2023 08. 연구실 하계 세미나

Cost-Efficient Utilization Of LLM

문현석

Natural Language Processing & Artificial Intelligence

Overview

Parameter-Efficient Tuning

Why PEFT?

Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning

(NIPS 2022)
https://papers.nips.cc/paper_files/paper/2022/file/0cde695b83bd186c1fd456302888454c-Paper-Conference.pdf

How to PEFT?

Parameter-Efficient Fine-Tuning Design Spaces

(ICLR 2023)
https://openreview.net/pdf?id=XSRSWxyJIC

Only Train?

Distill or Annotate?
Cost-Efficient Fine-Tuning of Compact Models

(ACL 2023)
https://aclanthology.org/2023.acl-long.622.pdf

중심주제

용어정의:

PEFT	= Parameter-Efficient Tuning
ICL	= In-Context Learning

PEFT는 ICL보다 성능도 좋고 효율도 좋다

새로운 PEFT방법론을 제안한다 (T-Few)

Related Works 이면서 저자의 의견

ICL에 대하여

장점

- Enable a single model to perform many tasks immediately **without fine-tuning**.
- Enable **mixed-task batches**, where different examples in a batch of data correspond to different tasks by using different contexts in the input.

단점

Computational Cost

- model needs to process all of the in-context labeled examples
- self-attention operation만을 고려하면
  k길이의 데이터를 k-shot ICL하는 것은
  k길이의 데이터 k+1개를 처리하는 것과 같음
- For GPT3, 32-shot ICL
  512 tokens per in-context example ➔ 144gb cache

  : 32 examples × 512 tokens × 96 layers
  × 12288 d_model × 32 bits

ICL's unexpected behavior

- **ordering of examples** in the context heavily influences the model's predictions
- ICL can still perform well even if the **in-context example labels are swapped**

+ ICL typically produces **inferior performance compared to fine-tuning**

Related Works 이면서 저자의 의견

PEFT에 대하여

- 기존에 쓰이고 있는 adapter 기반의 PEFT는 필연적으로 새로운 parameter를 추가

: resulting in small but **non-negligible increases in computational costs and memory.**

- PEFT도 효과적인 방법이긴 하나, **very little labeled data setting**에 대해서는 잘 다루고 있지 않음

➔ **Few shot setting을 고려한 PEFT가 필요하다**

Proposed Method : T-Few recipe

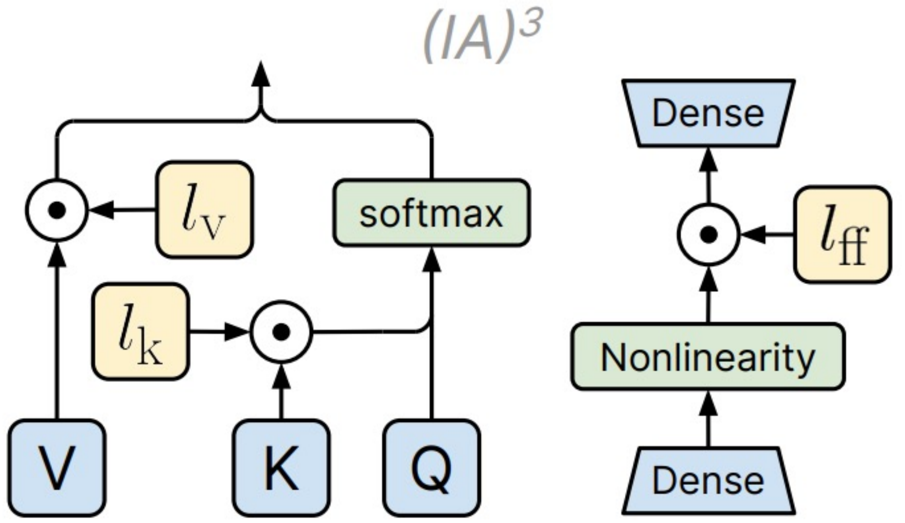
PEFT strategy : $(\mathbf{IA})^3$

Infused Adapter by Inhibiting and Amplifying Inner Activations

Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning

고려되는 사항

- must add or update **as few parameters as possible** to av oid incurring storage and memory costs
- it should achieve **strong accuracy after few-shot training** on new tasks.
- it must allow for **mixed-task batches,** since that is a capab ility of ICL



$$l \in \mathbb{R}^d$$

mixed-task batch관련

- P-tuning이나 prompt tuning도 가능 (task prompt를 여러개 붙여서 사용하면 되기에)
- 하지만 성능이 안나온다

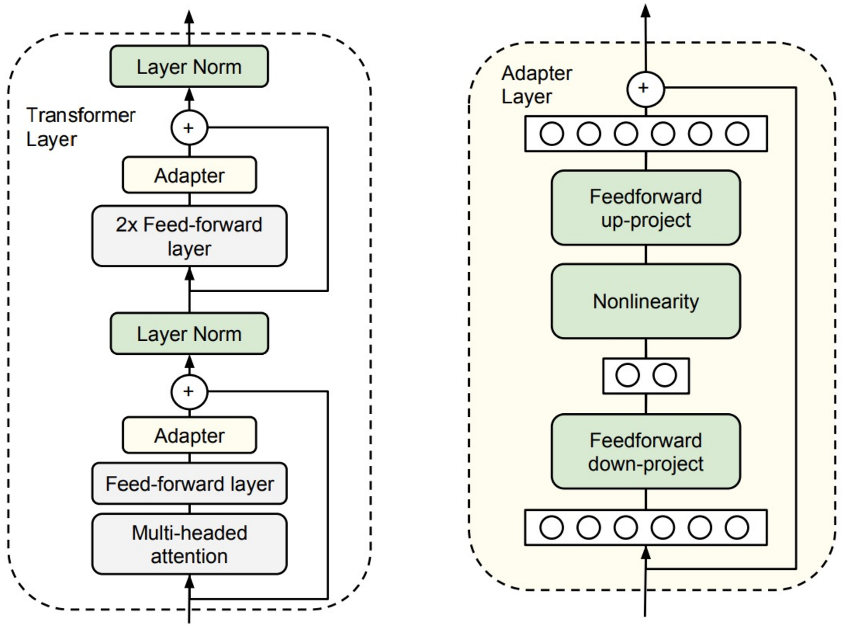
Mixed-task batch에 대해서는

Task에 맞는 l을 단순하게 더해서 사용하면 됨

Proposed Method : T-Few recipe

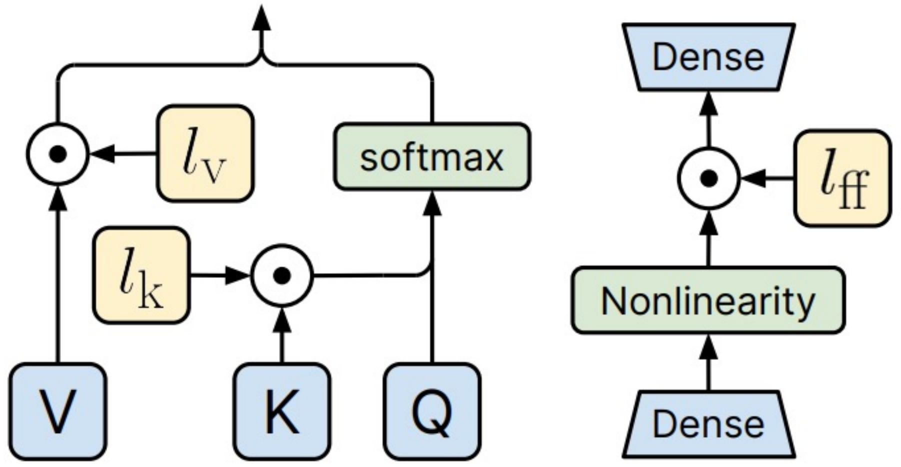
PEFT strategy : $(\mathbf{IA})^3$

Conventional Adapter



$$L\,(d\ * d_{mid}\ * 2\ * 2)$$

$(\mathbf{IA})^3$



$$l \in \mathbb{R}^d$$

$$L\left(l_v + l_k + l_{ff}\right) = \mathrm{L}\ * 3 * \mathrm{d}$$

Proposed Method : T-Few recipe

학습 방법

맞는 label생성 촉진

$$L_{\text{LM}} = -\frac{1}{T}\sum_t \log p(y_t|\mathbf{x}, y_{<t})$$

틀린 label생성 억제

(N개의 서로 다른 incorrect target sentences)

$$L_{\text{UL}} = -\frac{\sum_{n=1}^{N}\sum_{t=1}^{T^{(n)}} \log(1 - p(\hat{y}_i^{(n)}|\mathbf{x}, \hat{y}_{<t}^{(n)}))}{\sum_{n=1}^{N} T^{(n)}}$$

Short label을 선호하는 문제를 해결하기 위함

(length normalize term)

$$\beta(\mathbf{x}, \mathbf{y}) = \frac{1}{T}\sum_{t=1}^{T} \log p(y_t|\mathbf{x}, y_{<t})$$

$$L_{\text{LN}} = -\log \frac{\exp(\beta(\mathbf{x}, \mathbf{y}))}{\exp(\beta(\mathbf{x}, \mathbf{y})) + \sum_{n=1}^{N} \exp(\beta(\mathbf{x}, \hat{\mathbf{y}}^{(n)}))}$$

