파운데이션 모델의 미세 조정:
특정 작업을 위한 모델 맞춤화

2025-04-12

송태영

미세 조정 소개

- 미세 조정은 **사전 훈련된 모델의 전체 또는 일부를 추가로 훈련하여 특정 작업에 적합하도록 조정하는 과정**입니다
- **프롬프트 기반 방법**은 지시, 컨텍스트, 도구를 제공하여 모델을 조정하는 반면, **미세 조정은 모델의 가중치를 직접 변경**합니다
- 미세 조정은 모델의 **도메인별 능력 향상 (코딩, 의료 질의응답 등)**, **안전성 강화**, 특히 **특정 출력 스타일 및 형식 준수를 위한 명령 추종 능력 향상**에 사용됩니다
- 미세 조정은 요구 사항에 더 **맞춤화된 모델을 만드는 데 도움**을 줄 수 있지만, **더 많은 초기 투자**가 필요합니다

미세 조정 vs. 프롬프트 기반 방법

특징	미세 조정 (Finetuning)	프롬프트 기반 방법 (Prompt-based Methods)
접근 방식	**모델의 가중치 조정**	지시, 컨텍스트, 도구 등을 제공하여 **모델 활용**
자원 요구량	**더 많은 데이터, 하드웨어, ML 전문 인력 필요**	비교적 적은 자원 필요
적용 시점	일반적으로 프롬프트 기반 방법 실험 후 성능 부족 시 시도.	모델 활용의 초기 단계에서 시도
목표	모델의 내부 능력 자체를 특정 작업에 **최적화**	주어진 프롬프트에 따라 원하는 응답 **생성 유도**
상호 배타성	상호 배타적이지 않으며, 함께 사용하여 성능을 극대화할 수 있음	

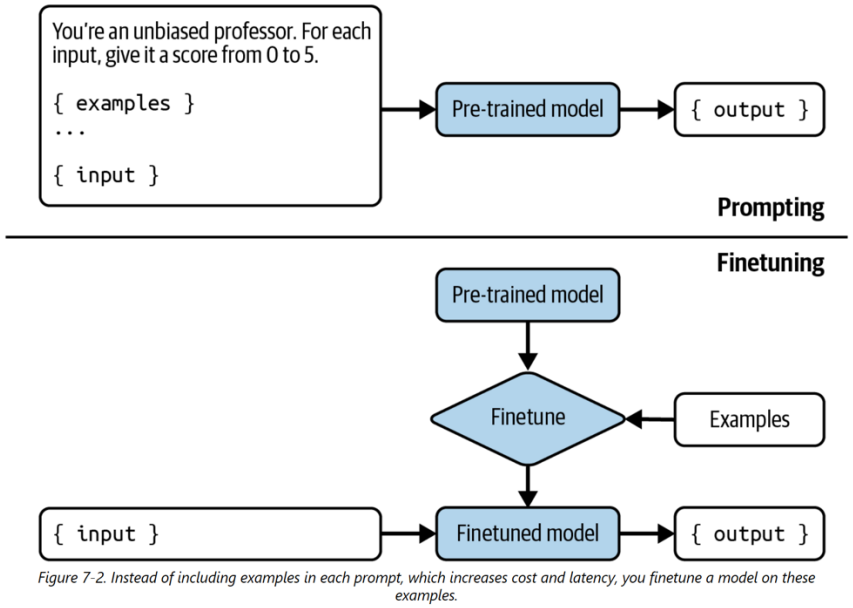


Figure 7-2. Instead of including examples in each prompt, which increases cost and latency, you finetune a model on these examples.

미세 조정 vs. RAG (Retrieval-Augmented Generation)

- **모델 실패 원인 분석**:
  - 정보 부족 (RAG) vs. 행동 문제 (미세 조정)
- **RAG**:
  - **최신 정보**가 필요한 작업에 **미세 조정보다 뛰어난 성능**을 보입니다
  - **정보 기반 오류 (factually wrong or outdated)**를 해결합니다
  - 구현이 일반적으로 더 쉽습니다
- **미세 조정**:
  - **특정 출력 형식 및 스타일**을 따르도록 모델을 훈련시키는 데 유용합니다
  - **행동 문제 (부적절하거나 형식에 맞지 않는 응답)**를 해결하는 데 도움이 됩니다
- **상호 보완적 사용**:
  - RAG와 미세 조정을 **결합하여** 애플리케이션 성능을 극대화할 수 있습니다

