LEGO: Language Model Building Blocks

Anonymous ACL submission

Abstract

Large language models (LLMs) are essential in natural language processing (NLP) but are costly in fine-tuning and inference, and involve invasive data collection. Task-specific small language models (SLMs) offer a cheaper alternative but lack robustness and generalization. This paper proposes a novel technique to combine SLMs and construct a robust, general LLM. Using state-of-the-art LLM pruning strategies, we create task- and user-specific SLM building blocks that are efficient for fine-tuning and inference while also preserving user data privacy. Utilizing Federated Learning and a novel aggregation scheme, we can compile an LLM from distributed SLMs, maintaining robustness without high costs and preserving user data privacy.

1 Introduction

Large Language Models (LLMs) represent a significant advance in Natural Language Processing (NLP) with their remarkable ability to generalize across queries and tasks. These models are typically fine-tuned using large, diverse datasets derived from high-quality instruction data (Gupta et al., 2022).

LLMs are not, however, a one-size-fits-all solution. Running LLMs on small devices like IoT devices or smartphones is not possible due to their resource limitations. Downstream LLM applications that value privacy, such as personal conversational AI, become untenable due to data privacy concerns, as user data must stay on personal devices or private networks and cannot be shared globally. These constraints, created by private user data, apply to both fine-tuning and inference.

LLMs are traditionally fine-tuned in a centralized manner, where data is aggregated from raw user interactions and shared globally to fine-tune a single global model. In contrast, Federated Learning (FL) is a collaborative learning approach that allows client models to learn from users while preserving their privacy (McMahan et al., 2017). FL utilizes distributed fine-tuning with localized client models trained on localized user interactions, resulting in a global model created by aggregating client model weights. While FL preserves data privacy and addresses the complexity of fine-tuning, it does not resolve the high cost of inference with LLMs.

Small Language Models (SLMs) address the high cost of inference as well as fine-tuning, allowing for a greater range of client devices. While SLMs are more efficient, the cheaper performance comes at the expense of robustness and generalization across broad tasks, conversational interactions, and advanced LLM capabilities. Furthermore, SLMs are not typically designed to be composable, constraining FL architecture to an either-or choice: choose SLMs at the cost of robustness, or choose the original LLMs that limit their utility due to size and complexity.

For resource-constrained scenarios like chatbots on small devices, there is a critical need for computationally efficient (fine-tuning and inference), robust, general, and private methods that facilitate different sizes and architectures of models depending on the computational resources of the device.

To enhance client flexibility in distributed conversational AI systems, we introduce **L**anguage **MO**del Buildin**G** Bl**O**cks (LEGO), a model-agnostic technique for integrating small language models (SLMs) with diverse heterogeneous architectures. LEGO enables efficient fine-tuning and inference, preserves privacy, optimizes performance across varied resource constraints, and aids in developing robust and generalizable large language models (LLMs). Our approach utilizes an SLM-based federated learning system where each SLM is derived from an LLM, allowing them to be combined to reconstruct the original LLM. By treating SLMs as building blocks, LEGO effectively assembles them into a cohesive LLM.

Through the use of LEGO, we demonstrate a flexible FL system that broadens the range of possible client devices by enabling different sized models for different sized devices. Through numerous experiments, we display that when using LEGO, smaller models are better learners and therefore yield more robust models. We also demonstrate that one client learning from local data translates to all clients having learned, and that SLMs can be treated as composable entities that can be combined to form an LLM.

With the proposed LEGO approach, the major contributions of this work include

- A method to compose SLMs together to yield a robust and generalizable LLM
- A privacy-preserving FL architecture to serve these composable client-side heterogeneous SLMs
- A method to optimize client-side SLMs against heterogeneous resource budgets for efficient fine-tuning and inference

The rest of this paper is organized as the following: Section 2 gives background information. Section 3 details the methodology behind the LEGO approach and its components. Section 4 covers the experiments we performed to validate LEGO and houses their results. Section 5 discusses the related work. Section 6 concludes the paper and Section 7 lists our study's limitations.

2 Background

2.1 Model Compression

In recent years, Knowledge Distillation (KD) has become widely used in NLP to compress LLMs (Hinton et al., 2015). Previous works have demonstrated that knowledge can be effectively distilled from LLMs to create task-specific small language models (SLMs). These KD-produced small models perform better than full-sized LLMs when fine-tuned on specific tasks, but do so at the cost of general robustness (Xu et al., 2024).

One alternative to KD is pruning, a method that involves the selective omission of model parameters with minimal contributions to the learning process. Primitive pruning techniques have proven successful, enhancing the cost-effectiveness of large pre-trained models (Xia et al., 2023).

Recently, more nuanced pruning approaches have been discussed in the literature, improving over more traditional methods like magnitude pruning. Specifically, two state-of-the-art pruning methods are widely discussed in the literature—SparseGPT (Frantar and Alistarh, 2023) and Wanda (Sun et al., 2023). Whereas traditional magnitude pruning operates by pruning weights with the largest magnitude, these pruning techniques instead track weight activations, and prune weights with the lowest amount of activation.

