자연어 처리 모델의 양자화를 활용한 Memory-efficient finetuning

2023.04.25.

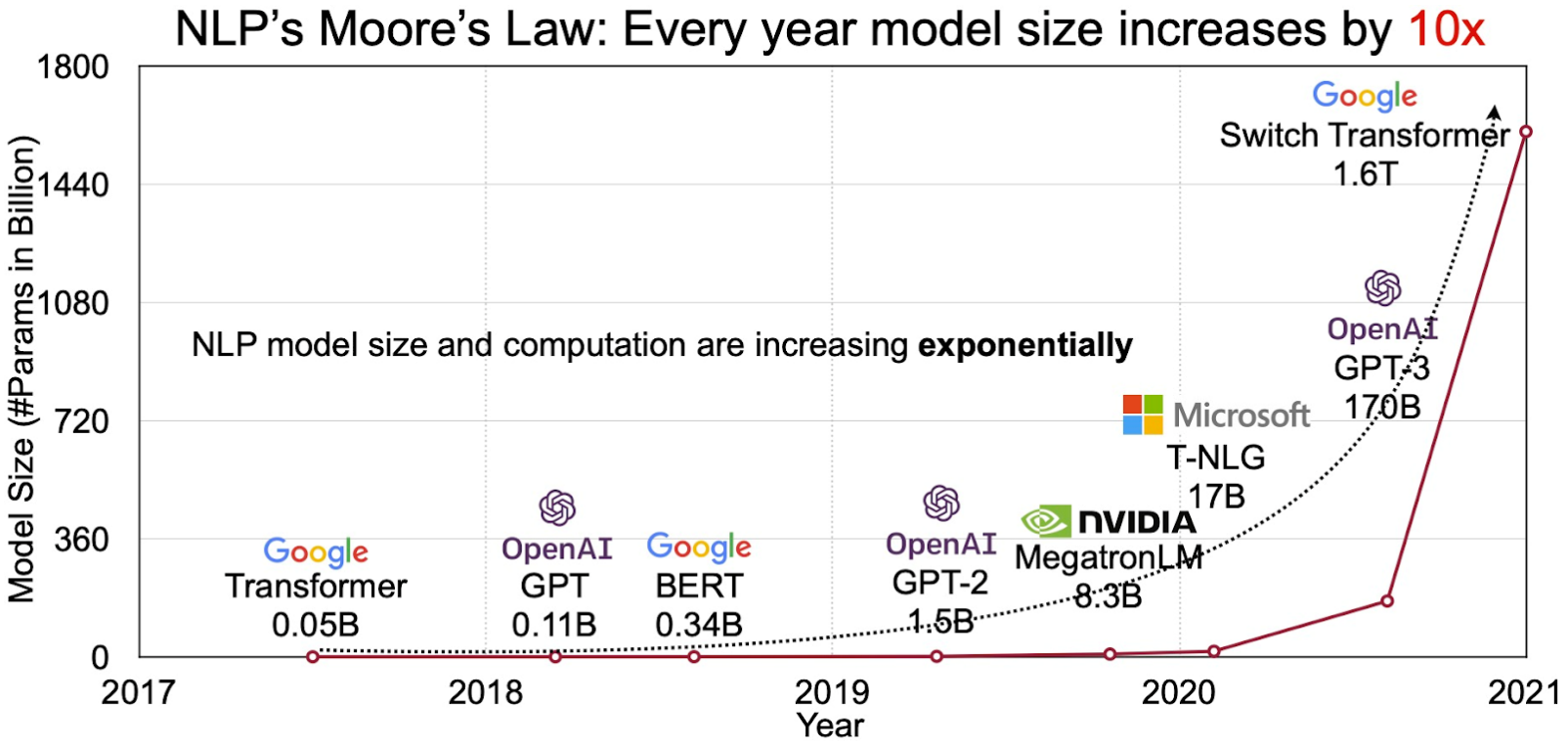
컴퓨터소프트웨어학부 이성진, 조한빛, 황태경

HANYANG UNIVERSITY

Contents

- Background
- Key idea
- Memory-efficient finetuning 평가를 위한 실험 계획
- Evaluation 결과
- 결론 및 시사점

Background: Large language model trends



- GPT-3를 비롯하여 Billion 단위의 파라미터를 가진 LLM이 널리 사용되는 상황
- 대규모의 고성능 인프라가 부족한 일반 개인들은 이러한 모델의 training과 inference를 수행하기 매우 어려움

Background: Power of medium-sized LLMs

Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality

by the Team with members from UC Berkeley, CMU, Stanford, and UC San Diego

* *According to a fun and non-scientific evaluation with GPT-4. Further rigorous evaluation is needed.*

We introduce Vicuna-13B, an open-source chatbot trained by fine-tuning LLaMA on user-shared conversations collected from ShareGPT. Preliminary evaluation using GPT-4 as a judge shows Vicuna-13B achieves more than 90%* quality of OpenAI ChatGPT and Google Bard while outperforming other models like LLaMA and Stanford Alpaca in more than 90%* of cases. The cost of training Vicuna-13B is around $300. The training and serving code, along with an online demo, are publicly available for non-commercial use.

Vicuna (generated by stable diffusion 2.1)

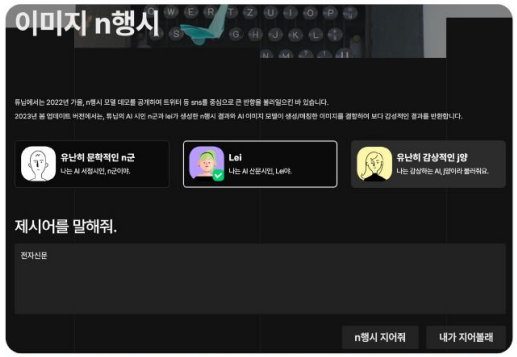
		0-shot	1-shot	5-shot	64-shot
GPT-3	175B	14.6	23.0	-	29.9
Gopher	280B	10.1	-	24.5	28.2
Chinchilla	70B	16.6	-	31.5	35.5
PaLM	8B	8.4	10.6	-	14.6
	62B	18.1	26.5	-	27.6
	540B	21.2	29.3	-	39.6
LLaMA	7B	16.8	18.7	22.0	26.1
	13B	20.1	23.4	28.1	31.9
	33B	**24.9**	28.3	32.9	36.0
	65B	23.8	**31.0**	**35.0**	**39.9**

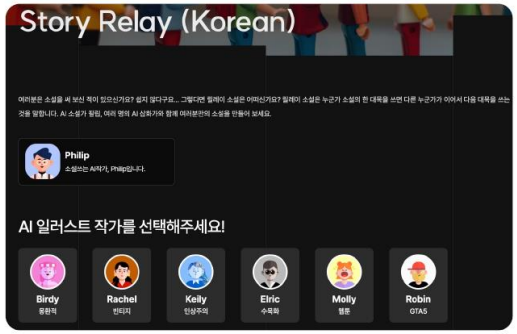
Table 4: **NaturalQuestions.** Exact match performance.

