, the math expert takes over, and finally, the seed model is most utilized for the TriviaQA prompt. This highlights BTS's capability to correctly activate the relevant experts for each task, even when transitioning between diverse contexts.

5 Related work

Weights merging Previous works have demonstrated that linearly interpolating the weights of multiple expert models with the same architecture can produce a more effective model. Model Souping [Wortsman et al., 2022] achieves this by uniformly averaging model weights, whereas methods like BTM [Li et al., 2022], C-BTM [Gururangan et al., 2023], and SMEAR [Muqeeth et al., 2023] dynamically compute the weighting of each expert's model parameters based on the given prompt.

Output ensembles In addition to averaging model weights, several works have explored averaging model outputs to create ensembles of expert models [Li et al., 2022, Gururangan et al., 2023]. However, these approaches are limited in expressivity.

Routing among dense models Another approach involves routing the entire input and generation to a single model selected from multiple expert LLMs [Filippova et al., 2024, Ong et al., 2024]. However, these methods are limited when the input requires expertise from multiple domains or involves context-switching between different tasks.

Mixture-of-Experts upcycling Several works have explored using pretrained dense models to initialize MoEs [Komatsuzaki et al., 2022, Sukhbaatar et al., 2024, Zhang et al., 2024]. These approaches copy each expert model's parameters to initialize the corresponding experts in the MoE. For the MoE's non-expert parameters, they average the parameters of the pretrained experts. The router is initialized from scratch. Following initialization, the MoE undergoes a training phase where all model parameters are updated.

Adding connections between models Recent works have proposed adapting language models to new modalities by composing modality-specific models, e.g., Alayrac et al. [2022] propose adding cross-attention parameters to allow a language model to condition on visual inputs, and Liang et al. [2024] uses global self-attention to fuse models for different modalities. Perhaps most similar to our work, Bansal et al. [2024] extend this idea to domain-specific language models, and propose augmenting an "anchor" language model with a single domain-specific model through cross-attention.

6 Conclusion

We introduced BRANCH-TRAIN-STITCH (BTS) an efficient and flexible method for merging expert models into a stronger, unified, generalist model. BTS combines expert models by inserting novel "stitch" layers between expert language model layers, which are learned in a lightweight training step. In experiments, we find that this approach outperforms competitive baselines, yielding the strongest generalist model performance with only a small number of training parameters. In some settings, BTS is shown to even outperform the expert models in their specialized domains. We further demonstrate that a BTS model can demonstrate new skills at the intersection of expert domains and motivate this architecture with extensive ablations and analysis. We hope this work furthers research into efficient and flexible methods for creating generalist large language models by re-using modular, independently-trained experts.

Limitations

Results on a parameter-matched data-matched dense model Scaling up dense models, such as training a parameter-matched data-matched dense 11B model, introduces infrastructure challenges and datamix trade-offs. BTS and other " Branch-Train" methods [Li et al., 2022, Sukhbaatar et al., 2024, Zhang et al., 2024] are explicitly designed to alleviate these challenges by asynchronously training smaller experts in parallel without massive synchronization overhead. And in many cases, the expert models already exist, making the method even more practical. Thus, "Branch-Train" methods, including BTS, are posed as *alternative* methods to scaling dense models. Although a parameter-matched data-matched baseline model would be an interesting comparison, we focused our limited compute on the main comparison and ablation showcasing the validity of our approach. For similar reasons, other "Branch-Train" methods [Zhang et al., 2024, Sukhbaatar et al., 2024, Li et al., 2022] also did not include such a baseline in their papers.

Active parameter count of BTS While BTS has the highest active parameter count, it uses far fewer training parameters than expert upcycling methods. As BTS inserts a stitch layer every five layers, we compare BTS to the strongest baseline, BAM, in Appendix E, finding similar inference efficiency per five layers, but significantly lower training cost during merging. One future direction is improving BTS to not only have fewer training parameters, but also fewer active parameters. One way of achieving this is adding a router layer after the input embedding layer, which dynamically selects a subset of relevant experts based on the input sequence. Then, only a subset of experts would be stitched for each input sequence. This modification would effectively introduce sparsity, reducing the number of active parameters and mitigating potential redundancy. Exploring this approach would be an interesting direction for future research.

