SCALER School of Technology × Priyansh Saxena

Transformers in Practice: BERT, GPT, and Beyond

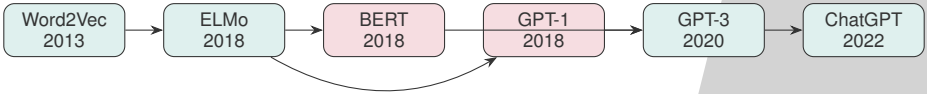
Pre-training • BERT • GPT • Tokenization • Practical Considerations

Priyansh Saxena

Scaler School of Technology | Deep Learning I: Neural Networks | Week [illegible] Session 2/2

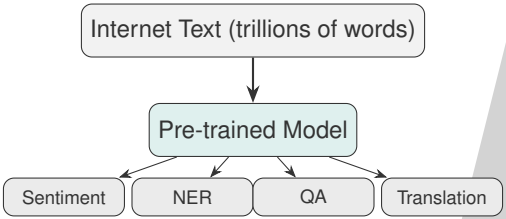
10 June 2026

The NLP Revolution



- **Before 2018:** Every NLP task needed its own model trained from scratch
- **After 2018:** Pre-train once, fine-tune for *any* task
- This is as big as ImageNet was for vision — the moment everything changed

The Pre-training Paradigm



Analogy: Medical school (8 years, expensive, once) $\rightarrow$ Residency (1 year, cheap, per specialty)

Why Pre-train?

Labeled data:

- Expensive (humans annotate)
- Small (thousands of examples)

Unlabeled text:

- Free (the internet)
- Massive (billions of words)

Language knowledge **transfers**: grammar, word meanings, world facts — all reusable across tasks.

Key Insight

If you can fill in blanks correctly, you understand language.

Two teams had this insight simultaneously:

- Google → **BERT**
- OpenAI → **GPT**

Two Philosophies, One Transformer

BERT (Google)

See everything
Understand deeply

Bidirectional

Open-book exam

GPT (OpenAI)

See the past
Predict the future

Left-to-right

Writing an essay

Same Transformer, **different attention masks**.

BERT = better at **understanding**. GPT = better at **generating**.

BERT — Bidirectional Encoder Representations

- Uses **only** the Transformer encoder (no decoder)
- Key insight: to understand a word, you need **both** sides of context

Why Bidirectional Matters

- "The **bank** was flooded" $\rightarrow$ river bank (need "flooded" on the right)
- "The **bank** approved the loan" $\rightarrow$ financial (need "loan" on the right)

Without right context, you cannot disambiguate.

Masked Language Modeling (MLM)

Predict: “cat”, “mat” using **all** surrounding context

Training procedure:

- Randomly mask 15% of tokens
- Predict masked tokens from context
- Forces bidirectional understanding

Analogy: A fill-in-the-blank exam. You need to read the *entire* sentence to answer correctly.

MLM — The 15% Rule

Of the 15% selected tokens:

Treatment	Proportion	Reason
Replace with [MASK]	80%	Main training signal
Replace with random word	10%	Prevent model from ignoring non-[MASK]
Keep unchanged	10%	Teach model that any word could be target

- Why not 100% [MASK]? At fine-tuning time, there are no masks!
- 15% rate: balance between enough signal and enough visible context

BERT Architecture

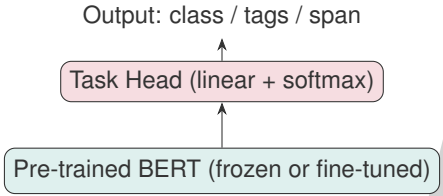
	BERT-base	BERT-large
Layers	12	24
Hidden size	768	1024
Attention heads	12	16
Parameters	110M	340M

- Pre-trained on Wikipedia + BookCorpus
- Training: 4 days on 64 TPUs — “you do NOT train this yourself”
- You **download** it. HuggingFace has it ready to go.

Also trained with Next Sentence Prediction (NSP) — later shown to be less important (RoBERTa dropped it).