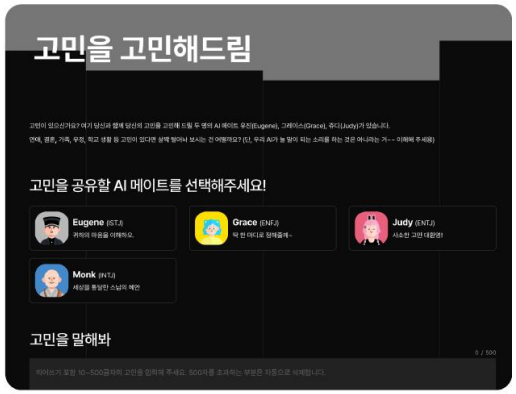
- 5B~20B 정도의 medium sized LM은 GPT-3와 같은 초거대 모델보다는 성능이 떨어지지만, 모델의 크기 대비 성능이 매우 높아 효율적
- Vicuna: Facebook의 Llama 13B LM을 튜닝한 모델로 ChatGPT의 90%에 가까운 성능을 보여줌

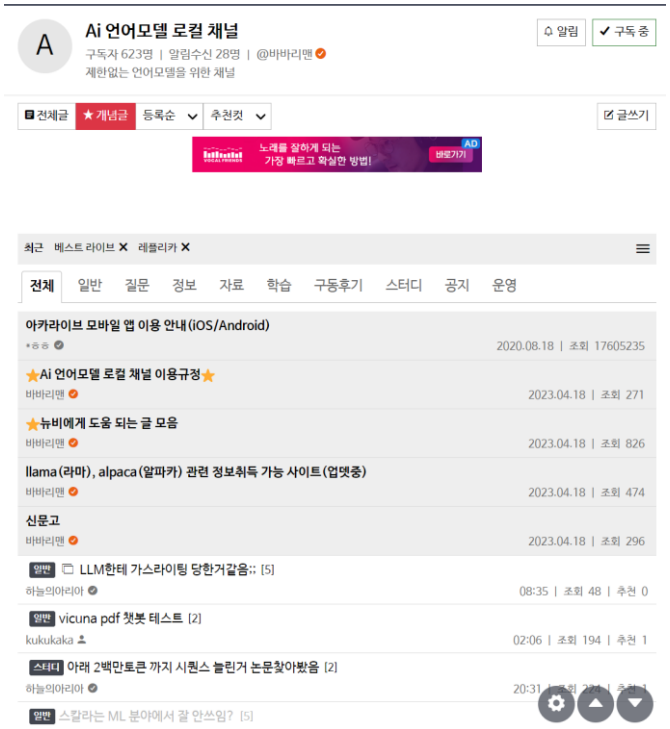
Background: General demand for fine-tuning LLMs

