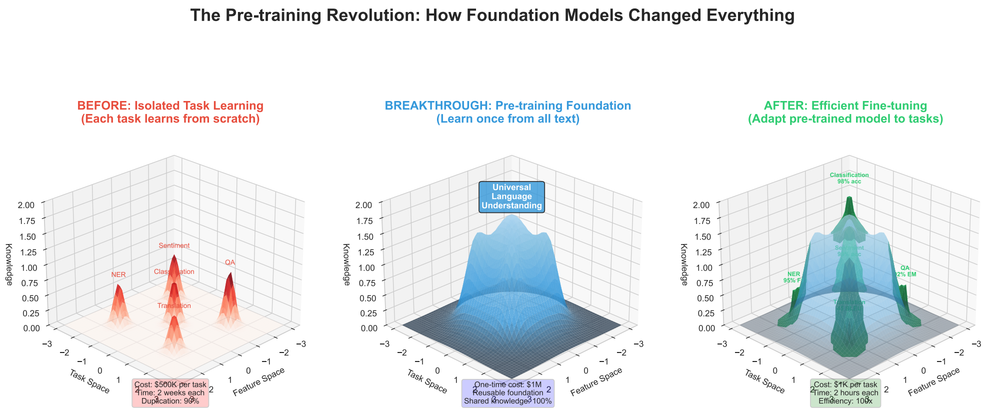
Natural Language Processing

Week 6: Pre-trained Language Models in 4 Parts

Joerg R. Osterrieder
www.joergosterrieder.com

Roadmap: Your Journey Through Pre-training

The Paradigm Shift That Changed Everything



Part 1

The Revolution

Why Pre-training Changed Everything

The $10 Million Waste Problem

Before 2018: Every Team Starting from Zero

Company A: Sentiment Analysis

- Cost: $500,000 in compute
- Time: 2 weeks on 64 GPUs
- Learns: Grammar, syntax, sentiment

Company B: Question Answering

- Cost: $500,000 in compute
- Time: 2 weeks on 64 GPUs
- Learns: Grammar, syntax... again!

The Insanity:

- 20 companies = $10 million wasted
- Each re-learning what "the" means
- 90% duplicate effort
- Only 10% on actual task

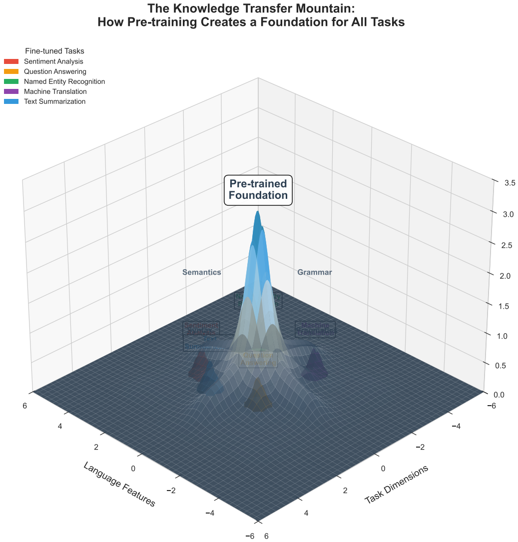
Reality Check:
Imagine teaching every medical student the alphabet before medicine!

2012: ImageNet Moment in Vision

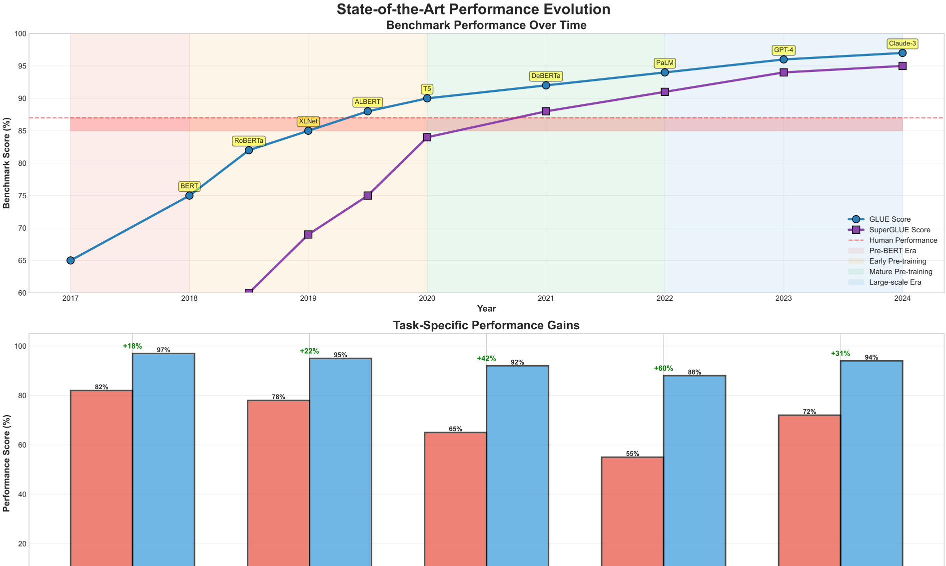
- Pre-train on millions of images
- Learn edges, shapes, objects
- Fine-tune for specific tasks
- 10x performance improvement