L_{LM} : 60.7

L_{LM} + L_{LN} : 62.71

L_{LM} + L_{LN} + L_{UL} : 63.3

Experiments

실험 세팅 및 실험 결과

Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning

사용한 모델: T0 - 3B/11B
(zero shot specialized training to T5)

실험 세팅: few shot 20~70 shots
(GPT3과 동일한 세팅)

: GPT3에서 사용했던 few shot 데이터가 공개되지 않아서, 5개의 few shot 데이터 자체제작 (실험 5배)

데이터셋: GPT3 검증시와 동일한 데이터셋

: COPA, H-SWAG, Story Cloze, ANLI, CB , RTE, WSC, Winogrande

+ RAFT benchmark
(real-world few shot benchmark)
banking / medical …

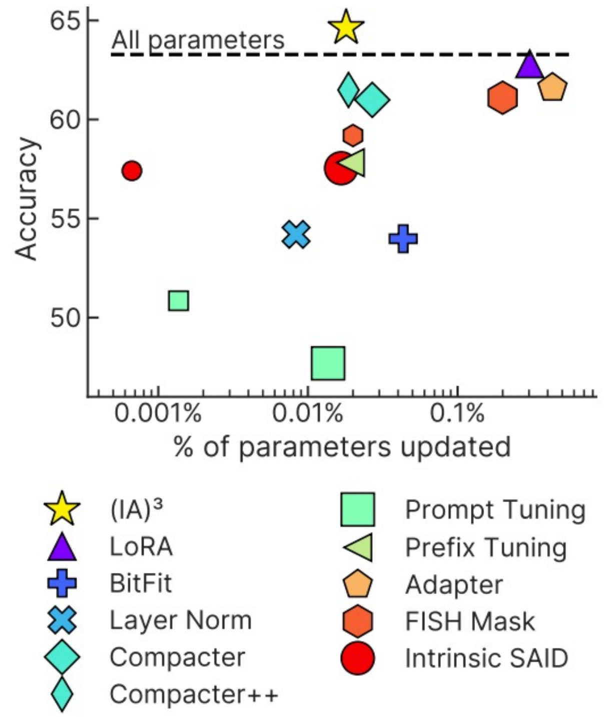


Figure 2: Accuracy of PEFT methods with L_{UL} and L_{LN} when applied to T0-3B. Methods that with variable parameter budgets are represented with larger and smaller markers for more or less parameters.

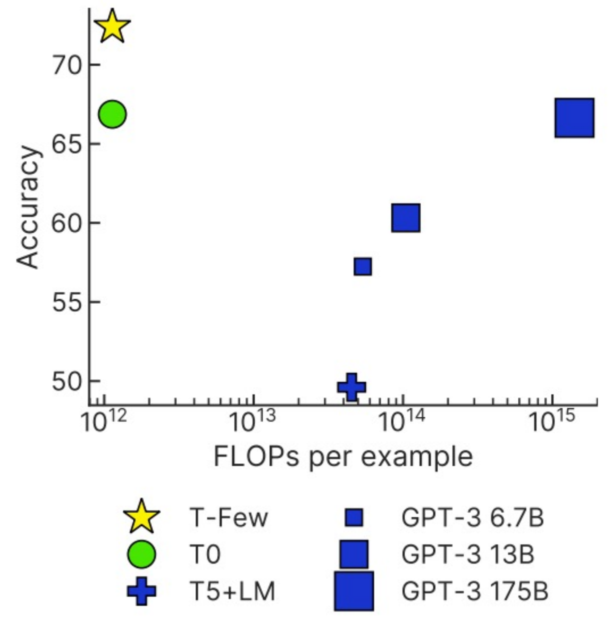
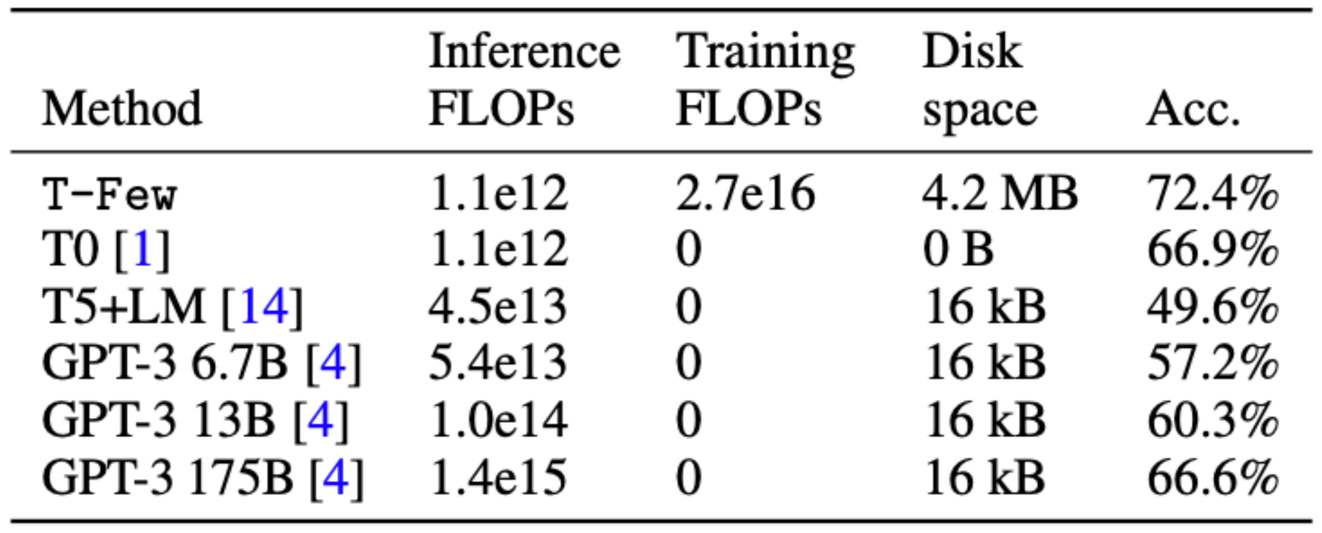


Figure 3: Accuracy of different few-shot learning methods. T-Few uses (IA)^3 for PEFT methods of T0, T0 uses zero-shot learning, and T5+LM and the GPT-3 variants use few-shot ICL. The x-axis corresponds to inference costs; details are provided in section 4.2.

Experiments

ICL vs T-Few

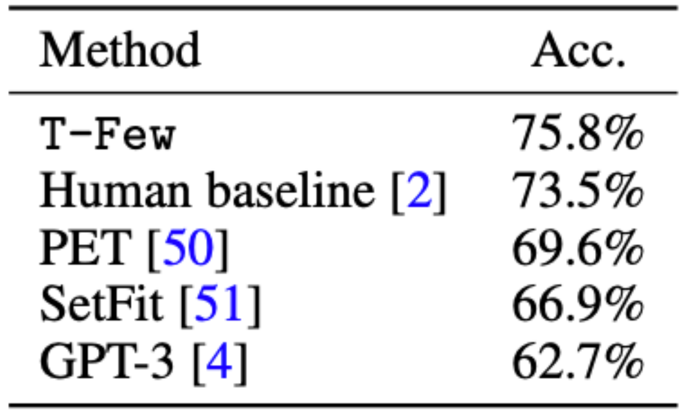
Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning



Method	Inference FLOPs	Training FLOPs	Disk space	Acc.
T-Few	1.1e12	2.7e16	4.2 MB	72.4%
T0 [1]	1.1e12	0	0 B	66.9%
T5+LM [14]	4.5e13	0	16 kB	49.6%
GPT-3 6.7B [4]	5.4e13	0	16 kB	57.2%
GPT-3 13B [4]	1.0e14	0	16 kB	60.3%
GPT-3 175B [4]	1.4e15	0	16 kB	66.6%

Table 1: Accuracy on held-out T0 tasks and computational costs for different few-shot learning methods and models. T-Few attains the highest accuracy with 1,000× lower computational cost than ICL with GPT-3 175B. Fine-tuning with T-Few costs about as much as ICL on 20 examples with GPT-3 175B.

FLOPS: FLoating point Operations Per Second



Method	Acc.
T-Few	75.8%
Human baseline [2]	73.5%
PET [50]	69.6%
SetFit [51]	66.9%
GPT-3 [4]	62.7%

Table 2: Top-5 best methods on RAFT as of writing. T-Few is the first method to outperform the human baseline and achieves over 6% higher accuracy than the next-best method.

**속도도 빠르고,
T-few는 심지어
사람보다 성능 좋음**

Experiments

ICL vs T-Few

FLOPs are not a direct measurement of real-world computational cost
(latency, power usage, and other costs can vary significantly depending on hardware and other factors)
- https://arxiv.org/pdf/2110.12894.pdf (ICLR2022)

However, we focus on **FLOPs** because it is a hardware-independent metric that closely with **real-world costs**

with N parameters

- **Decoder-only Transformer** (e.g. the GPT series)
  - **Inference:** 2N FLOPs per token
  - **Training:** 6N FLOPs per token
- **Encoder-Decoder Transformer** (e.g. T0, T5)
  - **Inference:** N FLOPs per token
  - **Training:** 3N FLOPs per token

Input sequence 길이의 중앙값은 103
ICL에 쓰였던 데이터 길이의 중앙값은 98

Inference:

T-Few requires **11e9×103 = 1.1e12** FLOPs

GPT-3 175B requires **2×175e9× (41 × 98 + 103) = 1.4e15** FLOPs

Training: (1000 with batch size 8, input length 103)

→ 3 × 11e9 × 1, 000 × 8 × 103 = 2.7e16 FLOPs

T-few(11B)를 학습시키는 데에는 GPT3 175B에서 20개 샘플에 대한 ICL을 돌리는 것과 같은 비용이 든다.

