Chapter 9: Pre-training Paradigms — BERT, GPT, and T5

Learning Objectives

After reading this chapter, the reader should be able to:

1. Explain why pre-training on large unlabeled corpora followed by task-specific fine-tuning outperforms training from scratch, and quantify the data-efficiency gains with concrete examples.
2. Derive and implement the masked language modeling (MLM) objective used in BERT, including the 80/10/10 masking strategy, and contrast it with the causal language modeling (CLM) objective used in GPT.
3. Compare encoder-only (BERT), decoder-only (GPT), and encoder-decoder (T5) architectures in terms of their pre-training objectives, computational trade-offs, and downstream task suitability.
4. Fine-tune a pre-trained transformer on a text classification task using the Hugging Face Transformers library, evaluate transfer learning performance against a randomly initialized baseline, and interpret fine-tuning loss curves.

In Chapter 8, we assembled the full Transformer architecture: scaled dot-product attention captures pairwise relationships, multi-head attention captures diverse relationship types, positional encodings inject word order, feed-forward layers add per-position nonlinearity, and residual connections with layer normalization enable deep stacking. The architecture comes in three variants: encoder-only, decoder-only, and encoder-decoder. But an architecture is only a blueprint — it requires a training procedure and data to become a useful model. The question that defined the 2018–2020 era of natural language processing was deceptively simple: how should we *train* a Transformer? Should we mask tokens and predict them bidirectionally? Predict tokens left to right? Corrupt spans and reconstruct them? Each answer produces a fundamentally different model with different strengths. In this chapter, we present the three paradigms that answered this question — BERT's masked language modeling, GPT's causal language modeling, and T5's span corruption — and explain how they transformed the Transformer from an architectural blueprint into the foundation of modern artificial intelligence. The unifying thread, as established in Chapter 1, is prediction: all three paradigms are about predicting tokens. They differ only in which tokens are predicted and what context is available.

9.1 The Pre-training Revolution

9.1.1 From Task-Specific to General-Purpose Representations

What if a model could learn the structure of language itself before it ever encounters a single labeled example?

Before 2018, the dominant paradigm in natural language processing required assembling task-specific labeled data for every problem one wished to solve. A practitioner building a sentiment classifier would collect thousands of annotated reviews; a team developing a named entity recognizer (NER) would annotate news articles with entity spans; a question answering (QA) system would be trained

on manually constructed question-answer pairs. Each of these models was built from scratch, learning the grammar, vocabulary, semantics, and pragmatics of language simultaneously with the task-specific mapping from input to output. This approach is deeply inefficient. Linguistic knowledge — the understanding that "bank" can refer to a financial institution or a riverbank, that adjectives modify nouns, that modal verbs express possibility rather than fact — is shared across every downstream task. A sentiment classifier and a named entity recognizer are both processing the same language; they should not need to re-learn its structure independently. The pre-2018 paradigm forced them to do exactly that, repeating the same expensive linguistic learning for every new application.

The key insight that animates the pre-training revolution is that language structure learned from massive unlabeled text transfers to downstream tasks. A model trained to predict missing words in Wikipedia articles has necessarily learned that "The prime minister of [MASK] is" is likely followed by a country or a person's name, not a verb or an adjective. It has learned syntactic agreement, semantic coherence, and world knowledge from billions of examples, none of which required expensive human annotation. This model is not merely a better initialization for downstream tasks; it has already encoded syntactic structure, semantic relationships, and factual associations that would otherwise take millions of labeled examples to develop. Fine-tuning on a small labeled dataset then adapts this rich general representation to the specific mapping required by the target task. Students often think pre-training is just a form of warm-starting parameter optimization. The distinction matters: a warm start begins optimization from a non-random point in the same loss landscape; pre-training creates a fundamentally different representation space in which downstream learning operates, one rich with linguistic structure that the downstream task alone could never instill. Consider the analogy of a medical student who spends years studying general anatomy, physiology, and pharmacology from vast observational experience before choosing a specialty. The general training does not merely accelerate specialization — it provides the conceptual vocabulary, the pattern recognition, and the integrative knowledge that make specialization possible at all. A student who attempted to learn cardiology without the general foundation would require far more examples, would miss cross-disciplinary connections, and would likely develop a narrower and more brittle competence.

9.1.2 Transfer Learning: The ImageNet Moment for NLP

In 2012, the computer vision community experienced a watershed event. A deep convolutional neural network trained by Krizhevsky, Sutskever, and Hinton [Krizhevsky et al., 2012] on the ImageNet dataset — 1.2 million labeled images across 1,000 categories — achieved a top-5 error rate of 15.3% on the ImageNet Large Scale Visual Recognition Challenge, dramatically outperforming the previous state of the art at 26.2%. More consequential than the benchmark result, however, was what followed: researchers discovered that the features learned by this network transferred remarkably well to other vision tasks. Edge detectors, texture analyzers, and shape descriptors learned on ImageNet proved equally useful for medical image classification, satellite imagery analysis, and object detection. The standard workflow in computer vision became: pre-train on ImageNet, fine-tune the features for the specific task. The discovery that visual representations generalize across domains reduced the data requirements for new vision applications by orders of magnitude and democratized access to state-of-the-art performance.

