TextBrewer: An Open-Source Knowledge Distillation Toolkit for Natural Language Processing

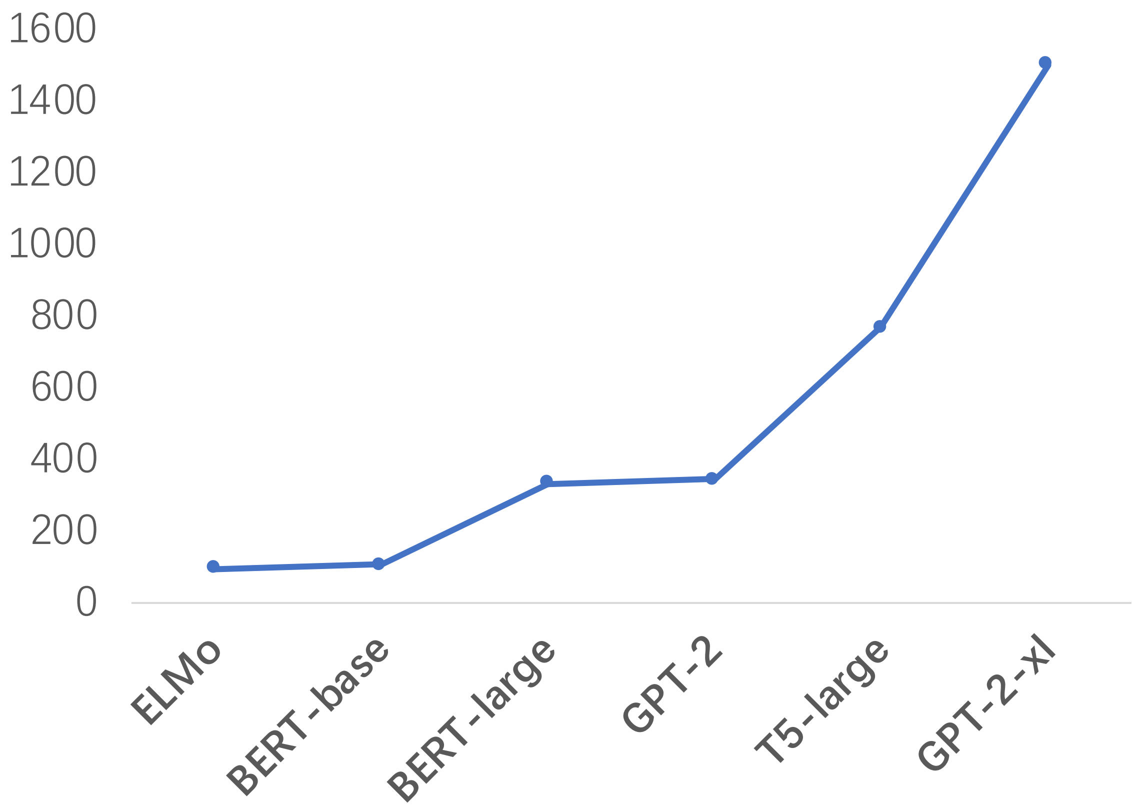
Ziqing Yang, Yiming Cui, Zhipeng Chen, Wanxiang Che, Ting Liu, Shijin Wang, Guoping Hu

ACL 2020

State Key Laboratory of Cognitive Intelligence, iFLYTEK Research,
Research Center for SCIR, Harbin Institute of Technology,
iFLYTEK AI Research (Hebei)

Pretrained Models in NLP

- Size of large pretrained models
  - **ELMo** – 94M parameters
  - **BERT$_{\text{base}}$** – 108M parameters
  - **BERT$_{\text{large}}$** – 334M parameters
  - **GPT2** – 1.5B parameters
  - **T5-11B** – 11B parameters
  - **GPT3** – 170B parameters



