Modular Deep Learning

Edoardo M. Ponti
eponti@ed.ac.uk

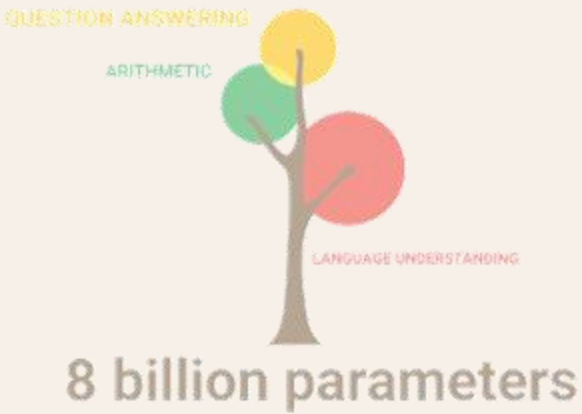
Sapienza, 9 Jun 2023

THE UNIVERSITY of EDINBURGH
UNIVERSITY OF CAMBRIDGE

Background and Motivation

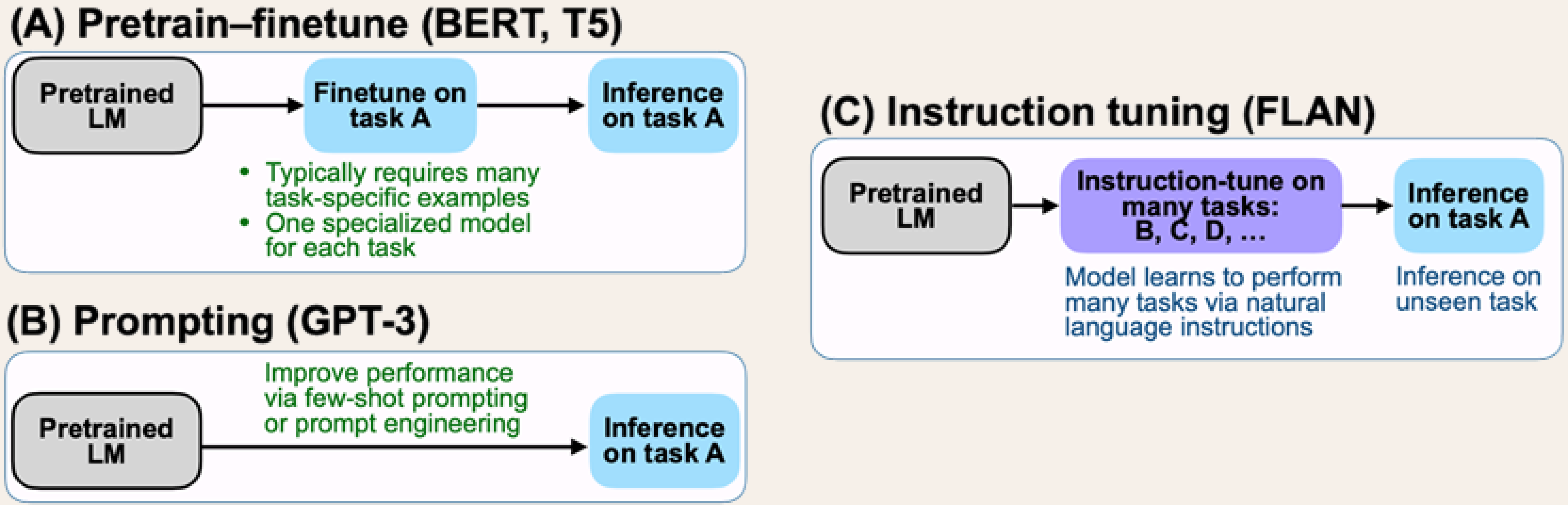
Pfeiffer, Jonas, Sebastian Ruder, Ivan Vulić, and Edoardo M. Ponti. *Modular deep learning*. arXiv 2023

Emergence of NLP Abilities via Scaling



Credits: Google AI Blog

Zero/few-shot generalisation to new tasks

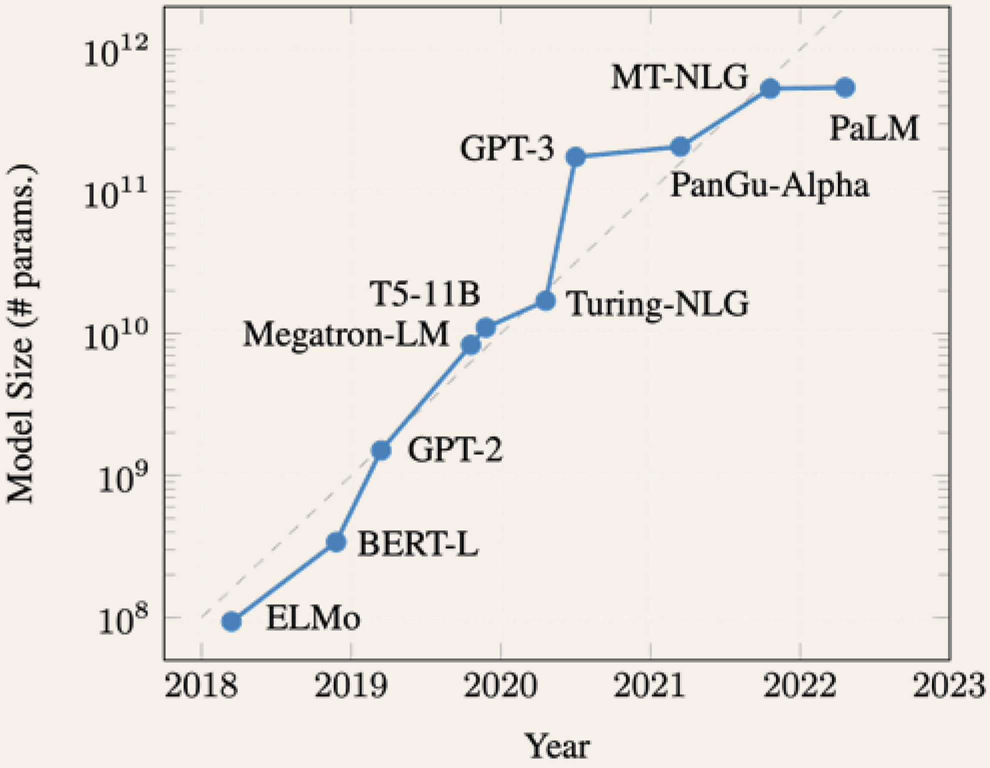


[Wei et al. 2022]

Challenges of Existing Methods

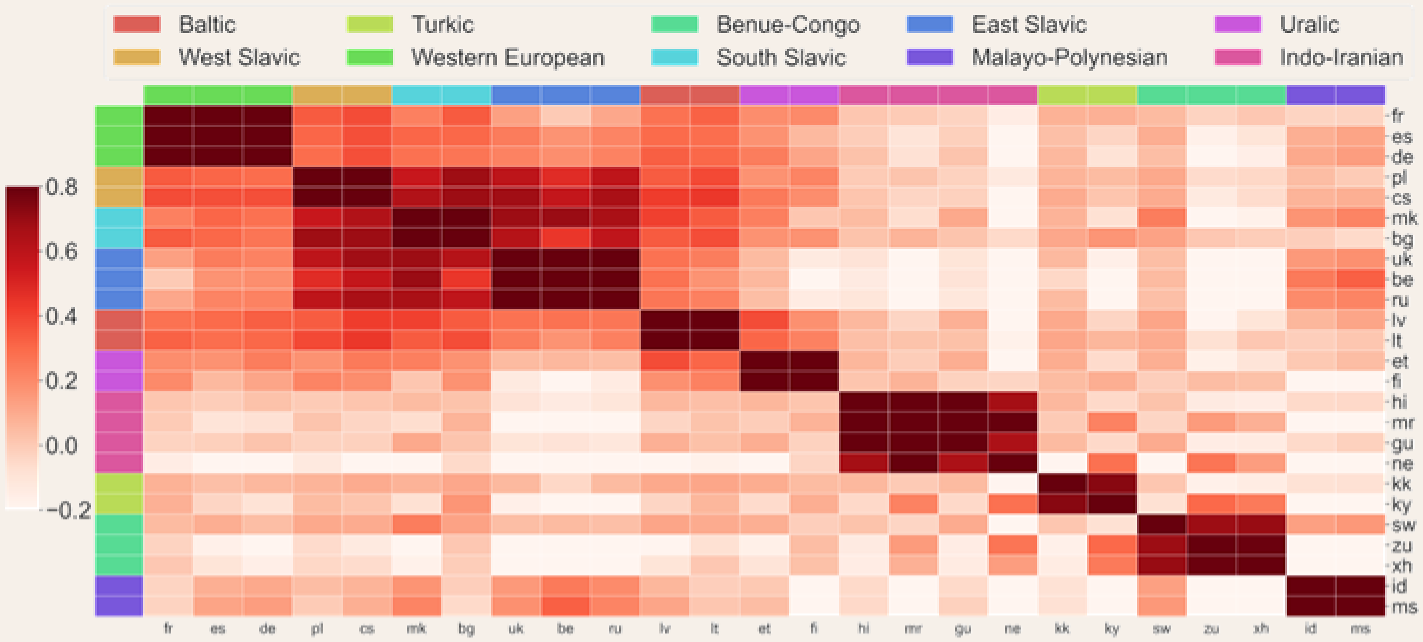
- **Inefficiency due to model size:**
  time and space complexity
- **Negative transfer:**
  interference (multi-task learning)
  catastrophic forgetting (continuous learning)
- **Systematic generalisation:**
  sub-problem recombination
  local distribution shifts

Inefficiency due to Model Size



Evolution of the size of large pre-trained models [Treviso et al., 2022]

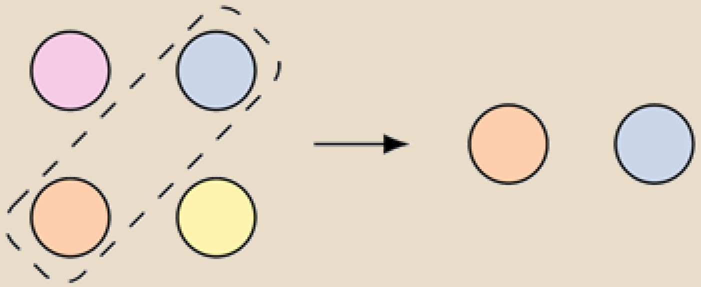
Negative Transfer



Cosine similarity between gradients for NMT from multiple languages to English [Wang et al. 2021, Gradient Vaccine].

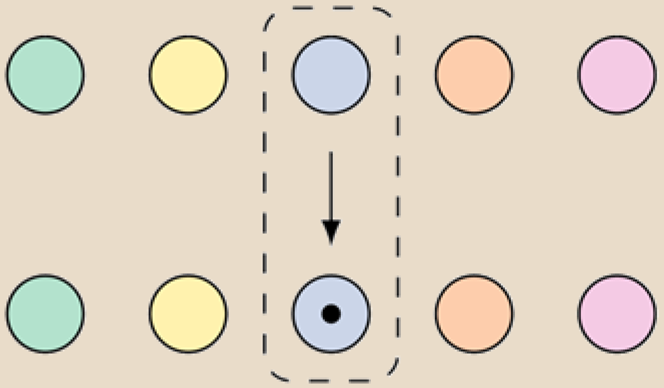
Systematic Generalisation [Hupkes et al. 2020]

Disentangling and recombining autonomous facets of knowledge



Skill recombination

Adapting to local distribution shifts based on few examples



Local adaptation

Modular Deep Learning [Pfeiffer et al. 2023]

Correspondence between **modules** of a network and the **specialised functions** they perform.