NLP's analogous moment came in 2018, but it was preceded by important precursors. The ELMo model [Peters et al., 2018], which used bidirectional LSTMs pre-trained on a language modeling objective, demonstrated that contextual word representations — representations that change based

on the surrounding context, so that `bank'' in a financial context receives a different vector than`bank" in a geographical context — transfer strongly across tasks. ELMo features improved performance on six diverse NLP benchmarks simply by being substituted for static word embeddings, without any fine-tuning of the ELMo parameters themselves. ULMFiT [Howard and Ruder, 2018] introduced the first complete pre-train/fine-tune framework for NLP, demonstrating that a language model pre-trained on Wikipedia could be fine-tuned for text classification with as few as 100 labeled examples and match models trained on tens of thousands. We remember the moment when BERT's results first appeared on the GLUE leaderboard — the margins were so large that many researchers initially suspected a bug in the evaluation. BERT achieved improvements of 7.7 percentage points over the previous best system on the GLUE benchmark, and similar gains on SQuAD and CoNLL NER, all using the same pre-trained model with minimal task-specific adaptation. The magnitude of the improvement was qualitatively different from incremental progress; it was the NLP equivalent of the 2012 ImageNet shock.

Sidebar: Historical Note: ELMo and ULMFiT — The Precursors

The pre-training revolution did not begin with BERT. Two pivotal contributions set the stage in early 2018. *ELMo* (Embeddings from Language Models), introduced by Peters et al. [2018], produced contextualized word representations by running a deep bidirectional LSTM over a text sequence and using the concatenated hidden states as word features. Unlike static embeddings (Chapter 4), ELMo representations change based on context, enabling disambiguation of polysemous words. ELMo was evaluated as a drop-in replacement for word embeddings across six tasks, improving performance by 4.1–9.6% without any fine-tuning. *ULMFiT* (Universal Language Model Fine-tuning), introduced by Howard and Ruder [2018], went further: it proposed a three-stage training protocol (pre-training a language model on Wikipedia, domain-adaptive pre-training on the target domain, and fine-tuning the classifier), plus critical training techniques including discriminative fine-tuning (different learning rates per layer) and slanted triangular learning rates. ULMFiT established the template that BERT would follow at vastly greater scale. Both ELMo and ULMFiT demonstrated the core principle: language model pre-training transfers. BERT's contribution was to scale this principle to deep bidirectional Transformer encoders, to train on orders of magnitude more data, and to demonstrate simultaneous state-of-the-art performance across 11 diverse benchmarks.

9.1.3 The Pre-train / Fine-tune Paradigm

The story of pre-training did not emerge from a single paper but from a decade of accumulated evidence that representation learning matters. By 2017, word embeddings (Chapter 4) were ubiquitous, but they were context-independent: the same vector represented "bank" regardless of whether it appeared in a sentence about finance or geology. The natural extension was clear: if representations could encode context, they would transfer even more effectively. What BERT, GPT, and T5 did was realize this extension at scale, using Transformer architectures capable of capturing arbitrarily long-range contextual dependencies, trained on corpora large enough to develop world knowledge alongside linguistic structure.

The pre-train/fine-tune paradigm operates in two distinct phases. In the pre-training phase, a Transformer is trained on a massive unlabeled corpus using a self-supervised objective: the training signal is derived from the text itself, with no human annotation required. The model learns to solve a prediction task — predicting masked tokens, predicting the next token, reconstructing corrupted

spans — and in doing so develops rich internal representations of linguistic structure. The resulting model, often containing hundreds of millions or billions of parameters, encodes syntax, semantics, coreference relations, and world knowledge. In the fine-tuning phase, this pre-trained model is adapted to a specific downstream task by training on a small labeled dataset. All parameters are updated, but the learning rate is much smaller (typically 2×10^{-5} versus 10^{-4} for pre-training) and training runs for only three to five epochs. The key data-efficiency gain is striking: BERT fine-tuned on 5,000 labeled examples routinely outperforms models trained from scratch on 50,000 examples for the same task [Devlin et al., 2019]. The reason is that fine-tuning operates in a parameter space already shaped by rich linguistic knowledge; it needs only to learn the task-specific mapping, not the entire structure of language. Figure 9.1 illustrates the complete pipeline from raw text through pre-training to task-specific fine-tuning.

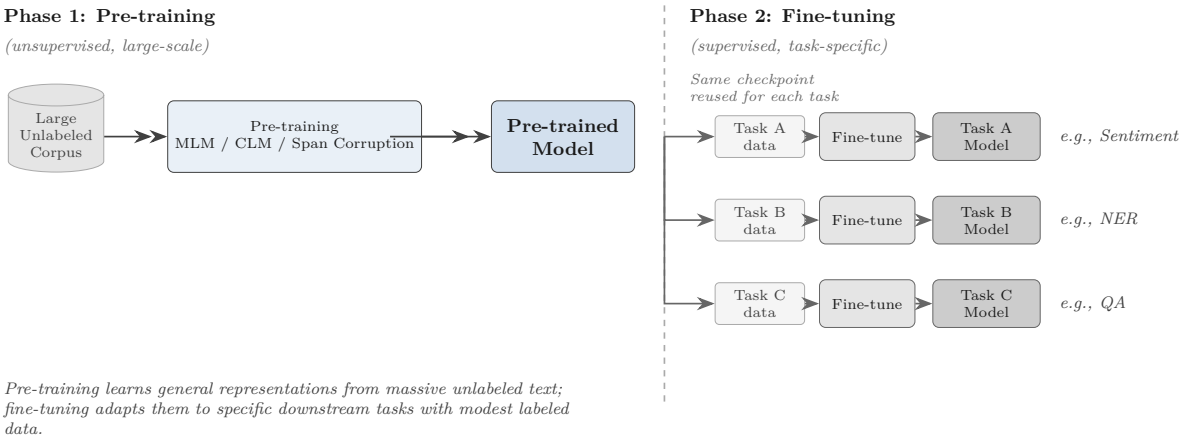

Figure 1: Figure 9.1 – Pre-training and Fine-tuning Pipeline

The paradigm is significant not only for its practical impact but for what it reveals about the nature of language learning. The success of pre-training confirms that the bulk of what a model needs to know about language can be learned from prediction alone — from asking and answering the question "what token belongs here?" billions of times. This connects directly to the central thesis of this book (Chapter 1): language modeling, the task of predicting the next word, is not merely a benchmark but a rich learning signal that drives the development of deep linguistic competence.