DearCreators, DearCoaches: TUNiB-GPT 6B *(Y23)*

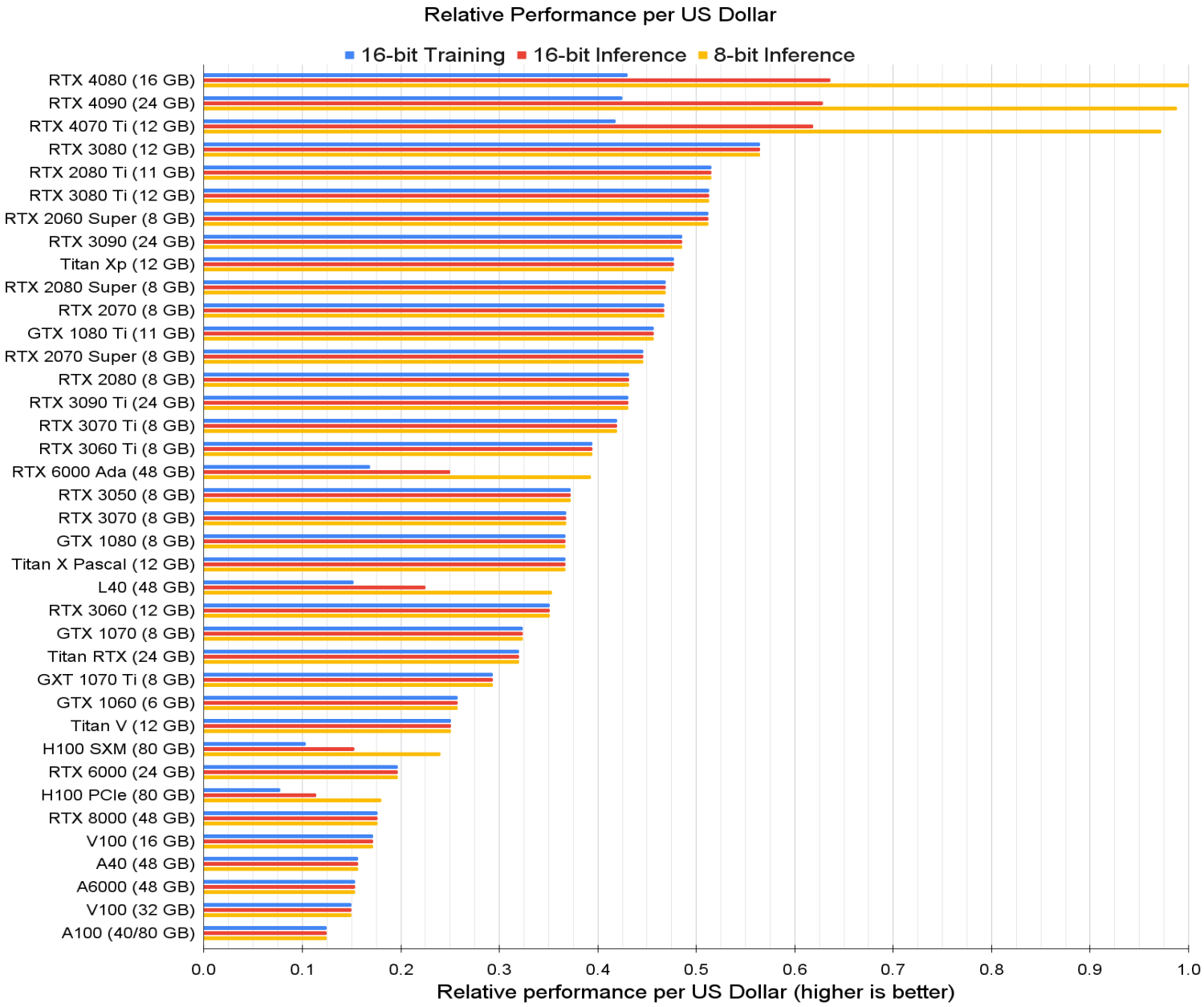




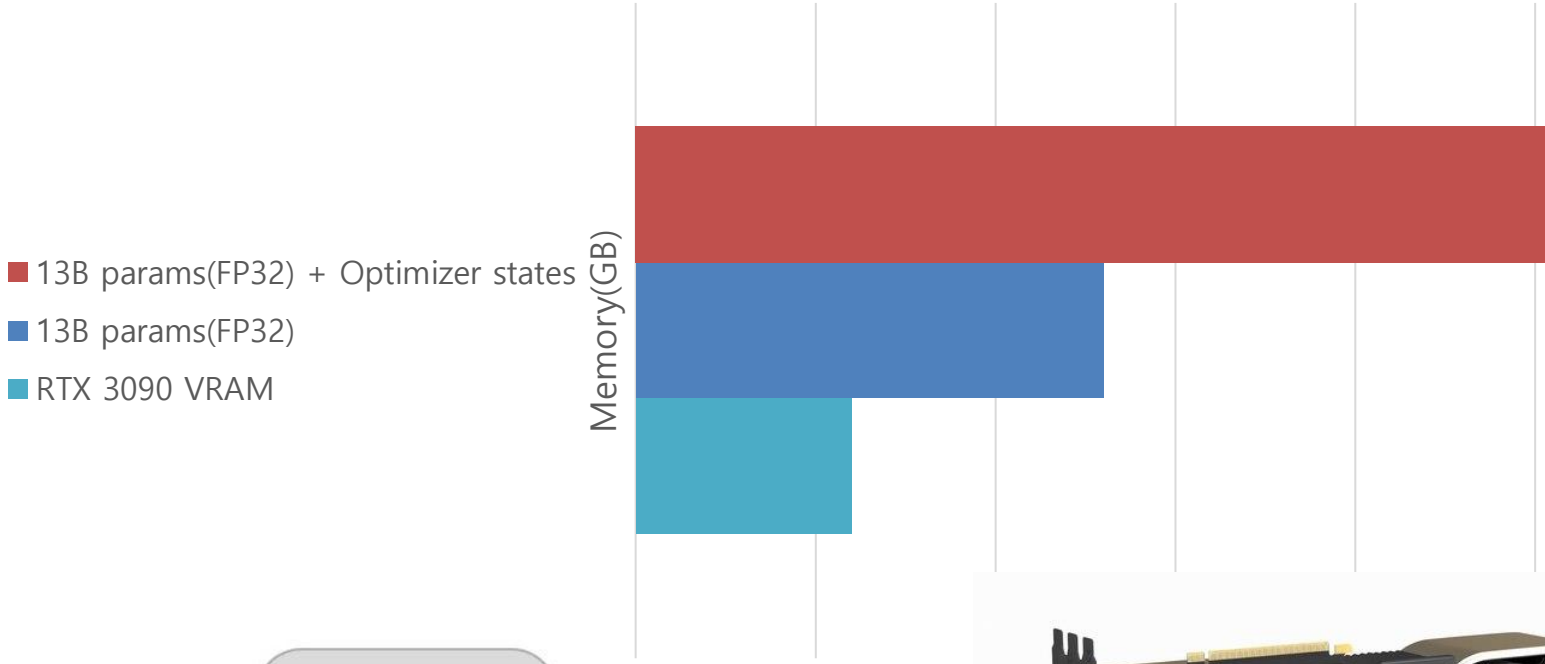




- GPT-3보다 훨씬 규모가 작으면서도, 특정 domain/task에 특화하여 튜닝하면 성능이 매우 뛰어난 middle-size LM을 튜닝하기 위한 수요가 매우 증가함(개인, 기업 등)
- 대규모 언어 모델들이 open-source로 공개되었음에도 불구하고, 일반 consumer device로 튜닝할 수 있는 모델의 사이즈엔 한계가 존재함



