Lecture 14:

Pre-trained Language Models & LLM Era

Deep Learning for Natural Language Processing

Ho-min Park

homin.park@ghent.ac.kr

powersimmani@gmail.com

Lecture Contents

Part 1/9:

Introduction & Paradigm Shift

1. Review of Previous Lessons
2. Paradigm Shift in AI

NLP Evolution: From Rules to Neural Networks

Rule-Based Systems

Hand-crafted rules & feature engineering

RNN & LSTM

Sequential processing with memory

Seq2Seq Models

Encoder-decoder for translation

Word Embeddings

Word2Vec, GloVe semantic relationships

Attention Mechanism

Dynamic focus on relevant parts

Next Generation →

Transfer learning & deep context

⚠ Limitations

Shallow representations • Task-specific training

💡 The Need

Better context understanding • Transfer learning capabilities

Paradigm Shift in AI

❌ Old Paradigm

Task-Specific Models

- 🎯 **Task-specific** models trained from scratch
- 📊 Requires **labeled data** for each task
- 🔧 Heavy **feature engineering** needed
- 📉 **Limited scale** and capabilities
- 🚧 No knowledge **transfer** between tasks

→

✨ New Paradigm

Foundation Models

- 🌐 **General-purpose** foundation models
- 🔄 **Pre-train** on massive text, then adapt
- 📚 **Self-supervised** learning from unlabeled data
- 🚀 **Scale matters:** emergent capabilities
- 🤝 **Democratization:** accessible to all

Part 2/9:

Pre-training Concepts

3. What is Pre-training?
4. Language Modeling Objective Function
5. The Importance of Scale

Pre-training → Fine-tuning Pipeline

1

Pre-training

Self-Supervised Learning

Learn from **massive text corpora**
No manual labels required
Predict masked/next words

→

2

Fine-tuning

Task Adaptation

Adapt to **specific tasks**
Requires minimal data
Pre-train once, fine-tune many

Captures **syntax & semantics**

Learns **world knowledge**

Creates **universal extractors**

Reduces **data requirements**

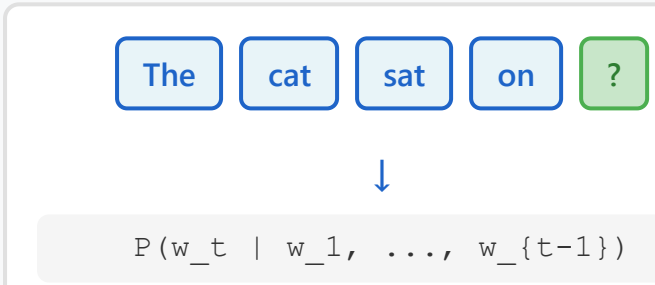
Key Advantages

✓ General language representations ✓ Two-stage learning process ✓ Efficient transfer to downstream tasks

Language Modeling Objective Functions

Autoregressive LM

Causal / Left-to-Right



→ Predict **next token**

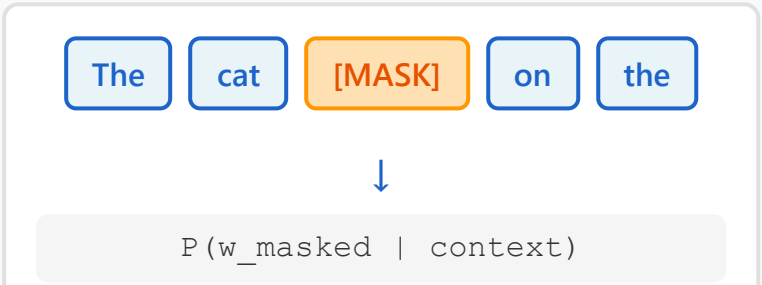
⚡ **Causal masking** (uni-directional)

✏️ Good for **generation** tasks

📝 GPT-style models

Masked LM

Bidirectional Context



🎯 Predict **masked tokens**

↔️ **Bidirectional** context (left+right)

🧠 Better **understanding**

📖 BERT-style models

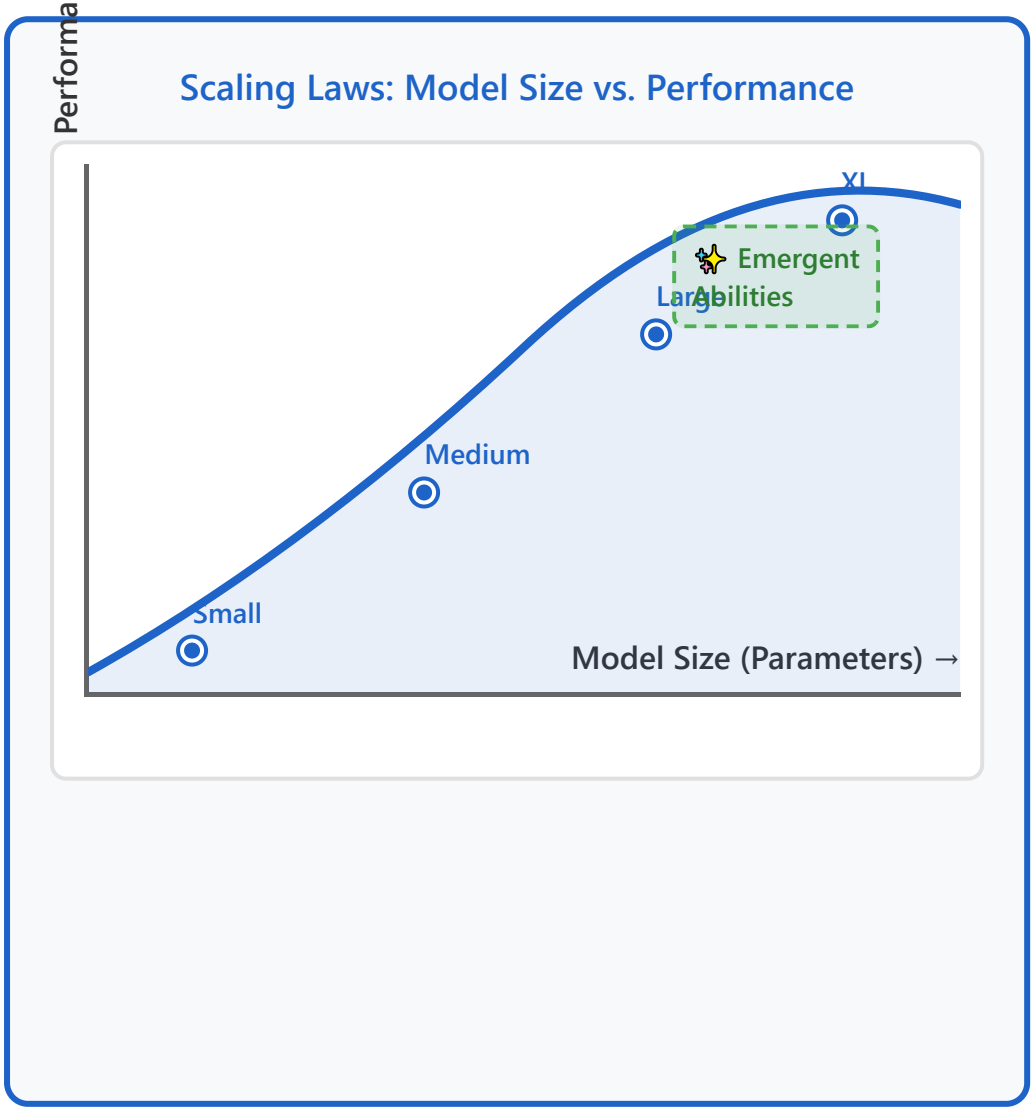
🎓 Training Approach

Self-supervision enables learning from unlimited text using **cross-entropy loss**