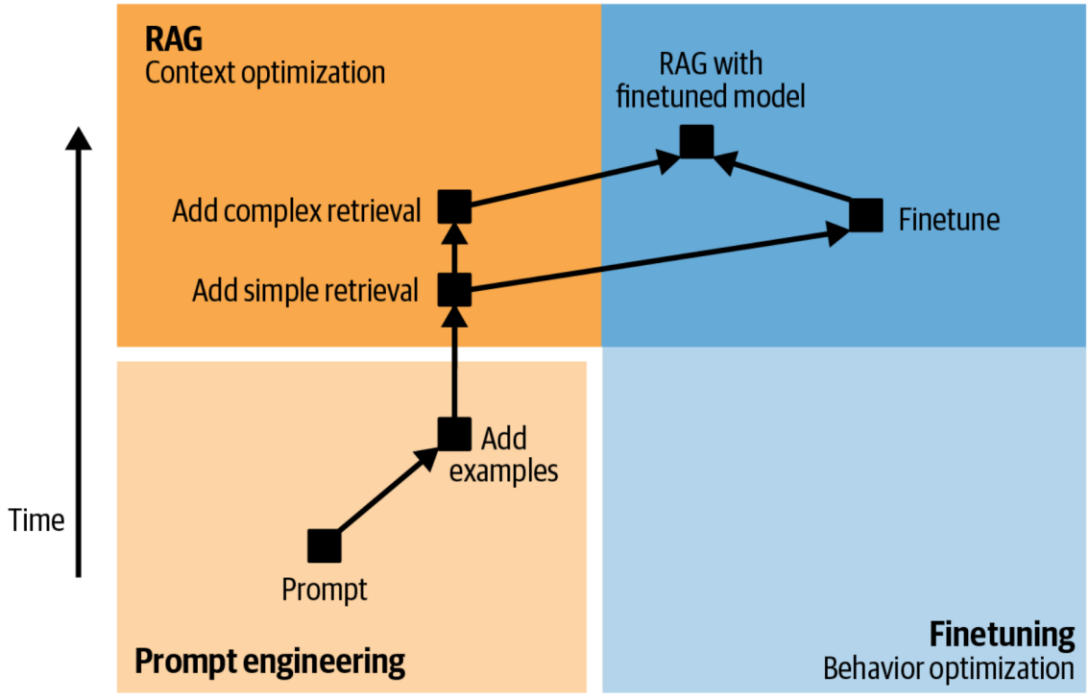


Figure 7-3. Example application development flows. After simple retrieval (such as term-based retrieval), whether to experiment with more complex retrieval (such as hybrid search) or finetuning depends on each application and its failure modes.

메모리 병목현상 (Memory Bottlenecks)

- **미세 조정은 메모리 집약적인 작업**입니다
- 기반 모델의 규모로 인해 **메모리는 추론과 미세 조정 모두에서 병목 현상**이 될 수 있지만, **미세 조정에 필요한 메모리가 일반적으로 훨씬 더 높습니다**
- 미세 조정 중 모델의 메모리 사용 공간에 **주요 기여 요인**은 다음과 같습니다
  - **모델의 총 파라미터 수**
  - **훈련 가능한 파라미터 수**: 많을수록 메모리 사용량 증가
  - **수치 표현 방식**: 낮은 정밀도 형식 (양자화)은 메모리 사용량 감소
  - **기울기 및 옵티마이저 상태**
  - **활성화 (Activations)**: 훈련 중 기울기 계산을 위해 저장될 수 있으며 상당한 메모리를 차지할 수 있습니다

모델 메모리 계산

추론 시 메모리

- **필요 요소**: 모델 가중치 + 활성화 + 키-값 벡터
- **계산식**: 총 메모리 $= N \times M \times 1.2$
  - N : 파라미터 수 (예: 13B = 130억)
  - M : 파라미터 당 메모리 (예: 2바이트)
- **예**: 13B 모델 → 26GB × 1.2 = **31.2GB**

훈련 시 메모리

- **필요 요소**: 모델 가중치 + 활성화 + 기울기 + 옵티마이저 상태
- **계산식** (Adam 기준): 메모리 $= N \times 3 \times M$
- 예: 13B 모델 → $13B \times 3 \times 2 =$**79GB**

수치 표현 (Numerical Representations)