+ 학습하는 데 A100으로 1시간 30분 걸렸다고 함 -> Microsoft Azure 기준 2달러 소요

Introduction

Main Focus

현재 활용되고 있는 대표적인 PEFT 전략들

: Adapter, Prefix Tuning, BitFit, LoRA … …

Find Underlying design patterns of PEFT

(Rather than developing individual tuning strategy)

Main Focus

논문에서 밝히고 있는 Design Pattern of PEFT

(i) Layer를 특성별로 묶을 때, 어떤 기준을 적용해야 하는가? **(Group layers in a spindle pattern)**

(ii) Layer별로 학습할 parameter를 어떻게 결정해야 하는가? **(Allocate evenly among layers)**

(iii) 특정 Layer group만을 학습시킬까 모든 Layer를 학습시킬까? **(Tune all groups)**

(iv) Layer별로 어떤 PEFT전략을 적용시킬까? (Assign appropriate tuning strategies to each group)

Our goal is not to list all possible design choices,
but to show **how design spaces can guide parameter-efficient fine-tuning research.**

Design Pattern

Design Space

S0 → S1 → … → S4 순차적으로 design space를 줄여가면서 패턴 파악

S0: 가능한 모든 선택지 고려

- 학습전략 :	Adapter, Prefix, BitFit, LoRA
- 학습 대상 Layer:	각 layer가 학습될 확률은 0.5
- Layer 별 trainable parameters의 수 :	Random

이들 선택지 중에서, 자체적으로 선정한 4가지 기준을 통해 100개 모델을 랜덤하게 선택:

→ 선택한 모델들의 성능 비교를 통한 패턴 분석

→ 분석한 패턴을 바탕으로, 다음 단계의 refining 과정을 거칠 S1 design space 결정

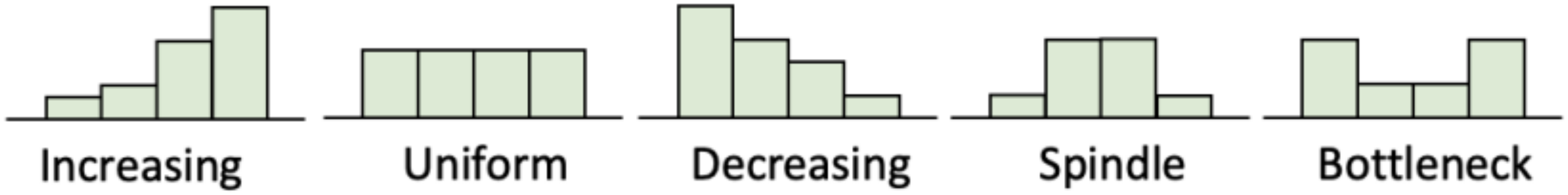
Design Pattern

S1 : Grouping Constraints

how to assemble the layers into groups that will be tuned using the same strategy

- T0의 24개 layer 각각에 대한 variant를 모두 고려하면 선택지가 너무 과하게 넓어짐
- 24개의 layer를, **특성을 공유하는 4개 그룹으로 나누어 variant 고려**
- 이 때**, 4개의 layer그룹을 어떻게 결정할지**에 대한 논의 (ex: G1-앞6개 layer G4-뒤6개 layer 등)

Increasing (Ni+1 > Ni):	the number of layers in groups gradually increases;
Uniform (Ni+1 = Ni):	the number of layers in groups is the same
Decreasing (Ni+1 < Ni):	the number of layers in groups gradually decreases
Spindle (N1 < N2 = N3 > N4):	the numbers of layers in groups at both ends are smaller
Bottleneck (N1 > N2 = N3 < N4):	the numbers of layers in groups at both ends are bigger



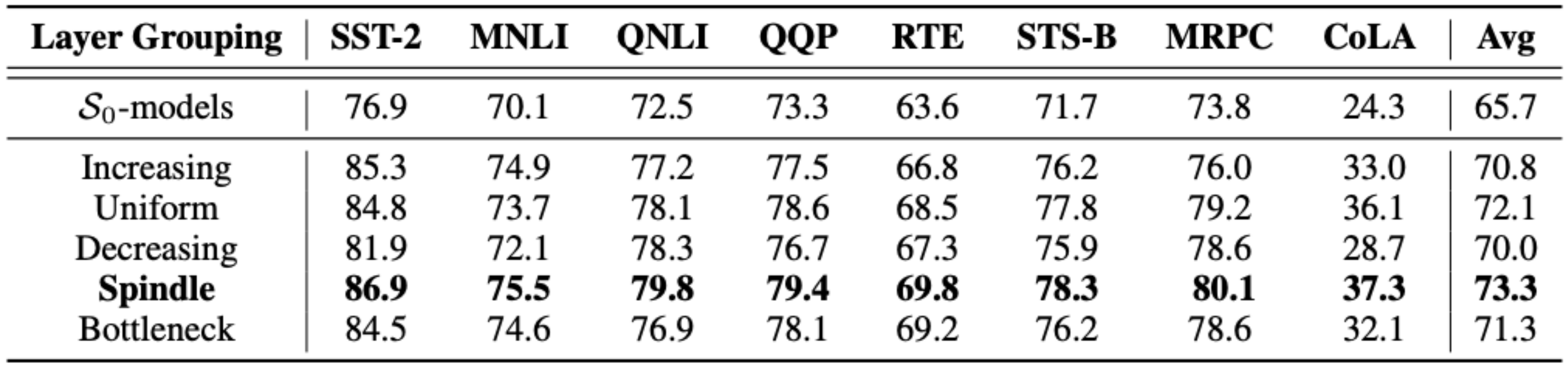
Design Pattern

S1 : Grouping Constraints

실험 방법

(i) randomly **sample 100 models** from the S0 design space that **satisfy each grouping pattern constraint**
(ii) fine-tune with 3 epochs
(iii) compute the average performance for each design space.

Table 1: Average performance (low-compute, low-epoch regime: 100 random models, 3 tuning epochs) on the GLUE datasets using the T5-base pretrained backbone. We compare adding different layer grouping constraints to the $\mathcal{S}_0$ design space.



Layer Grouping	SST-2	MNLI	QNLI	QQP	RTE	STS-B	MRPC	CoLA	Avg
$\mathcal{S}_0$-models	76.9	70.1	72.5	73.3	63.6	71.7	73.8	24.3	65.7
Increasing	85.3	74.9	77.2	77.5	66.8	76.2	76.0	33.0	70.8
Uniform	84.8	73.7	78.1	78.6	68.5	77.8	79.2	36.1	72.1
Decreasing	81.9	72.1	78.3	76.7	67.3	75.9	78.6	28.7	70.0
Spindle	**86.9**	**75.5**	**79.8**	**79.4**	**69.8**	**78.3**	**80.1**	**37.3**	**73.3**
Bottleneck	84.5	74.6	76.9	78.1	69.2	76.2	78.6	32.1	71.3

➔ Applying the **spindle grouping partitioning** to S0 yields the **new design space S1.**

Ex) G1: L1~L3 / G2: L4~L12 / G3: L13~L21 / G4: L22~L24

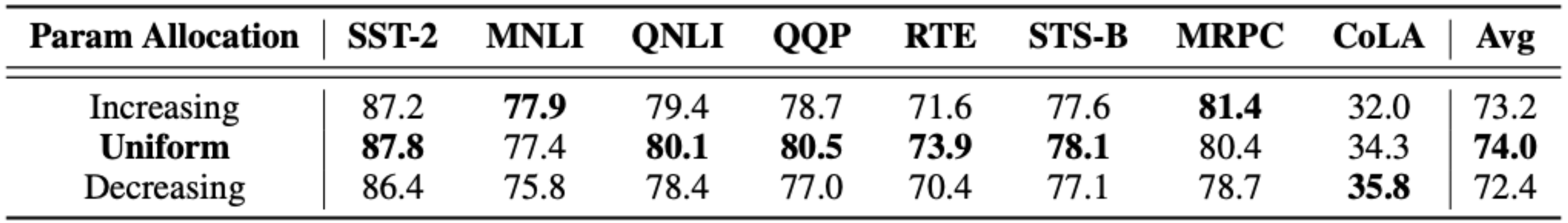
Design Pattern

S2 : Trainable Parameters per Layer

how to allocate the parameters within the layers of each group.

Increasing (ni+1 ≥ ni):	number of trainable parameters per layer increases or remains the same
Uniform (ni+1 = ni):	number of trainable parameters in every layer is constant
Decreasing (ni+1 ≤ ni):	number of trainable parameters per layer decreases or remains the same

Table 2: Average performance (low-compute, low-epoch regime: 100 random models, 3 tuning epochs) on the GLUE datasets using the T5-base pretrained backbone model. We compare adding different parameter allocation constraints to the S_1 design space.



