AIE1902: Exploring Language Models with AI

Xiaopeng Li
CUHK(SZ)
SAI

March 30, 2026

Outline

BERT: Bidirectional Encoder

GPT-2 & Prompting

LoRA & PEFT

Recap

We have studied many language models:

- **N-gram:** count-based, simple, no long-range context.
- **Word2Vec:** learned embeddings, similar words are close, static.
- **RNN/LSTM:** sequential models with context, slow and forgetful.
- **Transformer/GPT-1:** self-attention, parallelized, fast, powerful.

GPT-1:

- **Pretrain** a Transformer decoder as a language model for next-token prediction.
- **Fine-tune** on downstream tasks, e.g., classification, QA, etc.

Today: GPT-1 has a fundamental limitation. Can we do better?

The Limitation of Left-to-Right

GPT-1 is a decoder: each token can only attend to tokens *before* it.

Example: fill in the blank.

`I went to the bank to deposit my _____.`

Answer: "money" or "check". The left context alone is enough.

Example: fill in the blank.

`I sat on the bank of the _____ and watched the water.`

Answer: "river". But a left-to-right model sees "`I sat on the bank of the`" and don't know "bank" means river bank or financial bank.

The word "`water`" on the right differentiates it.

Problem: GPT-1's mask prevents each token from seeing the future. We lose right context, which is often critical for understanding.

BERT

If our goal is to understand language, e.g., classify, extract, Q&A, not to generate it, we want bidirectional context.

Solution: use the Transformer Encoder, not the Decoder.

	GPT-1 (Decoder)	**BERT (Encoder)**
Attention	left only	bidirectional (all tokens)
Each token sees	past tokens	all tokens
Good for	generation	understanding

Architecture: same Transformer building blocks you already know, except we remove the mask in masked self-attention.

New problem: if every token can see every other token, what do we train it to predict?

$$\text{BERT} \implies \text{Masked Language Modeling}$$

BERT Input Format

BERT takes a pair of sentences A and B as input.

Special tokens:

- `[CLS]`: placed at the beginning, for classification tasks later.
- `[SEP]`: placed between A and B, and after B, to mark boundaries.

Example:

`[CLS]` `my dog is cute` `[SEP]` `he likes playing` `[SEP]`

Three embeddings added together for each token:

- **Token embedding:** the meaning of the token itself (Word2Vec).
- **Segment embedding:** is this token in Sentence A or B?
- **Position embedding:** where is this token in the sequence?

$$x_t = E_{\text{token}}(t) + E_{\text{segment}}(t) + E_{\text{position}}(t)$$

Masked Language Modeling (MLM)

BERT's solution: hide some tokens and ask the model to recover them.

This is the **Cloze task** you did everyday in high school:

`I am _____ to the store but my car broke down.`

You can probably guess: "driving".

MLM procedure:

1. Randomly select 15% of all tokens in the input.
2. Replace them with a special `[MASK]` token.
3. Ask the model to predict the token at each masked position.

Example:

`[CLS] my dog is cute [SEP] he likes playing [SEP]`

`[CLS] my dog is [MASK] [SEP] he [MASK] playing [SEP]`

Predict: "cute", "likes".

MLM: The 80/10/10 Rule

Issue: at fine-tuning and test time, there is no `[MASK]` token in the input, but during pre-training, the model always sees `[MASK]`.

Solution: don't always replace with `[MASK]`.

Proportion	Action	Purpose
80%	Replace with `[MASK]`	The main training signal
10%	Replace with a random word	Force model to not rely on identity
10%	Keep unchanged	Teach model about correct tokens

Example: original token is "cute"

- 80%: `my dog is [MASK]` $\rightarrow$ predict "cute"
- 10%: `my dog is banana` $\rightarrow$ predict "cute"
- 10%: `my dog is cute` $\rightarrow$ predict "cute"

Effect: the model must learn good representations for every token, not just the masked ones. It shouldn't be sure which tokens are masked.

What Does MLM Teach?

A single objective, i.e., fill in the blank, teaches the model a lot, all using unlabeled text data.