SparseGPT creates and solves a layer-wise reconstruction problem to determine the weight activations, while Wanda instead takes the product of a weight's magnitude and the norm of its associated input activations to determine what to prune.

Regardless, in the context of LLM compression, both these techniques present significant advantages over KD, as pruning is less computationally expensive. Whereas KD requires substantial post-training time for the distilled models, pruning can produce SLMs without these costs.

2.2 Federated Fine-Tuning

Federated Learning (FL) is a distributed training methodology that trains a model across multiple decentralized devices while allowing data to remain on user machines (McMahan et al., 2017). In Conventional FL, each client device has its own native model and trains it on user inputs. Instead of sharing this client data globally, the models instead share their own model weights, aggregating them with other client weights. This creates a global update that encodes knowledge gained from all model updates without compromising data privacy.

This same methodology can be applied to LLM fine-tuning. Instead of fine-tuning on globally shared user data, client models can fine-tune on local data and have their weights shared and aggregated. This approach eases many of the barriers to data collection compared to traditional centralized fine-tuning, as users can retain privacy over their instructions while contributing to the model.

Two fundamental assumptions are often made in both traditional FL and FL for fine-tuning. The first is that all data is i.i.d, meaning that not only do all clients have similar amounts of data, but that the the ratio of content within each are similar. The study of non-i.i.d data distributions in FL is often referred to as heterogeneous FL, with many strategies and techniques being proposed to offset the effects of data heterogeneity.

The second assumption is that all model architec-

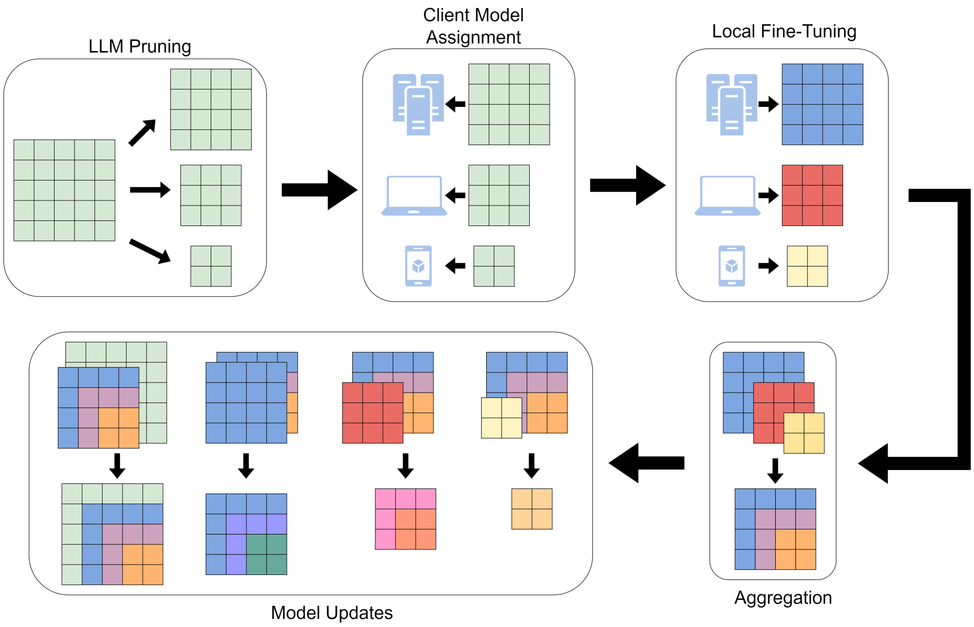

Figure 1: The LEGO workflow. An LLM is first pruned to create SLMs, then each SLM is assigned to a client. Each client then fine-tunes its SLM on its local data. After fine-tuning, the models are aggregated to create a global update. The global update is then applied to all the client SLMs as well as a global LLM. Eventually, after enough updates, a final global LLM is derived.

tures in FL systems are identical, allowing for the aggregation of model weights when creating global updates. Heterogeneity in model architecture therefore presents unique challenges in FL. Differing client model architectures impede the use of standard aggregation techniques like FedAvg due to varying parameter sizes.

Much like data-heterogeneous FL, many strategies have been proposed to offset the effect of model heterogeneity, allowing for model-agnostic FL. Previous work surrounding model-agnostic FL points towards using a proxy unlabeled public dataset to unify trained weights between different models (Huang et al., 2022). This approach allows the construction of a cross-correlation matrix to learn a generalizable representation under domain shift. However, due to the generality of LLMs, finding and using a large and diverse enough dataset to unify models distilled for diverse specific downstream tasks is impractical.

3 Methodology

Motivated by the need for efficient fine-tuning and inference for private, resource-constrained scenarios, we propose a model-agnostic FL system **L**anguage **MO**del Buildin**G** Bl**O**cks (LEGO). Much like stacking small building blocks together to create a larger structure, we stack small language models (SLMs) together to create a larger, more robust Large Language Model (LLM).