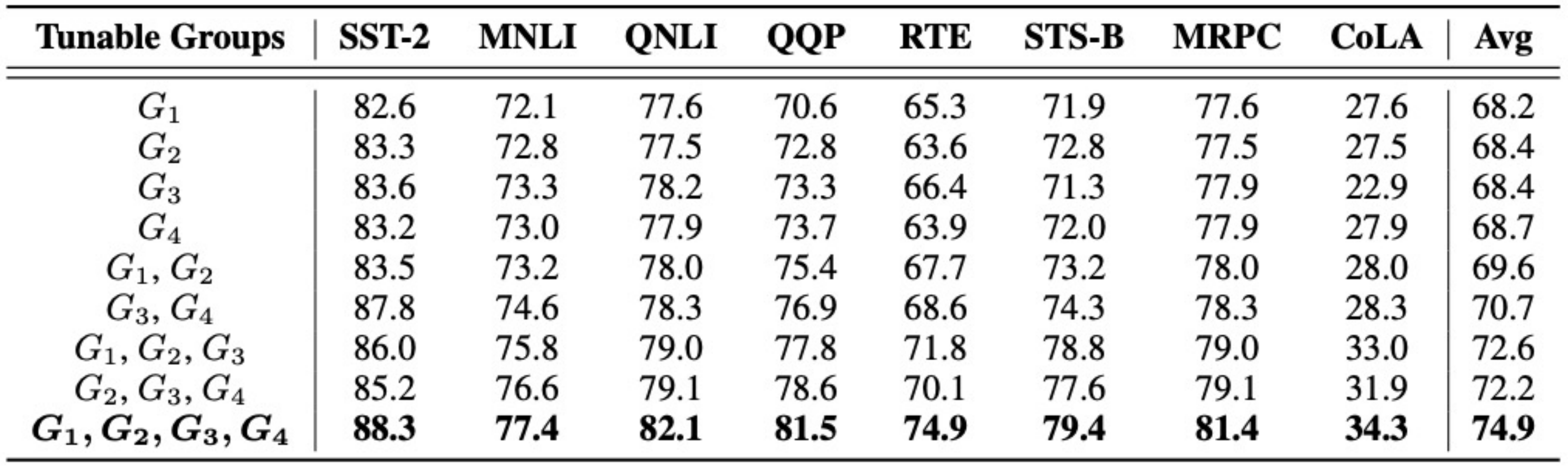
Param Allocation	SST-2	MNLI	QNLI	QQP	RTE	STS-B	MRPC	CoLA	Avg
Increasing	87.2	**77.9**	79.4	78.7	71.6	77.6	**81.4**	32.0	73.2
Uniform	**87.8**	77.4	**80.1**	**80.5**	**73.9**	**78.1**	80.4	34.3	**74.0**
Decreasing	86.4	75.8	78.4	77.0	70.4	77.1	78.7	**35.8**	72.4

➔ Allocating the number of trainable parameters to layers **uniformly**
yields the new design space S2.

S3 : Selecting the Groups

whether all groups actually need tuning (어떤 그룹을 실제 tuning할 것인지?)

Table 3: Average performance (low-compute, low-epoch regime: 100 random models, 3 tuning epochs) on the GLUE datasets using the T5-base pretrained backbone model. We compare adding different tunable group constraints to the S_2 design space.



Tunable Groups	SST-2	MNLI	QNLI	QQP	RTE	STS-B	MRPC	CoLA	Avg
G_1	82.6	72.1	77.6	70.6	65.3	71.9	77.6	27.6	68.2
G_2	83.3	72.8	77.5	72.8	63.6	72.8	77.5	27.5	68.4
G_3	83.6	73.3	78.2	73.3	66.4	71.3	77.9	22.9	68.4
G_4	83.2	73.0	77.9	73.7	63.9	72.0	77.9	27.9	68.7
G_1, G_2	83.5	73.2	78.0	75.4	67.7	73.2	78.0	28.0	69.6
G_3, G_4	87.8	74.6	78.3	76.9	68.6	74.3	78.3	28.3	70.7
G_1, G_2, G_3	86.0	75.8	79.0	77.8	71.8	78.8	79.0	33.0	72.6
G_2, G_3, G_4	85.2	76.6	79.1	78.6	70.1	77.6	79.1	31.9	72.2
$\boldsymbol{G_1, G_2, G_3, G_4}$	**88.3**	**77.4**	**82.1**	**81.5**	**74.9**	**79.4**	**81.4**	**34.3**	**74.9**

➔ **We will tune all the groups**.
We refer to S2 with this additional design pattern as the new **S3 design space**

Design Pattern

S4 : Selecting Strategies per Group

각 그룹에 어떤 학습 전략을 적용시킬까?

Strategy Assignment	SST-2	MNLI	QNLI	QQP	RTE	STS-B	MRPC	CoLA	Avg
G_1-Adapter (A)	89.8	83.5	84.9	80.8	72.5	80.8	**78.5**	**37.7**	76.1
G_1-Prefix (P)	89.3	83.1	84.4	80.1	70.1	80.0	77.6	33.0	74.7
G_1-BitFit (B)	89.0	82.9	84.1	81.4	72.0	81.1	77.0	30.8	74.8
G_1-LoRA (L)	89.9	83.6	85.0	81.1	71.8	81.0	78.8	35.3	75.8
G_1-(P, L)	89.1	82.8	85.1	81.2	71.9	81.5	79.1	35.0	75.7
G_1-(A, P)	89.8	82.8	84.8	81.1	72.2	81.3	79.2	36.4	75.9
G_1-**(A, L)**	89.6	**83.8**	**85.6**	81.3	**72.9**	**81.7**	**79.5**	**36.8**	**76.4**
G_1-(A, P, L)	89.6	83.5	85.2	81.5	72.2	81.4	79.2	35.2	75.9
G_1-(P, B, L)	89.3	83.6	85.5	81.6	72.3	81.0	78.8	35.7	76.0
G_1-(A, P, B)	89.2	83.3	84.8	**81.8**	72.5	81.1	78.6	35.6	75.8
G_1-(A, B, L)	89.8	83.4	84.8	81.1	72.6	81.6	79.4	34.8	75.9
G_1-(A, P, B, L)	**90.0**	83.1	85.3	81.6	72.6	81.4	79.2	36.5	76.1

Strategy Assignment	SST-2	MNLI	QNLI	QQP	RTE	STS-B	MRPC	CoLA	Avg
G_2-Adapter (A)	91.6	84.3	85.5	**82.3**	73.5	82.8	81.3	38.8	77.5
G_2-Prefix (P)	89.6	84.0	86.5	81.5	73.3	82.5	80.5	36.2	76.7
G_2-BitFit (B)	91.2	83.6	85.7	82.9	72.6	82.6	80.8	33.1	76.5
G_2-LoRA (L)	91.4	84.4	86.1	82.0	72.8	81.8	81.6	39.8	77.4
G_2-(P, L)	91.6	84.6	86.8	81.8	73.8	82.8	82.0	38.5	77.7
G_2-**(A, P)**	**92.2**	**84.2**	**87.1**	82.2	**74.4**	83.0	**82.5**	40.8	**78.3**
G_2-(A, L)	92.0	84.4	86.5	81.8	73.6	82.6	82.2	40.1	77.9
G_2-(A, P, L)	91.8	84.8	86.8	81.8	74.1	83.0	82.1	37.9	77.7
G_2-(P, B, L)	91.6	84.1	87.1	82.0	74.0	82.9	82.4	35.8	77.4
G_2-(A, P, B)	91.8	84.2	86.8	82.1	73.7	**83.3**	82.2	41.2	78.1
G_2-(A, B, L)	**92.2**	84.3	86.1	82.0	74.1	83.2	82.0	37.6	77.6
G_2-(A, P, B, L)	92.0	84.1	87.0	81.9	74.2	83.1	81.3	**42.4**	78.1

Strategy Assignment	SST-2	MNLI	QNLI	QQP	RTE	STS-B	MRPC	CoLA	Avg
G_3-Adapter (A)	92.5	85.3	87.5	**83.3**	73.9	84.0	83.8	**44.9**	79.4
G_3-Prefix (P)	91.5	84.7	86.7	82.6	74.2	83.8	82.9	40.5	78.4
G_3-BitFit (B)	91.9	84.3	87.0	82.0	73.6	84.1	83.3	36.1	77.8
G_3-LoRA (L)	92.8	85.4	87.8	83.5	74.7	82.4	84.0	44.0	79.3
G_3-(P, L)	93.0	85.2	88.3	83.8	75.2	84.4	84.2	37.9	79.0
G_3-(A, P)	92.4	85.6	88.1	83.6	75.0	84.2	84.0	41.8	79.3
G_3-(A, L)	92.0	85.9	88.2	83.1	75.3	84.3	83.9	42.2	79.4
G_3-(A, P, L)	92.6	86.0	87.5	83.4	75.6	84.6	83.5	43.9	79.6
G_3-(P, B, L)	92.7	85.8	87.2	83.7	75.2	84.5	83.8	40.8	79.2
G_3-**(A, P, B)**	93.3	**85.8**	**88.6**	**84.0**	75.5	**84.9**	84.1	42.1	**79.8**
G_3-(A, B, L)	**93.7**	86.5	88.0	83.2	**75.8**	84.2	84.2	39.7	79.4
G_3-(A, P, B, L)	93.3	85.6	87.7	83.8	75.2	84.3	**84.4**	41.6	79.4