Factual knowledge:

`CUHKSZ is located in [MASK], Shenzhen.`

Answer: "Longgang". The model learns real-world facts from text.

Syntax:

`I started [MASK] car and drove to the restaurant.`

Answer: "my" or "the". The model learns word categories and positions.

Coreference:

`I walked across the street, and checked the traffic over [MASK] shoulder.`

Answer: "my". The model learns that "`my`" refers back to "`I`". It captures long-range relationships between words.

Next Sentence Prediction (NSP)

MLM teaches word-level understanding, but some tasks need sentence-level understanding.

NSP task: given (A, B), predict: does B follow A in the original text?

Training data construction:

- **Positive (50%).** B is the actual next sentence after A.
- **Negative (50%).** B is a random sentence from the corpus.

Example:

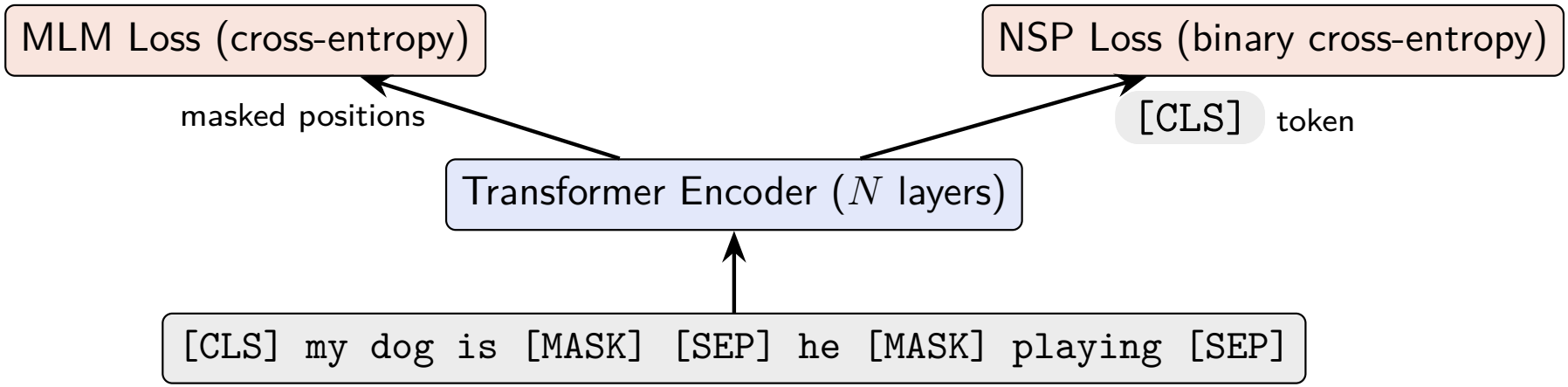
Positive: A = ``I went to the store.'' B = ``I bought some milk.'' $\rightarrow$ Label = 1

Negative: A = ``I went to the store.'' B = ``The sun is a star.'' $\rightarrow$ Label = 0

How? let the `[CLS]` token's output representation go through a linear head and use binary cross entropy loss.

BERT Pre-training: MLM+NSP

Two objectives, one model, no labeled data:



Total loss: $\mathcal{L} = \mathcal{L}_{\text{MLM}} + \mathcal{L}_{\text{NSP}}$.

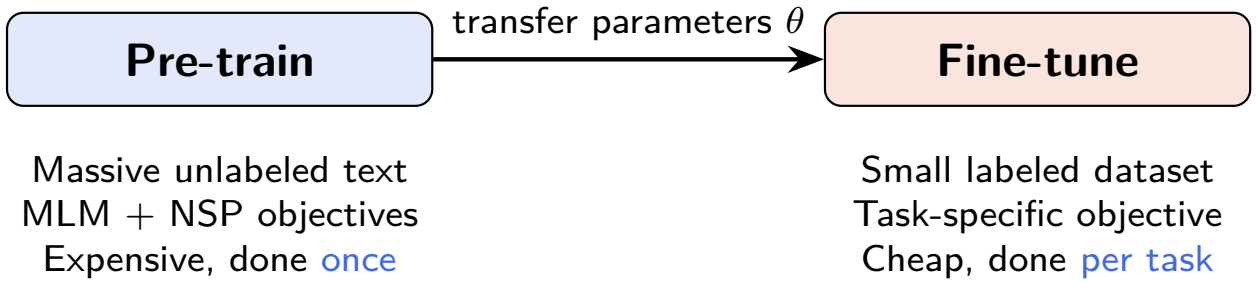
Pre-training setup:

- **Data:** BooksCorpus + English Wikipedia (∼3.3 billion words).
- **BERT$_{\text{BASE}}$:** 12 layers, $d_{\text{model}} = 768$, 12 heads, 110M paras.
- **BERT$_{\text{LARGE}}$:** 24 layers, $d_{\text{model}} = 1024$, 16 heads, 340M paras.
- **Tokenization:** WordPiece with 30,000 tokens.

Note: pre-training is expensive (days on many GPUs), but only done once. The result is a general-purpose language encoder.

From Pre-train to Fine-tune

The core idea behind BERT (GPT-1): two-stage training.



Analogy:

- **Pre-training** = going to school for years, learning to read and understand language in general.
- **Fine-tuning** = starting a new job, quickly adapting your general skills to a specific role.

Don't re-learn the language for every new job, instead, adapt what you already know.

Why Fine-tuning Works?

Without pre-training: randomly initialized using small dataset.

- The model knows nothing about language.
- Small dataset is easy to overfit, hard to generalize.

With pre-training: model has already seen billions of words.

- It already "knows" grammar, word meanings, world knowledge.
- Fine-tuning only makes small adjustments for the specific task.

Using pre-trained model:

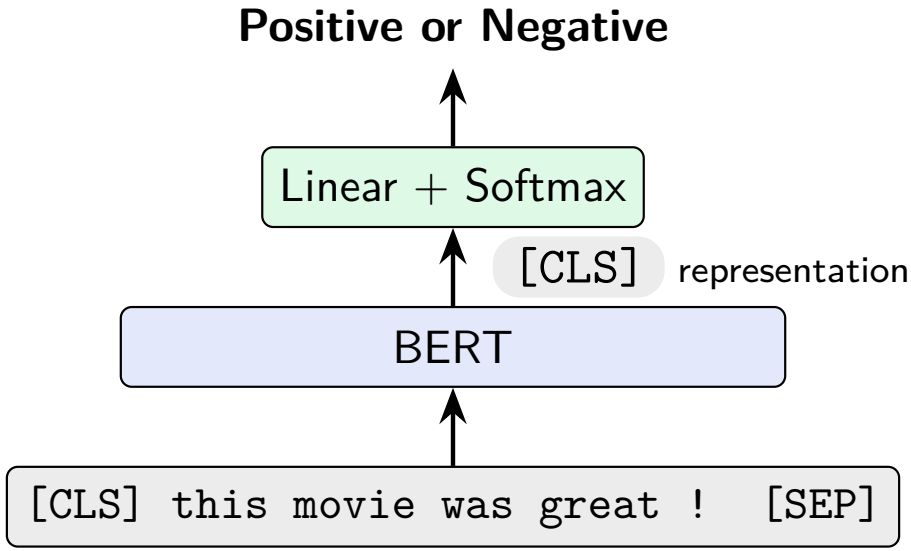
	What is trained	**Trade-off**
Feature extraction	only the new head	fast, but weaker
Full fine-tuning	head + all BERT layers	slower, but stronger

BERT uses full fine-tuning starting from pretrained weights.

BERT Fine-tuning for Classification

Task: given a sentence, predict a label.
Example: "This movie was great!" $\Longrightarrow$ **Positive**.

How? Use the `[CLS]` token's output as a summary of the whole input.