Fine-tuning BERT

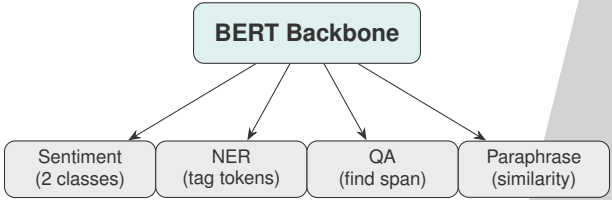
Take pre-trained BERT, add a **small output head** on top:



- **Classification:** [CLS] $\rightarrow$ linear $\rightarrow$ softmax
- **NER:** each token $\rightarrow$ label
- **QA:** predict start & end positions

- Learning rate: 2×10^{-5}
- Epochs: 3–5
- Hardware: 1 GPU, a few hours

One Model, Many Tasks



The Revolution

One expensive pre-training $\rightarrow$ cheap task adaptation.
With 1000 labeled examples and 3 epochs, achieve 85%+ accuracy.
Training from scratch would need 100$\times$ more data.

Quick Check — BERT

1. Why does BERT mask **randomly** instead of always masking the last word?
2. Can BERT generate text? Why or why not?

Quick Check — BERT

1. Why does BERT mask **randomly** instead of always masking the last word?
2. Can BERT generate text? Why or why not?

Answers

1. Random masking forces **bidirectional** context usage. Always-last = left-to-right only (becomes GPT!)
2. **No** — BERT sees all positions simultaneously. To generate word 5, it would peek at word 6+. No causal ordering.

Coming Up Next

Part 2: GPT, Scaling Laws, and Tokenization

GPT — Generative Pre-trained Transformer

- Uses **only** the Transformer decoder
- Autoregressive: predict next token given all previous
- Causal mask: position i can only attend to $1 \ldots i$

Core idea: your phone's autocomplete — predict the next word.

"GPT wears blinders — it can only look at what came before."

Predict: "Paris"

Autoregressive Generation

Training: predict next token at every position (teacher forcing)

Generation:

1. Predict next token distribution
2. Sample from distribution
3. Append sampled token
4. Repeat until stop token

Every word ChatGPT produces is generated this way — **one token at a time**.

Temperature

- Low (0.1): deterministic, safe
- High (1.5): creative, risky

Top-k / Top-p

Only sample from the most likely candidates — prevents nonsense.

GPT Scaling — Emergent Abilities

Model	Parameters	Key Achievement
GPT-1 (2018)	117M	Proved the concept
GPT-2 (2019)	1.5B	“Too dangerous to release”
GPT-3 (2020)	175B	Few-shot learning **emerges**
GPT-4 (2023)	$\sim$1.7T (est.)	Multimodal, powers ChatGPT

Key Observation

Abilities **emerge** at scale that weren’t explicitly trained.
Nobody programmed GPT-3 to do arithmetic or translate — it just learned.

Few-Shot Learning — No Weight Updates!

GPT-3 surprise: give examples in the prompt $\rightarrow$ it follows the pattern.

Zero-shot:
"Translate English to French: Hello $\rightarrow$"

One-shot:
"Hello $\rightarrow$ Bonjour. Goodbye $\rightarrow$"

Few-shot:
5 examples $\rightarrow$ new query

No weight updates! Pattern recognition purely in context.

Implication

This is why **prompt engineering** matters — you're programming with examples.

BERT vs GPT — Head to Head

	BERT	GPT
Architecture	Encoder-only	Decoder-only
Context	Bidirectional	Left-to-right
Training	MLM (fill blanks)	Next-token prediction
Strength	Understanding	Generation
Use cases	Classification, NER, QA	Chatbots, writing, code

Both are Transformers. The **only** difference is the attention mask.

Tokenization — Text to Numbers

Models need numbers, not text. Three strategies:

Word-level
“unhappiness” = 1 token
Problem: vocabulary explodes

Character-level
11 separate tokens
Problem: too long

Subword (BPE)
“un” + “happi” + “ness”
Best of both worlds

Goldilocks Solution

Common words stay whole. Rare words split into reusable pieces.
“un-” and “-ness” appear in hundreds of words.

