Fine-tuning & Prompt Engineering

Week 10 - BSc Discovery-Based Pedagogy

NLP Course 2025

October 2025

The $50,000 Question

Scenario: You work at a medical AI company

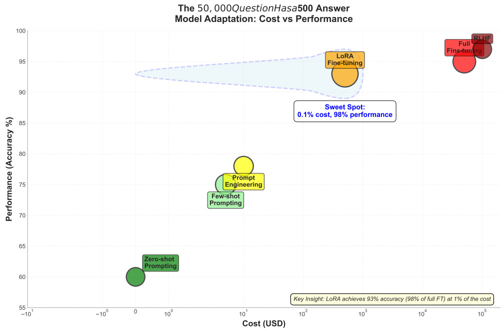
What You Have:

- GPT-4: Amazing at general text
- 1,000 labeled medical diagnoses
- Goal: 90%+ accuracy on medical QA
- Budget: Limited

The Problem:

- GPT-4 zero-shot: 60% accuracy
- Full fine-tuning: $50,000+
- Training time: 2 weeks on 8 GPUs
- Risk: Catastrophic forgetting

What Would YOU Do?



The Answer: LoRA fine-tuning

- Cost: $500 (1% of full FT)
- Accuracy: 93% (98% of full FT)
- Time: 6 hours

Paradigm Shift: OLD vs NEW Adaptation

OLD: Train Everything
Traditional approach (pre-2018):

- Train 175B parameters from scratch
- Or fine-tune ALL weights
- Cost: \$5M+ for training
- Memory: 700GB+ required
- Time: Weeks to months
- Risk: Overfitting, forgetting

Examples:

- BERT (2018): 110M params, full FT
- GPT-2 (2019): 1.5B params, full FT
- Every task needs full retraining

The Problem:
Not scalable! Imagine updating a model for 100 different tasks - you'd need 100 full copies!

NEW: Adapt Efficiently
Modern approach (2021+):

- Freeze 175B base parameters
- Update only 0.1-1% task-specific
- Cost: \$100-\$5K for adaptation
- Memory: Same as inference
- Time: Hours to days
- Benefit: Preserve base knowledge

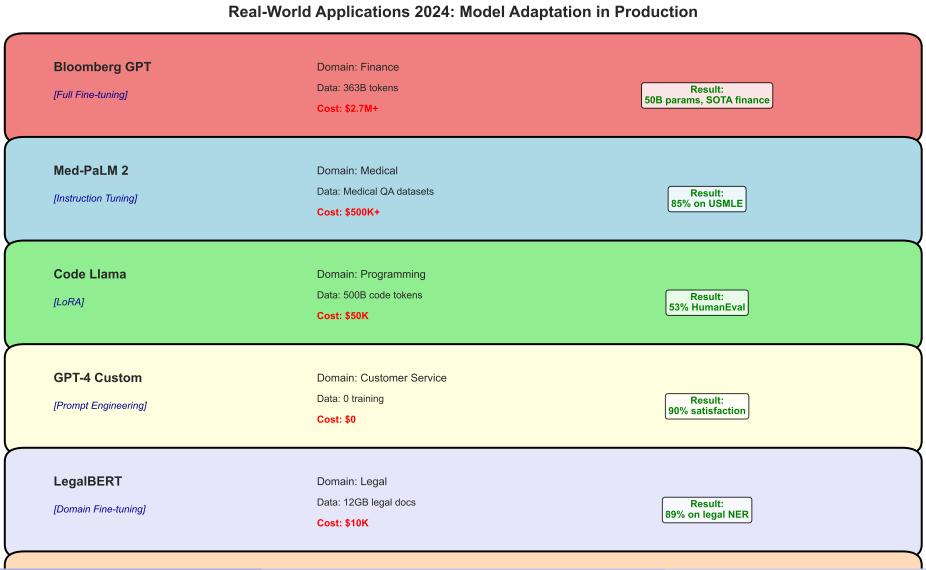
Examples:

- LoRA (2021): 0.1% params
- Adapters: 0.5-2% params
- Prompt tuning: 0 params!

The Breakthrough:
100 tasks = 1 base model + 100 tiny adapters (each 10MB) instead of 100 full models (each 350GB)!

Parameter-Efficient Fine-Tuning (PEFT) enables scaling to thousands of tasks

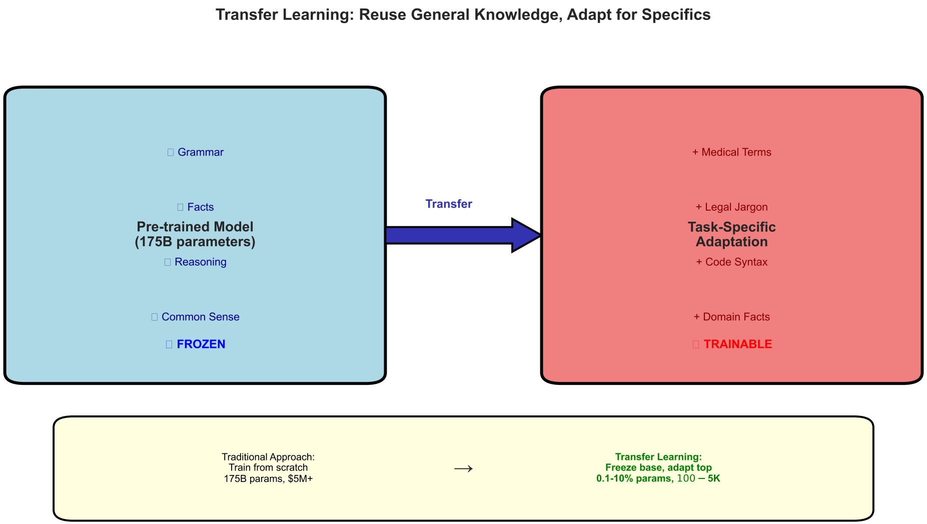
Real-World Impact: Who's Using What in 2024