Results beyond Pre-Training Our work focuses on the pretraining stage, so all models, including baselines, are trained on pretraining data only. This is consistent with prior works in the "Branch-Train" family [Sukhbaatar et al., 2024, Zhang et al., 2024]: while our approach can be directly applied in instruction finetuning or reinforcement learning finetuning procedures, we leave that for future work as we focused on the pretraining stage in this paper.

Acknowledgments

We extend our thanks to Sachin Mehta, Ruslan Mavlyutov, Alexei Baevski, Onur Çelebi, Niladri Chatterji and Mik Vyatskov for their assistance with the training infrastructure. We'd like to also thank Ravikumar Rajendran, Siarhei Loginau and Tim Wang for their assistance with the compute infrastructure. We thank Vedanuj Goswami for his assistance in training the models used in the experiments. We are also grateful to Anirudh Goyal, Akhil Mathur, and Wenhan Xiong for providing helpful pointers regarding data mixtures. We appreciate Ellen Tan, Rocky Wang, Todor Mihaylov, Xuchao Jia, and Mihir Sanjay Kale for helping with using the data preparation pipeline. We also thank Florian Laplantif, Norman Cheng, Lovish Madaan, and Kunal Bhalla for their support with the evaluation infrastructure. QZ thanks Chris Lu for discussions on the project and helpful suggestions during the rebuttals. Finally, we thank Melanie Kambadur for her general assistance and support of this project.

References

Armen Aghajanyan, Akshat Shrivastava, Anchit Gupta, Naman Goyal, Luke Zettlemoyer, and Sonal Gupta. 2021. Better fine-tuning by reducing representational collapse. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob L. Menick, Sebastian Borgeaud, and 8 others. 2022. Flamingo: a visual language model for few-shot learning. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program synthesis with large language models. *Preprint*, arXiv:2108.07732.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2023. Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631*.

Rachit Bansal, Bidisha Samanta, Siddharth Dalmia, Nitish Gupta, Shikhar Vashishth, Sriram Ganapathy, Abhishek Bapna, Prateek Jain, and Partha Talukdar. 2024. Llm augmented llms: Expanding capabilities through composition. *arXiv preprint arXiv:2401.02412*.

Terra Blevins and Luke Zettlemoyer. 2022. Language contamination helps explain the cross-lingual capabilities of english pretrained models. *arXiv preprint arXiv:2204.08110*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Ben Cottier, Robi Rahman, Loredana Fattorini, Nestor Maslej, Tamay Besiroglu, and David Owen. 2024. The rising costs of training frontier ai models. *arXiv preprint arXiv:2405.21015*.

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

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Jared Fernandez, Luca Wehrstedt, Leonid Shamis, Mostafa Elhoushi, Kalyan Saladi, Yonatan Bisk, Emma Strubell, and Jacob Kahn. 2024. Hardware scaling trends and diminishing returns in large-scale distributed training. *arXiv preprint arXiv:2411.13055*.

Anastasiia Filippova, Angelos Katharopoulos, David Grangier, and Ronan Collobert. 2024. No need to talk: Asynchronous mixture of language models. *arXiv preprint arXiv:2410.03529*.

Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W Mahoney, and Kurt Keutzer. 2024. Ai and memory wall. *IEEE Micro*.

Naman Goyal, Cynthia Gao, Vishrav Chaudhary, Peng-Jen Chen, Guillaume Wenzek, Da Ju, Sanjana Krishnan, MarcâĂŹAurelio Ranzato, Francisco Guzmán, and Angela Fan. 2022. The flores-101 evaluation benchmark for low-resource and multilingual machine translation. *Transactions of the Association for Computational Linguistics*, 10:522–538.

Suchin Gururangan, Margaret Li, Mike Lewis, Weijia Shi, Tim Althoff, Noah A Smith, and Luke Zettlemoyer. 2023. Scaling expert language models with unsupervised domain discovery. *arXiv preprint arXiv:2303.14177*.

Suchin Gururangan, Ana MarasoviÄĞ, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. *Preprint*, arXiv:2004.10964.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021a. Measuring massive multitask language understanding. *Preprint*, arXiv:2009.03300.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

Quzhe Huang, Mingxu Tao, Chen Zhang, Zhenwei An, Cong Jiang, Zhibin Chen, Zirui Wu, and Yansong Feng. 2023. Lawyer llama technical report. *arXiv preprint arXiv:2305.15062*.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

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

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. *arXiv preprint arXiv:2001.08361*.

