Lecture 4
Advanced Adaptation of Big Models: Prompt-learning & Delta Tuning

Ning Ding, Shengding Hu

THUNLP

Content

- Background & Overview
- Prompt-learning
  - Template
  - Verbalizer
  - Learning Strategy
  - Applications
- Delta Tuning
  - Addition-based Methods
  - Specification-based Methods
  - Reparameterization-based Methods
  - Advanced Topics
- OpenPrompt
- OpenDelta

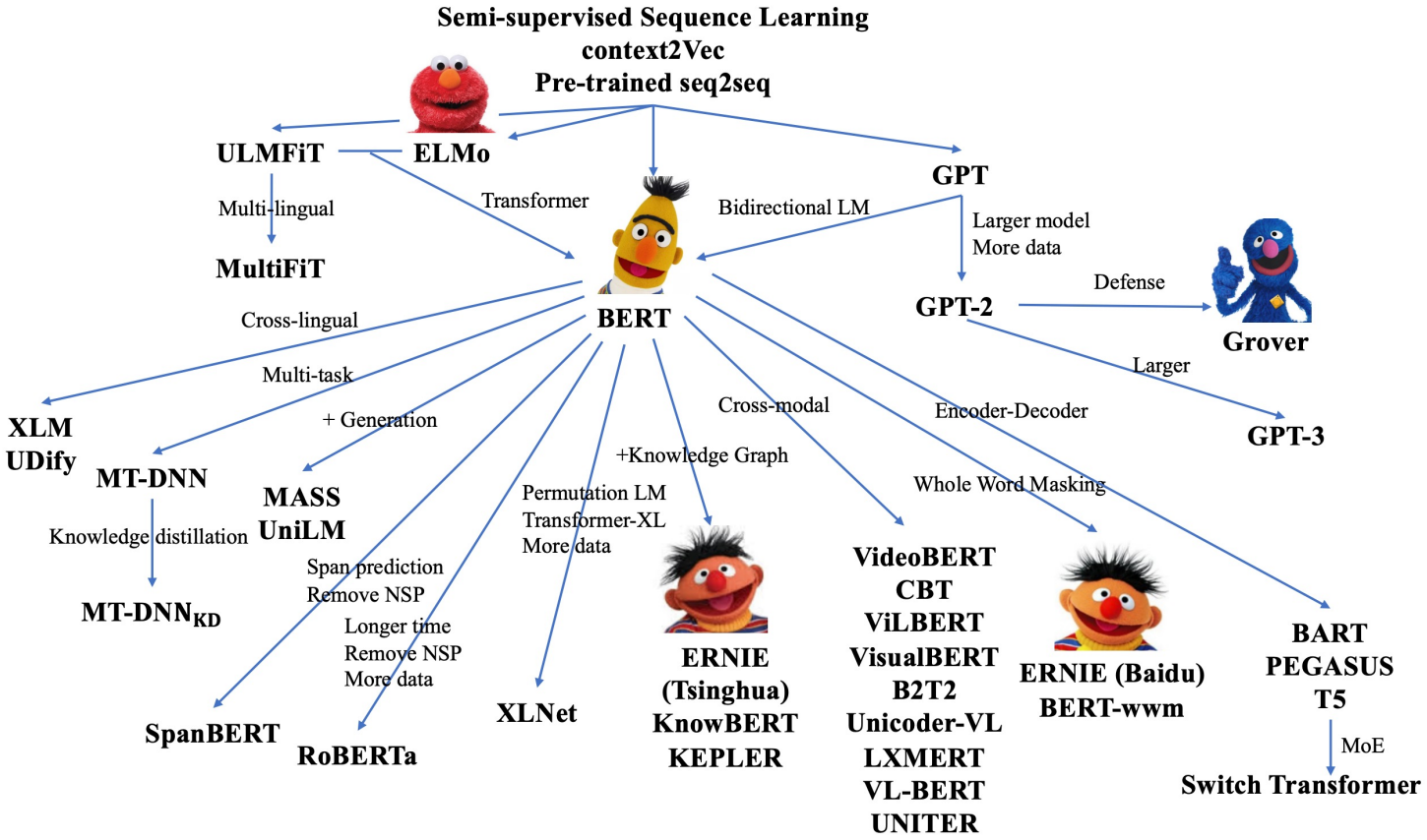
Background & Overview

Ning Ding

THUNLP

PLMs as Infrastructure

- Pre-trained Language Models are Infrastructure in NLP



The family of pre-trained language models (PLMs).

https://github.com/thunlp/PLMpapers

Downstream Tasks

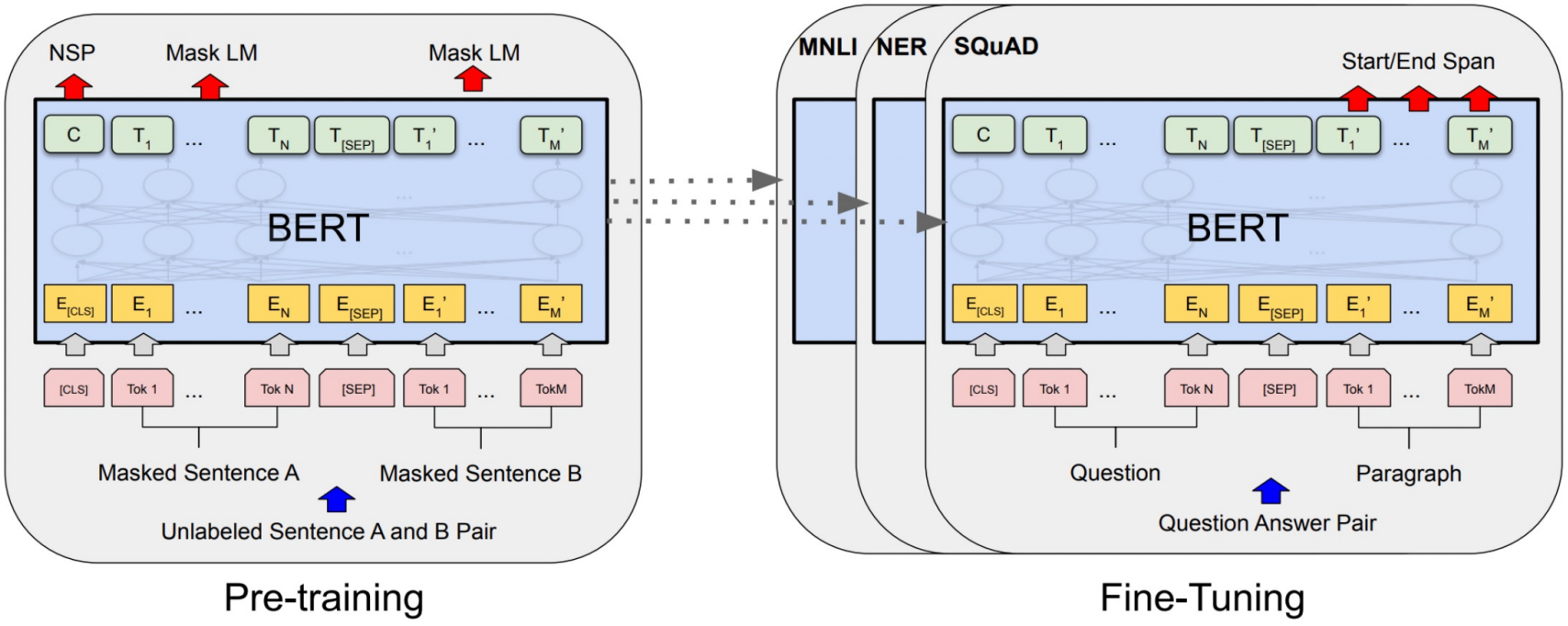
- There are Plenty of NLP tasks
  - How to adapt PLMs to them?

- Automatic speech recognition
- CCG
- Common sense
- Constituency parsing
- Coreference resolution
- Data-to-Text Generation
- Dependency parsing
- Dialogue
- Domain adaptation
- Entity linking
- Grammatical error correction
- Information extraction
- Intent Detection and Slot Filling
- Language modeling
- Lexical normalization
- Machine translation
- Missing elements
- Multi-task learning
- Multi-modal
- Named entity recognition
- Natural language inference
- Part-of-speech tagging
- Paraphrase Generation
- Question answering
- Relation prediction
- Relationship extraction
- Semantic textual similarity
- Semantic parsing
- Semantic role labeling
- Sentiment analysis
- Shallow syntax
- Simplification
- Stance detection
- Summarization
- Taxonomy learning
- Temporal processing
- Text classification
- Word sense disambiguation

http://nlpprogress.com/

Fine Tuning

- Example: BERT
  - Token representations for sequence tagging
  - [CLS] for text classification
  - Feed appropriate representations to output layers



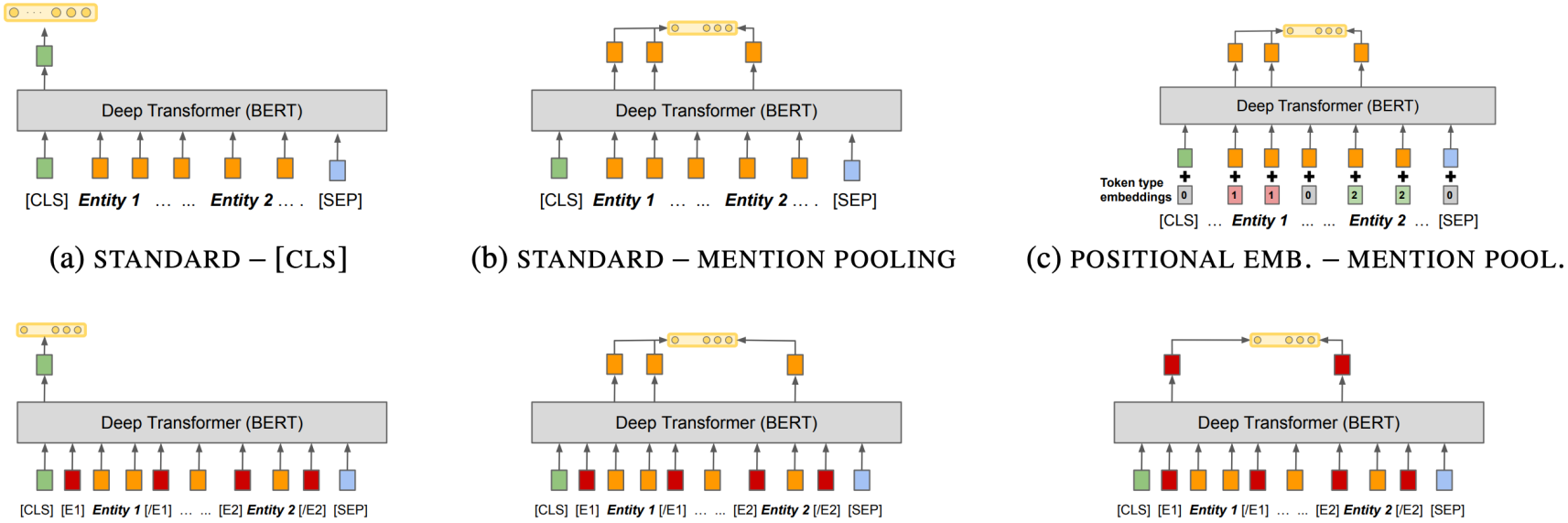
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,2019

Fine Tuning

- Example: Relation Extraction
  - Extract the relation between two marked entities

LocatePlace

Tsinghua University is located in the northwest of Beijing



(a) STANDARD – [CLS] (b) STANDARD – MENTION POOLING (c) POSITIONAL EMB. – MENTION POOL.

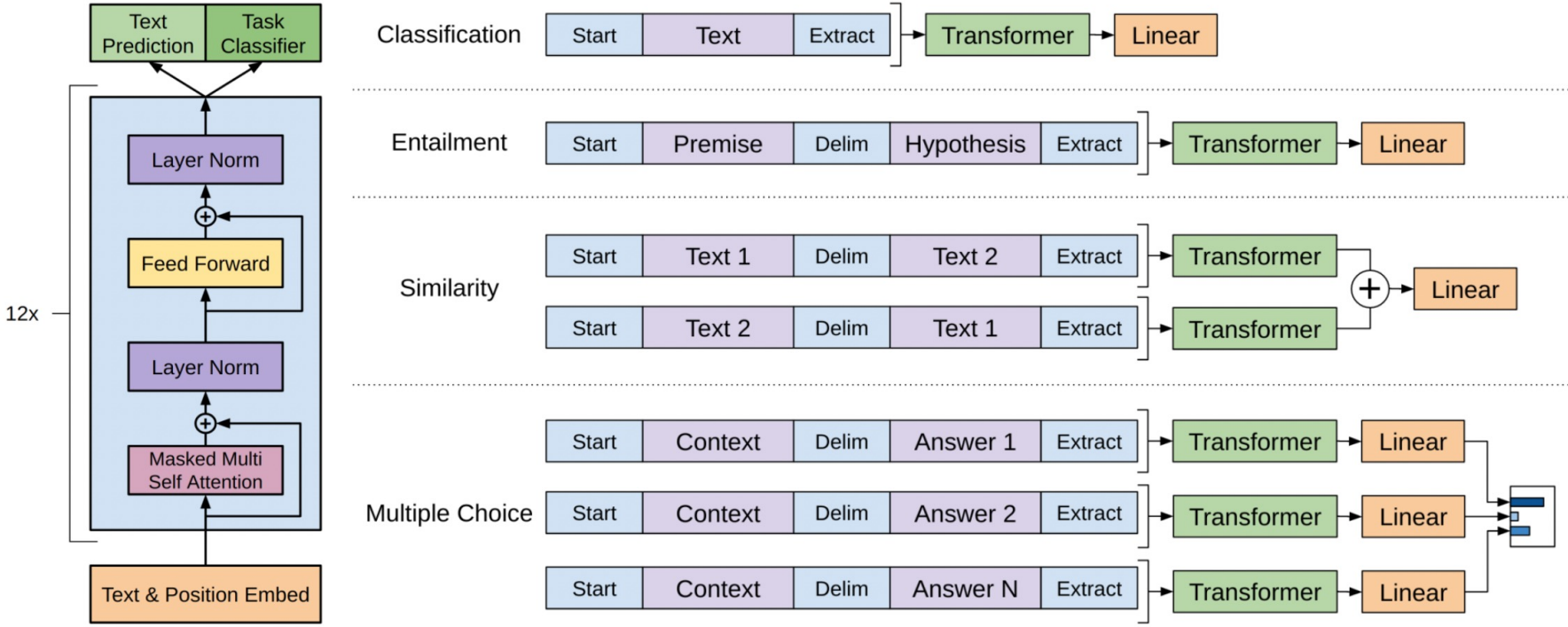
(d) ENTITY MARKERS – [CLS] (e) ENTITY MARKERS – MENTION POOL. (f) ENTITY MARKERS – ENTITY START

Matching the Blanks: Distributional Similarity for Relation Learning. 2019

Fine Tuning

- Example: GPT
  - Feed the last hidden state to a linear output layer

$$P(y|x^1,\ldots,x^m) = \texttt{softmax}(h_l^m W_y)$$



Improving Language Understanding by Generative Pre-Training, 2018.

Fine Tuning

- Example: T5
  - Encoder-decoder with 11 billion parameters
  - Cast tasks to seq2seq manner with simple demonstrations
  - A decoder is trained to output the desired tokens

D.5 QNLI

Original input:

Question: `Where did Jebe die?`

Sentence: `Genghis Khan recalled Subutai back to Mongolia soon afterwards, and Jebe died on the road back to Samarkand.`

Processed input: `qnli question: Where did Jebe die? sentence: Genghis Khan recalled Subutai back to Mongolia soon afterwards, and Jebe died on the road back to Samarkand.`

Original target: `0`

Processed target: `entailment`

Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

When it Comes to GPT-3….

- Example: GPT-3
  - Huge model with 175 billion parameters
  - No parameters are updated at all
  - Descriptions (Prompts) + Few-shot examples to generate tokens

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  cheese =>                           ← prompt
```

Few-shot

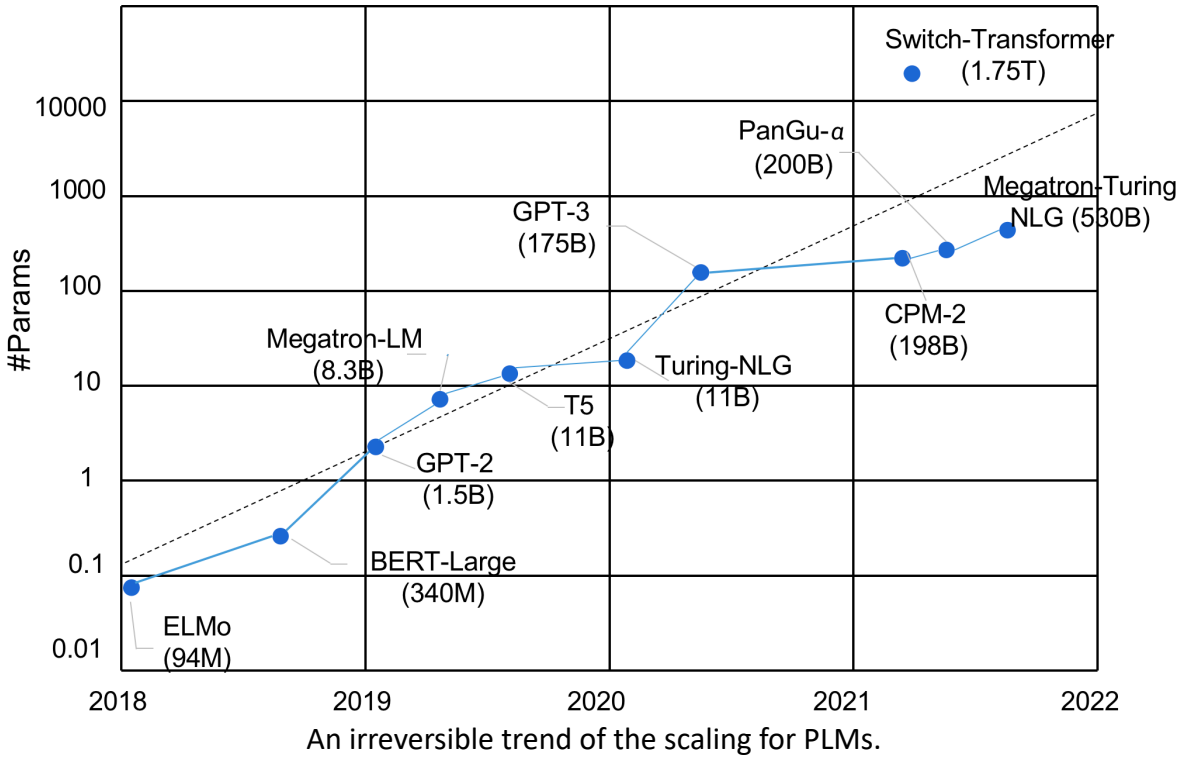
In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  sea otter => loutre de mer          ← examples
3  peppermint => menthe poivrée        ←
4  plush girafe => girafe peluche      ←
5  cheese =>                           ← prompt
```


Language Models are Few-Shot Learners, 2020.

An Irreversible Trend: Model Scaling

- Larger PLMs Tend to Lead Better Performance
  - Better natural language understanding capability
  - Better quality for natural language generation
  - Better capacity to continually learn novel knowledge



An irreversible trend of the scaling for PLMs.

An Irreversible Trend: Difficult Tuning

- How to Adapt Large-scale PLMs?
  - A Predominant Way — Fine-tuning
  - Prohibitive Computing: update all the parameters;
  - Prohibitive Storage: retaining separate instances for different tasks;
  - Poor generalization with supervision is insufficient
  - Results in scarce use for large-scale PLMs in research

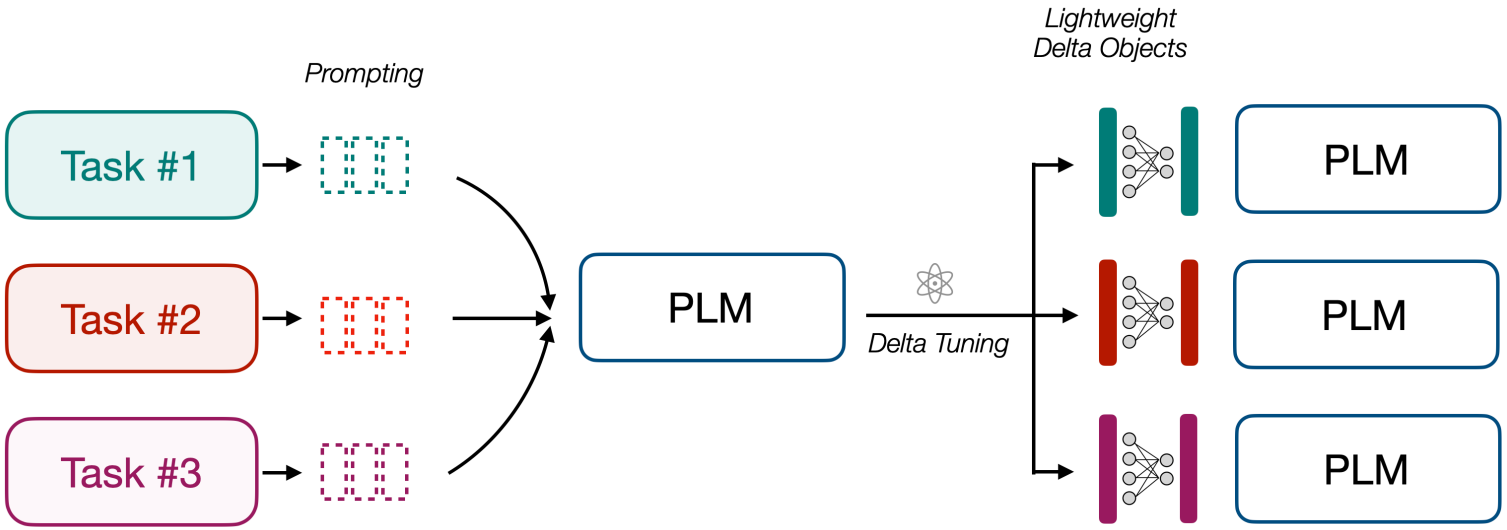
Venue	No PLMs	Small PLMs	Large PLMs	Per. of Large PLMs
ACL 2021	41	151	8	4.0%
EMNLP 2021	46	150	4	2.0%
NAACL 2021	37	158	5	2.5%
ACL 2020	107	92	1	0.5%
EMNLP 2020	62	137	1	0.5%

The usage of models of different sizes in research published in NLP conferences, the statistic is based on 1000 randomly selected papers. Large PLMs are defined as PLMs with over 1 billion parameters.

Delta Tuning: A Comprehensive Study of Parameter Efficient Methods for Pre-trained Language Models, 2022.