Strategy Assignment	SST-2	MNLI	QNLI	QQP	RTE	STS-B	MRPC	CoLA	Avg
G_4-Adapter (A)	93.8	85.8	88.6	84.8	76.3	85.8	86.0	**48.5**	81.2
G_4-Prefix (P)	93.5	85.2	88.3	83.6	76.8	85.3	85.6	44.8	80.3
G_4-BitFit (B)	94.1	85.3	88.9	84.4	77.1	85.4	86.2	46.1	80.9
G_4-LoRA (L)	94.0	86.0	89.2	85.0	77.2	85.5	85.8	47.7	81.3
G_4-(P, L)	94.3	86.2	89.3	85.8	78.0	86.0	88.2	47.2	81.8
G_4-(A, P)	94.1	86.2	89.6	85.4	77.9	86.2	86.9	45.3	81.4
G_4-(A, L)	94.2	85.9	89.2	85.5	77.8	86.2	88.0	46.8	81.7
G_4-(A, P, L)	94.1	85.8	88.8	85.7	77.4	86.5	87.9	44.8	81.3
G_4-**(P, B, L)**	**94.6**	**86.4**	**90.4**	**86.1**	78.2	**86.8**	**88.5**	47.2	**82.3**
G_4-(A, P, B)	94.5	86.0	89.6	86.0	78.0	86.2	88.1	44.8	81.6
G_4-(A, B, L)	94.3	**86.4**	89.2	85.6	78.2	86.4	88.3	46.6	81.9
G_4-(A, P, B, L)	94.2	86.2	89.2	85.9	**78.5**	86.1	88.0	45.3	81.6

$$G_1\text{: (A, L)} — G_2\text{: (A, P)} — G_3\text{: (A, P, B)} — G_4\text{:(P, B, L)}$$

Adapter is more recommended in groups **closer to input,**
BitFit is more recommended in groups **closer to the output.**

Experiments

After Selecting Design Space

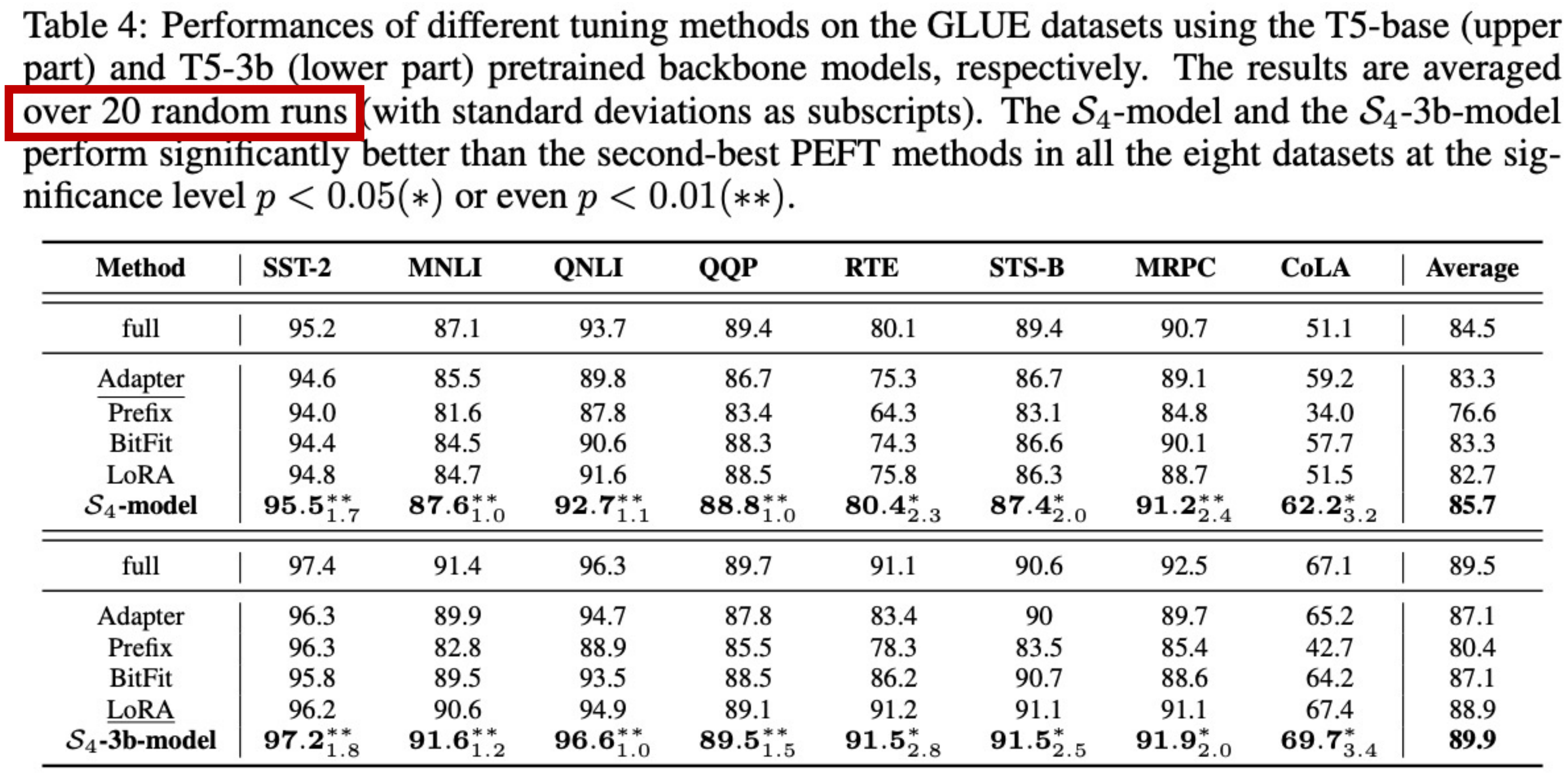


Table 4: Performances of different tuning methods on the GLUE datasets using the T5-base (upper part) and T5-3b (lower part) pretrained backbone models, respectively. The results are averaged over 20 random runs (with standard deviations as subscripts). The S_4-model and the S_4-3b-model perform significantly better than the second-best PEFT methods in all the eight datasets at the significance level $p < 0.05(*)$ or even $p < 0.01(**)$.

Method	SST-2	MNLI	QNLI	QQP	RTE	STS-B	MRPC	CoLA	Average
full	95.2	87.1	93.7	89.4	80.1	89.4	90.7	51.1	84.5
Adapter	94.6	85.5	89.8	86.7	75.3	86.7	89.1	59.2	83.3
Prefix	94.0	81.6	87.8	83.4	64.3	83.1	84.8	34.0	76.6
BitFit	94.4	84.5	90.6	88.3	74.3	86.6	90.1	57.7	83.3
LoRA	94.8	84.7	91.6	88.5	75.8	86.3	88.7	51.5	82.7
S_4-model	**$95.5^{**}_{1.7}$**	**$87.6^{**}_{1.0}$**	**$92.7^{**}_{1.1}$**	**$88.8^{**}_{1.0}$**	**$80.4^{*}_{2.3}$**	**$87.4^{*}_{2.0}$**	**$91.2^{**}_{2.4}$**	**$62.2^{*}_{3.2}$**	**85.7**
full	97.4	91.4	96.3	89.7	91.1	90.6	92.5	67.1	89.5
Adapter	96.3	89.9	94.7	87.8	83.4	90	89.7	65.2	87.1
Prefix	96.3	82.8	88.9	85.5	78.3	83.5	85.4	42.7	80.4
BitFit	95.8	89.5	93.5	88.5	86.2	90.7	88.6	64.2	87.1
LoRA	96.2	90.6	94.9	89.1	91.2	91.1	91.1	67.4	88.9
S_4-3b-model	**$97.2^{**}_{1.8}$**	**$91.6^{**}_{1.2}$**	**$96.6^{**}_{1.0}$**	**$89.5^{**}_{1.5}$**	**$91.5^{*}_{2.8}$**	**$91.5^{*}_{2.5}$**	**$91.9^{*}_{2.0}$**	**$69.7^{*}_{3.4}$**	**89.9**