Apostolos Kokolis, Michael Kuchnik, John Hoffman, Adithya Kumar, Parth Malani, Faye Ma, Zachary DeVito, Shubho Sengupta, Kalyan Saladi, and Carole-Jean Wu. 2025. Revisiting reliability in large-scale machine learning research clusters. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1259–1274. IEEE.

Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp, Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie, Yi Tay, Mostafa Dehghani, and Neil Houlsby. 2022. Sparse upcycling: Training mixture-of-experts from dense checkpoints. *arXiv preprint arXiv:2212.05055*.

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. *arXiv preprint arXiv:2208.03306*.

Weixin Liang, Lili Yu, Liang Luo, Srinivasan Iyer, Ning Dong, Chunting Zhou, Gargi Ghosh, Mike Lewis, Wen-tau Yih, Luke Zettlemoyer, and 1 others. 2024. Mixture-of-transformers: A sparse and scalable architecture for multi-modal foundation models. *arXiv preprint arXiv:2411.04996*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Michael McCloskey and Neal J. Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109–165. Academic Press.

Mohammed Muqeeth, Haokun Liu, and Colin Raffel. 2023. Soft merging of experts with adaptive routing. *arXiv preprint arXiv:2306.03745*.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, and 1 others. 2021. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pages 1–15.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. 2024. Routellm: Learning to route llms with preference data. *arXiv preprint arXiv:2406.18665*.

Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. 2023. Openwebmath: An open dataset of high-quality mathematical web text. *arXiv preprint arXiv:2310.06786*.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, and 6 others. 2023. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, and 1 others. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, and 1 others. 2022. Language models are multilingual chain-of-thought reasoners. *arXiv preprint arXiv:2210.03057*.

Sainbayar Sukhbaatar, Olga Golovneva, Vasu Sharma, Hu Xu, Xi Victoria Lin, Baptiste Rozière, Jacob Kahn, Daniel Li, Wen-tau Yih, Jason Weston, and 1 others. 2024. Branch-train-mix: Mixing expert llms into a mixture-of-experts llm. *arXiv preprint arXiv:2403.07816*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, , and Jason Wei. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

Yi-Lin Tuan, Xilun Chen, Eric Michael Smith, Louis Martin, Soumya Batra, Asli Celikyilmaz, William Yang Wang, and Daniel M. Bikel. 2024. Towards safety and helpfulness balanced responses via controllable large language models. *CoRR*, abs/2404.01295.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and 1 others. 2022. Model

soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International conference on machine learning*, pages 23965–23998. PMLR.

Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy Liang, Quoc V. Le, Tengyu Ma, and Adams Wei Yu. 2023. Doremi: Optimizing data mixtures speeds up language model pretraining. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Jiasheng Ye, Peiju Liu, Tianxiang Sun, Yunhua Zhou, Jun Zhan, and Xipeng Qiu. 2024. Data mixing laws: Optimizing data mixtures by predicting language modeling performance. *CoRR*, abs/2403.16952.

Qizhen Zhang, Nikolas Gritsch, Dwaraknath Gnaneshwar, Simon Guo, David Cairuz, Bharat Venkitesh, Jakob Foerster, Phil Blunsom, Sebastian Ruder, Ahmet Ustun, and 1 others. 2024. Bam! just like that: Simple and efficient parameter upcycling for mixture of experts. *arXiv preprint arXiv:2408.08274*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, and 1 others. 2022a. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Xiaofeng Zhang, Yikang Shen, Zeyu Huang, Jie Zhou, Wenge Rong, and Zhang Xiong. 2022b. Mixture of attention heads: Selecting attention heads per token. *arXiv preprint arXiv:2210.05144*.

Ming Zhong, Aston Zhang, Xuewei Wang, Rui Hou, Wenhan Xiong, Chenguang Zhu, Zhengxing Chen, Liang Tan, Chloe Bi, Mike Lewis, and 1 others. 2024. Law of the weakest link: Cross capabilities of large language models. *arXiv preprint arXiv:2409.19951*.

