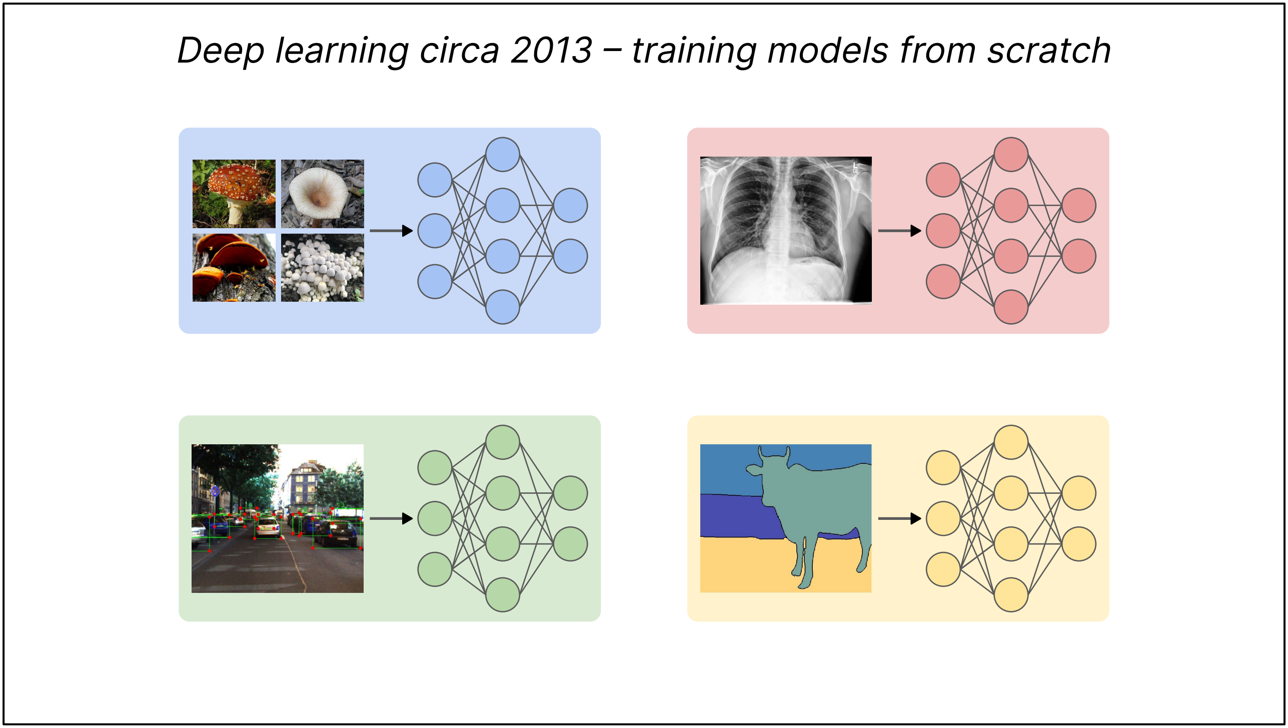
Collaborative, Communal, & Continual Machine Learning

Colin Raffel

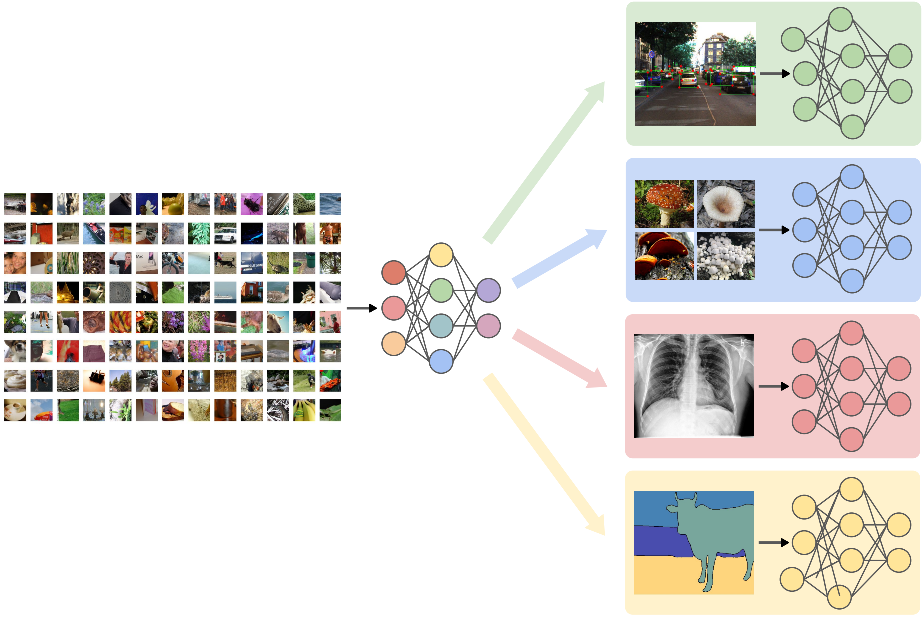
Hi everyone, today I'm going to be talking about a major research direction in my lab that aims to make it possible to build communal machine learning models in a collaborative and continual fashion.Hi everyone, today I'm going to be talking about a major research direction in my lab that aims to make it possible to build communal machine learning models in a collaborative and continual fashion.



Deep learning circa 2013 – training models from scratch

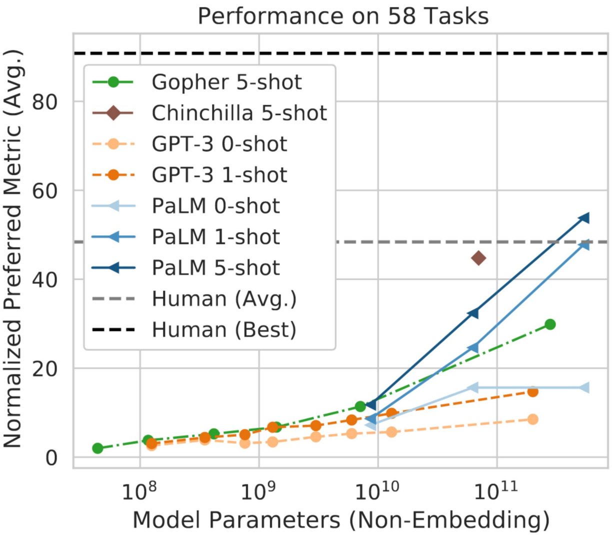
Ten years ago, when neural networks started to see their most recent major resurgence, the standard way to train a model was to collect a large collection of labeled data and train it from scratch to perform a specific task. On this slide, I'm showing examples of various image tasks like image classification, object detection, and image segmentation. In each case, a specialized model is trained on the target task dataset and no capabilities or information is shared across the models. This turned out to be a powerful paradigm, but only when you had enough labeled data for your target task.

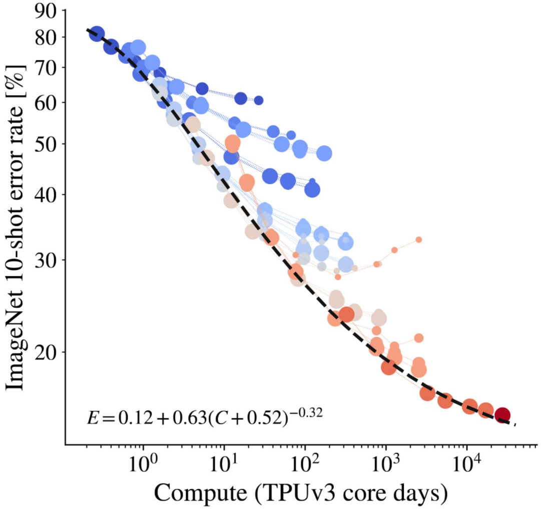
Deep learning in 2023 – pre-train then adapt



More recently, a different paradigm has become more common, where a model is first pre-trained on a diverse dataset coming from task that requires broad capabilities. The pre-trained model is then adapted to target tasks through various means, including fine-tuning through further training on target task data. In this case, I'm showing the widely-used ImageNet dataset, which is an incredibly common source of data for pre-training image models. Performing this pre-training step can allow a model to converge more quickly to a better solution with less labeled data from the target task and this paradigm has therefore become incredibly common.

The benefits – and costs – of scale

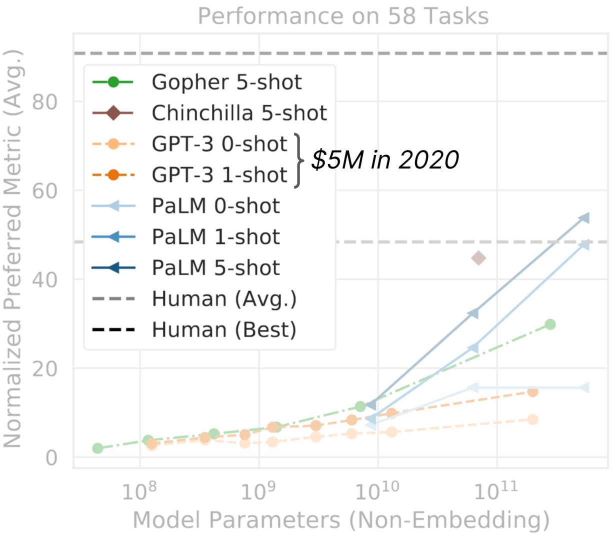


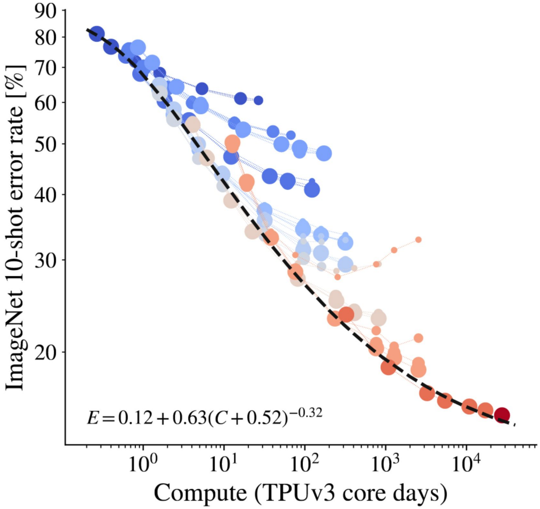


From "PaLM: Scaling Language Modeling with Pathways" by Chowdhery et al. and "Scaling Vision Transformers" by Zhai et al.

One of the major benefits of the pre-training step is that often pre-training can be done on extremely large and diverse datasets. **Often the pre-training data doesn't need to be labeled at all.** This has allowed scaling up both the size of the pre-training data and the size of the models they are trained on. Increased scale has demonstrated consistent improvements in many settings. Here I'm showing two figures from two recent papers, one focusing on language models on the left and one on vision models on the right. In both cases, a measurement of model scale is shown on the x-axis - parameter count on the left and training compute on the right - and a measure of performance is shown on the y-axis - accuracy on the left and error rate (where lower is better) on the right. As you can see, in both cases there is a reliable improvement in performance as models are scaled up. This trend has led to a major focus on scaling up models.

The benefits – and costs – of scale

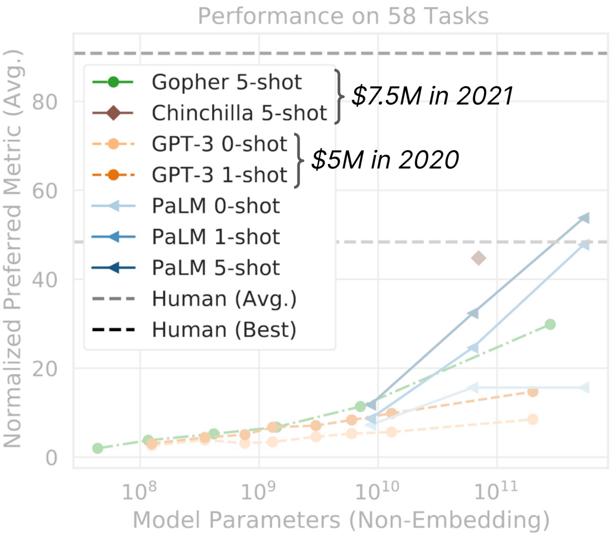




From "PaLM: Scaling Language Modeling with Pathways" by Chowdhery et al. and "Scaling Vision Transformers" by Zhai et al.

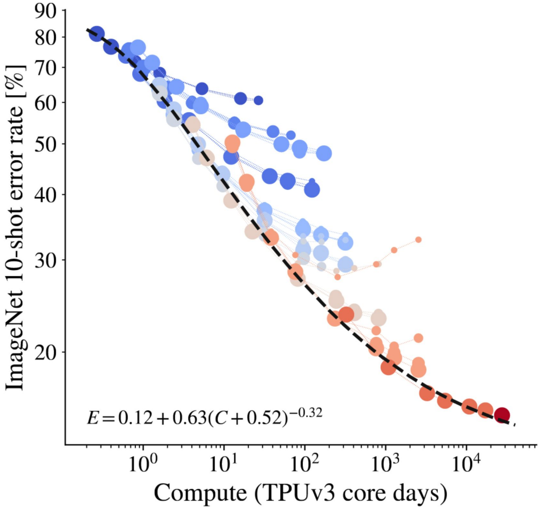
However, increased scale increases costs. To give a picture of these costs, GPT-3 -- one of the language models whose performance is shown on the left graph -- would have cost about $5M to train in 2020...

The benefits – and costs – of scale



$7.5M in 2021

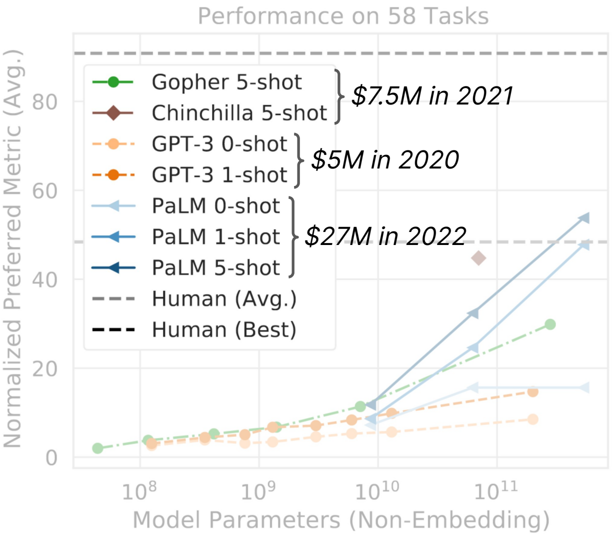
$5M in 2020



From "PaLM: Scaling Language Modeling with Pathways" by Chowdhery et al. and "Scaling Vision Transformers" by Zhai et al.

... Chinchilla and Gopher would have cost about $7.5M to train in 2021 ...

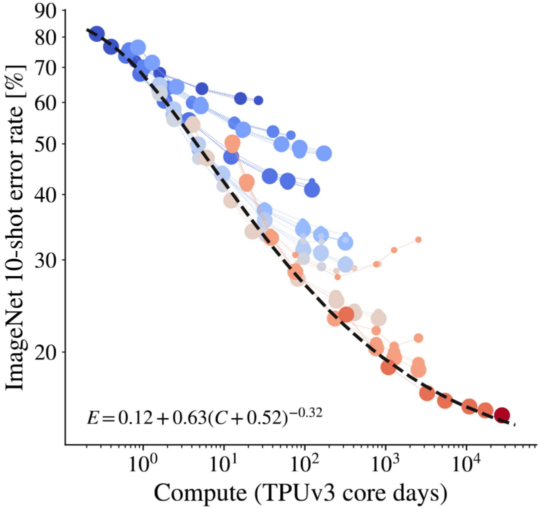
The benefits – and costs – of scale



$7.5M in 2021

$5M in 2020

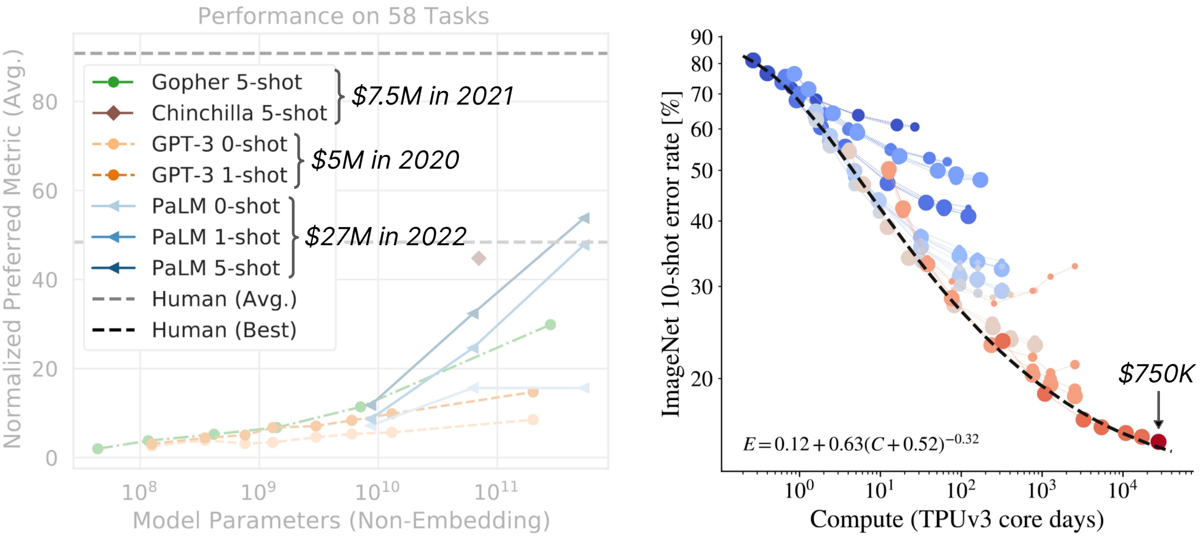
$27M in 2022



From “PaLM: Scaling Language Modeling with Pathways” by Chowdhery et al. and “Scaling Vision Transformers” by Zhai et al.

... and the recent PaLM model would have cost about $27M to train in 2022. Clearly these costs are out of reach for most researchers, including myself and I assume all of us in the room.

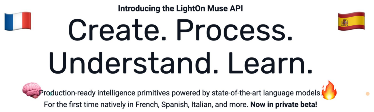
The benefits – and costs – of scale



From "PaLM: Scaling Language Modeling with Pathways" by Chowdhery et al. and "Scaling Vision Transformers" by Zhai et al.

The situation in vision is a little less dire -- the most expensive model on the plot on the right would have cost about $750k to train -- but this is still a huge amount of money that is out of most researchers budgets.

Increased costs have decreased sharing


Introducing the LightOn Muse API

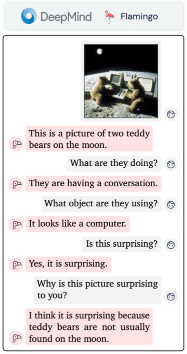
Create. Process. Understand. Learn.

Production-ready intelligence primitives powered by state-of-the-art language models.
For the first time natively in French, Spanish, Italian, and more. **Now in private beta!**

HyperCLOVA

Designing businesses to fill with ideas & creativity.

"あなたならではの" 想い、創造力を発揮できる余白をつくる


DeepMind Flamingo

This is a picture of two teddy bears on the moon.

What are they doing?

They are having a conversation.

What object are they using?

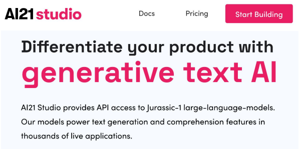
It looks like a computer.

Is this surprising?

Yes, it is surprising.

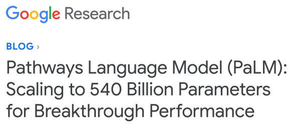
Why is this picture surprising to you?

I think it is surprising because teddy bears are not usually found on the moon.