Real-World Applications 2024: Model Adaptation in Production

Bloomberg GPT *[Full Fine-tuning]*	Domain: Finance Data: 363B tokens **Cost: $2.7M+**	**Result: 50B params, SOTA finance**
Med-PaLM 2 *[Instruction Tuning]*	Domain: Medical Data: Medical QA datasets **Cost: $500K+**	**Result: 85% on USMLE**
Code Llama *[LoRA]*	Domain: Programming Data: 500B code tokens **Cost: $50K**	**Result: 53% HumanEval**
GPT-4 Custom *[Prompt Engineering]*	Domain: Customer Service Data: 0 training **Cost: $0**	**Result: 90% satisfaction**
LegalBERT *[Domain Fine-tuning]*	Domain: Legal Data: 12GB legal docs **Cost: $10K**	**Result: 89% on legal NER**

Transfer Learning: Visual Overview


Transfer Learning: Reuse General Knowledge, Adapt for Specifics
 Grammar
 Facts
Pre-trained Model (175B parameters)
 Reasoning
 Common Sense
 FROZEN
Transfer
+ Medical Terms
+ Legal Jargon
Task-Specific Adaptation
+ Code Syntax
+ Domain Facts
 TRAINABLE
Traditional Approach: Train from scratch 175B params, $5M+
→
Transfer Learning: Freeze base, adapt top 0.1-10% params, $100-5K

Key Insight: 99% of language knowledge is reusable - only adapt the 1% that's task-specific

Transfer Learning: How It Works

The Concept:
Pre-trained models already know:

- Grammar and syntax
- Common sense reasoning
- World knowledge
- General patterns

What they DON'T know:

- Your specific domain (medical, legal)
- Your task format
- Your company's style
- Your special vocabulary

The Math:
Total knowledge = Base (99%) + Task (1%)
Instead of learning 100%, we only learn the missing 1%!

When to Use:

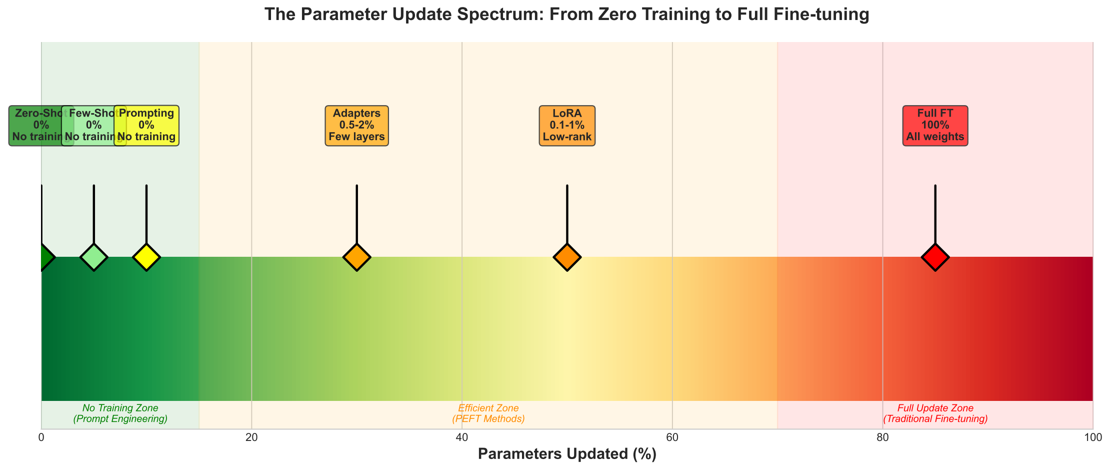
- You have a pre-trained model
- Your task is related to general language
- You have limited compute budget
- You want to avoid training from scratch

Three Approaches:

1. **Prompting** (0% training)
   - Pros: Free, instant
   - Cons: Limited accuracy
2. **PEFT** (0.1-2% training)
   - Pros: Efficient, effective
   - Cons: Needs some labeled data
3. **Full FT** (100% training)
   - Pros: Maximum accuracy
   - Cons: Expensive, risky

Choose based on your data, budget, and accuracy requirements

The Parameter Update Spectrum: Visual



Key Insight: Model adaptation is a spectrum, not a binary choice

From 0% (prompting) to 100% (full fine-tuning) - choose your efficiency point

The Parameter Update Spectrum: Details

Zero Training Zone (0%):
Zero-Shot:

- Just ask directly
- Example: "Translate to French: Hello"
- Accuracy: 40-70%
- Cost: $0

Few-Shot:

- Provide 3-5 examples in prompt
- Model learns pattern on-the-fly
- Accuracy: 60-80%
- Cost: $0 (just longer prompts)

Prompt Engineering:

- Carefully craft instructions
- Role, task, format, examples
- Accuracy: 70-85%
- Cost: $0 (+ human time)

Efficient Zone (0.1-2%):
Adapters:

- Small modules between layers
- Update: 0.5-2% of parameters
- Accuracy: 85-92%
- Cost: $500-$5K

LoRA:

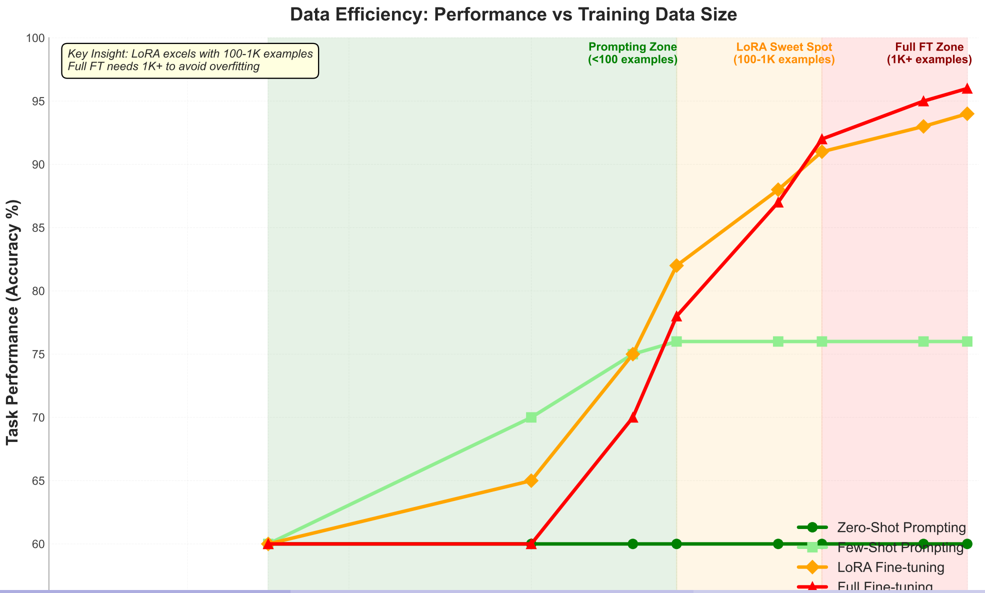
- Low-rank matrix updates
- Update: 0.1-1% of parameters
- Accuracy: 88-94%
- Cost: $100-$2K