🔧 Architecture Fit

Different objectives suit different architectures and **downstream tasks**

The Importance of Scale



Predictable Growth
Performance improves **predictably** with size

More Parameters
Capture **more patterns** & knowledge

Larger Datasets
Richer knowledge coverage

Emergent Abilities
New capabilities at **scale thresholds**

Few-shot Learning
Dramatically **improves** with scale

Trade-off
Performance vs. **Efficiency & Accessibility**

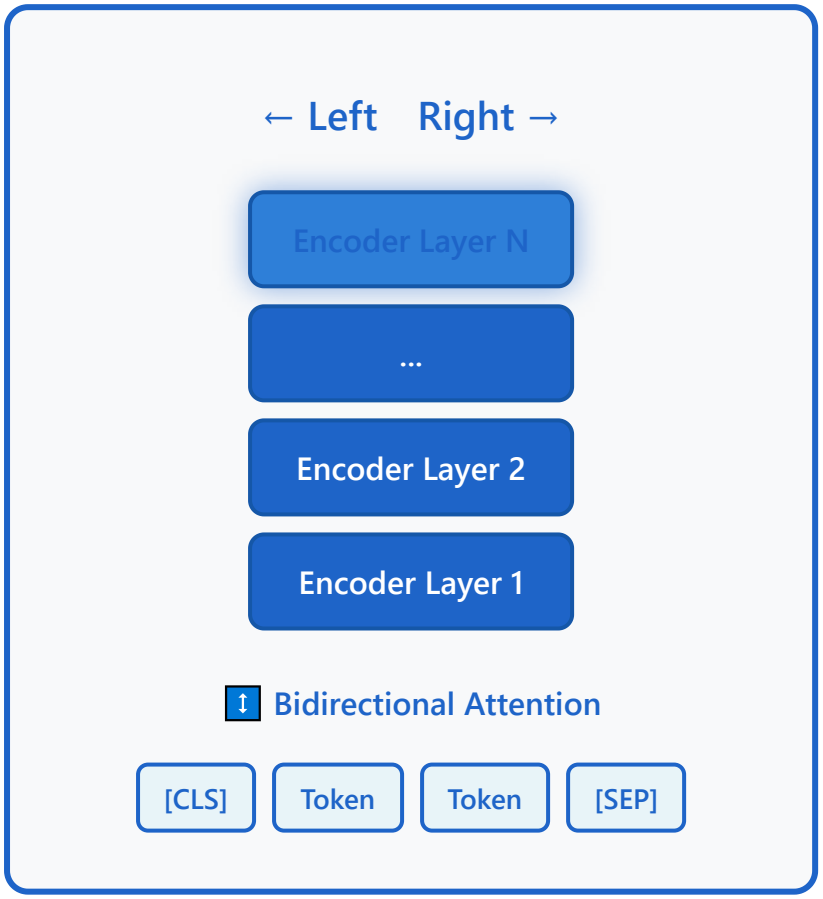
Part 3/9:

BERT - Encoder-based Models

6. Introduction to BERT
7. BERT Pre-training
8. BERT Fine-tuning
9. BERT Family Models

BERT: Bidirectional Encoder Representations from Transformers

Google, 2018 - Revolutionary NLP Model



Base: 110M params **Large:** 340M params

Dataset: BooksCorpus + Wikipedia (3.3B words)

Bidirectional Context

Deep **bidirectional** understanding from both directions

Transformer Encoder

Stack of **Transformer encoders** with self-attention

Pre-training Corpus

Trained on **BooksCorpus** and **Wikipedia**

Two Model Sizes

BERT-Base (110M) and BERT-Large (340M) parameters

State-of-the-Art Performance

Achieved SOTA on **11 NLP benchmarks**
Revolutionized NLP in 2018

🎓 BERT Pre-Training Process

1 Input Preparation

Tokenize the input text and add special tokens. [CLS] marks the start of the sentence, and [SEP] marks the end.

📝 **Example Input:**

```
"The cat sat on the mat"
→ [CLS] The cat sat on the mat [SEP]
```

2 Masked Language Modeling (MLM)

Randomly mask 15% of input tokens. The model learns to predict masked words using bidirectional context.

🎭 **Masking Example:**

```
Input: [CLS] The cat sat on the mat [SEP]
Masked: [CLS] The [MASK] sat on the mat [SEP]
Target: Predict "cat"
```

💡 **Key Points**

- 80% replaced with [MASK]
- 10% replaced with random word
- 10% keep original word

3 Next Sentence Prediction (NSP)

Learn to determine whether two sentences are consecutive. This improves the ability to understand relationships between sentences.

🔗 NSP Example:

Case 1 (IsNext):

```
[CLS] The cat sat on the mat [SEP] It was sleeping [SEP]
Label: IsNext ✓
```

Case 2 (NotNext):

```
[CLS] The cat sat on the mat [SEP] Paris is beautiful [SEP]
Label: NotNext ✗
```

4 Training Objective

Train the model by simultaneously optimizing both MLM and NSP losses.

MLM Loss

Cross-Entropy Loss aiming for accurate prediction of masked tokens

NSP Loss

Binary Classification Loss for correctly judging sentence pair continuity

🎯 **Total Loss**

Loss = MLM Loss + NSP Loss

🎯 BERT Fine-tuning Process

1 Load Pre-trained Model

Load the pre-trained BERT model. It has already learned general patterns of language.

✨ Pre-trained Knowledge

- Understanding grammatical structure
- Capturing semantic relationships
- Bidirectional context comprehension

2 Add Task-Specific Layer

Add an output layer specific to the task. Different structures are used depending on the task.

📊 Classification

[CLS] token output + Softmax Layer
(Sentiment analysis, topic classification, etc.)

🏷️ Token Classification

Each token output + Classification Layer
(Named entity recognition, POS tagging, etc.)