Differentiate your product with **generative text AI**

AI21 Studio provides API access to Jurassic-1 large-language-models. Our models power text generation and comprehension features in thousands of live applications.

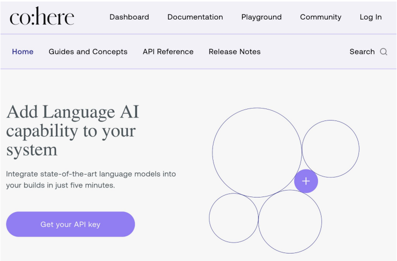

Google Research

BLOG ›

Pathways Language Model (PaLM): Scaling to 540 Billion Parameters for Breakthrough Performance

Imagen

unprecedented photorealism × deep level of language understanding

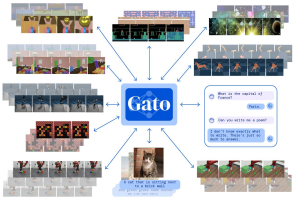


Add Language AI capability to your system

Integrate state-of-the-art language models into your builds in just five minutes.

Microsoft Megatron-Turing NLG 530B

The World's Largest and Most Powerful Generative Language Model



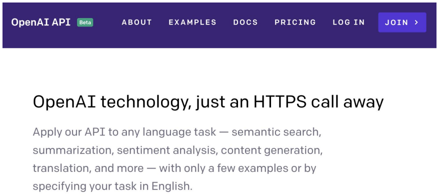
What is the capital of France?

Paris.

Can you write me a poem?

I don't know exactly what to write. There's just so much to answer.

A cat that is sitting next to a brick wall and green grass some bushes as the sun sets

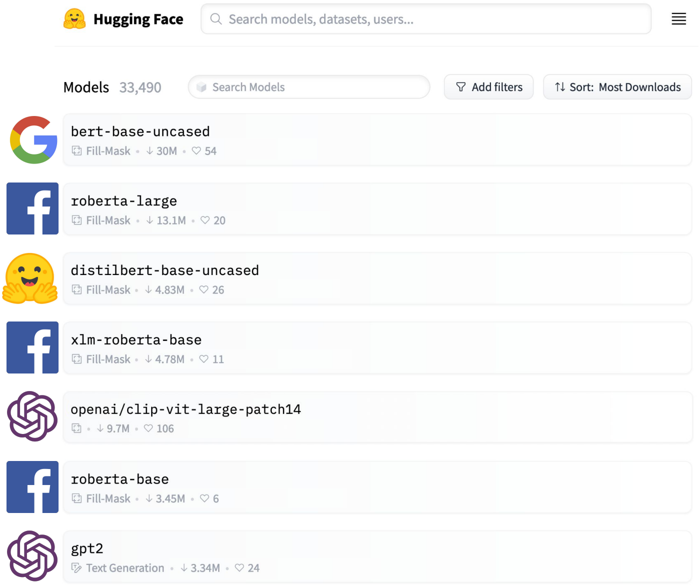


OpenAI technology, just an HTTPS call away

Apply our API to any language task — semantic search, summarization, sentiment analysis, content generation, translation, and more — with only a few examples or by specifying your task in English.

A consequence of this cost has been that pre-trained models are increasingly created by resource-rich groups and kept proprietary - either via a commercial product or API or without any public access at all. **Companies can decide not to release models for other reasons such as safety**, but this trend of not releasing models is a huge change from the way things were 5 or so years ago, where it was standard for researchers in industry to freely share pre-trained models.

Popular public models often come from resource-rich groups



Models 33,490

bert-base-uncased
Fill-Mask • 30M • 54

roberta-large
Fill-Mask • 13.1M • 20

distilbert-base-uncased
Fill-Mask • 4.83M • 26

xlm-roberta-base
Fill-Mask • 4.78M • 11

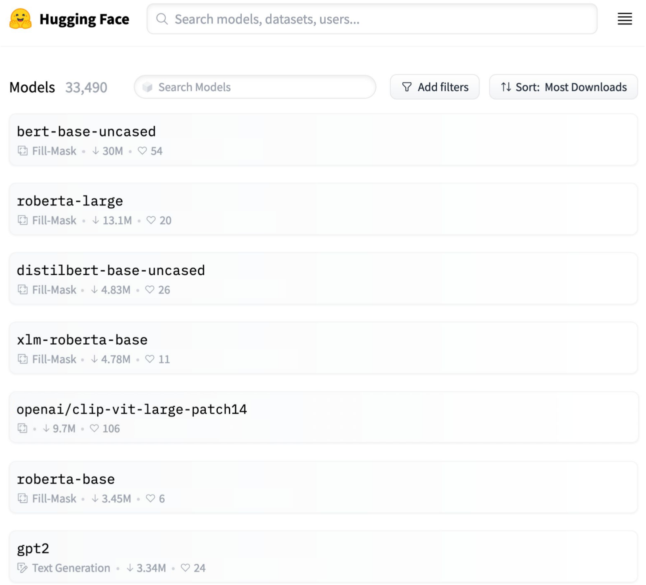
openai/clip-vit-large-patch14
• 9.7M • 106

roberta-base
Fill-Mask • 3.45M • 6

gpt2
Text Generation • 3.34M • 24

Another consequence of these costs has been that many of the most popular pre-trained models come from resource-rich groups. Here's a screenshot I recently took from the Hugging Face Model Hub, a popular repository for accessing public pre-trained models. As you can see, the most popular models all come from relatively resource-rich institutions such as Google, Facebook, and OpenAI - none of them come from academia or other smaller groups.

… and the models themselves are rarely updated

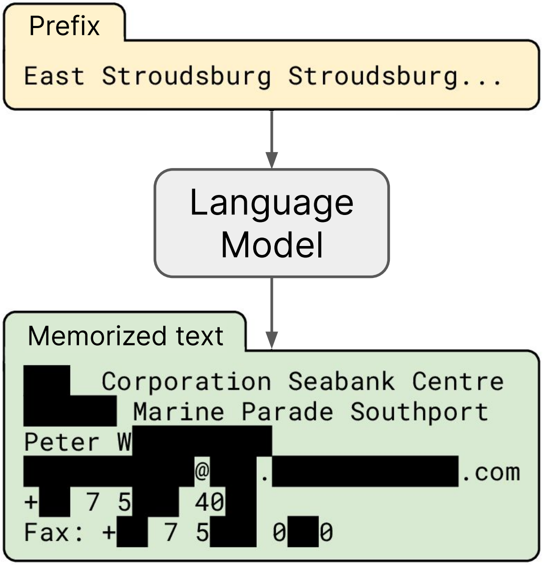


Models 33,490

2018	bert-base-uncased	Fill-Mask	↓ 30M	♡ 54
2019	roberta-large	Fill-Mask	↓ 13.1M	♡ 20
2019	distilbert-base-uncased	Fill-Mask	↓ 4.83M	♡ 26
2019	xlm-roberta-base	Fill-Mask	↓ 4.78M	♡ 11
2021	openai/clip-vit-large-patch14		↓ 9.7M	♡ 106
2019	roberta-base	Fill-Mask	↓ 3.45M	♡ 6
2019	gpt2	Text Generation	↓ 3.34M	♡ 24

Another issue that becomes apparent from looking at these models is that they are never updated! For example, BERT, the most popular model with tens of millions of downloads in the last month alone, remains exactly the same as it was when it was released 5 years ago.

Models can exhibit issues, like memorized training data



Prefix

East Stroudsburg Stroudsburg...

Language Model

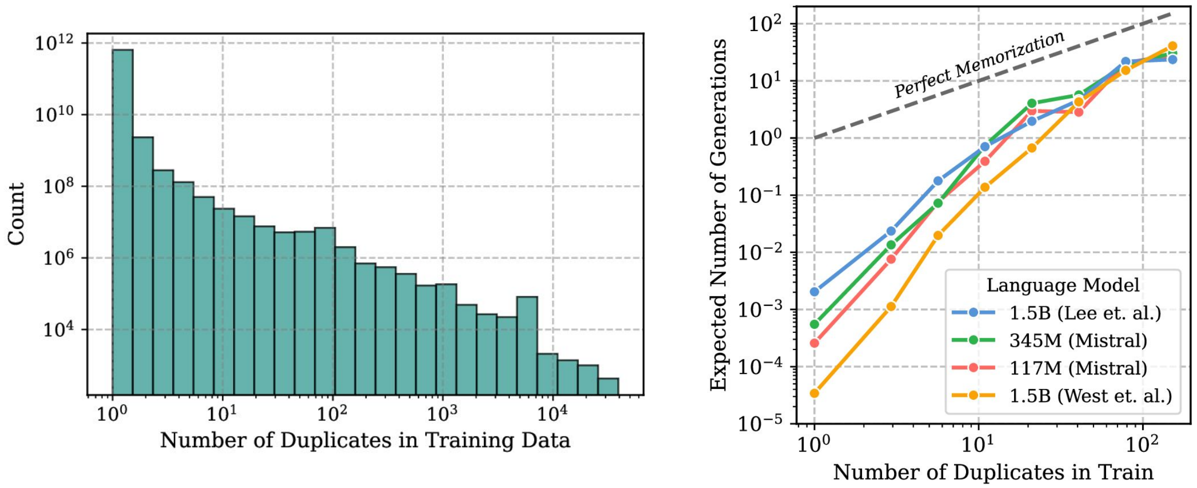
Memorized text

[redacted] Corporation Seabank Centre
Marine Parade Southport
Peter W[redacted]
[redacted]@[redacted].[redacted].com
+[redacted] 7 5[redacted] 40[redacted]
Fax: +[redacted] 7 5[redacted] 0[redacted]0

From "Extracting Training Data from Large Language Models" by Carlini et al.

There are many reasons we might want to update a model. For example, the model could exhibit some problematic behavior we want to fix. In work published a few years ago with my collaborators at Google, we showed that language models can memorize and regurgitate their pre-training data. We discovered this behavior by comparing the likelihood that a model assigns its own generations to the likelihood assigned by a different model. If the model assigns an incongruously high likelihood to its own generation, then we flag it as possible memorized, and we found that the popular GPT-2 model had memorized many such examples this way. Some examples included PII like phone numbers or addresses that we wouldn't necessarily want to store indefinitely in the parameters of our model.

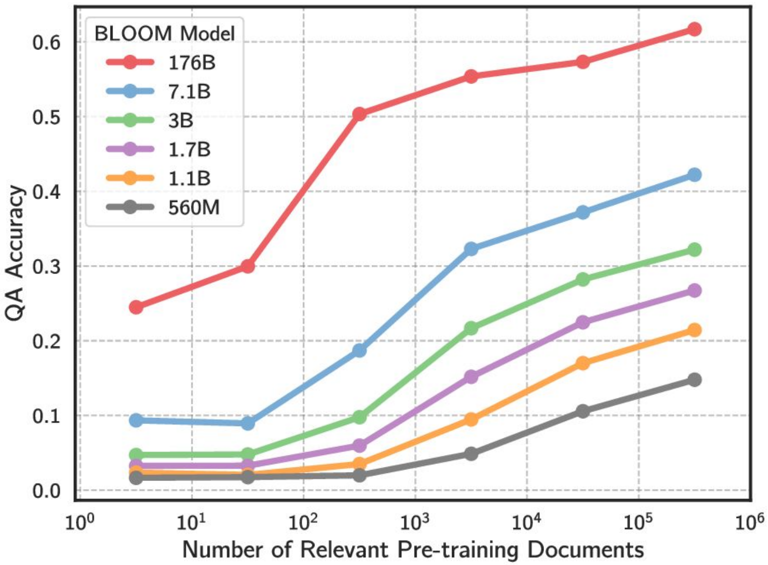
Issues with a model can be caused by issues with a dataset



From "Deduplicating Training Data Mitigates Privacy Risks in Language Models" by Kandpal et al.

In more recent work, we found that this problematic behavior is mainly caused by issues with the datasets used to train these models. Specifically, we found that duplication in a pre-training dataset is a major cause of memorization. As you can see on the left, the text datasets used to pre-train language models are not duplicated - some examples are repeated tens of thousands of times. And, on the right, you can see that as a given chunk of text is repeated more times in the pre-training dataset, the number of times it is generated by the model increases superlinearly. For example, a chunk of text that appears 10 times in the pre-training dataset is about 1,000 times more likely to be generated than a chunk of text that only appears once. So, in summary, issues with pre-trained models can stem from issues in the datasets used to train them. The unfathomably large size of pre-training datasets makes it hard to audit them, meaning it is likely that we will uncover additional issues we want to fix in the future.

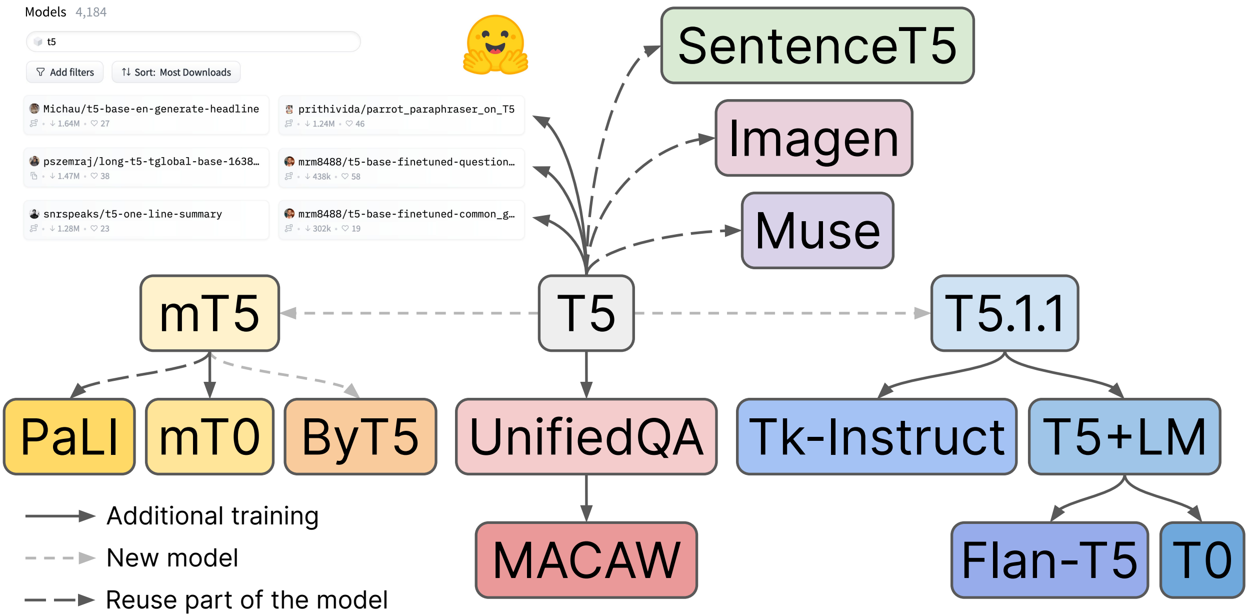
Pre-training datasets can also fail to address downstream needs



From "Large Language Models Struggle to Learn Long-Tail Knowledge" by Kandpal et al.

On the other hand, pre-training datasets can also be insufficient to address target uses of a model. In a recent paper we studied whether a language model's ability to answer a question about a fact was related to the number of times the fact appeared in the pre-training dataset. Sure enough, models struggled to answer questions about long-tail knowledge that rarely appeared in the pre-training dataset. If we want a model to obtain this knowledge, we would need to update it in some way. So updating a model can also aim to improve it, rather than just fix it.

Pre-trained models are often used as the basis for derivative models



Models 4,184

t5

Add filters

Sort: Most Downloads

Michau/t5-base-en-generate-headline
↓ 1.64M • ♡ 27

prithivida/parrot_paraphraser_on_T5
↓ 1.24M • ♡ 46

pszemraj/long-t5-tglobal-base-1638…
↓ 1.47M • ♡ 38

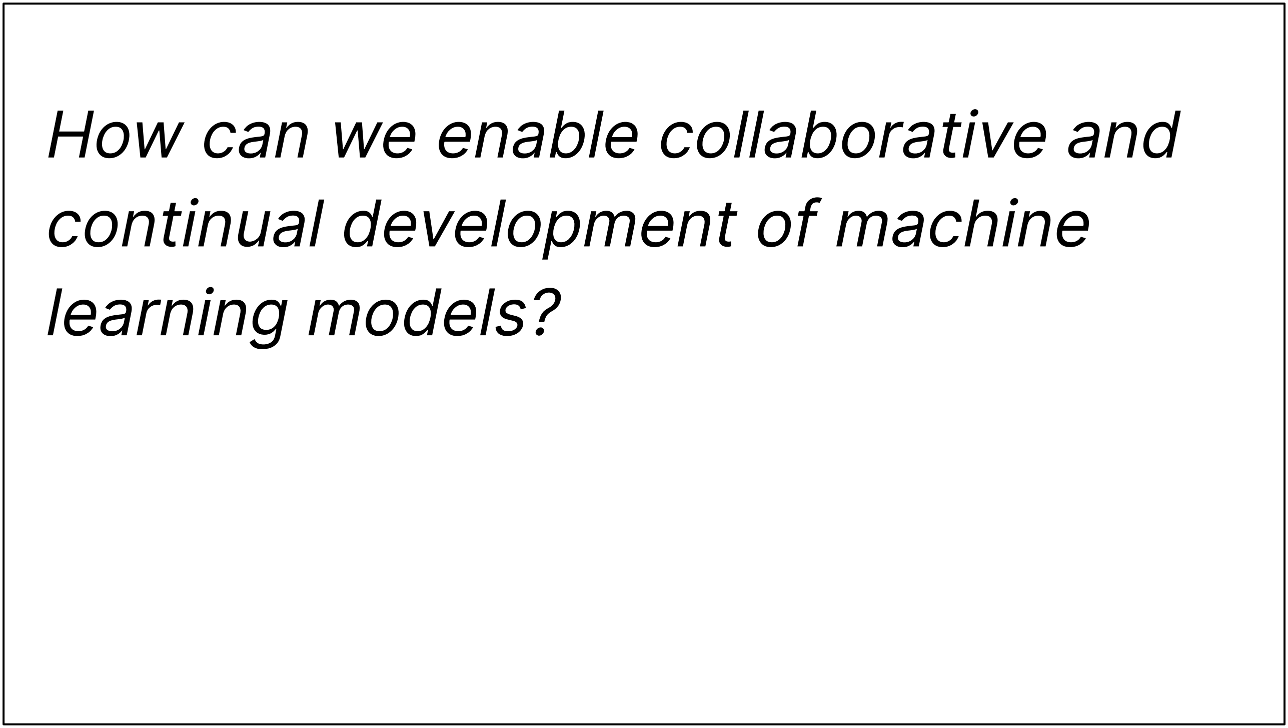
mrm8488/t5-base-finetuned-question…
↓ 438k • ♡ 58