- 모델의 각 값을 표현하는 데 필요한 메모리는 모델 전체 메모리 사용량에 직접적인 영향을 미칩니다

형식	비트	설명
FP32	32	단정밀도, 딥러닝 기본 형식
FP16	16	반정밀도, 메모리 절약용
BF16	16	TPU 최적화, 범위↑ 정밀도↓
TF32	19	NVIDIA GPU 최적화, 성능/정밀도 균형
INT8	8	정수 기반, 양자화 모델에 사용
INT4	4	극한 압축, 성능 유지 어려움

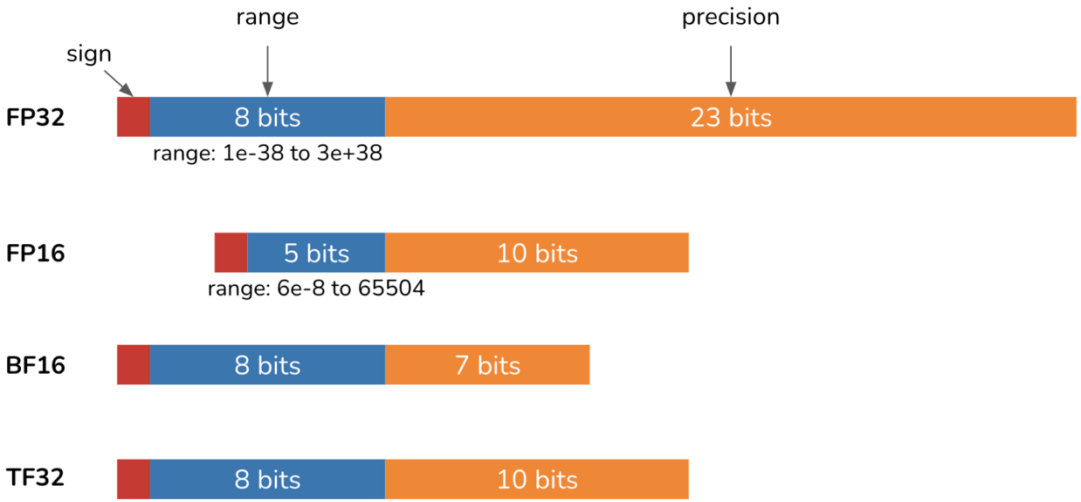


Figure 7-6. Different numerical formats with their range and precision.

- **주의**: 모델 사용 시 의도된 수치 형식으로 로드해야 합니다. 잘못된 형식으로 로드 하면 모델 동작이 예기치 않게 변경될 수 있습니다

양자화(Quantization): 메모리와 속도를 줄이는 핵심 기법

양자화란?

- **정밀도를 줄여 (bit 수↓)** 모델 크기와 연산량을 줄이는 기술
- 예: 32비트 → 8비트로 줄이면 메모리 사용 **4배 감소**
- 정수(INT8, INT4) 또는 저정밀 부동소수(FP16, BF16, FP8 등) 사용

무엇을 / 언제 양자화할까?

구분	내용
무엇을	주로 **가중치(weights)** 양자화 → 메모리 ↓, 성능 유지 쉬움
	활성화(activations) 양자화는 더 어려움
언제	훈련 후 양자화(PTQ*): 가장 보편적, 서빙에 최적화
	학습 중 양자화(QAT**): 성능 저하 최소화, **학습 시간**은 늘 수 있음

최신 흐름 & 사례

- **QLoRA (2023)**:
  - 고정된 사전학습 모델을 **4bit로 양자화**한 뒤, **LoRA 어댑터로 파인튜닝**하여 **성능 손실 없이 메모리 사용량을 크게 줄인 기법**
- **Apple (2024)**:
  - 평균 3.5비트 양자화를 적용해 약 30억 파라미터 모델을 iPhone에서 온디바이스로 실행하며,
  - 메모리 절감과 성능, 프라이버시를 모두 확보한 사례.
- **BitNet b1.58 (2024)**:
  - 모든 가중치를 {-1, 0, 1}의 삼진 값으로 양자화하여 평균 1.58비트로 표현
  - 기존 FP16 모델과 유사한 성능을 유지하면서도 메모리 사용량과 연산 비용을 획기적으로 줄임

***PTQ (Post-Training Quantization)**:모델 훈련이 끝난 후, **추론 효율을 높이기 위해 정밀도를 낮추는 양자화 기법**으로, 간단하고 빠르지만 경우에 따라 성능 저하가 발생할 수 있음.
****QAT(Quantization-Aware Training)란? 훈련 중에 정밀도가 낮은 형식(INT8 등)의 동작을 시뮬레이션하여, 실제 양자화 이후에도 성능이 유지되도록 학습하는 방법**입니다.