It is also worth examining what pre-training does not do, because the boundaries of transfer matter as much as its extent. Pre-training on general text does not substitute for domain-specific fine-tuning when the target domain is highly specialized. A BERT model pre-trained on Wikipedia and books has limited knowledge of clinical trial protocols, software commit messages, or financial earnings calls; fine-tuning on domain-specific unlabeled text (domain-adaptive pre-training) before task-specific fine-tuning consistently improves performance in such settings [Gururangan et al., 2020]. Similarly, pre-training does not eliminate the need for labeled data entirely: even the most capable pre-trained models benefit from at least a few labeled examples when fine-tuned for specific tasks, and for tasks with very narrow output formats — such as structured information extraction or formal document generation — substantial labeled data remains necessary. The pre-train/fine-tune paradigm dramatically reduces data requirements; it does not eliminate them. Understanding these boundaries helps set appropriate expectations for practitioners: pre-training is a powerful prior over language structure, not a substitute for all forms of supervision. Section 9.2 turns to the first of the

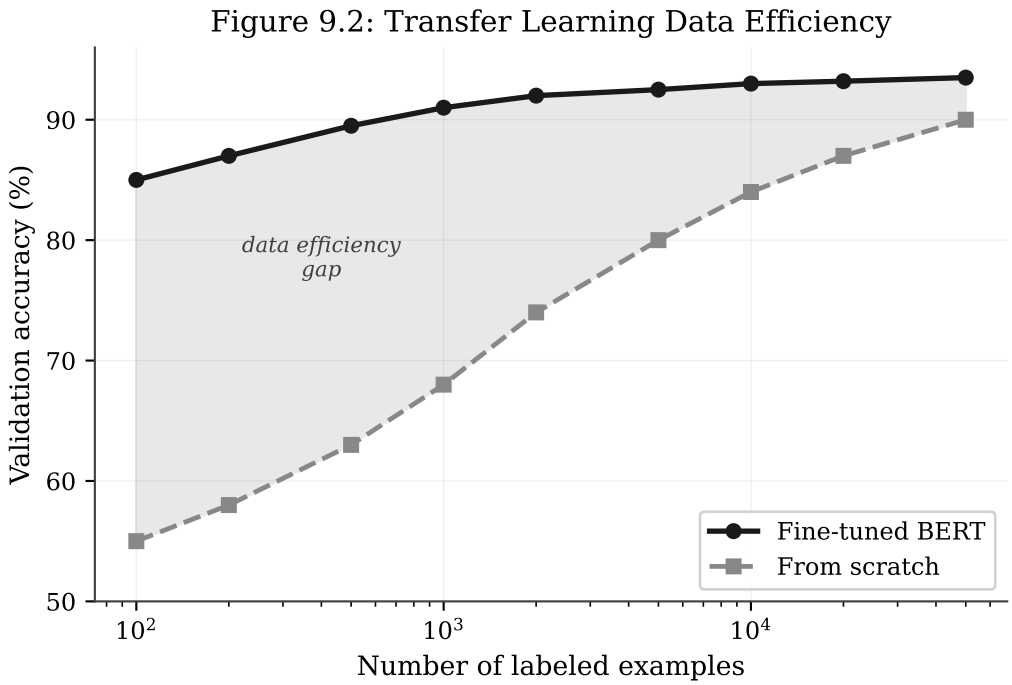

Figure 2: Figure 9.2 – Transfer Learning Data Efficiency

three paradigms, examining how BERT formalized this insight into the masked language modeling objective.

9.2 BERT and Masked Language Modeling

9.2.1 BERT Architecture and Input Representation

A surprising but consequential fact underlies the design of BERT: the simplest possible modification to the standard Transformer encoder — training it to predict masked tokens rather than the next token — turned out to produce representations that outperformed every prior approach across a wide range of understanding tasks simultaneously. BERT (Bidirectional Encoder Representations from Transformers) [Devlin et al., 2019] is a bidirectional Transformer encoder (Chapter 8, Section 8.6). Unlike the original Transformer architecture of Vaswani et al. [2017], which combined an encoder and a decoder for sequence-to-sequence generation, BERT uses only the encoder stack and trains it on a self-supervised prediction task that requires bidirectional context. The architectural choice is fundamental: by attending to both left and right context simultaneously, the BERT encoder can build richer representations of token meaning than any unidirectional model.

BERT was released in two sizes. BERT-base contains $L = 12$ Transformer layers, model dimension $d_{\text{model}} = 768$, $h = 12$ attention heads, and a feed-forward dimension of 3,072, yielding approximately 110 million parameters. BERT-large contains $L = 24$ layers, $d_{\text{model}} = 1024$, $h = 16$ heads, and a feed-forward dimension of 4,096, yielding approximately 340 million parameters. Both models were pre-trained on the BooksCorpus (800 million words) and English Wikipedia (2.5 billion words), using the same WordPiece tokenizer with a vocabulary of 30,522 tokens. These scale decisions

reflect a careful balance between representational capacity and computational feasibility with the hardware available in 2018; subsequent models would push both dimensions considerably further.