Separation between **routing** (controlling information flow) and **computation** (modules).

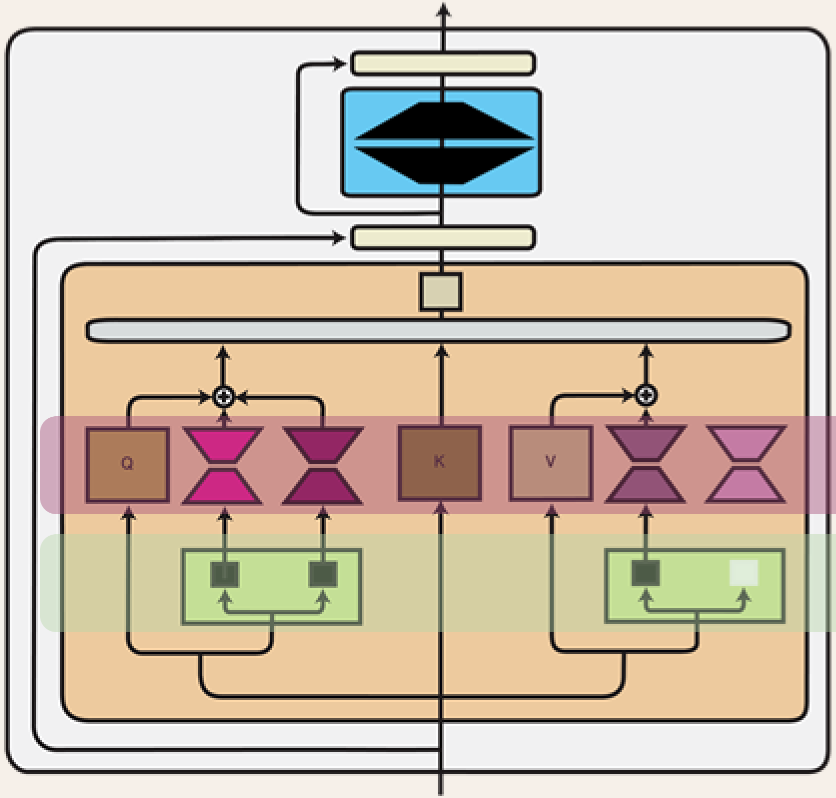
In **multitask learning**, modules should specialise for **skills**, subproblems common to multiple tasks which can be recombined or locally updated.

Advantages

- **Over in-context learning:**
  - **Robustness** (no instability due to ordering [Zhao et al., 2021], wording [Webson & Pavlick, 2022], etc.)
  - **Higher performance** [Brown et al. 2020]

- **Over (instruction) fine-tuning:**
  - Only **positive transfer** across tasks (no catastrophic **forgetting and interference**) [Caccia et al. 2023]
  - Compositionality, reusability, and local updates of modules: **systematic generalisation** [Ponti et al. 2020]
  - **Parameter efficiency** (no large full-model copies) [Liu et al., 2022]
  - **Scaling** (e.g., through MoE) [Shazeer et al. 2017]

A Blueprint of Modular Deep Learning

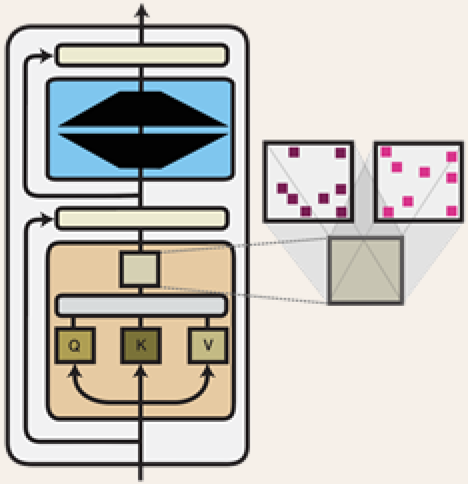


1) Modules

2) Routing

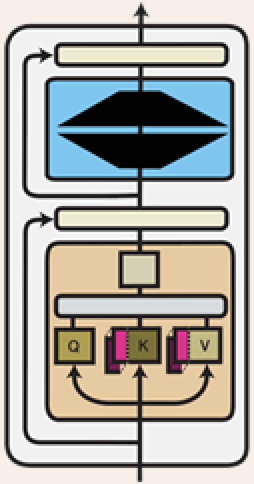
3) Applications

Modules: Parameter-Efficient Fine-Tuning



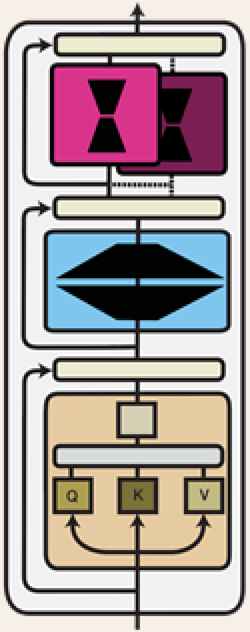
Parameter Composition

$$f_i'(\boldsymbol{x}) = f_{\theta_i \oplus \phi}(\boldsymbol{x})$$



Input Composition

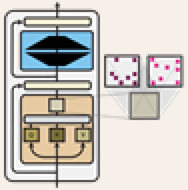
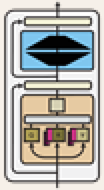
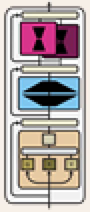
$$f_i'(\boldsymbol{x}) = f_{\theta_i}([\boldsymbol{x}, \phi])$$



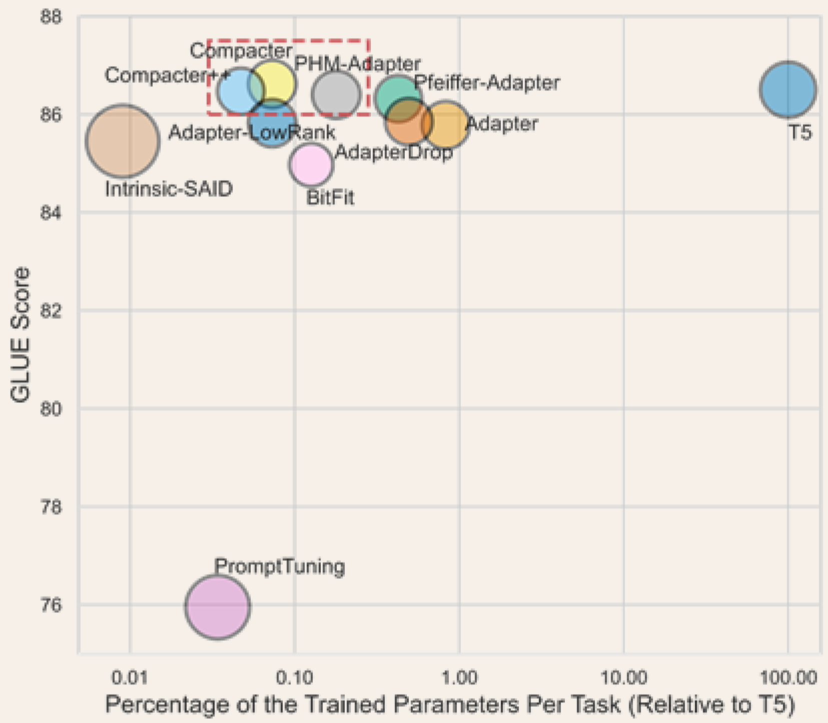
Function Composition

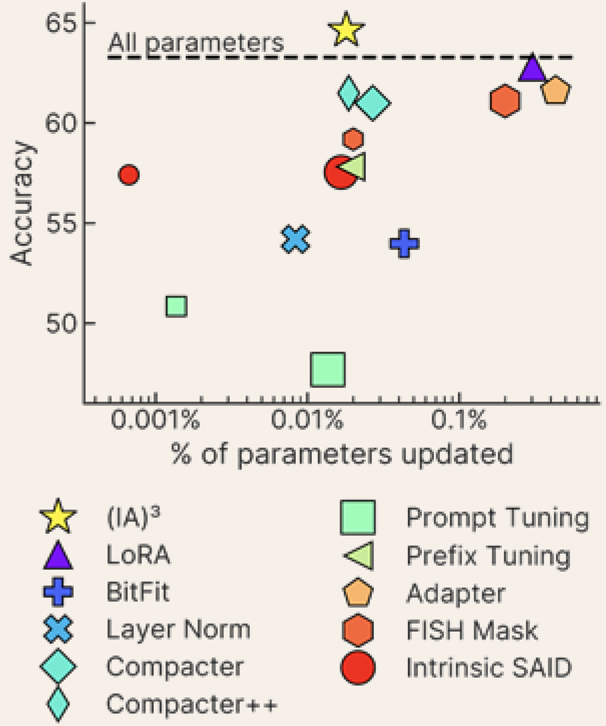
$$f_i'(\boldsymbol{x}) = f_{\theta_i} \odot f_\phi(\boldsymbol{x})$$

Modules: Parameter-Efficient Fine-Tuning

	Parameter efficiency	Training efficiency	Inference efficiency	Performance	Composition-ality
Parameter composition 	+	-	++	+	+
Input composition 	++	--	--	-	+
Function Composition 	-	-	-	++	+

Performance and Efficiency Comparison

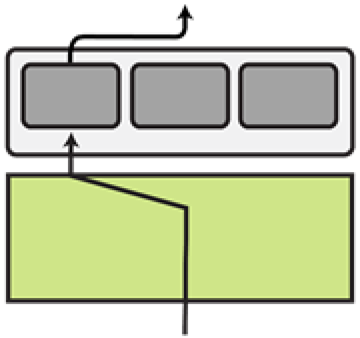




Performance, param efficiency, and memory footprint of different methods on T5-Base (222M params; left) [Mahabadi et al., 2021] and T5-3B (3B params; right) [Liu et al., 2022]