A Architecture details for seed and expert models

Layers	20
Model Dimension	3072
FFN Dimension	12288
Attention Heads	24
Key/Value Heads	1
Activation Function	SwiGLU
Vocabulary Size	128,000
Positional Embeddings	RoPE ($\theta = 500{,}000$)

Table 6: **Architecture details** for the 2.7B parameter seed model and expert models.

See Table 6 for the architecture details for the 2.7B parameter seed model and expert models.

B Language model architecture background

Transformer The typical architecture of large language models (LLMs) is built by stacking multiple Transformer blocks [Vaswani et al., 2017]. Each Transformer block consists of a Multi-Headed Attention module, commonly referred to as the attention layer, followed by a residual connection and a feed-forward neural network (FFN).

Mixture-of-Experts The Mixture of Experts [MoE; Shazeer et al., 2017] model replaces the FFN in the Transformer by an MoE layer. An MoE layer consists of a linear router and a set of N FFN experts, denoted as $\{\text{FFN}_i(x)\}_{i=1}^{N}$. The router produces router logits, which we denote as $p(x)$ after normalization, for the input representation x. $p_i(x)$ is the gating value for the i-th FFN expert, FFN_i. The router assigns the input representation x to a subset of experts, $\mathcal{T}$, with the highest gate values. The final output of the MoE layer is the weighted sum of the selected experts' outputs, weighted by their gating values:

$$y_{\text{MoE}} = \sum_{i \in \mathcal{T}} p_i(x)\text{FFN}_i(x). \tag{4}$$

Mixture-of-Attention Mixture of Attention [MoA; Zhang et al., 2022b] extends MoE by also replacing the attention layer in Transformers with an MoA layer. Similar to the MoE layer, an MoA layer comprises of a set of N attention experts (denoted as $\{\text{Attention}_j(x)\}_{j=1}^{N}$), a linear router that outputs router logits. We denote the normalized router logits as $q(x)$. The MoA layer's final output is a gating-value weighted sum of the computations from the selected attention experts $\mathcal{M}$:

$$y_{\text{MoA}} = \sum_{i \in \mathcal{M}} q_i(x)\text{Attention}_i(x). \tag{5}$$

C Additional model training details

Seed model We pretrain a 2.7B parameter language model, following the same text recipe used in Llama 3 [Dubey et al., 2024]. See Table 6 for architecture details. We employ a learning rate schedule that warms up from 0 to 4e-4 over 2000 steps, then undergoes a cosine decay to 1% of the peak learning rate. The seed model is trained for 2.2 million steps on 15T tokens.

Expert models We create three copies of the seed model, each of which is continually trained for 96k training steps over a 200B token specialized data mixture to produce expert models for code, math, and multilingual tasks. During the continued pretraining phase, we use a batch size of 2M tokens and a learning rate of 5e-6, followed immediately by a cosine decay schedule that reduces the learning rate to 1% of its initial value. This learning rate is derived by annealing from the final learning rate used at the end of seed model pretraining, adjusted to account for the reduced batch size in this continued pretraining phase. We adopted this learning rate strategy as it yielded the most stable learning during the continued pretraining phase

BTS model We use four stitch layers to combine the seed model together with the three expert models. The four stitch layers are inserted after every five layers in the seed and expert models. We refer to the resulting model as the BTS model. As described in §2, the four stitch layers alternate between a *Hub-into-Expert stitch layer* and *Experts-into-Hub stitch layer*. Upon initialization, the BTS model is further trained for 15B tokens over 7000 steps using a batch size of 2M tokens. The optimization objective is to minimize the cross entropy loss from the hub model's output. The learning rate schedule warms up from 0 to 5e-6 over 2000 steps, then undergoes a cosine decay to 1% of the peak learning rate. Note that during the BTS training phase, only the stitch layers are updated while all

		Flores		
	GSM8K	**En/Ru**	**Ru/En**	**Ru-MGSM**
Dense models				
Seed Model	10.5	22.8	32.8	**12.8**
Math Expert	*__20.5__	10.2	28.9	10.8
Russian Expert	9.48	**32.3**	*__34.6__	9.60
Expert upcycling				
BTX Sample	18.3	30.4	34.0	10.0
BTX Soft	18.0	30.0	33.9	**12.4**
BAM	*__20.5__	**30.6**	**34.5**	10.8
Expert merging				
Model Soup	17.5	14.7	32.3	*__13.2__
BTM	20.5	32.3	34.6	9.60
Expert Routing	9.48	**32.3**	*__34.6__	9.60
BAM Adapter	18.1	30.9	34.1	12.8
BTS	**19.0**	31.6	33.0	10.0