❓ Question Answering

Start/End Position Prediction Layers
(SQuAD, reading comprehension, etc.)

🔄 Sequence Pairing

[CLS] token + Binary Classifier
(Natural language inference, sentence similarity, etc.)

3 Fine-tune on Task Data

Fine-tune the entire model with task-specific training data. Typically, 2-4 epochs are sufficient.

📝 **Sentiment Analysis Example:**

```
Input: [CLS] This movie is amazing [SEP]
→ BERT Encoding →
→ Classification Layer →
Output: Positive (95% confidence)
```

⚙️ **Training Settings**

- Learning Rate: 2e-5 ~ 5e-5
- Epochs: 2-4
- Batch Size: 16 or 32

4 Inference & Prediction

Use the fine-tuned model to make predictions on new data.

🔮 **Prediction Pipeline:**

```
Step 1: Tokenize new input
Step 2: Pass through fine-tuned BERT
Step 3: Apply task-specific head
Step 4: Generate prediction with confidence score
```

🎯 **Final Output**

Generate results in the appropriate format for the task:
Classification labels, token tags, answer spans, etc.

BERT revolutionized NLP by introducing bidirectional pre-training and transfer learning to the field.

BERT Pre-training: Two Main Tasks

1 Masked Language Model

MLM - 15% masking

Input:

[CLS] The cat [MASK] on the mat

↓

Predict: "sat"

Uses **bidirectional context** to predict masked tokens

2 Next Sentence Prediction

NSP - Binary classification

Input:

A: The man went to the store.

B: He bought a gallon of milk.

↓

IsNext ✓

[CLS] token for sentence-level representation

Tokenization

WordPiece

30K vocabulary

Special Tokens

[CLS] [SEP] [MASK] [PAD]

Training

4 days

on 4-16 Cloud TPUs

Numerical Vector Calculation Examples

Training Phase: Forward & Backward Pass

Let's walk through BERT training with simplified dimensions (d=4) for clarity. Real BERT uses d=768.
Input: **"[CLS] The cat [MASK] on"** where [MASK] should predict "sat"

1 Tokenization & Token IDs

[CLS]	The	cat	[MASK]	on
ID: 101	ID: 1996	ID: 4937	ID: 103	ID: 2006

Token ID Sequence:

```
input_ids = [101, 1996, 4937, 103, 2006]
```

2 Embedding Lookup (Token + Position + Segment)

Token Embeddings (from embedding matrix, d=4):

$E_{[CLS]}$ = [0.12, -0.34, 0.56, 0.23]
E_{The} = [0.45, 0.67, -0.12, 0.89]
E_{cat} = [0.78, -0.45, 0.34, -0.67]
$E_{[MASK]}$ = [**0.11, 0.22, 0.33, 0.44**]
E_{on} = [-0.23, 0.56, 0.78, 0.12]

Position Embeddings (learned):

P_0 = [0.01, 0.02, 0.01, 0.02]
P_1 = [0.03, 0.01, 0.04, 0.02]
P_2 = [0.02, 0.05, 0.01, 0.03]
P_3 = [0.04, 0.02, 0.03, 0.01]
P_4 = [0.01, 0.03, 0.02, 0.04]

$$Input_i = TokenEmbed_i + PositionEmbed_i + SegmentEmbed_i$$

Final Input Embeddings (after addition):

$H^0_{[CLS]}$ = [0.13, -0.32, 0.57, 0.25]

H^0_{The} = [0.48, 0.68, -0.08, 0.91]

H^0_{cat} = [0.80, -0.40, 0.35, -0.64]

$H^0_{[MASK]}$ = [**0.15, 0.24, 0.36, 0.45**]

H^0_{on} = [-0.22, 0.59, 0.80, 0.16]

3 Self-Attention Calculation

$$Attention(Q, K, V) = softmax(QK^T / \sqrt{d_k}) \cdot V$$

Query for [MASK] position:

$Q_{[MASK]} = H^0_{[MASK]} \cdot W_Q$

= [0.15, 0.24, 0.36, 0.45] · W_Q

= [**0.42, -0.18, 0.31, 0.55**]

Keys for all positions:

$K_{[CLS]}$ = [0.22, 0.45, -0.12, 0.33]

K_{The} = [0.56, 0.11, 0.44, -0.23]

K_{cat} = [0.78, -0.34, 0.21, 0.67]

$K_{[MASK]}$ = [0.33, 0.22, 0.55, 0.11]

K_{on} = [-0.11, 0.67, 0.33, 0.44]

Attention Scores ($Q_{[MASK]} \cdot K^T / \sqrt{4}$):

score([MASK]→[CLS]) = (0.42×0.22 + (-0.18)×0.45 + 0.31×(-0.12) + 0.55×0.33) / 2 = **0.12**

score([MASK]→The) = (0.42×0.56 + (-0.18)×0.11 + 0.31×0.44 + 0.55×(-0.23)) / 2 = **0.14**

score([MASK]→cat) = (0.42×0.78 + (-0.18)×(-0.34) + 0.31×0.21 + 0.55×0.67) / 2 = **0.41**

score([MASK]→[MASK]) = **0.18**

score([MASK]→on) = **0.25**

After Softmax (Attention Weights):

α = softmax([0.12, 0.14, 0.41, 0.18, 0.25])
= [0.14, 0.15, **0.25**, 0.16, 0.19]

💡 Notice "cat" gets the highest attention (0.25) - BERT learns that "cat" is crucial context for predicting what action the cat performed!

Weighted Sum of Values → New [MASK] representation:

$H^1_{[MASK]} = 0.14 \times V_{[CLS]} + 0.15 \times V_{The} + \mathbf{0.25 \times V_{cat}} + 0.16 \times V_{[MASK]} + 0.19 \times V_{on}$

$= [\mathbf{0.52, -0.15, 0.48, 0.33}]$

4 MLM Output: Predict Masked Token

Final hidden state at [MASK] position (after 12 layers):

$H^{12}_{[MASK]} = [\mathbf{0.89, -0.23, 0.67, 0.45}]$

$$logits = H^{12}_{[MASK]} \cdot W_{vocab}{}^{T} + b \quad (30{,}522 \text{ vocabulary scores})$$

Vocabulary Logits (showing top words):

```
logits["sat"]    = 3.45  ← Highest!
logits["jumped"] = 2.12
logits["ran"]    = 1.89
logits["slept"]  = 1.67
logits["the"]    = -2.34
... (30,517 more)
```

After Softmax → Probabilities:

"ran"	0.08
others	0.05

5 Loss Calculation & Backpropagation

Cross-Entropy Loss for MLM

$$L_{MLM} = -log(P(sat \mid context)) = -log(0.72) = \mathbf{0.329}$$

NSP Loss (if IsNext=True, P=0.85):

$L_{NSP} = -\log(0.85) = \mathbf{0.163}$

Total Pre-training Loss:

$L_{total} = L_{MLM} + L_{NSP}$

$= 0.329 + 0.163 = \mathbf{0.492}$

💡 **Backpropagation:** Gradients flow back through all 12 layers. If prediction was wrong (e.g., predicted "jumped" instead of "sat"), the loss would be higher, causing larger gradient updates to better capture the cat-sitting relationship.