Experiments

Design Space Ablation

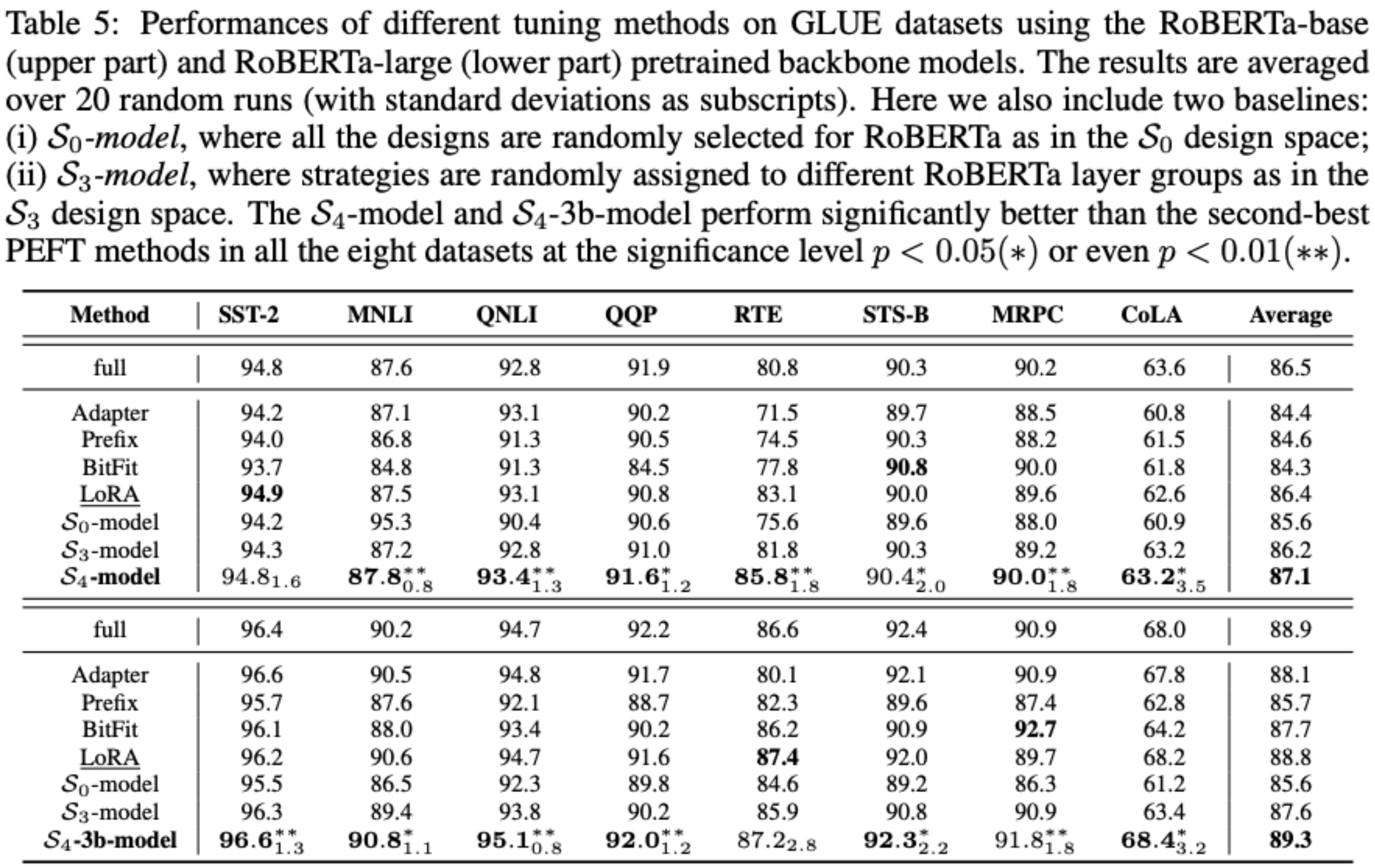


Table 5: Performances of different tuning methods on GLUE datasets using the RoBERTa-base (upper part) and RoBERTa-large (lower part) pretrained backbone models. The results are averaged over 20 random runs (with standard deviations as subscripts). Here we also include two baselines: (i) $\mathcal{S}_0$*-model*, where all the designs are randomly selected for RoBERTa as in the $\mathcal{S}_0$ design space; (ii) $\mathcal{S}_3$*-model*, where strategies are randomly assigned to different RoBERTa layer groups as in the $\mathcal{S}_3$ design space. The $\mathcal{S}_4$-model and $\mathcal{S}_4$-3b-model perform significantly better than the second-best PEFT methods in all the eight datasets at the significance level $p < 0.05(*)$ or even $p < 0.01(**)$.

Method	SST-2	MNLI	QNLI	QQP	RTE	STS-B	MRPC	CoLA	Average
full	94.8	87.6	92.8	91.9	80.8	90.3	90.2	63.6	86.5
Adapter	94.2	87.1	93.1	90.2	71.5	89.7	88.5	60.8	84.4
Prefix	94.0	86.8	91.3	90.5	74.5	90.3	88.2	61.5	84.6
BitFit	93.7	84.8	91.3	84.5	77.8	**90.8**	90.0	61.8	84.3
LoRA	**94.9**	87.5	93.1	90.8	83.1	90.0	89.6	62.6	86.4
$\mathcal{S}_0$-model	94.2	95.3	90.4	90.6	75.6	89.6	88.0	60.9	85.6
$\mathcal{S}_3$-model	94.3	87.2	92.8	91.0	81.8	90.3	89.2	63.2	86.2
$\mathcal{S}_4$**-model**	$94.8_{1.6}$	$\mathbf{87.8}^{**}_{0.8}$	$\mathbf{93.4}^{**}_{1.3}$	$\mathbf{91.6}^{*}_{1.2}$	$\mathbf{85.8}^{**}_{1.8}$	$90.4^{*}_{2.0}$	$\mathbf{90.0}^{**}_{1.8}$	$\mathbf{63.2}^{*}_{3.5}$	**87.1**
full	96.4	90.2	94.7	92.2	86.6	92.4	90.9	68.0	88.9
Adapter	96.6	90.5	94.8	91.7	80.1	92.1	90.9	67.8	88.1
Prefix	95.7	87.6	92.1	88.7	82.3	89.6	87.4	62.8	85.7
BitFit	96.1	88.0	93.4	90.2	86.2	90.9	**92.7**	64.2	87.7
LoRA	96.2	90.6	94.7	91.6	**87.4**	92.0	89.7	68.2	88.8
$\mathcal{S}_0$-model	95.5	86.5	92.3	89.8	84.6	89.2	86.3	61.2	85.6
$\mathcal{S}_3$-model	96.3	89.4	93.8	90.2	85.9	90.8	90.9	63.4	87.6
$\mathcal{S}_4$**-3b-model**	$\mathbf{96.6}^{**}_{1.3}$	$\mathbf{90.8}^{*}_{1.1}$	$\mathbf{95.1}^{**}_{0.8}$	$\mathbf{92.0}^{**}_{1.2}$	$87.2_{2.8}$	$\mathbf{92.3}^{*}_{2.2}$	$91.8^{**}_{1.8}$	$\mathbf{68.4}^{*}_{3.2}$	**89.3**

Main Focus

how to most efficiently use a **fixed budget to build a compact model.**

Distillation from LLM
(purchase or rent GPUs to distill LLM)

vs

Annotating more data by hiring annotators
(directly fine-tune a small model)

Main Focus

For smaller budgets

- Increase the amount of labeled data

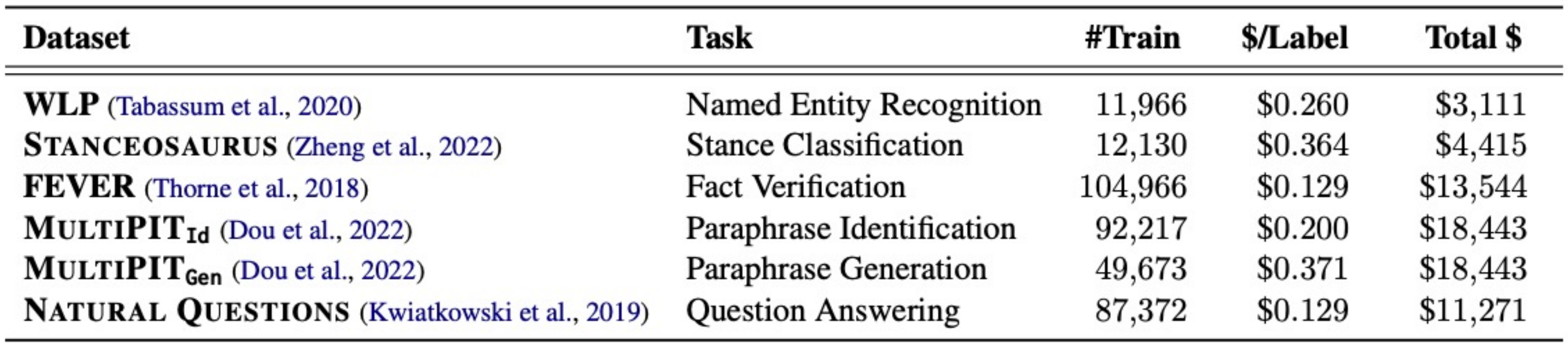
As the budget increases

- Distill using larger unlabeled datasets

Study Design

Cost for each Objective

Annotation Cost



Dataset	Task	#Train	$/Label	Total $
WLP (Tabassum et al., 2020)	Named Entity Recognition	11,966	$0.260	$3,111
STANCEOSAURUS (Zheng et al., 2022)	Stance Classification	12,130	$0.364	$4,415
FEVER (Thorne et al., 2018)	Fact Verification	104,966	$0.129	$13,544
MULTIPIT$_{\text{Id}}$ (Dou et al., 2022)	Paraphrase Identification	92,217	$0.200	$18,443
MULTIPIT$_{\text{Gen}}$ (Dou et al., 2022)	Paraphrase Generation	49,673	$0.371	$18,443
NATURAL QUESTIONS (Kwiatkowski et al., 2019)	Question Answering	87,372	$0.129	$11,271

Table 1: Data annotation costs for various NLP datasets/tasks.