2018: The NLP Awakening

- "Why not do this for language?"
- Pre-train on all of Wikipedia
- Learn grammar, facts, reasoning
- Fine-tune for any language task



Timeline: The Pre-training Revolution



The Scale of Transformation

Before Pre-training (2017)

- Training cost: $500K per task
- Training time: 2 weeks
- Data needed: 100K+ labeled examples
- Performance: 70-80% accuracy
- Accessibility: PhD required
- Deployment: Months

With Pre-training (2024)

- Training cost: $1K per task
- Training time: 2 hours
- Data needed: 1K examples
- Performance: 95%+ accuracy
- Accessibility: API call
- Deployment: Minutes

Impact: 100x cost reduction, 100x speed increase, 10x performance gain

Democratization: From Elite Labs to Everyone

The Old World (Pre-2018)

- Only Google, Facebook, Microsoft
- Requires ML PhD team
- Millions in infrastructure
- Months to deploy

The New World (Post-2018)

- Any developer with an API key
- No ML expertise needed
- $20/month subscription
- Deploy in minutes

Who Benefits:

- Startups
- Students
- Researchers
- Small businesses
- Non-profits
- Individual developers

Checkpoint

Can you explain why pre-training is like teaching someone to read?

Part 1 Summary: The Revolution

Key Takeaways:

1. **The Problem**: Every team re-learning language basics = \$10M waste
2. **The Insight**: Pre-train once on everything, fine-tune for specific tasks
3. **The Impact**: 100x efficiency gain across cost, time, and accessibility
4. **The Timeline**: 2018 BERT $\rightarrow$ 2023 ChatGPT = 5 years to change the world
5. **The Democratization**: From PhD labs to every developer

Next: How do models actually learn from raw text?

Part 2

Understanding Pre-training

How Models Learn from Raw Text

The Magic: Self-Supervised Learning

Traditional Supervised Learning

- Need: (text, label) pairs
- Example: ("I love this!", positive)
- Problem: Expensive to label
- Scale: Thousands of examples

Self-Supervised Learning

- Need: Just raw text
- Example: "The cat sat on the [?]"
- Advantage: Free labels from text itself
- Scale: Billions of examples

Creating Labels from Text

Original: "The quick brown fox jumps"

↓ Hide words

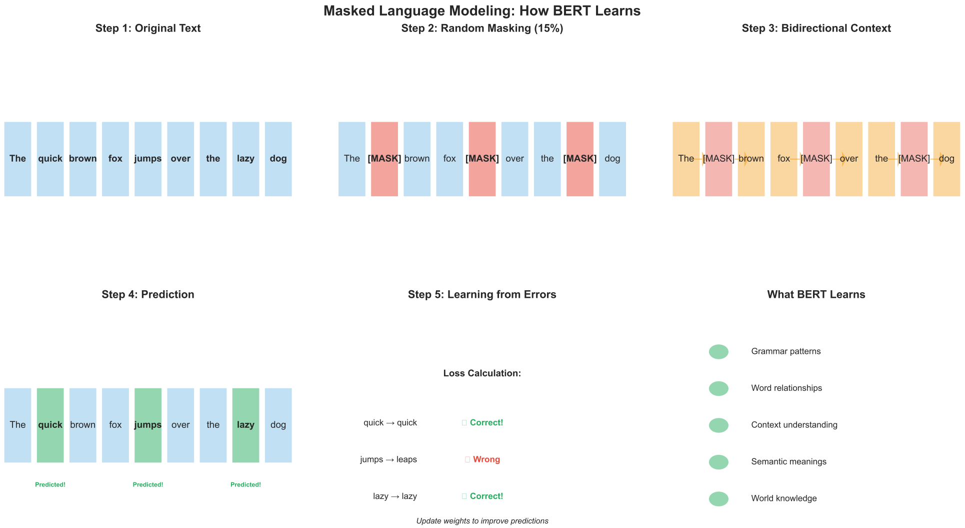
Input: "The [MASK] brown fox jumps"

Target: "quick"

Intuition

The text teaches itself! Every sentence becomes a training example.