snrspeaks/t5-one-line-summary
↓ 1.28M • ♡ 23

mrm8488/t5-base-finetuned-common_g…
↓ 302k • ♡ 19

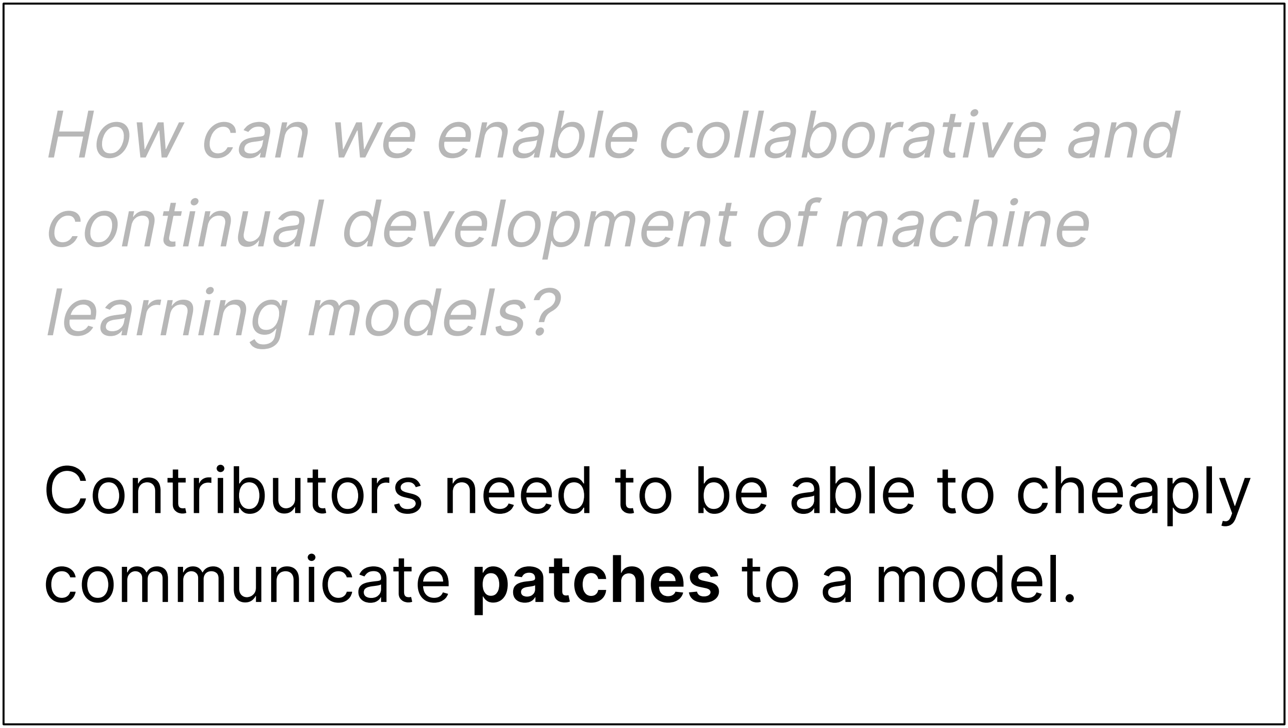
In fact, pre-trained models often are updated or reused to expand their capabilities or make them applicable in new settings. Here I'm showing a sort of family tree of the T5 pre-trained model that I created with my collaborators at Google some years ago. T5 has been reused in many models, and the relationships between these models can be quite complex - for example, the T5 recipe was re-used to create T5.1.1, which underwent an adaptation step before becoming the popular instruction-tuned models Flan-T5 and T0. T5's encoder has also been reused in various text-to-image models like PaLI, Imagen, and Muse. And finally, T5 has frequently been adapted to downstream tasks via further training - there are thousands of fine-tuned variants of T5 on the Hugging Face model hub. So, clearly there are people who are motivated to update and improve a pre-trained model. **However, I only am able to draw this family tree because I pay attention to what people are doing with T5 - there is no explicit or principled way to keep track of the provenance and progression of a given model.**

Contrast this state of affairs with the development of open-source software. As many of you know, open-source softtware underlies much of the technology that we use today. For example, you might use an open-source web-browser to communicate with a server running an open-source web framework written in an open-source programming language running an open-source operating system and connecting to an open-source database system that was compiled with an open-source compiler. **Importantly, all of this software was developed by a distributed community of contributors who were able to collaboratively and iteratively build the software through a mature set of tools, including patching, merging, version control, and more. Compared to the development of open-source software, the development of machine learning models is in the dark ages - no such tooling or concepts exist to enable collaborative and continual development of machine learning models.**



How can we enable collaborative and continual development of machine learning models?

Motivated by this, today I am going to be presenting some preliminary work that aims to answer the following question - how can we enable collaborative and continual development of machine learning models, in the way that open-source software is built?

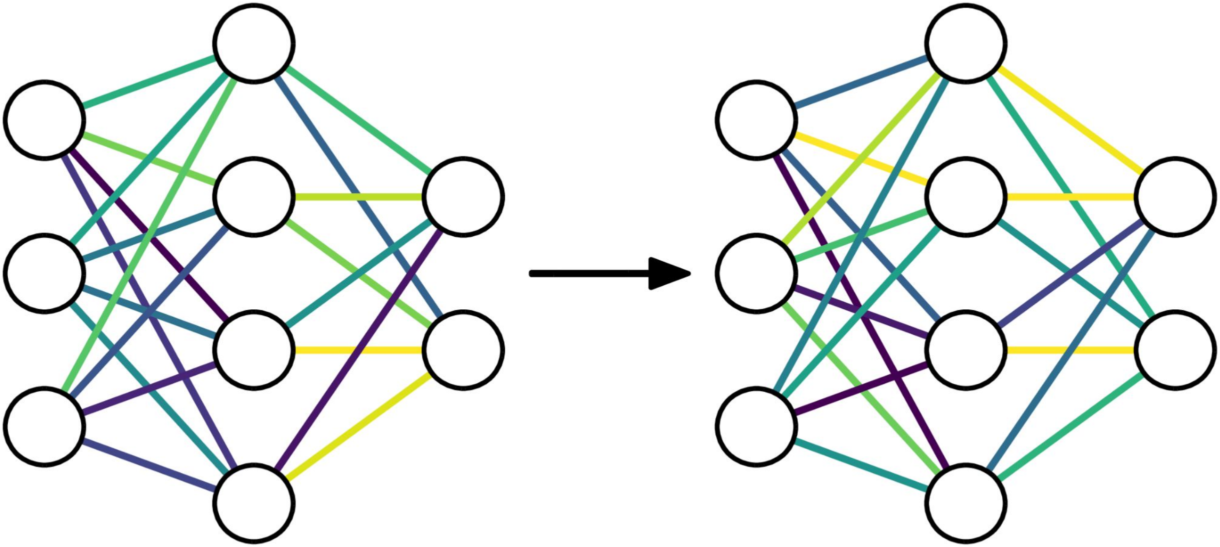


How can we enable collaborative and continual development of machine learning models?

Contributors need to be able to cheaply communicate **patches** to a model.

First and foremost, contributors to a given model need to be able to cheaply communicate patches to a model. In open-source software, patches allow contributors to propose small, focused changes that meaningfully improve the software. How can we enable patching of models?

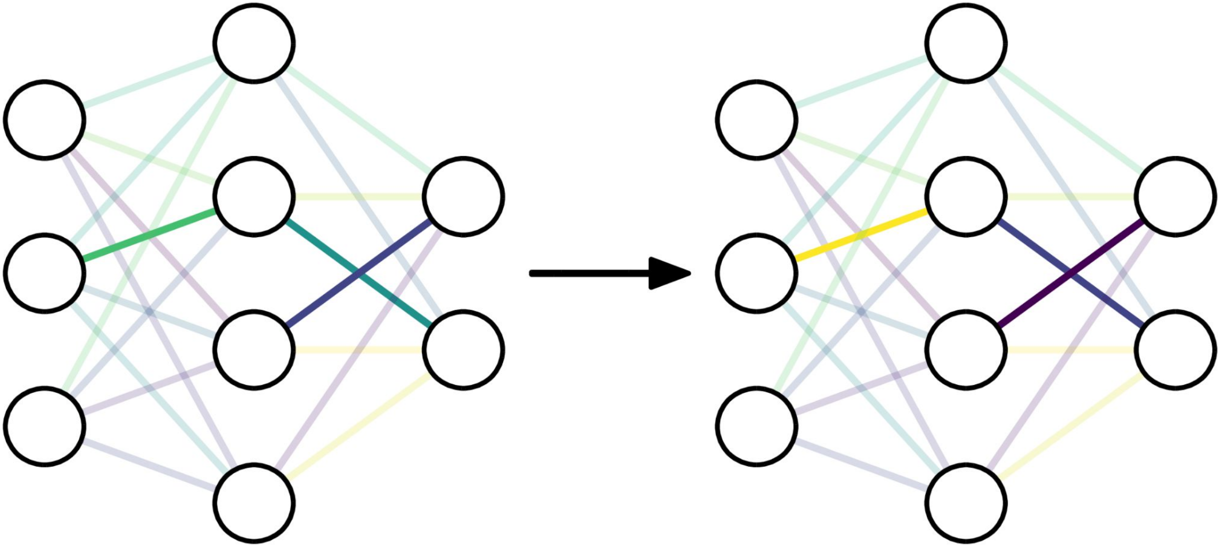
Gradient descent creates a new set of parameters at every iteration



From "Training Neural Networks with Fixed Sparse Masks" by Sung et al.

Today, most neural networks are trained via gradient descent, which updates every parameter of the model by moving in the negative of the gradient direction with respect to a loss. Crucially, gradient descent updates every parameter at every training iteration, so any amount of training will result in an entirely new set of parameters. This would mean that any improvement to the model through training would require communicating and storing an entirely new set of model parameters. Modern pre-trained model checkpoints can often be tens of gigabytes in size, so this could get infeasible quickly.

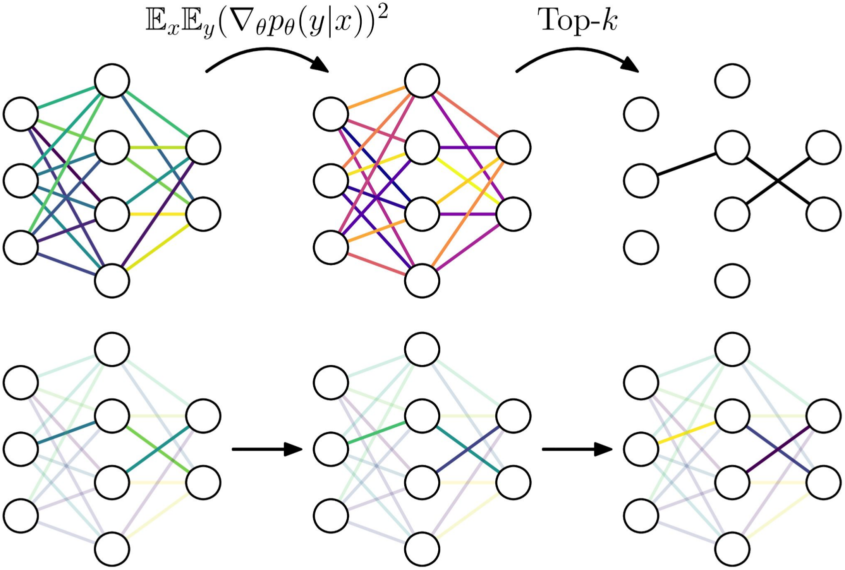
Updating a subset of parameters reduces communication costs



From "Training Neural Networks with Fixed Sparse Masks" by Sung et al.

To motivate our work on patching, consider the possibility that instead of updating all of the model's parameters at every training iteration, we only update a subset. If the size of this subset is sufficiently small, we can significantly reduce communication and storage costs. Ideally, we would identify a subset of parameters that could be updated while producing the same performance of full-model training. So, the main question is how to identify such a subset.

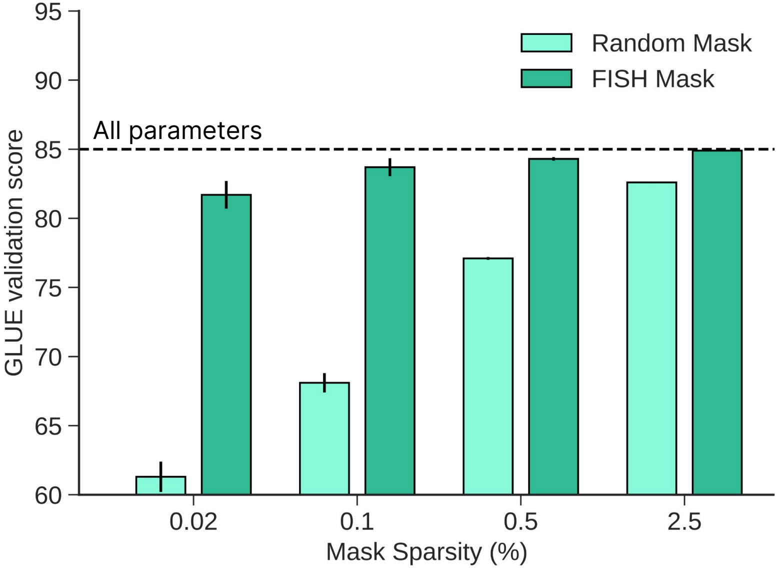
FISH Mask uses the Fisher information to choose a parameter subset



From “Training Neural Networks with Fixed Sparse Masks" by Sung et al.

To choose a subset of parameters, we make use of a quantity called the Fisher information, or **specifically a diagonal approximation of the Fisher information matrix**. Each entry in the diagonal essentially measures how much the model's output changes with a given parameter is perturbed. So, if a parameter has a large Fisher value, it means that the model is very sensitive to changes in the parameter's value. We therefore choose the k parameters with the largest entries in the diagonal Fisher as our subset of parameters to update, and then only update those parameters over many iterations of training. **We call this method FISH Mask, because it's a Fisher-Induced Sparse Unchanging mask over the model parameters.**

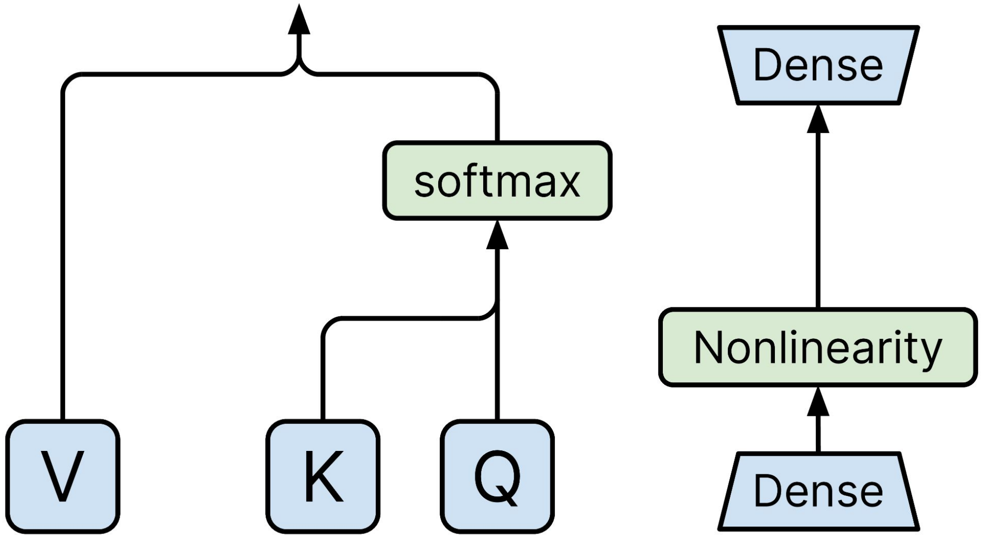
FISH Mask retains performance without updating all parameters



From "Training Neural Networks with Fixed Sparse Masks" by Sung et al.

We tested FISH Mask in various experimental settings, including a common setting called parameter-efficient fine-tuning, where the goal is to update as few parameters as possible during fine-tuning while still attaining good performance. Specifically, we focused on parameter-efficient fine-tuning of the BERT model on the popular GLUE meta-benchmark of NLP datasets. Notably, FISH Mask retains the performance of updating all parameters while only updating 2.5% of model parameters, and attains good performance with as few as 0.1% of model parameters. Importantly, choosing a subset of parameters using FISH Mask rather than choosing them at random leads to a significant improvement in performance, which validates our use of the Fisher information in FISH Mask.

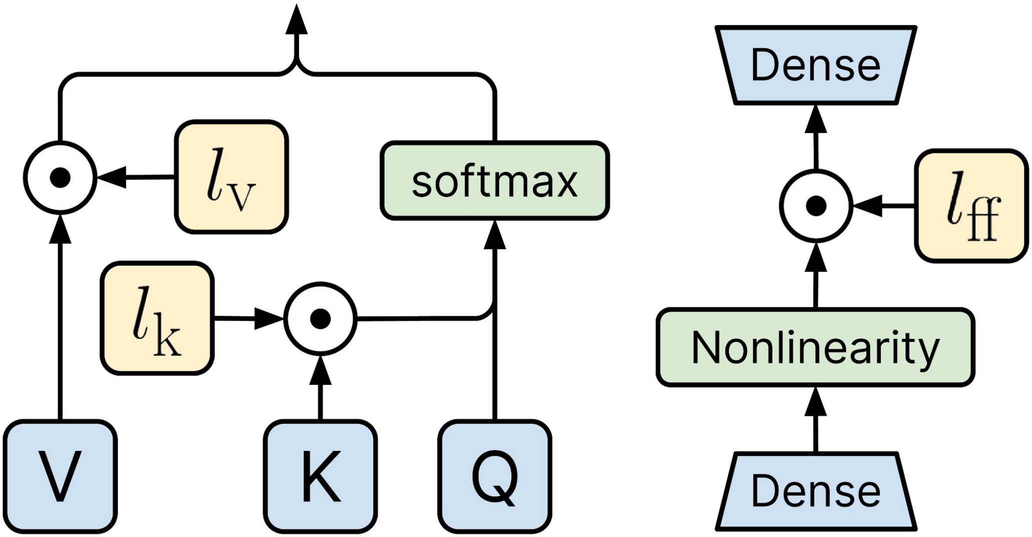
Sublayers in the Transformer architecture



From “Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning" by Liu et al.

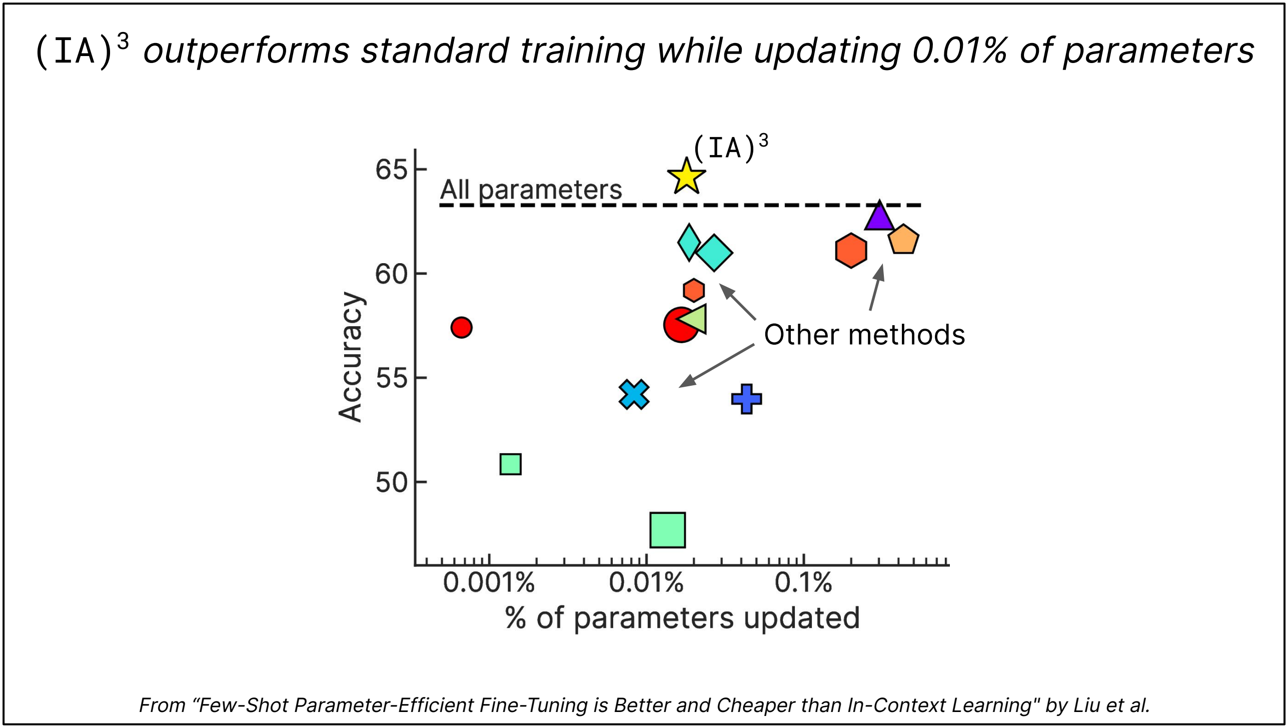
FISH Mask is an architecture-agnostic method that can be applied to any model. In subsequent work, we investigated whether we could design a better parameter-efficient update method that was targeted to a specific model architecture -- the Transformer. Here I've visualized two subnetwork blocks that are applied repeatedly in the Transformer, the self-attention on the left and the feed-forward network on the right.