Computational Cost

Google Cloud 기준

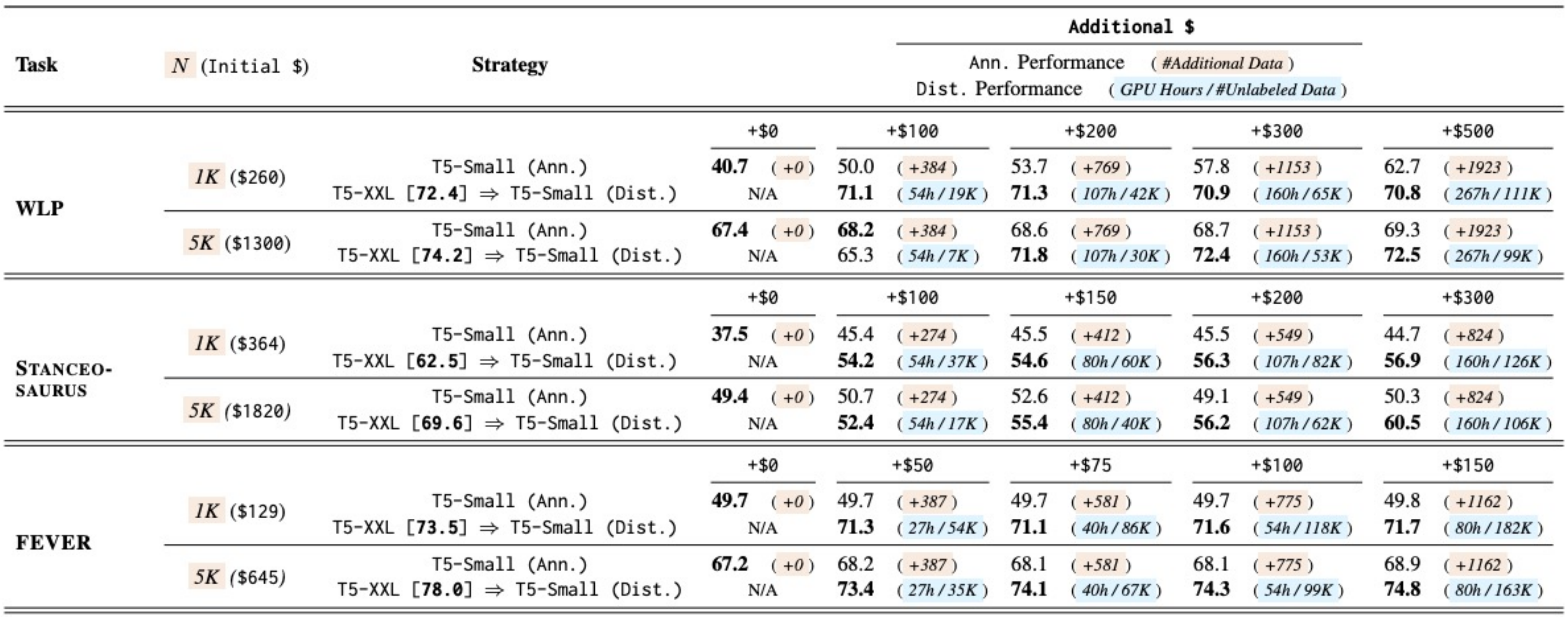
A100 (40GB of VRAM) :	$ 3.75 per 1 GPU hour
A40 (A100보다 2배 느림) :	$ 1.875 per 1 GPU hour

Student Model: T5-base
Fine-tuning T5-Small (60M) on 5K data
takes less than half an hour, which costs approximately $1

Experiments

소요 비용 기준 성능

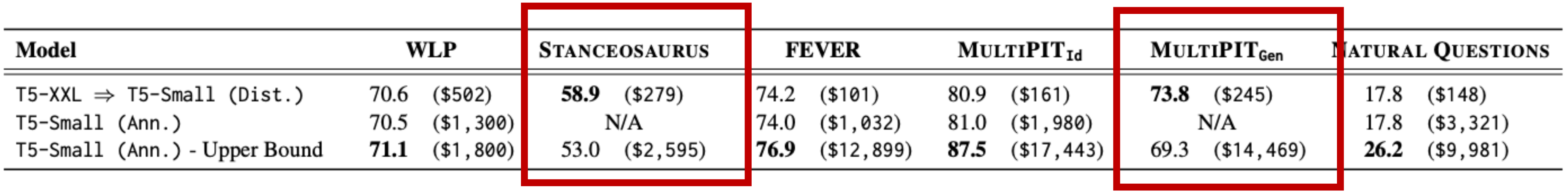
Distill or Annotate?
Cost-Efficient Fine-Tuning of Compact Models



Task	*N* (Initial $)	Strategy	Additional $				
			Ann. Performance (*#Additional Data*) / Dist. Performance (*GPU Hours / #Unlabeled Data*)				
WLP			+$0	+$100	+$200	+$300	+$500
	1K ($260)	T5-Small (Ann.)	**40.7** (*+0*)	50.0 (*+384*)	53.7 (*+769*)	57.8 (*+1153*)	62.7 (*+1923*)
		T5-XXL [**72.4**] ⇒ T5-Small (Dist.)	N/A	**71.1** (*54h / 19K*)	**71.3** (*107h / 42K*)	**70.9** (*160h / 65K*)	**70.8** (*267h / 111K*)
	5K ($1300)	T5-Small (Ann.)	**67.4** (*+0*)	**68.2** (*+384*)	68.6 (*+769*)	68.7 (*+1153*)	69.3 (*+1923*)
		T5-XXL [**74.2**] ⇒ T5-Small (Dist.)	N/A	65.3 (*54h / 7K*)	**71.8** (*107h / 30K*)	**72.4** (*160h / 53K*)	**72.5** (*267h / 99K*)
STANCEO-SAURUS			+$0	+$100	+$150	+$200	+$300
	1K ($364)	T5-Small (Ann.)	**37.5** (*+0*)	45.4 (*+274*)	45.5 (*+412*)	45.5 (*+549*)	44.7 (*+824*)
		T5-XXL [**62.5**] ⇒ T5-Small (Dist.)	N/A	**54.2** (*54h / 37K*)	**54.6** (*80h / 60K*)	**56.3** (*107h / 82K*)	**56.9** (*160h / 126K*)
	5K ($1820)	T5-Small (Ann.)	**49.4** (*+0*)	50.7 (*+274*)	52.6 (*+412*)	49.1 (*+549*)	50.3 (*+824*)
		T5-XXL [**69.6**] ⇒ T5-Small (Dist.)	N/A	**52.4** (*54h / 17K*)	**55.4** (*80h / 40K*)	**56.2** (*107h / 62K*)	**60.5** (*160h / 106K*)
FEVER			+$0	+$50	+$75	+$100	+$150
	1K ($129)	T5-Small (Ann.)	**49.7** (*+0*)	49.7 (*+387*)	49.7 (*+581*)	49.7 (*+775*)	49.8 (*+1162*)
		T5-XXL [**73.5**] ⇒ T5-Small (Dist.)	N/A	**71.3** (*27h / 54K*)	**71.1** (*40h / 86K*)	**71.6** (*54h / 118K*)	**71.7** (*80h / 182K*)
	5K ($645)	T5-Small (Ann.)	**67.2** (*+0*)	68.2 (*+387*)	68.1 (*+581*)	68.1 (*+775*)	68.9 (*+1162*)
		T5-XXL [**78.0**] ⇒ T5-Small (Dist.)	N/A	**73.4** (*27h / 35K*)	**74.1** (*40h / 67K*)	**74.3** (*54h / 99K*)	**74.8** (*80h / 163K*)

Experiments

소요 비용 기준 성능



Model	WLP		STANCEOSAURUS		FEVER		$\text{MULTIPIT}_{\text{Id}}$		$\text{MULTIPIT}_{\text{Gen}}$		NATURAL QUESTIONS	
T5-XXL ⇒ T5-Small (Dist.)	70.6	($502)	**58.9**	($279)	74.2	($101)	80.9	($161)	**73.8**	($245)	17.8	($148)
T5-Small (Ann.)	70.5	($1,300)	N/A		74.0	($1,032)	81.0	($1,980)	N/A		17.8	($3,321)
T5-Small (Ann.) - Upper Bound	**71.1**	($1,800)	53.0	($2,595)	**76.9**	($12,899)	**87.5**	($17,443)	69.3	($14,469)	**26.2**	($9,981)

Table 4: Performances along with (the corresponding budget) of Dist., Ann. that performs the same/similar to Dist., and Ann. upper bound by leveraging all existing annotated data. The best performance for each task is in bold.