BERT's Approach: Masked Language Modeling



Try It Yourself: Fill in the Blanks

BERT's Training Game - You Try:

1. The capital of France is [MASK].
   - Your guess: ____________________
   - Answer: Paris
2. The [MASK] rises in the east.
   - Your guess: ____________________
   - Answer: sun
3. She [MASK] to the store yesterday.
   - Your guess: ____________________
   - Answer: went

```python
# BERT learns by playing this game billions of times
def train_bert(text):
    masked_text = randomly_mask(text, rate=0.15)
    prediction = bert_model(masked_text)
    loss = compare(prediction, original_text)
    update_weights(loss)
```

GPT's Approach: Next Token Prediction

Autoregressive Learning

- Predict next word given previous
- Left-to-right processing
- Natural for generation
- Simpler than BERT's MLM

Training Example:

- Input: "The cat sat"
- Target: "on"
- Next: "The cat sat on"
- Target: "the"
- Continue for all text...

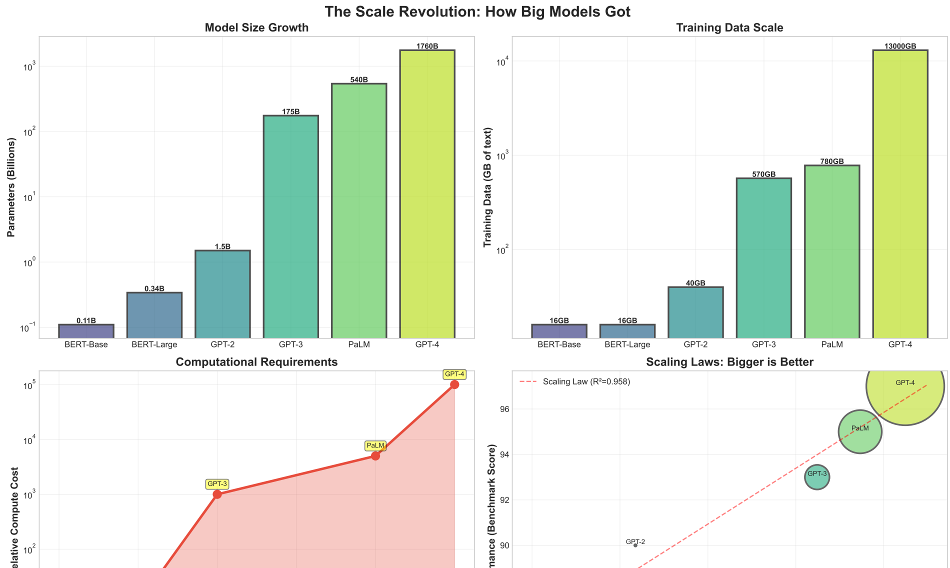
Interactive Example:
Complete these sentences:

1. "Once upon a..." → ____________
2. "To be or not to..." → ____________
3. "The weather today is..." → ____________

Key Insight

GPT learns to write by predicting what comes next, just like you just did!

Scale Matters: The Power of Big



What Do Models Actually Learn?

Layer 1-4: Syntax & Grammar

- Parts of speech
- Word order rules
- Basic grammar patterns

Layer 5-8: Semantics

- Word meanings
- Concept relationships
- Context understanding

Layer 9-12: High-level Reasoning

- World knowledge
- Logical relationships
- Abstract concepts

Emergent Abilities:

- Translation (never explicitly taught!)
- Summarization
- Question answering
- Code generation
- Mathematical reasoning
- Creative writing

Intuition

Like a child learning language: first sounds, then words, then meaning, then reasoning

Part 2 Summary: Understanding Pre-training

Key Concepts:

1. **Self-Supervised**: Text provides its own labels - no manual annotation!
2. **BERT's MLM**: Fill in the blanks using bidirectional context
3. **GPT's Autoregressive**: Predict next word from previous words
4. **Scale Advantage**: Billions of parameters + terabytes of text = intelligence
5. **Emergent Learning**: Models learn tasks they were never explicitly taught

Checkpoint

Can you explain the difference between BERT's and GPT's training approach?