BERT's input representation combines three distinct embedding sources into a single vector for each token. The *token embedding* maps each token identifier to a learned dense vector. The *segment embedding* distinguishes tokens from the first input sequence (sentence A) from tokens in the second input sequence (sentence B), using one of two learned segment vectors. The *position embedding* is a learned (not sinusoidal) absolute position encoding supporting sequences up to 512 tokens. The final input representation for token w_i is the elementwise sum of these three embeddings:

$$\mathbf{e}_i = \mathbf{e}_{\text{token}}(w_i) + \mathbf{e}_{\text{seg}}(s_i) + \mathbf{e}_{\text{pos}}(i),$$

where $s_i \in \{A, B\}$ indicates which sentence the token belongs to. BERT augments every input with three special tokens: `[CLS]` is prepended to the entire sequence and serves as the aggregate sequence representation for classification tasks; `[SEP]` separates sentence A from sentence B and terminates each sentence; `[MASK]` replaces tokens selected for the masked language modeling objective. The `[CLS]` token deserves emphasis because it is commonly misunderstood. It has no special pre-programmed semantic content; its aggregate utility for classification is emergent from the pre-training process. Because `[CLS]` attends to every other token through self-attention across all 12 layers, its final-layer representation accumulates information from the entire input sequence. This emergent function — a summary token that listens to all others — makes it useful as a classification feature, but it is trained into existence rather than designed explicitly. One useful analogy is that of a moderator in a panel discussion: positioned at the start, attending to every participant through the bidirectional self-attention mechanism, and producing a summary of the conversation as its representation.

The choice of learned position embeddings over the sinusoidal encodings used in the original Transformer [Vaswani et al., 2017] reflects a practical design decision: empirically, learned embeddings performed similarly to sinusoidal ones for sequences within the training length limit (512 tokens), but they are simpler to implement and more easily tuned during pre-training. The maximum sequence length of 512 tokens represents a constraint with practical implications: documents longer than 512 tokens must be truncated or processed in sliding windows, and the model has no capacity to attend across window boundaries. This limitation motivated subsequent work on long-range Transformers (Longformer [Beltagy et al., 2020], BigBird [Zaheer et al., 2020]) that extend the attention mechanism to sequences of thousands of tokens through sparse attention patterns. The BERT architecture is therefore best understood as a design tuned for sentence-level and short-passage tasks rather than document-level processing, which partially explains why encoder-decoder models like T5 — which can process longer inputs more naturally through the encoder — are preferred for tasks involving long documents such as scientific paper summarization or legal document analysis.

9.2.2 The Masked Language Modeling Objective

How can a model learn from text when the training signal must be derived from the text itself, without human annotation and without the model simply copying its input?

The core technical challenge of self-supervised pre-training is designing an objective that provides a meaningful learning signal from raw text. The most natural approach — predict the next token given all previous tokens, as in a standard language model — does not allow bidirectional context, because allowing the model to see the token it is trying to predict would trivialize the task. BERT's solution is Masked Language Modeling (MLM): randomly replace a subset of input tokens with a

special [MASK] token, then train the model to predict the original tokens at those positions using bidirectional context from all unmasked tokens.

Formally, let $\mathbf{w} = (w_1, \ldots, w_T)$ be a token sequence, and let $\mathcal{M} \subset \{1, \ldots, T\}$ be the set of masked positions, selected uniformly at random to cover approximately 15% of all tokens. Let $\mathbf{w}_{\setminus \mathcal{M}}$ denote the sequence with the selected tokens replaced according to the 80/10/10 strategy described below. The MLM objective is:

$$\mathcal{L}_{\text{MLM}} = -\sum_{m \in \mathcal{M}} \log P(w_m \mid \mathbf{w}_{\setminus \mathcal{M}}; \theta) \tag{9.1}$$

The probability $P(w_m \mid \mathbf{w}_{\setminus \mathcal{M}}; \theta)$ is computed by passing $\mathbf{w}_{\setminus \mathcal{M}}$ through the full BERT encoder to produce a contextual representation $\mathbf{h}_m$ for position m, then projecting through a learned linear layer followed by softmax over the vocabulary:

$$P(w_m \mid \mathbf{w}_{\setminus \mathcal{M}}; \theta) = \text{softmax}(\mathbf{W}_{\text{proj}} \mathbf{h}_m + \mathbf{b})_{w_m}.$$

Crucially, only the masked positions $m \in \mathcal{M}$ contribute to the loss. The unmasked tokens provide context but receive no gradient from the MLM objective. This is a recognized inefficiency: per training example, only 15% of tokens generate a learning signal. We discuss the implications for training efficiency when comparing MLM with the causal language modeling objective in Section 9.3.

The 15% masking rate is a hyperparameter balancing two competing pressures. Increasing the masking rate provides more gradient signal per forward pass but destroys more context, making predictions harder and less informative. Decreasing it preserves context but reduces signal density. The 15% figure was chosen empirically in the original BERT work and has remained relatively standard across subsequent encoder models.