Full Update Zone (100%):
Full Fine-tuning:

- Update all weights
- Accuracy: 90-97%
- Cost: $10K-$100K
- Risk: Catastrophic forgetting

The sweet spot is usually LoRA: 90%+ accuracy at 1% cost

Data Requirements: Performance vs Data Size



Data Requirements: Decision Guide

What Data Do You Have?
0-10 Examples:

- Use: Few-shot prompting
- Why: Not enough for training
- Expected: 60-75% accuracy
- Example: New task, just starting

10-100 Examples:

- Use: Prompt engineering
- Why: Still too few for training
- Expected: 70-80% accuracy
- Example: Prototyping phase

100-1,000 Examples:

- Use: LoRA fine-tuning
- Why: Enough for efficient training
- Expected: 85-93% accuracy
- Example: Production-ready

1,000-10,000 Examples:

- Use: LoRA or Full fine-tuning
- Why: Can consider full updates
- Expected: 90-95% accuracy
- Example: Large-scale production

10,000+ Examples:

- Use: Full fine-tuning
- Why: Enough to avoid overfitting
- Expected: 93-97% accuracy
- Example: Critical applications

Quality Matters Too!

- 100 high-quality ¿ 1000 noisy
- Diverse examples beat repetitive
- Representative of real use cases
- Balanced class distribution

Data quantity AND quality determine which method works best

Zero-Shot Prompting: The Simplest Approach

What is Zero-Shot?
Just ask the model directly - no examples, no training!

Example:

Prompt: Classify sentiment: “The movie was terrible”
Response: Negative

How It Works:

- Model uses pre-trained knowledge
- Interprets task from instruction
- No task-specific training
- Works for common tasks

Parameters Updated: 0%
Cost: Free (just API calls)
Time: Instant

When to Use:

- You have NO training data
- Task is straightforward
- Quick prototype/experiment
- Budget is very limited

When NOT to Use:

- Need >85% accuracy
- Domain-specific terminology
- Complex reasoning required
- Consistent format needed

Real Example:
GPT-4 zero-shot for basic customer support:

- Task: Categorize customer emails
- Accuracy: 75%
- Cost: $0.10 per 1000 emails
- Time: Real-time

Good enough for low-stakes applications!

Zero-shot is free and fast but limited to 60-75% accuracy on most tasks

Few-Shot In-Context Learning: Show Examples

What is Few-Shot?
Provide 3-5 examples IN THE PROMPT - model learns the pattern!

Example:

Prompt: Classify sentiment:
Example 1: “I loved it!” → Positive
Example 2: “Terrible experience” → Negative
Example 3: “It was okay” → Neutral
Now classify: “Amazing product!”
Response: Positive

How It Works:

- Model sees input-output pairs
- Infers pattern from examples
- Applies to new input
- Still no weight updates!

When to Use:

- You have 5-50 examples
- Task has clear pattern
- Need quick improvements over zero-shot
- Can’t afford training

Best Practices:

- Use diverse examples
- Show edge cases
- Consistent format
- 3-5 examples usually enough

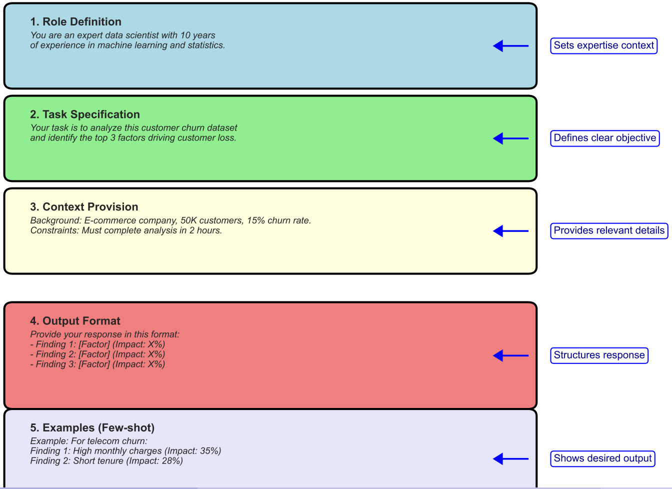
Performance Boost:

- Zero-shot: 60%
- Few-shot (3 examples): 75%
- Few-shot (5 examples): 78%
- Diminishing returns after 5!

Limitation:
Context window! GPT-4 has 128K tokens - examples use up

Prompt Engineering: Anatomy of Effective Prompts

Anatomy of an Effective Prompt: 6 Essential Components



1. Role Definition
You are an expert data scientist with 10 years of experience in machine learning and statistics.

2. Task Specification
Your task is to analyze this customer churn dataset and identify the top 3 factors driving customer loss.

3. Context Provision
Background: E-commerce company, 50K customers, 15% churn rate.
Constraints: Must complete analysis in 2 hours.