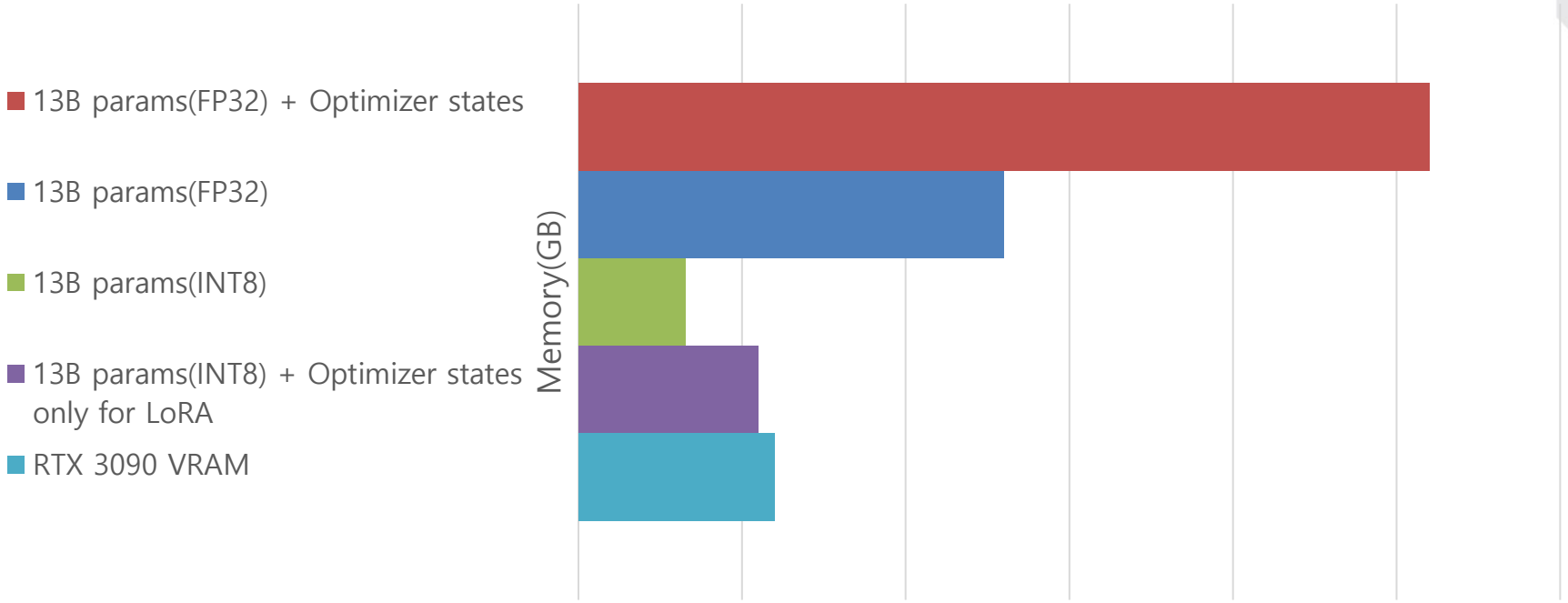
Difficulties of training: Insufficient memory



LLaMA 13B

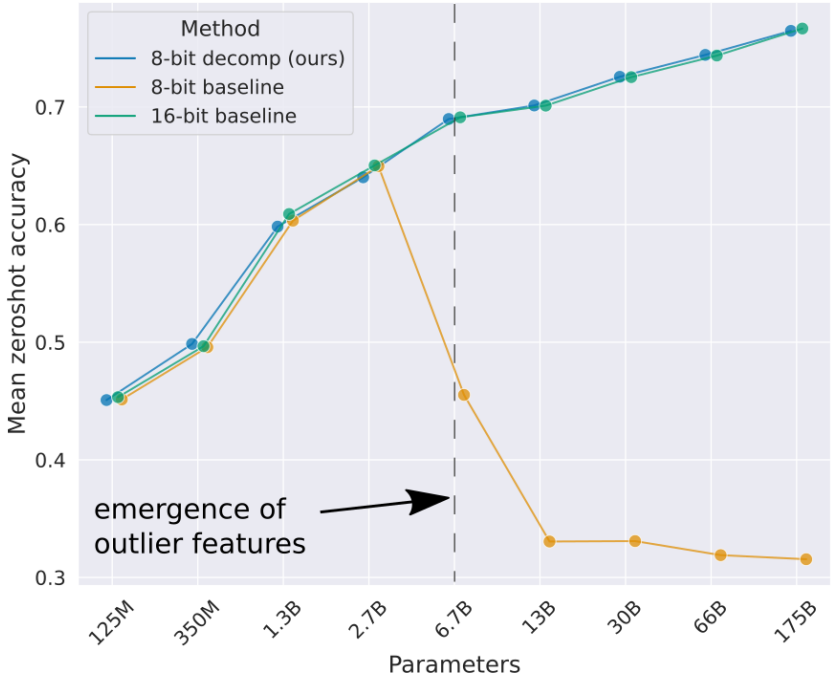
- Llama-13B: about 52GB size of weight parameters(FP32)
- Consumer GPU(RTX 3090): 24GB VRAM

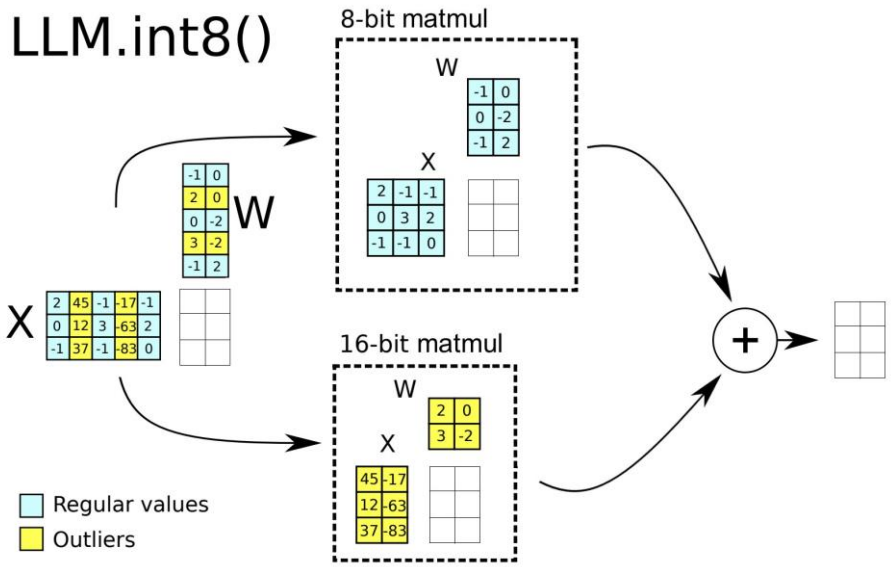
Key Idea: Quantization + Parameter efficient finetuning



- 기존 full-precision 모델에 quantization을 적용하여 pipeline parallel 등 방법을 사용하지 않고서도 일반 consumer용 GPU에 모델을 로드할 수 있도록 함
- LoRA와 같은 parameter efficient finetuning 기법을 사용하여, 학습 과정에 필요한 메모리를 더욱 더 절약 가능, full-finetuning 대비 빠른 학습속도
- 대규모 GPU 인프라를 보유하지 않은 개인도 10B+ 모델을 쉽게 학습할 수 있도록 하는 것이 목표

LLM.int8(): Outlier-aware quantization for LLMs





- 기존 static/dynamic quantization 방식의 단점: precision을 낮추는 과정에서 outlier에 대한 정보가 크게 손실됨. 이는 Billion scale 모델에서 더 치명적이어서 quantization을 적용하는데 큰 문제가 됨
- LLM.int8()은 matrix multiplication을 수행할 때 mixed precision decomposition을 수행하여 outlier에 대한 정보가 손실되지 않도록 하면서 memory-efficient한 inference가 가능해짐

Finetuning LMs with LLM.int8(), not only inference!