LEGO employs a two-step approach. First, we obtain SLMs of different sizes by pruning an LLM. We then deploy these SLMs in an FL environment, eventually aggregating them into an LLM. Figure 1 shows the LEGO workflow in greater detail. The SLMs produced by the pruning process are the local client models in the FL environment. We produce SLMs of different sizes and model architectures to better match the various computational budgets of client devices. We use a full-sized LLM as the global model, meaning that every client model is a sub-network of the global model.

To produce a fine-tuned LLM using the client SLMs, we begin with the process of federated fine-tuning. First, the selected client SLMs for each round are fine-tuned on their respective client's local data. They are then aggregated with each other, creating a global update. This global update is then applied to all client SLMs and the global LLM. We repeat this process for every round of FL, eventually forming a robust, fine-tuned LLM built from the updates supplied by the fine-tuned client SLMs.

For all studies and experiments, we impose the following conditions:

- All fine-tuning is done using LoRA (Hu et al., 2021), resulting in a more computationally efficient fine-tuning process

- All aggregation occurs over the LoRA adapters, allowing for decreased communication cost and more efficient aggregation.
- All fine-tuning is done over the databricks-dolly-15k dataset or a subset of it. This dataset was generated by Databricks and covers eight different capability domains from the InstructGPT paper (Ouyang et al., 2022).

3.1 Model Pruning

For our experiments, we simulate an FL system on our cluster. We examine 4 model sparsity levels (0%, 25%, 50%, and 75%), where each percentage indicates the proportion of weights that have been removed. To create the SLMs, we use SparseGPT (Frantar and Alistarh, 2023) to remove the weights from a LLaMA-7B LLM, inducing the specified level of sparsity in each model.

3.2 Model-agnostic Federated Learning

If SLMs are the building blocks, then FL is the process of assembling the blocks into a structure, and the resulting LLM is the final structure built from those blocks. We create a model-agnostic FL environment to allow aggregation between different sized SLMs, and the global LLM. At the end of the FL process, we obtain a fine-tuned global LLM constructed through the aggregation of SLMs. We select SLMs that would be representative of client devices depending on the experiment.

Algorithm 1 Federated Fine-Tuning with Heterogeneous Models

Initialization:
Each client n initializes LLM with parameter sparsity w_n.
$M \leftarrow \emptyset$; K communication rounds; $k \leftarrow 0$.
Training Loop:
while $k \leq K$ **do**
  Update M to select clients based on sparsity.
  for each client $n \in M$ **do**
    Select model for n with w_n.
    $\Delta w_{k+1,n} \leftarrow \text{InstructionTune}(\Delta w_{k,n})$.
  end for
  $\Delta w_{k+1} \leftarrow \text{HeteAgg}(\{\Delta w_{k+1,n} : n \in M\})$.
  $k \leftarrow k + 1$.
end while
Outcome:
Derive final adapters Δw_K; update global LLM w.

Algorithm 1 details our FL system, where clients would be assigned their respective SLMs with w_n sparsity, representing the sparsity present in both the model and the LoRA adapter. The clients are selected for fine-tuning through a client selection process (dependent on the scenario). During the training loop, clients fine-tune their LoRA adapters on local data created from a subset of the databricks-dolly-15k dataset. After fine-tuning, each of the selected clients has their LoRA adapters aggregated with each other to form a global update through the HeteAgg method—our heterogeneous model aggregation scheme detailed in Algorithm 2 . This global update is then applied to each of the client SLMs in addition to the global LLM. After the training loop is complete, we can derive our final adapters and global updates.

Algorithm 2 Model Heterogeneous Aggregation (HeteAgg)

Define global model g initialized to a baseline state.
for each client in selected clients set **do**
  Load client model state dictionary: s
  Identify $\mathcal{P}_{\text{common}}$, the set of common parameters between s and g
  Initialize $\mathcal{P}_{\text{avg}} \leftarrow \emptyset$
  for each parameter $p \in \mathcal{P}_{\text{common}}$ **do**
    Load p_s from s and p_g from g
    Define masks $M_s \leftarrow (p_s \neq 0)$, $M_g \leftarrow (p_g \neq 0)$
    $M_{\text{combined}} \leftarrow M_s \wedge M_g$
    $p_{\text{new}} \leftarrow \text{where}(M_{\text{combined}}, (p_s + p_g)/2, p_s + p_g - \text{where}(M_s, p_s, p_g))$
    $\mathcal{P}_{\text{avg}}[p] \leftarrow p_{\text{new}}$
  end for
  Update g with $\mathcal{P}_{\text{avg}}$
end for

In our HeteAgg approach, we begin by instatiating a global LLM to hold the eventual global update. This global update is formed by aggregating the client SLMs. This is done by accessing each of the selected client's LoRA adapters, and creating a mask for it based on its sparsity. This sparse mask is then aggregated with the global LLM's LoRA adapter wherever there is overlap between the mask and the adapter. Since sparsity is represented by a parameter magnitude '0' in the SLM's LoRA adapters, this process effectively averages the nonzero parameters between the client and global models.