Advanced Model Adaptation

- Effective Model Adaptation
  - Task&Data-wise: Use prompt-learning to enhance the few-shot learning capability by bridging the gap between model tuning and pre-training
  - Optimization-wise: Use delta tuning to stimulate models with billions of parameters with optimization of a small portion of parameters



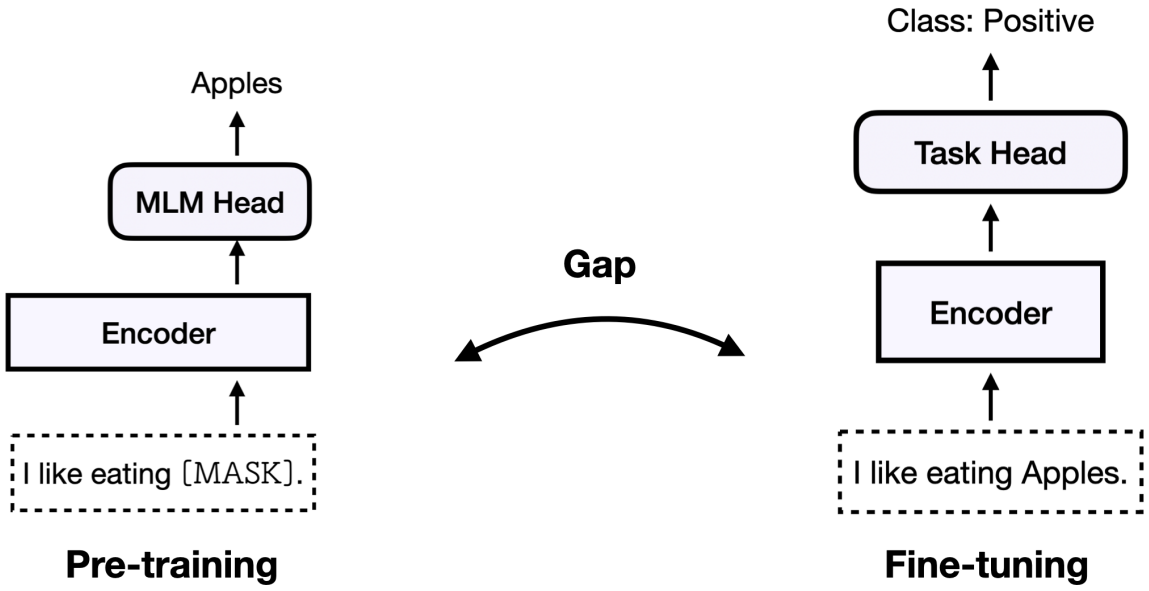
Prompt-learning

Ning Ding

THUNLP

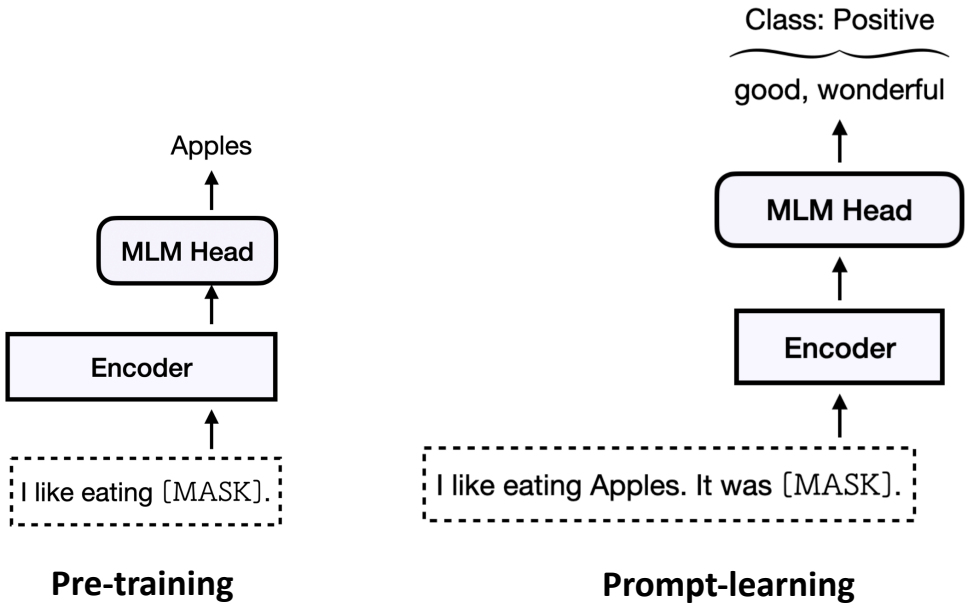
Prompt-learning

- Prompt-learning
  - Use PLMs as base encoders
  - Add additional neural layers for specific tasks
  - Tune all the parameters
  - There is a GAP between pre-training and fine-tuning



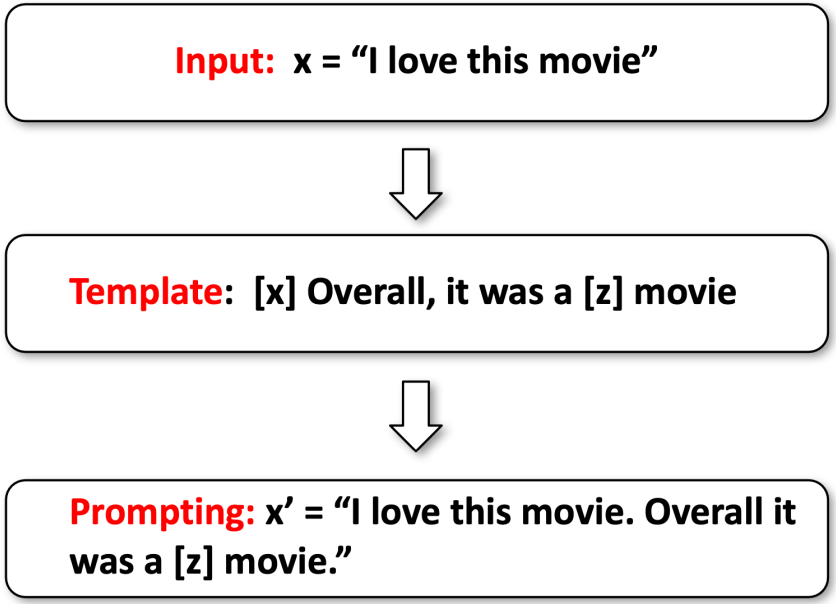
Prompt-learning

- Prompt-learning
  - Use PLMs as base encoders
  - Add additional context (template) with a [MASK] posistion
  - Project labels to label words (verbalizer)
  - Bridge the GAP between pre-training and fine-tuning



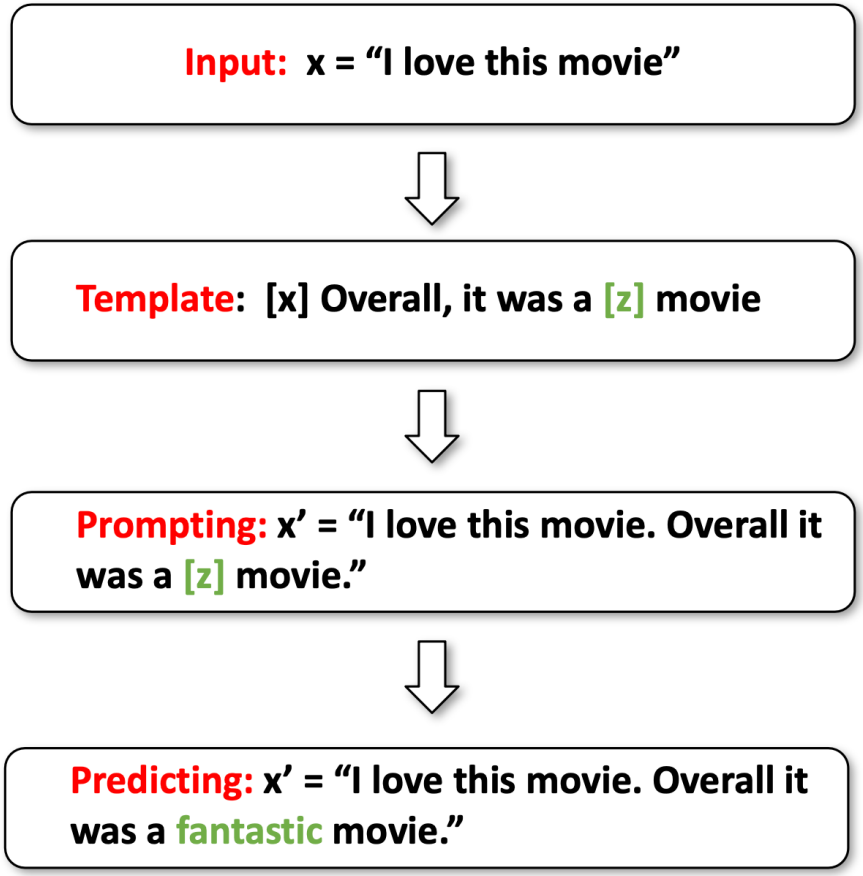
Prompt-learning: Example

- Sentiment Classification
  - Prompting with a Template



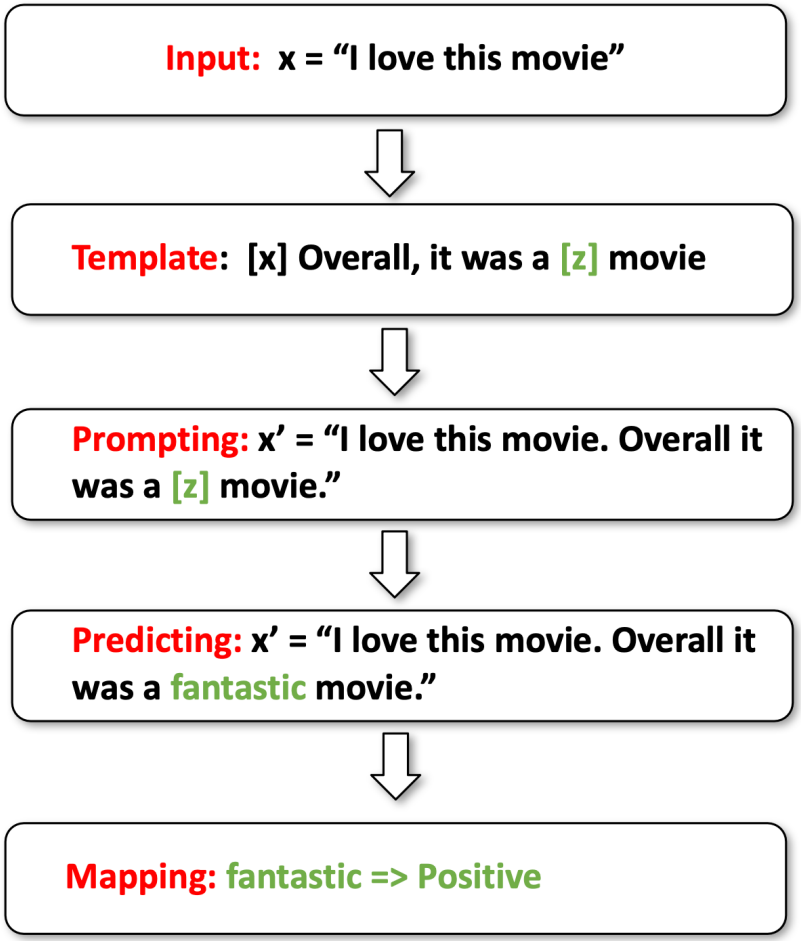
Prompt-learning: Example

- Sentiment Classification
  - Predict an answer



Prompt-learning: Example

- Sentiment Classification
  - Map the answer to a class label with a Verbalizer



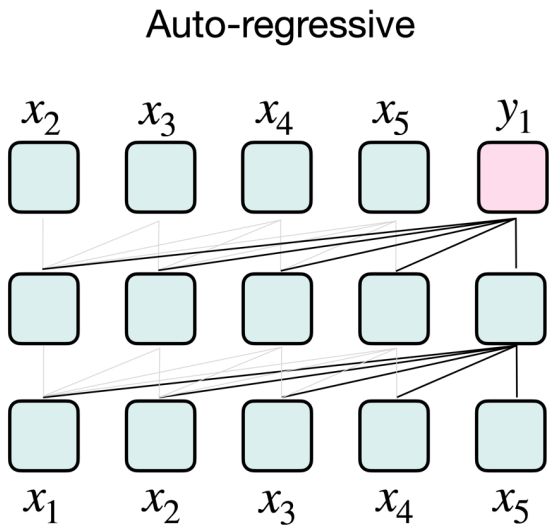
Prompt-learning: Considerations

- Pre-trained Model
  - Auto-regressive (GPT-1, GPT-2, GPT-3; OPT…)
  - Masked Language Modeling (BERT, RoBERTa, DeBERTa)
  - Encoder-Decoder (T5, BART)
- Template
  - Manually Design
  - Auto Generation
  - Textual or Continuous…
- Verbalizer
  - Manually Design
  - Expanding by external knowledge…

Prompt-learning: Considerations

- Pre-trained Model
  - Auto-regressive (GPT-1, GPT-2, GPT-3; OPT…)
  - Suitable for super-large pre-trained models
  - Autoregressive Prompt

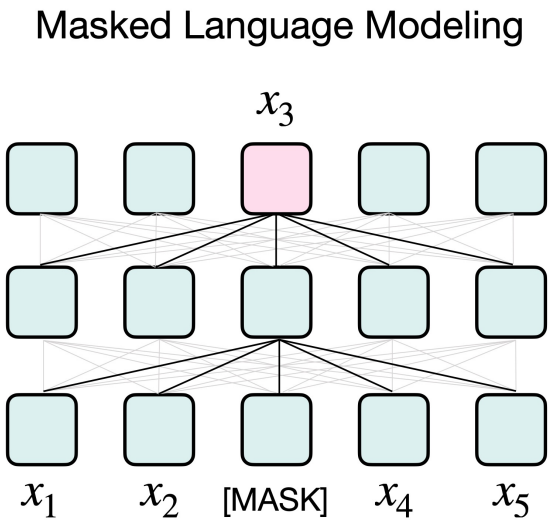
I love this movie. It is [MASK]



Prompt-learning: Considerations

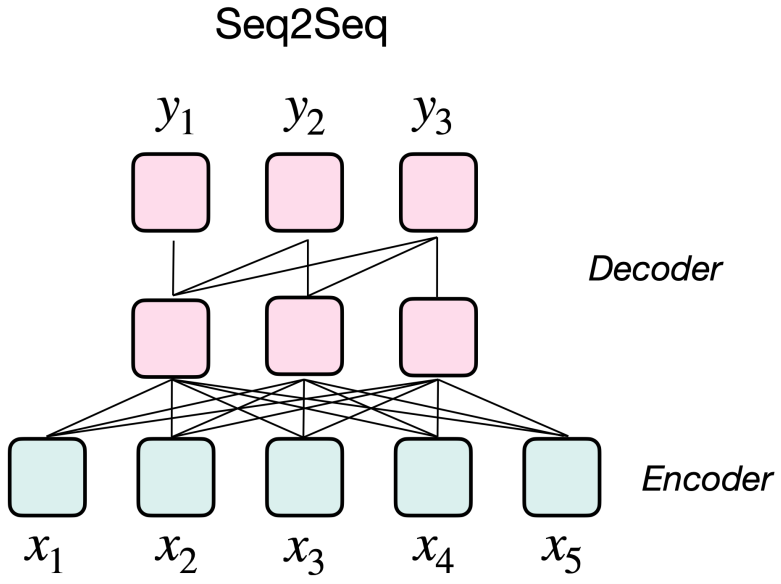
- Pre-trained Model
  - Masked Language Modeling (BERT, RoBERTa, DeBERTa)
  - Suitable for natural language understanding
  - Cloze-style Prompt

I love this movie. It is a [MASK] movie.



Prompt-learning: Considerations

- Pre-trained Model
  - Encoder-Decoder (T5, BART)
  - Bidirectional attention for encoder
  - Autoregressive for decoder



Template

- Template Construction
  - Manually Design based on the characteristics of the task
  - Auto Generation with search or optimization
  - Textual or Continuous
  - Structured, incorporating with rules

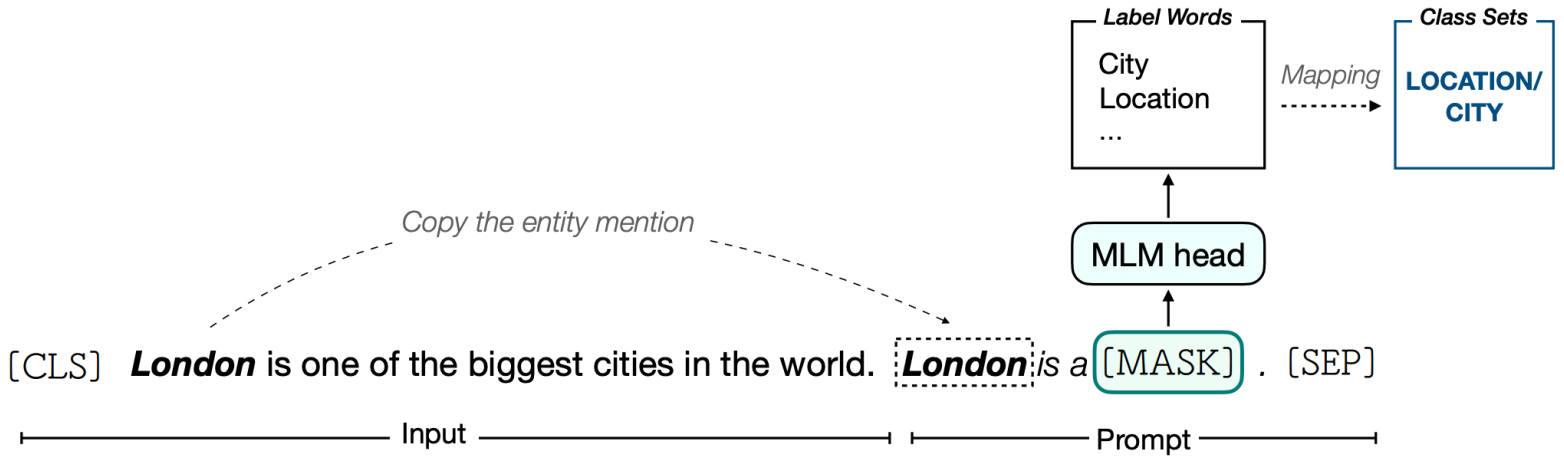
Template

- Examples of Tasks and Corresponding Templates

Type	Task	Input ([X])	Template	Answer ([Z])
Text CLS	Sentiment	I love this movie.	[X] The movie is [Z].	great fantastic ...
	Topics	He prompted the LM.	[X] The text is about [Z].	sports science ...
	Intention	What is taxi fare to Denver?	[X] The question is about [Z].	quantity city ...
Text-span CLS	Aspect Sentiment	Poor service but good food.	[X] What about service? [Z].	Bad Terrible ...
Text-pair CLS	NLI	[X1]: An old man with ... [X2]: A man walks ...	[X1]? [Z], [X2]	Yes No ...
Tagging	NER	[X1]: Mike went to Paris. [X2]: Paris	[X1][X2] is a [Z] entity.	organization location ...
Text Generation	Summarization	Las Vegas police ...	[X] TL;DR: [Z]	The victim ... A woman ... ...
	Translation	Je vous aime.	French: [X] English: [Z]	I love you. I fancy you. ...

Template

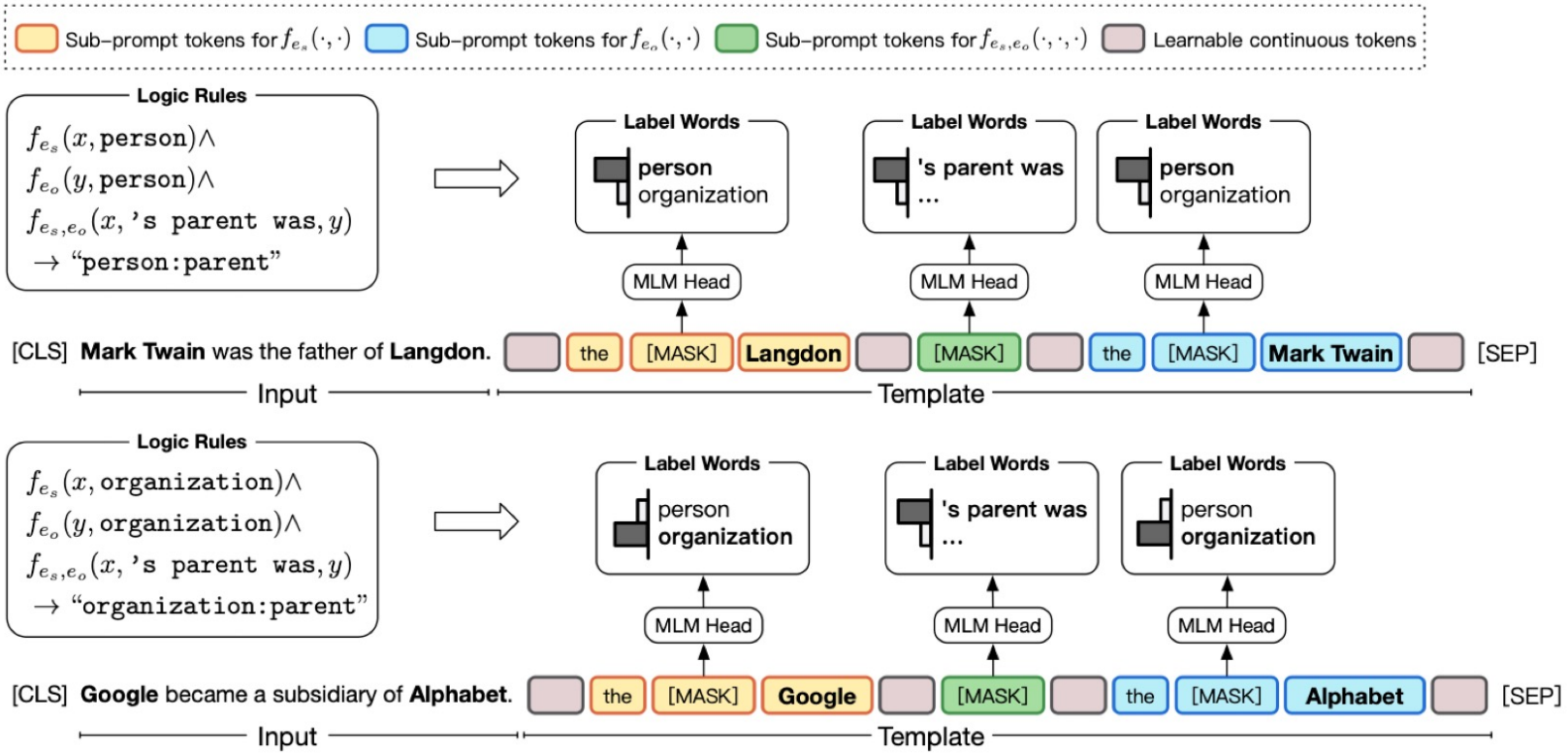
- Template: Extract World Knowledge
  - Copy the entity in the Template
  - Predict fine-grained entity types
  - Extract world knowledge



Prompt-learning for fine-grained entity typing. 2021

Template

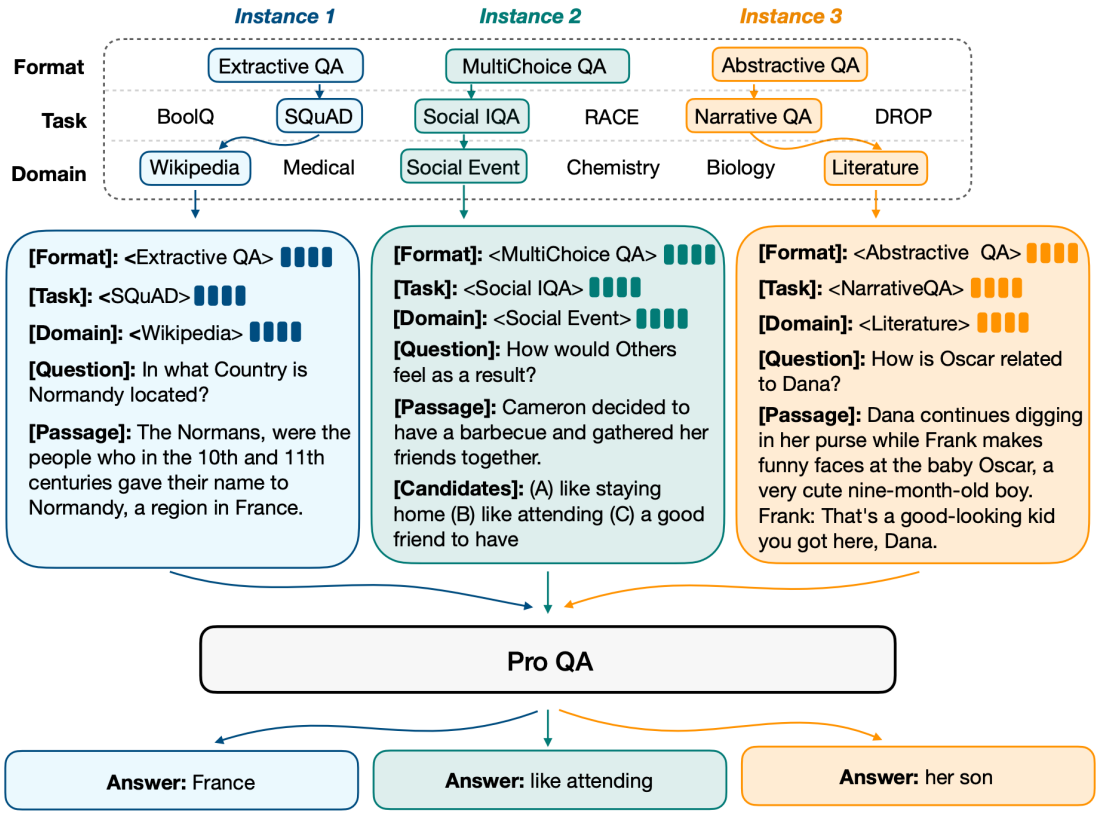
- Template: Incorporating Rules and Logic
  - Prompt-learning with logic-enhanced templates