6 Discussion and Limitations

We have demonstrated for the first time that multi-billion parameter transformers can be quantized to Int8 and used immediately for inference without performance degradation. We achieve this by using our insights from analyzing emergent large magnitude features at scale to develop mixed-precision decomposition to isolate outlier features in a separate 16-bit matrix multiplication. In conjunction with vector-wise quantization that yields our method, LLM.int8(), which we show empirically can recover the full inference performance of models with up to 175B parameters.

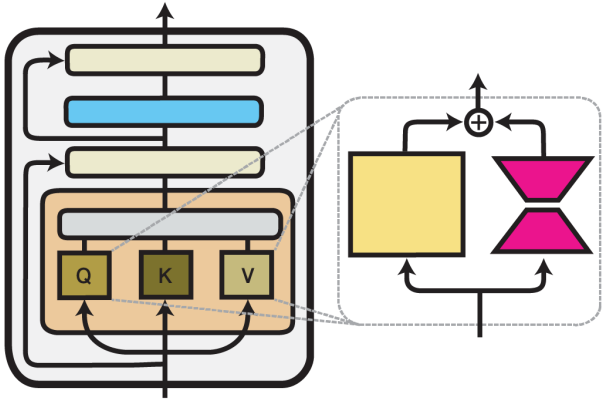
A final limitation is that we focus on inference but do not study training or finetuning. We provide an initial analysis of Int8 finetuning and training at scale in Appendix E. Int8 training at scale requires complex trade-offs between quantization precision, training speed, and engineering complexity and represents a very difficult problem. We again leave this to future work.

- LLM.int8() quantization을 적용하면 memory-efficient한 inference가 가능하면서도 높은 precision(FP16)에 비해 거의 정확도가 손실되지 않음
- 효율적인 inference를 위해 quantization을 적용했던 기존 방식과 달리, **낮은 parameter precision에도 불구**하고 높은 정확도를 보여주는 점을 활용해서 **모델 finetuning에 적용**하고자 함

LoRA: parameter-efficient finetuning

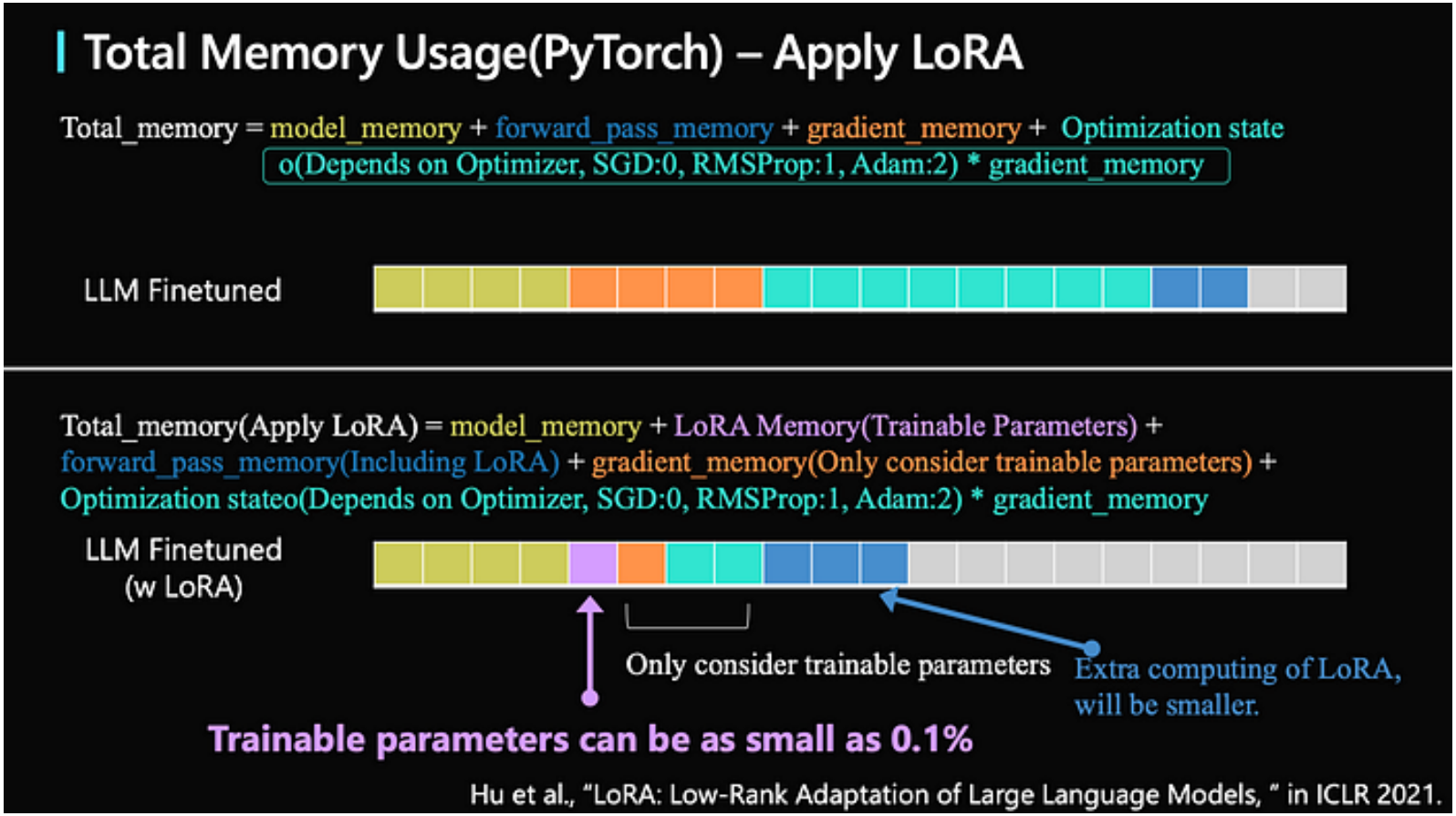
Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} (Adpt^{D})*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB_{base} (Adpt^{D})*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	**87.2**
$\text{RoB}_{\text{large}}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
$\text{RoB}_{\text{large}}$ (LoRA)	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.9}_{\pm1.2}$	$\mathbf{68.2}_{\pm1.9}$	$\mathbf{94.9}_{\pm.3}$	$91.6_{\pm.1}$	$\mathbf{87.4}_{\pm2.5}$	$\mathbf{92.6}_{\pm.2}$	**89.0**
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	$\mathbf{68.3}_{\pm1.0}$	$\mathbf{94.8}_{\pm.2}$	$\mathbf{91.9}_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	0.8M	$\mathbf{90.5}_{\pm.3}$	$\mathbf{96.6}_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	$\mathbf{94.8}_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	87.9
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	6.0M	$89.9_{\pm.5}$	$96.2_{\pm.3}$	$88.7_{\pm2.9}$	$66.5_{\pm4.4}$	$94.7_{\pm.2}$	$92.1_{\pm.1}$	$83.4_{\pm1.1}$	$91.0_{\pm1.7}$	87.8
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	0.8M	$90.3_{\pm.3}$	$96.3_{\pm.5}$	$87.7_{\pm1.7}$	$66.3_{\pm2.0}$	$94.7_{\pm.2}$	$91.5_{\pm.1}$	$72.9_{\pm2.9}$	$91.5_{\pm.5}$	86.4
$\text{RoB}_{\text{large}}$ (LoRA)†	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.2}_{\pm1.0}$	$68.2_{\pm1.9}$	$\mathbf{94.8}_{\pm.3}$	$91.6_{\pm.2}$	$\mathbf{85.2}_{\pm1.1}$	$\mathbf{92.3}_{\pm.5}$	**88.6**
DeB_{XXL} (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	$\mathbf{91.9}_{\pm.2}$	$96.9_{\pm.2}$	$\mathbf{92.6}_{\pm.6}$	$\mathbf{72.4}_{\pm1.1}$	$\mathbf{96.0}_{\pm.1}$	$\mathbf{92.9}_{\pm.1}$	$\mathbf{94.9}_{\pm.4}$	$\mathbf{93.0}_{\pm.2}$	**91.3**