(IA)^3 = *Infused Adapter by Inhibiting and Amplifying Inner Activations*



From "Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning" by Liu et al.

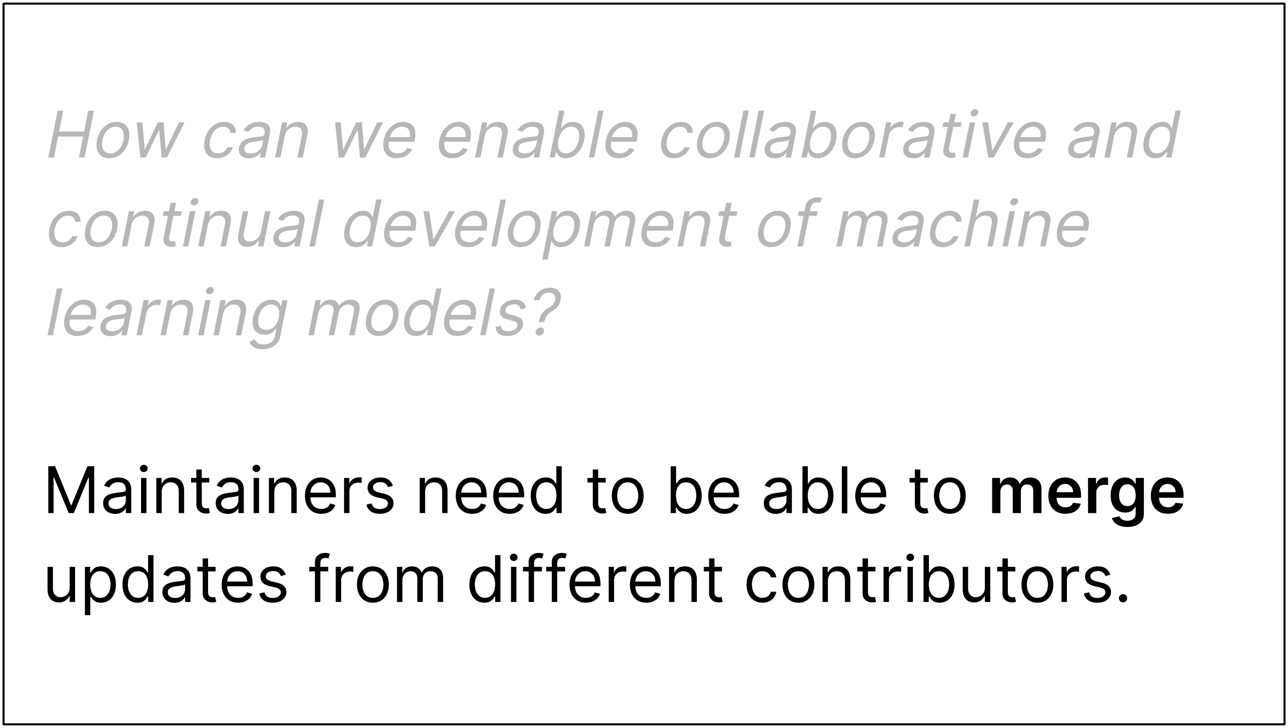
To design a more parameter-efficient updating method, we experimented with adding learned vectors that are multiplied elementwise by the activations or intermediate values within each of these layer blocks. **Since the dimensionality of the activations is much smaller that the dimensionality of the weight matrices used to produce them, adding these learned vectors only introduces a very small number of parameters.** We call this method IA3, for infused adapter by inhibiting and amplifying inner activations. Empirically we found that updating the learned vectors in IA3 was an effective way of updating a model.



(IA)^3 *outperforms standard training while updating 0.01% of parameters*

From “Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning" by Liu et al.

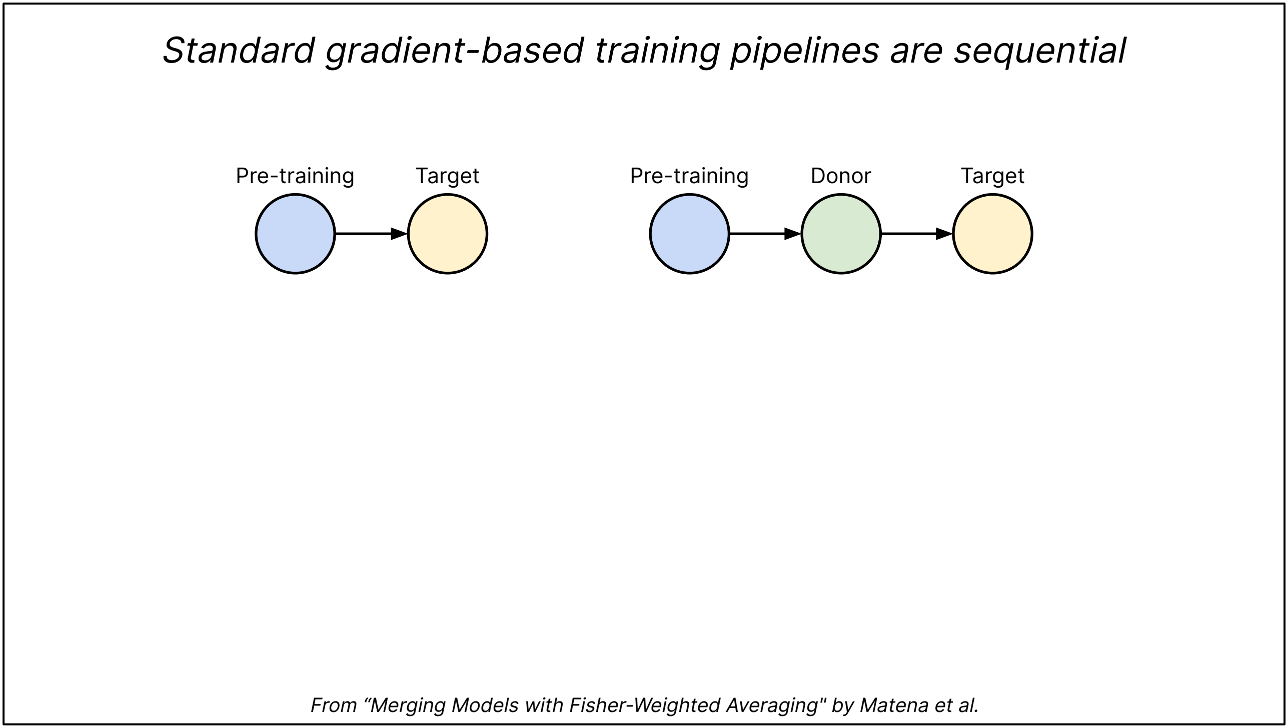
Specifically, we again focused on parameter-efficient fine-tuning, but this time with the T5 model and on a diverse set of **few-shot datasets that contain very small training sets**. Notably, we found that training with IA3 actually outperformed full-model training in this setting, despite only updating 0.01% of the full model's parameters. IA3 also outperformed every other parameter-efficient training method we experimented with. Communicating the IA3 vectors is much more efficient than communicating the full model - it would only require a few megabytes for a model with tens of billions of parameters. This, coupled with the fact that it can effectively update a model, makes it a promising method for patching models.



How can we enable collaborative and continual development of machine learning models?

Maintainers need to be able to **merge** updates from different contributors.

Ok, so now we can patch models. This means that you could take a model and update it, I could in parallel take a model and update it, and now we have conflicting updates that were produced in parallel to the same version of the model. In version control, this is called a merge conflict, and version control software has sophisticated functionality for handling merges. To enable collaborative development of machine learning models, we need to be able to merge models too.



Standard gradient-based training pipelines are sequential

Pre-training

Target

Pre-training

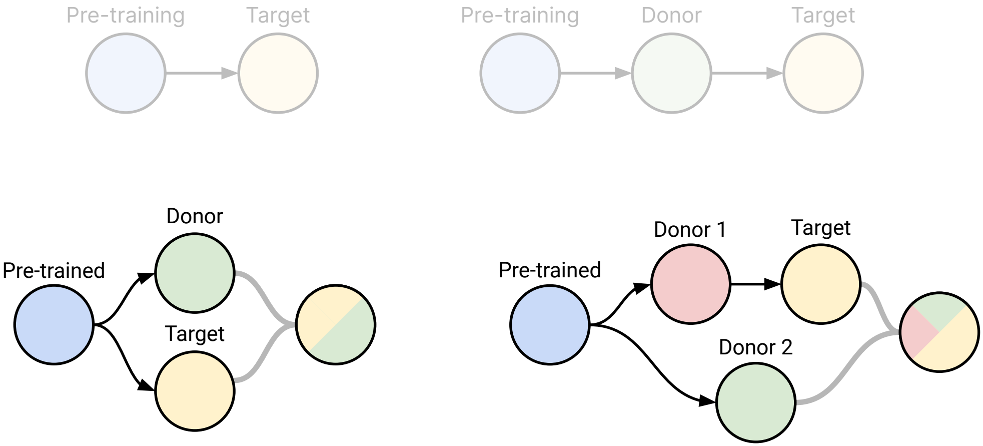
Donor

Target

From “Merging Models with Fisher-Weighted Averaging" by Matena et al.

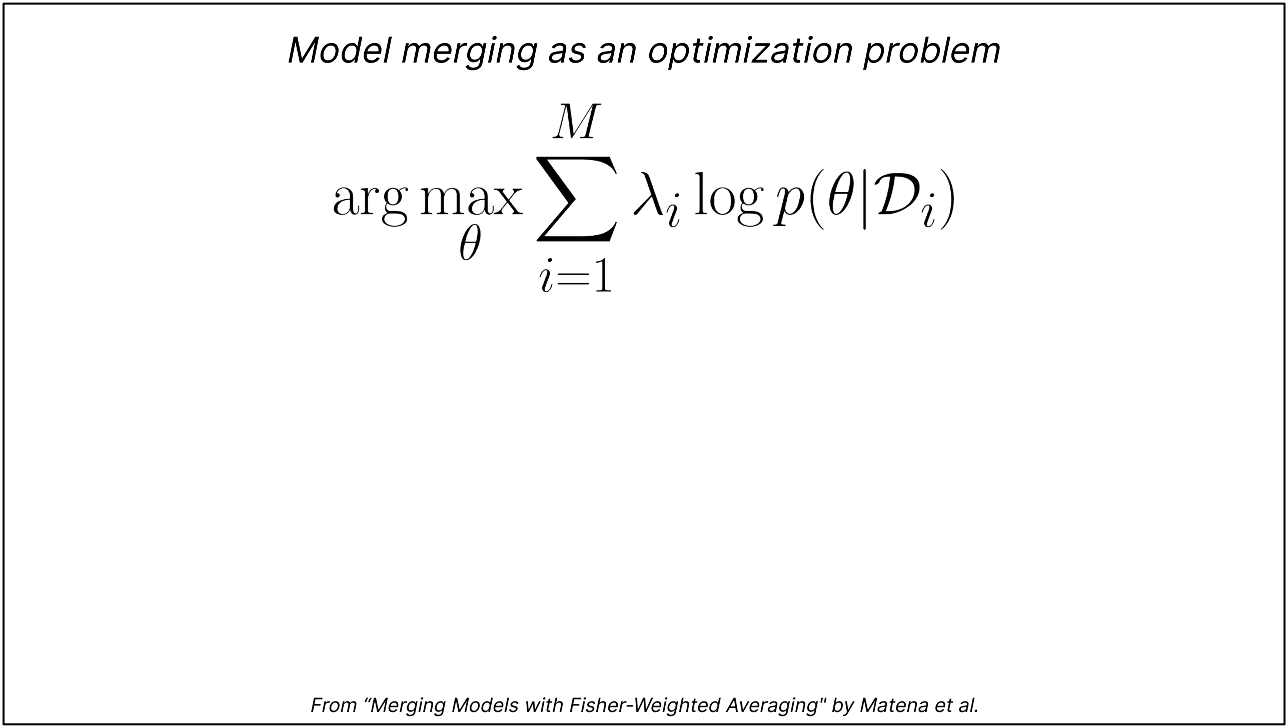
To think about what merging might look like, I'll first point out that standard training pipelines look something like what's shown here - we take a model and perform additional training, then perhaps perform some more training, and so on. For example, on the right is what's called intermediate-task training, where a pre-trained model is first trained on a donor task before being trained on a target task. The hope is that training on an appropriate donor task can improve target-task performance. In order to do so, the full training pipeline needs to be totally sequential - each model depends on exactly one prior model.

Merging models would enable new paths for transferring capabilities



From “Merging Models with Fisher-Weighted Averaging" by Matena et al.

Merging would allow for other training pipelines. For example, we could replicate something like intermediate-task training by taking a pre-trained model, training it on the donor and target tasks in parallel, and then merging the donor and target models together in hopes of improving the performance on the target task. We also could experiment with training pipelines that would onerous or impossible to achieve using standard sequential training. For example, on the right is what you might call doubly intermediate task training, where a pre-trained model is first trained on one donor task, then on a target task, then in parallel the pre-trained model is trained on another donor task, and now we want to merge the second donor model with the intermediate-task-trained target model to improve performance even more on the target task.



Model merging as an optimization problem

$$\arg\max_{\theta} \sum_{i=1}^{M} \lambda_i \log p(\theta|\mathcal{D}_i)$$

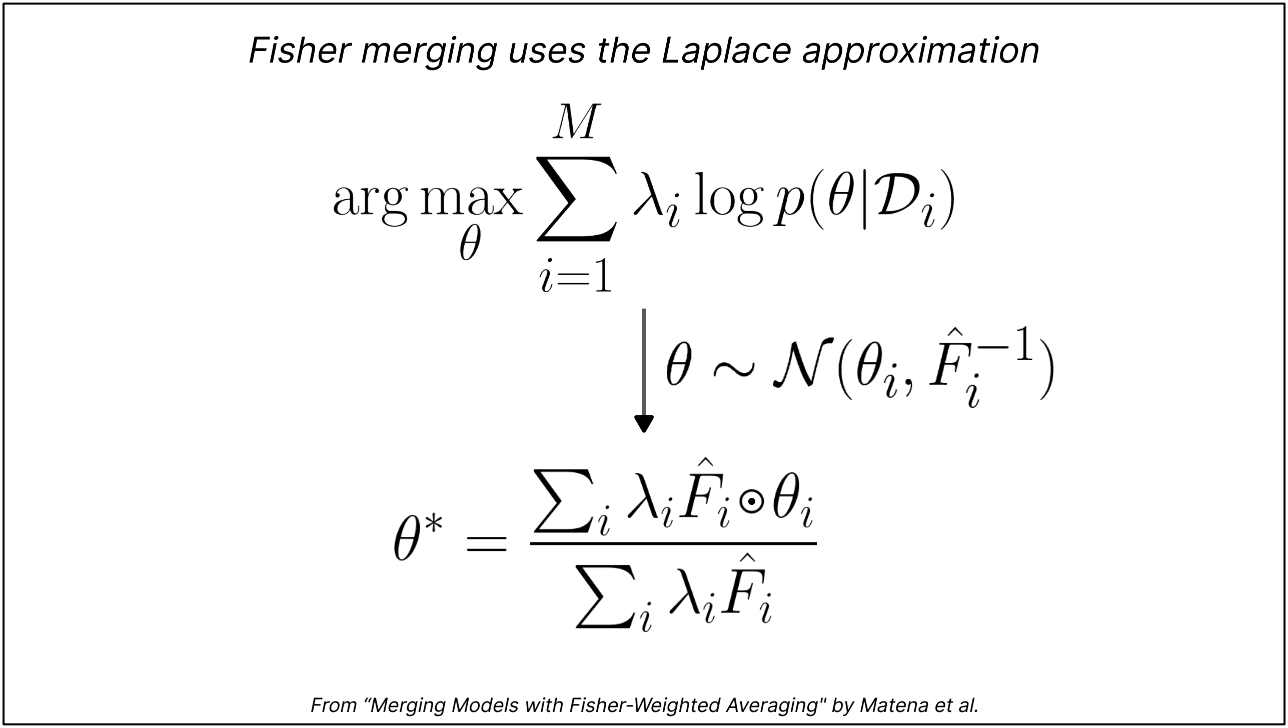
From “Merging Models with Fisher-Weighted Averaging" by Matena et al.

How could we perform such a merging operation? Well, we can formulate it as the following optimization problem. Let me explain the pieces here.

$$\arg\max_{\theta} \sum_{i=1}^{M} \underset{\substack{\uparrow \\ \textit{Hyperparameter} \\ \textit{controlling the} \\ \textit{importance of model } i}}{\lambda_i} \overbrace{\log p(\theta|\mathcal{D}_i)}^{\substack{\textit{Log posterior} \\ \textit{for model } i}}$$

From “Merging Models with Fisher-Weighted Averaging" by Matena et al.

We are looking for a single set of parameters theta that maximizes the sum of log posteriors of M individual models that are being merged. In other words, we are looking for a set of parameters that is assigned high likelihood under each individual model's posteriors. We can also introduce this lambda_i hyperparameter to manually control the importance of each respective model. Unfortunately, we generally don't have access to the model's posterior distribution under standard maximum likelihood-based training. So we need a way to approximate these posteriors.



Fisher merging uses the Laplace approximation

$$\arg\max_{\theta} \sum_{i=1}^{M} \lambda_i \log p(\theta|\mathcal{D}_i)$$

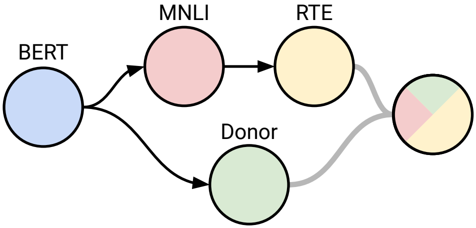
$$\Big\downarrow \; \theta \sim \mathcal{N}(\theta_i, \hat{F}_i^{-1})$$

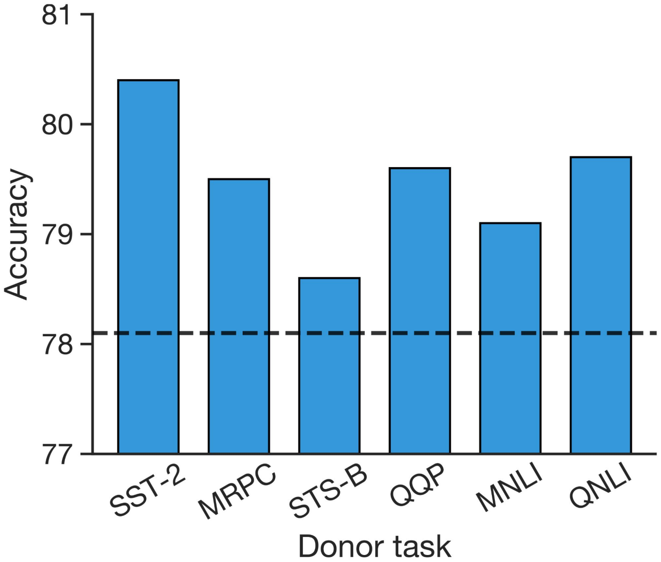
$$\theta^* = \frac{\sum_i \lambda_i \hat{F}_i \odot \theta_i}{\sum_i \lambda_i \hat{F}_i}$$

From “Merging Models with Fisher-Weighted Averaging" by Matena et al.