By only aggregating across the nonzero weights, we can retain the sparsity in the client model's adapter without halving the global adapter's weights when there is no corresponding nonzero value. This process of mask creation and aggregation occurs for every client in the selected client group, forming a global update through the global LLM's adapter. Since every client SLM is a submodel of the LLM, we can apply the global update to each client in the same manner again using HeteAgg, averaging across each client's nonzero weights.

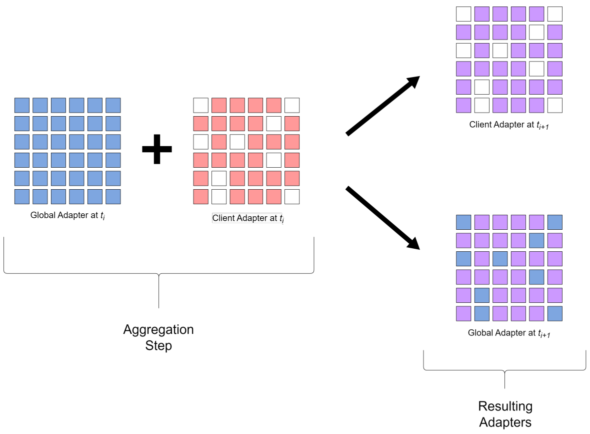

Figure 2: A symbolic representation of our heterogeneous aggregation method

Figure 2 represents our heterogeneous aggregation method, where the blue matrix represents the global LoRA adapter, and the red matrix represents a sparsified client LoRA adapter. The left-hand side displays each adapter at timestep t_i, before aggregation. During aggregation, the blue and red parameters average to create purple parameters for non-zero red (client) parameters. For zero-valued red (client) parameters, the updated client model retains its sparsity (upper right matrix), whereas the updated global LoRA adapter uses the blue (global) parameter values. As a result, the updated global adapter is a 0% sparsity adapter. Thus, the right-hand side displays each adapter at timestep t_{i+1}, where the parameters are aggregated only when there is an overlap between the corresponding non-zero parameters of each model.

4 Experiments

To rigorously examine the efficacy of our LEGO methodology, we conduct experiments to answer the following questions:

- Do different sparsity models learn differently? By federating and aggregating SLMs of strictly different sizes, we can test if the specific weights being tuned are similar in each size of model, allowing for knowledge transfer.

- Can the composition of SLMs yield a robust LLM? By strictly using SLMs in an FL system, we can test if their aggregation produces a robust LLM.

- Can task-specific SLMs stack together like building blocks to construct a generalizable LLM? By fine-tuning each client SLM on a unique, specific task, and aggregating them together, we can test if they can produce a single, robust LLM that retains each component SLM's domain knowledge.

We compare LEGO with these baselines:

- A FedIT-produced global model resulting from 4 LLaMA-7B models fine-tuned over i.i.d data. This baseline is the ideal case for FedIT.

- A FedIT-produced global model resulting from 8 task-specifc LLaMA-7B models where each model is only fine-tuned on one of the 8 different domain areas of databricks-dolly-15k.

FedIT is a foundational FL framework that our code extends (Zhang et al., 2023). The authors use an LLaMA-7B model with LoRA adapters and they sequentially fine-tune each adapter and then aggregate using FedAvg into the global model.

Since the computational cost of HeteAgg is the same as FedAvg, all speedups in LEGO are a direct result of model pruning (Sun et al., 2023; Frantar and Alistarh, 2023). During our experiments, we observe up to a $1.7\times$ speedup in inference and up to a $1.4\times$ speedup in fine-tuning using SparseGPT-produced SLMs when compared to 0% sparsity LLMs.

4.1 Heterogeneous Aggregation Validation

When using building blocks, we often encounter blocks of varying sizes. To create a cohesive structure, we must stack these differently sized blocks ontop of one another. This concept is the central to our LEGO methodology, as much like the blocks, different sized SLMs must be assembled together to create a robust LLM.

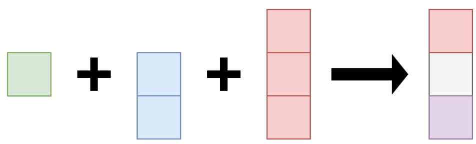

Figure 3: A representation of how three different SLMs can be stacked (aggregated) together using blocks, where each color is representative of the SLM's knowledge.

Table 1: Average Model Performance Over Benchmarks

Composition	Sparsity Level	Pruned	Fine-Tuned	Aggregated
4 Strictly Heterogeneous Models	0%	0.559	0.563	0.568
	25%	0.554	0.561	0.565
	50%	0.529	0.526	0.542
	75%	0.384	0.412	0.396
5 SLMs With i.i.d Data Distribution	0%	0.559	-	0.568
	50%	0.529	-	0.541
8 Task-Specific SLMs	0%	0.559	-	0.571
	75%	0.240	-	0.411
FedIT: 4 LLMs With i.i.d Data Distribution	0%	0.569	-	0.567
FedIT: 8 Task-Specific LLMs	0%	0.569	-	0.563

Figure 3 illustrates how SLMs of various sizes—each being represented by different color blocks—are stacked together. When being stacked, similar to Figure 2, we see that wherever there is an overlap, the average is taken between the overlapping blocks. The final, resultant block consists of three sections: the top red layer, where the largest block does not overlap with others; the bottom purple layer, an average of the blue and red where two blocks overlap; and the middle white section, where all three blocks overlap. This averaging of colors is representative of the knowledge being transferred between the models.