Formally: let C be the number of classes, $h_{\texttt{[CLS]}}$ is BERT's output at `[CLS]` position,

$$\hat{y} = \text{softmax}(W \cdot h_{\texttt{[CLS]}} + b), \quad W \in \mathbb{R}^{C \times d_{\text{model}}}.$$

BERT Fine-tuning for Named Entity Recognition

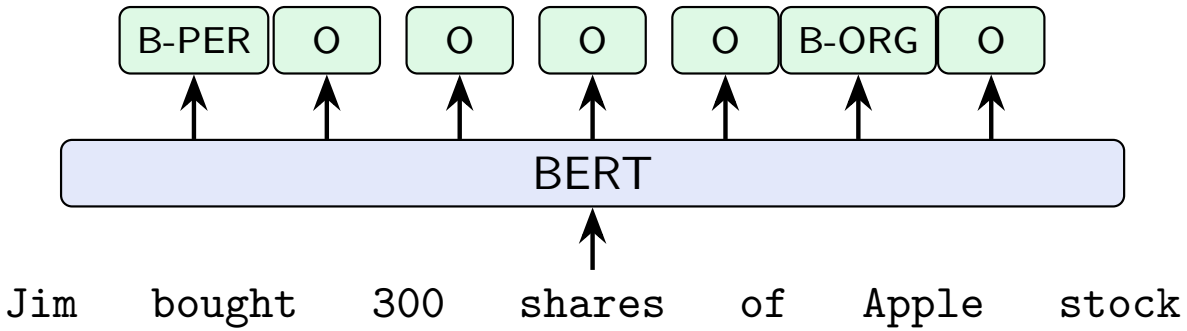
Task: label each token in a sentence with an entity type.

Example:

Jim	bought	300	shares	of	Apple	stock
B-PER	O	O	O	O	B-ORG	O

Tags: B-PER = beginning of person, B-ORG = beginning of organization, O = outside (not an entity).

How? Use every token's output representation instead of `[CLS]`.



Formally: for each token t, apply a linear head to its output:

$$\hat{y}_t = \text{softmax}(W \cdot h_t + b), \quad W \in \mathbb{R}^{C \times d_{\text{model}}},$$

where C is the number of entity tags.

BERT Fine-tuning for Question Answering

Task: given a question and a passage, find the span in the passage that answers the question.

Example:

- **Passage:** "The Transformer was introduced in 2017 at Google."
- **Question:** "When was the Transformer introduced?"
- **Answer:** "2017", i.e., tokens from position 6 to position 6.

Input format:

`[CLS]` `When was the Transformer introduced ?` `[SEP]` `The Transformer was introduced in 2017 at ...` `[SEP]`

How? The model predicts the start and end positions.

For each token t in the passage, compute:

$$p_{\text{start}}(t) = \text{softmax}(S \cdot h_t), \qquad p_{\text{end}}(t) = \text{softmax}(E \cdot h_t),$$

where $S, E \in \mathbb{R}^{d_{\text{model}}}$ are two learned vectors, the only new parameters.

BERT Results

BERT achieved state-of-the-art on a wide range of benchmarks:

Task	Type	BERT improved?
MNLI	sentence pair classification	✓
SST-2	sentiment analysis	✓
SQuAD 1.1	question answering	✓
CoNLL NER	named entity recognition	✓

Bigger is better:

	$\text{BERT}_{\text{BASE}}$	$\text{BERT}_{\text{LARGE}}$
Parameters	110M	340M
Layers	12	24
MNLI accuracy	84.6%	86.7%

BERT Summary

The big picture: one pre-trained model, fine-tuned with minimal changes, beats task-specific models across the board. The pre-train to fine-tune paradigm became the new standard in NLP.

Architecture: Transformer Encoder (bidirectional self-attention).

Pre-training: MLM + NSP.

Fine-tuning: add a small head, adapt to any task.

	GPT-1	**BERT**
Architecture	Decoder	Encoder
Context	left only	bidirectional
Pre-training	next token prediction	MLM + NSP
Downstream	fine-tune	fine-tune
Strength	generation	understanding

Outline

BERT: Bidirectional Encoder

GPT-2 & Prompting

LoRA & PEFT

Back to the Decoder

BERT: Encoder + fine-tuning is powerful for understanding tasks.