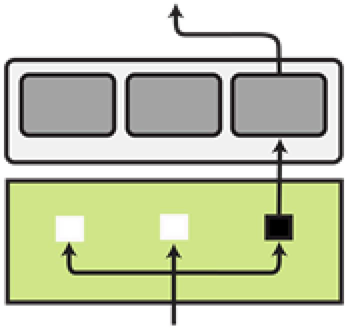
Routing

Fixed Routing

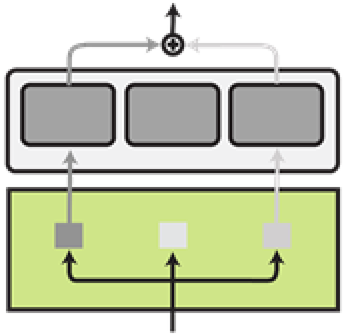


Learned Routing

Hard Learned Routing



Soft Learned Routing



Applications

- **Zero-shot Cross-lingual Transfer**

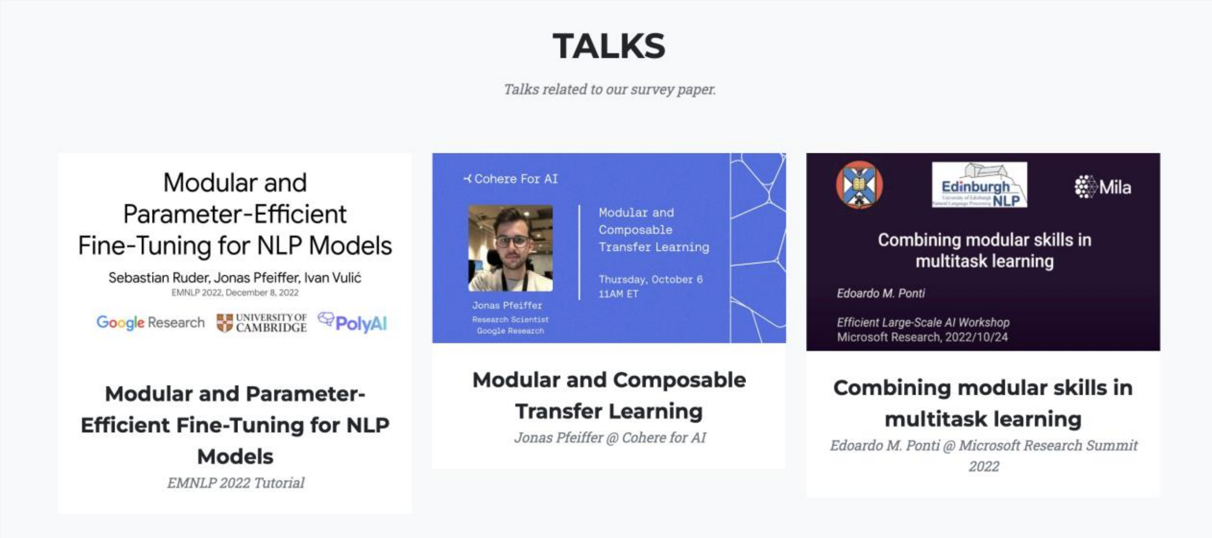
- **Faithful and Abstractive Dialogue Generation**

- **Few-shot Adaptation to New RL / NLP Tasks**

- …and many more (including programme induction and causal inference / discovery)!

Survey on Modular Deep Learning

https://www.modulardeeplearning.com

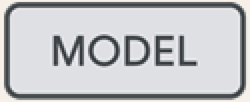


Composition of Sparse Adapters

Ansell, Alan, Edoardo M. Ponti, Anna Korhonen, and Ivan Vulić.
Composable Sparse Fine-Tuning for Cross-Lingual Transfer. ACL 2022

Standard Zero-Shot Cross-Lingual Transfer

Step 1:



Pre-train a multilingual model.

Step 2:

Fine-tune the model on a **task** in a high-resource **source language.**

Step 3:

Transfer and **evaluate** the model on a low-resource **target language.**

Why?

Training **data** is **expensive** and not available for many languages, especially ones that are considered “low-resource”.

Modular Zero-Shot Cross-Lingual Transfer

Step 1: Train Language Adapters

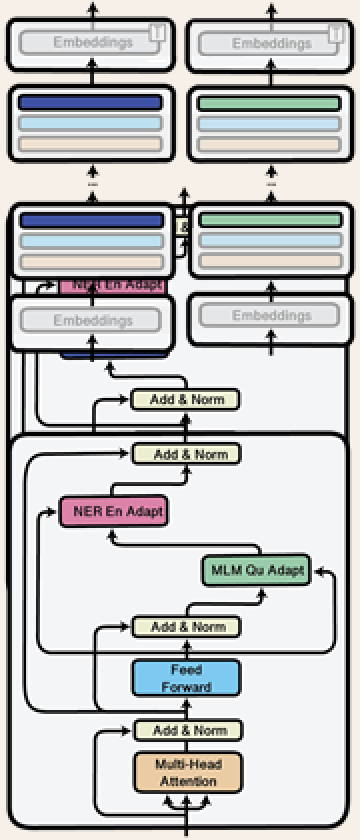
We train **language adapters** for the **source language** and the **target language** with masked language modelling on Wikipedia.

Step 2: Train a Task Adapter

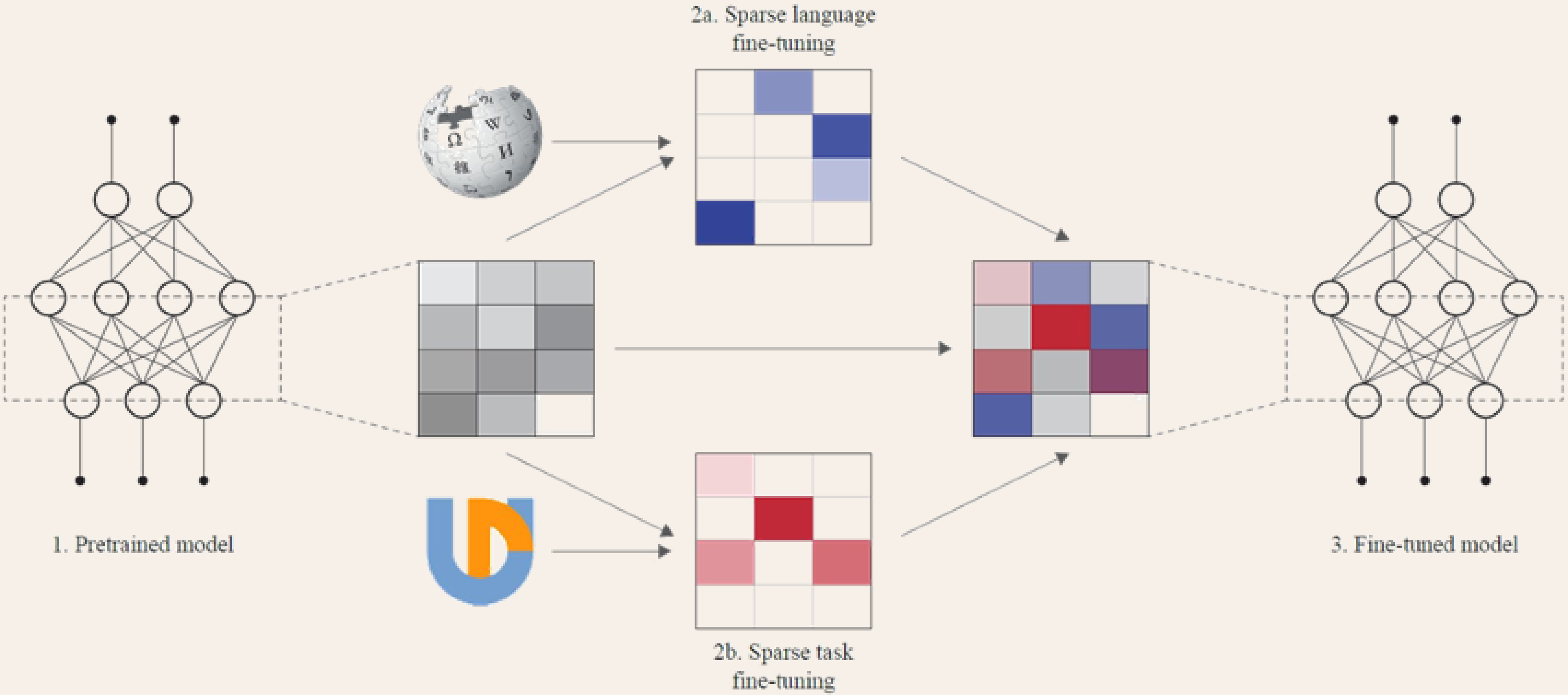
We **train** **task adapters** in the source language **stacked** on top of the source **language adapter**.

Step 3: Zero-Shot transfer to unseen language

We **replace** the **source** language adapter with the **target** language adapter, while **keeping** the “language agnostic” **task adapter**.



Sparse Fine-Tuning



Lottery Ticket-inspired Algorithm

Algorithm 1 Cross-Lingual Transfer with Lottery-Ticket Sparse Fine-Tuning

function LTSFT($\mathcal{D}$, $\mathcal{L}$, $\boldsymbol{\theta}^{(0)}$, η, K)

$\boldsymbol{\theta}^{(1)} \leftarrow \boldsymbol{\theta}^{(0)}$

while not converged **do**

$\boldsymbol{\theta}^{(1)} \leftarrow \boldsymbol{\theta}^{(1)} - \eta \nabla \mathcal{L}(\boldsymbol{\theta}^{(1)}, \mathcal{D})$

$$\mu_i \leftarrow \begin{cases} 1 & \text{if } \theta_i^{(1)} \in \operatorname{argmax}_{\theta_1, \ldots, \theta_K} |\boldsymbol{\theta}^{(1)} - \boldsymbol{\theta}^{(0)}| \\ 0 & \text{otherwise} \end{cases}$$

$\boldsymbol{\theta}^{(2)} \leftarrow \boldsymbol{\theta}^{(0)}$

while not converged **do**

$\boldsymbol{\theta}^{(2)} \leftarrow \boldsymbol{\theta}^{(2)} - \boldsymbol{\mu} \odot \eta \nabla \mathcal{L}(\boldsymbol{\theta}^{(2)}, \mathcal{D})$

$\boldsymbol{\phi} \leftarrow \boldsymbol{\theta}^{(2)} - \boldsymbol{\theta}^{(0)}$

return $\boldsymbol{\phi}$

end function

function CROSSLINGUALTRANSFER($\mathcal{D}_{\text{src}}$, $\mathcal{D}_{\text{tar}}$, $\mathcal{D}_{\text{task}}$, $\mathcal{L}_{\text{task}}$, $\boldsymbol{\theta}^{(0)}$, η, K)

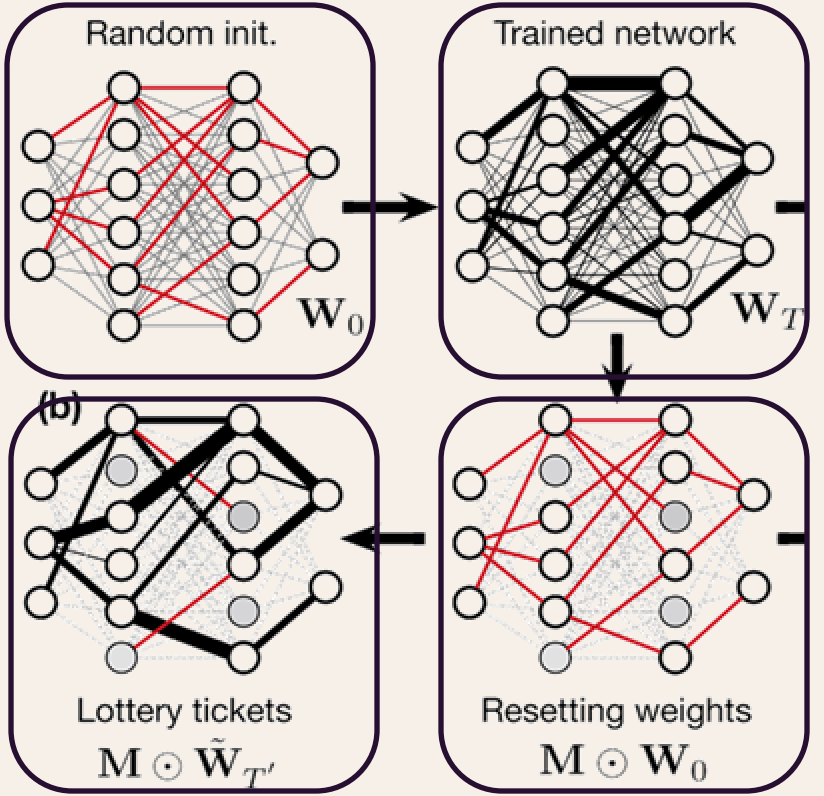
$\boldsymbol{\phi}_{\text{src}} \leftarrow \text{LTSFT}(\mathcal{D}_{\text{src}}, \mathcal{L}_{\text{MLM}}, \boldsymbol{\theta}^{(0)}, \eta, K)$