Table 7: **Cross capability performance of merged models without in-domain data.** We evaluate the seed model, Russian-language, and Math experts on Russian MGSM [Shi et al., 2022] and compare performance with merged and upcycled models. We do not use any in-domain training data during the merging or upcycling training process. The results indicate that a small amount of cross capability data is necessary for merged or upcycled models to effectively learn cross capabilities.

	General		**Code**		**Multilingual**		**Math**		
	MMLU	**BBH**	**MBPP**	**HE**	**Flores(S)**	**Flores(T)**	**GSM8K**	**MATH**	**Avg.**
BTS Alternate	35.8	36.9	32.2	22.0	30.9	36.2	**20.2**	10.6	28.1
Non Alternate	**36.1**	**37.9**	**32.4**	**22.6**	**31.4**	**36.4**	19.9	**10.8**	**28.4**

Table 8: **Comparison of alternating and non-alternating BTS variants on generalist tasks.** Both variants achieves similar performance on most domains, with the non-alternating variant slightly outperforming the alternating variant on average.

the parameters of the seed model and the expert models are frozen.

Expert Routing We train a linear router $\in \mathbb{R}^{\text{dim} \times n}$ that routes to either the seed model or one of the expert models. The router's training objective is a classification cross-entropy loss where the target is the model with the smallest next-token prediction loss for the input. Given a prompt, the router decides on the model and routes all subsequent tokens to the same model. During training, the routing decision is made based on the average embedding of the first t tokens in the input, where t is randomly sampled between 32 and 256. During inference, the routing decision is made based on the average embedding of the entire prompt. We train the linear router with a constant learning rate of 5e-4 and batch size of 1M. The model is trained for 1 billion tokens only, as we did not see an improvement in downstream metrics or training loss with further training.

D Additional Ablations

D.1 Cross capabilities: additional details and experiments

D.1.1 Further experiment details

Russian expert model training To enhance cross capabilities in mathematical skills for Russian, we train an additional expert specifically on Russian data. The expert training setup follows the same procedure outlined in §C. For training data, we utilize the Russian subset of the multilingual dataset previously used for the multilingual expert, as described in §3.2.

	General		Code		Multilingual		Math		
	MMLU	**BBH**	**MBPP**	**HE**	**Flores(S)**	**Flores(T)**	**GSM8K**	**MATH**	**Avg.**
10 Layers	**36.1**	37.8	31.8	22.0	31.2	**36.5**	19.1	10.4	28.1
4 Layers	35.8	36.9	**32.2**	**22.0**	**33.9**	36.2	**20.2**	**10.6**	**28.1**
1 Layer	34.9	**37.8**	29.6	19.5	30.8	35.9	17.7	9.9	27.0

Table 9: **Ablations on the effect of varying number of stitch layers on downstream task performance.** The first two rows are configurations with 10 and 4 stitch layers distributed uniformly throughout the seed and expert models. The third row is a configuration with a single Experts-into-Hub stitch layer placed after the last dense model layers. The 10 and 4 layers configuration performs similarly, but the single-layer configuration lags behind model performance significantly.

Merged models training As an additional baseline, we continually pretrain the strongest dense model in russian MGSm, the seed model, on the same Russian math pretraining data used for the merged models. All experiments share the following training configuration:

- Learning rate schedule: we warm up from 0 to $5e-6$ over 1000 steps, then undergoes a cosine decay to 10% of the peak learning rate. The merged models are trained for a total of 2000 steps. One exception is the expert routing model is trained for 1000 steps in total with a constant learning rate of $5e-4$. This was chosen upon tuning the hyperparameters.
- Batch size: we use a batch size of 1M tokens.
- Token count: All models were trained on 2B tokens of Russian math over 2000 training steps. The exception is expert routing, which only trained on 1B tokens over 1000 steps, as we did not see performance improvement with further training.