PTR: Prompt Tuning with Rules for Text Classification. 2021

Template

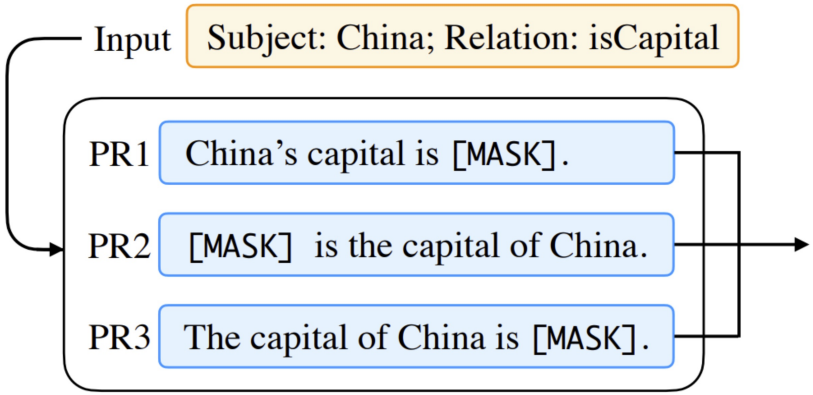
- Structured Template
  - Key-value Pairs for all the prompts
  - Organize different tasks to a structured format



ProQA: Structural Prompt-based Pre-training for Unified Question Answering. 2021

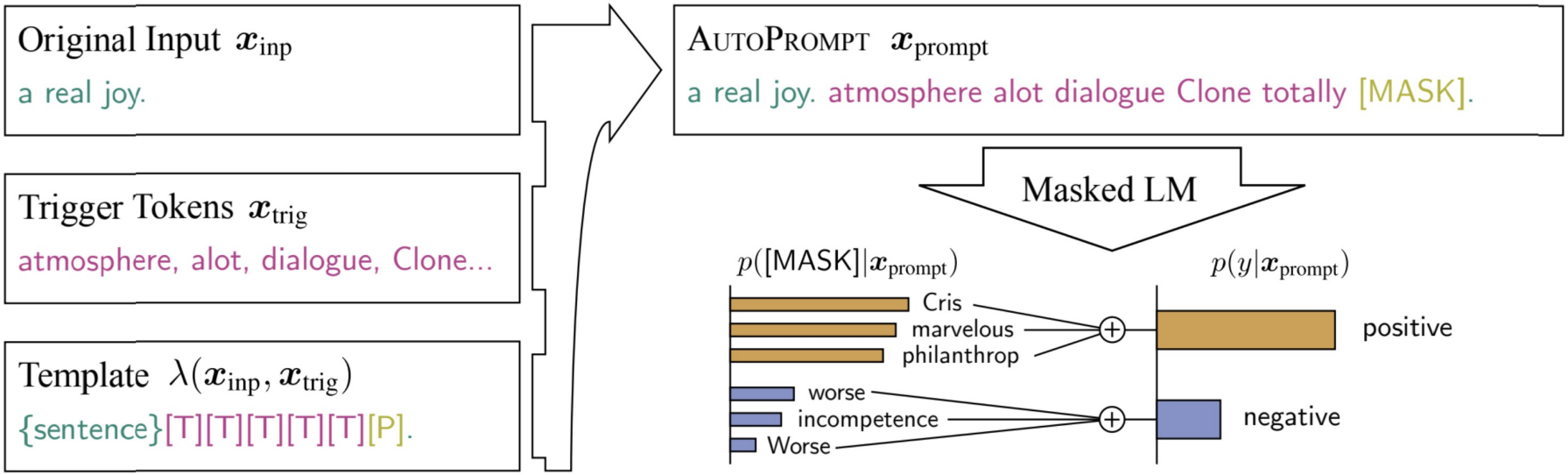
Template

- Ensembling Templates
  - Use multiple different prompts for an input instance
  - Alleviate the cost of prompt engineering
  - Stabilize performance on tasks
- Methods
  - Uniform Averaging
  - Weighted Averaging



Template

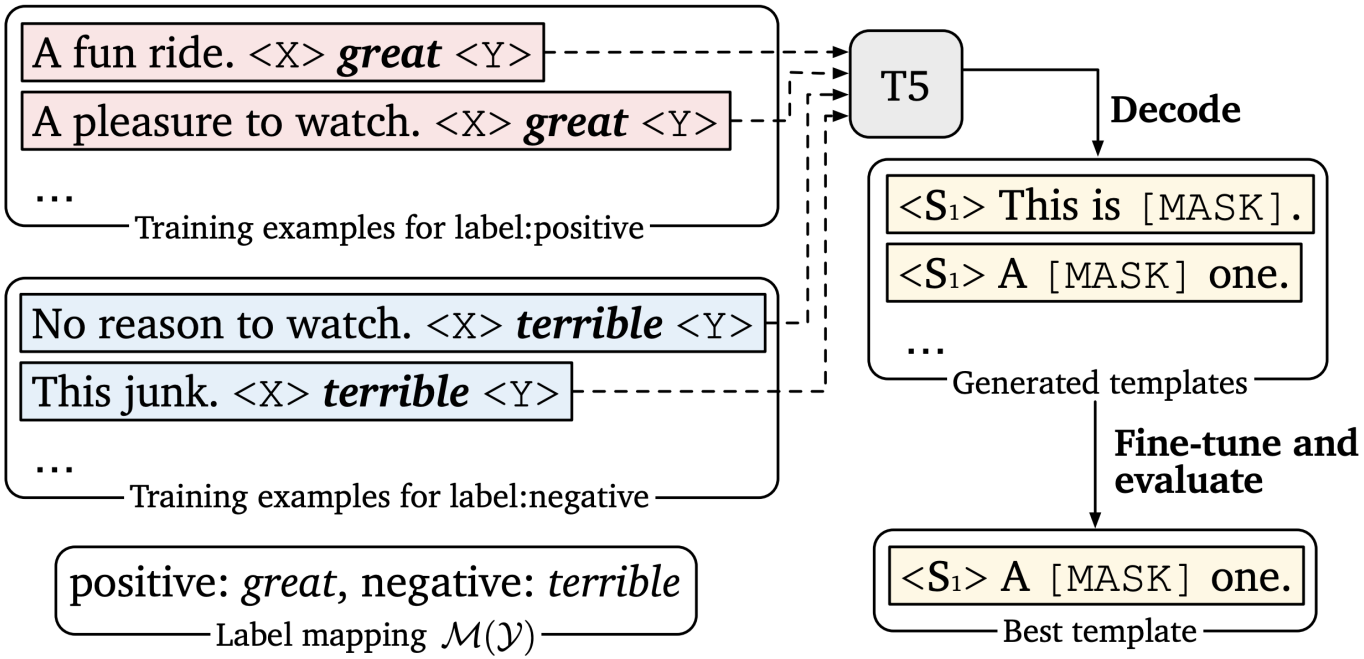
- Template: Automatic Search
  - Gradient-based search of prompts based on existing words



AUTOPROMPT: Eliciting Knowledge from Language Models with Automatically Generated Prompts. 2020

Template

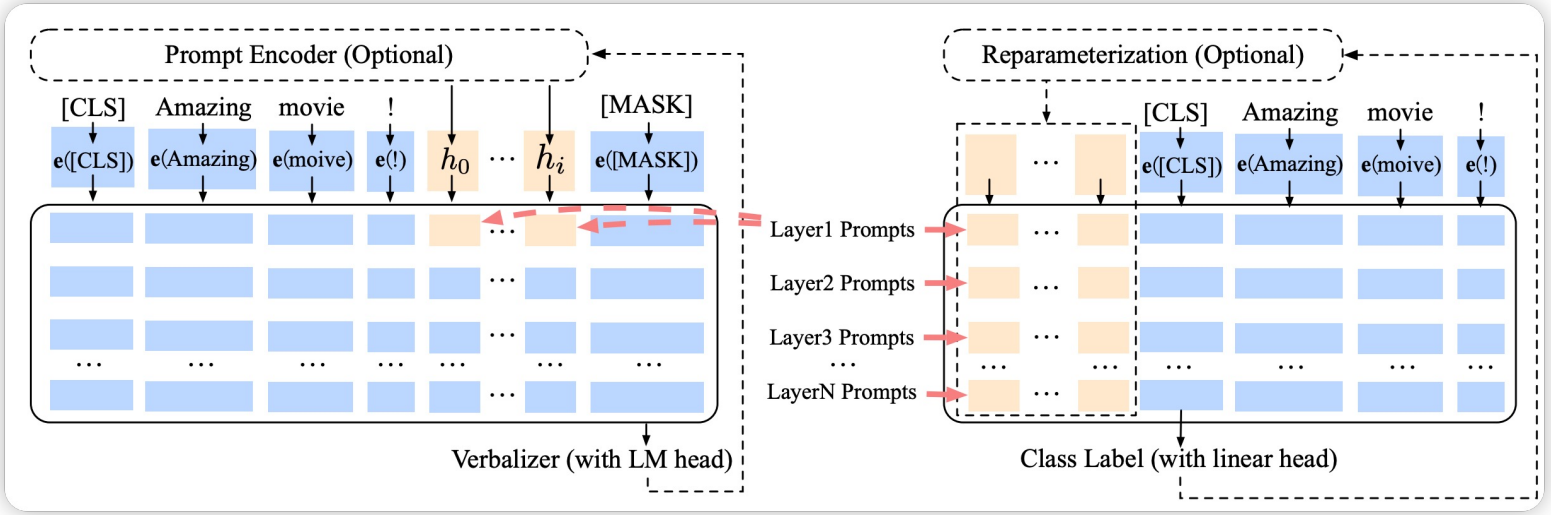
- Template: Automatic Search
  - Use a encoder-decoder model to generate prompts



LM-BFF: Making Pre-trained Language Models Better Few-shot Learners. 2021

Prompting with Continuous Templates

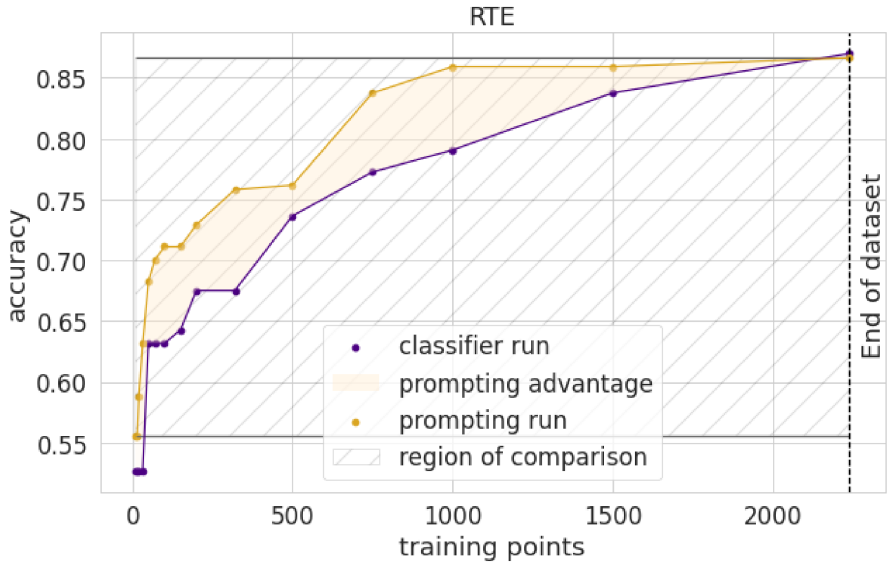
- Optimization of Continuous Prompts
  - Generative models for NLU by optimizing continuous prompts
  - P-tuning v1: prompts to the input layer (with Reparameterization)
  - P-tuning v2: prompts to every layer (like prefix-tuning)

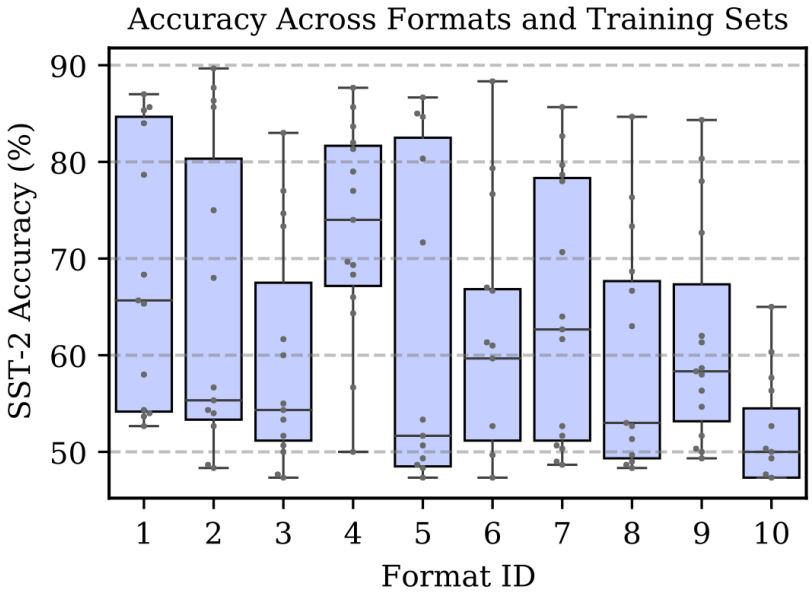


GPT Understands, Too..
P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks

Analysis

- Performance of Prompt-learning
  - Exdraordinary few-shot learning performance
  - Huge impact from the templates





How Many Data Points is a Prompt Worth? 2021
Calibrate Before Use: Improving Few-Shot Performance of Language Models. 2021

Prompt-learning: Verbalizer

- Verbalizer
  - Mapping: Answer -> Unfixed Labels

> Positive: great, wonderful, good..
> Negative: terrible, bad, horrible…

  - Tokens: One or more tokens in the pre-trained language model vocabulary
  - Chunks: Chunks of words made up of more than one tokens
  - Sentence: Sentences in arbitrary length
- Construction
  - Hand-crafted
  - Auto-generation

Verbalizer

- Verbalizer Construction
  - Manually design with human prior knowledge
  - Start with an initial label word, paraphrase & expand
  - Start with an initial label word, use external knowledge & expand
  - Decompose the label with multiple tokens
  - Virtual token and optimize the label embedding

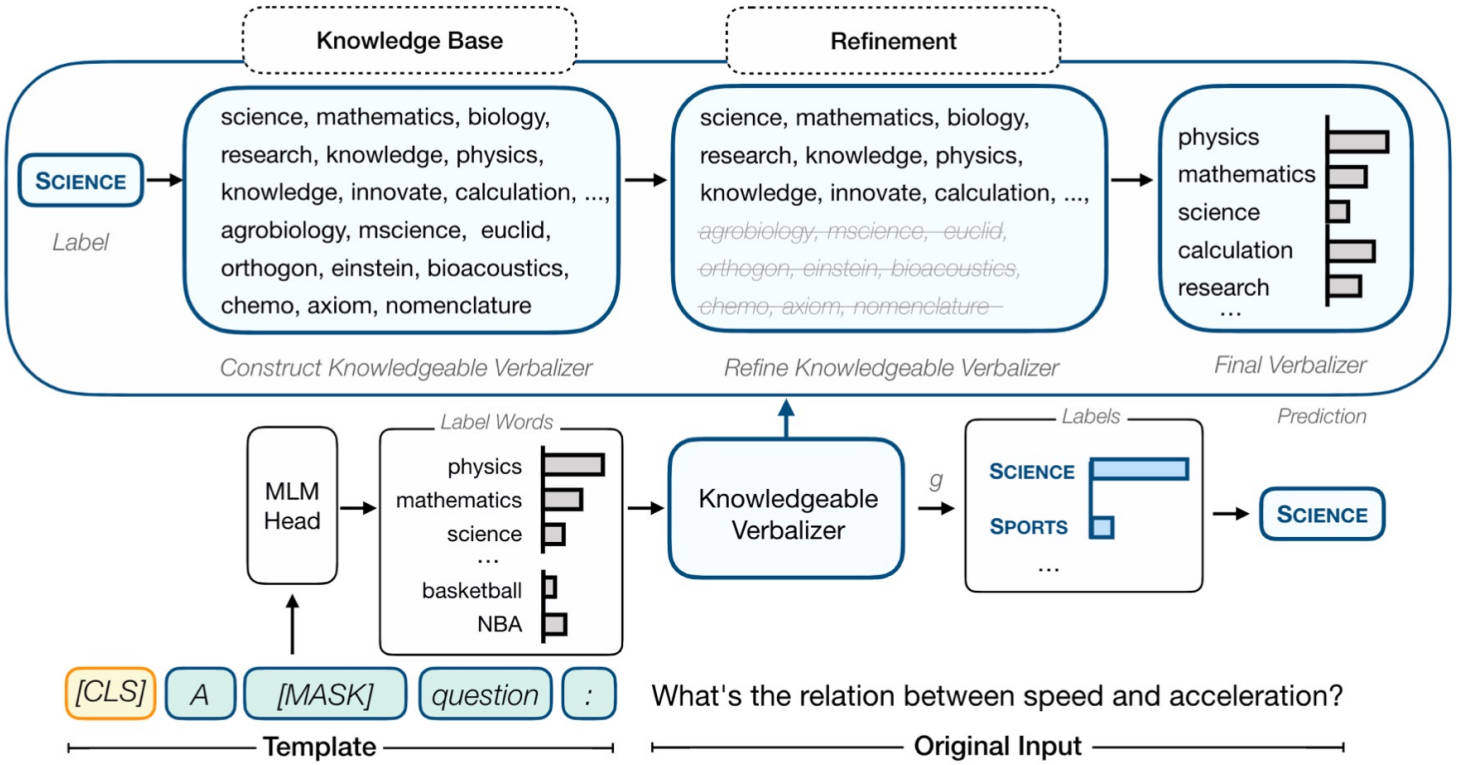
Verbalizer

- Examples of Tasks and Corresponding Verbalizers

Type	Task	Input (`[X]`)	Template	Answer (`[Z]`)
Text CLS	Sentiment	I love this movie.	`[X]` The movie is `[Z]`.	great fantastic ...
	Topics	He prompted the LM.	`[X]` The text is about `[Z]`.	sports science ...
	Intention	What is taxi fare to Denver?	`[X]` The question is about `[Z]`.	quantity city ...
Text-span CLS	Aspect Sentiment	Poor service but good food.	`[X]` What about service? `[Z]`.	Bad Terrible ...
Text-pair CLS	NLI	`[X1]`: An old man with ... `[X2]`: A man walks ...	`[X1]`? `[Z]`, `[X2]`	Yes No ...
Tagging	NER	`[X1]`: Mike went to Paris. `[X2]`: Paris	`[X1][X2]` is a `[Z]` entity.	organization location ...
Text Generation	Summarization	Las Vegas police ...	`[X]` TL;DR: `[Z]`	The victim ... A woman ... ...
	Translation	Je vous aime.	French: `[X]` English: `[Z]`	I love you. I fancy you. ...

Verbalizer

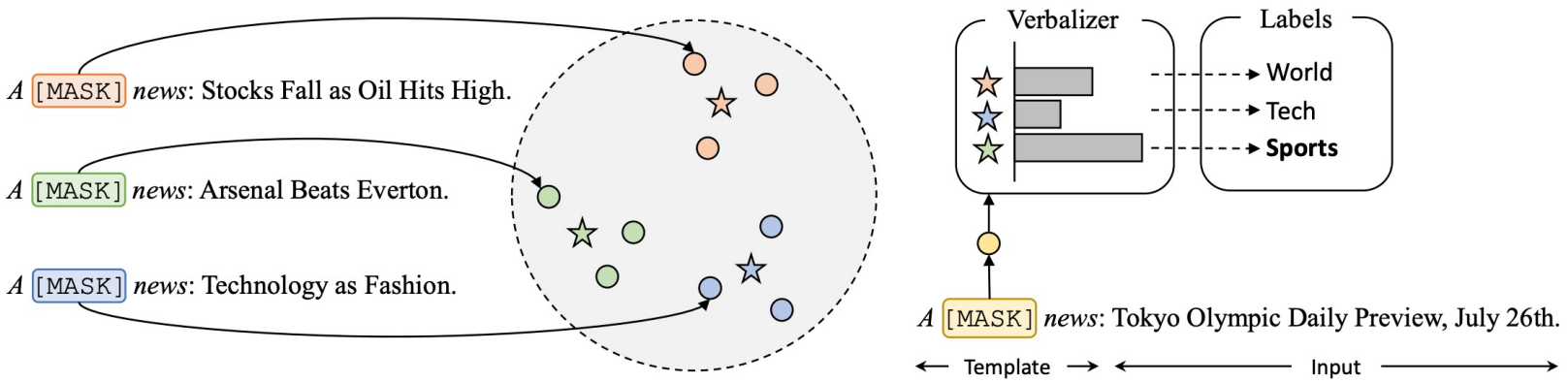
- Knowledgeable Prompting
  - Label -> Words
  - Use External Knowledge to expand the label words



Knowledgeable prompt-tuning: Incorporating knowledge into prompt verbalizer for text classification. 2021

Verbalizer

- Virtual Tokens as Label Words
  - Project the hidden states of [MASK] tokens to the embedding space and learn prototypes
  - The learned prototypes constitute the verbalizer and map the PLM outputs to corresponding labels.