Parameter-Efficient Finetuning, PEFT

- **훈련 가능한 파라미터 수를 줄여 메모리 요구 사항을 최소화**하는 기법
- **전체 미세 조정 (Full Finetuning)**: 모델의 모든 파라미터를 업데이트하는 방식, 메모리 요구 사항이 높고 많은 데이터 필요
- **부분 미세 조정 (Partial Finetuning)**: 모델의 일부 파라미터만 업데이트하는 방식, 메모리 사용량은 줄일 수 있지만, 전체 미세 조정만큼의 성능을 얻으려면 많은 훈련 가능한 파라미터 필요
- PEFT는 **적은 수의 훈련 가능한 파라미터로 전체 미세 조정에 가까운 성능을 달성**하는 것을 목표로 합니다, 메모리 효율적일 뿐만 아니라 샘플 효율적이기도 합니다

PEFT 기법 상세 - 어댑터 기반 방법

- 기존 모델 아키텍처에 **작은 추가 레이어나 모듈을 도입**하여 학습시키는 방법입니다.
- 추론 시 추가적인 지연 시간을 유발하지 않도록 원래 레이어와 병합될 수 있습니다.
- 예시: BitFit, IA3, LongLoRA.

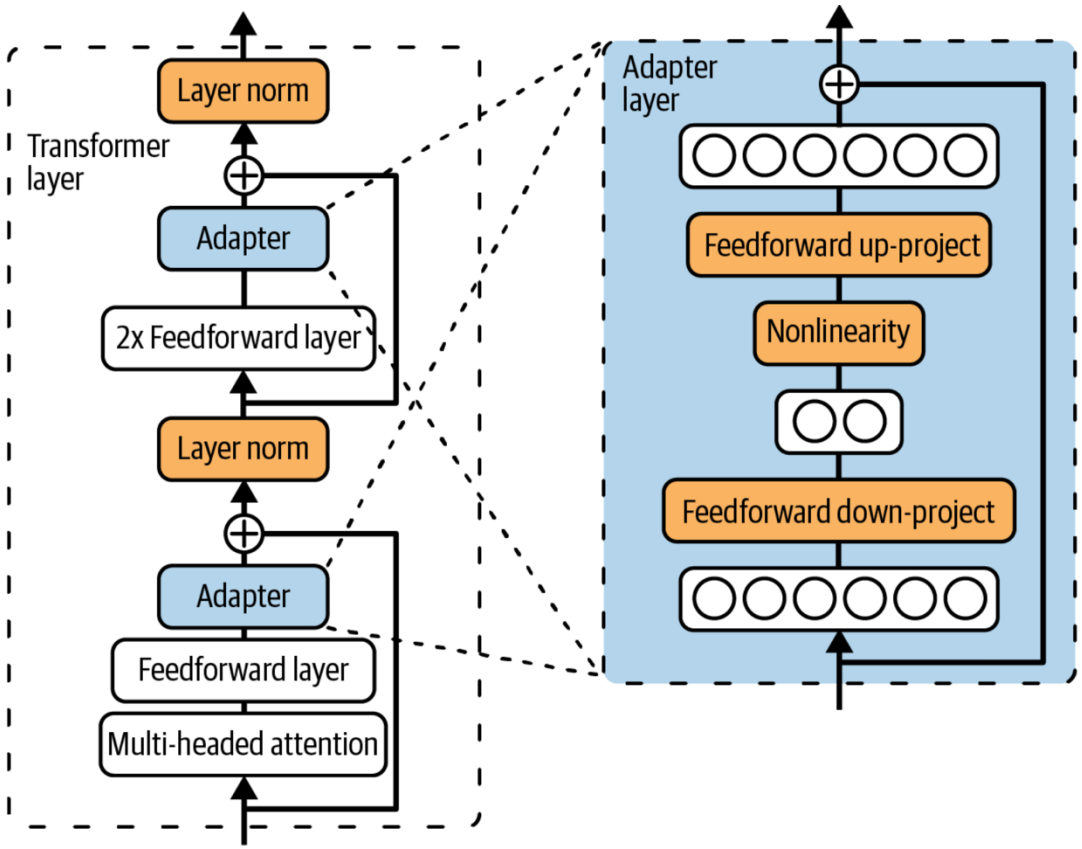


Figure 7-8. By inserting two adapter modules into each transformer layer for a BERT model and updating only the adapters, Houlsby et al. (2019) were able to achieve strong finetuning performance using a small number of trainable parameters.