Gradient Update Example (simplified):

$\partial L/\partial W_{vocab}[\text{"sat"}] = H^{12}_{[MASK]} \times (P_{sat} - 1) = [0.89, -0.23, 0.67, 0.45] \times (0.72-1)$

$= [-0.25, 0.06, -0.19, -0.13]$

$W_{vocab}[\text{"sat"}]^{new} = W_{vocab}[\text{"sat"}]^{old} - \eta \times gradient$

$= [...] - 0.0001 \times [-0.25, 0.06, -0.19, -0.13]$

🔮 Inference Phase: Using Pre-trained BERT

After pre-training, BERT can be used for various downstream tasks. Here's how inference works:

A Fill-in-the-Blank (MLM-style Inference)

Input: "The weather today is [MASK]."

Step 1: Tokenize & Embed

```
tokens = ["[CLS]", "The", "weather", "today", "is", "[MASK]", ".", "[SEP]"]
input_ids = [101, 1996, 4633, 2651, 2003, 103, 1012, 102]
```

↓

Step 2: Forward Pass through 12 Transformer Layers

H^0 → Attention → FFN → LayerNorm → H^1 → ... → H^{12}

$H^{12}_{[MASK]}$ = [**0.76, 0.34, -0.52, 0.91**]

↓

Step 3: Vocabulary Projection → Top Predictions

#1: "sunny" (P=0.31) #2: "nice" (P=0.18) #3: "beautiful" (P=0.12) #4: "cold" (P=0.09)

B Sentiment Classification (Fine-tuned BERT)

Input: "I absolutely loved this movie!"

Step 1: Get [CLS] Token Representation

$H^{12}_{[CLS]}$ = [0.82, -0.45, 0.67, 0.23, ..., 0.91] (768 dimensions)

↓

Step 2: Classification Head (Linear Layer)

$$\text{logits} = H^{12}_{[CLS]} \cdot W_{classifier} + b$$
$$= [-1.23,\ 2.87] \quad \leftarrow [\text{Negative, Positive}]$$

↓

$$P = softmax([-1.23, 2.87]) = [0.02, 0.98]$$

Final Prediction

✅ Positive Sentiment (98% confidence)

C Named Entity Recognition (Token Classification)

Input: "Barack Obama visited Paris"

Each Token's Hidden State → Entity Label

$H^{12}_{Barack} \rightarrow W_{NER} \rightarrow$ logits → softmax → **B-PER** (0.94)

$H^{12}_{Obama} \rightarrow W_{NER} \rightarrow$ logits → softmax → **I-PER** (0.91)

$H^{12}_{visited} \rightarrow W_{NER} \rightarrow$ logits → softmax → **O** (0.97)

$H^{12}_{Paris} \rightarrow W_{NER} \rightarrow$ logits → softmax → **B-LOC** (0.89)

Extracted Entities

Barack Obama → PERSON Paris → LOCATION

Key Differences: Training vs Inference

🎓 Training

- Random 15% tokens masked
- Loss calculated & backpropagation
- Weights updated via optimizer
- Batch processing (256-512 samples)
- Dropout enabled (p=0.1)

🚀 Inference

- No masking (or intentional [MASK])
- Forward pass only
- Weights frozen (no updates)
- Single sample or small batches
- Dropout disabled (deterministic)

BERT Fine-tuning for Different Tasks

Pre-trained BERT (All Parameters)

+

Sequence Classification

Uses:

[CLS]

token representation

Sentiment Analysis, Text Classification

Token Classification

Uses:

T_1 T_2 T_3

each token's output

NER, POS Tagging

Question Answering

Predicts:

Start & End spans

SQuAD, Reading Comprehension

↓

End-to-End Fine-tuning

All parameters updated with task-specific data

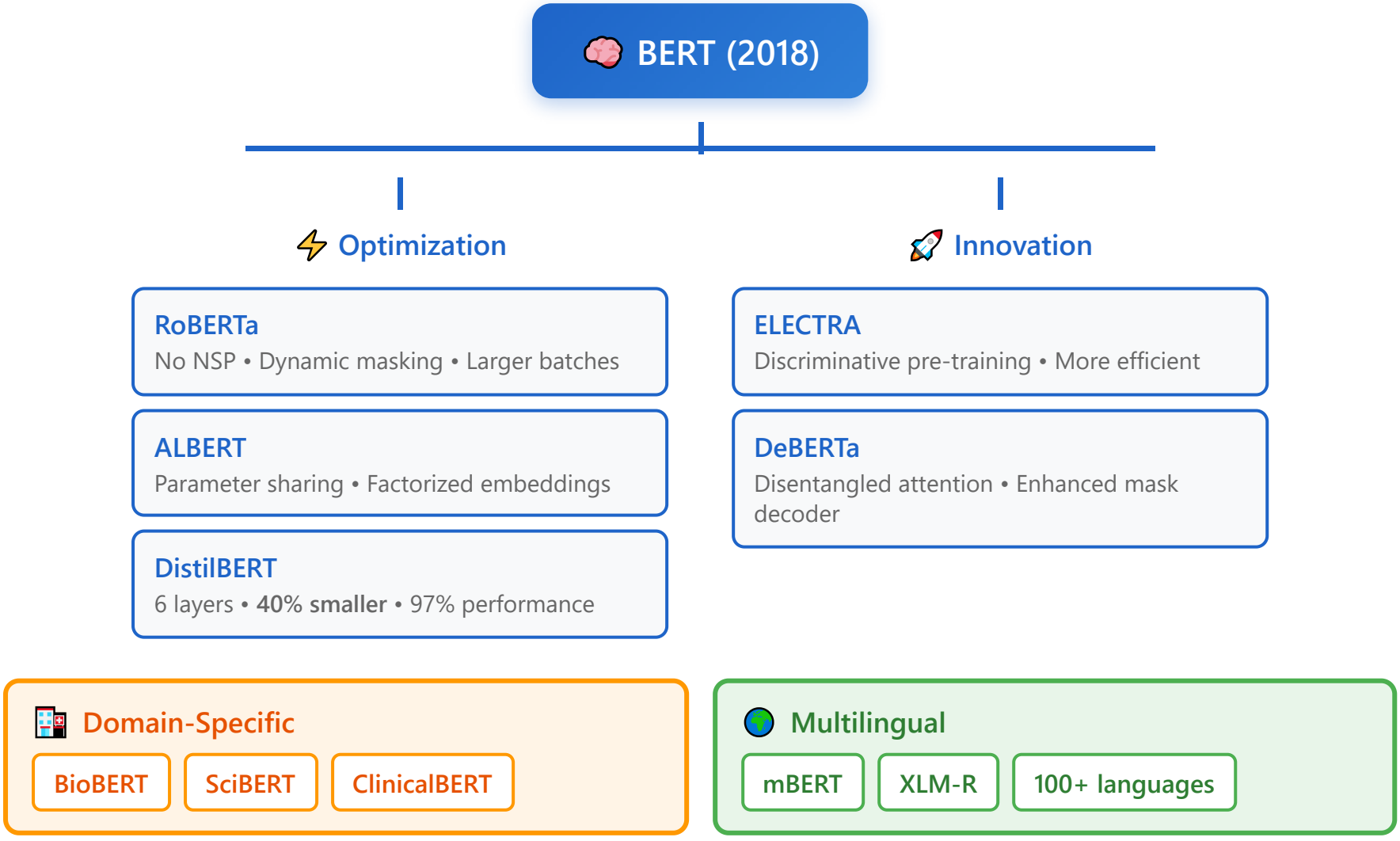
Fast Training	Task-Specific	Transfer Learning