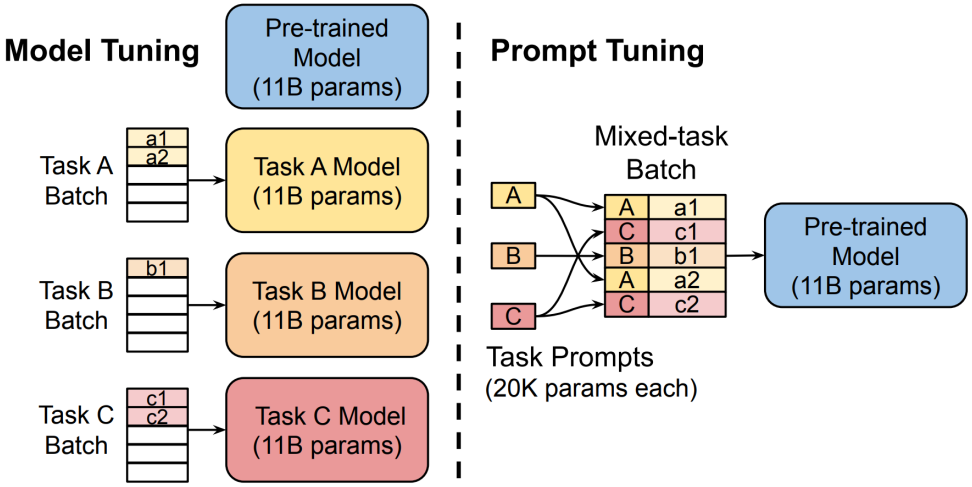
Prototypical Verbalizer for Prompt-based Few-shot Tuning. 2021

Learning Strategy

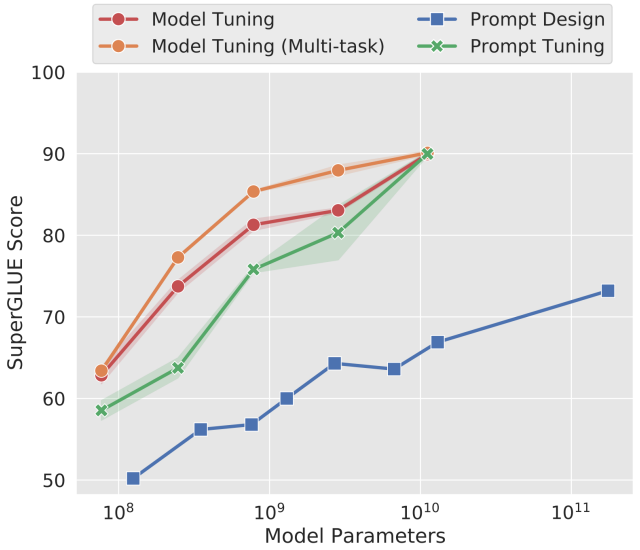
- The Evolvement
  - Traditional: Learning from scratch;
  - After BERT: Pre-training-then-fine-tuning;
  - T5: Pre-training-then-fine-tuning with text-to-text format;
  - GPT: Pre-training, then use prompt & in-context for zero- and few- shot;
- Prompt-learning Introduces New Learning Strategies
  - Pre-training, prompting, optimizing all the parameters (middle-size models, few-shot setting)
  - Pre-training, adding soft prompts, freezing the model and optimizing the prompt embeddings (delta tuning perspective)
  - Pre-training with prompted data, zero-shot inference (Instruction tuning& T0)

Prompting with Frozen Pre-trained Model

- Prompt-Tuning
  - Injecting soft prompts (embeddings) to the input layer
  - Extraordinary power of scale
  - Comparable results to fine-tuning conditioned on 11B PLM
  - Essentially a parameter efficient (delta tuning) method



An illustration of prompt-tuning.

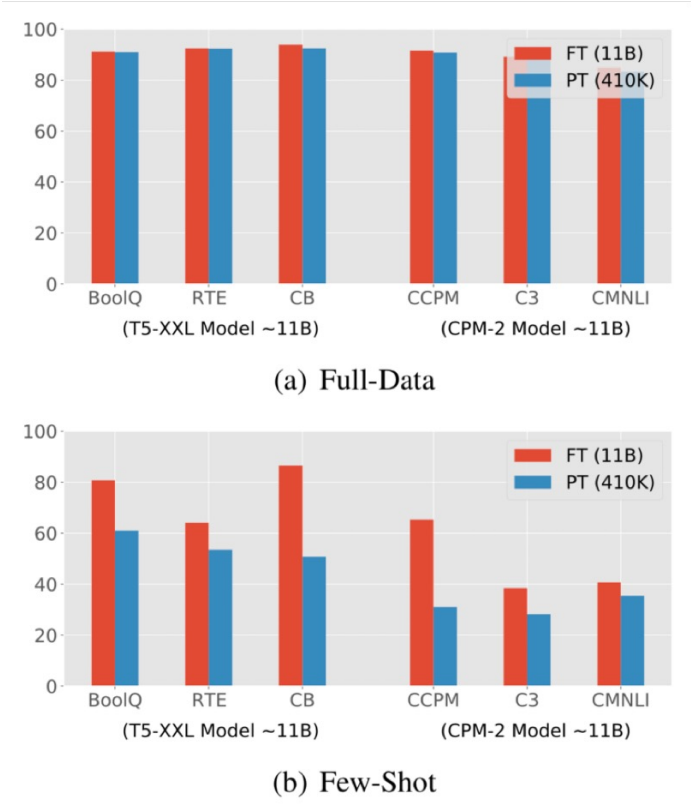


The power of scale of prompt-tuning.

The Power of Scale for Parameter-Efficient Prompt Tuning, 2021.

Pre-trained Prompt Tuning

- Injecting Prompts to Pre-training
  - Full data: fine-tuning and prompt-tuning are comparable
  - Few data: only tuning prompts have poor performance
  - The vanilla prompt tuning cannot generalize effectively in low-data situation
  - Injecting soft prompts to pre-training improve the generalization of prompt tuning

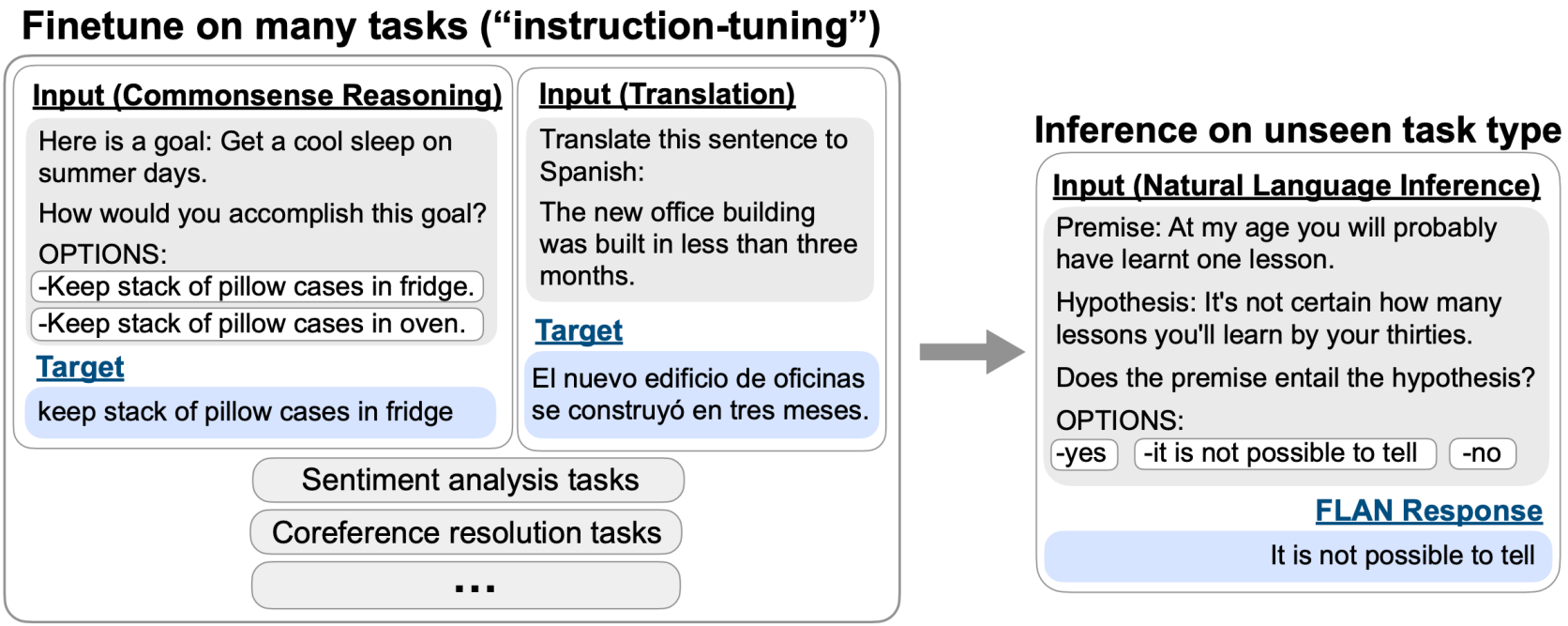


Comparisons of prompt tuning and pre-trained prompt tuning.

PPT: Pre-trained Prompt Tuning for Few-shot Learning, 2021.

Fine-tuning with Prompted Data

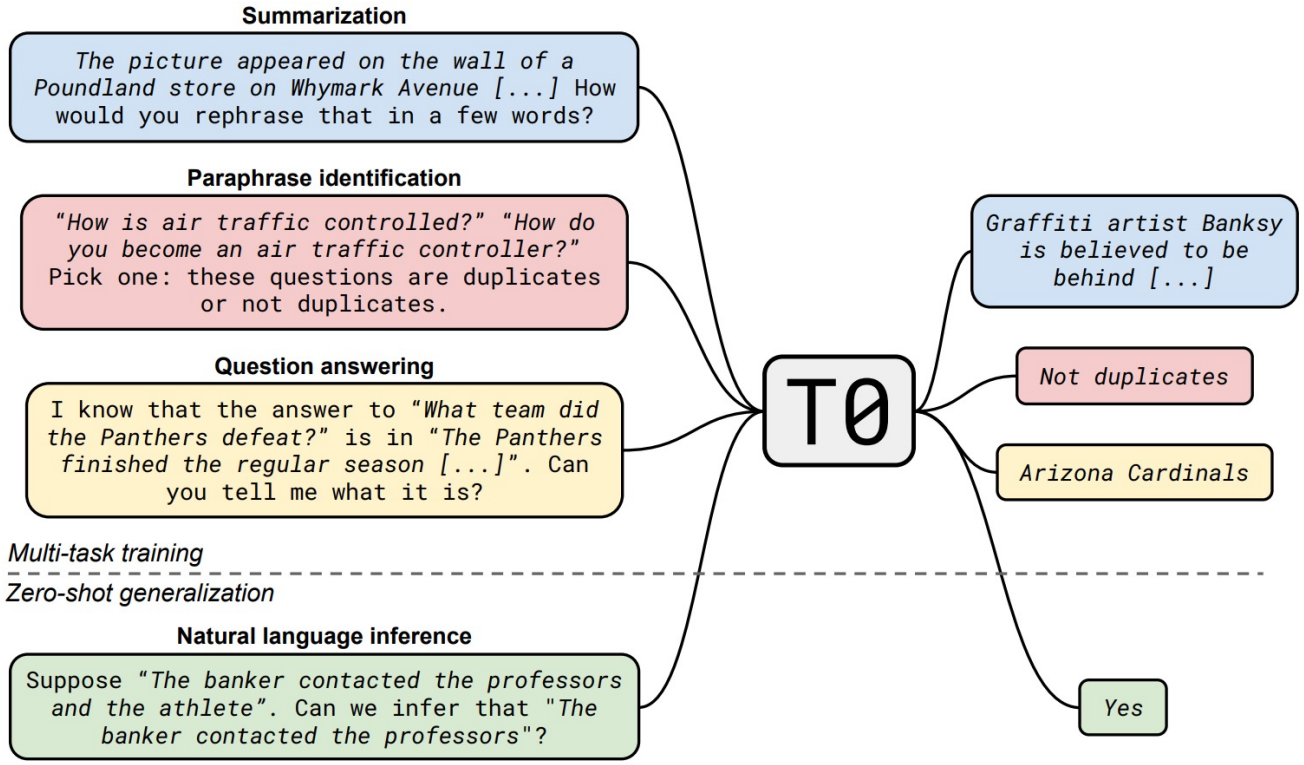
- Multi-task Pre-training with Hand-crafted Prompts
  - Fine-tuning a 130B PLM with prompts on 60 tasks
  - Substantially improve the zero-shot capability



FINETUNED LANGUAGE MODELS ARE ZERO-SHOT LEARNERS. 2021

Pre-training with Prompted Data

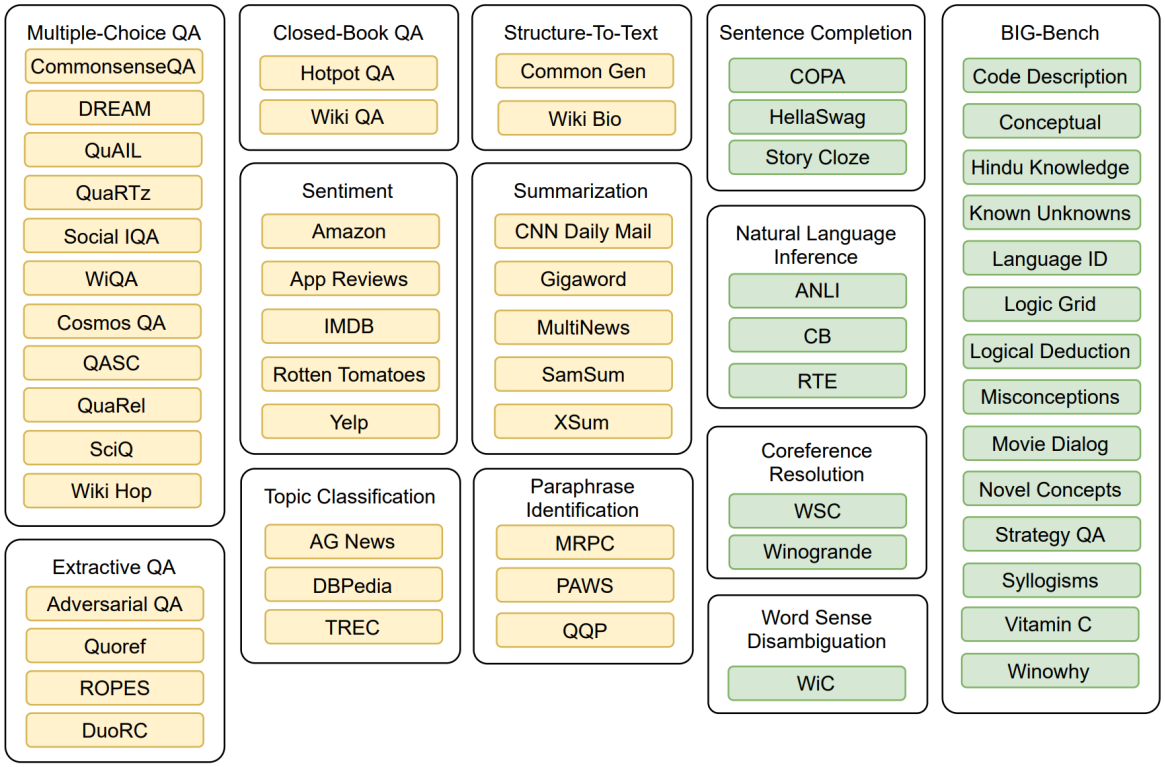
- Multi-task Pre-training with Hand-crafted Prompts
  - Use manually written prompts to train encoder-decoder model



Multitask Prompted Training Enables Zero-Shot Task Generalization. 2021.

Pre-training with Prompted Data

- Multi-task Pre-training with Hand-crafted Prompts
  - Use manually written prompts to train encoder-decoder model
  - Zero-shot generalization on unseen tasks



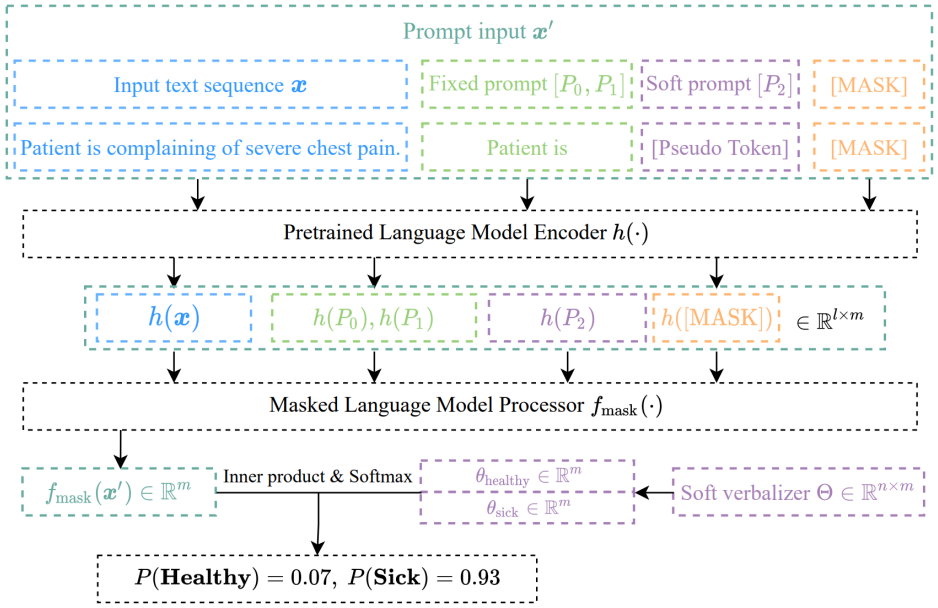
Multitask Prompted Training Enables Zero-Shot Task Generalization. 2021.

Applications

- Known applications
  - Most NLP tasks: NLU, Generation, Information Extraction, QA, translation …
  - Tasks with position-wise correlations may be difficult such as sequence tagging
- Can prompt-learning apply to Vision?
- Cross-Modality Application
- Biomedical, Clinical Application

Biomedical Prompt-learning

- Prompt-learning can support Clinical Decision
  - Big models in general domain (like GPT-3) can't perform well on specific domain like biomedical
  - Prompt-learning shows significantly effectiveness

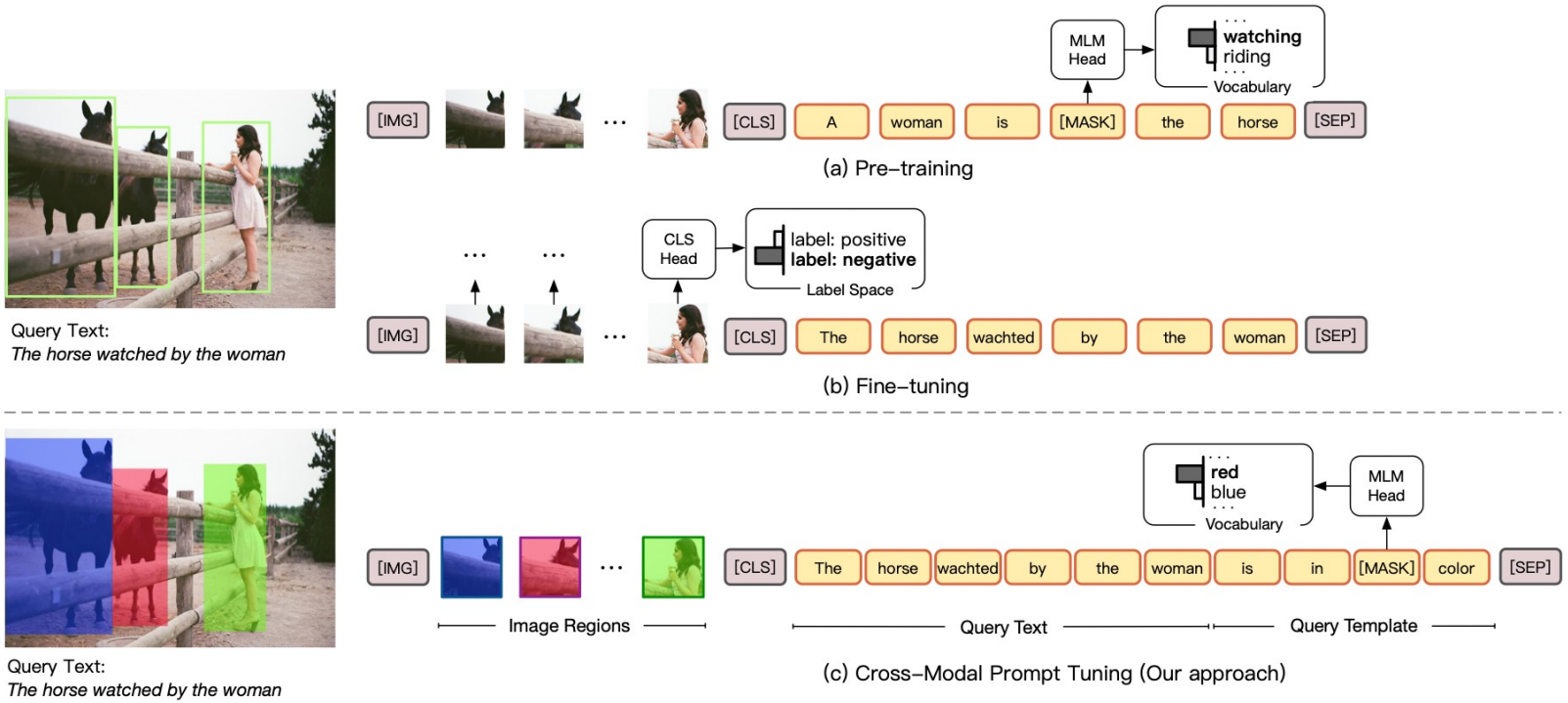


Paradigm	Balanced accuracy
Traditional fine-tuning	0.8162
Prompt learning	**0.8698**

Clinical Prompt Learning with Frozen Language Models. 2021

Cross-Modality Prompt-learning

- Cross-Modal Prompt-learning
  - Create colorful frames in images
  - Add color-wise textual prompts to input data



CPT: Colorful Prompt Tuning for Pre-trained Vision-Language Models. 2021

Summary

- Prompt-learning
  - A comprehensive framework that considers PLMs, downstream tasks, and human prior knowledge
  - The design of Template & Verbalizer is crucial
  - Prompt-learning has promising performance in low-data regime, and high variance with the select of templates
  - Prompt-learning has broad applications

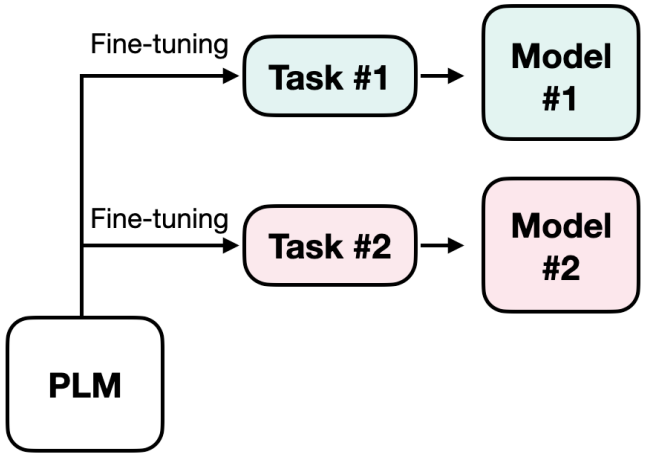
Delta Tuning

Ning Ding

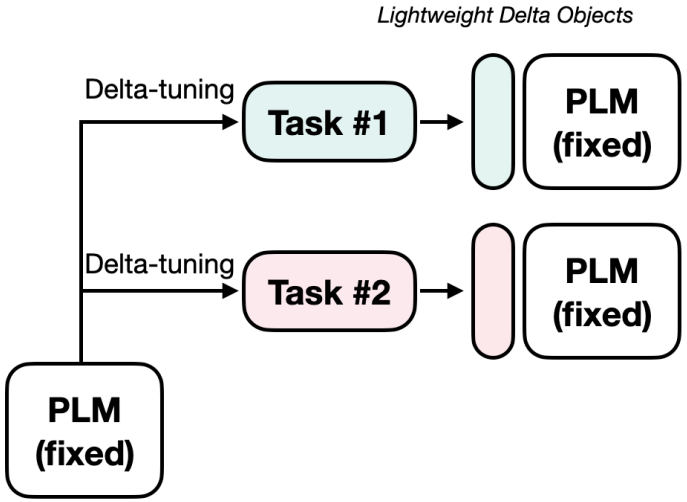
THUNLP

Delta Tuning

- How to Adapt Large-scale PLMs?
  - An Efficient Way — Delta Tuning
  - Only updating a small amount of parameters of PLMs
  - Keeping the parameters of the PLM fixed



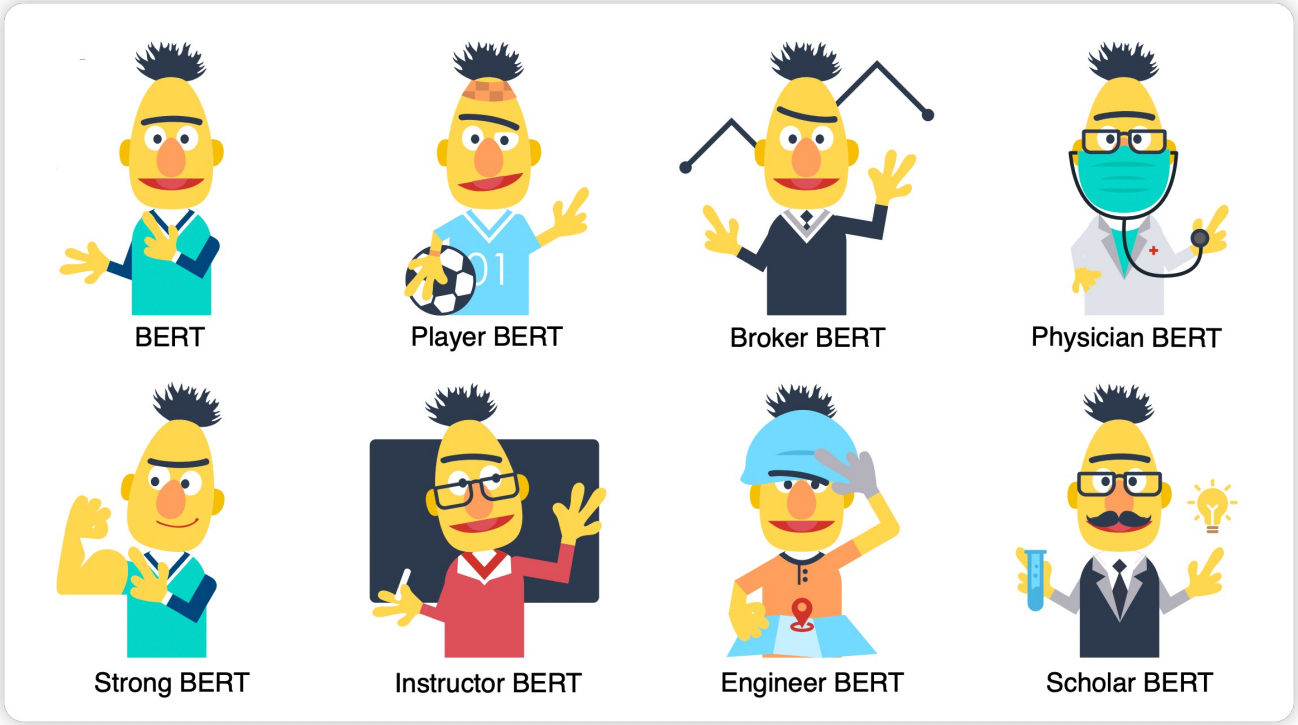
An illustration of fine-tuning.