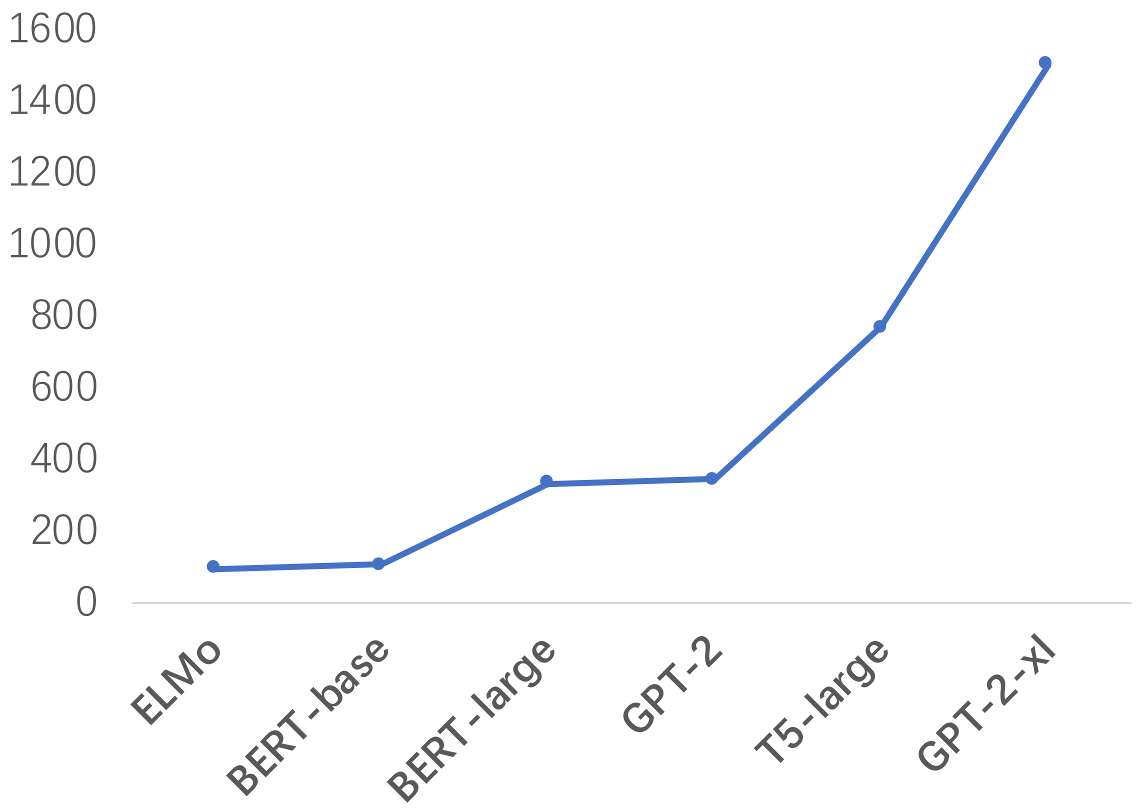
Pretrained Models in NLP

Large models

- Slow inference speed
- State-of-the-art performance
- Resource-demanding

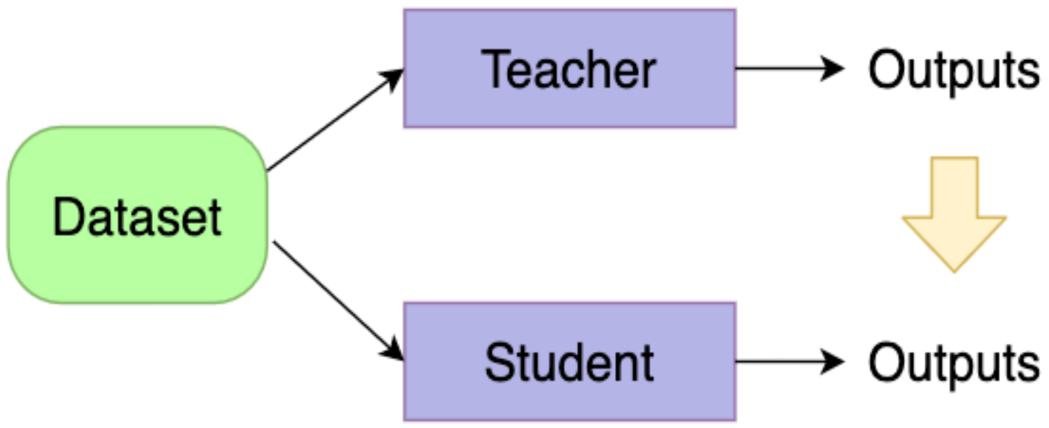
Small models

- Fast inference speed
- Inferior performance
- Resource-friendly



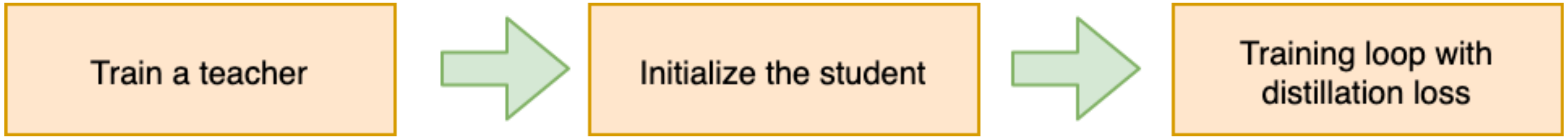
What is knowledge distillation (KD)?

Knowledge distillation is a technique of transferring knowledge from a large (teacher) model to a small (student) model, <u>without significant loss in performance.</u>



Motivation

- Various distillation methods usually share a common workflow:

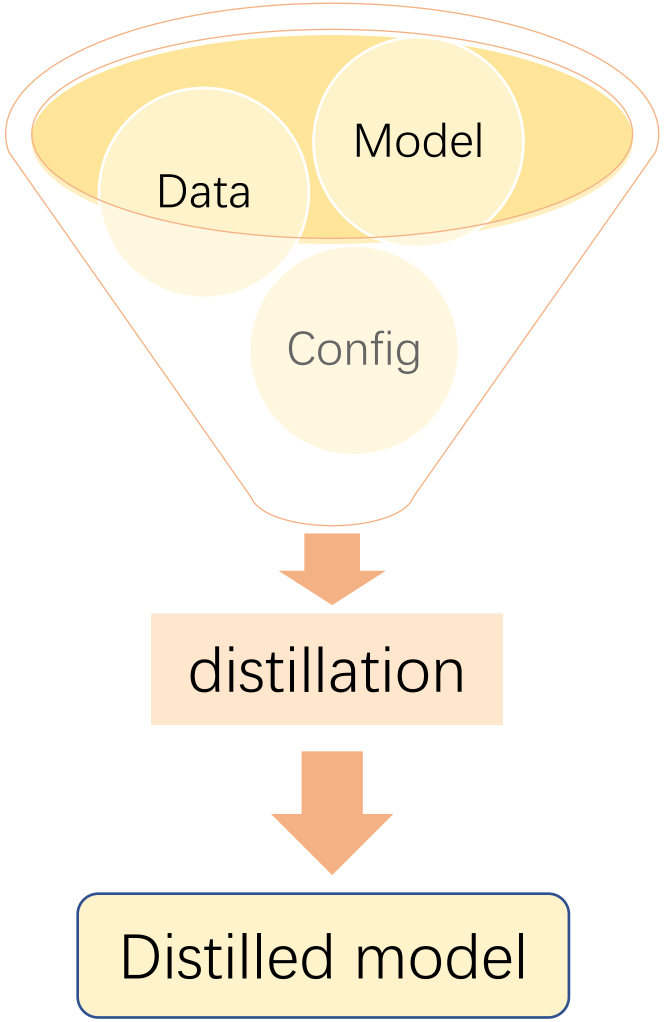


- A reusable distillation workflow framework
    1. Reduce redundant coding
    2. Distillation losses/strategies as plugins
    3. Achieve great flexibility in experimenting with different methods

TextBrewer

- A PyTorch based knowledge distillation toolkit for NLP

- Features
  - Flexibility
    - customizable configurations
  - Easy-to-use
    - re-uses the most parts of their existing training scripts
  - Wide-model-support
    - especially transformer-based models
  - Built for NLP
    - have been test on different tasks