$\boldsymbol{\phi}_{\text{task}} \leftarrow \text{LTSFT}(\mathcal{D}_{\text{task}}, \mathcal{L}_{\text{task}}, \boldsymbol{\theta}^{(0)} + \boldsymbol{\phi}_{\text{src}}, \eta, K)$

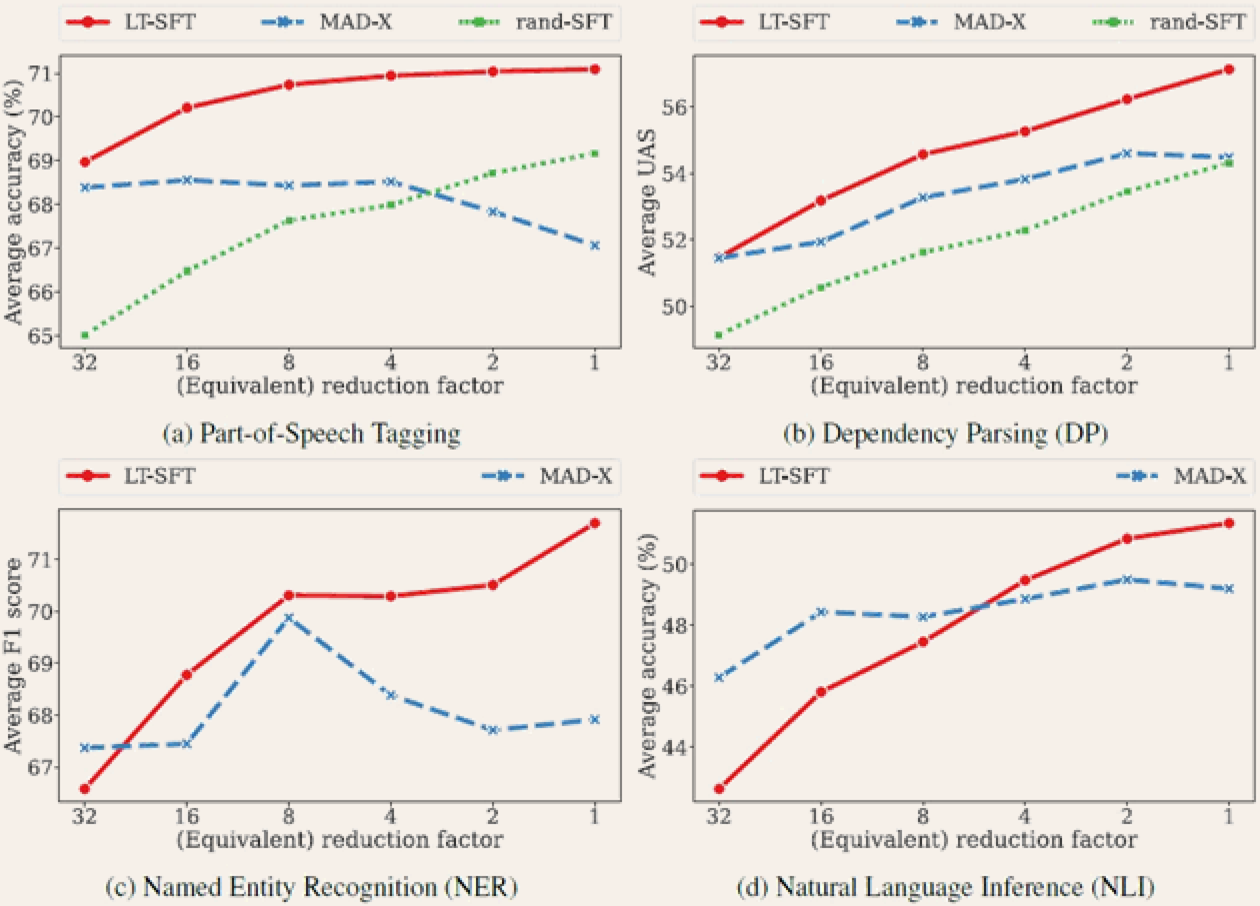
$\boldsymbol{\phi}_{\text{tar}} \leftarrow \text{LTSFT}(\mathcal{D}_{\text{tar}}, \mathcal{L}_{\text{MLM}}, \boldsymbol{\theta}^{(0)}, \eta, K)$

return $\boldsymbol{\theta}^{(0)} + \boldsymbol{\phi}_{\text{task}} + \boldsymbol{\phi}_{\text{tar}}$

end function



Results for Zero-shot Transfer



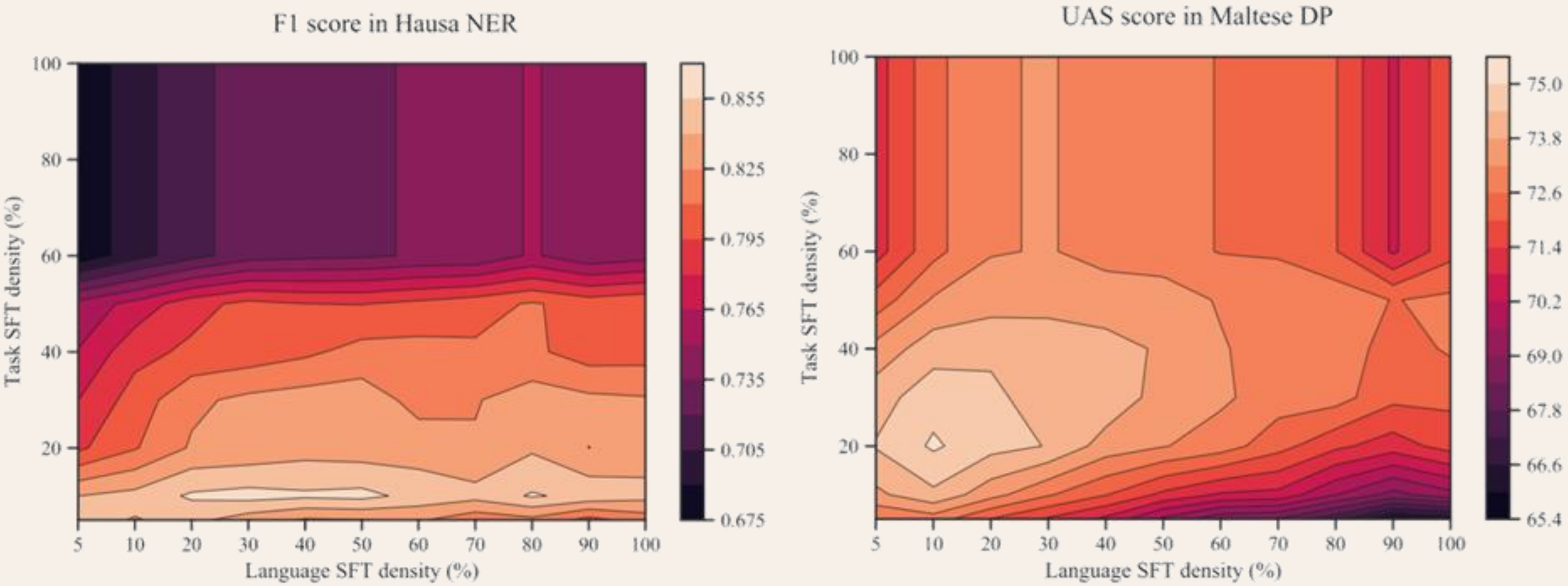
(a) Part-of-Speech Tagging

(b) Dependency Parsing (DP)

(c) Named Entity Recognition (NER)

(d) Natural Language Inference (NLI)

On the Importance of Sparsity



Code

github.com/cambridgeltl/composable-sft

Language SFTs

Identifiers for language SFTs are of the form `cambridgeltl/{base_model}-lang-sft-{lang_code}-small`, e.g. `cambridgeltl/mbert-lang-sft-en-small`. "Small" SFTs have ~7.6M parameters - we may release larger models in the future. Language SFTs are currently available for the following languages/models:

Language	Code	bert-base-multilingual-cased (mbert)	xlm-roberta-base (xlmr)
Acehnese	ace	✕	✓
Amharic	amh	✕	✓
Arabic	ar	✓	✓
Ashaninka	cni	✕	✓
Balinese	ban	✕	✓
Bambara	bm	✓	✕
Banjarese	bjn	✕	✓
Basque	eu	✓	✕
Bengali	bn	✓	✕
Bribri	bzd	✕	✓
Bulgarian	bg	✕	✓
Buryat	bxr	✓	✕
Cantonese	yue	✓	✕
Chinese	zh	✓	✓
Czech	cs	✓	✕
English	en	✓	✓
Erzya	myv	✓	✕
Estonian	et	✓	✕
Faroese	fo	✓	✕
French	fr	✓	✓
German	de	✓	✓

Faithful and Abstractive Dialogue Generation

Daheim, Nico, Nouha Dziri, Mrinmaya Sachan, Iryna Gurevych, and Edoardo M. Ponti. *Elastic Weight Removal for Faithful and Abstractive Dialogue Generation*. arXiv 2023

Knowledge-grounded dialogue generation

$$p_\theta(u_{T+1} \mid u_1^T, \hat{\mathcal{K}}) = \prod_{n=1}^{N_{T+1}} p_\theta([u_{T+1}]_n \mid [u_{T+1}]_1^{n-1}, u_1^T, \hat{\mathcal{K}})$$

Faithfulness (opposite: hallucination) is the adherence of the generated response *u* to the knowledge *K*

Abstractive (opposite: extractive) responses *u* do not copy-paste spans from knowledge *K* but rephrase them.

$\mathcal{K}$: The Flash first appeared in “Showcase” #4 (October 1956) [...]
u_T: What comic series is he from?

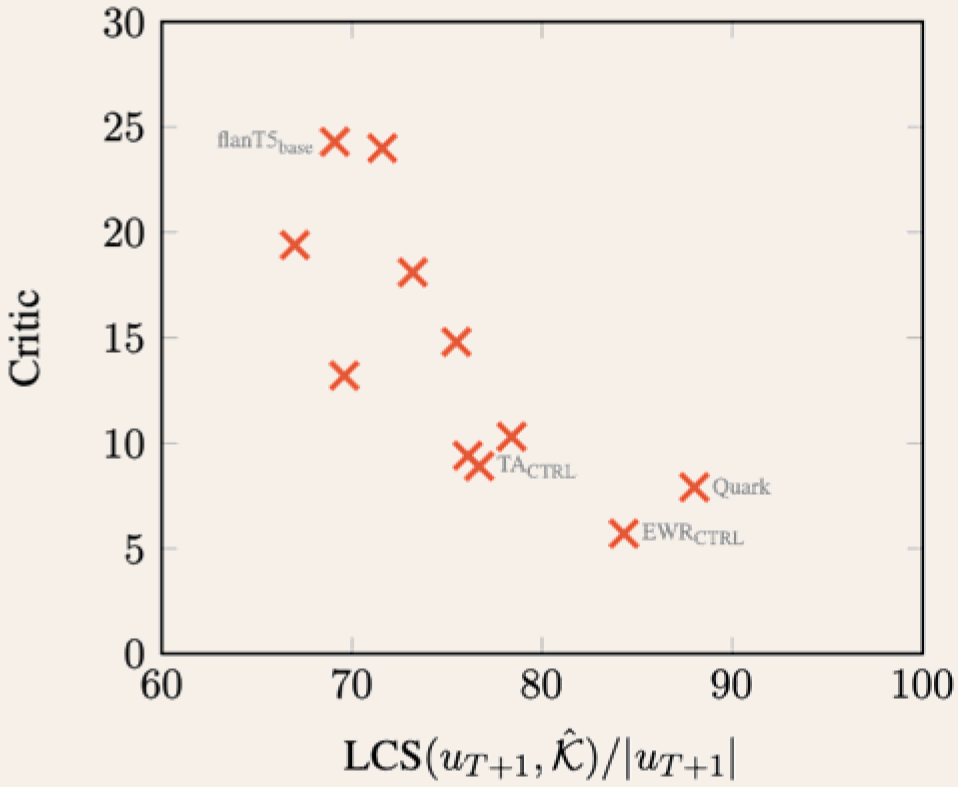
u_{T+1}	F	A
He first appeared in “Showcase” #4 (November 1956).	✗	✗
He first appeared in “Showcase” #4 (October 1956).	✓	✗
His first appearance was in Showcase #4 in October 1956.	✓	✓

Faithfulness—Abstractiveness Trade-off

Faithfulness can be measured by a Critic (a binary classifier)

Abstractiveness can be measured by normalised LCS (longest common span)

Faithful models generally incur **ex**tractive generation. Can we have the best of both worlds?