Only **few epochs** needed

Add layer on **top of BERT**

Few-shot & zero-shot

BERT Family: Variants and Improvements



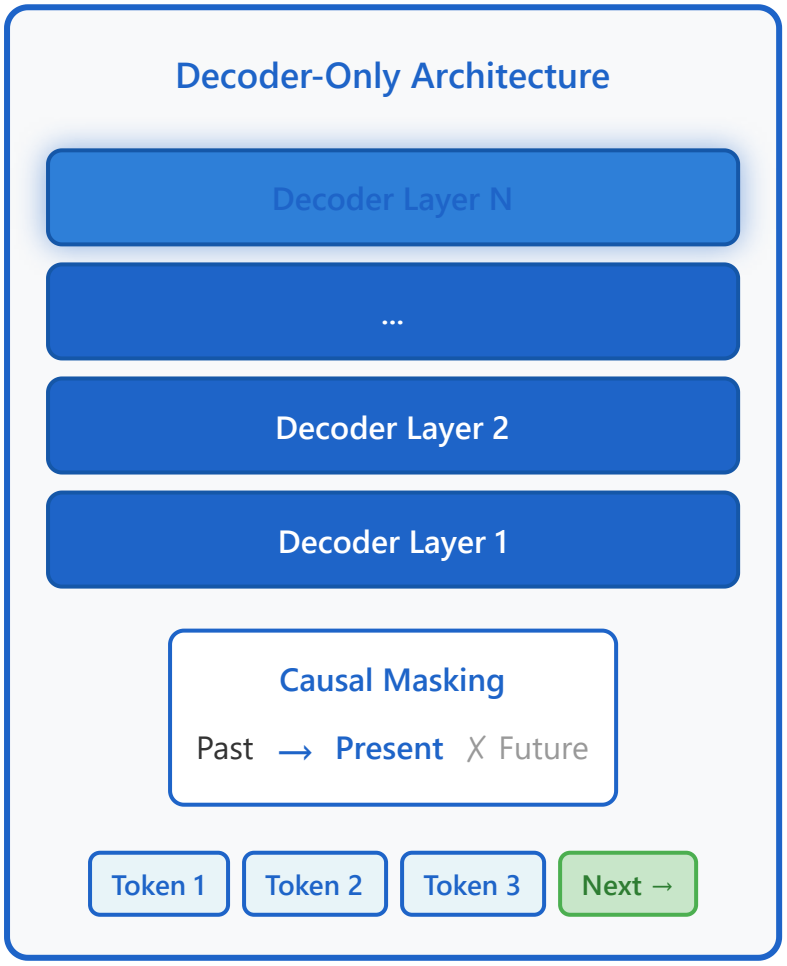
Part 4/9:

GPT - Decoder-based Models

10. Introduction to the GPT Series
11. GPT Pre-training
12. GPT-3 and Few-shot Learning
13. GPT Family Development

GPT: Generative Pre-trained Transformer

OpenAI - Autoregressive Decoder-only Architecture



Key Features

- → **Unidirectional** (left-to-right) attention
- Optimized for **text generation**
- **Autoregressive** prediction
- Decoder-only transformer stack

GPT Evolution

2018	**GPT-1**	117M params • Proof of concept
2019	**GPT-2**	1.5B params • Impressive generation
2020	**GPT-3**	175B params — Paradigm Shift

GPT Pre-training: Causal Language Modeling

Autoregressive Prediction Process

1. The → cat — *Predict next token*
2. The cat → sat — *Use previous outputs*
3. The cat sat → on — *Continue generation*

Training Data
Diverse internet text: **Common Crawl, WebText**

Unidirectional Attention
Enables **efficient generation**

BPE Tokenization
Byte-pair encoding for vocabulary

No Explicit Fine-tuning
GPT-3: **Direct inference** capable

Pattern Learning
Learns patterns, facts, and reasoning from data

Massive Scale
Unlocks in-context learning abilities

Efficiency
Fast generation through causal masking

🎓 Training Process (Pre-training)

1

Input Tokenization

Text is converted into tokens using vocabulary

```
"The cat sat" → [1045, 2635, 2068]
```

→

2

Forward Pass

Tokens pass through decoder layers with causal masking

```
Input → Embeddings → Decoder Stack → Logits
```

→

3

Predict Next Token

Model predicts probability distribution over vocabulary

$$P(\text{token}_4 \mid \text{token}_1, \text{token}_2, \text{token}_3)$$

→

4

Compute Loss

Compare prediction with actual next token using cross-entropy

```
Loss = CrossEntropy(predicted, actual)
```

→

5

Update Weights

Backpropagate gradients and update model parameters

$$\theta \leftarrow \theta - \alpha \nabla L(\theta)$$

🚀 Inference Process (Text Generation)

1

Input Prompt

User provides initial text prompt

```
"Once upon a time"
```

→

2

Tokenize & Encode

Convert prompt to token IDs and create embeddings

```
Tokens: [2949,
```

→

3

Autoregressive Generation

Generate one token at a time, appending each to input

```
Loop: predict → sample → append
```

→

4

Sampling Strategy

Use temperature, top-k, or top-p to select next token

```
Temperature = 0.7, Top-p = 0.9
```

→

5

Decode Output

Convert tokens back to text until stopping condition

```
Tokens → "there
```

2402, 257, 640]

lived a princess"

Step-by-Step Example

- Input: `"The weather is"`
- Step 1: `predict → "sunny"`
- Step 2: `"The weather is sunny" → predict → "today"`
- Step 3: `"The weather is sunny today" → predict → "."`
- Final Output: `"The weather is sunny today."`

GPT-3 and Few-shot Learning

175B parameters • 300B tokens • In-context Learning

0 Zero-shot

Task description only

Translate to French:

"Hello, how are you?"