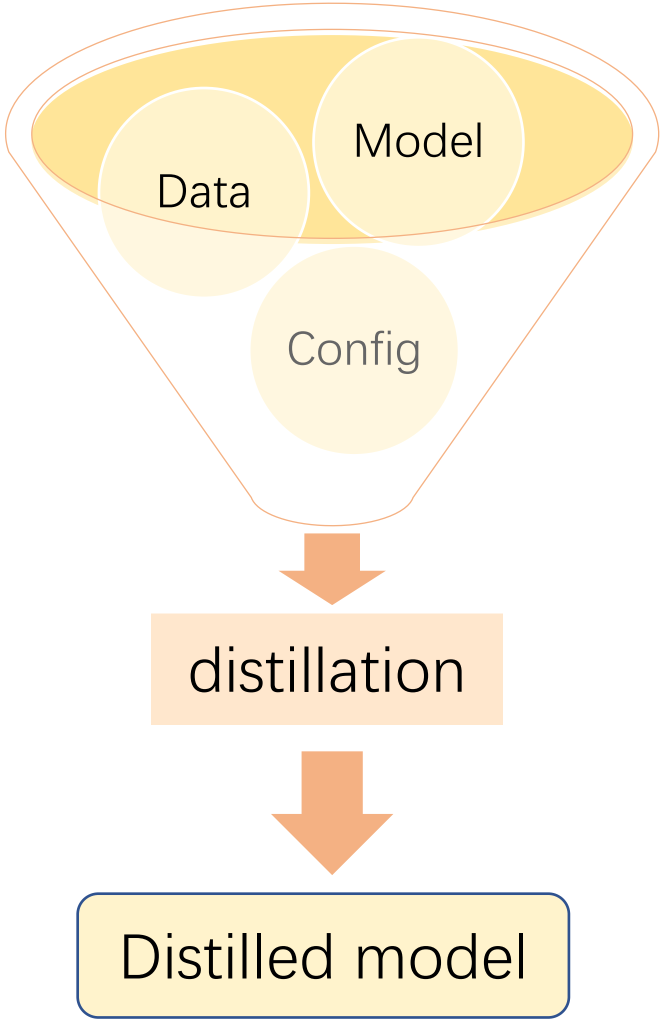


- Available at : http://textbrewer.hfl-rc.com

TextBrewer

- A PyTorch based knowledge distillation toolkit for NLP

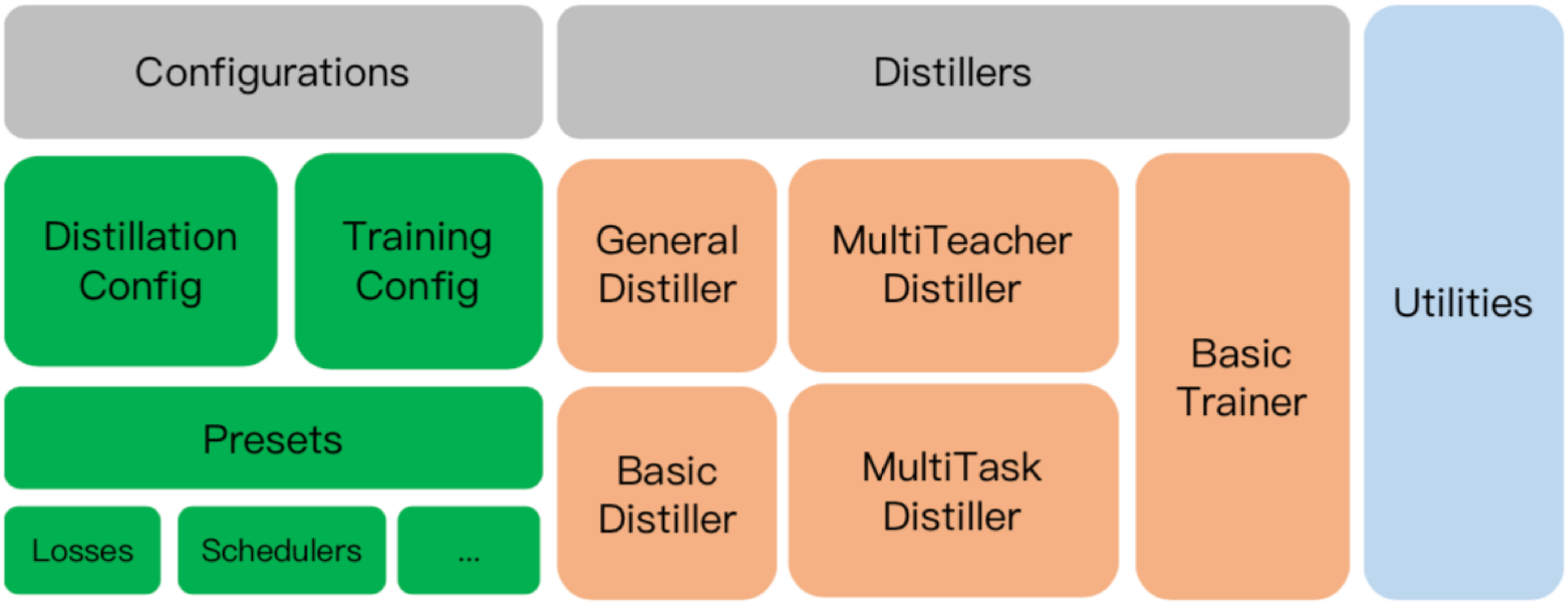
Tasks	Models	Distillation modes
Text Classification	BERT/RoBERTa	Traditional
MRC	Electra	Multi-teacher
Sequence Labeling	LSTM/GRU	Multi-task
…	…	normal training



- Available at : http://textbrewer.hfl-rc.com

Architecture

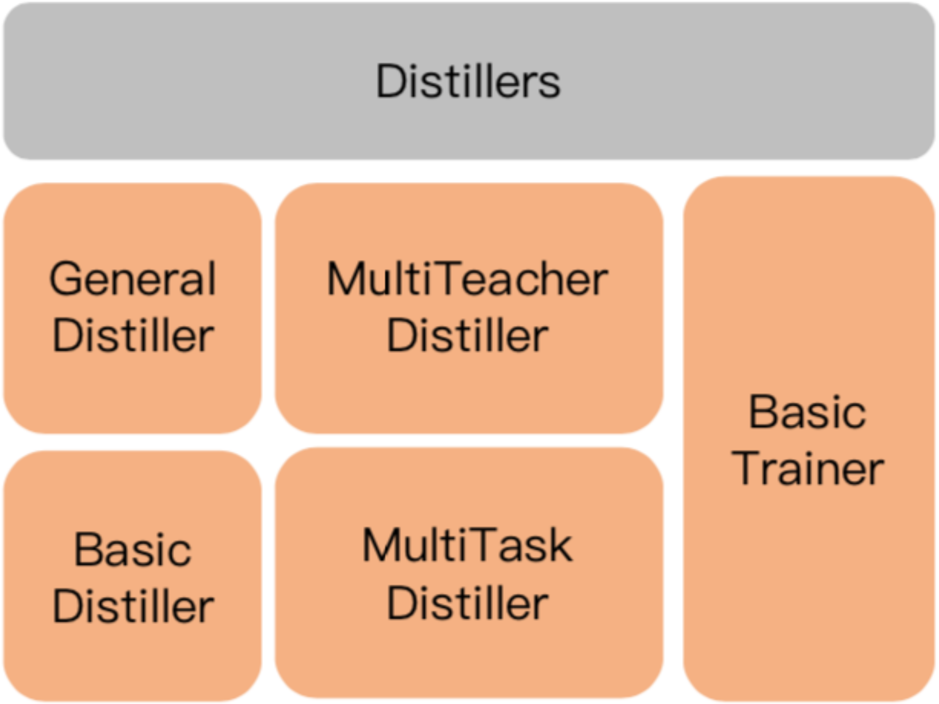
- An overview of the main functionalities of TextBrewer



Distillers

- Automatically train/distill and save models

- Five distillers:
  1. `BasicDistiller`
  2. `GeneralDistiller`
  3. `MultiTeacherDistiller`
  4. `MultiTaskDistiller`
  5. `BasicTrainer`



Configurations

1. `TrainingConfig`
   - gradient accumulation steps
   - checkpoint frequency
   - log directory
   - output directory
   - device

2. `DistillationConfig`
   - temperature
   - KD loss type
   - KD loss weight
   - hard-label loss weight
   - intermediate matches
   - if enable caching logits
   - …