In the case of LEGO, successful stacking of heterogeneous SLMs causes each model to learn from each other, with knowledge transferring between models. Thus, this experiment tests the effectiveness of HeteAgg, our "stacking" mechanism, by creating an FL environment with exclusively heterogeneous clients. We set a scenario with four clients, each with different sparsity levels (0%, 25%, 50%, and 75%). Each client has an i.i.d portion of localized data to fine-tune on.

Table 1 displays the performance of different-sized models for a model composition with 4 strictly heterogeneous models. We benchmark performance at three different stages: when the LLM was initially pruned before fine-tuning (Pruned), when the model is fine-tuned on local data (Fine-Tuned), and the final adapters after all FL rounds and global updates (Aggregated). As displayed in the table, we see that fine-tuning improves performance for all model sizes, with a significant performance gain at the 75% sparsity level. The aggregation stage improves performance for all model sizes at 0%-50% sparsity but degrades at 75% sparsity.

Comparing against the FedIT-produced baseline with 4 strictly homogeneous LLMs, we see that when using heterogeneous models, an equally robust 0% LLM is produced. While, the 25% sparsity model is equally robust, performance begins degrading at 50% sparsity.

The 75% sparsity model's degraded performance is likely due to the SLM's limited size. Previous work has shown that smaller models are better learners for specific tasks, resulting in more strongly tuned weights to offset size constraints (Turc et al., 2019; Raffel et al., 2020). During aggregation with larger models, the stronger learned representation in smaller models become diluted by the larger model's weaker representation, causing degraded performance in the smaller model.

The 0% sparsity LLM resulting from our four aggregated heterogeneous client models matches the FedIT benchmark performance of four aggregated LLMs. These results show that LEGO can account for clients that have diverged from their learned representations due to high sparsity or overfitting client data..

4.2 Building Blocks Methodology Validation

When building large structures, it is common to assemble smaller sub-units individually before combining them into the final form. Similarly, with LEGO, we can fine-tune smaller models individually, treating them as sub-units that are then aggregated together to produce a final LLM.

We test whether LEGO has the same capability by exclusively composing SLMs, and aggregating them together to create a robust LLM. This experiment tests the transferability of knowledge from SLMs to an LLM using LEGO. We employ five

50% sparsity client SLMs for fine-tuning and aggregating, and apply the resulting global updates to a 0% sparsity global LLM.

The results of this experiment, composed with 5 SLMs with i.i.d data distribution, are in Table 1. Despite only fine-tuning SLMs, we achieved a 0% LLM better than the FedIT LLM produced from 4 LLMs with an i.i.d data distribution. These results demonstrate that LEGO allow for knowledge transfer from strictly smaller models to a larger model in an effective manner.

4.3 Task-specific Knowledge Transfer Validation

Just as not all (SLM) building blocks are the same size, they may not necessarily be the same shape. Regardless of the size or shape, the requirement is that they can stack together. LEGO demonstrates this principle.

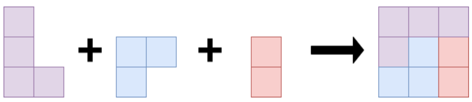

Figure 4: 3 differently shaped building blocks being combined to create a larger block

Figure 4 shows three blocks of differing shapes being combined to create a new, larger block that encompasses the different shapes. The same can be done with SLMs, where each SLM can be covering a different task or scenario, but be aggregated together to create a robust LLM that covers the diverse tasks of its components.

The experiment of this section evaluates knowledge transfer in a non-i.i.d data distribution scenario. We use eight 75% sparsity client SLMs; each fine-tuned on one of the eight capability domains in the databricks-dolly-15k dataset. We apply the resulting global updates from the client aggregation stages to a global LLM.

The results of this experiment consisting of 8 task-specific SLMs are in Table 1, demonstrating that despite each model being fine-tuned on a different task, the knowledge transfers between models, resulting in a more robust global 0% sparsity LLM than any of the previous experiments.

This can most likely be attributed to the small size of the SLMs. As discussed before, previous work in KD has shown that smaller models are more adept learners when it comes to task specific models. To our knowledge, no previous study has explored task-specific SLMs in the context of pruning. However, our results demonstrate that the same task-specific adaptation strength present in KD-produced SLMs is also present in pruning-produced SLMs, despite not distilling over select tasks.

The learned representations in the SLMs are more strongly reflective of their fine-tuning data due to their limited size. Thus, when aggregating the SLMs with the global LLM, the LLM obtains the stronger task specific representations from the SLMs. The LLM gains this knowledge while being bolstered by its larger size, creating a more robust model.