GPT-1: a Decoder trained to predict the next token, also fine-tuned for downstream tasks.

But the GPT-2 authors asked a different question:

> What if we just make the model much bigger and train it on much more data?

Hypothesis: a large enough language model will learn to perform tasks implicitly, without any fine-tuning at all.

You just need to prompt it in the right way.

From GPT-1 to GPT-2

Same fundamental architecture. Just bigger in every dimension:

	GPT-1	GPT-2
Parameters	117M	1.5B
Layers	12	48
d_{model}	768	1600
Context window	512	1024 tokens
Vocabulary	40K (BPE)	50K (BPE)
Training data	BookCorpus (5GB)	WebText (40GB)

Minor architectural difference:

- Layer normalization moved before attention and feed-forward:

  GPT-1: $\text{LN}(x + f(x))$ GPT-2: $x + f(\text{LN}(x))$

- An additional layer norm added after the final layer.

Zero-Shot

GPT-2's hypothesis: a large enough LM learns tasks implicitly during pre-training. You don't need to fine-tune, just prompt it.

Why? The training data naturally contains patterns like:

- Articles that summarize other articles
- Translated sentences appearing side by side
- Questions followed by answers

By predicting the next token across all this text, the model accidentally learns how to do these tasks.

Zero-shot: give the model a task description as input, get the answer as output. No gradient updates. No labeled data. No task-specific head.

GPT-1: studies a textbook, finish sample exams, and take true exams.
GPT-2: read the entire internet and answer exams without preparation.

Prompting

Idea: format your task as a text completion problem.

Translation:

`translate English to French:`
`cheese` $\Rightarrow$ `fromage`

Give examples in the prompt, and GPT-2 continues the pattern.

Question answering:

`Answer the question based on the context.`
`Context: The Transformer was introduced in 2017.`
`Q: When was the Transformer introduced?`
`A: 2017`

Summarization:

`[long article text ...]`
`TL;DR: The article discusses ...`

Decoding

Greedy decoding: always pick the highest-probability token.

$$y_t = \arg\max_y p(y \mid y_1, \ldots, y_{t-1})$$

Problem: the generated text becomes repetitive and boring.

```
Prompt:  The city was beautiful.
Output:  The city was beautiful.  The city was beautiful.  The
         city was beautiful.  The city was ...
```

Why? The generated words become part of the context for the next step. The distribution $p(y_t \mid y_1, \ldots, y_{t-1})$ happens to assign high probability to the same phrase again. Greedy decoding always picks the top choice, so the model gets stuck in a loop.

How do we sample from the distribution in a smarter way?

Top-K and Nucleus Sampling

Top-K sampling: at each step, keep only the K most probable tokens, zero out the rest, re-normalize, then sample.

Example: vocabulary has 50,000 tokens, but with $K = 10$:

$$\underbrace{p(\text{“river”}) = 0.3,\ \ p(\text{“lake”}) = 0.2,\ \ \ldots}_{\text{top 10 tokens, re-normalized}} \qquad \underbrace{p(\text{all others}) = 0}_{\text{49,990 tokens removed}}$$

Problem: A fixed K cannot adapt to confident and uncertain situation.

Nucleus sampling: fix a probability threshold p (e.g., $p = 0.95$) and keep the smallest set of tokens whose cumulative probability $\geq p$.

	Top-K $(K = 10)$	**Nucleus** $(p = 0.95)$
Confident distribution	keeps 10 tokens	keeps 2 or 3 tokens
Flat distribution	keeps 10 tokens	keeps 50+ tokens

Nucleus sampling produces text that is diverse yet coherent, closest to human writing in experiments.

GPT-2 Results

Zero-shot results:

Task	Performance	Impressive?
Language modeling	SOTA on 7 of 8 benchmarks	yes
Reading comprehension	better than random, not great	somewhat
Translation (En $\rightarrow$ Fr)	5 BLEU (trained models: 40+)	weak
Summarization	better with "TL;DR" prompt	somewhat

The numbers look mediocre, but the fact that a language model can do different tasks with zero task-specific training, was the breakthrough.