The 80/10/10 masking strategy addresses a fundamental mismatch between pre-training and fine-tuning. If the model always encountered [MASK] tokens during pre-training, it would learn to produce representations conditioned on the presence of [MASK], but during fine-tuning no [MASK] tokens appear. The model's representations would be trained on a different token distribution than they would encounter at inference time. To mitigate this, Devlin et al. [2019] propose the following procedure for each selected position $m \in \mathcal{M}$: with probability 80%, replace w_m with [MASK]; with probability 10%, replace w_m with a randomly chosen token from the vocabulary; and with probability 10%, leave w_m unchanged. The three components serve distinct purposes. The 80% [MASK] replacement provides the primary training signal. The 10% random replacement introduces noise that teaches the model not to blindly trust its input — even visible tokens might be wrong, requiring the model to maintain uncertainty about all positions and produce contextually grounded representations everywhere. The 10% unchanged component forces the model to develop representations for real tokens at potentially masked positions, directly closing the training-inference mismatch. We omit the full derivation of the MLM gradient with respect to the projection weights, which is straightforward but notationally heavy — the interested reader can find it in Devlin et al.'s supplementary materials.

MLM masking and loss computation with Hugging Face. The `DataCollatorForLanguageModeling` handles the 80/10/10 strategy automatically; here we show explicit masked prediction for transparency.

```python
from transformers import BertTokenizer, BertForMaskedLM, pipeline

# BERT predicts masked tokens using full bidirectional context
unmasker = pipeline("fill-mask", model="bert-base-uncased")

# The model sees BOTH left context ("The capital of France is")
# AND right context (".") to predict "paris"
results = unmasker("The capital of France is [MASK].")
for r in results[:3]:
    print(f"  {r['token_str']:>12s}  (score: {r['score']:.3f})")
# Output: paris (0.988), london (0.002), rome (0.001)
# Compare: GPT would see only "The capital of France is" -- no right context
```

9.2.3 Next Sentence Prediction (and Its Limitations)

In our experience teaching this material, the question of why next sentence prediction was dropped generates more classroom debate than almost any other design decision in the BERT literature. BERT's original pre-training combines the MLM objective with a second task: Next Sentence Prediction (NSP). The motivation was to improve BERT's performance on tasks requiring sentence-pair reasoning, such as natural language inference (NLI) and question answering, which require understanding the relationship between two text segments rather than the individual meaning of either.

The NSP objective is formulated as follows. Given an input of the form `[CLS]` + sentence A + `[SEP]` + sentence B + `[SEP]`, the model is trained to predict whether sentence B actually follows sentence A in the original document (labeled `IsNext`) or is a randomly selected sentence from elsewhere in the corpus (labeled `NotNext`). The training data is constructed so that 50% of examples are true consecutive sentence pairs (`IsNext`) and 50% are random pairs (`NotNext`). The prediction is made using the final `[CLS]` representation, passed through a binary classifier. The intuition is that recognizing sentence continuity requires discourse-level understanding: the model must track topic consistency, coreference, and causal or temporal sequence.

The problem with this reasoning, identified empirically by Liu et al. [2019] in the RoBERTa study, is that NSP does not actually require the discourse understanding it was designed to develop. Because the negative examples (`NotNext`) are drawn randomly from the corpus rather than from topically related passages, they differ from the positive examples not only in continuity but in topic. A model can therefore achieve high NSP accuracy through simple topic matching — detecting that sentence B discusses `sports'' while the context discusses`finance" — without learning anything about logical, causal, or referential relationships. NSP thus wastes training capacity on a task that is simultaneously too easy (solvable by topic overlap) and too artificial (random negative examples do not appear in real texts). One frequently hears the claim that NSP helps multi-sentence tasks even if it does not help single-sentence tasks; RoBERTa's ablation, however, found no consistent pattern of this kind. On NLI benchmarks like MNLI and RTE, which explicitly require sentence-pair reasoning, removing NSP improved or maintained performance rather than degrading it, suggesting that the MLM objective alone is sufficient to develop the inter-sentence representations needed for these tasks.

A more principled alternative to NSP, introduced in the ALBERT model [Lan et al., 2020], is the *Sentence Order Prediction* (SOP) objective: rather than distinguishing a true next sentence from

a random sentence, SOP distinguishes a correctly ordered sentence pair from the same sentence pair with the order swapped. The negative examples in SOP are topically coherent but structurally incorrect, forcing the model to learn about discourse coherence rather than topic consistency. ALBERT's ablations showed SOP outperforms NSP on tasks requiring sentence-pair understanding, validating the analysis that NSP's negative examples are too topically distinct to provide meaningful discourse training. RoBERTa's ablation study showed that removing NSP entirely and training on full-document sequences rather than sentence pairs consistently improved downstream performance across GLUE, SQuAD, and RACE, confirming that NSP was not contributing meaningful training signal. Virtually all subsequent encoder models — RoBERTa, ALBERT, ELECTRA, DeBERTa — omit NSP. It remains historically significant as an early attempt to incorporate discourse-level objectives into pre-training, but it is not a necessary ingredient for strong encoder performance.

Sidebar: The BERT Family — RoBERTa, ALBERT, ELECTRA, and DeBERTa

BERT's success motivated a productive sequence of improvements, each targeting a specific limitation of the original design. **RoBERTa** [Liu et al., 2019] is the most direct extension: it retains BERT's architecture while optimizing the training procedure. By removing NSP, adopting dynamic masking (generating new mask patterns for each epoch rather than fixing masks at data preprocessing time), training on ten times more data (160 GB versus 16 GB), and using larger batches and longer training, RoBERTa improved GLUE accuracy from 80.5% to 88.5% without any architectural change. The result is a powerful demonstration that BERT's original training recipe was significantly undertrained.