Configurations

1. TrainingConfig

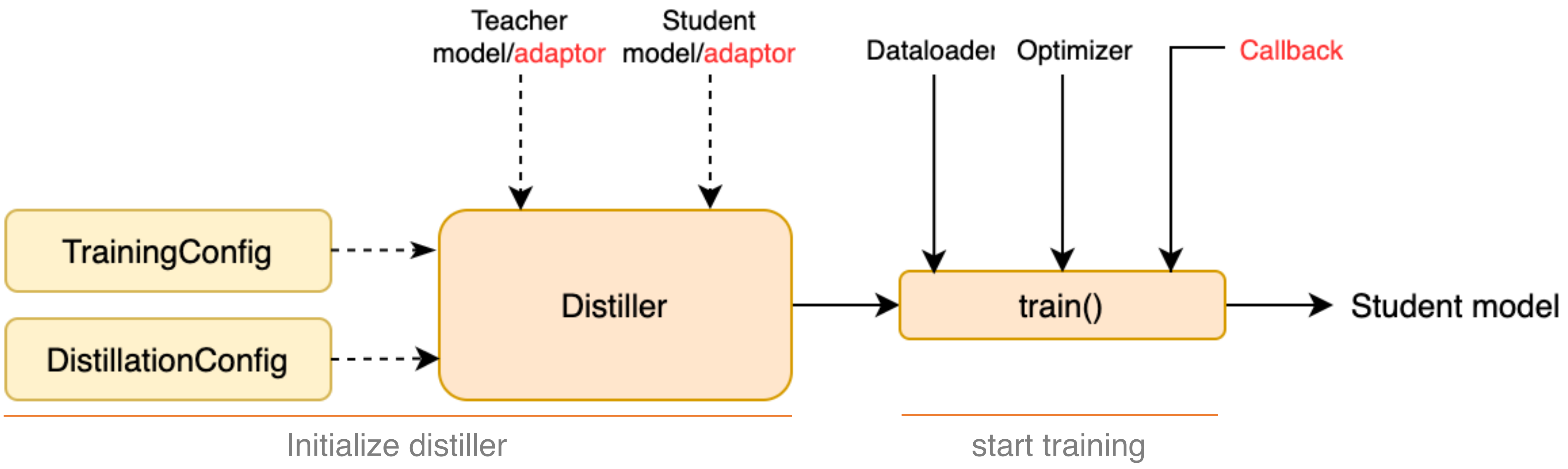
```
{"gradient_accumulation_steps" : 1,
 "ckpt_epoch_frequency" : 1,
 "ckpt_steps" : None,
 "log_dir" : "./logs",
 "output_dir" : "./saved_models",
 "device" : "cuda"}
```

2. DistillationConfig

```
{"temperature" : 8,
 "temperature_scheduler": None,
 "hard_label_weight": 0,
 "hard_label_weight_scheduler": None,
 "kd_loss_type": "ce",
 "kd_loss_weight": 1,
 "kd_loss_weight_scheduler": None,
 "probability_shift": False,
 "intermediate_matches": [
 1  {"layer_T": 0, "layer_S": 0, "feature": "hidden",
     "loss": "hidden_mse", "weight": 1, "proj": ["linear",312,768]},
 2  {"layer_T": 3, "layer_S": 1, "feature": "hidden",
     "loss": "hidden_mse", "weight": 1, "proj": ["linear",312,768]},
 3  {"layer_T": 6, "layer_S": 2, "feature": "hidden",
     "loss": "hidden_mse", "weight": 1, "proj": ["linear",312,768]},
 4  {"layer_T": 9, "layer_S": 3, "feature": "hidden",
     "loss": "hidden_mse", "weight": 1, "proj": ["linear",312,768]},
 5  {"layer_T": 12, "layer_S": 4, "feature": "hidden",
     "loss": "hidden_mse", "weight": 1, "proj": ["linear",312,768]},
 6  {"layer_T": [0,0], "layer_S":[0,0], "feature": "hidden",
     "loss": "nst", "weight": 1},
 7  {"layer_T": [3,3], "layer_S":[1,1], "feature": "hidden",
     "loss": "nst", "weight": 1},
 8  {"layer_T": [6,6], "layer_S":[2,2], "feature": "hidden",
     "loss": "nst", "weight": 1},
 9  {"layer_T": [9,9], "layer_S":[3,3], "feature": "hidden",
     "loss": "nst", "weight": 1},
 10 {"layer_T": [12,12], "layer_S":[4,4], "feature": "hidden",
     "loss": "nst", "weight": 1}]}
```