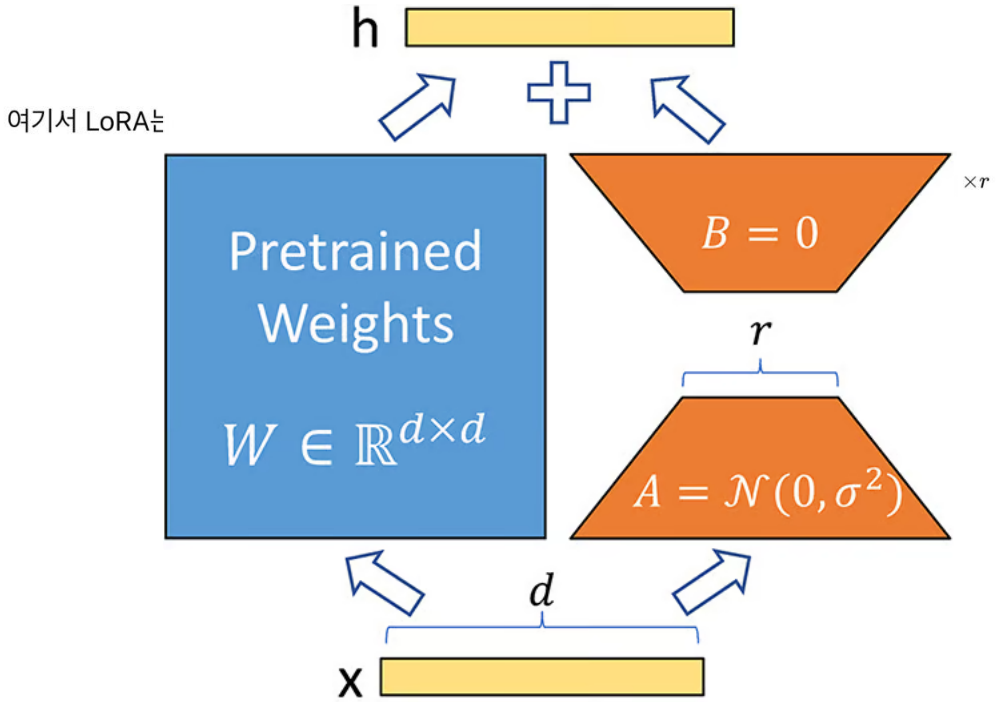
LoRA (Low-Rank Adaptation) 심층 분석

- 가장 인기 있는 어댑터 기반 방법
- 추가적인 레이어를 도입하지 않고 원래 레이어와 병합될 수 있는 모듈 사용으로 **추론 시 추가 지연 시간 없음**
- 가중치 행렬을 두 개의 작은 행렬의 곱으로 분해하여 업데이트합니다
- 적은 수의 훈련 가능한 파라미터로 높은 성능 달성
- **LoRA rank (r)**: LoRA의 성능은 순위에 따라 달라지지만, **작은 rank (4-64)로도 많은 사용 사례에서 충분**합니다
- **알파 값 (α)**: 병합 시 LoRA 가중치가 새로운 행렬에 얼마나 기여해야 하는지 결정합니다, 비율 (α/r) 조정이 중요합니다

$$W = W_0 + \alpha \cdot \frac{1}{r} \cdot BA$$

LoRA의 행렬 분해 기반 공식

기존 모델의 weight $W_0 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$에 대해 LoRA는 다음과 같은 방식으로 weight 업데이트를 합니다:



소스 : https://www.datacamp.com/tutorial/mastering-low-rank-adaptation-lora-enhancing-large-language-models-for-efficient-adaptation

PEFT 기법 상세 - 소프트 프롬프트 기반 방법

- **소프트 프롬프트 기반 방법**: 특별한 훈련 가능한 토큰을 입력에 추가하여 모델의 입력 처리 방식을 수정하는 방식
- **소프트 프롬프트**: 임베딩 벡터와 유사한 연속적인 벡터로, 사람이 읽을 수 없으며 훈련을 통해 최적화될 수 있습니다
- **하드 프롬프트**: 사람이 읽을 수 있는 이산적인 토큰으로, 훈련 불가능한 정적 프롬프트입니다
- **주요 소프트 프롬프트 기법**: Prefix-tuning, P-tuning, Prompt tuning 등. 소프트 프롬프트가 삽입되는 위치에 따라 다릅니다