Next: Deep dive into BERT vs GPT architectures

Part 3

Architecture Deep Dive

BERT vs GPT - Two Paradigms

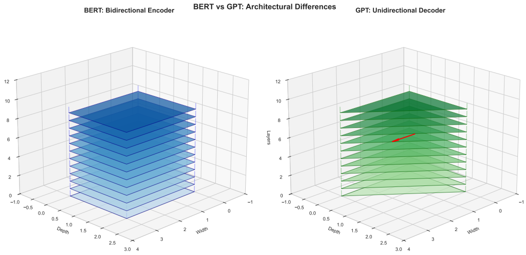
BERT: Bidirectional Encoder Representations

Architecture Components:

- **Input**: Token + Position + Segment embeddings
- **Core**: 12/24 Transformer encoder layers
- **Attention**: Bidirectional self-attention
- **Output**: Contextualized representations

Key Innovation:

- Sees both left AND right context
- Example: "The [MASK] barked loudly"
- Uses: "The" + "barked loudly"
- Better understanding than left-only



BERT Sizes:

- BERT-Base: 110M parameters
- BERT-Large: 340M parameters

BERT Training: Two Objectives

1. Masked Language Model (MLM)

- 15% tokens masked
- 80% replaced with [MASK]
- 10% replaced with random word
- 10% kept unchanged

Why the 80-10-10 split?

- Prevents overfitting to [MASK]
- Forces robust representations
- Handles noise in real data

2. Next Sentence Prediction (NSP)

- Input: Sentence A + Sentence B
- Task: Are they consecutive?
- 50% true next sentence
- 50% random sentence

Example:

- A: “The weather is nice.”
- B: “Let’s go for a walk.” ✓
- B: “Cats like fish.” ×

Key Insight

MLM teaches word understanding, NSP teaches sentence relationships

GPT: Generative Pre-trained Transformer

Architecture Components:

- **Input**: Token + Position embeddings
- **Core**: 12/24/48 Transformer decoder layers
- **Attention**: Causal (left-to-right) mask
- **Output**: Next token predictions

Key Design:

- Only sees previous tokens
- Natural for generation
- Autoregressive decoding
- Simpler training objective

GPT Evolution:

- GPT-1 (2018): 117M parameters
- GPT-2 (2019): 1.5B parameters
- GPT-3 (2020): 175B parameters
- GPT-4 (2023): 1.76T parameters*

Training Process:

1. Input: "The cat sat"
2. Predict: "on"
3. Input: "The cat sat on"
4. Predict: "the"
5. Continue...

BERT vs GPT: When to Use Which?

Aspect	BERT	GPT
Context	Bidirectional	Left-to-right
Best for	Understanding	Generation
Training	MLM + NSP	Next token
Speed	Faster inference	Slower (autoregressive)
Use Cases		
Classification	**Excellent**	Good
NER	**Excellent**	Good
QA	**Excellent**	Good
Generation	Poor	**Excellent**
Translation	Good	**Excellent**
Summarization	Good	**Excellent**

Intuition

BERT reads the whole page to understand, GPT writes one word at a time

Tokenization: Breaking Text into Pieces

BERT: WordPiece Tokenization

- Vocabulary: 30,000 tokens
- Subword units
- Handles unknown words

```
"unbelievable" ->
["un", "##believ", "##able"]

"COVID-19" ->
["COVID", "-", "19"]
```

GPT: Byte-Pair Encoding (BPE)

- Vocabulary: 50,000+ tokens
- Learned from frequency
- Efficient for generation

```
"unbelievable" ->
["un", "believ", "able"]

"COVID-19" ->
["COVID", "-19"]
```

Why Subword Tokenization?