Workflow

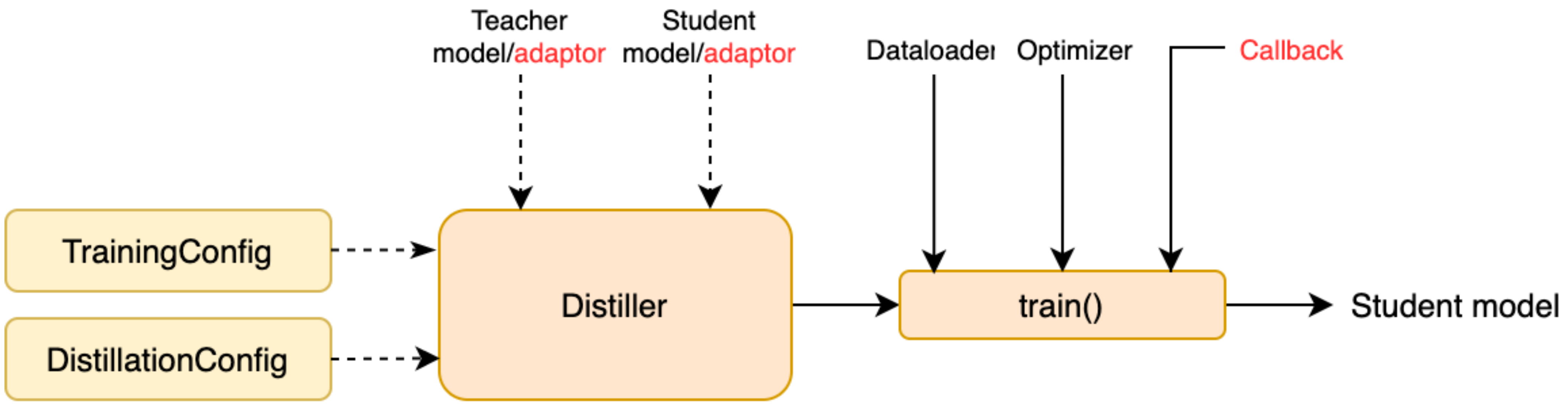
- Distillation with TextBrewer:



1. Initialize configurations and a distiller
2. Define **adaptors** and a **callback function**
3. Call the **train** method of the distiller

Workflow

- Distillation with TextBrewer



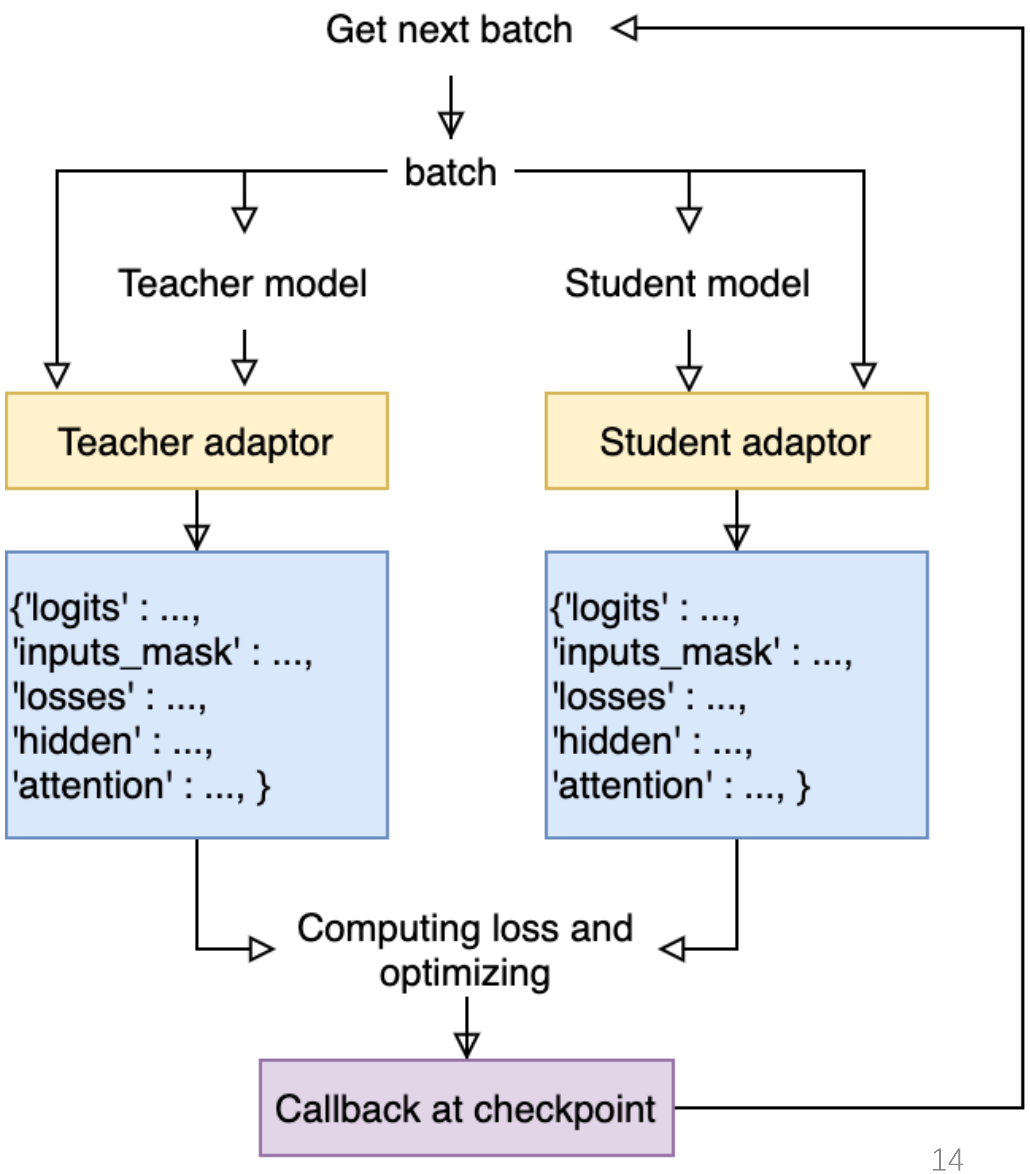

```
train_config = TrainingConfig()
distill_config = DistillationConfig()

distiller = GeneralTrainer(train_config = train_config, distill_config = distill_config,
                           model_T = teacher, model_S = student,
                           adatpor_T = my_adaptor_T, adaptor_S = my_adaptor_S)
with distiller:
    distiller.train(optimizer, dataloader, num_epochs, callback=my_callback)
```