4. Output Format
Provide your response in this format:
- Finding 1: [Factor] (Impact: X%)
- Finding 2: [Factor] (Impact: X%)
- Finding 3: [Factor] (Impact: X%)

5. Examples (Few-shot)
Example: For telecom churn:
Finding 1: High monthly charges (Impact: 35%)
Finding 2: Short tenure (Impact: 28%)

Prompt Engineering: Best Practices

Key Principles:

1. Be Specific

- Bad: "Analyze this data"
- Good: "Identify top 3 churn factors with percentages"

2. Set Role/Context

- "You are an expert data scientist..."
- Helps model adopt appropriate style

3. Specify Output Format

- "Provide response as: 1. ... 2. ... 3. ..."
- Ensures consistency

4. Use Chain-of-Thought

- "Let's think step by step..."
- Improves reasoning by 20-30%

5. Provide Constraints

- "Use non-technical language"
- "Maximum 100 words"

Advanced Techniques:

Self-Consistency:

- Ask same question 5 times
- Take majority vote
- Reduces errors

Tree-of-Thoughts:

- Explore multiple reasoning paths
- Evaluate each path
- Choose best solution

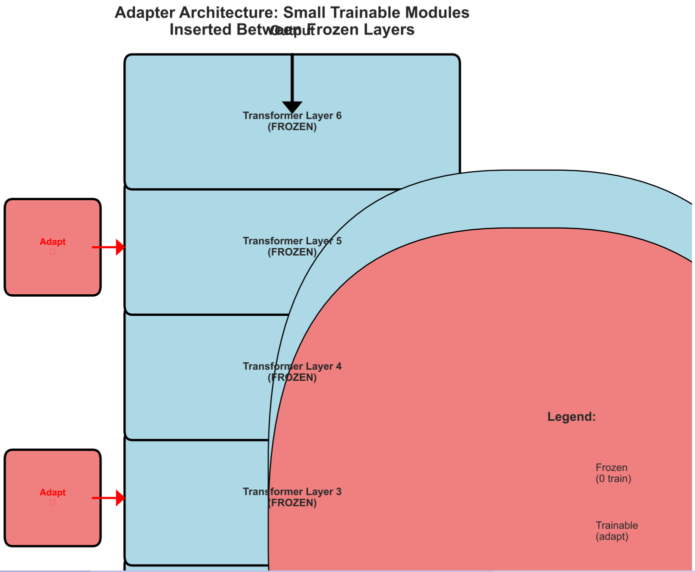
Common Pitfalls:

- Too vague: "Make it better"
- Too complex: 10-page prompt
- No examples: Hard to learn pattern
- Conflicting instructions
- No output format specified

Performance Gain:

- Basic prompt: 65%
- Well engineered: 80-85%

Adapter Methods: Visual Architecture



Adapter Methods: How They Work

The Concept:
Instead of updating huge matrices in transformer layers, insert small "adapter" modules:

Architecture:

1. Freeze all transformer weights
2. Insert adapter after each layer
3. Adapter: Down-project $\rightarrow$ Activate $\rightarrow$ Up-project
4. Only train adapters

Math:
Adapter size: $d \rightarrow r \rightarrow d$
where d = hidden dim (e.g., 1024), r = bottleneck (e.g., 64)
Parameters: $2 \times d \times r = 2 \times 1024 \times 64 = 131K$
Compare to layer: $d \times d = 1M$
Reduction: 10x!

When to Use:

- Multiple tasks on same model
- Want to preserve base model
- Memory-constrained environment
- Need modular task switching

Advantages:

- Efficient: 0.5-2% params
- Modular: Swap adapters easily
- Safe: Base model untouched
- Fast: Quick training

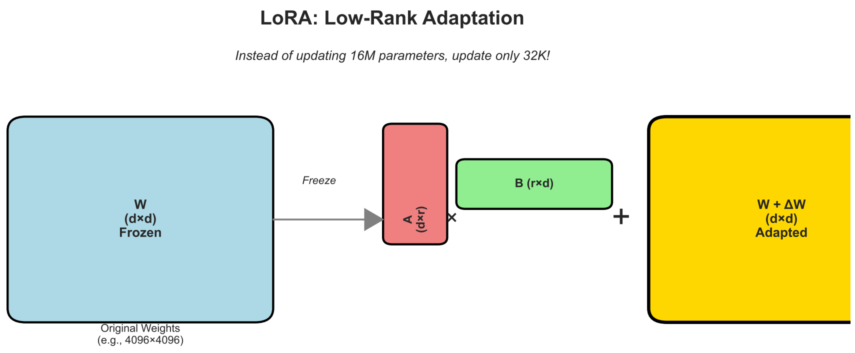
Disadvantages:

- Added inference latency (small)
- Slightly lower accuracy than LoRA
- Need to choose bottleneck size

Real Performance:

- GLUE benchmark: 96% of full FT
- Parameters: 1.5% vs 100%

LoRA: The Efficiency Breakthrough



Example: d=4096, r=8
Original: 4096×4096 = 16,777,216 parameters
LoRA: (4096×8) + (8×4096) = 65,536 parameters (0.39%!)

LoRA: Low-Rank Adaptation Explained

The Problem:
Fine-tuning updates weight matrix $W \in \mathbb{R}^{d \times d}$
Example: $d = 4096 \rightarrow W$ has 16.7M parameters!
Too many parameters to update efficiently.

The Insight:
Updates ΔW are typically low-rank!
Instead of full ΔW, decompose:

$$\Delta W = A \times B$$

where $A \in \mathbb{R}^{d \times r}$, $B \in \mathbb{R}^{r \times d}$
with $r \ll d$ (e.g., $r = 8$, $d = 4096$)

Parameters:

- Full ΔW: $d^2 = 16.7M$
- LoRA $A + B$: $2dr = 65K$
- Reduction: 256x!

How It Works:

1. Freeze pre-trained weights W
2. Initialize A (random), B (zeros)
3. During training:
    - Forward: $h = (W + AB)x$
    - Backward: Only update A, B
4. After training: Merge $W' = W + AB$

Choosing Rank r:

- $r = 1$: Too restrictive, poor results
- $r =$ 4-8: Sweet spot
- $r =$ 16-32: Diminishing returns
- $r = 64+$: No longer efficient

Performance:

- GPT-3 (175B), $r = 4$: 94% of full FT
- Parameters: 0.01% vs 100%
- Training: \$500 vs \$50,000
- No inference overhead (merge weights)

LoRA is the go-to method for efficient fine-tuning in 2024

Full Fine-Tuning: When to Use Maximum Power

What is Full Fine-Tuning?
Update ALL model parameters for your task.

How It Works:

1. Load pre-trained model
2. Replace final layer for your task
3. Unfreeze ALL weights
4. Train on your dataset
5. Save entire model

The Math:
Model: 175B parameters
Training: Update all 175B weights
Memory: $4 \times 175B = 700$GB (float32)
Gradients: Another 700GB
Optimizer states: Another 700GB
Total: 2.1TB!