D.1.2 Results on merging and upcycling models without in-domain data

In Table 7, we show results on merging and upcycling models without in-domain data. The merging phase is instead trained on a data mixture composed of 50% of math expert and 50% of Russian expert's continue pretraining data mixture.

We observe that despite being trained with more tokens during the merging phase, all baseline methods does not significantly outperform the seed model on the cross capability task Russian MGSM. This indicates that in-distribution data is essential.

D.2 Ablations on the alternating architecture

The BTS architecture involves alternating between the Experts-into-Hub stitch layer and the Hub-into-Experts stitch layer. We ablate the impact of adopting this alternating architecture as opposed to utilizing all Experts-into-Hub layers. As shown in Table 5, the alternating architecture (first row) yields significantly better cross capability performance compared to using only homogeneous Experts-into-Hub stitch layers (second row). However, both the alternating and non-alternating architectures achieve comparable performance on generalist tasks, as shown in Table 8. These results demonstrate that an alternating architecture is essential for achieving cross capability performance while maintaining strong generalist performance.

D.3 Ablations on the impact of the number of stitch layers

We measure the impact of varying the number of stitch layers on model performance, as shown in Table 9. The first two rows present configurations with 10 and 4 stitch layers, respectively, distributed uniformly throughout the seed and expert models. In the third row, we investigate a configuration with a single Experts-into-Hub stitch layer placed after the final language model layers.

Our ablations show that a single stitch layer is insufficient for learning to effectively merge capabilities, as its performance lags significantly behind configurations with 4 or 10 layers. This also demonstrates that the BTS models with more than one stitch layer combine models in a more expressive way than than simply combining output representations. The 4 and 10 layer configurations perform similarly, however, we note that this may be due to under-training of the 10 layer variant as all models are trained on the same number of tokens.

Model	MMLU	BBH	MBPP	HE	Flores(S)	Flores(T)	GSM8K	MATH	Average
BTS	**35.8**	**36.9**	**32.2**	22.0	**30.9**	**36.2**	**20.2**	**10.6**	**28.1**
BTS (no upweight)	30.7	36.2	31.4	**23.8**	30.7	36.2	13.0	7.3	26.2
BAM	**35.2**	**37.1**	29.8	22.6	**31.0**	**36.1**	**20.3**	**10.1**	**27.8**
BAM (no upweight)	28.9	36.4	**31.4**	**23.8**	30.8	35.9	13.2	6.2	25.8
BAM Adapters	**34.1**	**37.0**	28.8	**22.6**	**31.0**	**36.1**	**18.8**	**10.0**	**27.3**
BAM Adapters (no upweight)	29.0	36.3	30.8	22.0	30.7	36.0	12.7	6.2	25.5

Table 10: **We compare BTS and its strongest baselines trained with and without upweighting.** Without upweighting, BTS and baselines almost always do worse on every benchmark, as they do not see enough expert data to integrate the expertise of different experts.

D.4 Importance of Data Upweighting

We constructed the data mixture for the BTS training phase by starting from the seed model data mix and up-weighting the proportion of data from each expert's domain to 15%. This upweighting ensures that the BTS stitching weights are sufficiently exposed to domain-specific data, enabling them to learn how to effectively leverage each expert.

To highlight the importance of expert domain upweighting, we compare BTS and its strongest baselines trained with and without upweighting. In Table 10, we show BTS and baselines almost always do worse on every benchmark without upweighting, as they do not see enough expert data to integrate the expertise of different experts.