- Handles any word (even made-up ones)
- Reduces vocabulary size
- Captures morphology (prefixes, suffixes)

Input Representations: More Than Just Words

BERT's Three-Part Input:

Token	[CLS]	The	cat	sat	[SEP]
Token Emb	E_{CLS}	E_{the}	E_{cat}	E_{sat}	E_{SEP}
Position Emb	E_0	E_1	E_2	E_3	E_4
Segment Emb	E_A	E_A	E_A	E_A	E_A

Final Input	Sum of all three embeddings

Special Tokens:

- CLS : Classification token (sentence representation)
- SEP : Separator between sentences
- MASK : Masked token for MLM
- PAD : Padding for batch processing

Key Insight

Position embeddings tell the model word order, segment embeddings separate sentences

Part 3 Summary: Architecture Insights

BERT (Bidirectional)

- Sees full context (left + right)
- Best for understanding tasks
- Two training objectives (MLM + NSP)
- Cannot generate text naturally

GPT (Autoregressive)

- Sees only previous context
- Best for generation tasks
- Single training objective (next token)
- Scales to trillions of parameters

Checkpoint

Why can't BERT generate text as naturally as GPT?

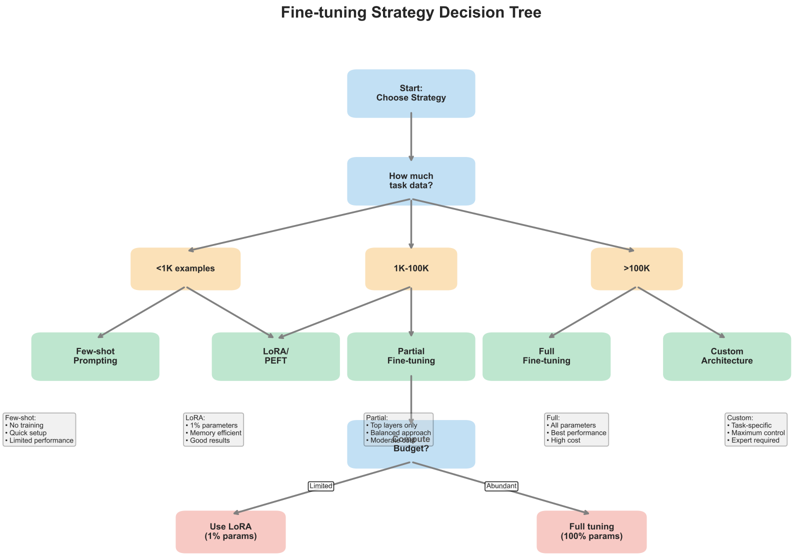
Next: How to adapt these models to your specific tasks

Part 4

Fine-tuning and Applications

From Foundation to Task-Specific Excellence

Fine-tuning Strategy Decision Tree



Task-Specific Heads: Adapting to Your Task

Classification Head

```
# Sentiment analysis
bert_output = bert(text)
cls_token = bert_output[0]  # [CLS]
logits = linear(cls_token)
class = softmax(logits)
# Output: positive/negative
```

Token Classification (NER)

```
# Named entity recognition
bert_output = bert(tokens)
token_logits = linear(bert_output)
entities = softmax(token_logits)
# Output: PER, ORG, LOC per token
```

Question Answering

```
# Extract answer span
bert_output = bert(question, context)
start_logits = linear_start(bert_output)
end_logits = linear_end(bert_output)
answer = context[start:end]
```

Sequence-to-Sequence

```
# Summarization, translation
encoder_out = bert_encoder(source)
summary = gpt_decoder(encoder_out)
# Output: Generated text
```

Key Insight

The pre-trained model stays mostly the same, only the task head changes!

The Fine-tuning Process: Step by Step

Step 1: Choose Pre-trained Model

- BERT for understanding
- GPT for generation
- T5 for any text-to-text

Step 2: Prepare Task Data

- Format: (input, label) pairs
- Quality ¿ Quantity
- 1K-10K examples usually enough

Step 3: Add Task Head

- Classification: Linear + Softmax
- NER: Token classifier
- QA: Span predictor

Step 4: Fine-tune

- Lower learning rate (2e-5)
- Few epochs (2-4)
- Watch for overfitting

Step 5: Evaluate

- Hold-out test set
- Task-specific metrics
- Compare to baseline

Intuition

Like teaching a well-educated person a new skill - they learn fast!