An illustration of delta tuning.

Delta Tuning

- How to Adapt Large-scale PLMs?
  - An Efficient Way — Delta Tuning
  - Only updating a small amount of parameters of PLMs
  - Keeping the parameters of the PLM fixed

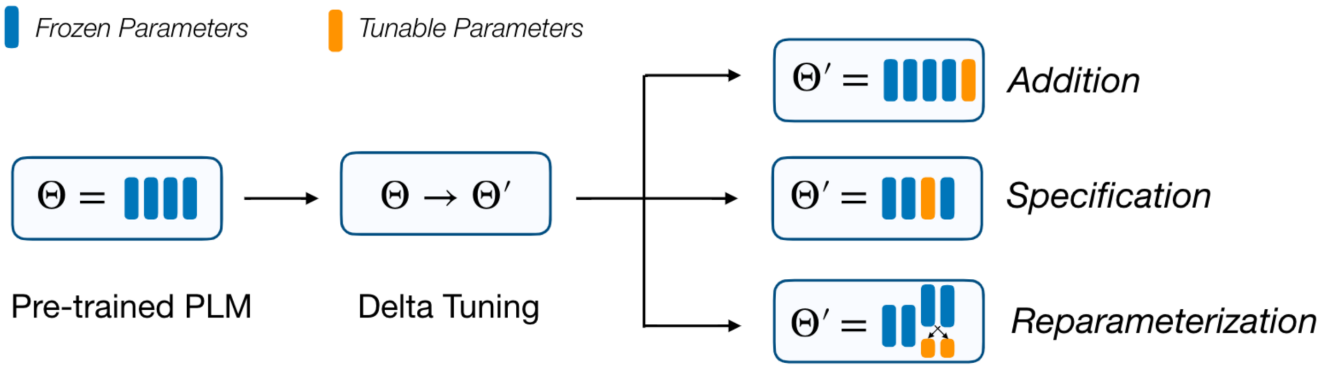


Why Parameter Efficient Work?

- In the Past Era
  - Parameter efficient learning can’t be realized in the past
  - Because all the parameters are randomly initialized
- With Pre-training
  - Pre-training can learn Universal Knowledge
  - Adaptation of downstream
  - Imposing universal knowledge to specific tasks

Delta Tuning

- Delta Tuning: Parameter Efficient Model Tuning
  - Addition-based methods introduce extra trainable neural modules or parameters that do not exist in the original model;
  - Specification-based methods specify certain parameters in the original model or process become trainable, while others frozen;
  - Reparameterization-based methods reparameterize existing parameters to a parameter-efficient form by transformation.



Categorization of delta tuning.

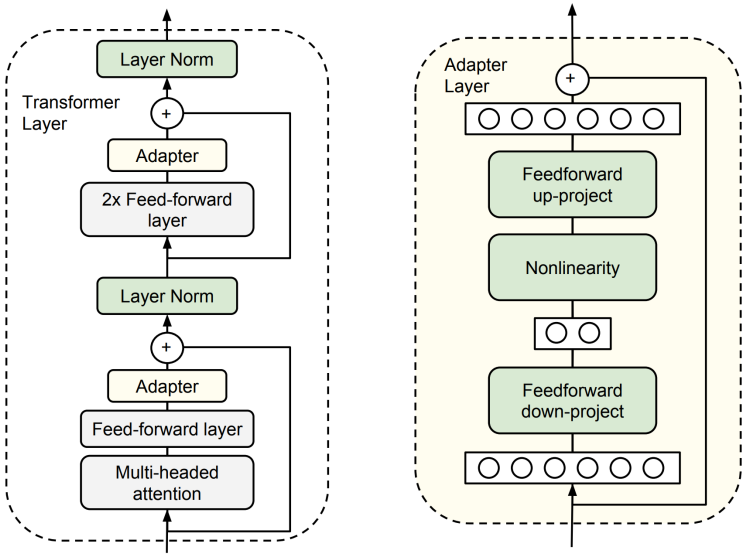
Delta Tuning: A Comprehensive Study of Parameter Efficient Methods for Pre-trained Language Models, 2022.

Addition-based: Adapters

- Adapter-Tuning
  - Injecting small neural modules (adapters) into Transformer Layer
  - Only fine-tuning adapters and keeping other parameters frozen
  - Adapters are down-projection and up-projection

$$\mathbf{h} \leftarrow f(\mathbf{h}\mathbf{W}_d)\mathbf{W}_u + \mathbf{h}.$$

  - Tunable parameters: 0.5%~8% of the whole model

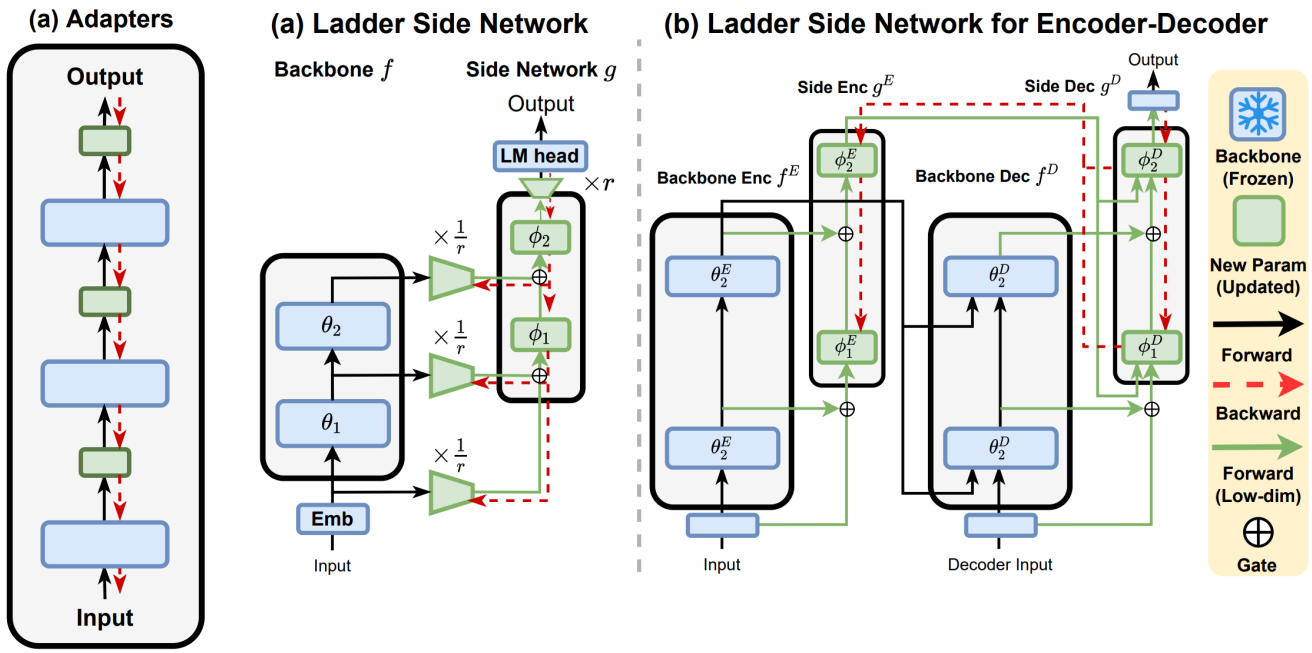


An illustration of Adapter-tuning.

Parameter-Efficient Transfer Learning for NLP, 2019.

Addition-based: Adapters

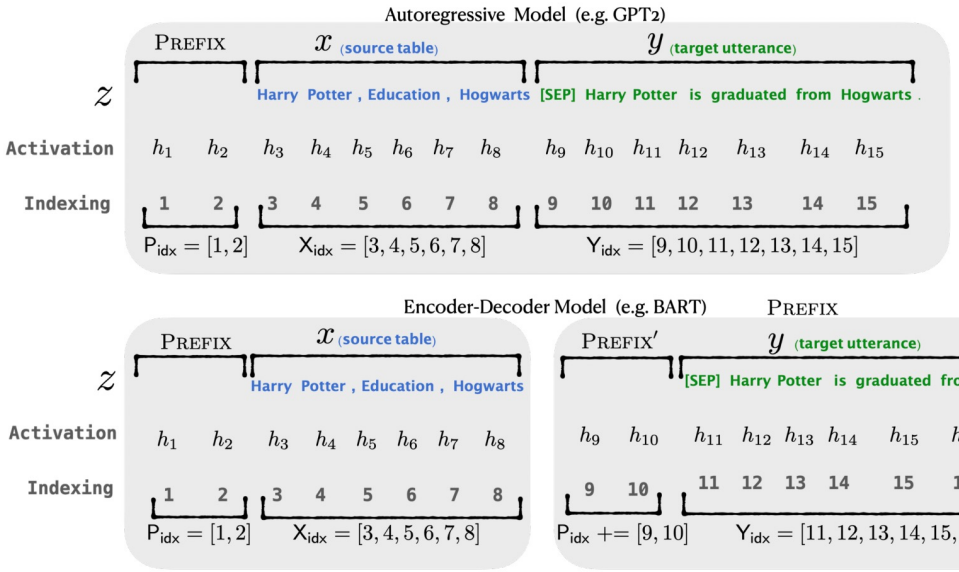
- Move the Adapter Out of the Backbone
  - Bridge a ladder outside the backbone model
  - Save computation of backpropagation
  - Save memory by shrinking the hidden size



LST: Ladder Side-Tuning for Parameter and Memory Efficient Transfer Learning, 2019.

Addition-based: Prompt-learning

- Prefix-Tuning
  - Inject prefixes (soft prompts) to each layer of the Transformer
  - Only optimizing the prefixes of the model



Summarization Example

```
Article: Scientists at University College London discovered people
tend to think that their hands are wider and their fingers are
shorter than they truly are.They say the confusion may lie in the
way the brain receives information from different parts of the
body.Distorted perception may dominate in some people, leading to
body image problems ... [ignoring 308 words] could be very
motivating for people with eating disorders to know that there was
a biological explanation for their experiences, rather than
feeling it was their fault."

Summary: The brain naturally distorts body image -
a finding which could explain eating disorders like
anorexia, say experts.
```

Table-to-text Example

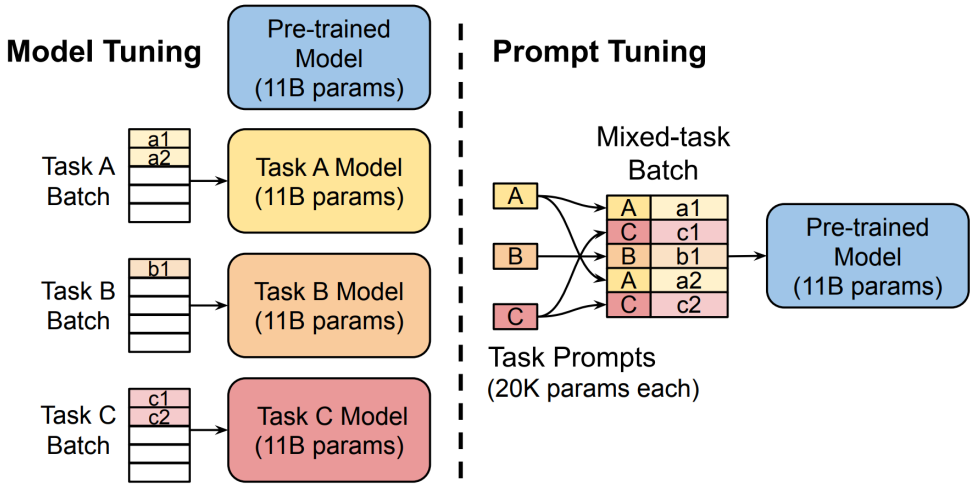
```
Table:  name[Clowns] customer-
rating[1 out of 5]  eatType[coffee
shop] food[Chinese] area[riverside]
near[Clare Hall]

Textual Description: Clowns is a
coffee shop in the riverside area
near Clare Hall that has a rating
1 out of 5 . They serve Chinese
food .
```

An illustration of prefix-tuning.

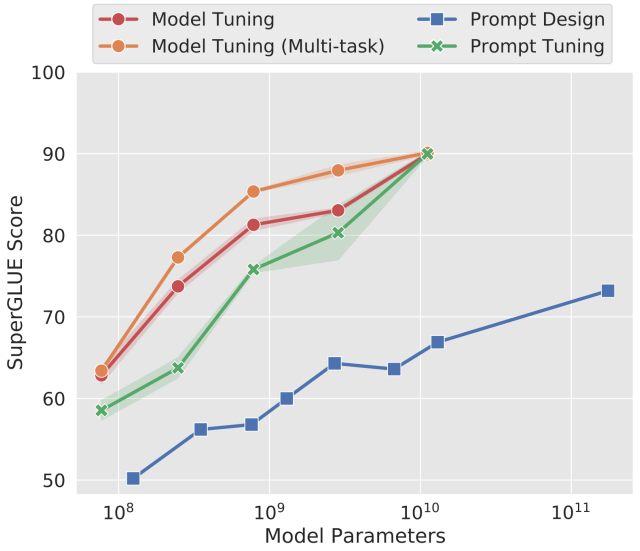
Prefix-Tuning: Optimizing Continuous Prompts for Generation, 2021.

Addition-based: Prompting

- Prompt-Tuning
  - Injecting soft prompts (embeddings) only to the input layer
  - Extraordinary power of scale
  - Comparable results to fine-tuning conditioned on 11B PLM



An illustration of prompt-tuning.



The power of scale of prompt-tuning.

The Power of Scale for Parameter-Efficient Prompt Tuning, 2021.

Specification-based: BitFit

- BitFit
  - A simple strategy: only updating the bias terms
  - Comparable performance of full fine tuning

$$\mathbf{Q}^{m,\ell}(\mathbf{x}) = \mathbf{W}_q^{m,\ell}\mathbf{x} + \mathbf{b}_q^{m,\ell}$$
$$\mathbf{K}^{m,\ell}(\mathbf{x}) = \mathbf{W}_k^{m,\ell}\mathbf{x} + \mathbf{b}_k^{m,\ell}$$
$$\mathbf{V}^{m,\ell}(\mathbf{x}) = \mathbf{W}_v^{m,\ell}\mathbf{x} + \mathbf{b}_v^{m,\ell}$$

		%Param	QNLI	SST-2	MNLI_m	MNLI_mm	CoLA	MRPC	STS-B	RTE	QQP
	Train size		105k	67k	393k	393k	8.5k	3.7k	7k	2.5k	364k
(V)	Full-FT†	100%	93.5	94.1	86.5	87.1	62.8	91.9	89.8	71.8	87.6
(V)	Full-FT	100%	91.5	93.4	85.5	85.8	60.1	90.1	89.9	71.7	87.5
(V)	Diff-Prune†	0.1%	92.7	93.3	85.6	85.9	58	87.4	86.3	68.6	85.2
(V)	BitFit	0.08%	91.8	93.3	84.6	84.8	63.4	91.5	90.3	75.1	85.6
(T)	Full-FT‡	100%	91.1	94.1	86.7	86.0	59.6	88.9	86.6	71.2	71.7
(T)	Full-FT†	100%	93.4	94.9	86.7	85.9	60.5	89.3	87.6	70.1	72.1
(T)	Adapters‡	3.6%	90.7	94.0	84.9	85.1	59.5	89.5	86.9	71.5	71.8
(T)	Diff-Prune†	0.5%	93.3	94.1	86.4	86.0	61.1	89.7	86.0	70.6	71.1
(T)	BitFit	0.08%	92.0	94.1	84.5	84.8	59.7	88.9	85.5	72.0	70.5

Performance on the GLUE benchmark validation set (V) and test set (T).

BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models, 2021.

Manipulate NLP in Low-dimension Space

- Intrinsic Prompt Tuning
  - The Model tuning is mapped into a low-dimensional subspace
  - 89% of the full-parameter fine-tuning performance could be achieved in as low tasks as 5-dimensional a subspace in 120 NLP tasks

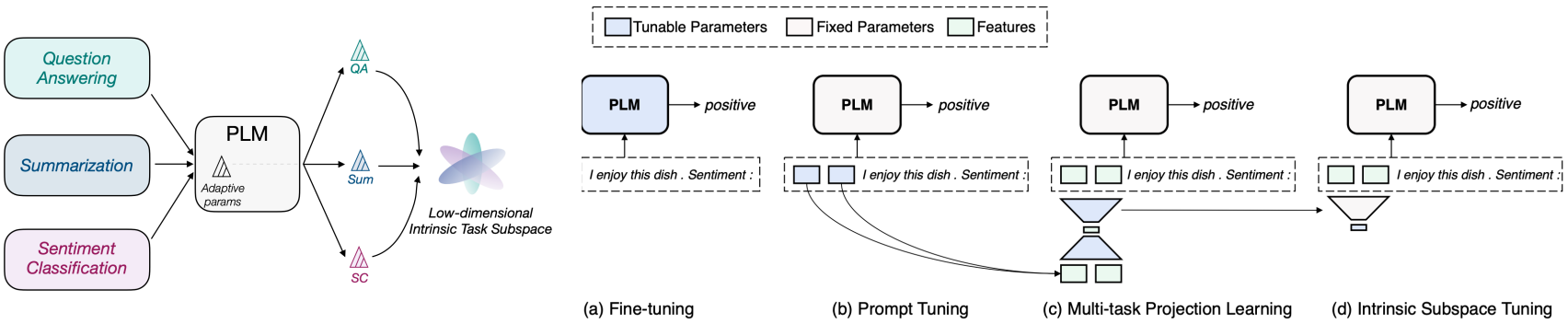
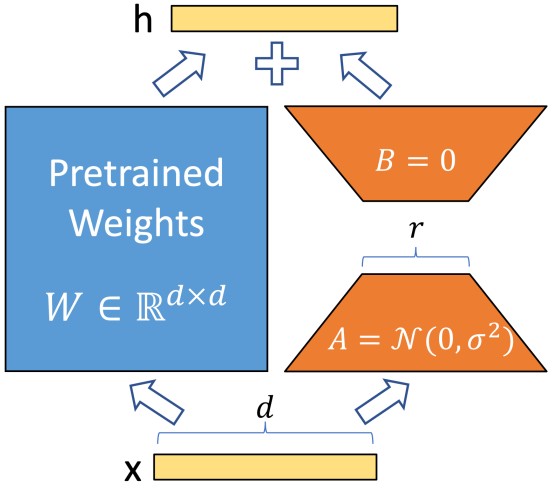


Illustration of intrinsic prompt tuning, a low-dimensional subspace is shared by multiple NLP tasks.

Exploring Low-dimensional Intrinsic Task Subspace via Prompt Tuning, 2021.

Manipulate NLP in Low-dimension Space

- LoRA: Low-Rank Adaptation
  - Freeze the model weights
  - Injects trainable rank-decomposition matrices to each Transformer layer
  - LoRA tunes 4.7 million paramters of the 175 billion parameters of the GPT-3 model

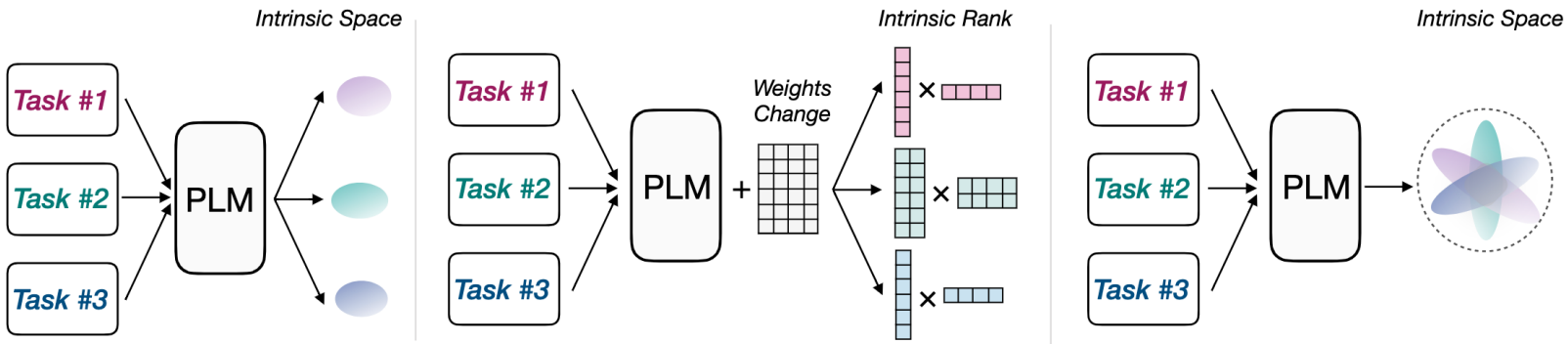


An illustration of the LoRA method.

LoRA: Low-Rank Adaptation of Large Langauge Models, 2021.