Operation	Type	BAM	BTS
Attention Router	Params	$5n_{\text{experts}}d_{\text{model}}$	–
	FLOPs	$10n_{\text{experts}}d_{\text{model}}$	–
Attention: QKV	Params	$15n_{\text{experts}}d_{\text{model}}d_{\text{attn}}$	$15n_{\text{experts}}d_{\text{model}}d_{\text{attn}}$
	FLOPs	$30n_{\text{experts}}d^2_{\text{model}}$	$30n_{\text{experts}}d^2_{\text{model}}$
Attention: Mask	Params	–	–
	FLOPs	$10n_{\text{experts}}n_{\text{ctx}}d_{\text{model}}$	$10n_{\text{experts}}n_{\text{ctx}}d_{\text{model}}$
Attention: Projection	Params	$5n_{\text{experts}}d_{\text{attn}}d_{\text{model}}$	$5n_{\text{experts}}d_{\text{attn}}d_{\text{model}}$
	FLOPs	$10n_{\text{experts}}d^2_{\text{model}}$	$10n_{\text{experts}}d^2_{\text{model}}$
FFN Router	Params	$5n_{\text{experts}}d_{\text{model}}$	–
	FLOPs	$10n_{\text{experts}}d_{\text{model}}$	–
FFN	Params	$7.5n_{\text{experts}}d_{\text{model}}d_{\text{ff}}$	$7.5n_{\text{experts}}d_{\text{model}}d_{\text{ff}}$
	FLOPs	$60d_{\text{model}}d_{\text{ff}}$	$60d_{\text{model}}d_{\text{ff}}$
BTS: Gate	Params	–	$n_{\text{experts}}d_{\text{model}}$
	FLOPs	–	$2n_{\text{experts}}d_{\text{model}}$
BTS: Expert Projections	Params	–	$d^2_{\text{model}}n_{\text{experts}}$
	FLOPs	–	$2n_{\text{experts}}d^2_{\text{model}}$

Table 11: We present the estimated *FLOPs per token* for every five layers, as BTS inserts one stitch layer every five layers in our experiments. We adopt the standard FLOPs counting methodology, excluding negligible operations such as non-linearities, biases, and layer normalization.

E Efficiency Analysis

We compare the efficiency between BTS and the most competitive generalist baseline from Table 2, BAM.

Table 11 and Table 12 present the estimated "FLOPs per token" for every five layers, as BTS inserts one stitch layer every five layers in our experiments. We adopt the standard FLOPs counting methodology from [Kaplan et al., 2020], where we exclude negligible operations such as non-linearities, biases, and layer normalization. The exact FLOPs count in Table 12 is calculated using the architecture details in Table 6.

	Inference FLOPs	Training Params
BAM	4,781,752,320	8.4 billion
BTS	4,857,028,608	264 million

Table 12: Based on the arithmetic breakdown from above, we obtain the inference FLOPs and training parameters using the architecture details in Table 6 in the Appendix. The first row shows the arithmetic breakdown for inference FLOPs per token for every five layers. The second row shows the number of training parameters during the training phase for the entire model.

We observe that BTS requries approximately the same inference compute as BAM, every five layers, with only a 1.6% increase in FLOPs. However, BTS uses 30× fewer trainable parameters, resulting in substantial training efficiency gains:

- **Memory savings**: BTS reduces optimizer state size by 30x. Since AdamW [Loshchilov and Hutter, 2017] stores two float32 values per parameter, BTS saves more than 65 GB of memory savings compared to BAM.
- **Training efficiency**: BTS computes 30× fewer gradients. As the backward pass typically costs twice the FLOPs of the forward pass [Kaplan et al., 2020, Dao, 2023], this yields significant reductions in training compute.

F Pseudo Code for BTS

```
def StitchLayer(xs, merge_into_hub=True):
    """
    xs: dense models' outputs
    """
    x_hub = x[0]
    x_experts = x[1:]

    g = w_gate(x_hub) # [bs, seq_len, dim, 1+n_experts]

    # Experts-into-Hub Layer
    if merge_into_hub:
        g = dropout(g).softmax(dim=-1)
        h_experts = [
            w_proj[i](x_experts[i]) for i in range(n_experts)
        ]
        h_hub = (g * stack([h] + h_experts, dim=-1)).sum(-1)


    # Hub-into-Expert Layer
    else:
        g = dropout(g).sigmoid()
        h_experts = = [
                (1 - g[..., i + 1]) * x_experts[i]
                + (g[..., i + 1] * w_proj[i](x_hub))
                for i in range(n_experts)
            ]
        h_hub = x_hub

    return stack([h_hub] + h_experts, dim=-1)

def BTSBlock(xs, ith_layer, BTS_freq):

    x_hub = hub_model_layer(xs[0])
    x_experts = [expert_model_layer[i](xs[i+1]) for i in n_experts]
    xs = stack([x_hub] + x_experts, dim=-1)

    if ith_layer % BTS_freq = = 0:
        # Alternate between two types of stitch layers
        hs = StitchLayer(xs, merge_into_hub=(ith_layer//BTS_freq)%2)
        return hs

    return xs
```