Byte Pair Encoding (BPE)

1. Start with character vocabulary
2. Count all adjacent character pairs in corpus
3. Merge the most frequent pair into a new token
4. Repeat N times (N = desired vocab size $-$ initial chars)

l, o, w, e, r $\rightarrow$ “lo” (most frequent)

lo, w, e, r $\rightarrow$ “low” (next)

low, e, r $\rightarrow$ “lower” (next)

- Result: common words stay whole, rare words split into frequent subwords
- GPT-2/3/4 all use BPE with $\sim$50K vocabulary

WordPiece (BERT's Tokenizer)

- Similar to BPE but uses **likelihood-based** merging
- "##" prefix marks continuation tokens:

"playing" $\rightarrow$ ["play", "##ing"]

- Vocabulary: 30K tokens for BERT
- Different algorithm, same idea — subword tokenization

BPE vs WordPiece

- BPE: frequency-based
- WordPiece: likelihood-based
- Both produce subwords
- In practice: similar results

Why Tokenization Matters

Why LLMs Can't Do Math

"427" might tokenize as "42" + "7" — the model literally cannot see individual digits!
"15213" $\rightarrow$ ["152", "13"] — this is why LLMs hallucinate on arithmetic.

Multilingual Cost

Same sentence in Japanese uses 3$\times$ more tokens than English.
More tokens $=$ more expensive, shorter effective context.

"The tokenizer is the most overlooked part of the system — but it determines what the model can even see."

Quick Check — GPT & Tokenization

1. Why do LLMs struggle with simple arithmetic?
2. What's the ONE architectural difference between BERT and GPT?

Quick Check — GPT & Tokenization

1. Why do LLMs struggle with simple arithmetic?
2. What's the ONE architectural difference between BERT and GPT?

Answers

1. Numbers get tokenized **inconsistently** — model doesn't see individual digits
2. The **attention mask**. BERT = full (bidirectional). GPT = causal (left-to-right only).

Memory & Sequence Length

Self-attention: $O(n^2)$ memory and compute

Model	Max Tokens
BERT	512
GPT-3	2,048
GPT-4	128,000

Double sequence length $\rightarrow$ **4**$\times$ memory for attention

Solutions

- FlashAttention (engineering)
- Sparse attention (algorithmic)
- Linear attention (research)

API pricing is per-token — longer prompts cost quadratically more to process.

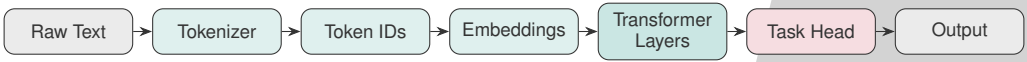
When to Use What — Decision Guide

Task	Model Family
Classification / NER / QA	BERT (RoBERTa, DeBERTa)
Generation / Dialogue	GPT family
Small dataset	Always fine-tune, never scratch
Domain-specific	BioBERT, CodeBERT, etc.
Budget-constrained	DistilBERT (66M, 97% quality)

Discussion Question

You have 1000 customer emails to classify as urgent/normal.
Train from scratch or fine-tune? Which model?

The Full NLP Pipeline



- This is **the** modern NLP stack
- Every chatbot, search engine, and AI assistant uses this pipeline
- Same pipeline whether BERT or GPT — just different Transformer variant
- The innovation is in the Transformer layers and how you pre-train them

Recap & Key Takeaways

- **Pre-training + Fine-tuning:** Learn language once, specialize cheaply
- **BERT:** Bidirectional, MLM, best for understanding tasks
- **GPT:** Autoregressive, next-token prediction, best for generation
- **Tokenization:** BPE/WordPiece — subword is the standard
- **Practical:** $O(n^2)$ attention, choose model by task type

Everything in modern NLP is a variation of these two ideas.

Next week: We've built the full stack — CNNs, RNNs, Transformers.
Next: what else can neural nets do? *Generative models.*

Thank You!

Questions?

Priyansh Saxena

All lecture materials available at:

www.priyanshsaxena.com/sst-deep-learning-nn