In our work, we introduce a "Fisher merging" method that approximates the posterior distributions using the Laplace approximation. Laplace approximation takes a **second-order Taylor expansion of the posteriors, which corresponds to assuming that parameter values are Gaussian distributed with the model's Hessian as the precision matrix. Computing and inverting the Hessian is intractable, so we further approximate the Hessian with the diagonal Fisher, which is a good approximation to the Hessian at a mode of the posterior.** If we make this approximation, the optimization problem has a closed form solution shown on the bottom - basically, we take a weighted average of each of the individual model's parameters, where the lambda_i hyperparameters form model-level weights and the diagonal Fisher information matrices produce parameter-level weights. Thinking about the Fisher values as measuring parameter importance, we can interpret this as upweighting a given parameter on a particular model if the parameter is important to that model.

Fisher merging can combine the capabilities of different models

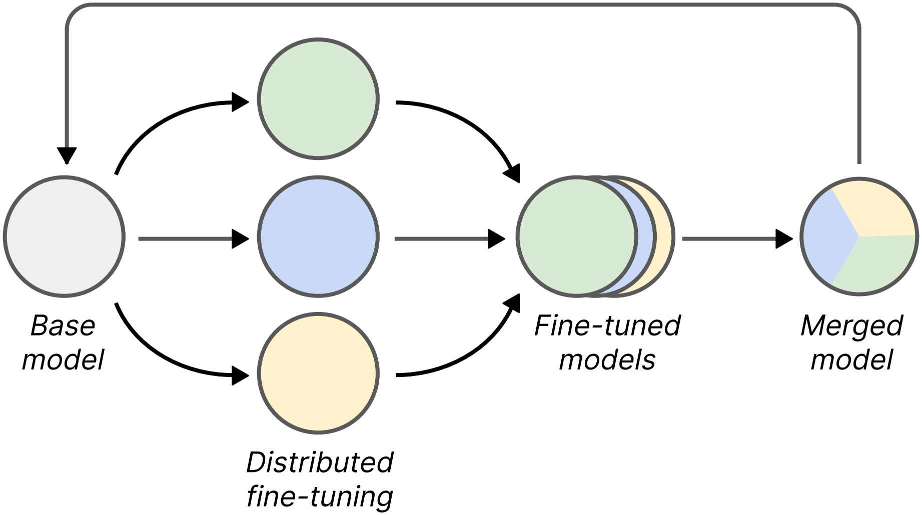




From “Merging Models with Fisher-Weighted Averaging" by Matena et al.

We experimented with Fisher merging in many settings, but here I'm focusing on the doubly intermediate task training setting I mentioned earlier, where the goal is to combine an intermediate-task trained model with an additional donor-task-trained model to boost target-task performance. Specifically, we used BERT as a pre-trained model, the natural language inference dataset MNLI as the first donor task, the smaller natural language inference dataset RTE as the target task, and various other datasets from GLUE as the donor tasks. On the right, you can see the performance attained by merging with different donor models. The performance of the original RTE model is shown via the dashed line, and merging with every donor-task model improved performance over that baseline. We found that if you just inserted the second donor-task training between MNLI and RTE training, the model would lose the boost attained via MNLI intermediate-task training, which is a phenomenon called catastrophic forgetting in continual learning. Merging therefore provides a totally new path to transferring and improving capabilities of a model.

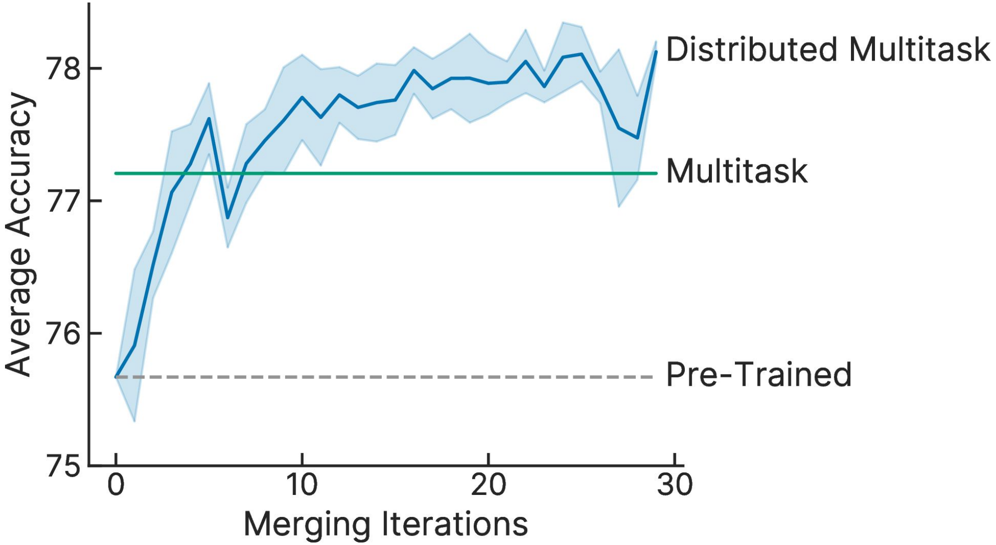
ColD Fusion: Merging fine-tuned models for better pre-trained models



From "ColD Fusion: Collaborative Descent for Distributed Multitask Finetuning" by Don-Yehiya et al.

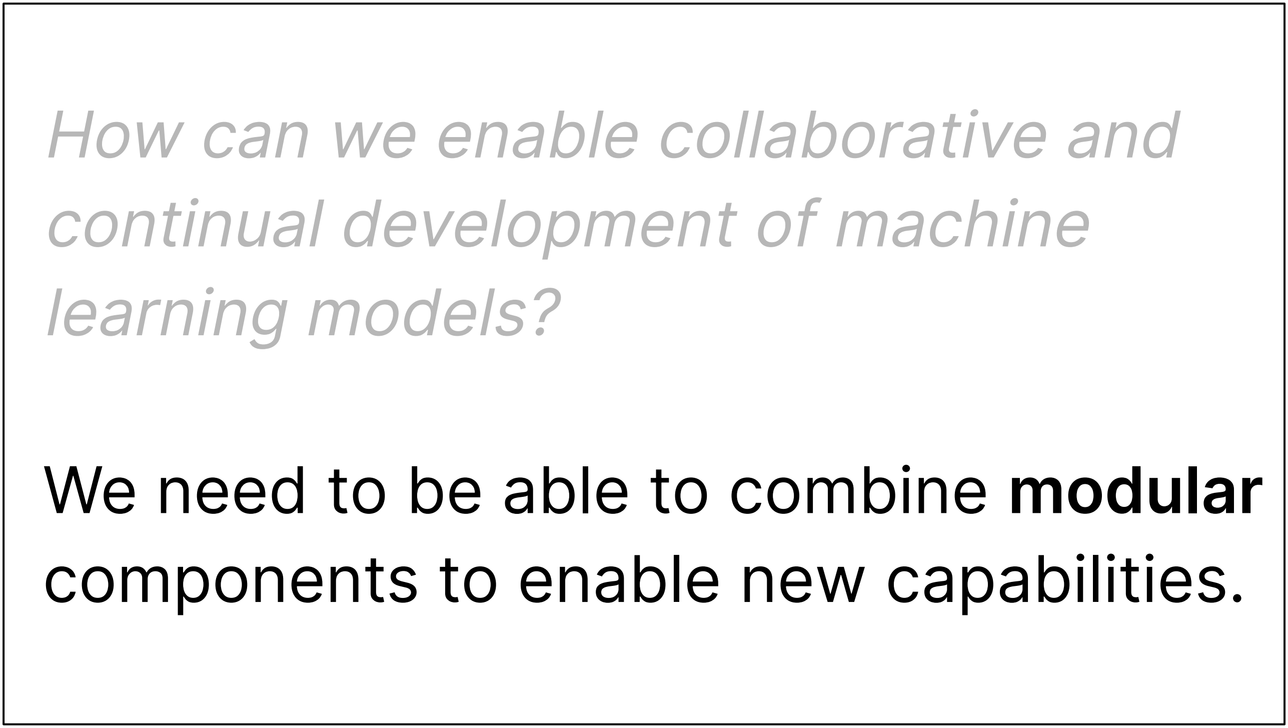
More recently, we showed that merging provides a way to reuse fine-tuned models in order to make a base pre-trained model better. Specifically, we proposed a technique we call ColD Fusion, where a base model is fine-tuned in parallel on diverse datasets, and then the individual fine-tuned models are collected and merged. The resulting merged model is then used as the new base model and the process is repeated over many iterations. Notably, we found that this made the base model better at better for subsequent fine-tuning.

Iterative merging of fine-tuned models improves the pre-trained model



From “ColD Fusion: Collaborative Descent for Distributed Multitask Finetuning" by Don-Yehiya et al.

Specifically, we took a variant of BERT and performed ColD Fusion for many iterations. At each iteration we randomly sampled a few datasets to fine-tune on and merged the resulting fine-tuned models to create a new base model. We found the performance of the base model, when fine-tuning on tasks that were held-out from the ColD Fusion procedure, to get continually better. Interestingly, the resulting model was better than simply fine-tuning on all of the datasets used for ColD Fusion at once, suggesting that there is a benefit to doing fine-tuning in a distributed fashion and merging the resulting models. This result provides a concrete way that fine-tuned models could be recycled for collaborative improvement of a shared base model.

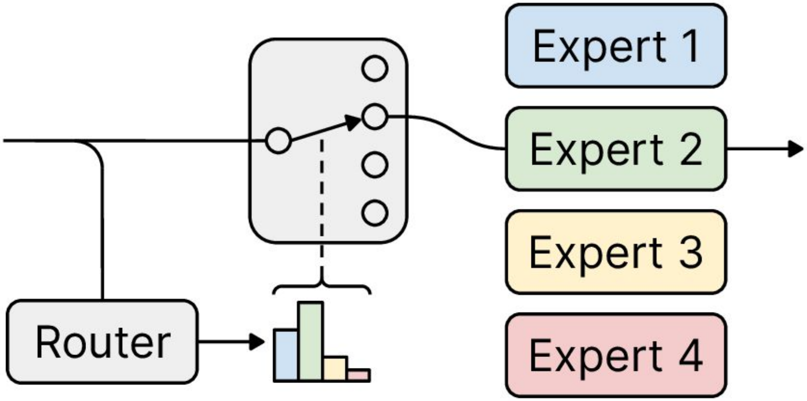


How can we enable collaborative and continual development of machine learning models?

We need to be able to combine **modular** components to enable new capabilities.

A key component of the open-source software ecosystem is the ability to combine different pieces of software to software with new and useful capabilities. This kind of modularity is not currently possible with most machine learning models. We therefore have done some preliminary work on exploring modular model architectures.

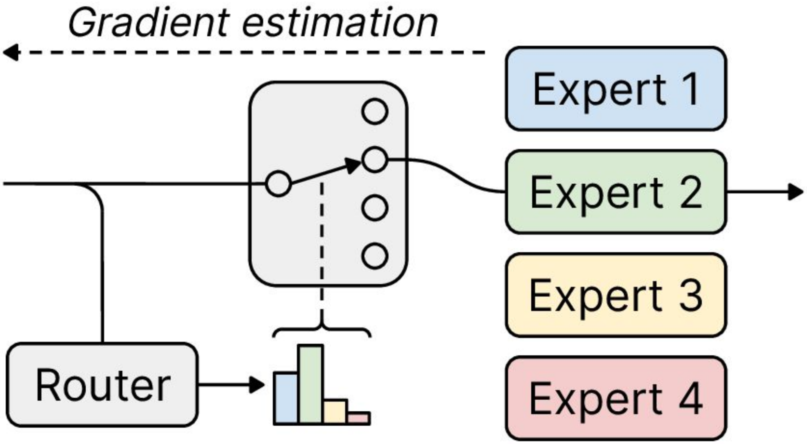
Mixture-of-experts models use specialized, modular subnetworks



From “Soft Merging of Experts with Adaptive Routing" by Muqeeth et al.

One model architecture that enables a form of modularity are mixture-of-expert models, which route activations through specialized subnetworks called experts. The decision of which expert to use is done by a router, which adaptively selects an expert based on its input. Note that a given expert could be added, removed, or updated without affecting the model's behavior when using other experts, which enables a form of modularity.

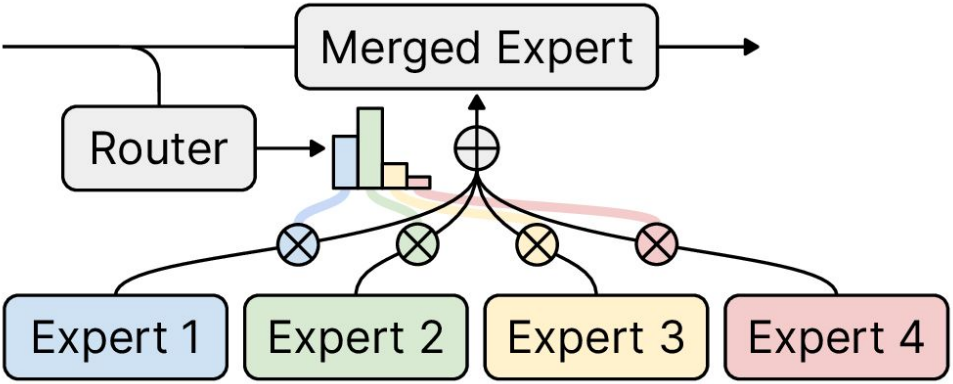
Learned routing requires gradient estimation



From "Soft Merging of Experts with Adaptive Routing" by Muqeeth et al.

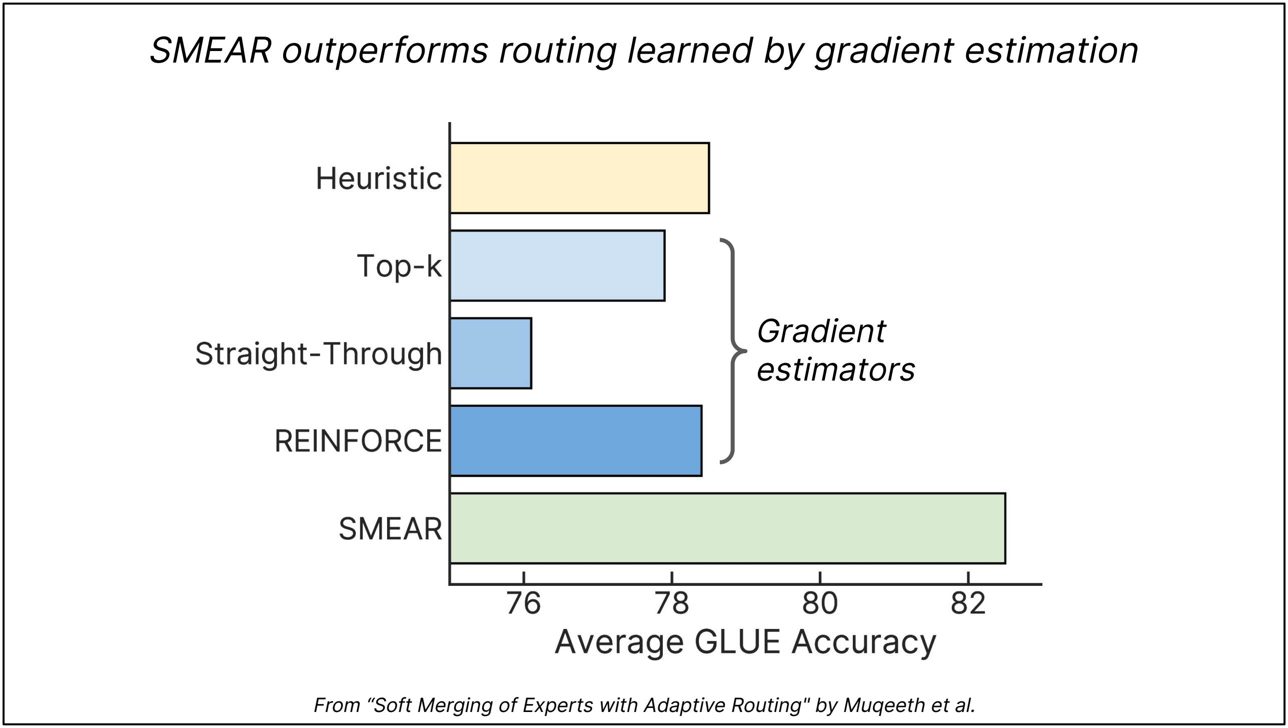
Since the choice of which expert to use is a discrete decision, it's not possible to backpropagate gradients through it. As such, mixture-of-experts models tend to use gradient estimation techniques to perform end-to-end training. Unfortunately, these gradient estimation techniques can be unstable and, as I'll discuss later, can result in models that fail to learn good routing strategies.

Soft Merging of Experts with Adaptive Routing (SMEAR)



From "Soft Merging of Experts with Adaptive Routing" by Muqeeth et al.

In recent work, we proposed an alternative architecture for models with specialized subnetworks that we call SMEAR, which stands for **soft merging of experts with adaptive routing**. Instead of using the router to choose a single expert, we use the router's probability distribution over experts to perform a weighted averaging, or merging, of the individual expert's parameters. Then, we pass the activations through the single merged expert. Since we only process the activations with a single expert, the computational cost remains about the same.

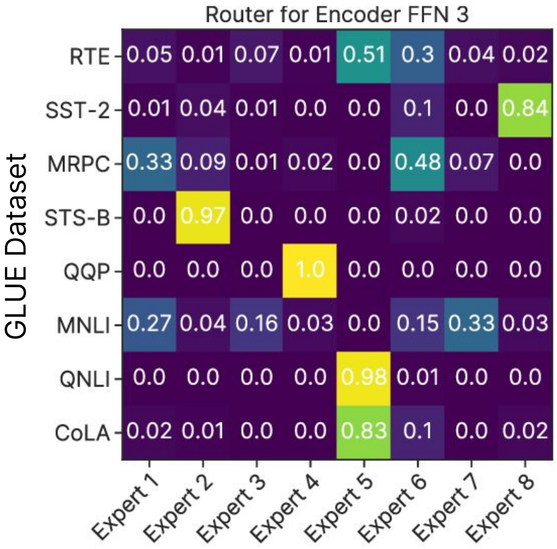


SMEAR outperforms routing learned by gradient estimation

From "Soft Merging of Experts with Adaptive Routing" by Muqeeth et al.