Thus, the results demonstrate that smaller models make better task-specific learners, and their knowledge can be effectively transferred to larger models, yielding robust LLMs while only fine-tuning SLMs.

The the LEGO produced 0% sparsity LLM formed by 8 task-specific SLMs outperforms the FedIT baseline with 8 task-specific LLMs, despite only using SLMs a quarter of the size.

Additionally, we further test how well knowledge transfers between the SLMs. To do so, we track the performance of client SLMs over time, evaluating their performance after every global update.

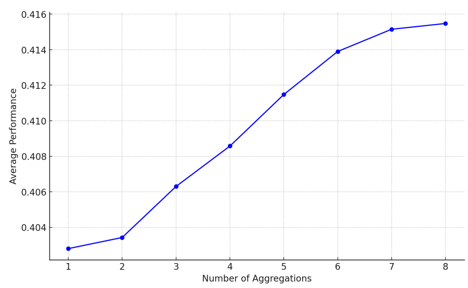

Figure 5: The performance of clients after each global update.

Figure 5 demonstrates that after every communication round, the performance of the client SLMs increase. Thus, we can determine that if one model learns, then they all learn.

5 Related Work

Works on heterogeneous federated learning in the context of pretrained language models are sparse.

The first paper to cover the topic in-depth was InclusiveFL (Liu et al., 2022), where the authors used layer-pruned BERT models in a federated system and aggregate across layers. The authors found layer-pruning to have a negligible effect on BERT's performance - something that does not apply to modern LLMs.

This can be attributed to the emergent large-magnitude features in LLMs, which are sparse and distributed randomly across layers and have a significant effect on LLM performance (Dettmers et al., 2022). While Wanda and SparseGPT avoid this, layer pruning cannot do so. We experimentally confirm this in Appendix A.2.

We can extend this reasoning to similar approaches focused on layer selection that are only tested on encoder-style LLMs, like FedPepTAO (Che et al., 2023).

We then look to homogeneous model FL applied to larger, decoder-style LLMs. FedIT (Zhang et al., 2023) acts as the representation of traditional FL throughout our work, using FedAvg for aggregation as mentioned in Section 4. However, FedAvg cannot adapt to heterogeneous models, and as pointed out by other works, cannot account for heterogeneous ranks in the LoRA adapter(Bai et al., 2024).

Newer works have continued to model themselves after FedIT's use of LoRA. Recently, enabling heterogeneous LoRA ranks in FL has been discussed in the literature. For example, FlexLORA computes a weighted average of LoRA adapters with different LoRA ranks, and then uses SVD for redistribution (Bai et al., 2024). However, FlexLoRA assumes model homogeneity among client models, which is what allows for adaptive rank pruning in the LoRA adapter.

The advantages of rank pruning do not translate to the advantages of model pruning. Model pruning allows for more efficient fine-tuning and inference, whereas pruning LoRA only translates to more efficient fine-tuning, with the same inference costs as the initial LLM. Thus, in FlexLoRA, model selection is constrained by weakest device. Pruning allows larger models (LLMs) to run on more powerful devices, and smaller models (SLMs) to run on weaker devices.

Additionally, this aggregation technique relies on multiplying each client's LoRA adapters, A and B, together, where $A \in \mathbb{R}^{r \times n}$ and $B \in \mathbb{R}^{n \times r}$. The multiplication results in the server creating the full-sized weights for every client model before aggregating them together. This extremely resource intensive operation limits the scalability of the technique relative to ours, where the LoRA modules stay separate.

However, LEGO does not have to exclusively operate over PEFT adapters. The same approach and aggregation methods used for LoRA adapters can be performed with the actual client weights, or with the multiplied LoRA adapters. This means that rank-pruning techniques can be applied with or on top of LEGO, further decreasing SLM size, at the cost of increased computation for the server.

6 Conclusions

In this work, we have introduced LEGO, a building block methodology for federated fine-tuning of LLMs. By allowing for the use of pruned LLMs, we can use SLMs as task-specific learners for resource-constrained devices, and use them as building blocks, stacking them into a fully robust LLM. This is enabled through our simple yet effective aggregation scheme, HeteAgg, which allows for the aggregation of heterogeneous SLMs. Through experimentation, we prove that LEGO is effective, allowing for SLMs to be stacked together like building blocks. We demonstrate that smaller models make better learners, which translates to stronger models, and also show that individual client learning translates to all models learning. By enabling heterogeneous client resource budgets, LEGO creates a more scalable and resource-efficient FL system for private conversational AI.

7 Limitations

Our approach has limitations caused by prioritizing efficiency. As mentioned in Section 3, we operate over client LoRA adapters. Each LoRA module A and B is aggregated separately, which introduces noise to the resulting weights, as

$$\underbrace{\sum A \times \sum B}_{\text{LEGO}} \neq \underbrace{\sum (A \times B)}_{\text{Noise-Free Aggregation.}}$$

Despite the noise, however, we show experimentally that LEGO produces robust models.