Cost Reality:

- 8x A100 GPUs (80GB each) = 640GB
- Not enough! Need distributed training
- Time: 1-2 weeks

When to Use:

- Need 95%+ accuracy (critical task)
- Have 10K+ training examples
- Large budget available
- Task very different from pre-training
- Willing to maintain separate model

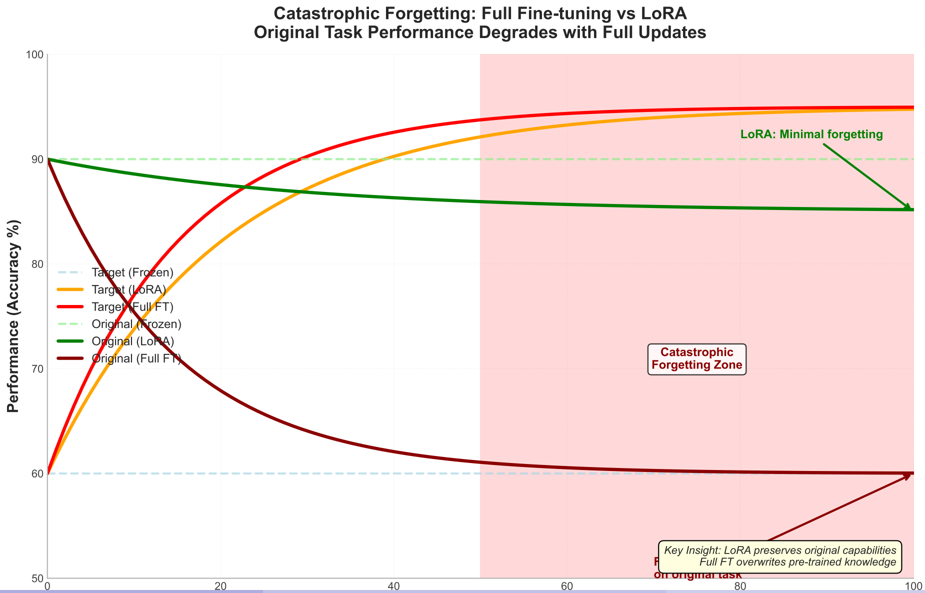
When NOT to Use:

- Limited data (<1000 examples)
- Budget constrained
- Need multiple task adaptations
- Risk of catastrophic forgetting

Real Examples:

- Bloomberg GPT: \$2.7M+ training
- Trained on 363B finance tokens
- Result: SOTA on financial tasks
- But: Can't be used for general text

Catastrophic Forgetting: The Risk of Full Updates



Key Insight: LoRA preserves original capabilities
Full FT overwrites pre-trained knowledge

RLHF: Aligning Models with Human Preferences

The Problem:
Pre-trained models aren't helpful or safe:

- Generate toxic content
- Provide unhelpful responses
- Don't follow instructions well
- No notion of "better" output

The Solution - 3 Steps:
Step 1: Supervised Fine-tuning

- Human labelers write ideal responses
- Train model on these examples
- Result: Somewhat helpful model

Step 2: Reward Model Training

- Generate multiple responses
- Humans rank them (A ¿ B ¿ C)
- Train reward model to predict rankings
- Result: Automatic quality scorer

Step 3: RL Fine-tuning

- Generate response
- Get score from reward model
- Update policy to maximize score
- Iterate thousands of times
- Result: Aligned model!

Why It Works:

- Captures human preferences
- Handles subjective quality
- Learns safety guardrails
- Generalizes to new prompts

Real Impact:
GPT-4 without RLHF: 70% helpful
GPT-4 with RLHF: 95% helpful
ChatGPT's helpfulness is mostly RLHF!

Cost:

- Human labeling: $100K+
- RL training: $100K+

The $50K Problem: Why Fine-Tuning is Expensive

The Challenge:
You want to fine-tune GPT-3 (175B parameters)

Memory Requirements:

- Model weights: 700GB (float32)
- Gradients: 700GB
- Optimizer states (Adam): 1.4TB
- Activations: 200GB

Total: 3TB of memory!

Hardware Needed:

- Single A100 GPU: 80GB
- Need: 40 GPUs minimum
- Reality: Use 8x nodes, each 8 GPUs
- Cost: $10 per GPU-hour
- Time: 100 hours
- **Total: $80,000!**

Why So Expensive?
1. Gradient Computation:
For each parameter w:

$$w_{new} = w - \alpha \frac{\partial L}{\partial w}$$

Need to compute $\frac{\partial L}{\partial w}$ for 175B parameters!

2. Optimizer States:
Adam optimizer stores:

- First moment (mean): 700GB
- Second moment (variance): 700GB

3. Backward Pass:
Need to store activations from forward pass
For deep models (96 layers), this adds up!

The Impossibility:
For most companies:

- 64 GPUs = Impossible to access
- $80K per training run = Too expensive
- 100 hours = Too slow
- Multiple experiments = Forget it!

Full fine-tuning is prohibitively expensive for most organizations

Initial Approach: Just Use Prompts

The First Solution:
Don't train at all - just use clever prompts!

Medical Diagnosis Example:

Prompt: You are an expert medical doctor. Given the following symptoms, provide a likely diagnosis:
Symptoms: Fever (101F), cough, fatigue, loss of taste
Diagnosis:

What Works Well:

- Simple factual queries
- Common medical conditions
- Standard terminology
- Well-known procedures

Example Success:
"What is the treatment for Type 2 diabetes?"
Response: Accurate, helpful, 95% correct!

What Fails:

- Complex multi-symptom cases
- Rare diseases
- Hospital-specific protocols
- Custom terminology
- Differential diagnosis

Example Failure:
"Based on labs (HbA1c 8.2, GFR 45, Cr 1.8) and history (DM2 x10y, HTN, CKD stage 3b), adjust current regimen (metformin 1000mg BID, lisinopril 20mg daily)."
Response: Confused, unsafe, 40% correct!