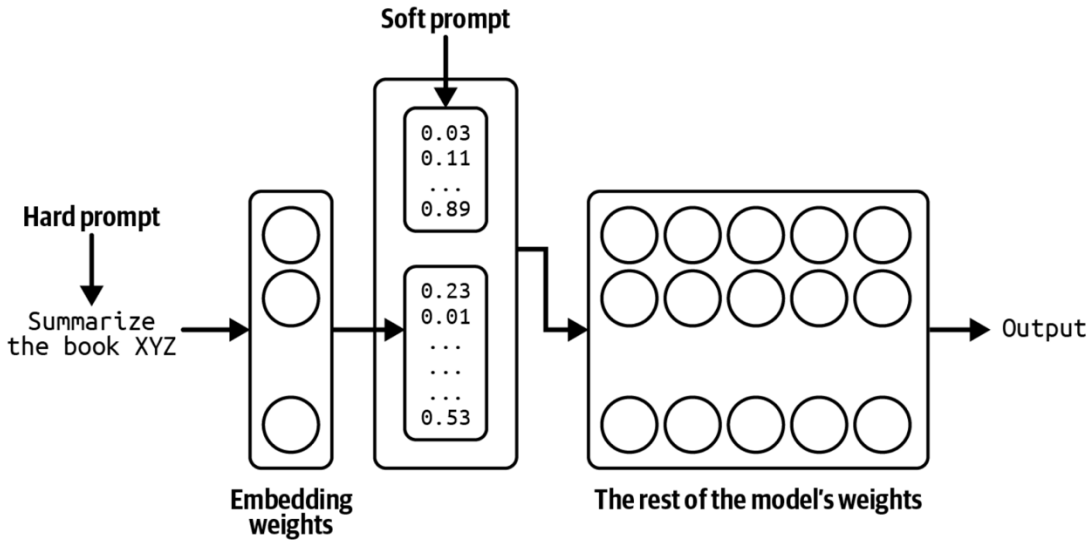


Figure 7-9. Hard prompts and soft prompts can be combined to change a model's behaviors.

모델 병합 (Model Merging)

- **여러 모델을 결합하여 새로운 기능을 가진 모델을 만드는 방식,** 미세 조정된 모델들을 병합하여 특정 목적에 더 적합한 모델을 만들 수 있습니다
- **목표**: 개별 모델보다 더 나은 성능을 제공하거나, 메모리 사용량을 줄이거나, 다양한 작업을 수행할 수 있는 단일 모델 생성
- GPU 없이도 가능하여 독립적인 개발자에게 매력적입니다
- **활용 사례**: 온디바이스 배포, 연합 학습, 모델 앙상블 대비 모델 병합

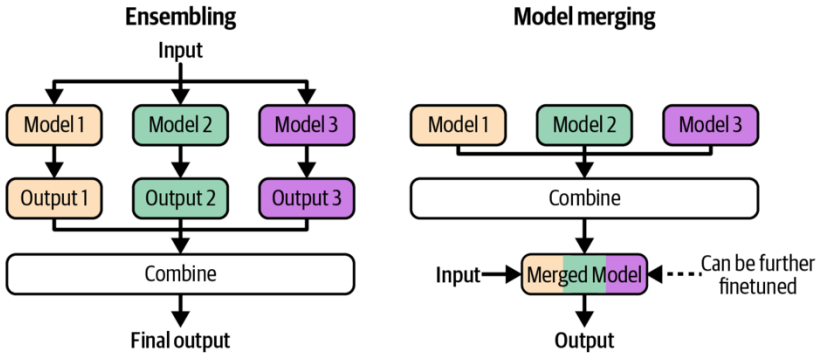


Figure 7-13. How ensembling and model merging work.