References

Jiamu Bai, Daoyuan Chen, Bingchen Qian, Liuyi Yao, and Yaliang Li. 2024. Federated fine-tuning of large language models under heterogeneous language tasks and client resources. *arXiv preprint arXiv:2402.11505*.

Tianshi Che, Ji Liu, Yang Zhou, Jiaxiang Ren, Jiwen Zhou, Victor S Sheng, Huaiyu Dai, and Dejing Dou. 2023. Federated learning of large language models with parameter-efficient prompt tuning and adaptive optimization. *arXiv preprint arXiv:2310.15080*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2023. A framework for few-shot language model evaluation.

Samyak Gupta, Yangsibo Huang, Zexuan Zhong, Tianyu Gao, Kai Li, and Danqi Chen. 2022. Recovering private text in federated learning of language models. *Advances in Neural Information Processing Systems*, 35:8130–8143.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models.

Wenke Huang, Mang Ye, and Bo Du. 2022. Learn from others and be yourself in heterogeneous federated learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10143–10153.

Ruixuan Liu, Fangzhao Wu, Chuhan Wu, Yanlin Wang, Lingjuan Lyu, Hong Chen, and Xing Xie. 2022. No one left behind: Inclusive federated learning over heterogeneous devices. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3398–3406.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2023. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962*.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2023. Sheared llama: Accelerating language model pre-training via structured pruning.

Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. 2024. A survey on knowledge distillation of large language models. *arXiv preprint arXiv:2402.13116*.

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

Jianyi Zhang, Saeed Vahidian, Martin Kuo, Chunyuan Li, Ruiyi Zhang, Guoyin Wang, and Yiran Chen. 2023. Towards building the federated gpt: Federated instruction tuning. *arXiv preprint arXiv:2305.05644*.

A Appendix

A.1 Comparison of Pruning Methods

As discussed in the Background section, there are two pruning techniques that dominate the literature. We test both SparseGPT and Wanda and analyze the best pruning technique to use.

The results in table 2 show that SparseGPT produces more robust models on average, with a significant advantage at higher levels of sparsity. However, SparseGPT is more computationally expensive when pruning, while Wanda is computationally inexpensive.

This provides us a few insights. The first is that regardless of pruning strategy, performance degrades significantly beyond 50% sparsity. The second is that while more computationally expensive, SparseGPT may be necessary at high sparsity

Table 2: Comparison of SparseGPT and Wanda Pruned Models

Sparsity Level	SparseGPT		Wanda	
	Pruned	Fine-tuned	Pruned	Fine-tuned
0%	0.5694	**0.5760**	0.5694	0.5741
25%	0.5654	**0.5784**	0.5672	0.5731
50%	0.5144	0.5244	0.5195	**0.5377**
75%	0.2989	**0.3631**	0.2692	0.2916

Table 3: All models were pruned from LLaMA-7B and evaluated over HellaSwag (Zellers et al., 2019). The Fine-tuned models were fine-tuned over databricks-dolly-15k. Bolded scores are the best in sparsity level.

levels or more resource constrained client devices, as it not only produced a more robust model, but the increase in performance due to fine-tuning was almost double that of Wanda.

Given these insights, the superior pruning method depends on the use case scenario. If we are defining rigid model sizes and assert that client devices will be initialized with one of these 'default' model sizes, then SparseGPT would be superior. This is especially true given our compute budget is capable of fine-tuning LLMs and performing inference, since SparseGPT is relatively cheap compared to those tasks if not being performed for ever device initialization. Thus, we can use SparseGPT to generate various model sizes/sparsity's before the FL process begins, and assign models accordingly.

However, in practice, creating a methodology to calculate the ideal model size given the device's compute budget would return more robust client models for users in the FL system. In this scenario, when a client is initialized, a model would be pruned according to their compute budget, meaning a lightweight process like Wanda would be superior.

However it is worth noting that, with the exception of high sparsity scenarios, the difference between the two pruning method's performances is negligible. Therefore, our results should be generalizable to both pruning methods.

Additionally, as pruning methods continue to evolve, the performance of pruned models will improve. Therefore its important evaluate model performance in our experiments with the limitations of current pruning techniques, but as pruning techniques improve, our methodologies and results would generalize to them and should scale accordingly.

In order to confirm if our experimental results are generalizable to other pruning techniques, we also test the Wanda-pruned SLMs for our HeteAgg experiment. We perform the same experiment involving 4 models at different sparsity levels, with its results displayed in table 4.

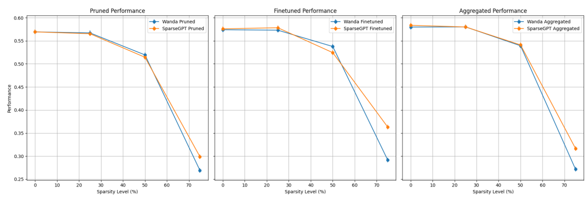

Figure 6: Performance of federated SparseGPT-pruned models relative to federated Wanda-pruned models when evaluated on HellaSwag (Zellers et al., 2019)

When plotted against SparseGPT's performance in figure 6, we see that the effects of our FL approach are near identical. For sparsity $\geq 50\%$, we see that the results are nearly identical, and the performance gap displayed by the fine-tuned 50% sparsity SparseGPT-pruned model is corrected after model aggregation.