Connections

- The Reparameterization-based Methods Are Connected
  - Based on similar hypothesis
  - The optimization process could be transformed to a parameter efficient version

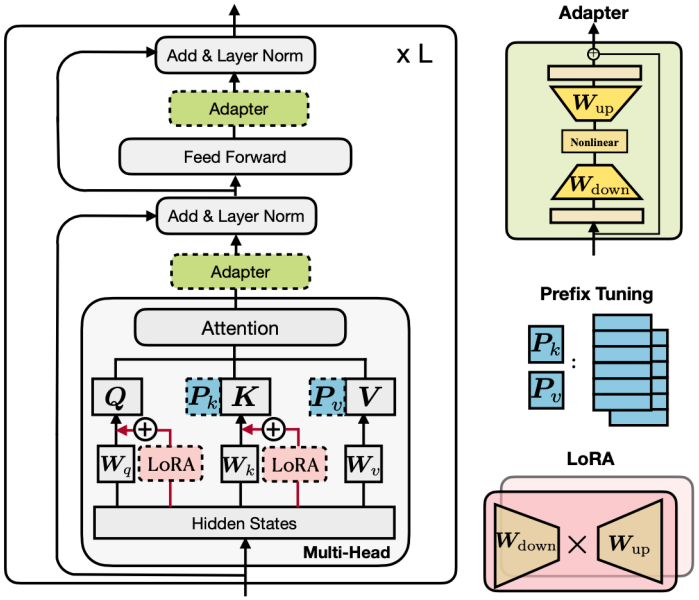


Conditioned on a PLM, [1] hypothesize that there exist a low-dimensional intrinsic subspace that could reparameterize one specific fine-tuning process (the left part). [2] hypothesize that the change of weights during adaptation has a low intrinsic rank (the middle part). And [3] hypothesize there may exist a common intrinsic space that could handle the fine-tuning for various NLP tasks (the right part).

[1] Intrinsic dimensionality explains the effectiveness of language model tuning, 2020.
[2] LoRA: Low-Rank Adaptation of Large Langauge Models, 2021.
[3] Exploring low-dimensional intrinsic task subspace via prompt tuning, 2021.

Connections

- A Unified View
  - Adapter, Prefix Tuning and LoRA could be connected
  - Function form
  - Insertion form
  - Modified Representation
  - Composition Function



Different delta tuning methods.

Method	Δh functional form	insertion form	modified representation	composition function
		Existing Methods		
Prefix Tuning	$\text{softmax}(xW_q P_k^\top)P_v$	parallel	head attn	$h \leftarrow (1-\lambda)h + \lambda\Delta h$
Adapter	$\text{ReLU}(hW_{\text{down}})W_{\text{up}}$	sequential	ffn/attn	$h \leftarrow h + \Delta h$
LoRA	$xW_{\text{down}}W_{\text{up}}$	parallel	attn key/val	$h \leftarrow h + s \cdot \Delta h$
		Proposed Variants		
Parallel adapter	$\text{ReLU}(hW_{\text{down}})W_{\text{up}}$	parallel	ffn/attn	$h \leftarrow h + \Delta h$
Muti-head parallel adapter	$\text{ReLU}(hW_{\text{down}})W_{\text{up}}$	parallel	head attn	$h \leftarrow h + \Delta h$
Scaled parallel adapter	$\text{ReLU}(hW_{\text{down}})W_{\text{up}}$	parallel	ffn/attn	$h \leftarrow h + s \cdot \Delta h$

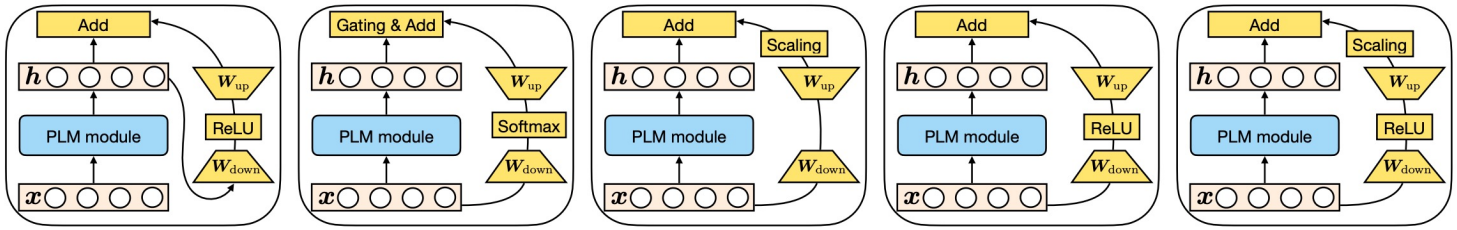
The framework to unify the three delta tuning methods.

Connections

- A Unified View
  - Adapter, Prefix Tuning, and LoRA could be connected in form
  - New variants could be derived under this framework

Method	Δh functional form	insertion form	modified representation	composition function
		Existing Methods		
Prefix Tuning	$\text{softmax}(xW_q P_k^\top)P_v$	parallel	head attn	$h \leftarrow (1-\lambda)h + \lambda\Delta h$
Adapter	$\text{ReLU}(hW_{\text{down}})W_{\text{up}}$	sequential	ffn/attn	$h \leftarrow h + \Delta h$
LoRA	$xW_{\text{down}}W_{\text{up}}$	parallel	attn key/val	$h \leftarrow h + s \cdot \Delta h$
		Proposed Variants		
Parallel adapter	$\text{ReLU}(hW_{\text{down}})W_{\text{up}}$	parallel	ffn/attn	$h \leftarrow h + \Delta h$
Muti-head parallel adapter	$\text{ReLU}(hW_{\text{down}})W_{\text{up}}$	parallel	head attn	$h \leftarrow h + \Delta h$
Scaled parallel adapter	$\text{ReLU}(hW_{\text{down}})W_{\text{up}}$	parallel	ffn/attn	$h \leftarrow h + s \cdot \Delta h$

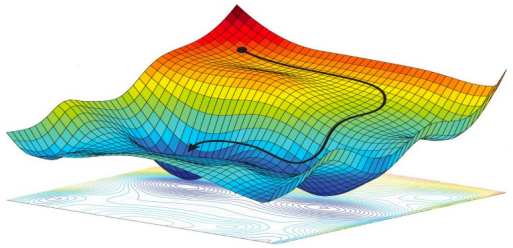
The framework to unify the three delta tuning methods.



(a) Adapter (b) Prefix Tuning (c) LoRA (d) Parallel Adapter (e) Scaled PA

Illutration of the derived variants of delta tuning.

Deep Analysis of Delta Tuning



- Theoretical Analysis
  - From optimization
  - Low-dimensional representation in solution space

$$\tilde{\mathcal{F}}(\theta, \delta_0) = \mathcal{F}(\theta), \quad \tilde{\mathcal{F}}(\theta_0, \delta) = \mathcal{F}(\psi(\delta)).$$

  - Low dimensional representation in functional space

$$|\mathcal{F}(\theta) - \hat{\mathcal{F}}(\delta)| < \epsilon,$$

  - From optimal control
  - Seek the optimal controller

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}_{tr}} \mathcal{L}(y|x;\theta) = \max_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}_{tr}} \log \left[W \left(h_o^{(L)} \right) \right]_y,$$

Delta Tuning: A Comprehensive Study of Parameter Efficient Methods for Pre-trained Language Models, 2022.

Deep Analysis of Delta Tuning

- A Rigorous Comparison of Performance
  - Experiments on 100+ NLP tasks
  - There is no way to gain an absolute advantage for delta tuning, fine-tuning is still the best model tuning method;

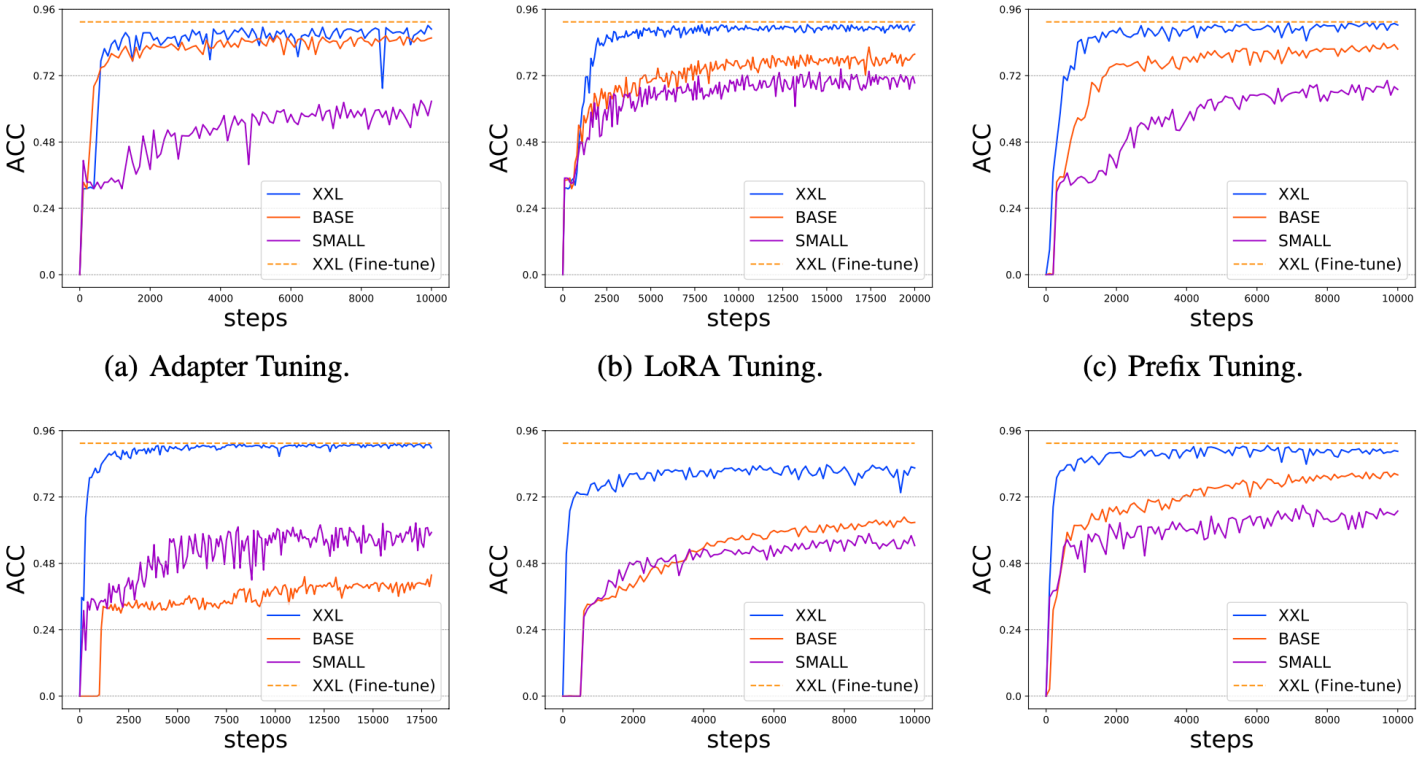
Task	PT (base)	PT (large)	PF	LR	AP	FT
Ratio of tunable parameters	0.03%	0.01%	7.93%	0.38%	2.38%	100%
ACRONYM_IDENTIFICATION	93.35	96.68	**96.12**	**96.12**	95.57	**96.12**
ADE_CORPUS_V2-CLASSIFICATION	41.76	94.42	93.25	**94.47**	93.91	94.27
ADE_CORPUS_V2-DOSAGE	78.57	89.29	82.14	**85.71**	82.14	82.14
ADE_CORPUS_V2-EFFECT	59.15	61.35	**63.25**	62.52	60.91	62.66
ADVERSARIAL_QA	34.10	54.60	43.17	46.40	45.35	**48.56**
AG_NEWS	91.37	93.61	93.42	94.63	94.60	**95.19**
ANLI	25.85	44.96	43.88	45.27	49.19	**50.54**
ASLG_PC12	15.78	44.07	47.71	73.72	80.65	**92.92**
BLIMP-ANAPHOR_GENDER_AGREEMENT	**100.00**	100.00	**100.00**	**100.00**	**100.00**	99.00
BLIMP-ANAPHOR_NUMBER_AGREEMENT	49.00	100.00	**100.00**	**100.00**	**100.00**	**100.00**
BLIMP-DETERMINER_NOUN_AGREEMENT	46.00	100.00	**100.00**	**100.00**	**100.00**	**100.00**
Average	48.81	65.92	64.07	66.06	65.58	**67.96**

Comparison over 100+ NLP tasks show that the fine-tuning method yields the best performance.

Delta Tuning: A Comprehensive Study of Parameter Efficient Methods for Pre-trained Language Models, 2022.

Deep Analysis of Delta Tuning

- Power of Scale
  - The power of scale is observed in all the methods, even random tuning



(a) Adapter Tuning. (b) LoRA Tuning. (c) Prefix Tuning.

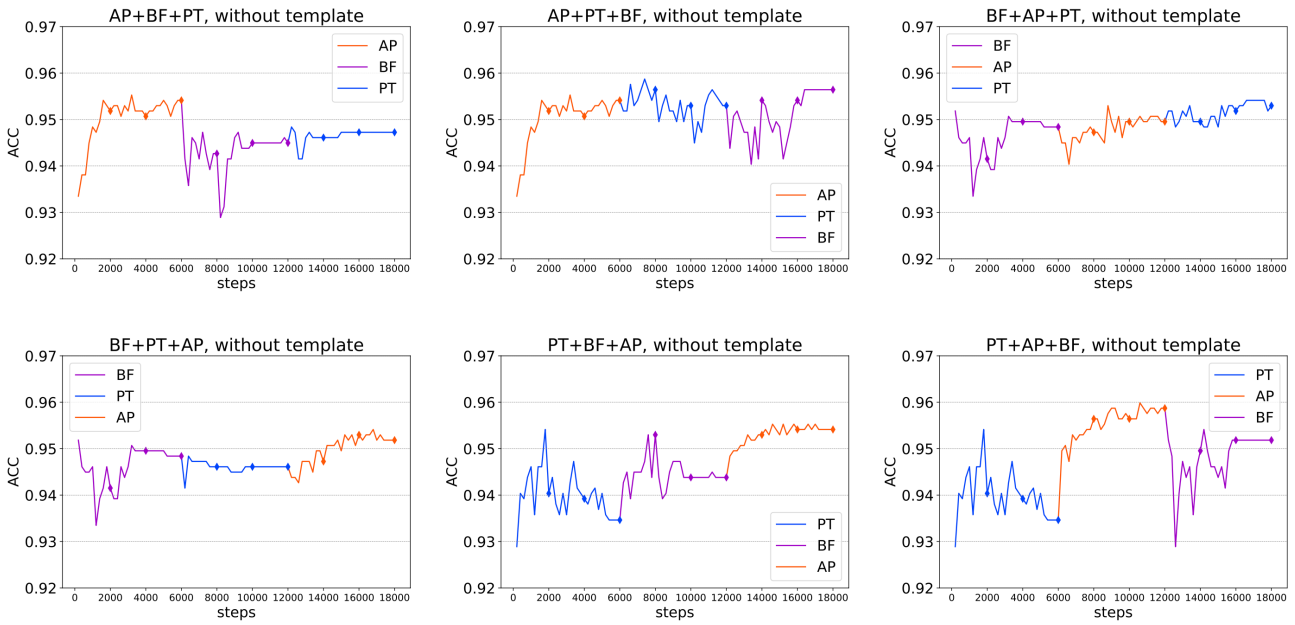
(d) Prompt Tuning. (e) Last Layer Tuning. (f) Selective Module Tuning.

Power of scale of different delta tuning methods.

Delta Tuning: A Comprehensive Study of Parameter Efficient Methods for Pre-trained Language Models, 2022.

Deep Analysis of Delta Tuning

- A Rigorous Comparison of Performance
  - Combination of different delta tuning methods
  - Implies the existence of Optimal Structure which is not defined manually

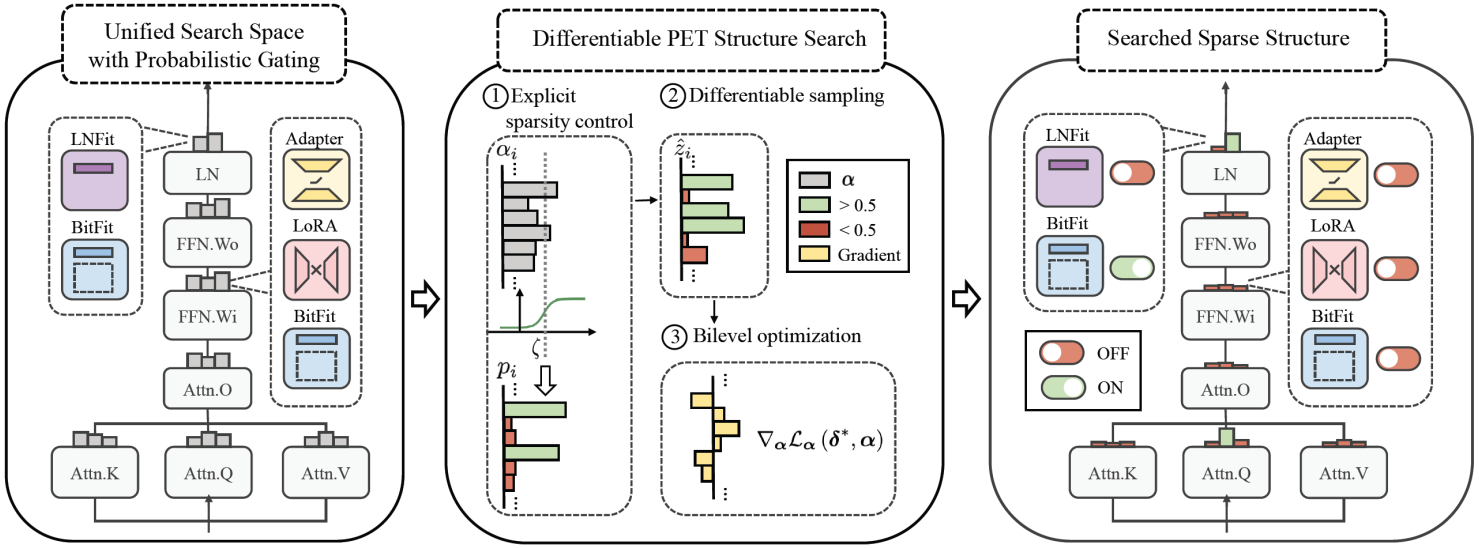


Conbination of different delta tuning methods.

Delta Tuning: A Comprehensive Study of Parameter Efficient Methods for Pre-trained Language Models, 2022.

Deep Analysis of Delta Tuning

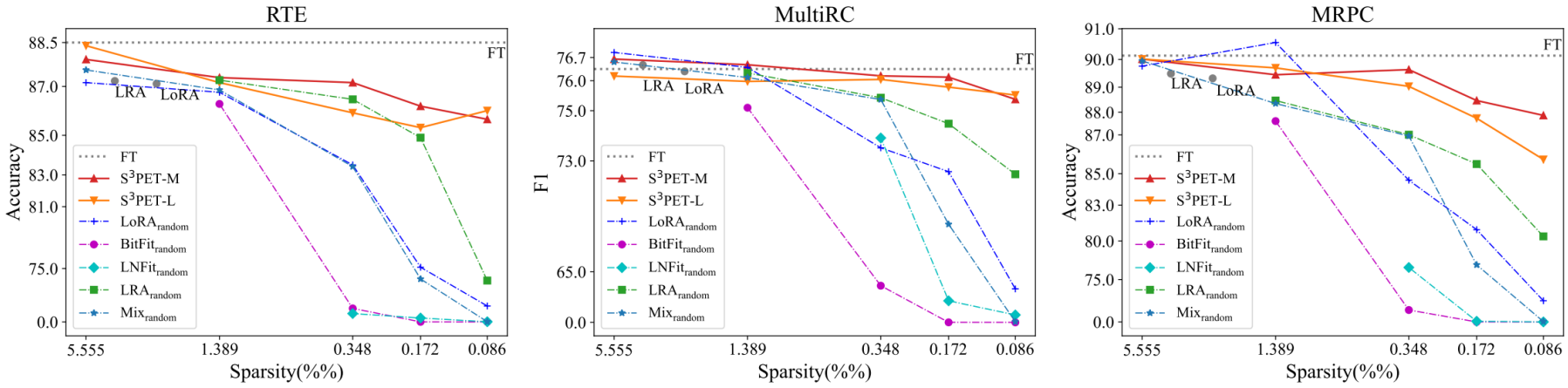
- A Rigorous Comparison of Performance
  - Implies the existence of Optimal Structure which is not defined manually
  - Automatically search the structure



Sparse Structure Search for Parameter-Efficient Tuning. 2022

Deep Analysis of Delta Tuning

- A Rigorous Comparison of Performance
  - Implies the existence of Optimal Structure which is not defined manually
    - Automatically search the structure
    - 1/10000 parameters could work

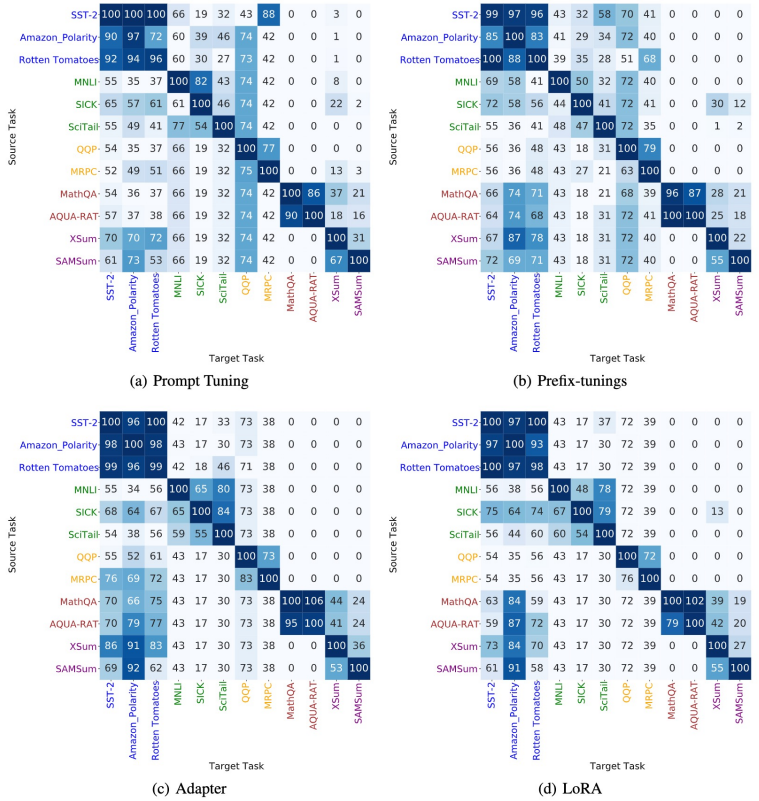


(a) RTE (b) MultiRC (c) MRPC

Sparse Structure Search for Parameter-Efficient Tuning. 2022

Deep Analysis of Delta Tuning

- Transferability
  - Delta Tuning shows non-trivial task-level transferability
  - Implies the possibility to construct a sharing platform

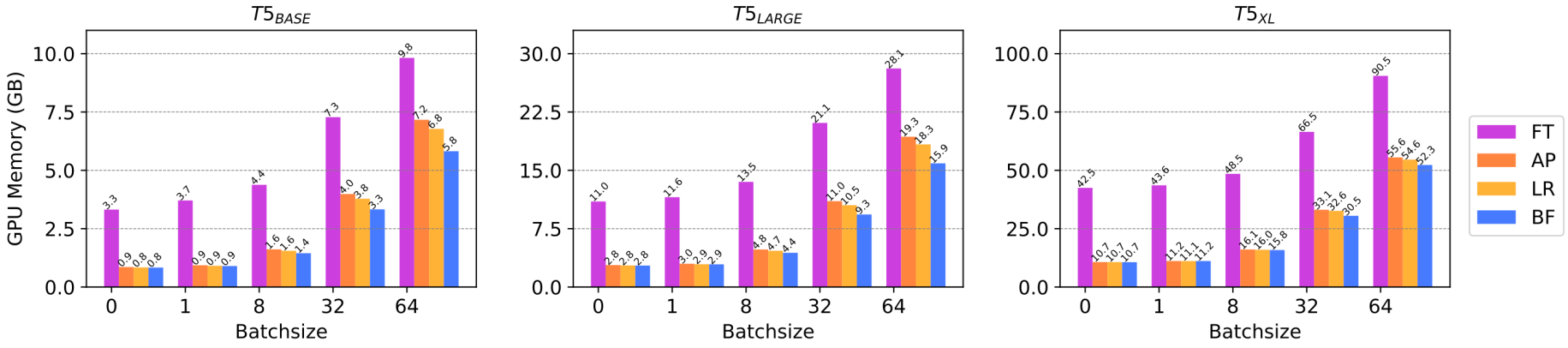


Transferability for different delta tuning methods on 10 NLP tasks.