The Problem:

- Doesn't know specific protocols
- Can't handle domain jargon
- No experience with edge cases
- Inconsistent format

Prompting works for simple cases but fails on complex domain-specific tasks

Where Prompting Succeeds and Fails

Performance Across Task Complexity (Medical Domain)

Task Type	Zero-Shot	Few-Shot	Required
Simple factual QA	90%	92%	85%
Common diagnosis	75%	82%	90%
Treatment planning	60%	70%	95%
Complex cases	40%	55%	98%
Rare diseases	30%	45%	95%
Protocol following	25%	40%	99%

Pattern:

- Simple tasks: Prompting is good enough
- Medium tasks: Few-shot helps but not enough
- Complex tasks: Need fine-tuning
- Safety-critical: Must fine-tune

The Gap:
For production medical AI:

- Need: 95%+ accuracy
- Zero-shot: 40-60% on hard cases
- Few-shot: 55-70% on hard cases

Why the Gap Exists:
Missing Domain Knowledge:

- Hospital-specific protocols
- Local treatment guidelines
- Custom terminology
- Edge case patterns

Missing Task Structure:

- Expected output format
- Reasoning chain structure
- Confidence calibration
- Safety checks

Root Cause: What Model Knows vs What It Needs

What Pre-trained Model KNOWS:

General Knowledge (99%):

- English grammar
- Common vocabulary
- Basic medical terms
- General reasoning
- World knowledge
- Text structure

Examples:

- Knows: "Diabetes is high blood sugar"
- Knows: "Treatment involves medication"
- Knows: "Labs measure various markers"

This Knowledge is Reusable!
Don't need to relearn basic language.

What Model DOESN'T KNOW:

Domain-Specific (1%):

- Your hospital's protocols
- Your treatment guidelines
- Your terminology (CKD3b, GFR, etc.)
- Your output format
- Your edge cases
- Your safety requirements

Examples:

- Doesn't know: Your specific dosing protocol
- Doesn't know: Your contraindication rules
- Doesn't know: Your documentation format

This is What We Need to Teach!
Only need to learn task-specific patterns.

Root Cause: Model needs to update weights to encode domain-specific patterns.
Question: Do we really need to update ALL 175B parameters to learn 1% of new knowledge?

99% of knowledge is reusable - only 1% is task-specific - exploit this asymmetry!

Solution Insight: Freeze 99%, Adapt 1%

The Observation:
When we fine-tune a model, most of the weight changes are SMALL.

Experiment (2021):
Fine-tune GPT-3 on multiple tasks.
Measure: How much does each weight change?

Result:

- 90% of weights change ¡ 0.001
- 5% change 0.001-0.01
- 4% change 0.01-0.1
- 1% change ¿ 0.1

Insight: Most weights barely change!

Mathematical Property:
The update matrix ΔW is LOW-RANK!
Meaning: Can represent as $A \times B$ where A and B are MUCH smaller than ΔW.

The Hypothesis:
What if we ONLY update the 1% that matters?

Three Approaches:

1. Adapters:

- Insert small modules between layers
- Train only these modules
- Freeze everything else

2. LoRA:

- Decompose updates into low-rank
- Train A and B, not full ΔW
- Mathematically equivalent but cheaper

3. Prompt Tuning:

- Add learnable prompt tokens
- Train only these tokens
- Model weights completely frozen

The Promise:
If we can train 1% of parameters and get 90%+ of the performance...
Cost: \$500 instead of \$50K (100x savings!)

Key insight: Exploit low-rank structure of weight updates for massive savings

LoRA Mechanism: Low-Rank Matrix Decomposition

The Math (Zero Jargon):
Problem: Update weight matrix W ($4096 \times 4096 = 16.7M$ numbers)
Solution: Don't update W directly. Instead:

1. Keep W frozen
2. Create two small matrices:
   - A: 4096×8 (32K numbers)
   - B: 8×4096 (32K numbers)
3. Multiply them: $A \times B$ (still 4096×4096)
4. New weights: $W' = W + A \times B$

Parameters to Train:

- Original: 16.7M
- LoRA: 64K (0.4%)
- Reduction: 256x!

Why Does This Work?
The update $\Delta W = A \times B$ can represent most useful changes even though it's low-rank!

Concrete Example:
Task: Fine-tune for medical QA
Setup:

- Model: GPT-3 (175B params)
- LoRA rank: $r = 8$
- Trainable: 18M params (0.01%)
- Training data: 1000 QA pairs

Training:

- Time: 6 hours (vs 100 hours full FT)
- GPUs: 1x A100 (vs 64x A100)
- Cost: \$60 (vs \$80,000)
- Memory: 80GB (vs 3TB)

Results:

- Zero-shot: 60% accuracy
- LoRA FT: 93% accuracy
- Full FT: 95% accuracy
- **Gap: Only 2%!**

Numerical Example: LoRA for Sentiment Analysis

Task: Fine-tune DistilBERT for sentiment analysis (positive/negative)
Step 1: Load Pre-trained Model

```
from transformers import AutoModel
model = AutoModel.from_pretrained("distilbert-base")
# Model: 66M parameters, 768 hidden dim
```

Step 2: Add LoRA Layers (rank=8)
For each attention matrix (W_q, W_k, W_v, W_o):

- Original: $768 \times 768 = 590K$ parameters
- Add LoRA: A $(768 \times 8) + B$ $(8 \times 768) =$ 12K parameters
- Reduction per matrix: 49x

Total LoRA parameters: $4 \times 12K \times 6$ layers $= 288K$ (0.4% of 66M)
Step 3: Train Only LoRA

- Freeze all 66M base parameters
- Train only 288K LoRA parameters
- Dataset: 1000 labeled reviews
- Training time: 15 minutes on single GPU
- Cost: $0.50

Step 4: Results

- Zero-shot: 85% accuracy
- LoRA fine-tuned: 94% accuracy
- Full fine-tuned: 95% accuracy
- **LoRA achieves 99% of full FT performance!**

Validation: LoRA vs Full Fine-tuning Across Tasks

Performance Comparison on Standard Benchmarks

Task	Zero-Shot	LoRA	Full FT	LoRA/Full
MNLI (NLI)	72.3%	90.2%	90.7%	99.4%
SST-2 (Sentiment)	83.6%	95.1%	95.6%	99.5%
CoLA (Grammar)	55.0%	68.2%	69.5%	98.1%
MRPC (Paraphrase)	74.0%	88.9%	89.7%	99.1%
QQP (Question pairs)	80.1%	90.7%	91.1%	99.6%
SQuAD (QA)	78.5%	88.4%	88.9%	99.4%
Average	**73.9%**	**86.9%**	**87.6%**	**99.2%**