→ Bonjour, comment allez-vous?

1 One-shot

Single example

Translate to French:

"Good morning" → "Bonjour"

"Thank you"

→ Merci

🎯 Few-shot

Multiple examples

Translate to French:

"Hello" → "Bonjour"
"Thank you" → "Merci"
"Goodbye" → "Au revoir"

"Please"

→ S'il vous plaît

In-context Learning

Learn from context **without weight updates**

Emergent Capabilities

Arithmetic, translation, **reasoning**

🚀 Democratization of AI

Natural language interfaces enable users to interact with AI without technical expertise

🔍 Understanding Few-shot Learning in GPT

What is Few-shot Learning?

Few-shot learning is a paradigm where a model learns to perform a new task from only **a handful of examples** provided in the input prompt. Unlike traditional machine learning that requires thousands of training samples, GPT-3 can understand and generalize from just **2-100 demonstrations** without any parameter updates.

How GPT Processes Few-shot Prompts

1. **Read Examples** — Model receives input-output pairs as context
2. **Pattern Recognition** — Attention identifies relationships between examples
3. **Task Inference** — Model infers the underlying task or rule
4. **Apply & Generate** — Applies learned pattern to new input

No Fine-tuning

The model weights remain frozen. Learning happens purely through context processing at inference time.

Scale Matters

Few-shot ability emerges with scale. Larger models (175B) dramatically outperform smaller ones in few-shot tasks.

Format Sensitivity

Performance depends heavily on prompt formatting and example selection—order and quality matter.

Why Does This Work?

- **Massive Pre-training:** GPT-3 has seen diverse text patterns during training on 300B tokens
- **Attention Mechanism:** Self-attention can dynamically route information from examples to queries
- **Implicit Meta-learning:** The model learns "how to learn" from examples during pre-training
- **Task Distribution:** Many tasks in pre-training data naturally appear in few-shot format

GPT Family Development Timeline

Evolution & Innovations

InstructGPT

RLHF for instruction following

ChatGPT

Conversation-optimized with safety alignment

GPT-4

Multimodal • Improved reasoning & reliability

Codex

Code generation • Powers GitHub Copilot

Open Source Alternatives

- GPT-Neo
- GPT-J
- Bloom
- LLaMA

Safety

Continuous improvements

Accuracy

Enhanced reliability

API Access

Widespread development

Part 5/9:

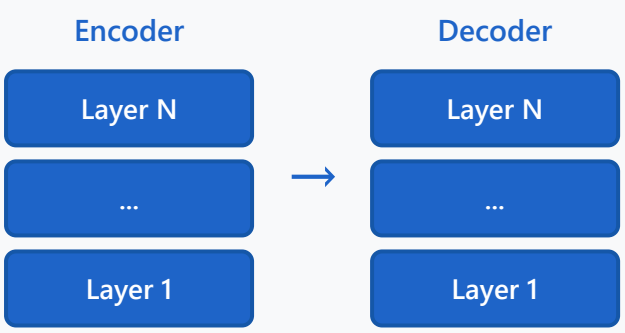
Encoder-Decoder Models

14. T5 - Text-to-Text Framework
15. BART and Other Models

T5: Text-to-Text Transfer Transformer

Google - Unified Framework for All NLP Tasks

Full Transformer



Pre-training

C4 corpus (750GB)
Span corruption objective

Scalability

T5-Base → T5-11B
Multiple model sizes

Performance

Strong results across diverse NLP tasks

All Tasks → Text Generation

translate English to German:
That is good. → Das ist gut.

summarize:
Long article text... → Brief summary

sentiment:
I love this movie! → positive

question answering:
What is the capital? → Paris

Model Sizes:

T5-Small | T5-Base | T5-Large | T5-3B | T5-11B

BART and Other Encoder-Decoder Models

BART: Denoising Autoencoder

Combines BERT (Encoder) + GPT (Decoder)

1. **Corrupt Input** — Add noise
2. **Encode** — BERT-style
3. **Decode** — GPT-style
4. **Reconstruct** — Original text

Token Masking
`The [MASK] sat`

Token Deletion
`The _ sat`

Span Infilling
`The [X] mat`

Sentence Permutation
`Sent2 Sent1 Sent3`

Document Rotation
`...end → start...`

Text Infilling
`A B [Y] E F`

mBART

Multilingual BART for **50+ languages**

PEGASUS

Gap sentence generation for summarization

Marian

Efficient **neural machine translation**

BART Strengths

Particularly strong for **generation & summarization**

Encoder-Decoder Models

Best suited for **sequence-to-sequence tasks** • Translation • Summarization • Question Answering

Part 6/9:

Fine-tuning Strategies

16. Full Fine-tuning vs. Transfer Learning
17. Parameter-Efficient Fine-tuning
18. Fine-tuning Practical Tips

Fine-tuning Strategies Comparison

Full Fine-tuning

Update all parameters

- Task Head ✓
- Top Layers ✓
- Middle Layers ✓
- Bottom Layers ✓

✓ Best performance
⚠ Slower training
⚠ Risk of forgetting

Feature Extraction

Freeze base model

- Task Head ✓
- Top Layers ✗
- Middle Layers ✗
- Bottom Layers ✗

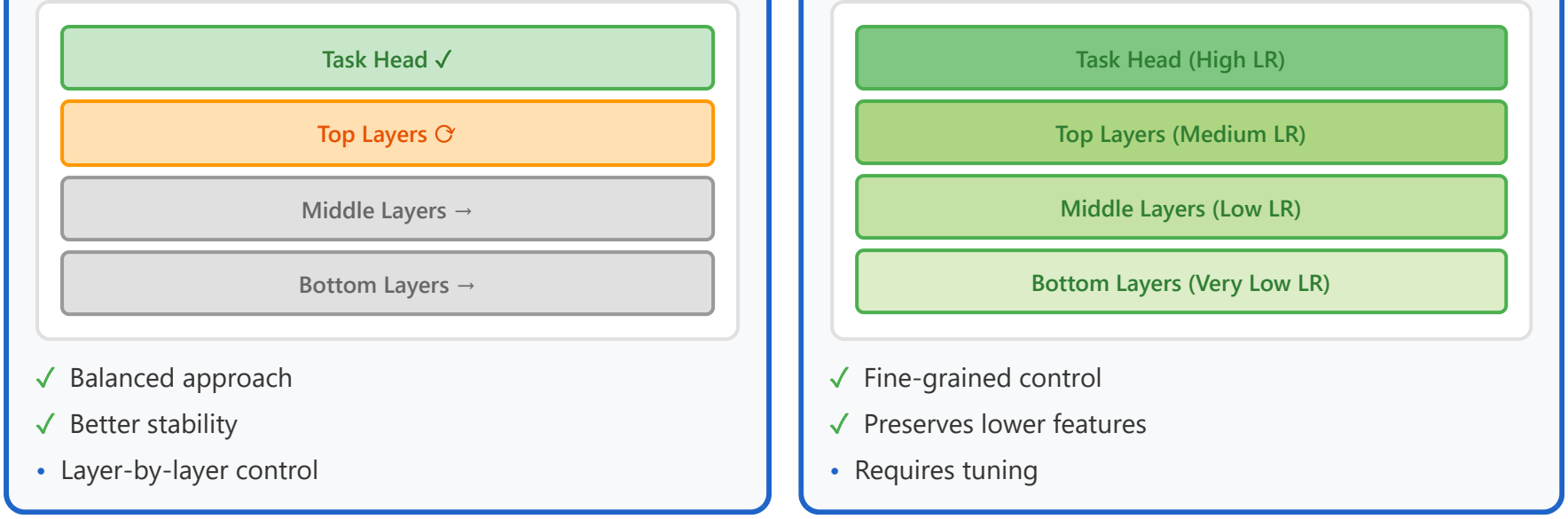
✓ Faster training
✓ No catastrophic forgetting
⚠ Lower performance