Table 2: $\text{RoBERTa}_{\text{base}}$, $\text{RoBERTa}_{\text{large}}$, and $\text{DeBERTa}_{\text{XXL}}$ with different adaptation methods on the GLUE benchmark. We report the overall (matched and mismatched) accuracy for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. Higher is better for all metrics. * indicates numbers published in prior works. † indicates runs configured in a setup similar to Houlsby et al. (2019) for a fair comparison.



- Transformer 모델의 self attention을 계산하는 부분에서, 기존 layer는 gradient가 업데이트되지 않도록 weight를 freeze하고, adapter만 튜닝하는 방법
- Updated 해야 하는 파라미터의 수가 전체의 매우 적은 부분이기 때문에, backward pass에 필요한 memory 사용량이 크게 줄어들면서 학습 task에 대한 성능이 full-finetuning과 큰 차이가 없음

LoRA의 메모리 사용량 절약 효과



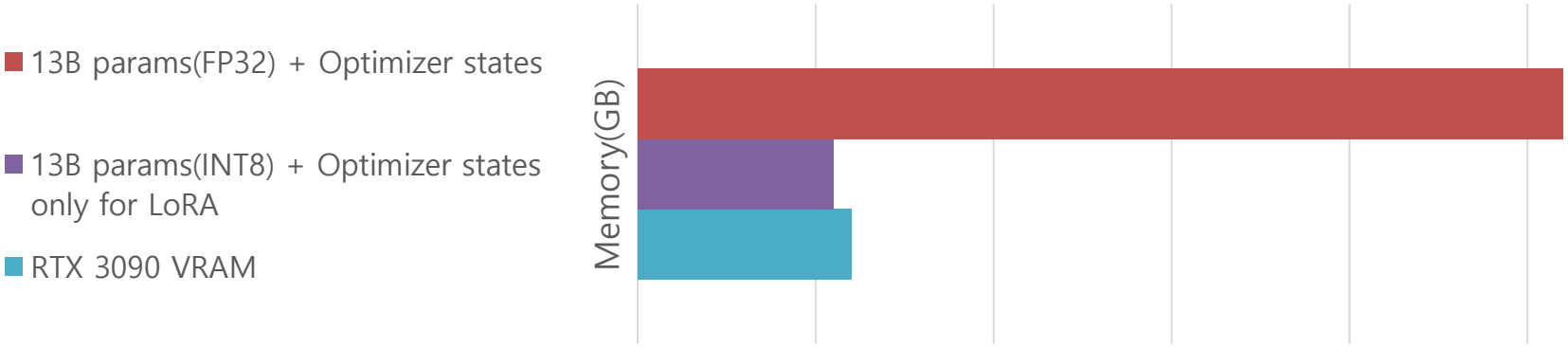
Quantization + LoRA의 메모리 절약 효과

Total_memory = model_memory + forward_pass_memory + gradient_memory + Optimization state
o(Depends on Optimizer, SGD:0, RMSProp:1, Adam:2) * gradient_memory