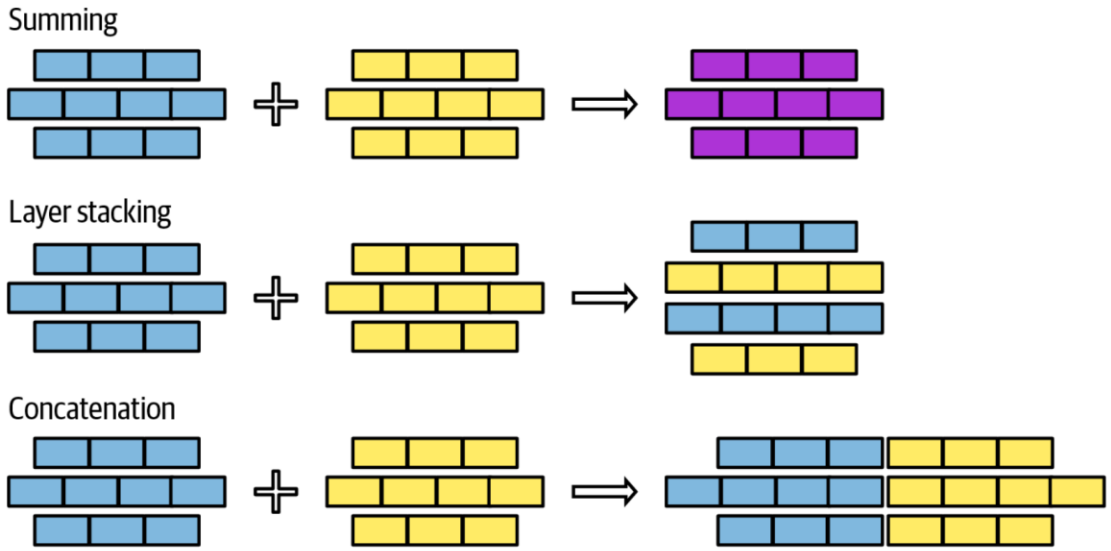


Figure 7-14. Three main approaches to model merging: summing, layer stacking, and concatenation.

모델 병합 접근 방식 - 합산 (Summing)

- 가중치 값을 더하는 방식
  - **선형 결합 (Linear Combination)**
  - **구형 선형 보간 (SLERP)**
- **작업 벡터 (Task Vector)** 개념 활용, 베이스 모델과 미세 조정된 모델의 차이를 나타냅니다
- **파라미터 가지치기 (Pruning)**를 통해 성능 향상 가능

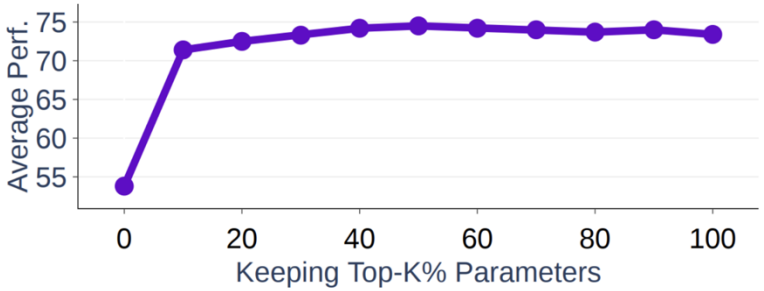


Figure 7-17. In Yadav et al.'s experiments, keeping the top 20% of the task vector parameters gives comparable performance to keeping 100% of the parameters.

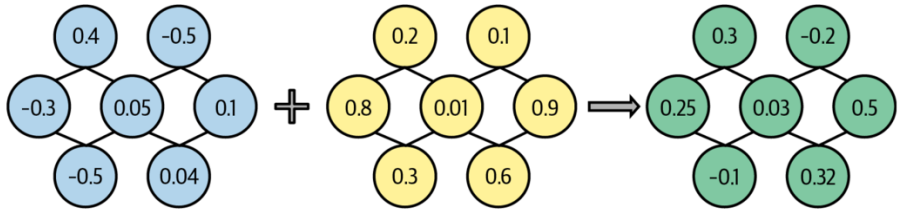


Figure 7-15. Merging parameters by averaging them.

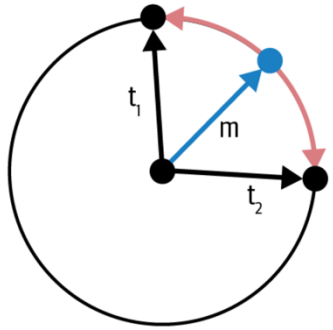


Figure 7-16. How SLERP works for two vectors t1 and t2. The red line is their shortest path on the spherical surface. Depending on the interpolation, the merged vector can be any point along this path. The blue vector is the resulting merged vector when the interpolation factor is 0.5.