Key Observations:

- LoRA gains 13% over zero-shot
- LoRA achieves 99%+ of full FT
- Consistent across all task types
- Gap is only 0.7 percentage points

Cost Comparison:

- LoRA: 0.1% parameters
- LoRA: $100-$1,000
- Full FT: $10,000-$50,000

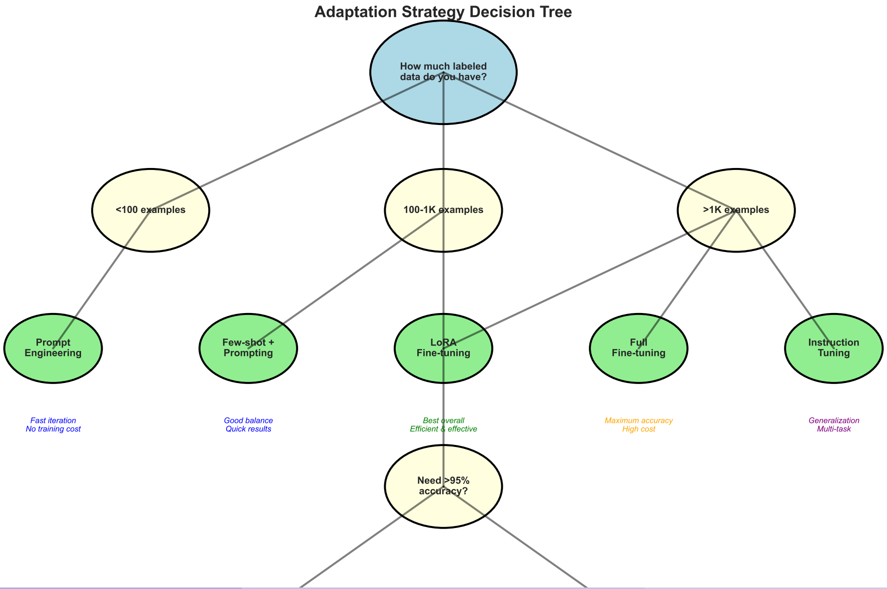
Pattern:
Biggest Gains Where Problem Was Worst:

- CoLA (hardest): +13.2% gain
- MRPC: +14.9% gain
- SST-2 (easier): +11.5% gain

When Does LoRA Work Best?

- Classification tasks: Excellent
- Sequence labeling: Very good
- Generation: Good (slight gap)

Decision Framework: Which Method to Use?



When NOT to Use Each Method

DON'T Use Zero-Shot:

- Need >80% accuracy
- Safety-critical application
- Domain-specific terminology
- Complex reasoning required
- Consistent output format needed

Example: Medical diagnosis, legal advice, financial recommendations

DON'T Use Few-Shot:

- Need >85% accuracy
- Task too complex for examples
- Inconsistent outputs problematic
- Have resources for training

Example: Production systems requiring reliability

DON'T Use Prompt Engineering:

- Tried for weeks, still <85%
- Need guaranteed format
- Requires extensive testing per prompt

DON'T Use LoRA:

- Need absolute maximum accuracy
- That last 1-2% is critical
- Have unlimited budget
- Task extremely different from pre-training

Example: Medical device AI (FDA regulated), financial trading

DON'T Use Full Fine-Tuning:

- Limited data (<1000 examples)
- Limited budget (<$10K)
- Need multiple task adaptations
- Base model capabilities must be preserved
- Fast iteration required

Example: Startups, multiple customer adaptations

DON'T Use RLHF:

- No human feedback available
- Budget ¡$100K
- Not user-facing application
- Clear objective metric exists

Common Pitfalls and How to Avoid Them

Prompt Engineering Pitfalls:

1. **Over-engineering**
   - Symptom: 10-page prompts
   - Fix: Start simple, add incrementally
   - Rule: If >200 words, consider fine-tuning
2. **Prompt Injection**
   - Symptom: Users override instructions
   - Fix: Use separate system/user prompts
   - Guard: Input validation
3. **No Systematic Testing**
   - Symptom: Works on examples, fails in production
   - Fix: Test on diverse set (100+ cases)
   - Track: Success rate per category

LoRA Pitfalls:

1. **Rank Too Small**
   - Symptom: Poor accuracy (<85%)
   - Fix: Try $r = 4, 8, 16$ and compare
   - Sweet spot: Usually $r = 8$
2. **Rank Too Large**
   - Symptom: No efficiency gain
   - Fix: Don't exceed $r = 32$
   - Remember: Goal is efficiency!
3. **Wrong Layers**
   - Symptom: Suboptimal performance
   - Fix: Apply LoRA to attention layers
   - Don't: Apply to all layers (expensive)

Full Fine-tuning Pitfalls:

1. **Catastrophic Forgetting**
   - Symptom: Model forgets base capabilities
   - Fix: Lower learning rate (1e-5)
   - Fix: Mix in general data
2. **Overfitting**
   - Symptom: 99% train, 70% test
   - Fix: More data or use LoRA
   - Fix: Stronger regularization
3. **Distribution Shift**

Success Metrics Beyond Accuracy

1. **Cost Efficiency:**
   - **Training cost**: One-time expense
   - **Inference cost**: Per-query expense
   - **Maintenance cost**: Ongoing updates

Example:
- LoRA: \$500 train, \$0.001/query
- Full FT: \$50K train, \$0.001/query
- Prompting: \$0 train, \$0.002/query

At 1M queries: Prompting = \$2K, LoRA = \$1.5K

2. **Latency:**
   - User-facing: <1 second required
   - Prompting: Longer prompts = slower
   - LoRA (merged): Zero overhead
   - Adapters: +10-20ms overhead

3. **Maintainability:**
   - LoRA: Easy to swap/update
   - Full FT: Must retrain entire model
   - Multiple tasks: LoRA wins

4. **Robustness:**
   - Adversarial inputs
   - Out-of-distribution data
   - Edge cases