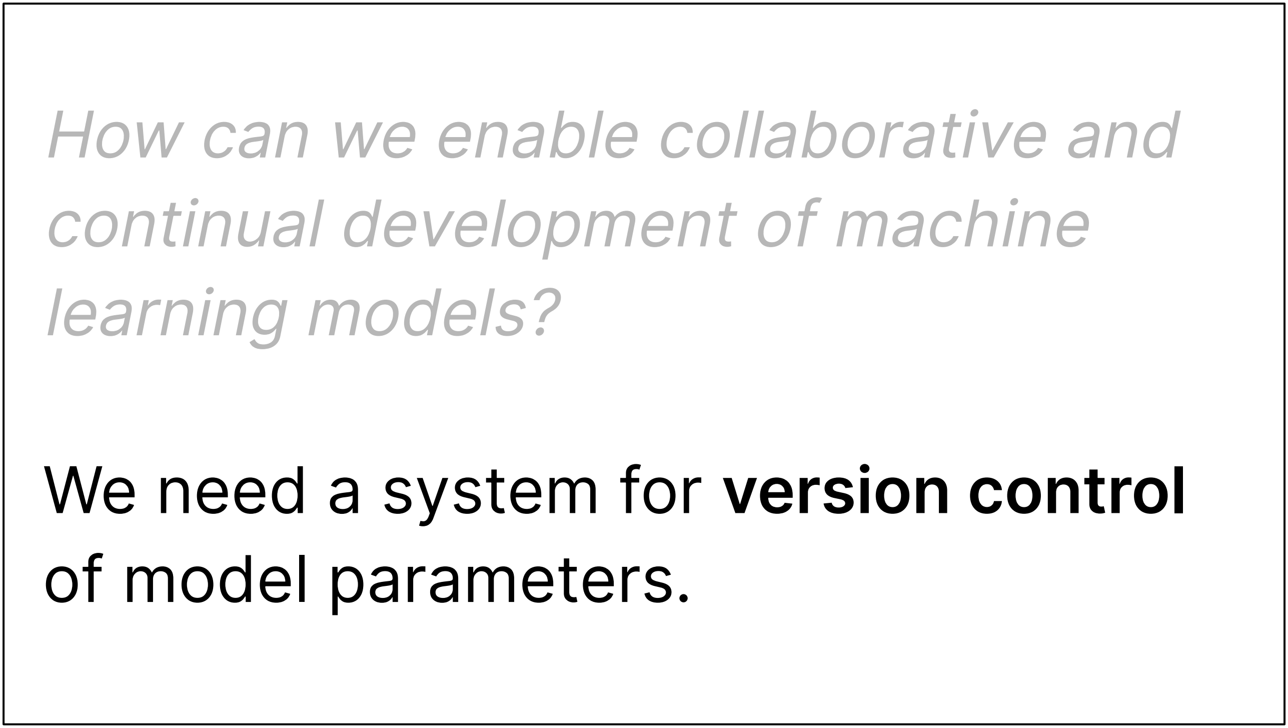
We experimented with SMEAR in settings where we could hand-design a good heuristic routing strategy. For example, in the experiment being shown on the screen here, we considered training T5 on the datasets from GLUE, with the same number of experts as there are datasets in GLUE. That way, a good heuristic routing could be found by simply hard-coding each expert to each datatset. Notably, none of the models trained via gradient estimation outperformed this simple heuristic routing strategy. However, SMEAR-based routing outperformed both the heuristic routing strategy and all those learned via gradient estimation, suggesting that it discovered a better way of specializing the subnetworks.

Individual experts specialize to different types of data



From "Soft Merging of Experts with Adaptive Routing" by Muqeeth et al.

We can actually visualize what each expert specialized to by looking at the proportion of time examples from a given dataset are sent to each expert. What we found was that certain datasets, like SST-2, STS-B, and QQP, were given a dedicated expert like in heuristic routing, but certain datasets like MNLI (which is relatively large and covers many domains) were given multiple experts and certain experts were used for various datasets. We hypothesize that this specialization and sharing of information across subnetworks is what makes SMEAR-based routing work better.

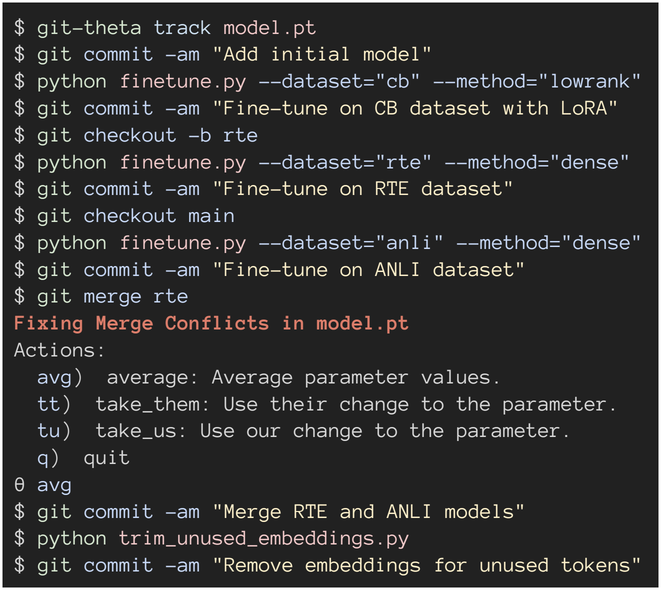


How can we enable collaborative and continual development of machine learning models?

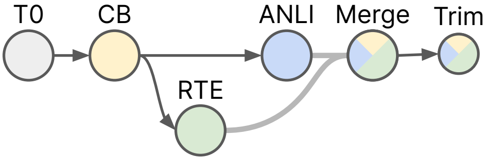
We need a system for **version control** of model parameters.

Now I have demonstrated some of our preliminary work on making patching, merging, and modularity possible. Given these first steps, we're now in a place where we can actually start building a version control system for model parameters that makes use of our work. So, we did.

git-theta tracks, merges, and updates models using the git workflow



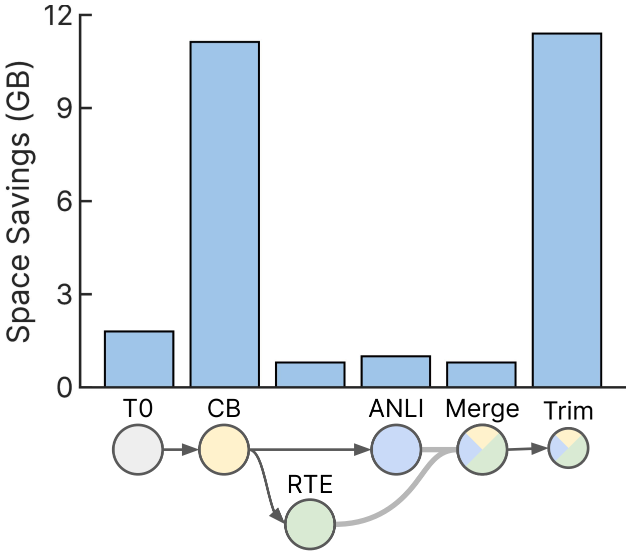
```
$ git-theta track model.pt
$ git commit -am "Add initial model"
$ python finetune.py --dataset="cb" --method="lowrank"
$ git commit -am "Fine-tune on CB dataset with LoRA"
$ git checkout -b rte
$ python finetune.py --dataset="rte" --method="dense"
$ git commit -am "Fine-tune on RTE dataset"
$ git checkout main
$ python finetune.py --dataset="anli" --method="dense"
$ git commit -am "Fine-tune on ANLI dataset"
$ git merge rte
Fixing Merge Conflicts in model.pt
Actions:
  avg)  average: Average parameter values.
  tt)  take_them: Use their change to the parameter.
  tu)  take_us: Use our change to the parameter.
  q)  quit
θ avg
$ git commit -am "Merge RTE and ANLI models"
$ python trim_unused_embeddings.py
$ git commit -am "Remove embeddings for unused tokens"
```



From “Git-Theta: A Git Extension for Collaborative Development of Machine Learning Models" by Kandpal et al.

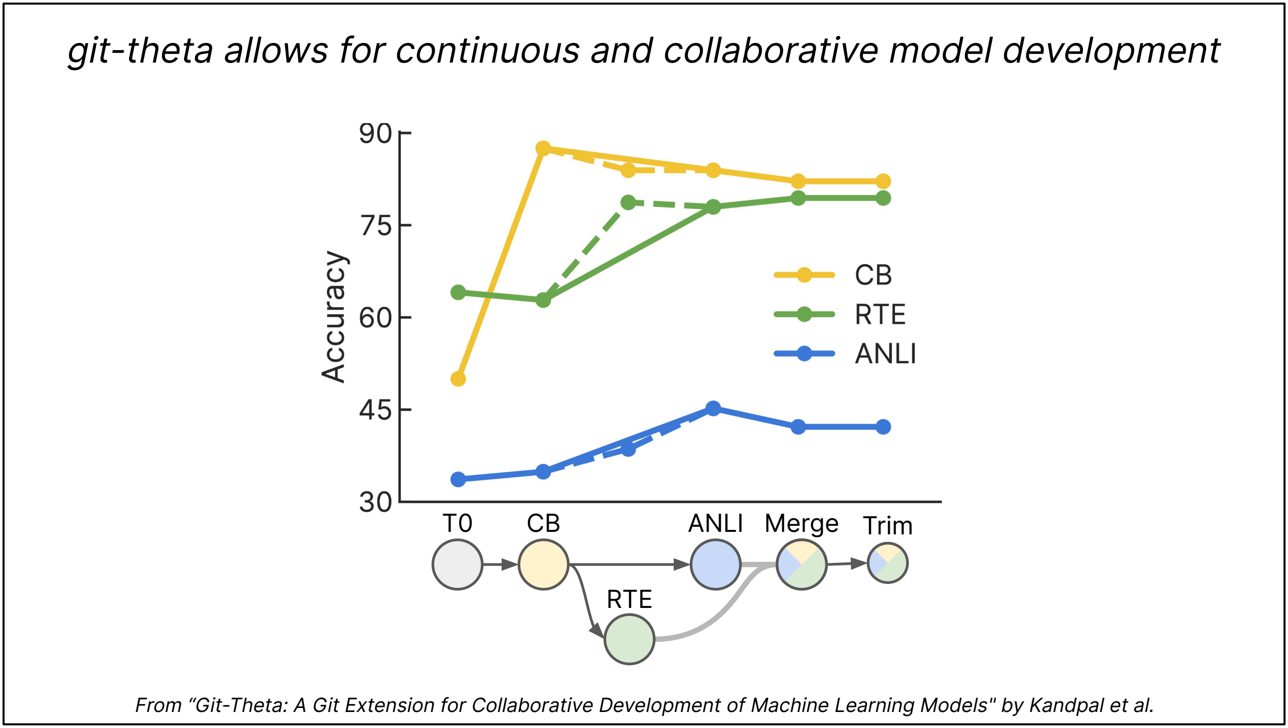
Our system is called git-theta, and it's built on top of git so it integrates with and follows the standard git workflow. Here's an example of how it's used that involves fine-tuning a variant of T5 called T0. The initial T0 checkpoint is first tracked with git-theta, and then the results of subsequent fine-tuning runs can be committed as standard git commits. **Note that if the fine-tuning is done with communication-efficient updates such as a low-rank update, git-theta will store and transmit less data**. Collaborative development can be done via standard branching, and when a merge is necessary git-theta provides automatic functionality for merging. Finally, git-theta natively supports operations like adding or removing parameters.

Communication-efficient updates result in significant space savings



From “Git-Theta: A Git Extension for Collaborative Development of Machine Learning Models" by Kandpal et al.

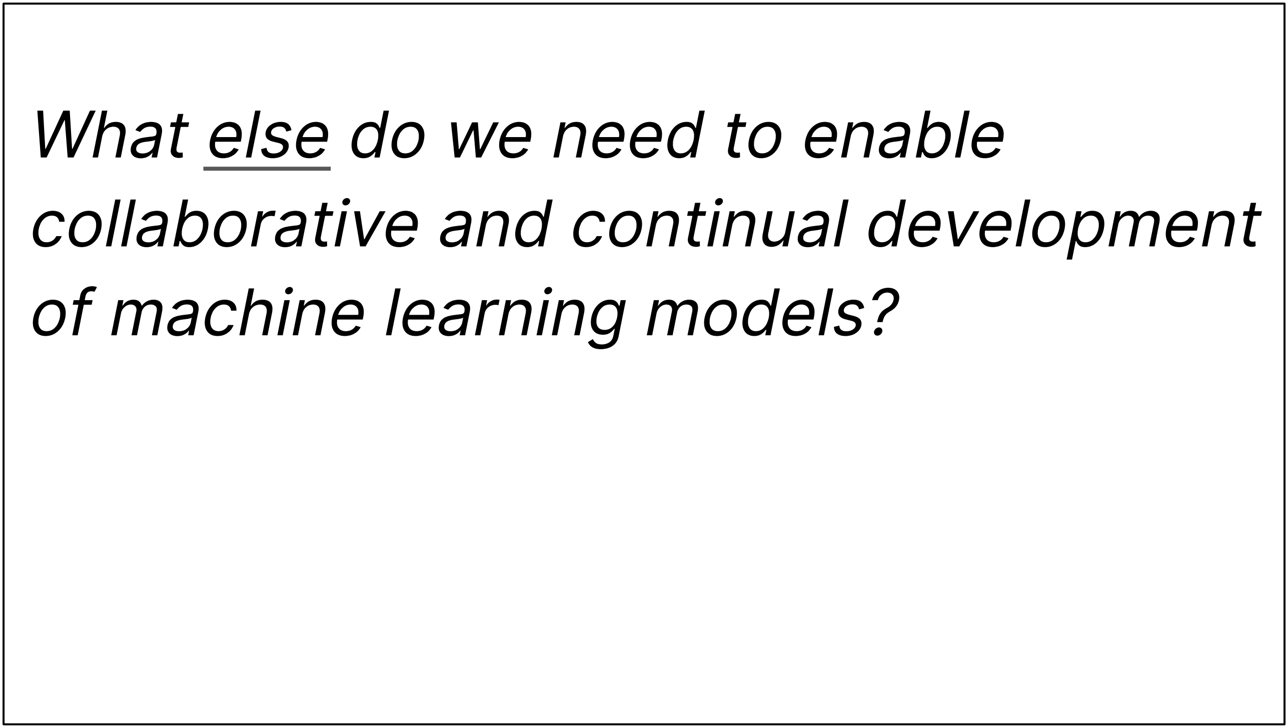
One way to characterize the benefit of git-theta is to measure the amount of space and communication saved compared to naively storing and transmitting a new copy of the model parameters every time the model is changed (which is what existing systems for tracking model checkpoints do). We can see that git-theta actually always saves some space thanks to the way it stores and compresses data, but saves the most space when communication-efficient updates are made or when parameters are removed which is an operation that shouldn't take any new storage space at all.



git-theta allows for continuous and collaborative model development

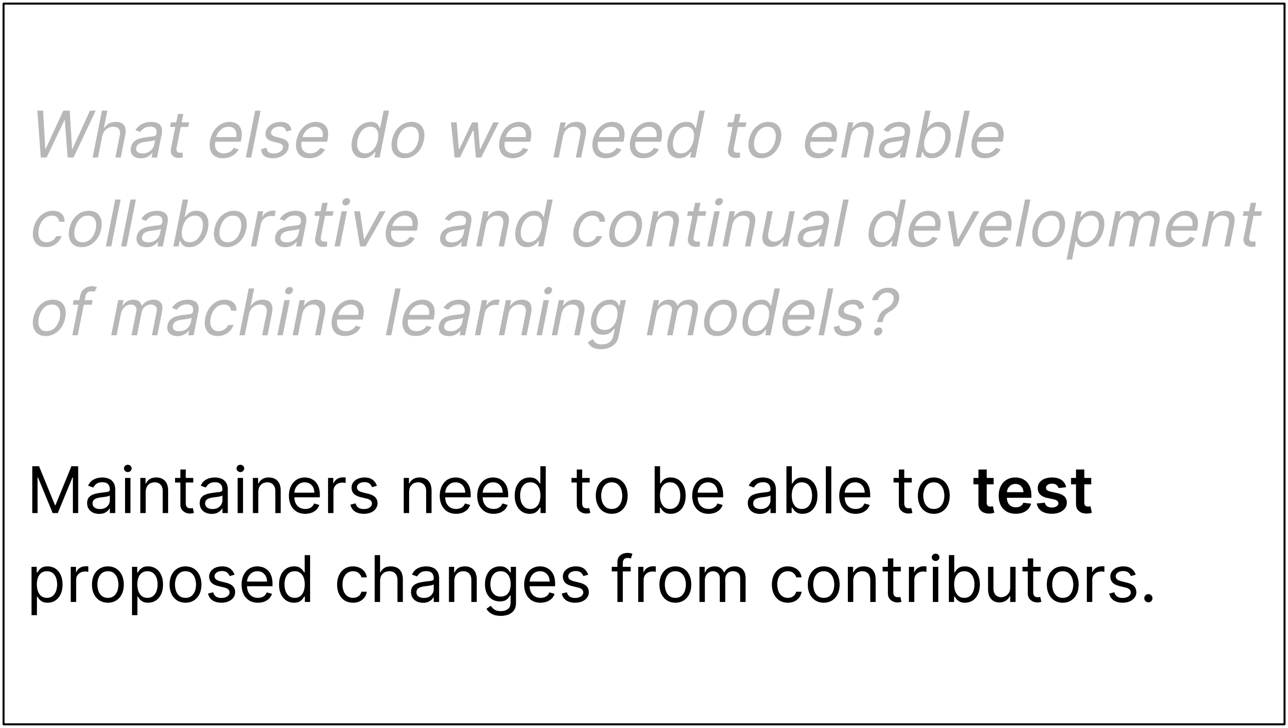
From "Git-Theta: A Git Extension for Collaborative Development of Machine Learning Models" by Kandpal et al.

If we measure the performance of the model over time, we can observe generally increasing performance on the tasks we're targeting. So, we are optimistic that the patching and merging methods I've been discussing could be useful in the real world when combined with git-theta.



What else do we need to enable collaborative and continual development of machine learning models?

Ok, so I've so far mainly been discussing work we've already done in this research direction. What else is left to be done before we enable truly collaborative and continual development of ML models?

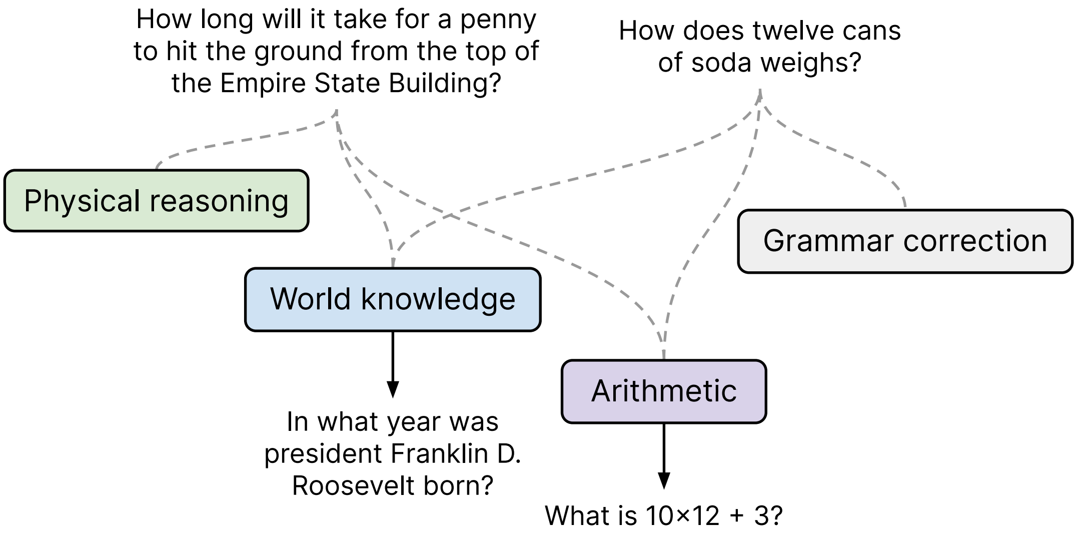


What else do we need to enable collaborative and continual development of machine learning models?

Maintainers need to be able to **test** proposed changes from contributors.

First, if contributors are able to propose changes to a shared model, maintainers of the model need a way to rapidly test the changes to decide whether they should be included.

Decomposing tasks as a composition of atomic subtasks



How long will it take for a penny to hit the ground from the top of the Empire State Building?

How does twelve cans of soda weighs?

Physical reasoning

World knowledge

Arithmetic

Grammar correction

In what year was president Franklin D. Roosevelt born?