Task Arithmetic

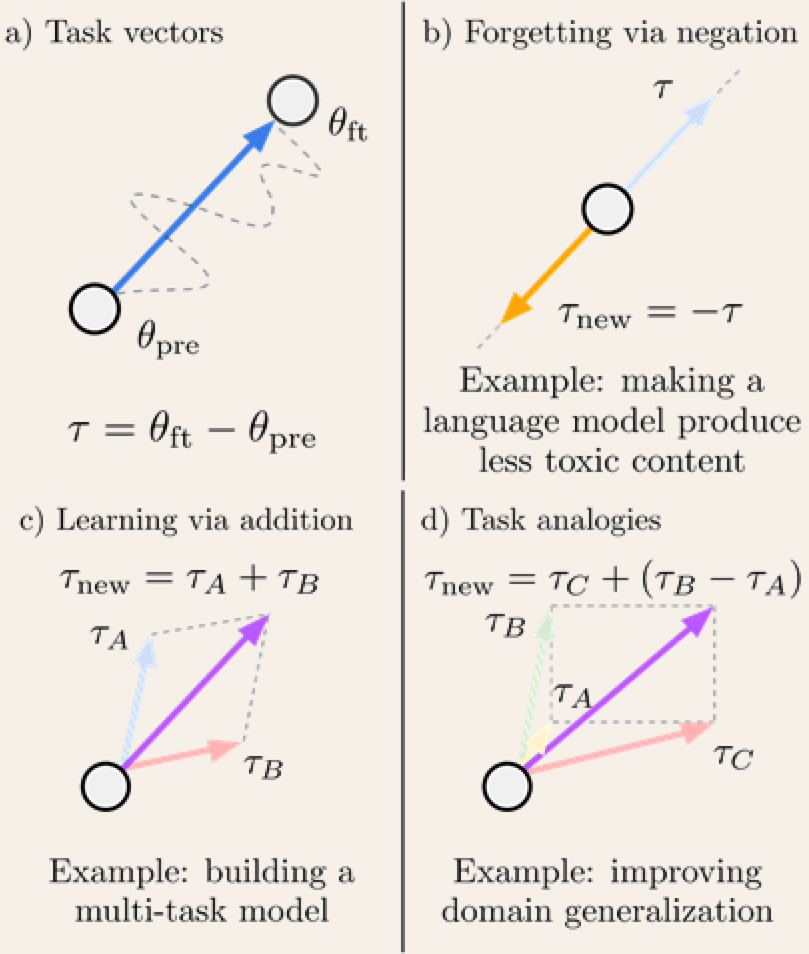
[Ilharco et al. 22]

Step 1: Create **task vectors** as the difference between a model fine-tuned on examples of (positive / negative) behaviour and a pre-trained model.

$$\tau_i \triangleq \theta_i - \theta_0$$

Step 2: Add / Subtract task vectors to the pre-trained model.

$$\theta' = \theta_0 + \sum_i \lambda_i \tau_i$$



Elastic Weight Addition and Subtraction

Limitations of task arithmetic:

1) task vectors may **interfere** with each other;
2) individual parameters have **higher importance** than others in controlling a certain behaviour;

Solution: weight task vectors by **Fisher Information** *f* to prevent interference and represent parameter importance.

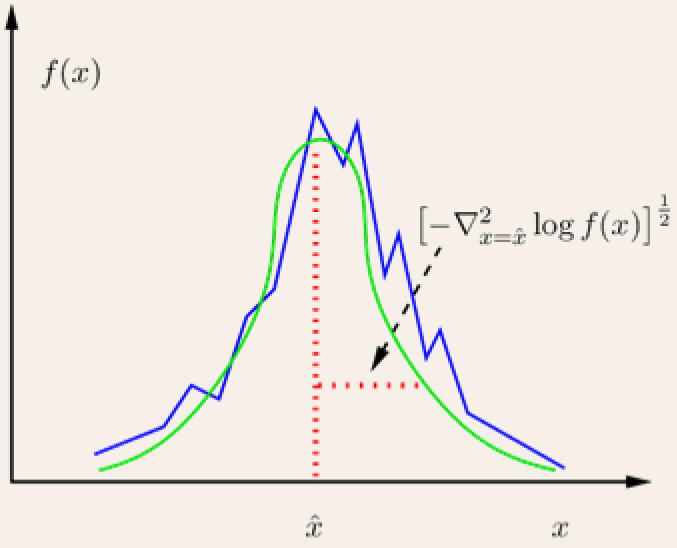
$$\frac{\lambda_0 \cdot f_{\theta_0} \cdot \theta_0 + \sum_{i=1}^{N} \lambda_i \cdot f_{\tau_i} \cdot \tau_i}{Z}$$

Estimating Fisher Information

$$F_\theta = \mathbb{E}_{p_\theta(y|x)} \nabla_\theta \log p_\theta(y \mid x) \nabla_\theta \log p_\theta(y \mid x)^\top$$

Empirical and diagonal approximation:

$$f_\theta = \frac{1}{|\mathcal{D}|} \sum_{\mathcal{D}} (\nabla_\theta \log p(y \mid x))^2$$



The algorithm

Step 1: Create task vectors

Step 2: Create Fisher vectors

Step 3: Merge

Algorithm 1 Pseudocode for removing hallucinations and promoting abstraction with EWR. Note that we apply $(\cdot)^2$ element-wise.

Input Dialogues $\mathcal{D}$, hallucinated anti-expert dataset $\mathcal{D}^{\text{AE}}$, abstractive expert dataset $\mathcal{D}^{\text{E}}$, initial parameter set $\boldsymbol{\theta}$

Output $\boldsymbol{\theta}'$

$$\boldsymbol{\theta}_0 \leftarrow \text{finetune}(\boldsymbol{\theta}, \mathcal{D})$$

$$\boldsymbol{\theta}_{\text{AE}} \leftarrow \text{finetune}(\boldsymbol{\theta}_0, \mathcal{D}^{\text{AE}})$$

$$\boldsymbol{\tau}_1 \leftarrow \boldsymbol{\theta}_{\text{AE}} - \boldsymbol{\theta}_0$$

$$\boldsymbol{\theta}_{\text{E}} \leftarrow \text{finetune}(\boldsymbol{\theta}_0, \mathcal{D}^{\text{E}})$$

$$\boldsymbol{\tau}_2 \leftarrow \boldsymbol{\theta}_{\text{E}} - \boldsymbol{\theta}_0$$

$$\mathbf{f}_{\boldsymbol{\theta}_0} \leftarrow \frac{1}{|\mathcal{D}|} \sum_{\mathcal{D}} (\nabla \log p_{\boldsymbol{\theta}_0}(u_{T+1} \mid u_1^T, \hat{\mathcal{K}}))^2$$

$$\mathbf{f}_{\boldsymbol{\tau}_1} \leftarrow \frac{1}{|\mathcal{D}^{\text{AE}}|} \sum_{\mathcal{D}^{\text{AE}}} (\nabla \log p_{\boldsymbol{\tau}_1}(u_{T+1} \mid u_1^T, \hat{\mathcal{K}}))^2$$

$$\mathbf{f}_{\boldsymbol{\tau}_2} \leftarrow \frac{1}{|\mathcal{D}^{\text{E}}|} \sum_{\mathcal{D}^{\text{E}}} (\nabla \log p_{\boldsymbol{\tau}_2}(u_{T+1} \mid u_1^T, \hat{\mathcal{K}}))^2$$

$$\boldsymbol{\theta}' \leftarrow \frac{\lambda_0 \cdot \mathbf{f}_{\boldsymbol{\theta}_0} \cdot \boldsymbol{\theta}_0 - \lambda_1 \cdot \mathbf{f}_{\boldsymbol{\tau}_1} \cdot \boldsymbol{\tau}_1 + \lambda_2 \cdot \mathbf{f}_{\boldsymbol{\tau}_2} \cdot \boldsymbol{\tau}_2}{Z}$$

Results

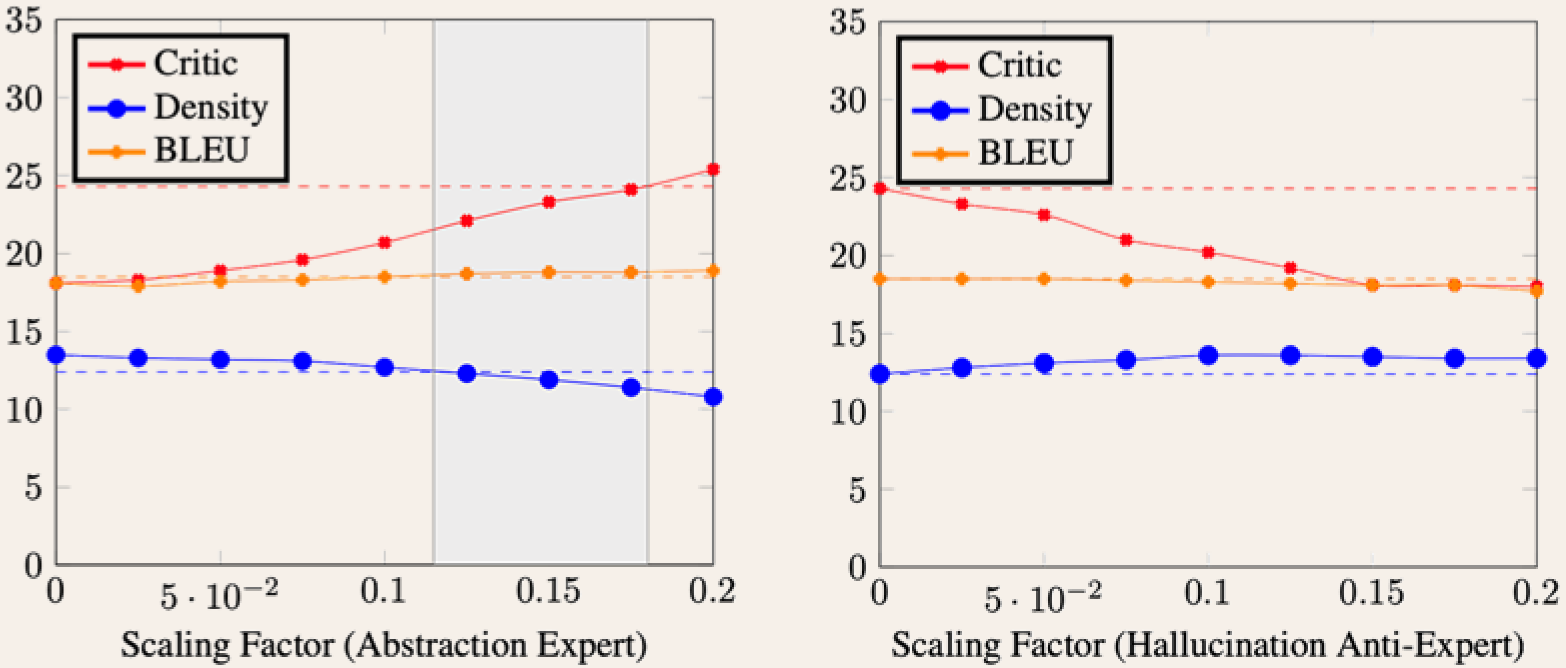
Model	BLEU(↑) $(y, \hat{y})$	Critic(↓) $(y, \hat{\mathcal{K}})$	Q^2(↑)	BF1(↑)	F1(↑)
			WoW$_{unseen}$		
Flan-T5$_{base}$	18.1	22.7	74.0	84.8	78.7
+ TA	18.8	19.2	75.7	82.8	75.0
+ EWR	17.4 (↓-0.7)	17.7 (↓-5.0)	78.4 (↑4.4)	86.9 (↑2.1)	81.6 (↑2.9)
			DSTC11		
Flan-T5$_{base}$	7.9	76.6	49.7	54.6	37.1
+ TA	8.0	60.0	51.0	59.9	**43.6**
+ EWR	**9.6** (↑1.7)	**41.1** (↓35.5)	**57.3** (↑7.6)	**60.0** (↑5.4)	38.6 (↑1.5)
			FaithDial		
Flan-T5$_{base}$	15.1	0.3	**66.4**	80.9	73.7
+ TA	**15.3**	**0.1**	57.5	77.3	67.6
+ EWR	14.9 (↓-0.2)	**0.1** (↓-0.2)	**66.4** (-0.0)	**81.7** (↑0.8)	**75.0** (↑1.3)