모델 병합 접근 방식 - 레이어 스태킹 및 연결

- **레이어 스태킹 (Layer Stacking)**:
  - 여러 모델의 레이어를 쌓는 방식
  - MoE (Mixture-of-Experts) 모델 생성 및 모델 업스케일링에 활용
  - 추가 미세 조정 필요
- **연결 (Concatenation)**:
  - 파라미터를 연결하는 방식
  - 메모리 절감 효과는 없으며
  - 성능 향상 대비 파라미터 증가가 클 수 있어 권장되지 않음
  - LoRA 어댑터 연결 시 Rank 증가

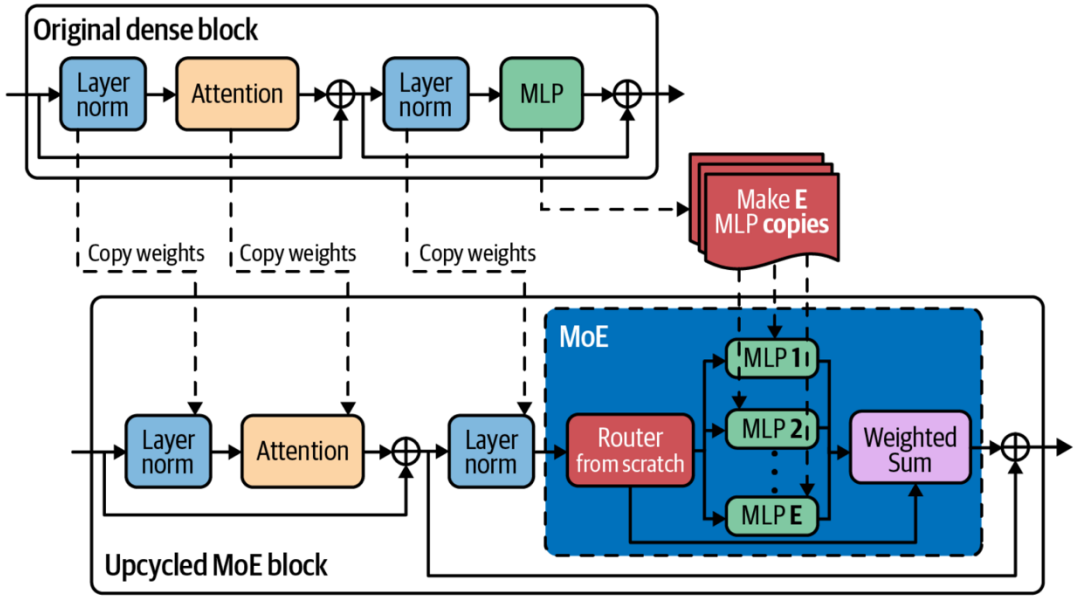


Figure 7-18. You can create an MoE model from a pre-trained model. Image adapted from Komatsuzaki et al. (2022).

파인튜닝 전략

1. 기본 모델 선택 전략

- **진행 경로**: 작은 모델 → 점점 강한 모델로 테스트하며 성능 확인
- **지식 증류 경로**: 큰 모델로 학습 → 생성한 데이터로 작은 모델 재학습

2. 파인튜닝 방법 선택

방식	특징
Full Finetune	성능 우수하지만 비용 큼 (수천~수만 예제 필요)
LoRA / PEFT	효율적이고 경량, 적은 데이터로도 효과적
QAT	낮은 정밀도 추론에 최적화, 학습 시간은 늘 수 있음

3. 파인튜닝 프레임워크

- **간편**: OpenAI API 등 (제한적이지만 쉬움)
- **유연함**: LLaMA-Factory, PEFT, Axolotl, LitGPT 등
- **분산 학습**: DeepSpeed, ColossalAI 등 대규모에 적합

4. 핵심 하이퍼파라미터

항목	설명
학습률	1e-7~1e-3, 손실 곡선 확인하며 조정
배치 크기	클수록 안정적이지만 메모리↑ → **gradient accumulation** 활용
에포크 수	작은 데이터셋은 더 많은 반복 필요, 과적합 여부 모니터링
프롬프트 손실 가중치	응답에 손실 가중치 집중 (보통 0~0.1 설정)

결론

- **미세 조정**은 대규모 파운데이션 모델을 특정 작업에 효과적으로 적응시키는 **강력한 방법**입니다.
- **PEFT 기법과 LoRA**의 발전으로 미세 조정의 접근성이 크게 향상되었고, 메모리 효율성과 모델 서빙 및 결합의 유연성이 증대되었습니다. **LoRA는 가장 많이 사용되는 PEFT 기술**입니다.
- 미세 조정, 프롬프트 엔지니어링, RAG 등의 방법을 **적절히 조합**하여 AI 애플리케이션의 성능을 최적화하는 것이 중요합니다.
- 다양한 오픈소스 프레임워크 (LLaMA-Factory, unsloth, PEFT, Axolotl, LitGPT 등)를 활용하여 효율적인 미세 조정을 수행할 수 있습니다.