ALBERT [Lan et al., 2020] addresses BERT's parameter inefficiency. Two techniques reduce parameter count while maintaining representational capacity: factorized embedding parameterization, which separates the large token embedding matrix into two smaller matrices, reducing the embedding size from $|V| \times d_{\text{model}}$ to $|V| \times d_e + d_e \times d_{\text{model}}$; and cross-layer parameter sharing, in which all layers share the same self-attention and feed-forward parameters. ALBERT-xxlarge achieves performance comparable to BERT-Large with 18 times fewer parameters.

ELECTRA [Clark et al., 2020] replaces MLM with *replaced token detection*: a small generator network corrupts the input by replacing tokens with plausible alternatives, and the main discriminator model predicts which tokens were replaced by the generator versus which are original. Since every token is either original or replaced, every token provides a training signal — the same efficiency advantage as CLM, while retaining bidirectionality.

DeBERTa [He et al., 2021] introduces disentangled attention, maintaining separate content and position vectors per token and computing attention scores using both content-to-content and content-to-position interactions. DeBERTa-V3 surpassed human performance on the SuperGLUE benchmark (90.3% versus 89.8% human). These four variants illustrate that within the encoder-only paradigm, training methodology, parameter efficiency, training objective, and attention mechanism design all offer substantial performance headroom beyond the original BERT recipe.

9.2.4 Fine-tuning BERT for Downstream Tasks

```
import torch
from transformers import BertForSequenceClassification, BertTokenizer
from transformers import Trainer, TrainingArguments

# Load pre-trained BERT and tokenizer
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased", num_labels=2)
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

# Prepare example training data (sentiment: 0=negative, 1=positive)
train_texts = [
    "This movie was absolutely wonderful and moving.",
    "Terrible film, a complete waste of time.",
    "The acting was superb and the story compelling.",
    "Boring, predictable, and poorly written.",
]
train_labels = [1, 0, 1, 0]

# Tokenize: [CLS] + text + [SEP], padded to 128 tokens
train_encodings = tokenizer(train_texts, truncation=True,
                            padding="max_length", max_length=128,
                            return_tensors="pt")

# Create a simple torch Dataset for the Trainer
class SentimentDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels
    def __len__(self):
        return len(self.labels)
    def __getitem__(self, idx):
        item = {k: v[idx] for k, v in self.encodings.items()}
        item["labels"] = torch.tensor(self.labels[idx])
        return item

train_dataset = SentimentDataset(train_encodings, train_labels)

# Fine-tune: 3 epochs at lr=2e-5 -- the standard BERT recipe
args = TrainingArguments(output_dir="./bert-imdb",
    num_train_epochs=3, learning_rate=2e-5,
    per_device_train_batch_size=4, evaluation_strategy="no")
trainer = Trainer(model=model, args=args,
    train_dataset=train_dataset)
trainer.train()  # With 2,000+ examples, reaches ~90% accuracy

# Verify the model produces predictions
```

```
test_input = tokenizer("A truly excellent performance.",
                       return_tensors="pt", padding=True)
with torch.no_grad():
    logits = model(**test_input).logits
    pred = torch.argmax(logits, dim=-1).item()
print(f"Prediction: {'positive' if pred == 1 else 'negative'}")
print(f"Logits: {logits.tolist()}")
```

The code above captures the canonical fine-tuning recipe for BERT, and the recipe is deliberately simple. One adds a single linear layer mapping the d_{model}-dimensional `[CLS]` representation to the number of task classes, initializes it randomly, and trains the entire model end-to-end on labeled data. For BERT-base on IMDB sentiment classification with 2,000 training examples, this procedure reaches approximately 90% accuracy in three epochs — a result that a model trained from scratch would require ten to fifty times more data to match.

The fine-tuning procedure is formally characterized by the objective:

$$\mathcal{L}_{\text{fine-tune}} = -\sum_{i=1}^{N} \log P(y_i \mid \mathbf{x}_i; \theta_{\text{pre-trained}}) \tag{9.2}$$

where $\theta_{\text{pre-trained}}$ initializes the model parameters from the pre-trained checkpoint and N is the number of labeled training examples. The notation $\theta_{\text{pre-trained}}$ emphasizes that fine-tuning begins from a pre-trained initialization, but all parameters — including the original BERT weights — are updated during fine-tuning. The learning rate of 2×10^{-5} is small enough to preserve the pre-trained representations while adapting them to the task. Training for more than five epochs frequently leads to catastrophic forgetting, in which the fine-tuned model loses the general linguistic knowledge encoded during pre-training as it overfits to the small labeled dataset.

The generality of the fine-tuning recipe is one of BERT's most important contributions. For classification tasks (sentiment, NLI, textual entailment), the `[CLS]` representation is connected to a linear classifier. For token-level tasks such as NER, each token's final-layer representation is connected to a per-token classifier predicting the entity label. For extractive question answering, two linear layers are applied to the token representations to predict the start and end positions of the answer span in the passage. In every case, the same pre-trained model — with no task-specific architectural modification beyond the output layer — serves as the starting point, and three to five epochs of fine-tuning on hundreds to thousands of labeled examples produces state-of-the-art performance. This universality was unprecedented: BERT achieved simultaneous state-of-the-art results on eleven NLP benchmarks upon its publication [Devlin et al., 2019], including GLUE, SQuAD 1.1, SQuAD 2.0, and CoNLL-2003 NER, using a single pre-trained model as the foundation for all eleven.