What is 10×12 + 3?

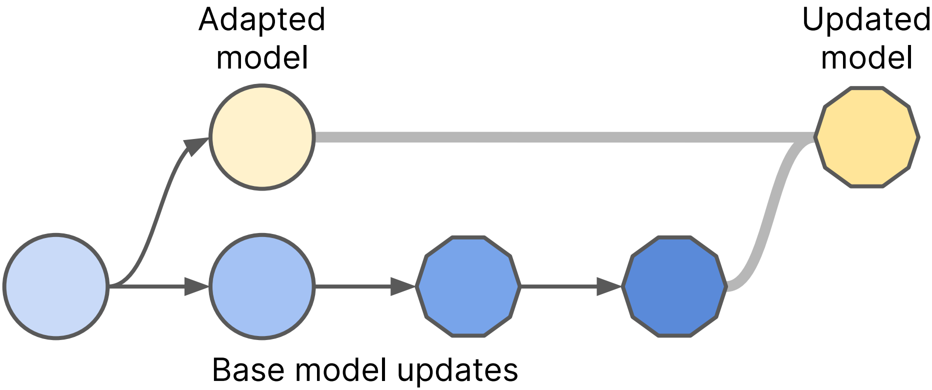
Exhaustively testing a given model on all applicable downstream tasks is likely prohibitively expensive. Instead, testing should be done on a minimal set of subtasks that test high-level capabilities that are used by real-world tasks. For example, given a task that requires physical reasoning, arithmetic, and world knowledge, ideally we should test those individual capabilities, since they will be shared by many other tasks. Determining how to decompose a given task into subtasks and create these subtasks automatically would be an interesting research question.

What else do we need to enable collaborative and continual development of machine learning models?

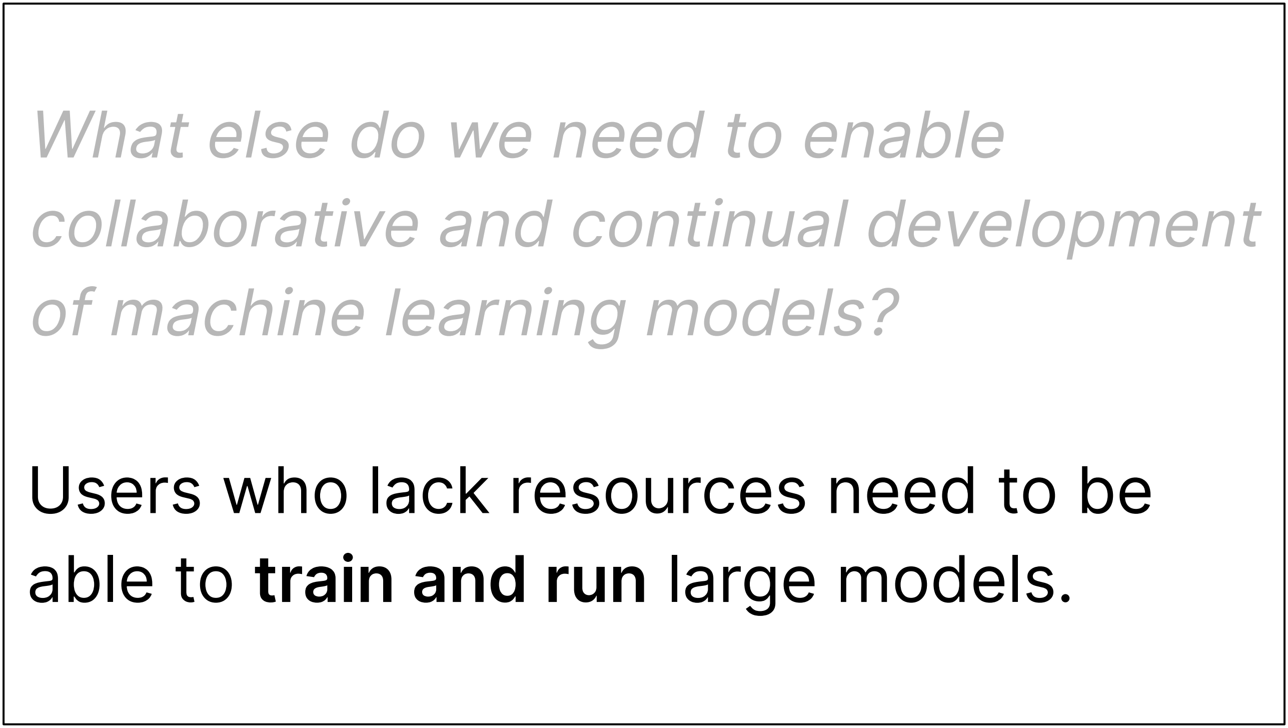
Users need to be able to **rapidly adapt** an updated model to their use-case.

If a base model is continually updated, downstream users could ideally avoid having to completely re-do any adaptation they had previously done. How can we provide a form of backward compatibility to downstream users?

Rapid adaptation as merging with an updated base model



One way would be to treat downstream model updating as merging - we want to merge in the updates from the improved base model into the adapted model. Whether our existing merging method will work out-of-the-box after the base model is significantly changed remains to be seen.

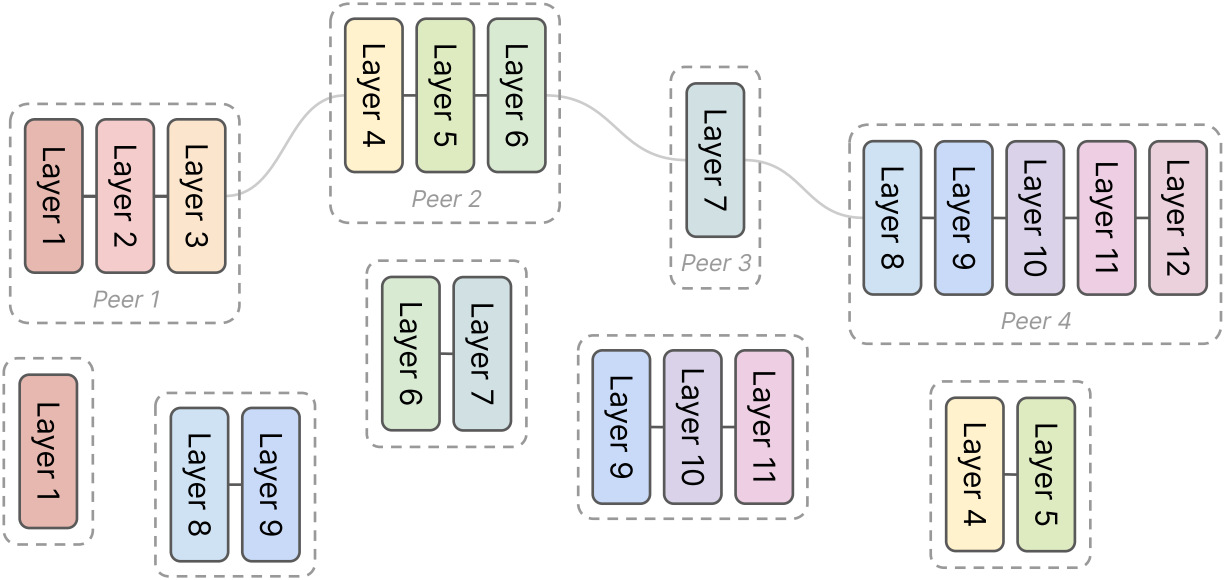


What else do we need to enable collaborative and continual development of machine learning models?

Users who lack resources need to be able to **train and run** large models.

One issue I haven't addressed directly is the fact that many of the most powerful models are currently too large to be developed, let alone used, by many users who lack substantial computational resources. Ideally, as part of enabling collaborative development, we would enable pooling resources to build and use big models too.

PETALS enables distributed inference of large models over the internet



From "Petals: Collaborative Inference and Fine-tuning of Large Models" by Borzunov et al.

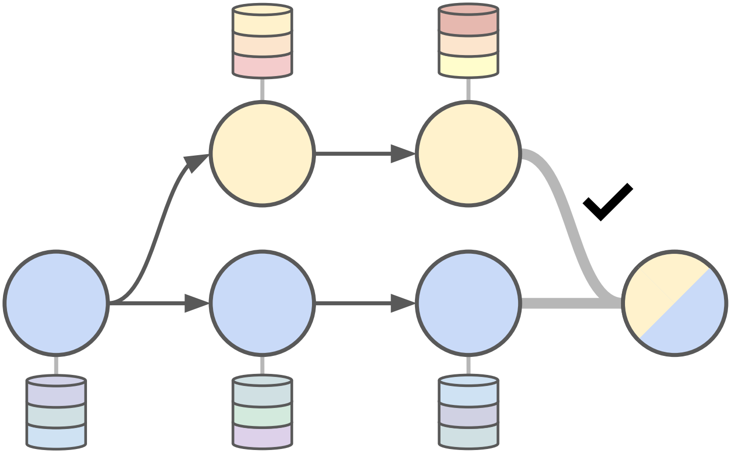
We actually have done some work along these lines with the goal of doing inference of extremely large models on volunteer computing through a system called PETALS. PETALS allows peers to announce that they can run inference for a subset of model parameters and then routes a given request through the optimal set of peers, taking load balancing into account. PETALS also compresses activations to allow them to be sent across the commodity internet. While PETALS is a valuable first step towards using models on distributed computing, it would be prohibitively slow to use for full-model training, so additional work is needed here.

What else do we need to enable collaborative and continual development of machine learning models?

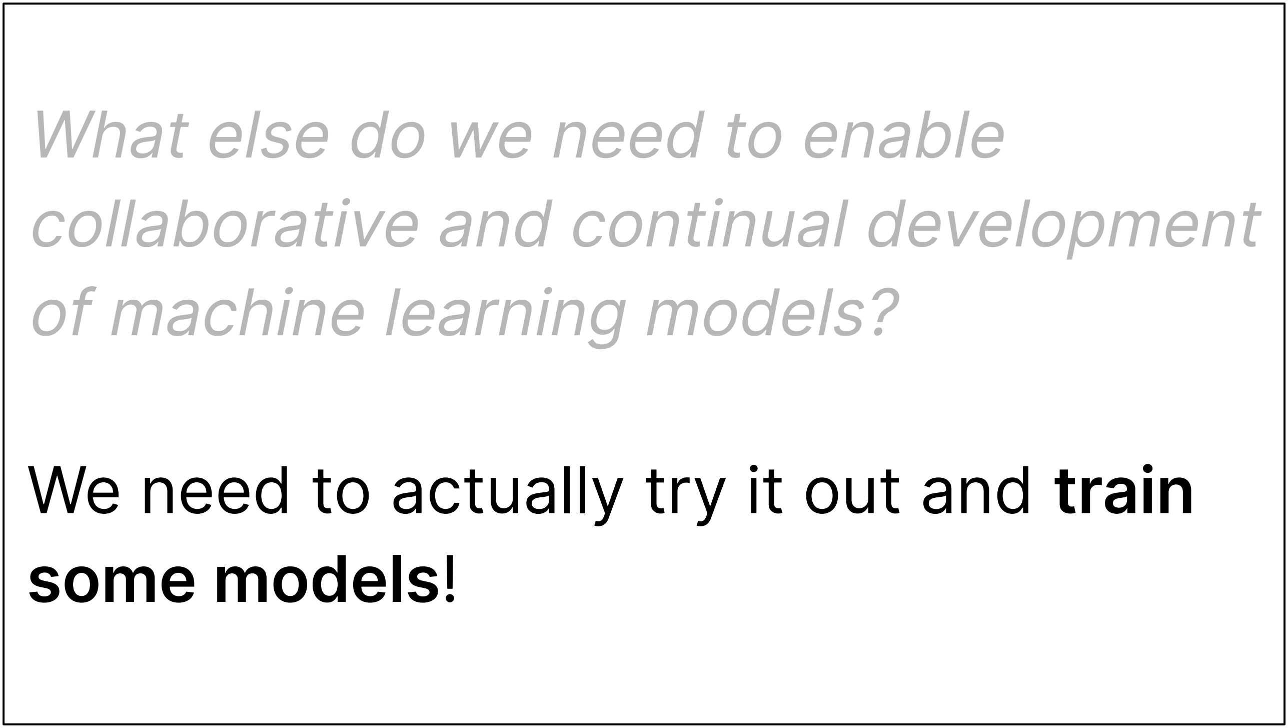
We need to be able to **track the training data** used to train the model.

Opening up contributions from external users opens the possibility that a malicious user could try to insert a backdoor into the model or otherwise train it on problematic data. One way to help mitigate this would be if it was possible to both track the training data used at each stage of development.

Tracking and verifying the data each model was trained on



Determining how best to track the training of the model - both the data and how the model was trained - is an interesting engineering problem for which there are already some solutions. However, a maintainer would also need the ability verify that a model was actually trained on a particular dataset, ideally without exhaustively replicating the training. How to efficiently verify that a particular model was trained on some specific data is an open problem.



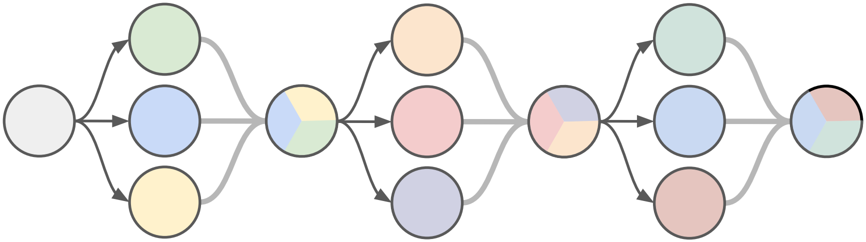
What else do we need to enable collaborative and continual development of machine learning models?

We need to actually try it out and **train some models**!

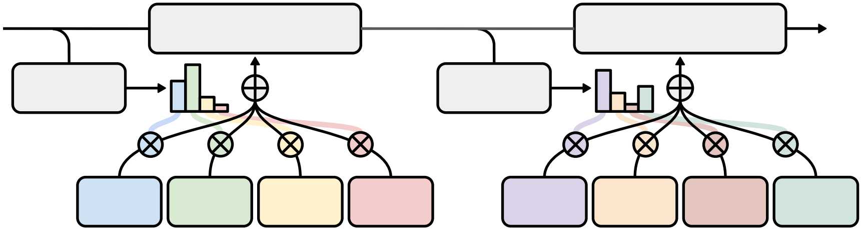
Finally, given that we have done preliminary work on merging and patching and modularity as well as a building version control system, it's time we actually try it all out and build some models!

Building the first collaboratively and continually developed models

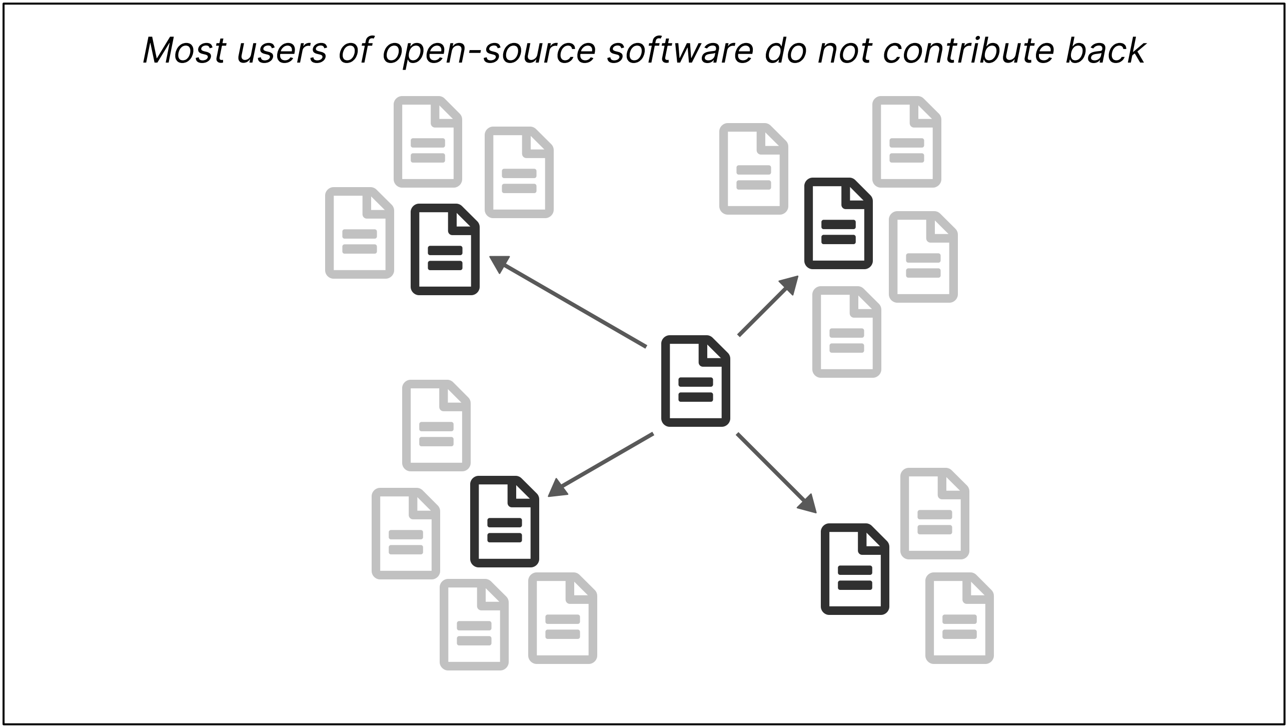
Large-scale continuous CoID Fusion



SMEAR model with specialized and individually updatable experts

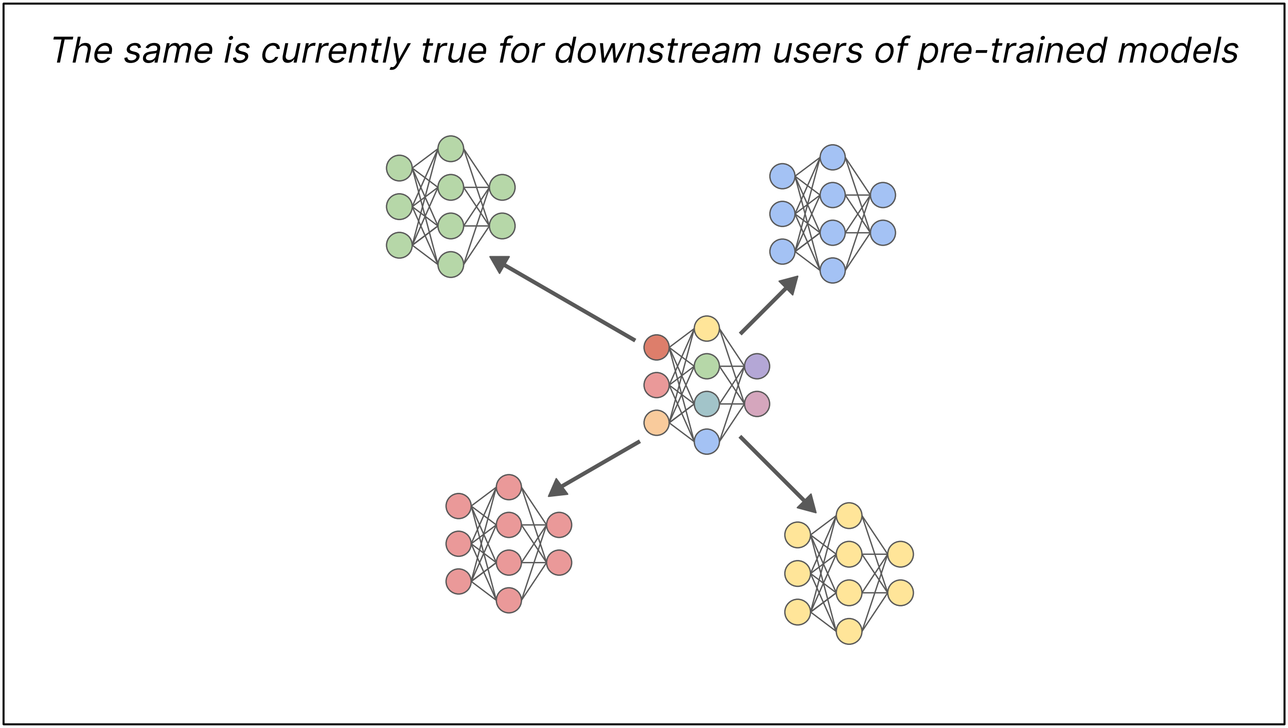


Over the next year, we aim to start developing two classes of models in this way. The first will be models based on the CoID fusion procedure, which will involve many stages of distributed fine-tuning and merging. Ideally, the individual fine-tuning runs could be done as part of routine adaptation of the shared model. The second class of models will be those with specialized subnetworks, trained with SMEAR. In this case, contributors could add, update, or remove a subnetwork without affecting the rest of the model. This would make operations like merging and testing straightforward.