Table 2: EWR improves faithfulness on unseen topics (WoW$_{unseen}$), multi-document corpora (DSTC11), and datasets with cleaned ground-truth annotations (Faith-

Model	WoW A (↑)	WoW C (↑)	WoW P (↑)	DSTC9 A (↑)	DSTC9 C (↑)	DSTC9 P (↑)
Flan-T5$_{base}$	72.3	1.74	1.19	89.7	2.83	1.71
+ EWR$_{abs}$	75.1	1.62	1.25	94.7*	2.41	1.49
CTRL	85.5*	1.58	1.12	94.7*	2.72	1.42
+ TA	88.8*	1.58	1.16	97.0*	2.63	1.40
+ EWR	96.8†	1.50	1.08	98.0†	2.50	1.36
Quark	93.1†	1.51	1.05	86.0	2.89	1.66

Table 3: Human evaluation on 218 examples annotated by 3 expert annotators each. We measure attributability (A), Co-cooperativeness (C), and paraphrasing (P). * indicates significant improvements wrt. Flan-T5$_{base}$ and † wrt. to the next best method with $p < 0.05$.

The Sweet Spot



(a) Faithfulness–Abstractiveness Trade-Off

(b) Faithfulness–Performance Trade-Off

Code

https://github.com/ndaheim/faithful-dialogue

Baseline	`document_grounded_generation`	(context, knowledge) -> response
CTRL	`document_grounded_generation_ctrl`	(context, knowledge, control tokens) -> response
Quark	`document_grounded_generation_Quark`	CTRL tokens as quantized reward for samples drawn from the model during training
Noisy Channel Reranking	`noisy_channel_reranking`	reranking controllably with a faithfulness and fluency expert
DExperts	`document_grounded_generation_density_ratio`	faithfulness expert and anti-expert combined at inference time
Task Arihmetic	`document_grounded_generation`	task vector subtracted from base model using hallucination anti-expert
CaPE	`document_grounded_generation`	task vector subtracted from base model using hallucination anti-expert and faithfulness expert
EWR	`document_grounded_generation`	task vector subtracted from base model using hallucination anti-expert weighted by Fisher Information

Note that TA, CaPE and EWR just change the model parameters via interpolation, but not the model architecture!

Datasets

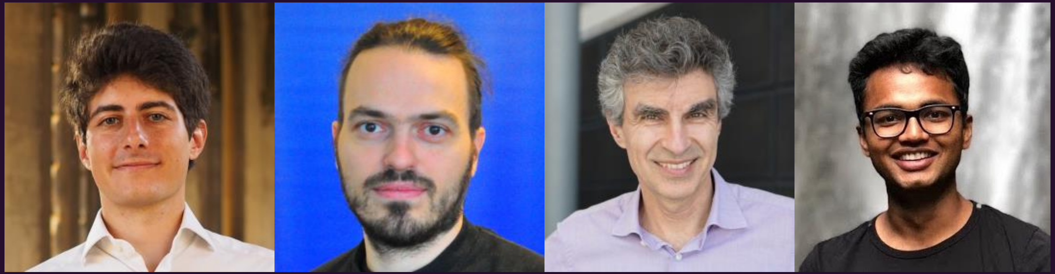
1. Wizard-of-Wikipedia
2. FaithDial
3. DSTC9
4. DSTC11

Metrics

1. BLEU
2. BERTScore
3. Faithfulness Critic
4. Q^2
5. Knowledge F1
6. Density & Coverage

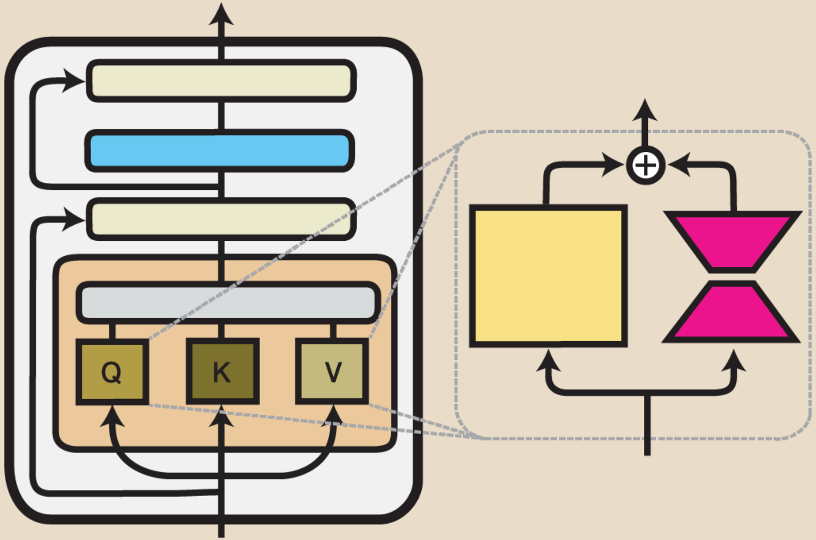
Polytropon: Joint Routing and Adaptation

Edoardo M. Ponti, Alessandro Sordoni, Yoshua Bengio, Siva Reddy.
Combining Parameter-efficient Modules for Task-level Generalisation.
EACL 23



Goal: adapting general-purpose LLMs efficiently and systematically to new tasks

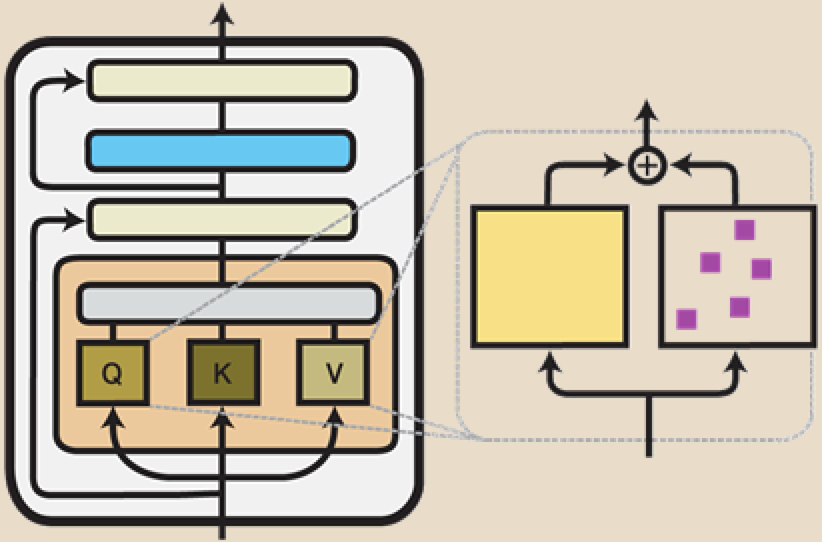
An inventory of Modules (=Adapters)



Low-rank Adapter [Hu et al. 2021]

$$\phi_j = W + A_j^\top B_j$$

$$A, B \in \mathbb{R}^{d \times r} \quad r \ll d$$

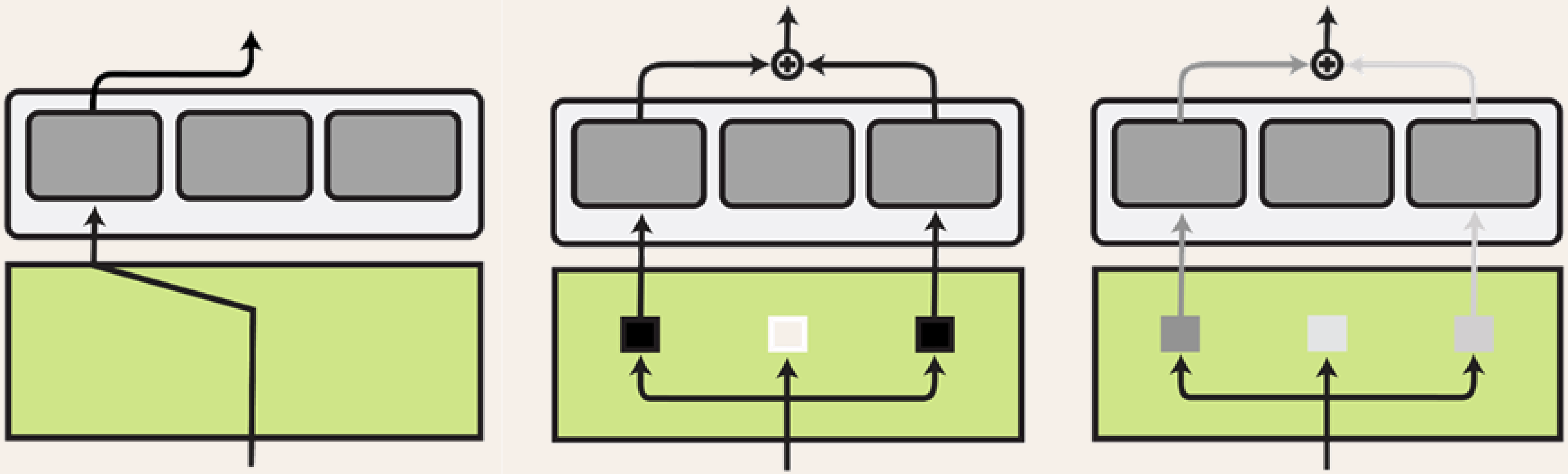


Sparse Adapter [Ansell et al. 2021]

$$\phi_j = W + A_j$$

$$A \in \mathbb{R}^{d \times d} \text{ is sparse}$$

Routing: Learned and Variable-size



(a) Fixed Routing (b) Learned Routing (Hard) (c) Learned Routing (Soft)

$$\alpha_{i,j} = \text{sigmoid}\left[\log \frac{\text{sigmoid}(\hat{\alpha}_{i,j})\, u}{(1-\text{sigmoid}(\hat{\alpha}_{i,j}))\,(1-u)}^{1/\tau}\right] \qquad u \sim \text{Uniform}(0,1).$$