Scaling: larger GPT-2 models consistently performed better, and performance never saturated. It was still underfitting.

What if we make it even bigger? $\Longrightarrow$ GPT-3, GPT-4, ...

BERT vs GPT

Difference:

	BERT	GPT-2
Architecture	Encoder	Decoder
Context	bidirectional	left-to-right
Pre-training	MLM + NSP	next token prediction
Adaptation	fine-tune on labeled data	prompt (zero-shot)
Strength	understanding	generation
Parameters	110M or 340M	1.5B

Common:

1. Pre-train on massive unlabeled text.
2. Adapt to specific tasks by fine-tuning or prompting.

Lesson: the pre-train to adapt paradigm works. The question is how to adapt most efficie ntly. We need efficient fine-tuning!

Outline

BERT: Bidirectional Encoder

GPT-2 & Prompting

LoRA & PEFT

The Cost of Fine-tuning

How many parameters do we need to update?

Model	Parameters	Storage per copy
$\text{BERT}_{\text{BASE}}$	110M	∼0.4 GB
$\text{BERT}_{\text{LARGE}}$	340M	∼1.3 GB
GPT-2	1.5B	∼6 GB
GPT-3	175B	∼700 GB

Problem: suppose you want to fine-tune for 10 different tasks.

- For GPT-3: $10 \times 700\text{GB} = 7\text{TB}$ of storage!
- Each fine-tuning run is also expensive in compute.

Question: do we really need to update *all* parameters?

Idea: what if we could fine-tune by changing only a tiny fraction of the parameters? Most of the knowledge is already in the pre-trained model.

LoRA: Low-Rank Adaptation

Key observation: during fine-tuning, the weight change $\Delta W = W_{\text{fine-tuned}} - W_{\text{pre-trained}}$ tends to be low-rank.

Idea: Represent the change as a product of two smaller matrices:

$$W_{\text{new}} = W + \Delta W = W + BA$$

where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times d}$, and $r \ll d$.

Freeze W and only train B and A.

Example: for a weight matrix in $\text{BERT}_{\text{BASE}}$ with $d = 768$:

	Parameters
Original W	$768 \times 768 = 590\text{K}$
LoRA with $r = 8$: $B + A$	$768 \times 8 + 8 \times 768 = 12\text{K}$

That is only $\sim 2\%$ of the original parameters!

LoRA: Why Does It Work?

Extremely few trainable parameters.

- Typically $< 1\%$ of the original model.
- Training is much faster and uses much less memory.

No extra inference cost.

- After training, merge: $W_{\text{new}} = W + BA$.
- At inference time, it is the same as not using LoRA.

Easy task adaption.

- Store one copy of the base model W.
- Store one small B, A pair per task.
- Adapt to different tasks by changing B, A.

LoRA achieves comparable performance to full fine-tuning (sometimes even better), while training only $\sim$0.2% of the parameters.

Summary

BERT (Encoder):

- Bidirectional self-attention: every token sees all other tokens.
- Pre-trained with MLM + NSP.
- Fine-tune with a small task-specific head for various tasks.
- The pre-train to fine-tune paradigm is standard now.

GPT-2 (Decoder):

- Causal language model, scaled up from GPT-1.
- A large enough model can do tasks via prompting alone.
- Better decoding: nucleus sampling produces more natural text.
- Performance scales with model size, bigger is better.

LoRA (Efficient adaptation):

- Full fine-tuning is expensive.
- LoRA: freeze W, only train small B, A where $W_{\text{new}} = W + BA$.
- $< 1\%$ parameters, low computation cost, easy task adaption.

Next Week

Thank you for your attendance!

Questions?

Next week (Lab session):

- Use GPT-2 for spam classification on SMS messages.
- Tokenize text with `GPT2Tokenizer`, set up `GPT2ForSequenceClassification`
- Compare two strategies: **full fine-tuning** vs. **feature extraction**.
- Evaluate with accuracy, precision, recall, AUROC.