Finetune LLM

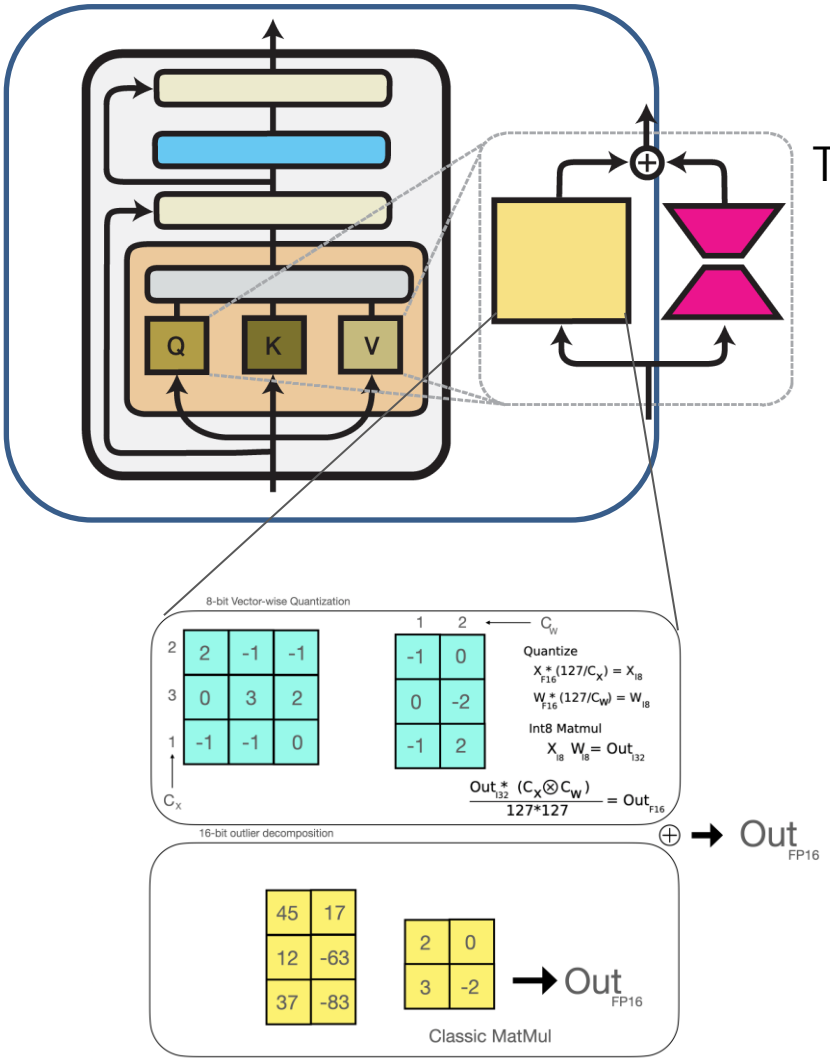
Total_memory(Apply LoRA) = model_memory + LoRA Memory(Trainable Parameters) +
forward_pass_memory(Including LoRA) + gradient_memory(Only consider trainable parameters) +
Optimization stateo(Depends on Optimizer, SGD:0, RMSProp:1, Adam:2) * gradient_memory

Apply LoRA
Apply LoRA
+ INT8 quant



LLM.int8() + LoRA

Do not update backbone model's parameters



Train LoRA adapter only

1. Backbone 모델의 weight를 INT8로 quantization을 적용하여 로드한다.
2. Backbone 모델의 weight의 gradient를 계산하지 않도록 freeze한다.
3. Attention q, k를 계산하는 linear layer에 LoRA 어댑터를 적용한다.
4. LoRA 어댑터에 대해서만 gradient descent를 적용하여 파라미터를 업데이트한다.

Memory-efficient finetuning의 평가를 위한 실험 계획

실험 목표: 고성능 인프라에서 **FP16 precision으로 모델을 LoRA 튜닝**했을 때와, **Quantization을 적용한 상태에서 LoRA 튜닝**을 했을 때의 **task 성능, 학습 시간, 메모리 사용량 등 비교**

- 실험 1: **같은 device**, **같은 batch size**에서 **quantization 적용 여부**에 따른 task 성능 비교
  - 사용된 device: RTX 3090 24GB
  - 학습에 사용된 모델: Facebook-**OPT-1.3B**
  - NLU task: BoolQ, MRPC, HellaSwag, SAMSum
- 실험 2: **같은 device**, **quantization 적용에 따라 다른 batch size** 사용하여 task 성능 비교
  - 사용된 device: RTX 3090 24GB
  - 학습에 사용된 모델: Facebook-**OPT-1.3B**
  - Train batch size: **64**(FP16), **128**(INT8)
  - NLU task: MNLI
- 실험 3: **다른 device** 환경, **같은 batch size**에서 **quantization 적용 여부**에 따른 task 성능 비교
  - 사용된 device: **RTX A6000 48GB**(FP16), **RTX 3090 24GB**(INT8)
  - 학습에 사용된 모델: Facebok-**OPT-13B**
  - NLU task: SAMSum

실험 1 Evaluation: Quantization 적용 여부에 따른 실험 결과

같은 device, 같은 batch size에서 **quantization 적용 여부**에 따른 task 성능 비교

- 사용된 device: RTX 3090 24GB
- 학습에 사용된 모델: Facebook-**OPT-1.3B**

Task	Method	Accuracy	f1-score	Training time	Memory allocated
MRPC	FP16 + LoRA (baseline)	0.8358	0.8878	1.00x (18m 37s)	1.00x
	INT8 + LoRA	**0.8382**	**0.8896**	**1.46x (27m 9s)**	**0.86x**
BOOLQ	FP16 + LoRA (baseline)	**0.8242**	**0.8583**	1.00x (3h 45m 44s)	1.00x
	INT8 + LoRA	0.8226	0.8571	**1.28x (4h 47m 54s)**	**0.97x**
HallaSWAG	FP16 + LoRA (baseline)	0.7879	0.7878	1.00x (4h 48m 52s)	1.00x
	INT8 + LoRA	**0.7901**	**0.7901**	**1.32x (6h 22m 26s)**	**0.83x**

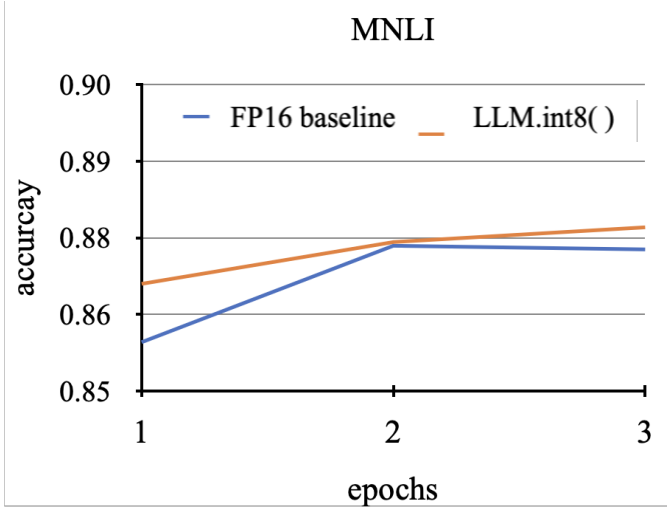
Task	Method	Rouge-1	Rouge-2	Rouge-L	Training time	Memory allocated
SAMSum	FP16 + LoRA (baseline)	0.5014	0.2560	0.4140	1.00x (9h 12m 15s)	1.00x
	INT8 + LoRA	**0.5018**	**0.2585**	**0.4187**	**1.23x (11h 20m 35s)**	**0.85x**