Polytropon: Discovering Skills End-to-end

Core idea: jointly learn *adapters* (modules) and *variable-size* routing to fine-tune a LLM.

$$f'_i(x) = \sum_j \frac{\alpha_j}{Z} f(x; \theta, \phi_j)$$

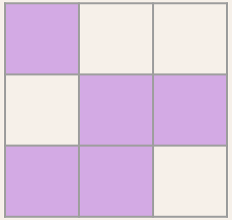
modules Θ

skills

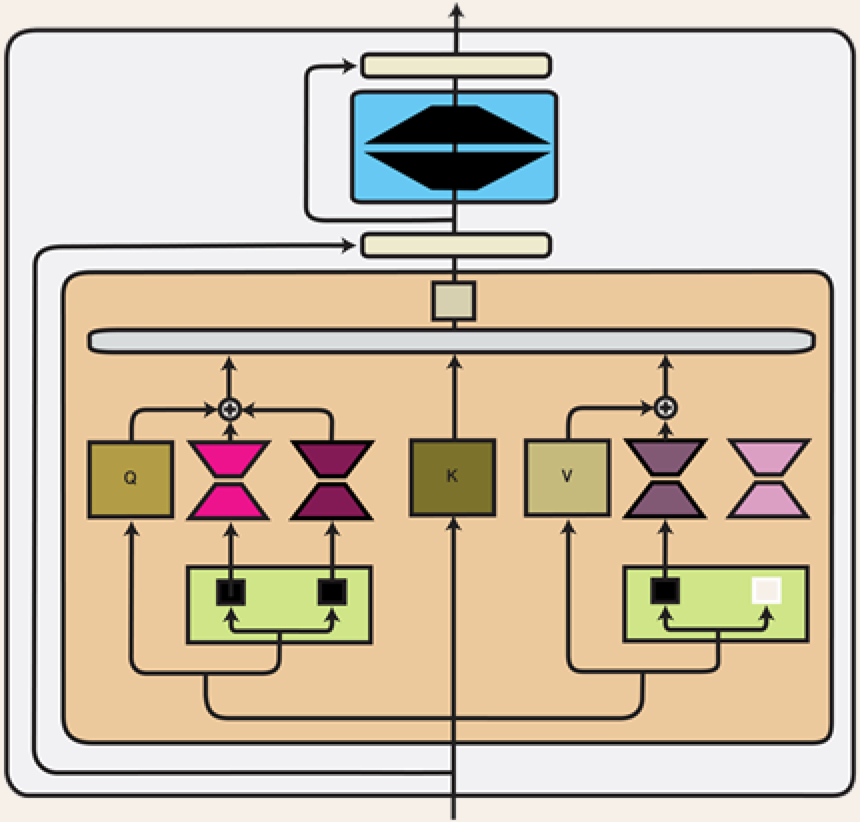
Adapter 1
Adapter 2
Adapter 3

routing *Z*

tasks



skills



Comparison with Mixture-of-Experts

	Polytropon	**MoEs**
Purpose	Few-shot generalisation	Scaling LLMs with sparsity
Training	Fine-tuning	Pre-training
Modules	Adapters	FFNs
Routing	Variable-size, task-level	Top-*k*, token-level

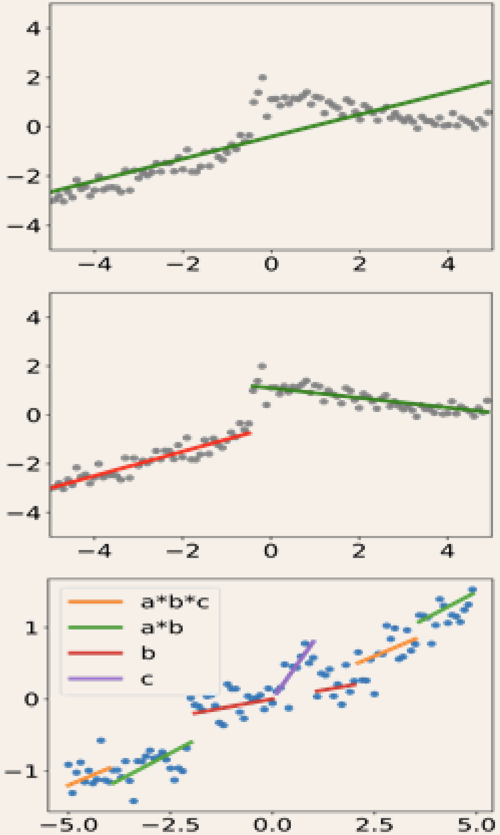
Challenges of Learned Routing

Plots courtesy of [Rosenbaum et al. (2017)]

- **Training Instability**
  Router and modules are untrained → routing dynamics never stabilise.

- **Underfitting (aka module collapse)**
  The router falls into a local optimum, choosing a few modules exclusively

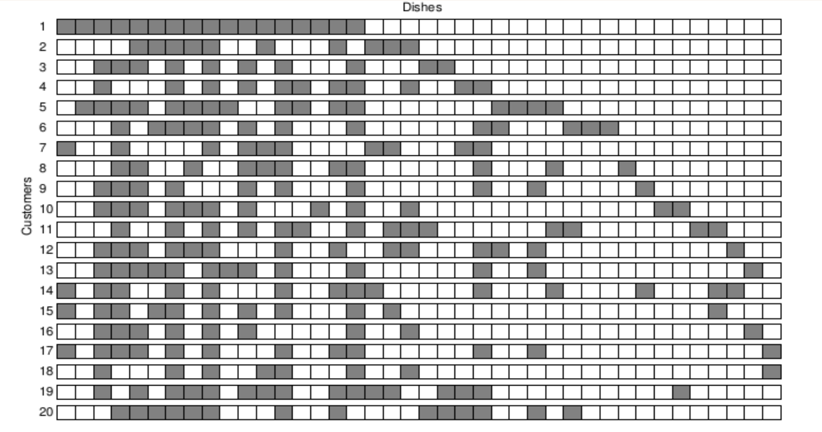
- **Overfitting**
  Risk of overfitting to the noise.



Inductive Biases

Module collapse [Rosenbaum et al. 2019]
Only a small number of modules from the inventory are selected.
Results from excessively favouring *exploitation over exploration*.

Indian Buffet Process [Griffiths and Ghahramani 2011]



Dual-speed Learning Rate

Higher for the routing function than the module parameters.
Intuition: *coarse-to-fine dynamic*.

Instruction Following in RL: BabyAI [Chevalier-Boisvert et al. 2018]

Input

***Pick up** the grey box behind you, then **go to** the green key and **open** a door.*

Actions

Open

GoNextTo(door)

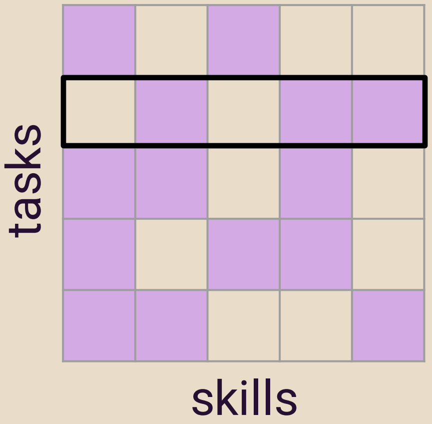
GoNextTo(key)

Drop

PickUp

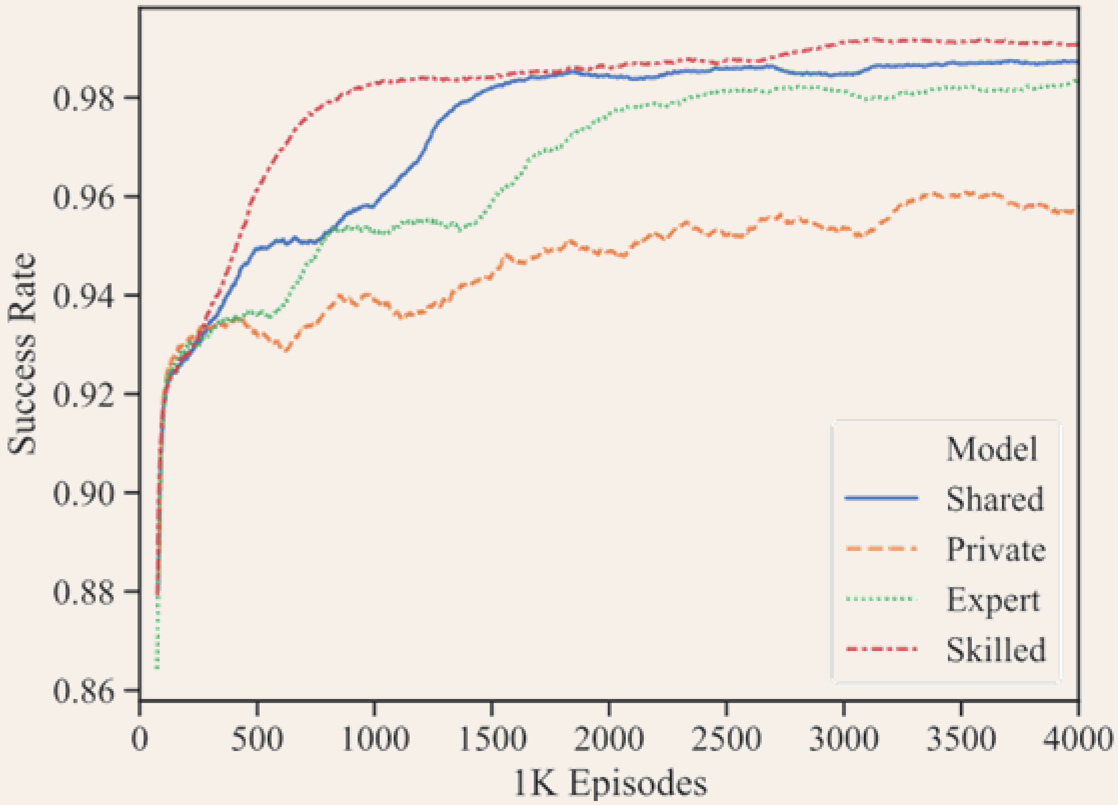
GoTo(box)