Delta Tuning: A Comprehensive Study of Parameter Efficient Methods for Pre-trained Language Models, 2022.

Deep Analysis of Delta Tuning

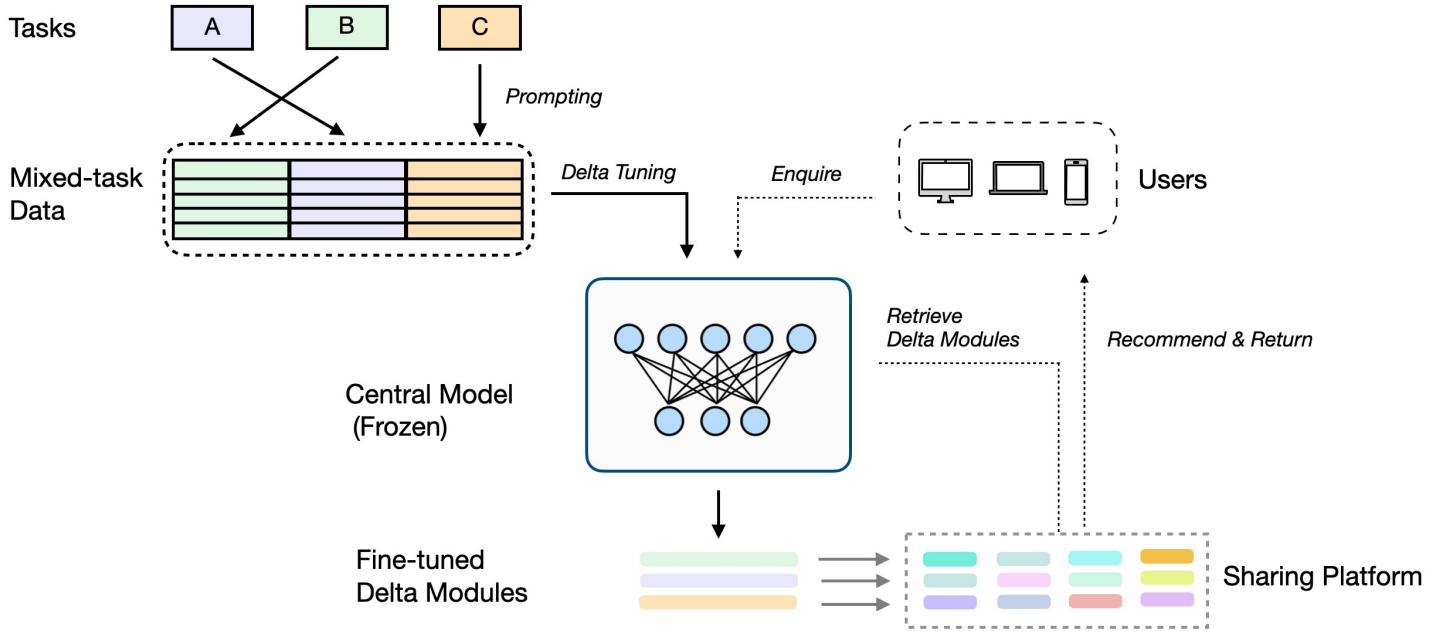
- **Efficient Tuning with low GPU RAM**
  - Tune T5-large on 11G single GPU (Nvidia 1080Ti, 2080, etc.)
  - Tune T5-3b on 24G single GPU （Nvidia 3090 and V100)
  - Tune T5-11b on 40G single GPU （Nvidia A100, with BMTrain)



Delta Tuning: A Comprehensive Study of Parameter Efficient Methods for Pre-trained Language Models, 2022.

Summary

- Delta tuning could effectively work on super-large models
  - Optimizing only a small portion of parameters could stimulate big models
- The structure may become less important as the model scaling
- What's NeXT?



Further Readings

- Paper List
  - PromptPapers: https://github.com/thunlp/PromptPapers
  - DeltaPapers: https://github.com/thunlp/DeltaPapers
- Programming Toolkit
  - OpemPrompt: https://github.com/thunlp/OpenPrompt
  - OpenDelta: https://github.com/thunlp/OpenDelta

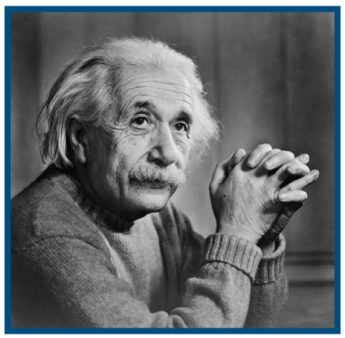
OpenPrompt

Shengding Hu

THUNLP

OpenPrompt

- A Unified Framework can to….



Albert Einstein was one of the greatest intellects in his time.

Sentiment Classification

Template: It was__
Verbalizer: *positive*→great…

Entity Typing

Template: Albert Einstein is a __
Verbalizer: *person/scholar*→scientist…

Knowledge Probing

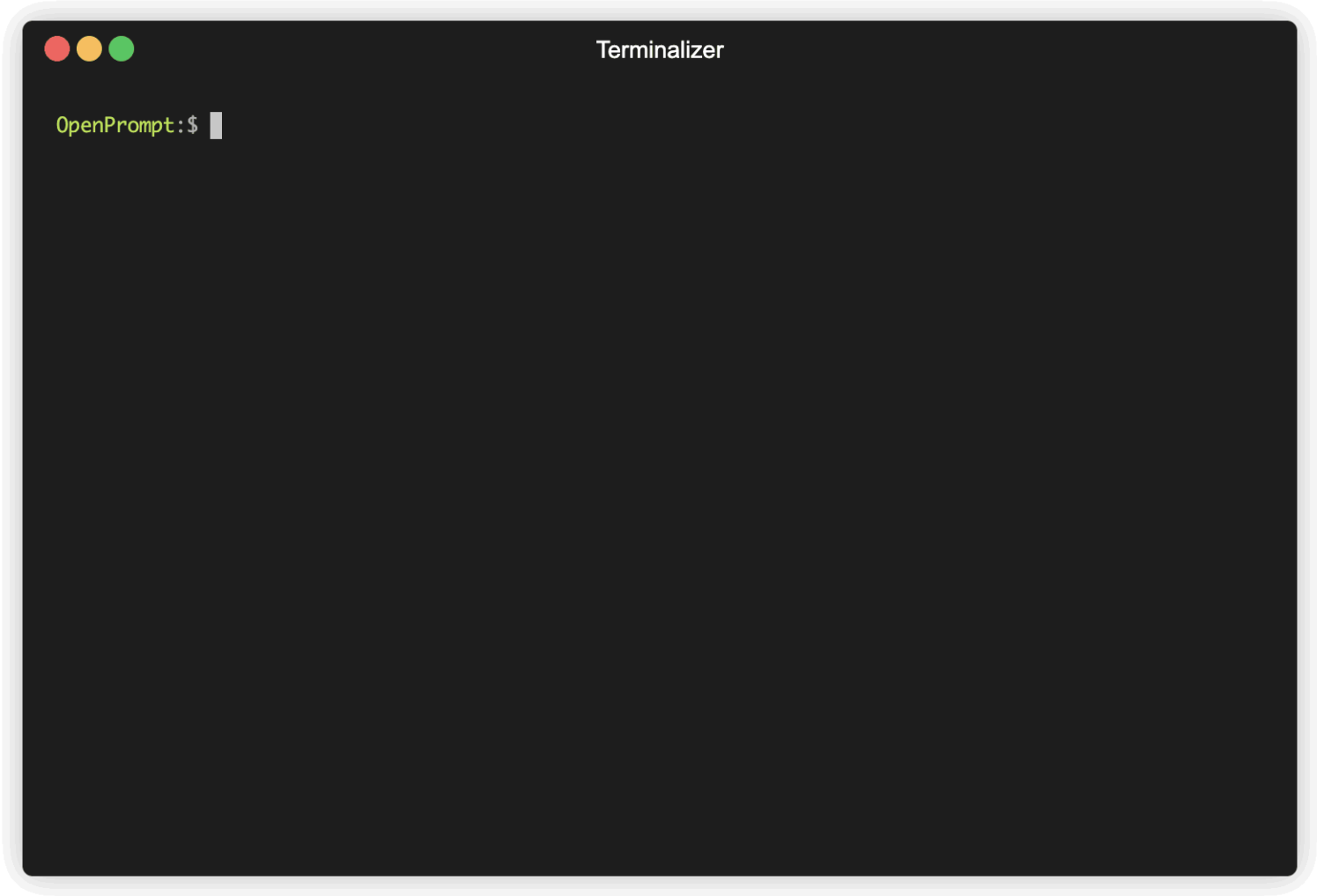
Template: Albert Einstein was born in__
Verbalizer: *None*

PLM

https://github.com/thunlp/OpenPrompt

OpenPrompt

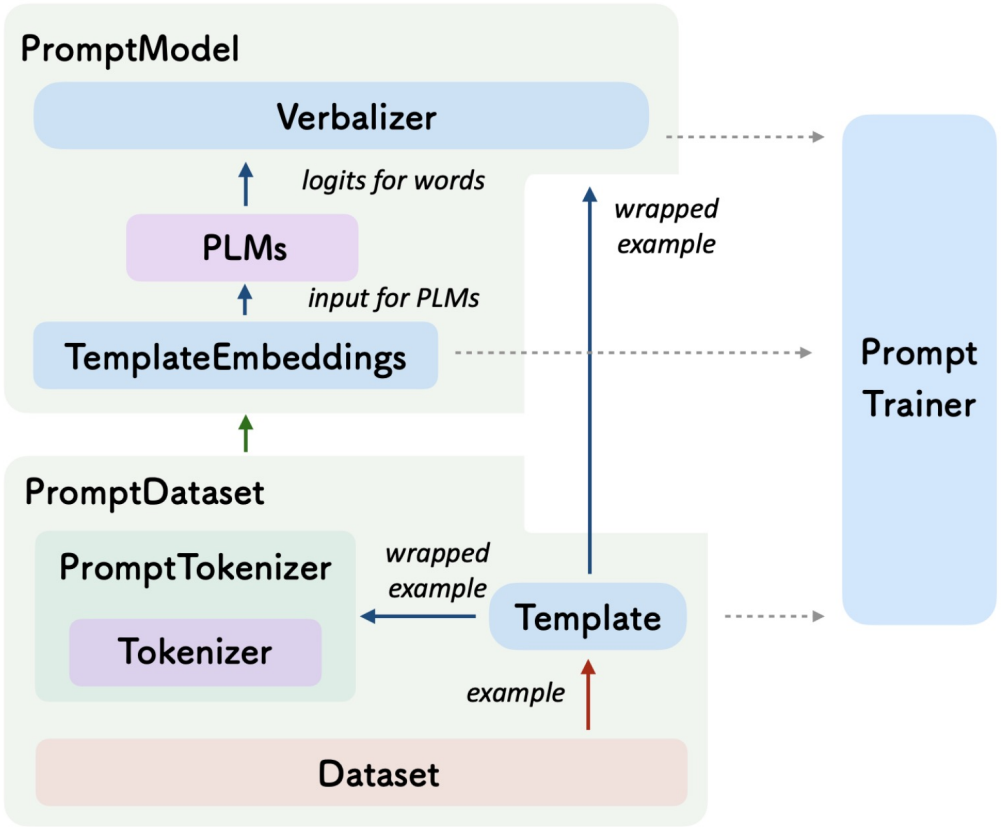
- A Unified Framework can to….



https://github.com/thunlp/OpenPrompt

OpenPrompt

- API design
  - Modularity, Flexibility, Uniformity



Wrapper Class: *These classes aim to make prompt-learning align with PyTorch pipeline, and users do not need to modify them.*

PLM-related Class: *These classes support the calling and management of various PLMs.*

Prompt-related Class: *These classes are unique modules for prompt-learning, and they can be implemented by users.*

Dataset-related Class: *These classes support the utilities for datasets across different NLP tasks.*

https://github.com/thunlp/OpenPrompt

OpenPrompt

- How to use OpenPrompt
  - Step 1: Define a task
    - Think about what's your data looks like and what do you want from the data!
  - Step 2: Obtain a PLM
    - Choose a PLM to support your task;
    - Different models have different attributes;
    - Essentially obtain a modeling strategy with pre-trained tasks;
    - Support 🤗 **Transformers** , more coming…

```
from openprompt.plms import get_model_class
model_class = get_model_class(plm_type = "bert")
model_path = "bert-base-cased"
bertConfig = model_class.config.from_pretrained(model_path)
bertTokenizer = model_class.tokenizer.from_pretrained(model_path)
bertModel = model_class.model.from_pretrained(model_path)
```


https://github.com/thunlp/OpenPrompt

OpenPrompt

- Step 3: Define a Template
  - A Template is a modifier of the original input text, which is also one of the most important modules in prompt-learning.

```
from openprompt.prompts import ManualTemplate
promptTemplate = ManualTemplate(
    text = ["<text_a>", "It", "was", "<mask>"],
    tokenizer = bertTokenizer,
)
```

- Step 4: Define a Verbalizer (optional)
  - A Verbalizer projects the original labels to a set of label words.

```
from openprompt.prompts import ManualVerbalizer
promptVerbalizer = ManualVerbalizer(
    classes = classes,
    label_words = {
        "negative": ["bad"],
        "positive": ["good", "wonderful", "great"],
    },
    tokenizer = bertTokenizer,
)
```


https://github.com/thunlp/OpenPrompt

OpenPrompt

- Step 5: Define a PromptModel
  - A PromptModel is responsible for training and inference
  - It defines the (complex) interactions of mentioned modules

```
from openprompt import PromptForClassification
promptModel = PromptForClassification(
    template = promptTemplate,
    model = bertModel,
    verbalizer = promptVerbalizer,
)
```

```
from openprompt import PromptDataLoader
data_loader = PromptDataLoader(
    dataset = dataset,
    tokenizer = bertTokenizer,
    template = promptTemplate,
)
```

OpenPrompt

- Step 6: Train and Inference
  - Train and evaluate the PromptModel in PyTorch fashion

PyTorch

```
promptModel.eval()
with torch.no_grad():
    for batch in data_loader:
        logits = promptModel(batch)
        preds = torch.argmax(logits, dim = -1)
        print(classes[preds])
# predictions would be 1, 0 for classes 'positive', 'negative'
```

OpenPrompt

Mixed Template

- Basic hard and soft template
- Incorporation of meta information

```
a {"mask"} news: {"meta": "title"} {"meta": "description"}
```

- Soft template initialized with textual tokens

```
{"meta": "premise"} {"meta": "hypothesis"} {"soft": "Does the first sentence entails
    the second ?"} {"mask"} {"soft"}.
```

- Post-processing

```
{"meta": "context", "post_processing": lambda s: s.rstrip(string.punctuation)}. {"
    soft": "It was"} {"mask"}
```

- Fast token duplication

```
{"soft": None, "duplicate": 100} {"meta": "text"} {"mask"}
```

OpenPrompt

Generation Verbalizer

- Label words defined as part of input
  - Similar fashion with mixed template

```
{"meta":"choice1"} {"placeholder", "text_a"} .
```

- Especially powerful in transforming ALL NLP tasks to generation tasks

OpenPrompt

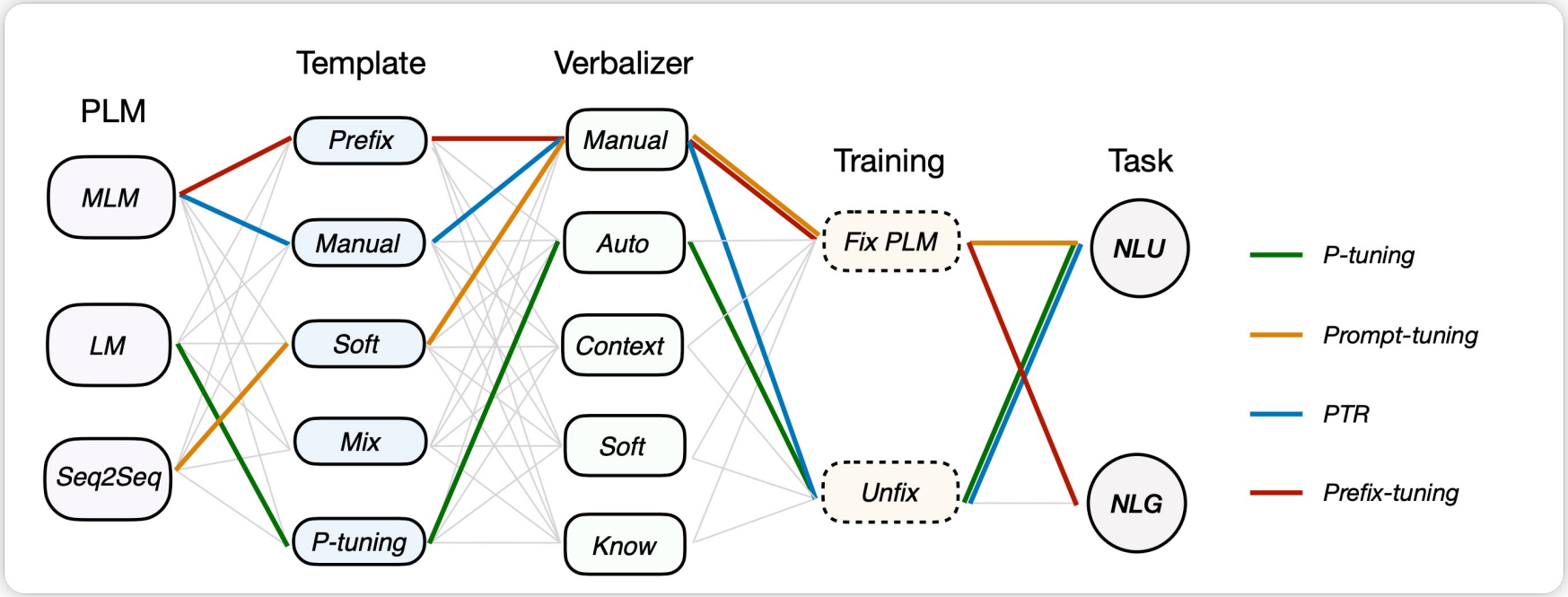
- Newly Designed Template Language - Mixed Template
  - Write Template in a flexible way

```
# Example A. Hard prompt for topic classification
a {"mask"} news: {"meta": "title"} {"meta": "description"}

# Example B. Hard prompt for entity typing
{"meta": "sentence"}. In this sentence, {"meta": "entity"} is a {"mask"},

# Example C. Soft prompt (initialized by textual tokens)
{"meta": "premise"} {"meta": "hypothesis"} {"soft": "Does the first sentence entails the second ?"} {"mask"} {"soft"}.

# Example D. The power of scale
{"soft": None, "duplicate": 100} {"meta": "text"} {"mask"}

# Example E. Post processing script support
# e.g. write an lambda expression to strip the final punctuation in data
{"meta": "context", "post_processing": lambda s: s.rstrip(string.punctuation)}. {"soft": "It was"} {"mask"}

# Example F. Mixed prompt with two shared soft tokens
{"meta": "premise"} {"meta": "hypothesis"} {"soft": "Does"} {"soft": "the", "soft_id": 1} first sentence entails {"soft_id": 1} second?

# Example G. Specify the title should not be truncated
a {"mask"} news: {"meta": "title", "shortenable": False} {"meta": "description"}
```


https://github.com/thunlp/OpenPrompt

Implementation Flow of OpenPrompt

- **Implement All Kinds of Prompt-Learning Pipelines**
  - Modify separate modules and create new methods
  - Apply existing methods to other scenarios



https://github.com/thunlp/OpenPrompt

OpenPrompt

- 1.7k stars for our Github repository
  - https://github.com/thunlp/OpenPrompt
- Along with 2.0k stars for referenced paper list
  - https://github.com/thunlp/PromptPapers

OpenPrompt

- DEMO
  - Integrating several tutorial into one colab
  - https://colab.research.google.com/drive/10syott1zXaQkjnlxOiSXKDFGy68SWR0y?usp=sharing

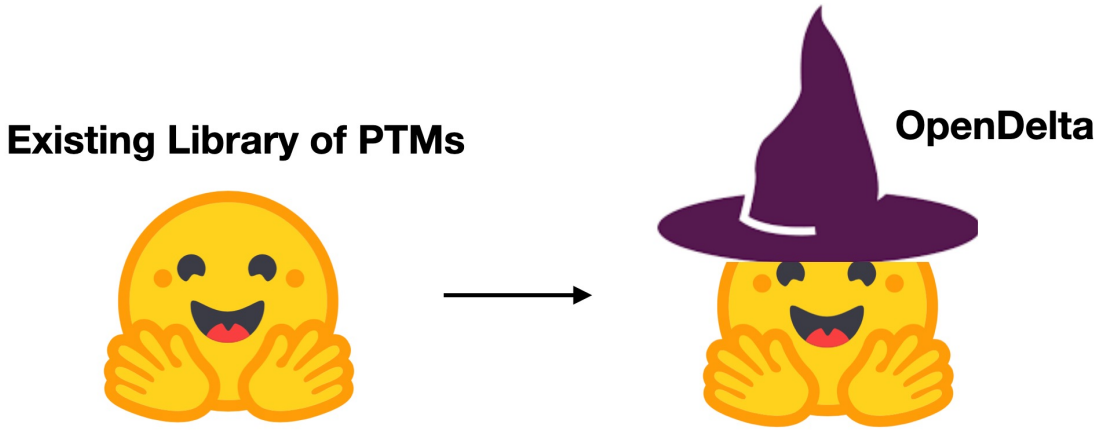
OpenDelta

Shengding Hu

THUNLP

OpenDelta

- OpenDelta: Toolkit for Delta Tuning
  - Clean: No need to edit the backbone PTM's codes.
  - Simple: Migrating from full-model tuning to delta-tuning needs as little as 3 lines of code.
  - Sustainable: Evolution in external libraries doesn't require update.
  - Extendable: Various PTMs can share the same delta-tuning codes.
  - Flexible: Able to apply delta-tuning to (almost) any position.



https://github.com/thunlp/OpenDelta

OpenDelta

- Apply OpenDelta to Various Models
  - Supported models

	Lora	Bias Tuning	Adapter Houstbly	Adapter Preffier	Adapter Drop	Adapater Low-Rank	Compactor	Prefix Tuning
T5	✅	✅	✅	✅	✅	✅	✅	✅
GPT-2	✅	✅	✅	✅	✅	✅	✅	✅
BART	✅	✅	✅	✅	✅	✅	✅	✅
DistilBERT	✅	✅	✅	✅	✅	✅	✅	✅
RoBERTa	✅	✅	✅	✅	✅	✅	✅	✅
BERT	✅	✅	✅	✅	✅	✅	✅	✅
T5-3b(parallel)	✅	✅	✅	✅	✅	✅	✅	✅
Deberta-v2	✅	✅	✅	✅	✅	✅	✅	
CTRL	✅	✅	✅	✅	✅	✅	✅	
ViT	✅							