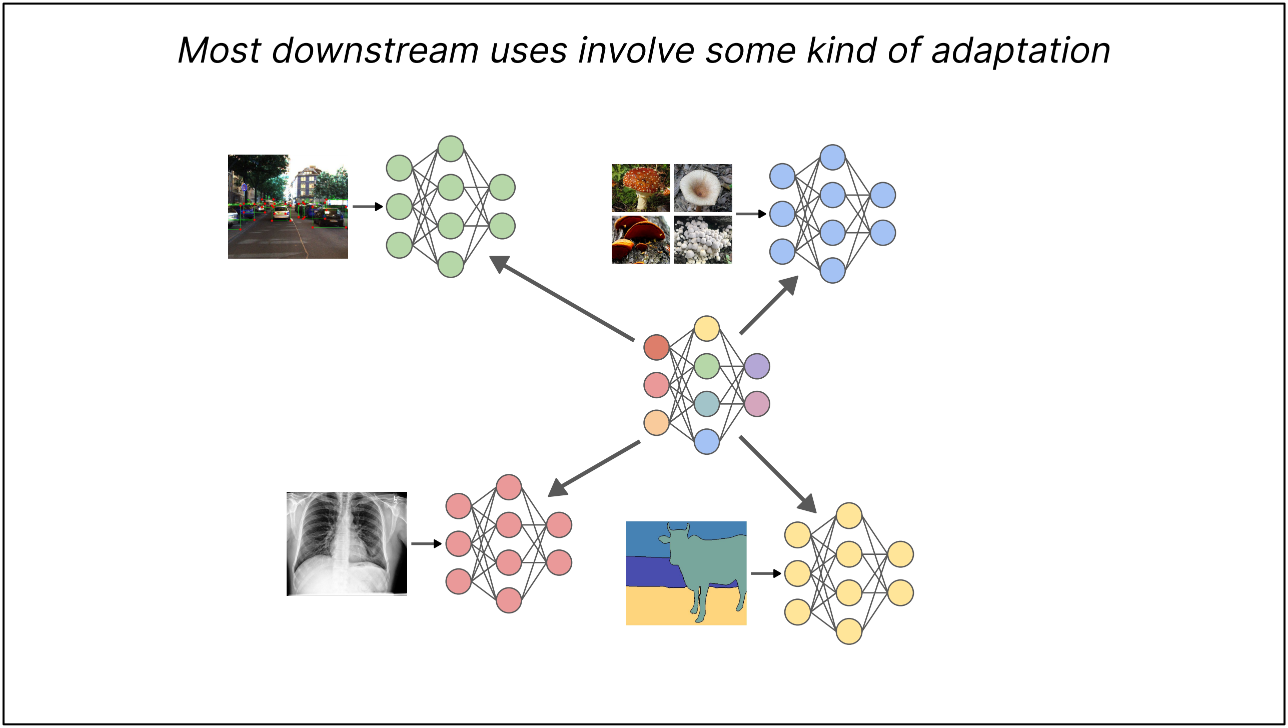
Most users of open-source software do not contribute back

On that note, I'd like to close by first revealing that I've been misleading you all somewhat this whole time. Let me explain how. First, note that in open-source software, users of the software rarely contribute back anything. For example, how many of you have contributed to CPython? Or git? But presumably most or all of you use them.



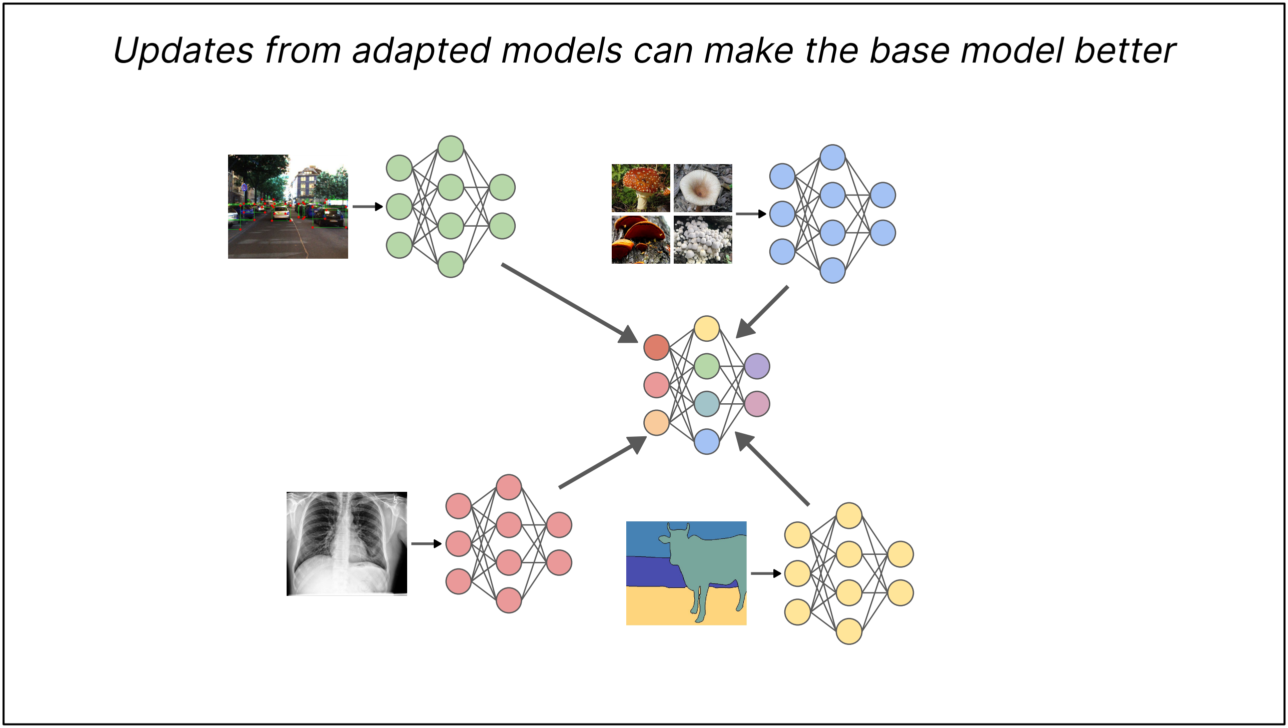
The same is currently true for downstream users of pre-trained models

That's kind of like the way things currently are with using pre-trained models. Most downstream users of the model don't contribute back to the model itself - in fact, ignoring the work I've presented today, they can't really.



Most downstream uses involve some kind of adaptation

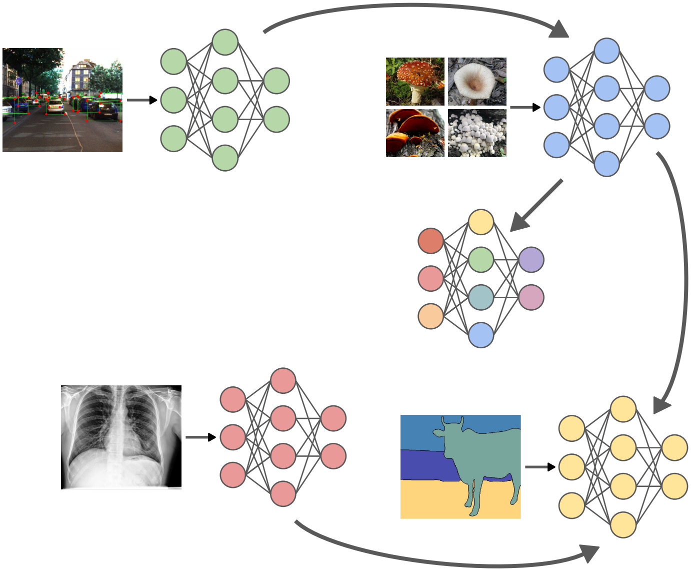
But note that downstream users of an machine learning model will always be using it to process or generate some data. This could involve training, or it could just involve feeding some data in and evaluating the results. Assuming this data would be valuable for the shared model, this means that all downstream users actually have something to contribute back if they could.



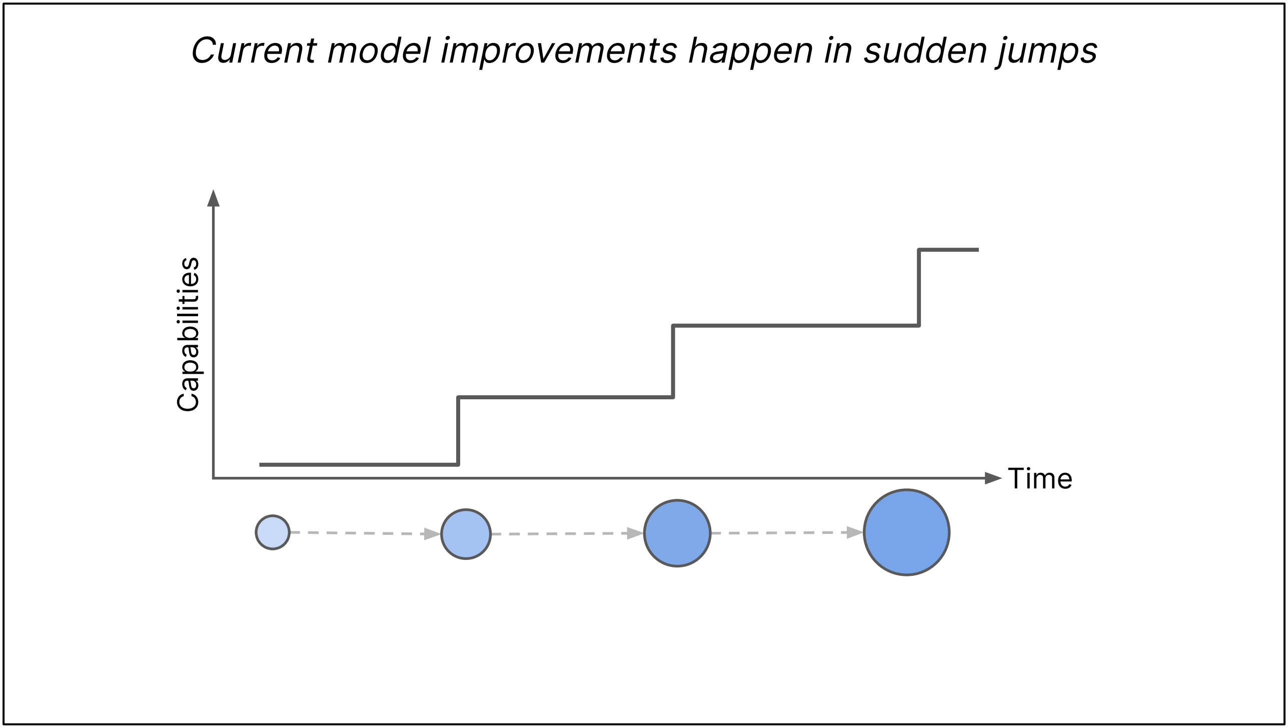
Updates from adapted models can make the base model better

So, I actually suspect that once truly collaborative and continual model development is possible, it'll be much more common for downstream users to contribute back to the base model than it is for downstream users of open-source software to contribute back. This makes me optimistic that applying this paradigm to model development will be even more effective and fruitful.

Adapted models share improvements without relying on the base model



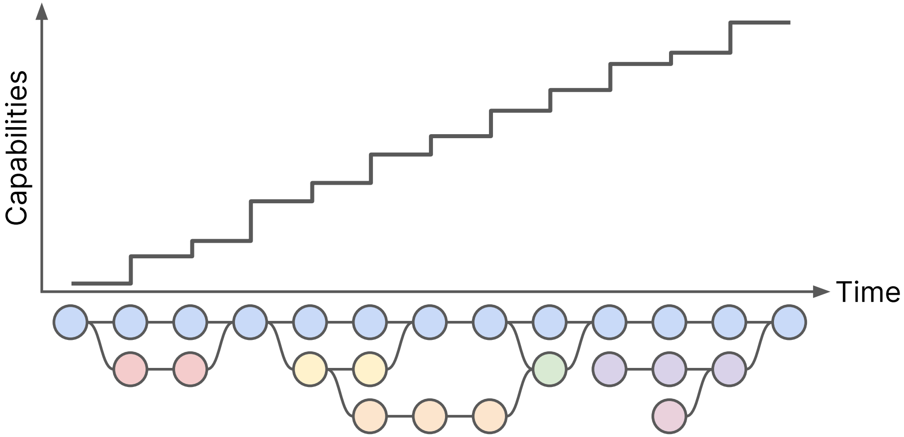
But I think it's even more different than that. In open-source software, because downstream users are rarely changing the software itself, there's rarely any reason for them to share updates with one another. But if downstream models are being updated and improved by all downstream users, the users could easily share their updates with each other without going through a centralized model. This means that the collection of possible base models to reuse and recombine will grow very quickly.



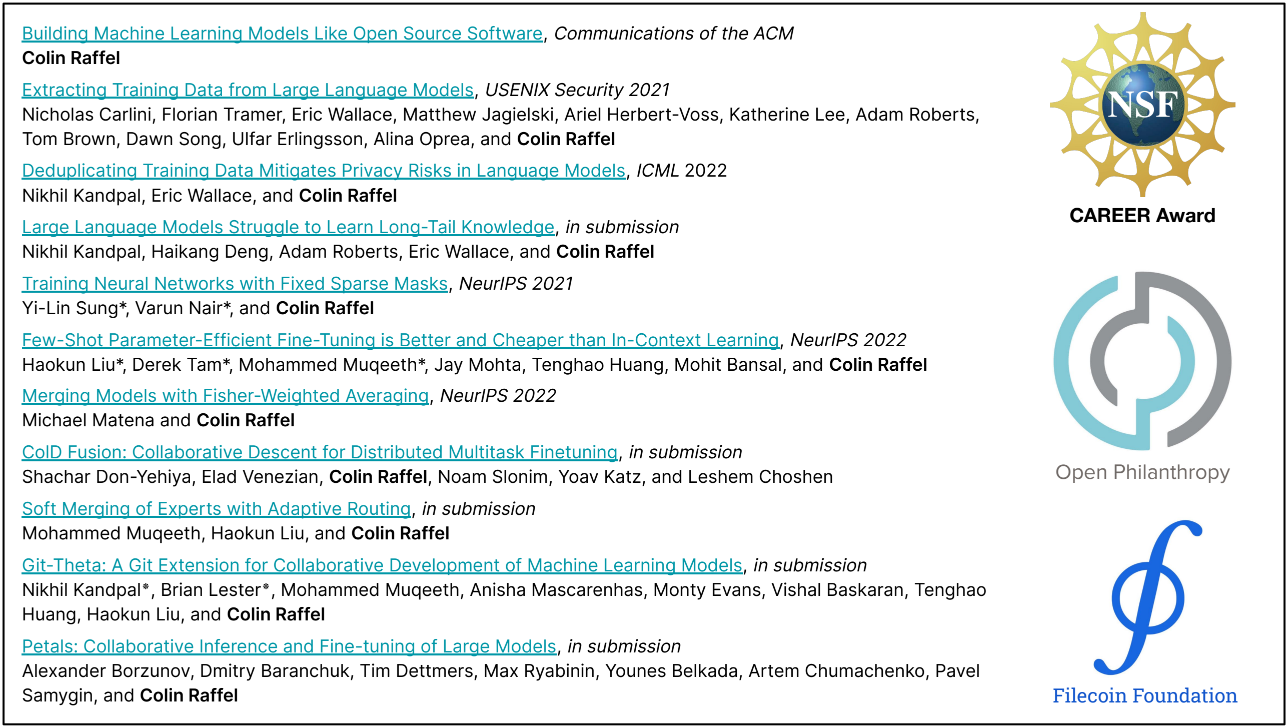
Current model improvements happen in sudden jumps

Now, let me use that perspective to make one more point. Currently, we expect improvements to look something like this. Each iteration of a model is a whole new model, oftentimes a much bigger one, and the new model is suddenly significantly better - we see expect and hope to see these big jumps in capabilities between models.

Collaborative development will lead to continual improvements



But once truly collaborative development is possible, I think we'll see a different paradigm where many different versions of a model are being developed in parallel and recombined and reused to incrementally and gradually improve capabilities of a model. I think this is a much more realistic way to get to true general-purpose models. We shouldn't throw out our old work and build something newer and bigger whenever we want an improvement. Instead, we should all work together to build models that have more and more capabilities, that are applicable in more and more settings, and that get better and better over time.



Building Machine Learning Models Like Open Source Software, *Communications of the ACM*
Colin Raffel

Extracting Training Data from Large Language Models, *USENIX Security 2021*
Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, Alina Oprea, and **Colin Raffel**

Deduplicating Training Data Mitigates Privacy Risks in Language Models, *ICML* 2022
Nikhil Kandpal, Eric Wallace, and **Colin Raffel**

Large Language Models Struggle to Learn Long-Tail Knowledge, *in submission*
Nikhil Kandpal, Haikang Deng, Adam Roberts, Eric Wallace, and **Colin Raffel**

Training Neural Networks with Fixed Sparse Masks, *NeurIPS 2021*
Yi-Lin Sung*, Varun Nair*, and **Colin Raffel**

Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning, *NeurIPS 2022*
Haokun Liu*, Derek Tam*, Mohammed Muqeeth*, Jay Mohta, Tenghao Huang, Mohit Bansal, and **Colin Raffel**

Merging Models with Fisher-Weighted Averaging, *NeurIPS 2022*
Michael Matena and **Colin Raffel**

ColD Fusion: Collaborative Descent for Distributed Multitask Finetuning, *in submission*
Shachar Don-Yehiya, Elad Venezian, **Colin Raffel**, Noam Slonim, Yoav Katz, and Leshem Choshen

Soft Merging of Experts with Adaptive Routing, *in submission*
Mohammed Muqeeth, Haokun Liu, and **Colin Raffel**

Git-Theta: A Git Extension for Collaborative Development of Machine Learning Models, *in submission*
Nikhil Kandpal*, Brian Lester*, Mohammed Muqeeth, Anisha Mascarenhas, Monty Evans, Vishal Baskaran, Tenghao Huang, Haokun Liu, and **Colin Raffel**

Petals: Collaborative Inference and Fine-tuning of Large Models, *in submission*
Alexander Borzunov, Dmitry Baranchuk, Tim Dettmers, Max Ryabinin, Younes Belkada, Artem Chumachenko, Pavel Samygin, and **Colin Raffel**

NSF
CAREER Award

Open Philanthropy

Filecoin Foundation

That's all I'll say - here is a list of some of the papers I mentioned today, along with all of the amazing collaborators that made them happen. I'd also like to thank funding agencies who made this work possible. I'm happy to take questions now.