Skills

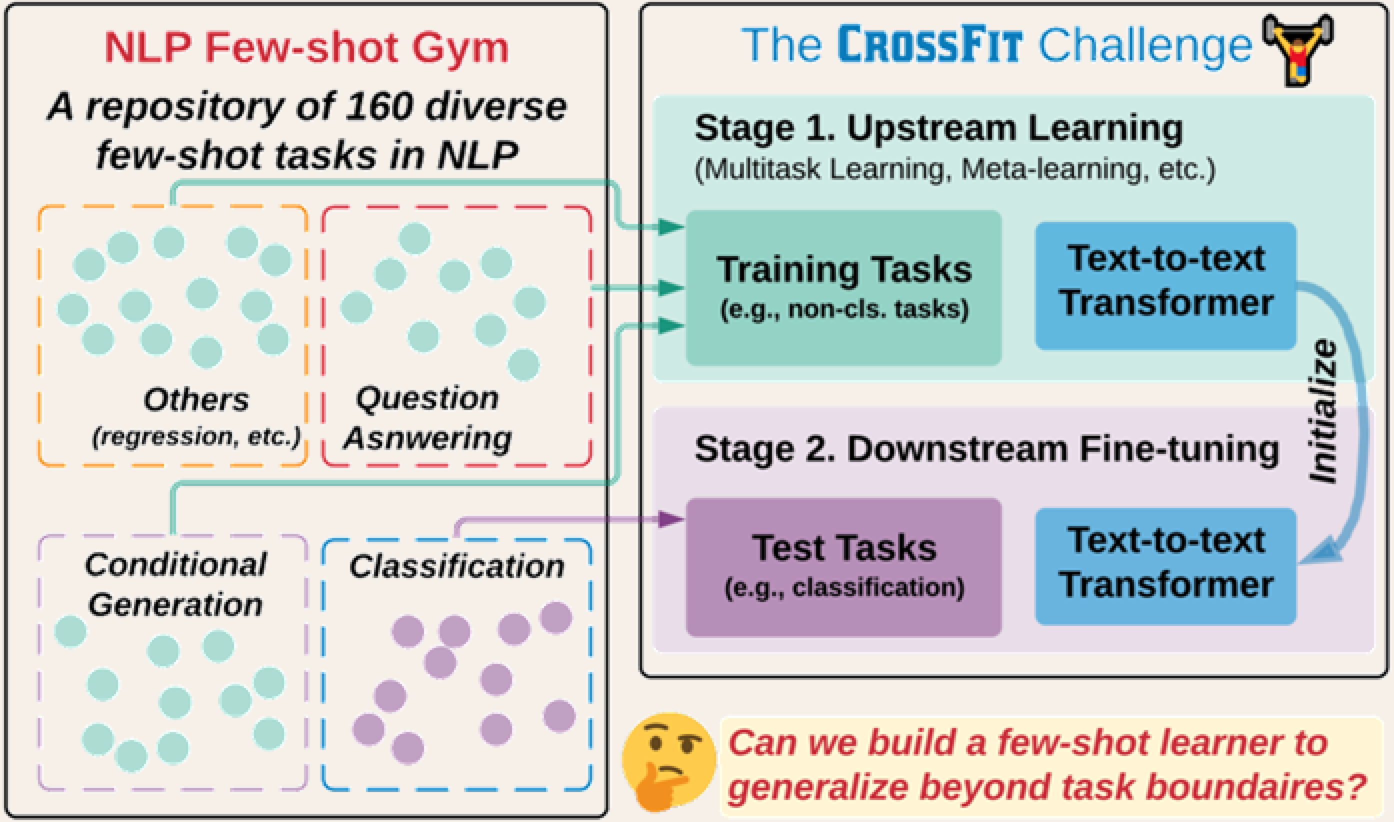


PICK UP
NAVIGATE ROOM
UNLOCK

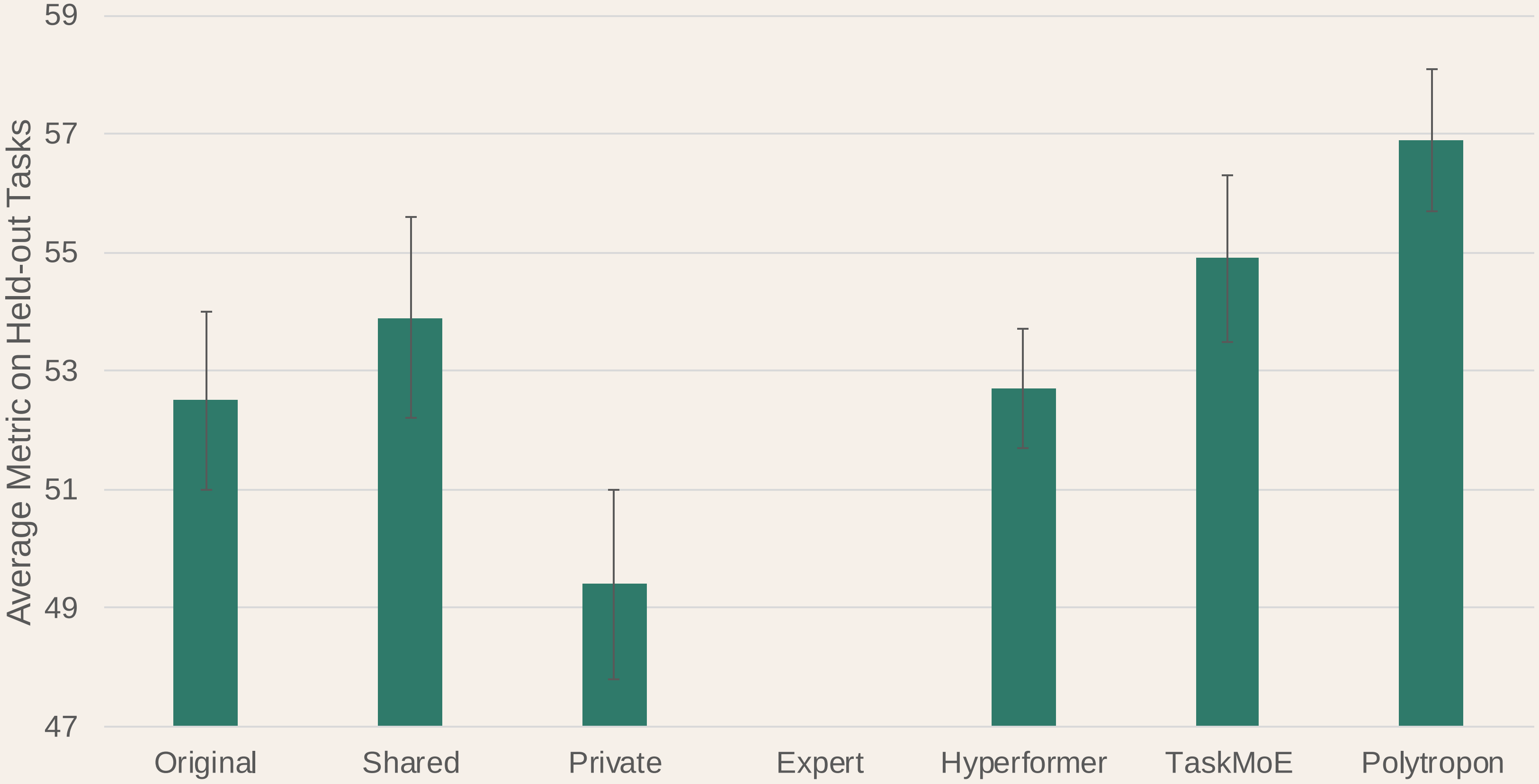
Reinforcement Learning Results: Sample Efficiency



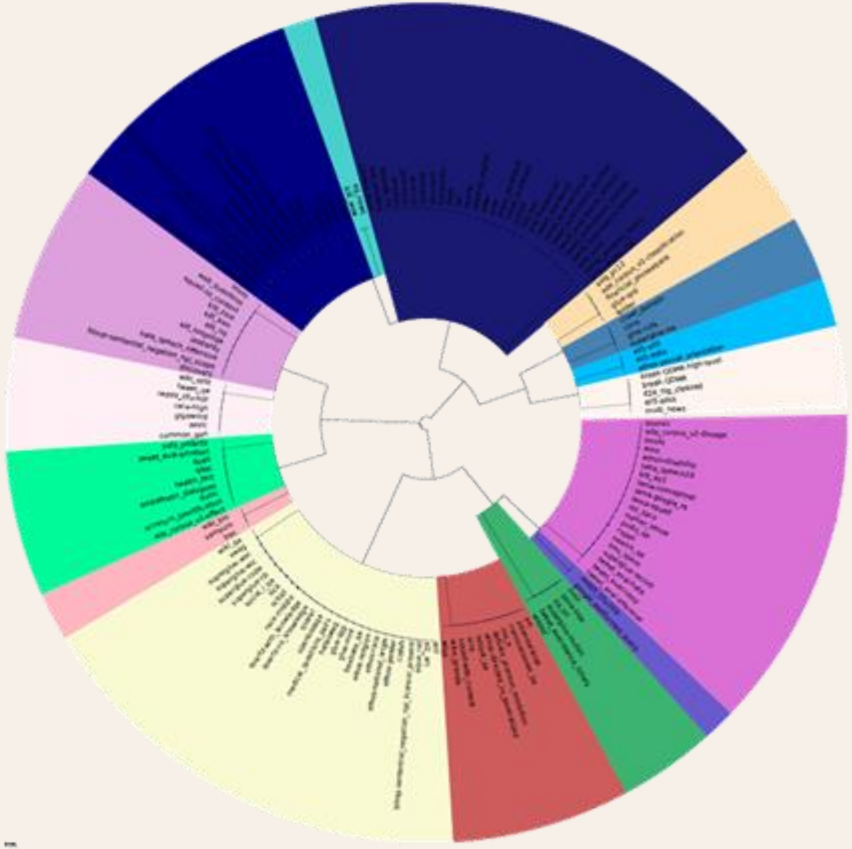
NLP: CrossFit [Ye et al. 2021]

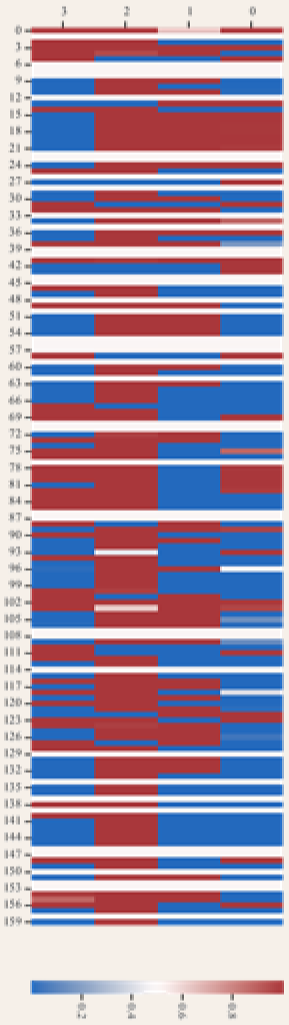


NLP Results:
Few-shot Learning in Unseen Tasks



Interpretability: Task Hierarchy





Conclusions

- Efficient multi-task learning by **implementing *skills* through adapters**.

- Inductive biases that encourage module (re)combination, e.g. allowing for **variable-size** module **routing**

- **Higher sample efficiency** in multi-task reinforcement learning and **better few-shot adaptation** in multi-task supervised learning

Code

https://github.com/microsoft/mttl

MTTL

MTTL - Multi-Task Transfer Learning

Setup

Install Python packages:

```
pip install -r requirements.txt
```

The package `promptsource` currently requires Python 3.7. Alternative versions require local installations (see their documentation).

Download the datasets:

```
bash scripts/create_datasets.sh
```

Multi-task Pre-training

The general command:

```
python pl_train.py -c $CONFIG_FILES -k $KWARGS
```

Multiple `CONFIG_FILES` can be concatenated as `file1+file2`. To modify defaults, `KWARGS` can be expressed as `key=value`.

Test Fine-Tuning

To perform finetuning for a test task, use the script `pl_finetune.py`

Hyper-parameter Search for Test Fine-Tuning

To perform an hyperparameter search for a test task, use the script `pl_finetune_tune.py`. The script will just call the functions in `pl_finetune.py` in a loop. The script itself defines hp ranges for different fine-tuning types.

Thanks for your attention!