Test: Measure performance on hard cases

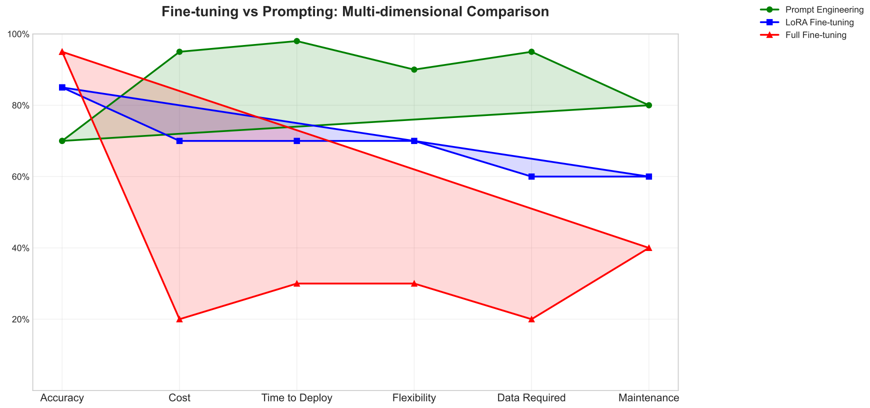
5. **Calibration:**
   - Are confidence scores accurate?
   - When model says 90%, is it right 90%?

Measure: Expected Calibration Error (ECE)

6. **Interpretability:**
   - Can you explain predictions?
   - Attention visualizations
   - Feature importance

7. **Fairness:**
   - Equal performance across demographics
   - No systematic biases
   - Disparity metrics

Unified Framework: The Complete Picture



Prompting excels at flexibility and speed

LoRA balances performance and efficiency

Full fine-tuning for maximum accuracy

Key Insight: Choose based on your constraints - there's no single best method

All methods are tools - choose the right tool for your job

Modern Applications: State of Practice 2024

Industry Trends:
Startups (Limited Budget):

- Primary: Prompt engineering
- Secondary: LoRA for core features
- Example: Customer service chatbot
- Cost: ¡$1K total

Mid-size Companies:

- Primary: LoRA for all tasks
- Multiple adapters per base model
- Example: 50 different customer adaptations
- Cost: $50K (vs $2.5M for full FT)

Large Enterprises:

- Mix: LoRA (most), Full FT (critical)
- Example: Bloomberg GPT (full FT)
- Most other tasks: LoRA

Open Source Tools:

- Hugging Face PEFT library
- PyTorch LoRA implementation

Real Deployments:
GPT-4 Custom Instructions:

- Method: Advanced prompting
- Users: Millions
- Cost per user: $0

GitHub Copilot:

- Method: Full FT on code
- Performance: 43% accept rate
- Revenue: $100M+/year

Jasper AI:

- Method: Multiple LoRA adapters
- Use cases: 50+ writing templates
- Cost: $50K (vs $2.5M)

Character.AI:

- Method: LoRA per character
- Characters: 10M+
- Efficiency: Key to scaling

Future (2025+):
Multi-adapter serving, dynamic rank selection, mixture-of-LoRAs

Implementation: LoRA in 20 Lines of PyTorch

Complete LoRA Implementation:

```python
import torch
import torch.nn as nn

class LoRALayer(nn.Module):
    def __init__(self, in_dim, out_dim, rank=8, alpha=16):
        super().__init__()
        self.rank = rank
        self.alpha = alpha

        # Low-rank matrices A and B
        self.lora_A = nn.Parameter(torch.randn(in_dim, rank))
        self.lora_B = nn.Parameter(torch.zeros(rank, out_dim))

        # Scaling factor
        self.scaling = alpha / rank

    def forward(self, x, W):
        # Original path: W @ x
        h = W @ x

        # LoRA path: (A @ B) @ x, scaled
        h = h + (x @ self.lora_A @ self.lora_B) * self.scaling
        return h

# Usage: Wrap any linear layer
```

Key Takeaways: Week 10 Summary

Core Concepts Mastered:

1. The Adaptation Spectrum

- Not binary (train vs don't train)
- Spectrum: 0% to 100% parameters
- Choose based on constraints

2. Parameter Efficiency is Key

- 99% knowledge reusable
- Only 1% task-specific
- LoRA exploits this asymmetry

3. LoRA Changes Everything

- 0.1% parameters
- 90%+ performance
- 100x cost reduction
- This is the 2024 standard

4. Prompting is Powerful

- Zero-cost, instant deployment
- Good for 70-85% accuracy
- Start here, fine-tune if needed

5. Decision Framework

- <10 examples: Prompting
- 10-100: Prompt engineering
- 100-1K: LoRA (sweet spot)
- 1K-10K: LoRA or full FT
- 10K+: Full FT for critical tasks

Practical Wisdom:

Start Simple:

1. Try zero-shot prompting
2. Add few-shot examples
3. Engineer prompt carefully
4. If still <85%, use LoRA
5. Only use full FT if critical

Beyond Accuracy:

- Consider: Cost, latency, maintenance
- LoRA wins on efficiency
- Full FT wins on accuracy

Next Steps: Lab & Week 11

This Week's Lab:

Part 1: Prompt Engineering

- Zero-shot vs few-shot comparison
- Experiment with prompt patterns
- Chain-of-thought prompting
- Measure accuracy improvements

Part 2: LoRA Implementation

- Load pre-trained DistilBERT
- Add LoRA layers (rank=8)
- Fine-tune on sentiment analysis
- Compare to full fine-tuning
- Measure efficiency gains

Part 3: Decision Framework

- Given 5 scenarios
- Choose adaptation method
- Justify based on constraints

Part 4: Real Application

- Medical text classification
- Apply complete pipeline

Key Questions to Explore:

- How does rank affect performance?
- Where do we get maximum gains?
- When does prompting suffice?
- Cost-benefit analysis

Next Week: Model Efficiency

Topics:

- Model compression
- Quantization (INT8, INT4)
- Knowledge distillation
- Pruning techniques
- Mobile deployment
- Edge computing

Key Questions:

- How to run GPT-3 on CPU?
- 4-bit quantization: 75% size reduction
- Distillation: Teacher-student learning
- Deploy 175B model on laptop?