Gradual Unfreezing

Progressive unlocking

Discriminative LR

Different rates per layer



✓ Balanced approach
✓ Better stability
• Layer-by-layer control

✓ Fine-grained control
✓ Preserves lower features
• Requires tuning

🎯 Performance

Full FT: **Best**

Feature Ext: **Good**

⚡ Speed

Feature Ext: **Fastest**

Full FT: **Slowest**

💾 Memory

Feature Ext: **Low**

Full FT: **High**

💡 Decision Factors

Choice depends on **data size** and **computational budget** • Large data + resources → Full FT • Small data / limited resources → Feature Extraction

Parameter-Efficient Fine-tuning (PEFT)

LoRA

Low-Rank Adaptation

W (Frozen) + A × B

Low-rank matrices

$W' = W + AB$

<1% parameters trainable

Adapter Layers

Bottleneck Modules

Transformer Layer (Frozen)

↓

Adapter (Down → Up)

↓

Transformer Layer (Frozen)

↓

Adapter (Down → Up)

Small modules between layers

Prefix Tuning

Optimize **continuous prompt vectors**

Prompt Tuning

Learn **soft prompts**, freeze model

BitFit

Only tune **bias terms**, freeze weights

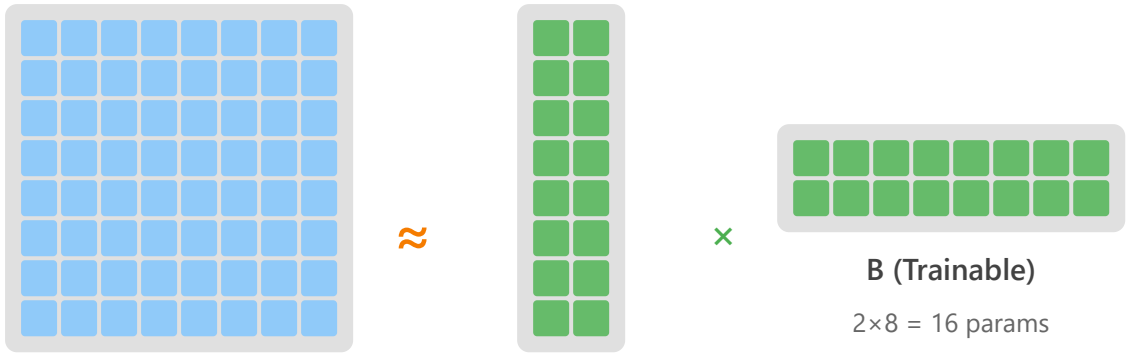
Single GPU Training

Fine-tune **large models** on limited hardware

High Efficiency

90-95% of full fine-tuning performance

Practical Example: LoRA Parameter Decomposition



W (Frozen)

8×8 = 64 parameters

A (Trainable)

8×2 = 16 params

B (Trainable)

2×8 = 16 params

64	32	50%
Original Parameters	LoRA Parameters (16 + 16)	Reduction

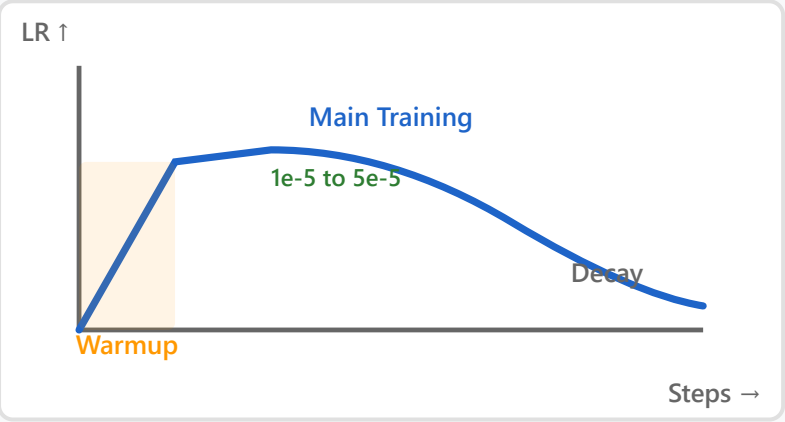
With Rank r = 2: (m×r) + (r×n) = (8×2) + (2×8) = 32

Lower rank = fewer parameters
(r=1: 16 params, r=2: 32 params, r=4: 64 params)

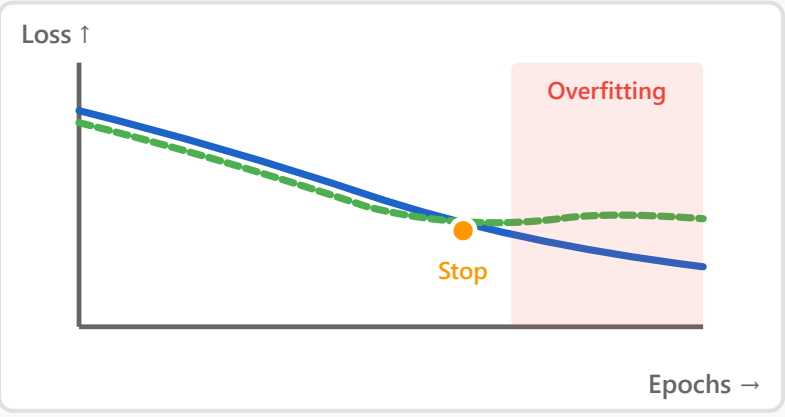
In large-scale models, over 99% parameter reduction is possible!