Annotation data를 최대한 써도
distillation의 성능을 뛰어넘지 못하는 경우가 있더라

Annotation을 통해 Distillation의 성능을 뛰어넘을 수는 있지만,
너무 비싸다

Experiments

초기 데이터 영향

Distill or Annotate?
Cost-Efficient Fine-Tuning of Compact Models

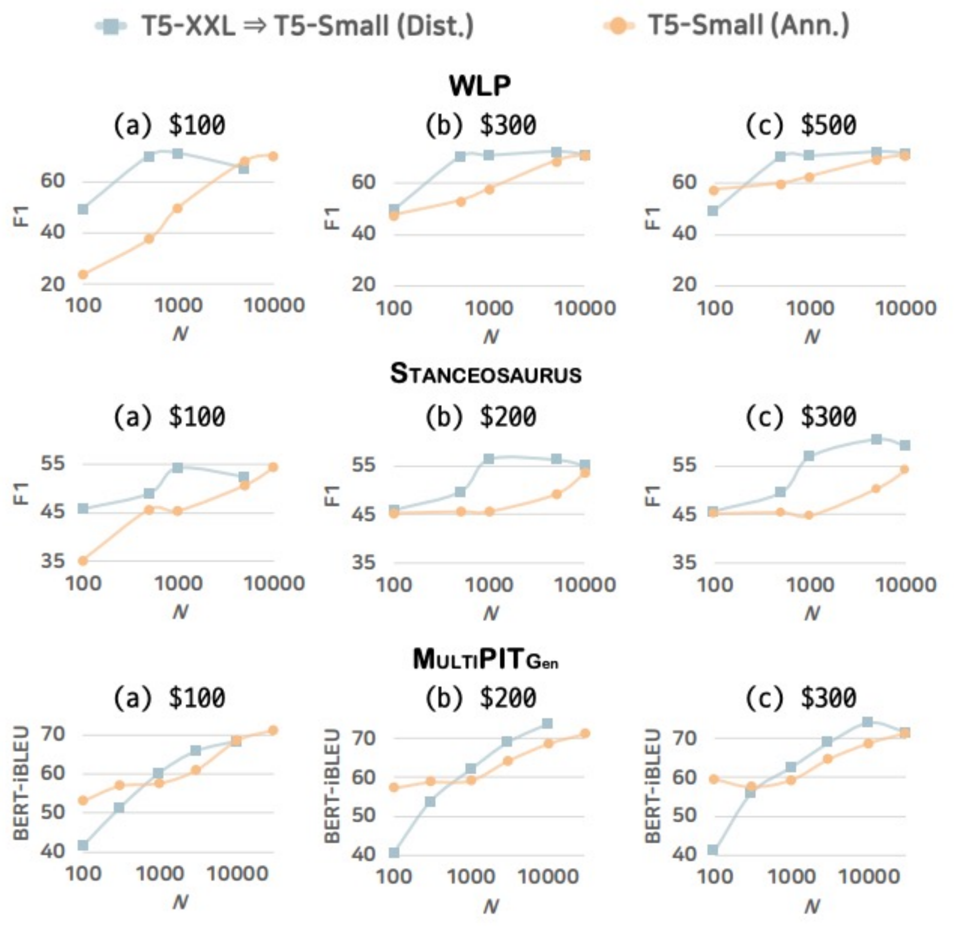


Figure 3: Results according to different number of starting annotated data (N) under fixed additional budgets.

초기 데이터가 극단적으로 적은 상황이 아닌 이상,

Distillation이 더 효율적이고 효과적인 방법이다.

Experiments

Model Variants

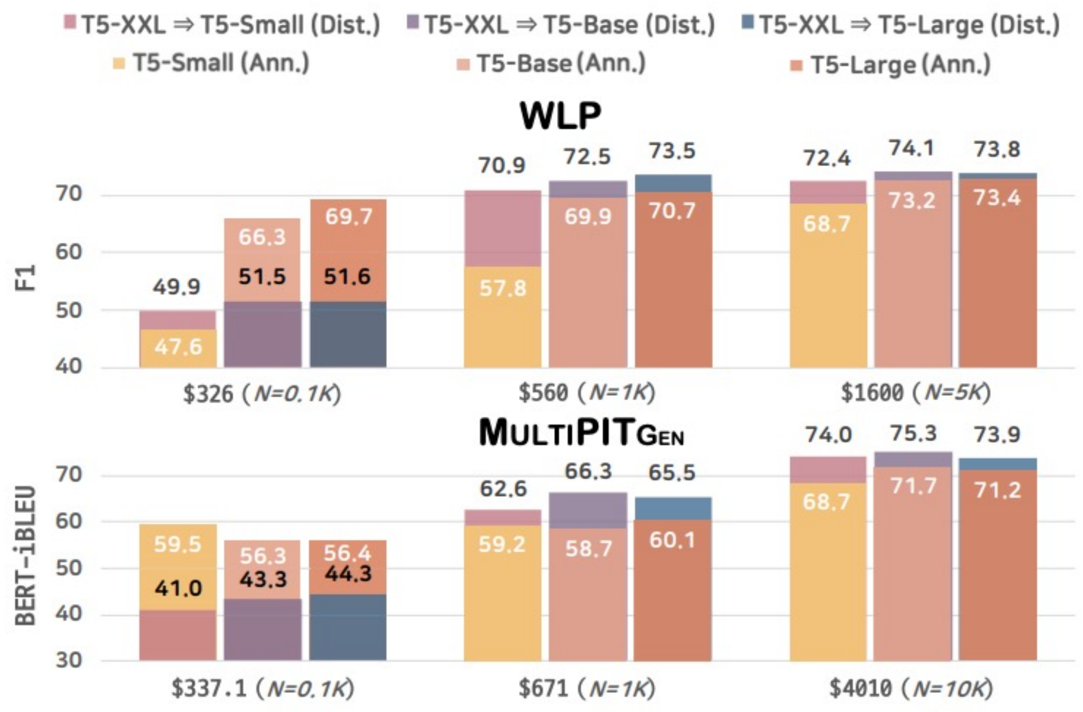


Figure 4: Results with different compact model sizes: Small (60M), Base (220M), Large (770M). For Dist., a teacher is fixed (XXL-11B), and the distillation cost is set to $300. Best viewed in color.

동일 budget기준

모델 사이즈에 상관없이
Distillation이 더 효과적이더라

더 작은 student model을
학습하는 경우일수록,
Distillation의 효과가 더 커지더라

Experiments

GPT3.5 as an Annotator

Distill or Annotate?
Cost-Efficient Fine-Tuning of Compact Models

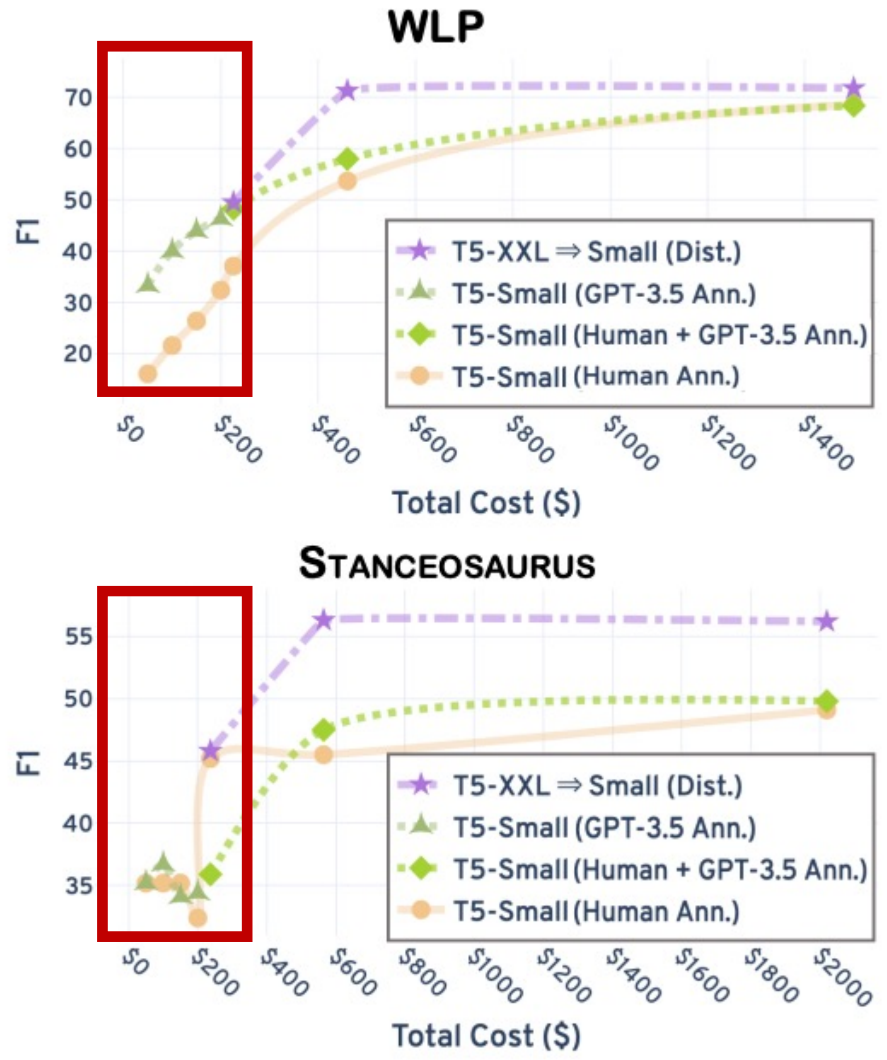


Figure 6: Comparisons with GPT-3.5 annotation. Given an initial human annotation N={$0.1K$, $1K$, $5K$} with the corresponding costs, $200 is additionally allocated for distillation or GPT-3.5 annotation (i.e., Human + GPT-3.5 Ann.).

**Data point 기준이 아니라,
Total budget을 기준으로 했을 때**

GPT3.5는 사람보다 더 효과적인
Annotator이다

단, 여전히 distillation보다는 성능이 낮다.

마치며..

Conclusion

LLM만 연구해야 하고 기존 연구들은 다 불필요할까? : X

- 단, LLM은 반드시 고려되어야 할 것
- Moderate PLM / LLM 모두 고려한다면.. PEFT

여전히 활발하게 연구되고 있음. 다양한 미개척분야

- 대부분의 NLP PEFT가 NLU에만 초점
- NLG를 위한 PEFT ?

감사합니다

Q&A