PyTorch

Hugging Face

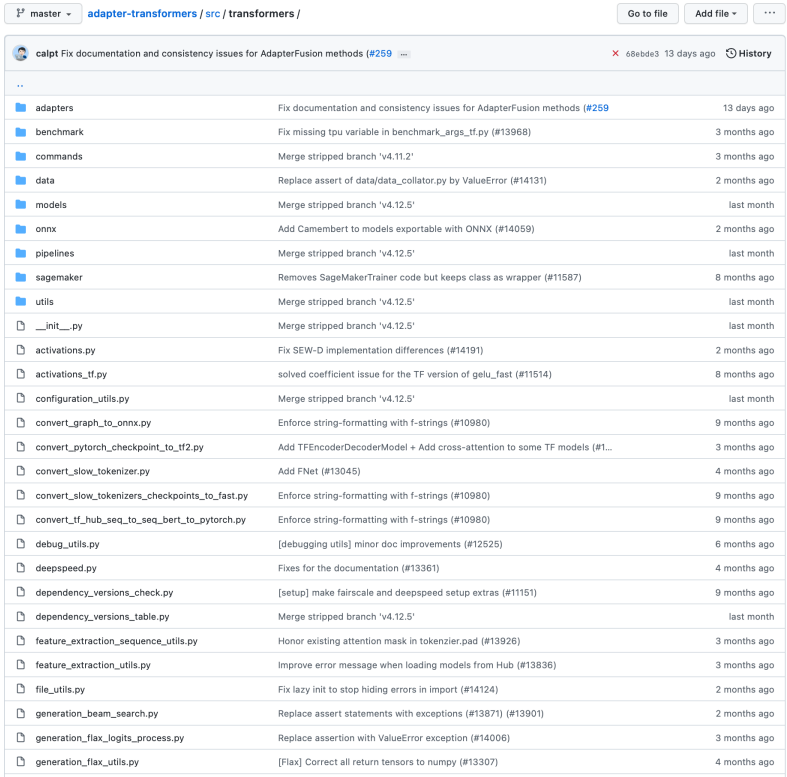
OpenPrompt

BMTrain

https://github.com/thunlp/OpenDelta

OpenDelta

- **Adapter Hub**
  - Need to modify the backbone code.
  - Need reimplementation for EVERY PTM.
  - Codes frozen at transformers version 4.12
  - Need constant update to suit Huggingface's update (to suit new feature)
  - Can only apply Adapter under existing mode (e.g. not supporting adding adapters to a fraction of layers or other places in the model)



OpenDelta

- Very simple interface:

```
delta_model = LoraModel(backbone_model, rank=7, modified_modules=[“attn.k”, “ffn”] )
or  delta_model = AdapterModel.from_finetuned(t5, “deltahub.com/adaptor_t5_mnli”)
or  delta_model = AutoDeltaModel.from_finetuned(bart, “deltahub.com/prefix_bart_mnli”)

delta_model.freeze_module(backbone_model, exclude=[“deltas”, “classifier”] )

## normal training pipeline ##

delta_model.save_finetuned(repo= …)
```

OpenDelta

- **How do we achieve it?**
  - **Key based addressing: Find the module according to the module/parameter key.**
  - **Three modification operations can cover most delta tuning:**
    - **Replace, Insert after, Insert before.**
  - **The modified model will have the same doc & I/O & address & Signature etc. to the original model.**
  - **Create pseudo data to automatically determine the parameter size of delta models.**

OpenDelta

- **How do we achieve it?**
  - **Alternating the flow of tensor.**
  - **Use a wrapper function to wrap the original forward function to let the tensor pass the delta models as well.**

```python
def _caller(_org_func, org_module, delta_name, *args, **kwargs):
    args = args[1:] # the first argument here is ``self``
    delta_module = getattr(org_module, delta_name)
    if hasattr(delta_module, "pre_forward"):# is not None:
        args, kwargs = delta_module.pre_forward(*args, **kwargs)
    # from IPython import embed
    # embed(header = "true")
    ret = _org_func(*args, **kwargs)
    if hasattr(delta_module, "post_forward"):# is not None:
        ret = delta_module.post_forward(ret)
    return ret
```

```python
new_forward = decorate(module.forward, _caller, extras=(module, _delta_info['delta_name']))
module.forward = new_forward.__get__(module, type(module))
```

OpenDelta

More than aggregating delta models …

1. **Visualize the parameters' location in the PTM.**

   Instead of printing the model using original PyTorch interface:

```
    )
  )
  (22): RobertaLayer(
    (attention): RobertaAttention(
      (self): RobertaSelfAttention(
        (query): Linear(in_features=1024, out_features=1024, bias=True)
        (key): Linear(in_features=1024, out_features=1024, bias=True)
        (value): Linear(in_features=1024, out_features=1024, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): RobertaSelfOutput(
        (dense): Linear(in_features=1024, out_features=1024, bias=True)
        (LayerNorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): RobertaIntermediate(
      (dense): Linear(in_features=1024, out_features=4096, bias=True)
    )
    (output): RobertaOutput(
      (dense): Linear(in_features=4096, out_features=1024, bias=True)
      (LayerNorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (23): RobertaLayer(
    (attention): RobertaAttention(
      (self): RobertaSelfAttention(
        (query): Linear(in_features=1024, out_features=1024, bias=True)
        (key): Linear(in_features=1024, out_features=1024, bias=True)
        (value): Linear(in_features=1024, out_features=1024, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): RobertaSelfOutput(
        (dense): Linear(in_features=1024, out_features=1024, bias=True)
        (LayerNorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): RobertaIntermediate(
      (dense): Linear(in_features=1024, out_features=4096, bias=True)
    )
    (output): RobertaOutput(
      (dense): Linear(in_features=4096, out_features=1024, bias=True)
```

1. To long to learn about the structure of PTMs

2. Which parameters are tunable?

OpenDelta

More than aggregating delta models …

1. Visualize the parameters' location in the PTM.

We visualize the location of all model's parameters in a neat way.

```
from opendelta.utils.visualization import Visualization

Visualization(model).structure_graph( )
```

```
prompt_model (PromptModel)
├── plm (RobertaForMaskedLM)
│   ├── roberta (RobertaModel)
│   │   ├── embeddings (RobertaEmbeddings)
│   │   │   ├── word_embeddings (Embedding) weight:[50265, 1024]
│   │   │   ├── position_embeddings (Embedding) weight:[514, 1024]
│   │   │   ├── token_type_embeddings (Embedding) weight:[1, 1024]
│   │   │   └── LayerNorm (LayerNorm) weight:[1024] bias:[1024]
│   │   └── encoder (RobertaEncoder)
│   │       └── layer (ModuleList)
│   │           └── 0-23(RobertaLayer)
│   │               ├── attention (RobertaAttention)
│   │               │   ├── self (RobertaSelfAttention)
│   │               │   │   └── query,key,value(Linear) weight:[1024, 1024] bias:[1024]
│   │               │   └── output (RobertaSelfOutput)
│   │               │       ├── dense (Linear) weight:[1024, 1024] bias:[1024]
│   │               │       └── LayerNorm (LayerNorm) weight:[1024] bias:[1024]
│   │               ├── intermediate (RobertaIntermediate)
│   │               │   └── dense (Linear) weight:[4096, 1024] bias:[4096]
│   │               └── output (RobertaOutput)
│   │                   ├── dense (Linear) weight:[1024, 4096] bias:[1024]
│   │                   └── LayerNorm (LayerNorm) weight:[1024] bias:[1024]
│   └── lm_head (RobertaLMHead) bias:[50265]
│       ├── dense (Linear) weight:[1024, 1024] bias:[1024]
│       ├── layer_norm (LayerNorm) weight:[1024] bias:[1024]
│       └── decoder (Linear) weight:[50265, 1024] bias:[50265]
└── template (SoftTemplate) soft_embeds:[20, 1024]
```

1. **Layers of the same structure are folded (In red)**

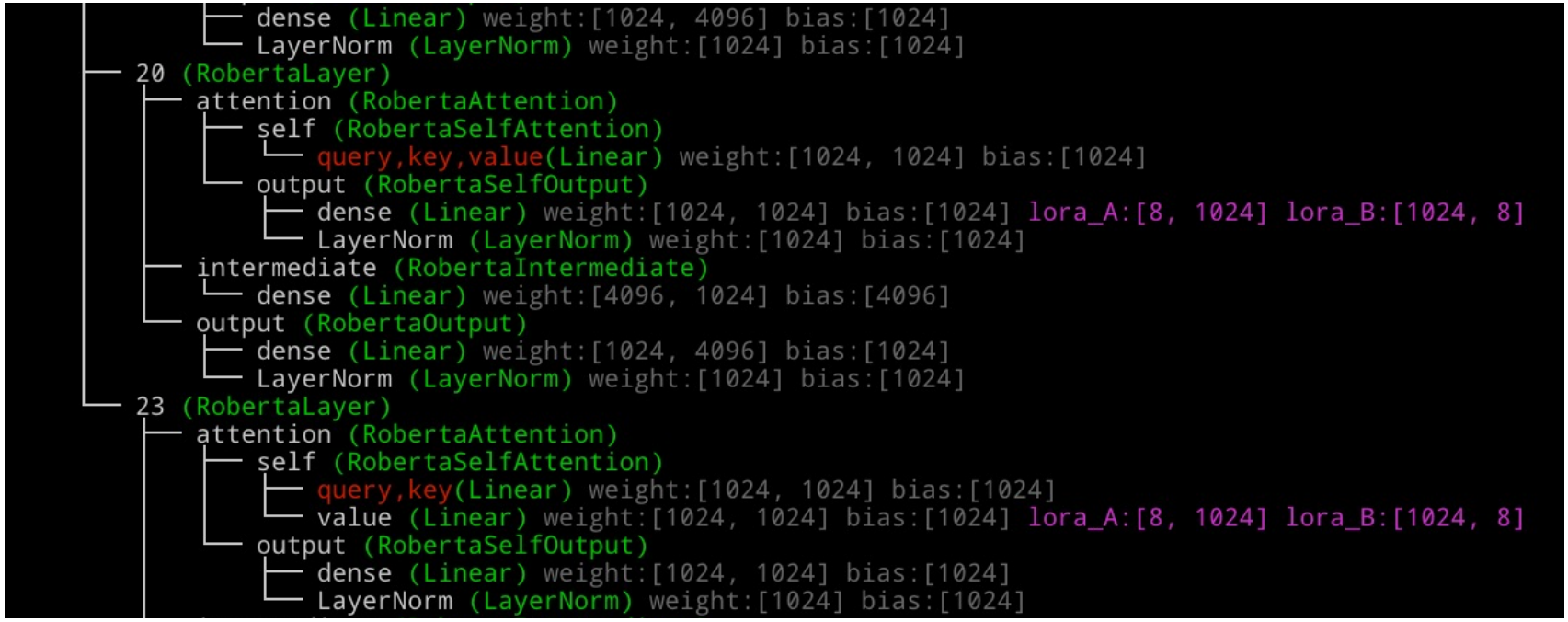
2. **All parameter informations are kept, blue ones are tunable, grey ones are frozen**

OpenDelta

More than aggregating delta models …

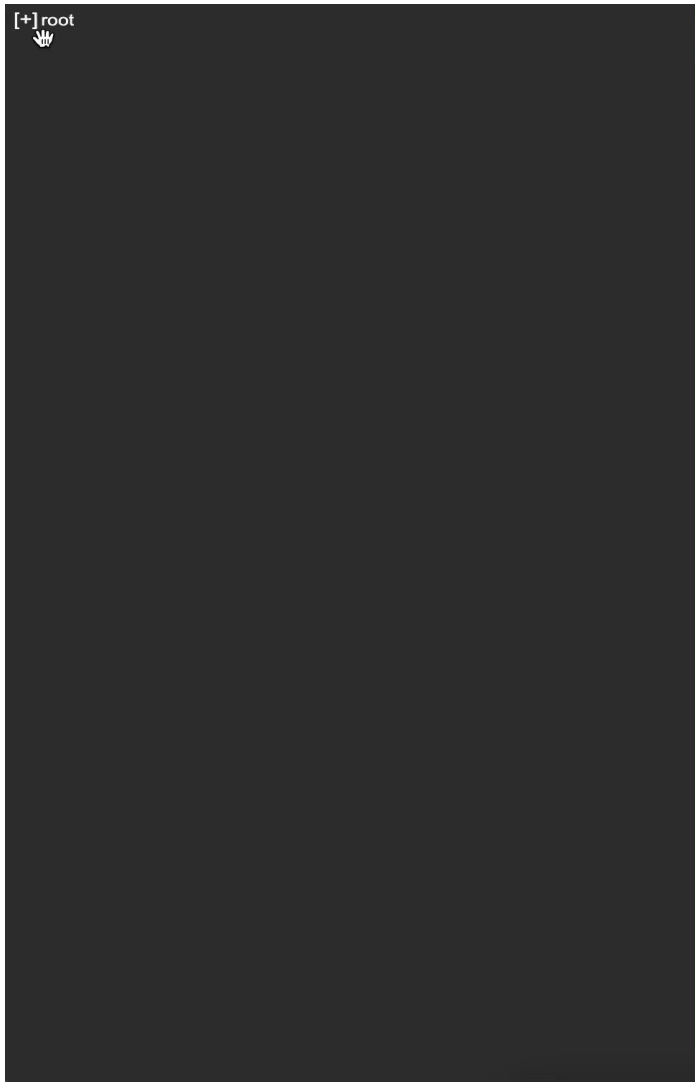
2. Insert delta modules in arbitrary layers.

```
delta_model = LoraModel(backbone_model, rank=8,
                        modified_keys=["20.attention.output.dense", "23.attention.self.value"])
```



OpenDelta

- Visualization
  - Visualize the DeltaModel in a webpage
  - Click Modules as Delta Object
  - Click Modules as Frozen Params
  - Submit to Generate Delta Tuning Code



https://github.com/thunlp/OpenDelta

OpenDelta

More than aggregating delta models …

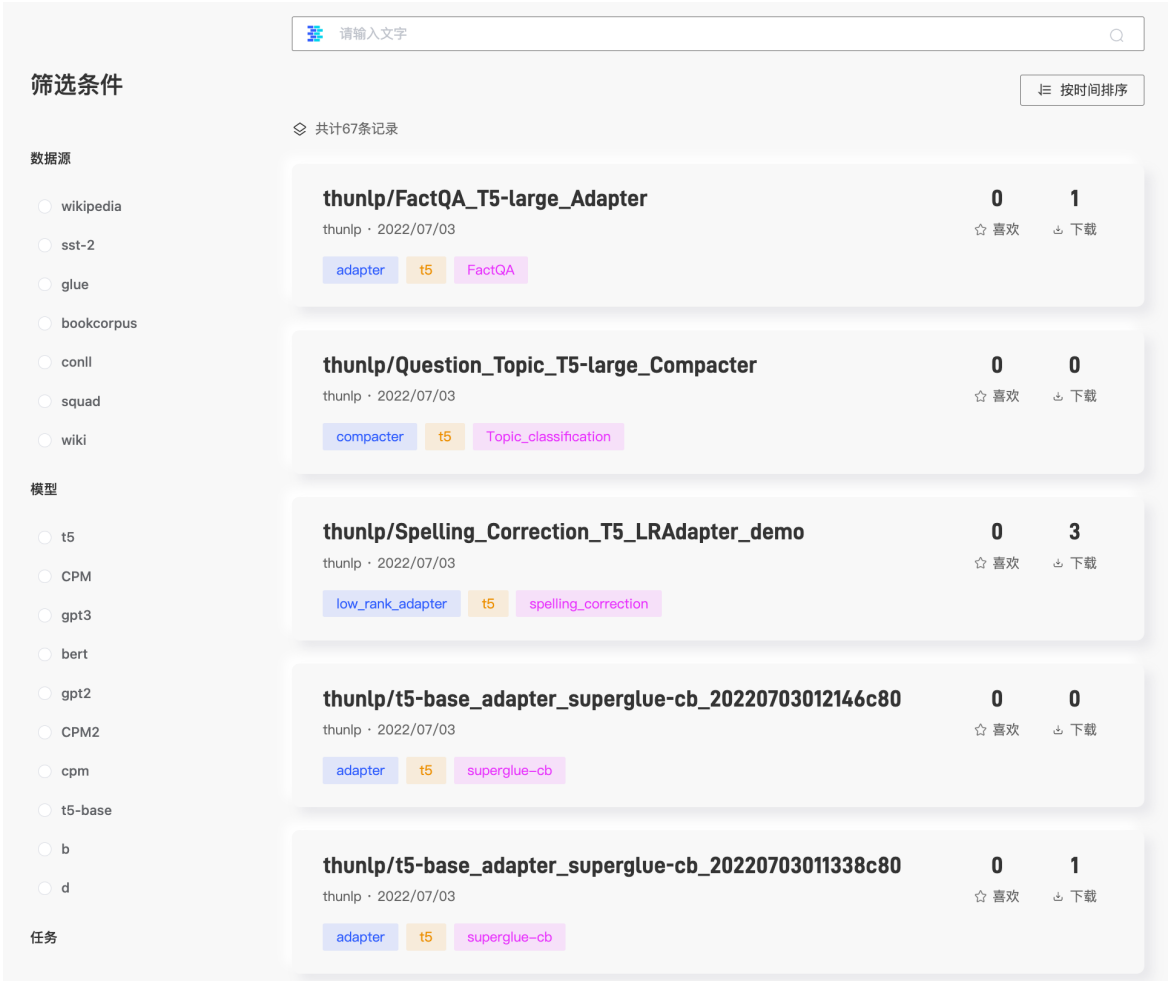
3. Delta center to save fine-tuned delta models

- Categorized by Delta Type，task， backbone…
- Automatically generate the hash for the backbone model.

```
delta_model.save_finetuned(
    finetuned_delta_path=delta_args.finetuned_delta_path,
    push_to_dc=training_args.push_to_dc,
    center_args={“test_performance”:all_results[‘test’][da
            ta_args.task_name]['test_average_metrics’],
                },
    center_args_pool = {**vars(model_args),
                    **vars(data_args),
                    **vars(training_args),
                    **vars(delta_args)},
    list_tags = ['NLI’],
    dict_tags = {'purpose':'for testing’},
    delay_push=delta_args.delay_push,
)
```

Delta Center

The platform to share and download various delta objects



Advanced Features of OpenDelta

- **AutoDelta Feature**
  - Automatically load and define delta moduels from **configuration**

```
delta_model = AutoDeltaModel.from_config(delta_config, backbone_model=backbone_model)
```

  - Automatically load and define delta moduels from **pre-trained**

```
delta_model = AutoDeltaModel.from_finetuned("DeltaHub/sst2-t5-base", backbone_model=backbone_model)
```

Advanced Features of OpenDelta

- **Multitask Serving**

```
from opendelta import AutoDeltaConfig, AutoDeltaModel

delta_model_spelling = AutoDeltaModel.from_finetuned("DeltaHub/Spelling_T5-lowrankadapter", backbone_model=model)
delta_model_spelling.detach()

delta_model_topic = AutoDeltaModel.from_finetuned("DeltaHub/QuestionTopic_T5-large_Compacter", backbone_model=model)
delta_model_topic.detach()

delta_model_fact = AutoDeltaModel.from_finetuned("DeltaHub/FactQA_T5-large_Adapter", backbone_model=model)
delta_model_fact.detach()
```

Advanced Features of OpenDelta

- **Multitask Serving**

```
def multitask_serving(input_text):
    input_ids = tokenizer(input_text, return_tensors="pt").input_ids#.cuda()
    delta_model_spelling.attach()
    answers_ids =model.generate(input_ids=input_ids, max_length=20, num_beams=4)
    input_text = tokenizer.decode(answers_ids[0], skip_special_tokens=True)
    print("Correct Spelling: {}".format(input_text))
    delta_model_spelling.detach()

    delta_model_topic.attach()
    input_ids = tokenizer(input_text, return_tensors="pt").input_ids#.cuda()
    answers_ids =model.generate(input_ids=input_ids, max_length=20, num_beams=4)
    topic = tokenizer.decode(answers_ids[0], skip_special_tokens=True)
    delta_model_topic.detach()
    print("Question Topic: {}".format(topic))

    delta_model_fact.attach()
    input_ids = tokenizer(input_text, return_tensors="pt").input_ids#.cuda()
    answers_ids =model.generate(input_ids=input_ids, max_length=20, num_beams=4)
    input_text = tokenizer.decode(answers_ids[0], skip_special_tokens=True)
    delta_model_fact.detach()
    print("Question Answer: {}".format(input_text))
```

Advanced Features of OpenDelta

- **Multitask Serving**
  - **(Runtime Polymorphism ☺)**

```
multitask_serving("When was Beiiing olymp#ic heldd ?")
multitask_serving("What the commmon career of Newton and Eintesin?")
```

```
Correct Spelling: When was Beijing Olympic held?
Question Topic: The question's topic is sports.
Question Answer: 2008
Correct Spelling: What was the common career of Newton and Einstein?
Question Topic: The question's topic is science.
Question Answer: Physicists
```

https://colab.research.google.com/drive/1uAhgAdc8Qr42UKYDlgUv0f7W1-gAFwGo?usp=sharing

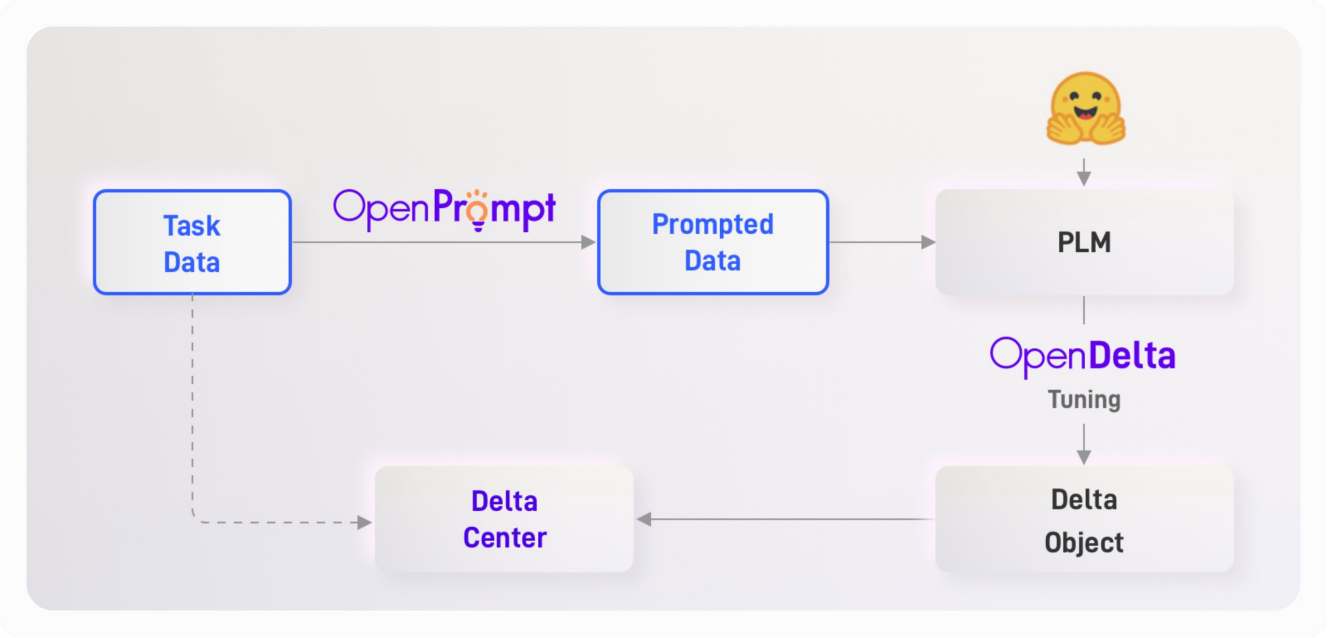
Advanced Features of OpenDelta

- **DEMO**

https://colab.research.google.com/drive/1uAhgAdc8Qr42UKYDlgUv0f7W1-gAFwGo?usp=sharing

OpenDelta & OpenPrompt

- **Collaboration of OpenDelta & OpenPrompt**
  - OpenDelta is a toolkit for Delta Tuning
  - Collaborated with OpenDelta, there is a loop to efficiently stimulate LMs



Thanks

THUNLP