Adaptor

```
adatpor(batch: Union[Dict,Tuple], model_outputs: Tuple) -> Dict
```

- Translates the inputs and outputs for the distiller

```
'''
class Model(nn.Module):
    def forward(self, input_ids, attention_mask, labels, ...):
        ...
        return logits, hidden_states, loss
'''
def simpleAdaptor(batch, model_outputs):
    return {'logits': (model_outputs[0],),
            'hidden': model_outputs[1],
            'input_mask': batch[1]}
```

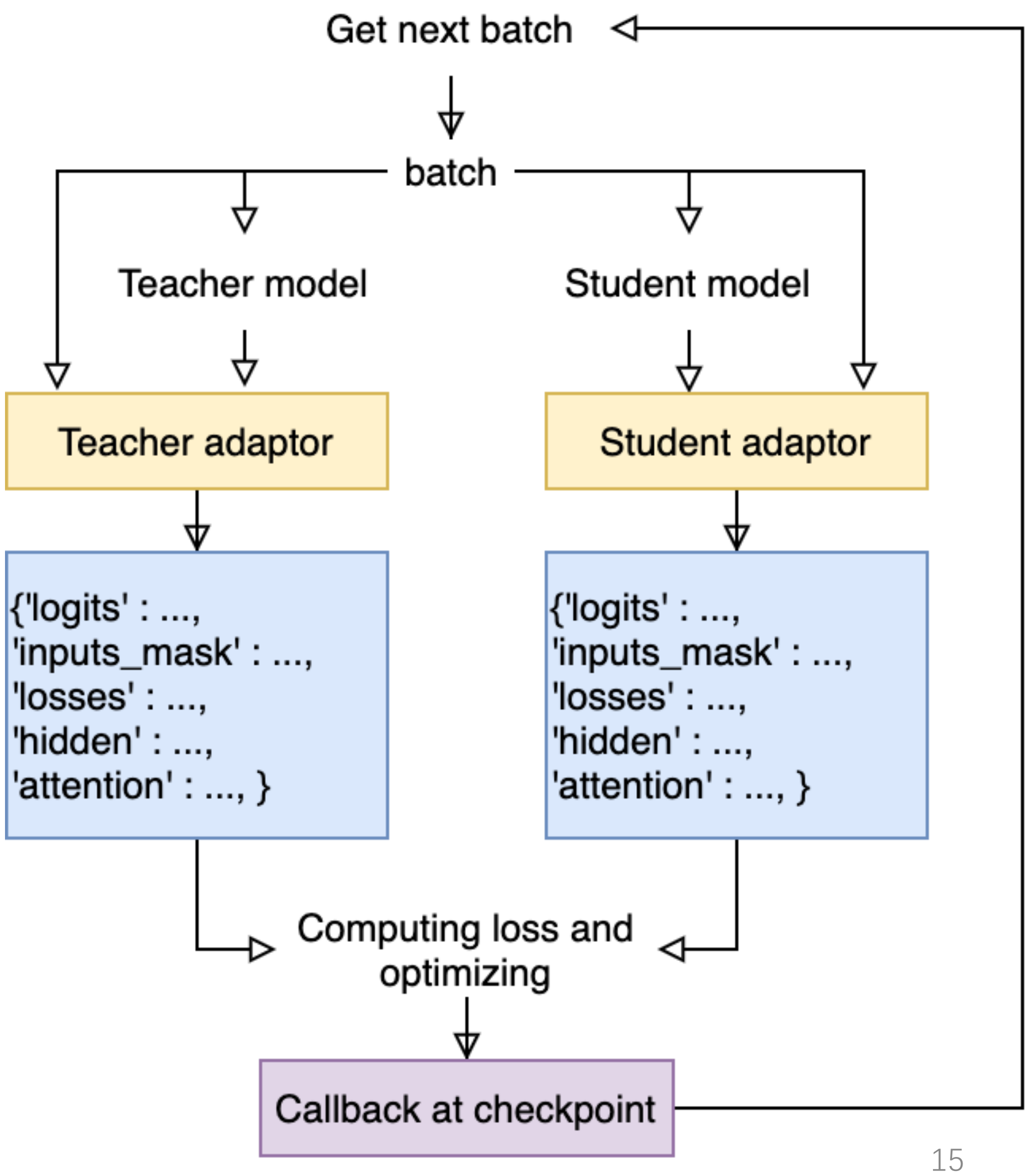


Callback

```
callback(model: torch.nn.Module, step: int) -> Any
```

- For monitoring performance during training

```python
def predict(model, eval_dataset, step, args):
    # your evaluation code here
    ...

my_callback = partial(predict, eval_dataset=my_eval_dataset, args=args)
with distiller:
    distiller.train(..., callback=my_callback)
```