실험 2 Evaluation: Quantization 적용 여부에 따른 batch size 변화

같은 device, quantization 적용에 따라 다른 batch size 사용하여 task 성능 비교

- 사용된 device: RTX 3090 24GB
- 학습에 사용된 모델: Facebook-**OPT-1.3B**
- Train batch size: **64**(FP16), **128**(INT8)

Task	Method	Batch size	Accuracy	F1 score	Training time	Memory allocated
MNLI	FP16 + LoRA (baseline)	64	0.8731	0.8725	1.00x (9h 43m 41s)	1.00x
	INT8 + LoRA	128	**0.8767**	**0.8761**	**1.47x (14h 16m 29s)**	**0.85x**



- FP16으로 학습 시 batch size 128을 사용할 경우 **OOM**이 발생하였으나, IN8 모델을 사용한 결과 model weight를 저장하기 위한 메모리가 절약되어 더 큰 batch size를 사용할 수 있었음

실험 3 Evaluation:
고성능 GPU vs 저사양 GPU 튜닝 결과 비교

다른 device 환경에서 **quantization 적용 여부**에 따른 task 성능 비교

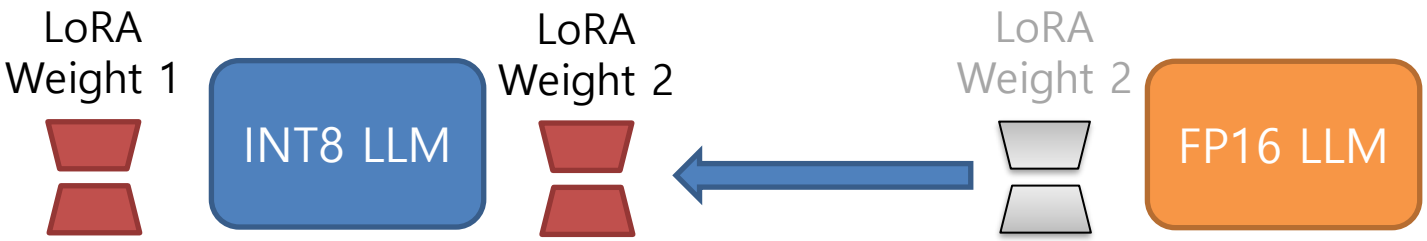
- 사용된 device: **RTX A6000 48GB**(FP16) vs **RTX 3090 24GB**(INT8)
- 학습에 사용된 모델: Facebok-**OPT-13B**

Task	Method	Hardware	Rouge-1	Rouge-2	Rouge-L	Training time	Memory allocated
SAMSum	FP16 + LoRA (baseline)	RTX A6000 48GB	0.5150	0.2799	0.4336	1.00x (14h 53m 33s)	1.00x (31.02GB)
	INT8 + LoRA	RTX 3090 24GB	**0.5232**	**0.2829**	**0.4388**	**1.62x (24h 4m 9s)**	**0.70x (21.60GB)**

- RTX A6000 48GB는 model parallel을 사용하지 않고서도 FP16 precision으로 13B 모델을 finetuning할 수 있으나, RTX 3090 24GB는 OOM으로 인해 불가능함
- Quantization과 LoRA를 적용하여, finetuning에 소모되는 메모리양을 절약한 결과 RTX 3090 24GB 1개에서 13B 모델 finetuning이 가능했으며, FP16으로 튜닝한 모델에 비하여 NLU task의 성능이 비슷하거나 더 좋은 결과를 보임
- **고성능 인프라를 갖추지 않아도** 파라미터 개수가 많은 **대규모 언어 모델을 정확도 손실 거의 없이 finetuning 가능**하도록 한다는 점에서 의의가 있음

같은 backbone에서 서로 다른 환경에서 학습된 LoRA 어댑터 사용

서로 다른 환경(Quantization 적용 여부/GPU device 등)에서 튜닝된 LoRA 어댑터를 같은 backbone 모델에 연결해서 사용했을 때의 task 성능 평가



Backbone Model	LoRA weight	SAMSum		
		Rouge-1	Rouge-2	Rouge-L
OPT-13B(INT8)	Weight 1(8bit trained)	0.5232	0.2828	0.4388
OPT-13B(FP16)	Weight 2(16bit trained)	0.5205	0.2868	0.4388
OPT-13B(INT8)	Weight 2(16bit trained	0.5230	0.2892	0.4411

결론 및 시사점

- 대규모의 고성능 GPU 인프라 및 model/data parallel을 수행하기 위한 엔지니어링 기술이 없어도 Billion scale의 대형 언어 모델을 consumer GPU에서 튜닝 가능

- 상대적으로 큰 모델에서 성능이 좋지 않은 QAT에 비해, quantization과 LoRA를 적용한 튜닝 방식은 높은 precision으로 튜닝한 것과 비교해도 정확도가 거의 떨어지지 않았음

- Backbone 모델의 파라미터를 업데이트하지 않고 LoRA weight만 학습했기 때문에, 학습이 완료된 adapter 결과물을 backbone 모델의 quantization 적용 여부와 관계없이 사용 가능

- 각 실험별로 평균적으로 FP16으로 학습하는 것에 비해 약 39%정도 시간을 더 소모하는 대신 정확도 손실 없이 task에 대한 모델의 finetuning을 진행할 수 있었음

- 주로 Vision/NLP 모델의 효율적인 inference에 사용되었던 기존의 quantization 방식과 다르게 outlier를 고려하여 정확도 손실을 방지한 LLM.int8()과 LoRA를 함께 사용하여 두 방법의 장점을 모두 사용할 수 있었음

End of document

감사합니다