Section 9.2 has presented BERT's architecture, its masked language modeling objective, the 80/10/10 masking strategy, the limitations of next sentence prediction, and the canonical fine-tuning recipe. The picture that emerges is of a model designed primarily for language understanding: bidirectional context makes BERT excellent at classification, NER, and extractive QA, but the very bidirectionality that enables rich understanding makes BERT unsuitable for text generation. Section 9.3 examines the complementary paradigm — GPT's autoregressive language modeling — which sacrifices bidirectionality in exchange for generative capability and, at scale, far more than compensates.

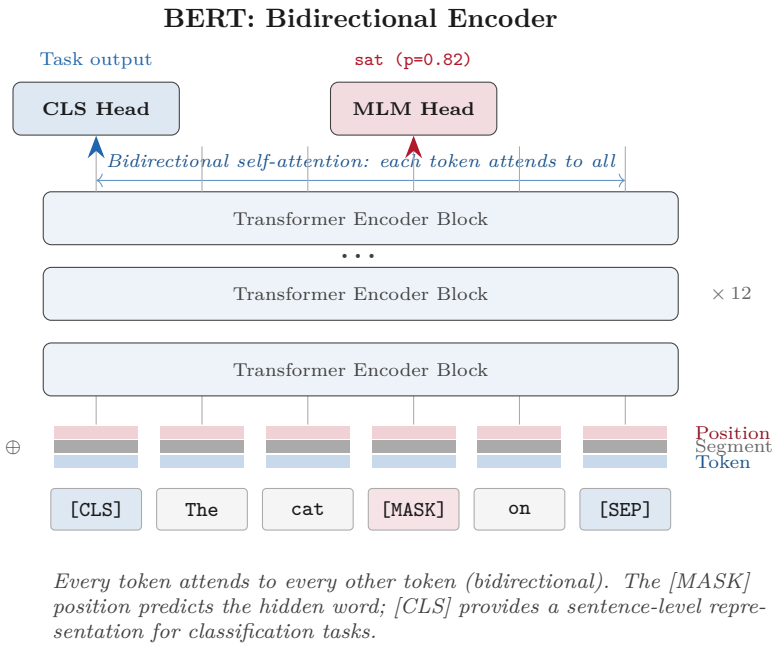

Figure 3: Figure 9.3 – BERT Architecture with Masked Language Modeling

---

9.3 GPT and Autoregressive Language Modeling

9.3.1 GPT Architecture and Causal Masking

The first GPT paper [Radford et al., 2018] appeared three months before BERT, in June 2018. At the time, its reception was somewhat muted: the NLP community had converged on a consensus that bidirectional context produces better representations for understanding tasks, and GPT's unidirectional design appeared to sacrifice too much. The decision by a small team at OpenAI — Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever — to commit to the autoregressive architecture was, in retrospect, one of the most consequential bets in the history of machine learning. Their reasoning was principled if contrarian: autoregressive training aligns naturally with text generation, left-to-right processing scales more simply, and the training objective has an elegant theoretical grounding in the chain rule of probability.

GPT is a decoder-only Transformer: it uses the same multi-head self-attention, position-wise feed-forward networks, residual connections, and layer normalization as the Transformer encoder (Chapter 8), but replaces bidirectional self-attention with *causal self-attention*. In causal self-attention, each position can attend only to itself and to positions that precede it in the sequence. This constraint is implemented by the causal mask: a lower-triangular binary matrix $\mathbf{M} \in \{0, -\infty\}^{T \times T}$ added to the attention logits before softmax, where $\mathbf{M}_{ij} = 0$ if $j \leq i$ and $\mathbf{M}_{ij} = -\infty$ if $j > i$. After adding $\mathbf{M}$, positions with $-\infty$ logits receive zero weight after softmax, effectively zeroing out all attention to

future positions. The full causal attention computation is:

$$\text{CausalAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top + \mathbf{M}}{\sqrt{d_k}}\right)\mathbf{V}.$$

GPT-1 contains 12 Transformer decoder blocks, $d_{\text{model}} = 768$, $h = 12$ attention heads, and approximately 117 million parameters. It was pre-trained on BookCorpus [Zhu et al., 2015], a dataset of approximately 7,000 unpublished books containing 800 million words of contiguous text. The use of long contiguous documents (books rather than web snippets) was a deliberate design choice to encourage the model to learn long-range dependencies across hundreds of tokens. GPT-1's input representation uses learned position embeddings and does not include segment embeddings, reflecting the absence of the two-sentence structure that BERT's NSP objective requires. Contrary to what one might expect, removing the segment embedding and the bidirectional attention mechanism does not cripple the model on understanding tasks: GPT-1 achieved state-of-the-art results on 9 of 12 evaluated NLP benchmarks through fine-tuning [Radford et al., 2018], demonstrating that autoregressive pre-training captures transferable linguistic knowledge comparable to bidirectional approaches at the same scale.

For fine-tuning, GPT-1 constructs task-specific input representations by linearizing structured inputs into sequences the model can process autoregressively. For a classification task with input text $\mathbf{x}$ and label y, the input is formatted as `[START]` $\mathbf{x}$ `[EXTRACT]`, where `[EXTRACT]` is a special token whose final-layer representation is passed to a linear classifier. For sentence pair tasks (entailment, similarity), the two sentences are concatenated with a delimiter token: `[START]` $\mathbf{x}_1$ `[DELIM]` $\mathbf{x}_2$ `[EXTRACT]`. This input linearization strategy allows GPT's autoregressive architecture to handle multi-sentence tasks without segment embeddings, at the cost of introducing artificial delimiter tokens that do not appear in pre-training. During fine-tuning, GPT-1 jointly optimizes the task-specific loss and the language modeling loss, with the language modeling objective weighted by a small coefficient $\lambda = 0.5$:

$$\mathcal{L}_{\text{GPT-finetune}} = \mathcal{L}_{\text{task}} + \lambda\, \mathcal{L}_{\text{CLM}}.$$

This joint optimization prevents catastrophic forgetting of the pre-trained language modeling capability during fine-tuning and serves as an auxiliary regularizer that improves generalization. The practice of retaining a language modeling auxiliary loss during fine-tuning was influential: variants of this idea appear in subsequent work on instruction tuning and continued pre-training.