Minimal workflow

```python
from textbrewer import GeneralDistiller
from textbrewer import TrainingConfig, DistillationConfig

# We omit the initialization of models, optimizer, and dataloader.
teacher_model : torch.nn.Module = ...
student_model : torch.nn.Module = ...
dataloader : torch.utils.data.DataLoader = ...
optimizer : torch.optim.Optimizer = ...
scheduler : torch.optim.lr_scheduler = ...

def simple_adaptor(batch, model_outputs):
    # We assume that the first element of model_outputs
    # is the logits before softmax
    return {'logits': model_outputs[0]}

train_config = TrainingConfig()
distill_config = DistillationConfig()
distiller = GeneralDistiller(
    train_config=train_config, distill_config = distill_config,
    model_T = teacher_model, model_S = student_model,
    adaptor_T = simple_adaptor, adaptor_S = simple_adaptor)

distiller.train(optimizer, scheduler,
    dataloader, num_epochs, callback=None)
```

define adaptor

Initialize configurations and distiller

RUN!

Experiments

- English datasets
  - **MNLI**
    sentence-pair classification
  - **SQuAD**
    machine reading comprehension
  - **CoNLL-2003**
    named entity recognition

- Chinese datasets
  - **XNLI** and **LCQMC**
    sentence-pair classification
  - **CMRC 2018** and **DRCD**
    SQuAD-like machine reading comprehension

Dataset	**Task**	**Metrics**	**#Train**	**#Dev**
MNLI	Classification	Acc	393K	20K
SQuAD	MRC	EM/F1	88K	11K
CoNLL-2003	NER	F1	23K	6K
XNLI	Classification	Acc	393K	2.5K
LCQMC	Classification	Acc	293K	8.8K
CMRC 2018	MRC	EM/F1	10K	3.4K
DRCD	MRC	EM/F1	27K	3.5K

Experiments

- Model configurations

Model	# Layers	Hidden size	Feed-forward size	# Parameters	Relative size
$\text{BERT}_{\text{BASE}}$ (teacher)	12	768	3072	108M	100%
T6	6	768	3072	65M	60%
T3	3	768	3072	44M	41%
T3-small	3	384	1536	17M	16%
T4-tiny	4	312	1200	14M	13%
BiGRU	1	768	-	31M	29%

- English models: initialized with the weight released by Google
- Chinese models: initialized with RoBERTa-wwm-ext

Results on English datasets

- Single-teacher distillation
  - All the T6 models achieve 99% performance of the teachers
  - T4-tiny outperforms TinyBERT
  - T4-tiny outperforms T3-small in most cases
  - Data augmentation (DA) is critical

Model	MNLI		SQuAD		CoNLL-2003
	m	mm	EM	F1	F1
$\text{BERT}_{\text{BASE}}$	83.7	84.0	81.5	88.6	91.1
Public					
DistilBERT	81.6	81.1	79.1	86.9	-
TinyBERT	80.5	81.0	-	-	-
+DA	82.8	82.9	72.7	82.1	-
TextBrewer					
BiGRU	-	-	-	-	85.3
T6	83.6	84.0	80.8	88.1	90.7
T3	81.6	82.5	76.3	84.8	87.5
T3-small	81.3	81.7	72.3	81.4	78.6
T4-tiny	82.0	82.6	73.7	82.5	77.5
+DA	-	-	75.2	84.0	89.1

Results on English datasets

- Multi-teacher distillation
  - All teachers are $\text{BERT}_{\text{base}}$
  - Student model is the same as the teacher
  - The student achieves the best performance, higher than the ensemble result

Model	MNLI		SQuAD		CoNLL-2003
	m	mm	EM	F1	F1
Teacher 1	83.6	84.0	81.1	88.6	91.2
Teacher 2	83.6	84.2	81.2	88.5	90.8
Teacher 3	83.7	83.8	81.2	88.7	91.3
Ensemble	84.3	84.7	82.3	89.4	91.5
Student	**84.8**	**85.3**	**83.5**	**90.0**	**91.6**

Results on Chinese datasets

- Single-teacher distillation
  - T4-tiny still outperforms T3-small on all tasks
  - Consistent with the observations on English tasks
  - CMRC 2018 has a relatively small training set, DA has a much more significant effect

Model	XNLI	LCQMC	CMRC 2018		DRCD	
	Acc	Acc	EM	F1	EM	F1
RoBERTa-wwm	79.9	89.4	68.8	86.4	86.5	92.5
T3	78.4	89.0	63.4	82.4	76.7	85.2
+DA	-	-	66.4	84.2	78.2	86.4
T3-small	76.0	88.1	46.1	71.0	71.4	82.2
+DA	-	-	58.0	79.3	75.8	84.8
T4-tiny	76.2	88.4	54.3	76.8	75.5	84.9
+DA	-	-	61.8	81.8	77.3	86.1

Summary

- Conclusion
  - We present TextBrewer, a flexible PyTorch-based distillation toolkit for NLP research and applications.
  - TextBrewer is easy-to-use, and provides rich customization options.
  - A series of experiments shows that the distilled models can achieve state-of-the-art results with simple settings.

- Future work
  - Expand TextBrewer's scope of application
  - Automatic search for student model structures

Get TextBrewer

TextBrewer

GitHub repo http://textbrewer.hfl-rc.com

Install via pip `pip install textbrewer`

Star If you like this project, you are welcome to give it a star!

Thanks for listening!

ziqingyang@gmail.com

http://textbrewer.hfl-rc.com