Fine-tuning Practical Tips

Learning Rate Schedule



Training & Validation Loss



1. **Learning Rate**
   Start with **1e-5 to 5e-5** range
2. **Warmup Steps**
   Use warmup to **stabilize training**
3. **Monitor Validation**
   Track validation loss to **prevent overfitting**
4. **Data Quality**
   Quality > Quantity for better results
5. **Batch Size**
   Affects **convergence & generalization**
6. **Early Stopping**
   Stop based on **validation metrics**
7. **Multi-task Learning**
   Consider for **related tasks**

Best Practice

Save checkpoints and test **multiple configurations** to find optimal settings

Part 7/9:

Prompting & In-Context Learning

19. Prompt Engineering Basics
20. Advanced Prompting Techniques
21. Fine-tuning vs. Prompting

Prompt Engineering: Good vs. Bad Examples

❌ Bad Prompts

📝 Example 1:

"Write something about AI."

⚠ Too vague

📝 Example 2:

✅ Good Prompts

📝 Example 1:

"You are an AI expert. Write a 3-paragraph explanation of machine learning for beginners, using simple analogies."

✓ Clear role & constraints

📝 Example 2:

"Summarize this article."

⚠ No context or format

📝 Example 3:

"Translate to French."

"Summarize the following article in 3 bullet points, focusing on key findings. Format: • Point 1\n• Point 2\n• Point 3"

✓ Specific format & context

📝 Example 3:

What is Prompt Engineering?

📖 Definition

Prompt Engineering is the art and science of **designing and optimizing input text (prompts)** to get desired outputs from AI language models (LLMs). It's not just about asking questions—it's about strategically structuring instructions so AI generates more accurate and useful responses.

🤔 Why is it Needed?

AI models provide completely different quality responses depending on **how you ask**:

- Vague question → Generic, superficial answer
- Specific question → Accurate, practical answer
- Contextual question → Tailored, relevant answer

💡 Core Meaning

Prompt engineering is **"the skill of effective communication with AI."** Just as you need to give clear instructions to people to get good results, you need the technique to precisely convey what you want to AI.

📅 Why Did It Emerge? (Background)

- **Early 2020s:** Large language models like GPT-3 emerged, enabling AI to perform diverse tasks
- **Problem arose:** Same model produced vastly different results based on how questions were asked → "How should I ask?"
- **Solution emerged:** Research on effective prompt writing became systematized, forming the field of "Prompt Engineering"
- **Today:** Established as a core AI skill, with "Prompt Engineer" emerging as a new profession

🎯 Key Takeaway

Prompt engineering is **the key to unlocking AI's full potential**.

Good Prompt = Good Results | Mastering AI = Mastering Prompts

Advanced Prompting Techniques

Chain-of-Thought (CoT)

Question:

Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 balls. How many tennis balls does he have now?

Prompt:

"Let's think step by step."

1. Roger starts with **5 tennis balls**
2. He buys **2 cans** of tennis balls
3. Each can has **3 balls**, so 2 × 3 = **6 balls**
4. Total: 5 + 6 = **11 balls**

✓ Final Answer:

Other Advanced Techniques

Few-shot Learning

Provide **multiple examples** in prompt for pattern recognition

Self-consistency

Sample **multiple outputs** and vote for most consistent answer

Tree of Thoughts

Explore **multiple reasoning paths** simultaneously

ReAct

Reasoning and acting in interleaved manner

Auto Prompt Optimization

Use **evolutionary methods** to improve prompts

11 tennis balls

🎯 **Meta-prompting**

Prompts that generate prompts for different tasks

Fine-tuning vs. Prompting: When to Use Each?

Prompting

- **Quick** - No training needed
- **Flexible** - Easy to iterate
- **API-based** - Simple access
- Pay per API call

✓ Ideal for:

Prototyping | Few examples | Varied tasks | Quick experiments

Fine-tuning

- **Better performance** - Task-specific
- **Consistent** - Reliable outputs
- **Private data** - Full control
- Training infrastructure needed

✓ Ideal for:

Production | Large datasets | Low latency | Custom domains

Requirements

Consider **task complexity** & goals

Data

Evaluate **dataset size** & quality

Resources

Balance **cost & infrastructure**

Hybrid Approach

Prompt-based fine-tuning (instruction tuning) combines both benefits

Future Trend

Prompting becoming more powerful with larger models

Part 8/9:

Current Trends

22. RLHF and Instruction Tuning
23. Present and Future

RLHF and Instruction Tuning

Reinforcement Learning from Human Feedback

Three-Stage RLHF Pipeline

1 Supervised Fine-tuning

Train on high-quality demonstration data

→

2 Reward Model

Learn human preferences from comparisons

→

3 PPO Training

Optimize policy using reward model

Helpfulness

Better at following instructions

✓

Truthfulness

More accurate and reliable

Safety

Reduces harmful outputs

Instruction Tuning

Train on **diverse task instructions** to improve generalization

Constitutional AI

Self-improvement through principles and guidelines

Key to Success

RLHF and Instruction Tuning are fundamental to the success of **ChatGPT** and other assistant models

Present and Future: Current Trends in LLMs

Multimodal
Text, image, audio, video integration

Mixture of Experts
Sparse activation for efficiency

Long Context
100K+ tokens processing

Better Reasoning
Math, logic, commonsense

Retrieval Augmentation
Combining **LMs with knowledge bases** for accurate information

Smaller Efficient Models
On-device and **edge deployment** capabilities

Enhanced Capabilities
Improvements in **mathematical reasoning**, logical thinking, and commonsense understanding

Context Processing
Processing **100K+ tokens** enables handling entire books and documents

Specialized Domain Models
Medical | Legal | Scientific | Finance | Engineering

AGI Considerations
Focus on **capabilities**, **safety**, and **alignment** as we approach AGI

Part 9/9:

Ethics and Practice

Summary and Practical Considerations

🚀 Pre-trained LLMs Revolutionized the NLP Landscape

Choose Your Architecture

- **Encoder** — BERT-style
- **Decoder** — GPT-style
- **Both** — T5-style

- **Bias & Fairness**
- **Privacy**
- **Misinformation**
- **Environment**

- **Diverse Evaluation** — Test on diverse datasets and scenarios
- **Document Limitations** — Clearly state model capabilities and constraints
- **Transparency** — Open about model behavior and decisions
- **Human Oversight** — Maintain human-in-the-loop for critical tasks

- **Environmental Cost** — **Training emissions** and energy consumption
- **Accessibility** — **Democratization** vs. resource concentration
- **Continuous Learning** — Field evolves **rapidly**, stay updated

Responsible Deployment

Transparency • **Safety measures** • **Human oversight** in all deployments

Thank you

Ho-min Park

homin.park@ghent.ac.kr

powersimmani@gmail.com