While the performance on HellaSwag is different at high sparsity, that can be attributed to Wanda's weaker pruning ability at high sparsity levels. When viewing the Wanda and SparseGPT pruned 75% sparsity models, we see the drop in performance due to aggregation after fine-tuning is nearly identical.

Therefore, since the performance is nearly identical, and the only significant difference in performance can be attributed to the initial model performance as opposed to our FL system, we can generalize our FL method to other current pruning techniques.

A.2 Experimental Comparison with InclusiveFL

In order to confirm the effect of emergent large-magnitude features in LLMs discussed in Section 5, we experimentally compare InclusiveFL and layer

Table 4: Performance of Wanda pruned models on HellaSwag (Zellers et al., 2019)

Sparsity Level	Pruned	Fine-Tuned	Aggregated
0%	0.5694	0.5741	**0.5799**
25%	0.5672	0.5731	**0.5802**
50%	0.5195	0.5377	**0.5393**
75%	0.2692	**0.2916**	0.2717

pruning to LEGO and activation pruning. To do so, we layer-prune LLaMA-7B and modify our HeteAgg function to perform layer-wise aggregation.

We pruned LLaMA-7B to 24 and 16 layers, equivalent to 25% and 75% sparsity. We then put these two models and a 0% sparsity LLaMA-7B model in the federated environment from Algorithm 1, modifying the HeteAgg function to follow the pseudocode in the InclusiveFL paper. For closest comparison we take select results from Section 4.1 and Table 1.

In Table 5, we can see that even before federation, layer pruning fails to conserve model performance after pruning. This can be attributed to the emergent large-magnitude features in LLMs, as described in Section 5 (Dettmers et al., 2022). After federation, the fine-tuning and aggregation process degraded the performance, proving that this approach does not work for LLMs.

A.3 Experimental Setup and Performance

For all of the experiments, due to hardware limitations we use a client selection strategy that sequentially chooses clients. We use a client participation rate of 0.1, with a local batch size of 64 and a maximum of 10 epochs. For our LoRA adapter settings, we chose a rank and alpha of 16, and only target the q_proj.

Table 1 showed the average model performance for each model. The individual results for each benchmark of each model is held in Table 6. We evaluate each model on HellaSwag (Zellers et al., 2019), MMLU (Hendrycks et al., 2021), SciQ, and ARC (Clark et al., 2018). We evaluate the models using the EleutherAI Language Model Evaluation Harness (Gao et al., 2023).

Table 5: Performance of layer-pruning (Liu et al., 2022) compared to activation pruning (our study).

Sparsity / Layers	**Pruned**		**Fine-tuned & Aggregated**	
	SparseGPT	**Layer-Pruning**	**SparseGPT**	**Layer-Pruning**
Full Sized	0.5694	0.5694	**0.5836**	0.5148
25% Sparsity / 24 Layers	0.5654	0.3957	**0.5801**	0.3658
50% Sparsity / 16 Layers	0.5144	0.3021	**0.5411**	0.3014

Sparsity (%)	**Stage**	**HellaSwag**	**MMLU**	**SciQ**	**Arc**
4 Strictly Heterogeneous Models					
0	Pruned	0.569	0.299	0.947	0.419
0	Fine-Tuned	0.576	0.295	0.950	0.429
0	Aggregated	0.584	0.301	0.953	0.435
25	Pruned	0.565	0.292	0.938	0.422
25	Fine-Tuned	0.578	0.286	0.944	0.437
25	Aggregated	0.580	0.295	0.944	0.442
50	Pruned	0.514	0.292	0.935	0.375
50	Fine-Tuned	0.524	0.267	0.932	0.379
50	Aggregated	0.541	0.292	0.932	0.404
75	Pruned	0.299	0.230	0.809	0.197
75	Fine-Tuned	0.363	0.237	0.828	0.221
75	Aggregated	0.317	0.229	0.832	0.206
5 SLMs With iid Data Distribution					
0	Pruned	0.569	0.299	0.947	0.419
0	Aggregated	0.581	0.296	0.953	0.443
50	Pruned	0.514	0.292	0.935	0.375
50	Aggregated	0.540	0.291	0.935	0.399
8 Task-Specific SLMs					
0	Pruned	0.569	0.299	0.947	0.419
0	Aggregated	0.586	0.298	0.953	0.446
75	Pruned	0.299	0.230	0.233	0.197
75	Aggregated	0.359	0.241	0.813	0.233
FedIT: 4 LLMs					
0	Aggregated	0.575	0.286	0.956	0.453
FedIT: 8 Task-Specific LLMs					
0	Aggregated	0.570	0.279	0.951	0.452

Table 6: Model Performance Across Different Configurations and Datasets