Few-shot and Zero-shot: Learning Without Training

Zero-shot (No Examples)

- Just describe the task
- Works with large models (GPT-3+)
- Example prompt:

"Classify sentiment: 'This movie is amazing!'
Answer: Positive"

Few-shot (1-10 Examples)

- Provide examples in prompt
- No actual training
- In-context learning

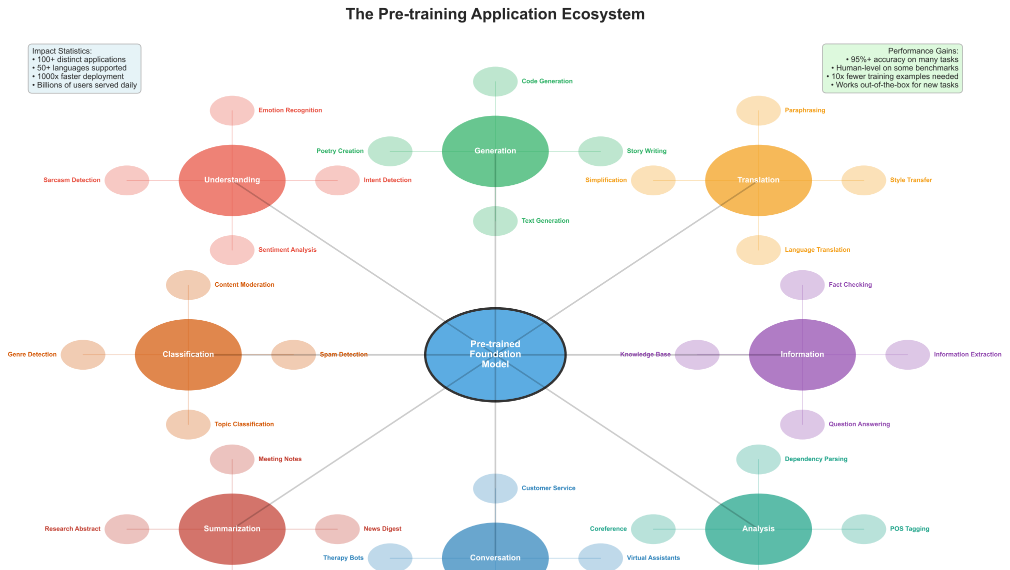
Few-shot Example:
"Translate English to French:
sea otter → loutre de mer
cheese → fromage
airplane → avion
teacher → ?"

Answer: professeur

Key Insight

Large models can learn new tasks just from instructions!

Real-world Success Stories



Environmental Considerations

The Carbon Cost

- GPT-3 training: 1,287 MWh
- = 552 tons CO2
- = 120 cars for a year

But Consider:

- Train once, use millions of times
- Replaces thousands of task-specific models
- Net positive if widely used

Efficiency Improvements

- LoRA: 1% of parameters
- Quantization: 4-bit models
- Distillation: Smaller models
- Better hardware: TPUs, H100s

Checkpoint

How does fine-tuning reduce environmental impact compared to training from scratch?

Modern Efficiency: LoRA and PEFT

LoRA (Low-Rank Adaptation)

- Only train 1% of parameters
- Add small matrices to layers
- Same performance as full fine-tuning
- 100x less memory

```
# Instead of updating W (d x d)
# Add low-rank matrices A (d x r) and B (r x d)
# where r << d
W_new = W + A @ B
# Only train A and B!
```

Benefits:

- Fine-tune GPT-3 on single GPU
- Switch tasks by swapping LoRA weights
- Merge multiple adaptations
- Deploy efficiently

Other PEFT Methods:

- Prefix Tuning
- Prompt Tuning
- Adapter Layers
- BitFit

Part 4 Summary: Practical Applications

Key Takeaways:

1. **Fine-tuning Strategy**: Choose based on data size and compute budget
2. **Task Heads**: Simple additions for specific tasks
3. **Few-shot Magic**: Large models learn from just examples
4. **Applications**: 100+ tasks enabled by pre-training
5. **Environmental**: Consider efficiency methods like LoRA
6. **Modern Methods**: 1% parameters, 100% performance

You now understand the complete pre-training pipeline!

Appendix

Resources and Advanced Topics

Your Toolkit for Getting Started

Model Zoo: Available Pre-trained Models

Hugging Face Hub

- 500,000+ models
- All major architectures
- Easy to use API
- Community contributed

Popular Models:

- **BERT**: bert-base-uncased
- **RoBERTa**: roberta-large
- **GPT-2**: gpt2-medium
- **T5**: t5-base
- **BART**: facebook/bart-large

Specialized Models:

- **Code**: Codex, CodeBERT
- **Science**: SciBERT, BioBERT
- **Legal**: LegalBERT
- **Finance**: FinBERT
- **Multilingual**: mBERT, XLM-R

Access Methods:

- Hugging Face Transformers
- OpenAI API
- Google Vertex AI
- AWS SageMaker

Computational Requirements

Model	Parameters	Memory	Fine-tune GPU
BERT-Base	110M	440MB	GTX 1080 (8GB)
BERT-Large	340M	1.3GB	RTX 3090 (24GB)
GPT-2	1.5B	6GB	A100 (40GB)
GPT-3	175B	700GB	8x A100 (320GB)
With LoRA (1% parameters):			
GPT-3 + LoRA	1.75B	7GB	RTX 4090 (24GB)

Cost Estimates (2024):

- Cloud GPU (A100): \$3-5/hour
- Fine-tuning BERT: \$10-50
- Fine-tuning GPT-3 with LoRA: \$100-500
- API calls: \$0.002 per 1K tokens

Latest Developments (2024)

New Models:

- **GPT-4**: Multimodal, 1.76T params
- **Claude 3**: Constitutional AI
- **Gemini**: Google's unified model
- **Llama 3**: Open-source 405B
- **Mistral**: Efficient 7B model

Trends:

- Mixture of Experts (MoE)
- Multimodal (text + image + audio)
- Longer context (1M+ tokens)
- Tool use and function calling

Research Directions:

- Constitutional AI
- Chain-of-thought reasoning
- Retrieval-augmented generation
- Sparse models
- Continuous learning

Open Problems:

- Hallucination
- Bias and fairness
- Interpretability
- Efficiency at scale

Quick Start: Your First Fine-tuning

5-Minute Setup with Hugging Face:

```
from transformers import AutoModelForSequenceClassification, AutoTokenizer
from transformers import TrainingArguments, Trainer
import torch

# 1. Load pre-trained model
model = AutoModelForSequenceClassification.from_pretrained(
    "bert-base-uncased", num_labels=2
)
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

# 2. Prepare your data
texts = ["I love this!", "This is terrible."]
labels = [1, 0]  # positive, negative

# 3. Tokenize
inputs = tokenizer(texts, padding=True, truncation=True, return_tensors="pt")
inputs["labels"] = torch.tensor(labels)

# 4. Fine-tune
training_args = TrainingArguments(
    output_dir="./results",
    num_train_epochs=3,
    per_device_train_batch_size=16,
)
trainer = Trainer(model=model, args=training_args, train_dataset=inputs)
trainer.train()
```

Best Practices and Tips

Data Preparation:

- Quality ¿ Quantity
- Balance your classes
- Clean and consistent formatting
- Augment if needed

Training Tips:

- Start with small learning rate (2e-5)
- Use warmup steps
- Monitor validation loss
- Early stopping to prevent overfit

Common Pitfalls:

- Overfitting on small data
- Wrong tokenizer for model
- Catastrophic forgetting
- Ignoring class imbalance

Debugging:

- Start with tiny dataset
- Verify input shapes
- Check gradient flow
- Visualize attention weights

Learning Resources

Papers to Read:

- BERT: Devlin et al. (2018)
- GPT-3: Brown et al. (2020)
- T5: Raffel et al. (2019)
- LoRA: Hu et al. (2021)

Courses:

- Hugging Face Course (free)
- CS224N Stanford NLP
- Fast.ai Practical Deep Learning
- Coursera NLP Specialization

Tools and Libraries:

- Hugging Face Transformers
- LangChain
- OpenAI API
- Weights & Biases

Communities:

- Hugging Face Forums
- r/MachineLearning
- Twitter ML Community
- Local AI Meetups

Quick Reference Card

Model Selection:

- Understanding → BERT
- Generation → GPT
- Both → T5
- Efficiency → DistilBERT

Data Requirements:

- Few-shot: 1-10 examples
- LoRA: 100-1K examples
- Full fine-tune: 1K-10K examples
- From scratch: 100K+ examples

Key Commands:

```
# Install
pip install transformers

# Load model
from transformers import AutoModel
model = AutoModel.from_pretrained("bert-base")

# Fine-tune
trainer.train()

# Inference
outputs = model(**inputs)
```

Congratulations! You're Ready to Build

What You've Learned:

1. Why pre-training revolutionized NLP (100x efficiency)
2. How models learn from raw text (self-supervised)
3. BERT vs GPT architectures (bidirectional vs autoregressive)
4. Fine-tuning strategies (full, LoRA, few-shot)
5. Real-world applications (100+ tasks)

You now have the knowledge to use pre-trained models!

Start with the Jupyter notebook exercises

Questions?