9.3.2 The Causal Language Modeling Objective

Why is the standard language modeling objective from Chapter 2 — predicting the next token from all previous tokens — sufficient to teach a model virtually everything it needs to know about language?

The causal language modeling (CLM) objective is not a new invention: it is the same autoregressive factorization of joint probability that underlies n-gram models (Chapter 2), recurrent language models (Chapter 5), and every other language model in this book. What changes in GPT is the model's capacity to exploit this objective. The probability of a token sequence is decomposed by the chain rule as:

$$P(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_1, \ldots, w_{t-1}; \theta).$$

Maximizing the log-likelihood of this factorization over a training corpus is equivalent to minimizing the CLM loss:

$$\mathcal{L}_{\text{CLM}} = -\sum_{t=1}^{T} \log P(w_t \mid w_1, \ldots, w_{t-1}; \theta) \tag{9.3}$$

Every token in the sequence contributes exactly one term to the loss. For a sequence of T tokens, the CLM objective provides T gradient signals per forward pass, compared to only $0.15T$ signals for MLM. This efficiency advantage compounds across training: for the same number of forward passes over the corpus, a model trained with CLM sees roughly 6.7 times as many gradient signals as a model trained with MLM. The practical implication is that CLM-trained models can be trained with less data or fewer compute steps to achieve the same level of linguistic coverage, though the quality of each gradient signal differs: MLM signals benefit from bidirectional context, while CLM signals rely solely on left context.

Autoregressive text generation with GPT-2 using top-p (nucleus) sampling. The model extends the prompt token by token, each step conditioned on all preceding tokens.

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer

model = GPT2LMHeadModel.from_pretrained("gpt2")
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

# The prompt IS the task specification -- no fine-tuning required
prompt = "The key insight behind pre-training is that"
input_ids = tokenizer.encode(prompt, return_tensors="pt")

# Each generated token is conditioned on all previous tokens (CLM)
output = model.generate(input_ids, max_length=80,
    temperature=0.7, do_sample=True,
    top_p=0.9, no_repeat_ngram_size=2)
print(tokenizer.decode(output[0], skip_special_tokens=True))
```

The CLM objective also aligns directly with the generation task. At inference time, a GPT model generates text by sampling $w_t \sim P(w_t \mid w_{<t}; \theta)$ autoregressively, appending each sampled token to the context before generating the next. This inference procedure is identical to the training objective: the model is never asked to do anything during generation that it was not asked to do during training. This alignment between training and inference — training on predicting the next token, generating text by predicting the next token — is a key reason for GPT's strong generative quality and the remarkable coherence of its outputs at scale. In contrast, BERT's MLM training objective has no direct inference-time counterpart: the model was trained to predict `[MASK]` tokens but is never given `[MASK]` tokens during fine-tuning or downstream use.

The connection to perplexity (Chapter 2) is immediate and important. The CLM loss computed on a held-out test set, when exponentiated, gives the model's perplexity on that set: $\text{PP}(\mathbf{w}) = \exp(\mathcal{L}_{\text{CLM}})$. Perplexity therefore serves as the natural evaluation metric for CLM-trained models on language modeling benchmarks. Improvements in CLM pre-training quality — better architectures, more data, more training compute — manifest directly as lower perplexity on held-out text, and the field tracks these improvements systematically. As discussed in Chapter 11, perplexity decreases predictably as a power law of model size, dataset size, and compute budget, providing a quantitative basis for scaling decisions. For BERT, there is no comparable universal evaluation metric: BERT's quality as a pre-trained model is evaluated indirectly, by measuring downstream task performance

after fine-tuning. This makes it harder to compare pre-trained encoder models on a single scale analogous to perplexity, and has contributed to the field's shift toward CLM-trained models for which pre-training progress is directly measurable.

9.3.3 From GPT to GPT-2 to GPT-3: The Power of Scale

```
# Illustrating the scale of the GPT family
# No external imports required -- pure data illustration
gpt_models = {
    "GPT-1 (2018)":  {"params": "117M",  "data": "BookCorpus (800M words)",
                      "capability": "Fine-tuning on downstream tasks"},
    "GPT-2 (2019)":  {"params": "1.5B",  "data": "WebText (40GB)",
                      "capability": "Zero-shot task solving"},
    "GPT-3 (2020)":  {"params": "175B",  "data": "300B tokens (mixed)",
                      "capability": "In-context few-shot learning"},
}

# Display the scaling progression
for name, info in gpt_models.items():
    print(f"{name}: {info['params']} params, {info['data']}")
    print(f"  Key capability: {info['capability']}")
```

We confess to being among those who were skeptical that simply scaling up GPT would produce qualitatively new capabilities. The few-shot results of GPT-3 changed our minds. The progression from GPT-1 to GPT-3 is one of the most important empirical findings in the history of machine learning, not because of any architectural innovation — all three models are the same decoder-only Transformer, differing only in scale — but because it demonstrated that scaling an unchanged architecture produces qualitatively different behaviors, not merely quantitative improvements on existing benchmarks.

GPT-1 [Radford et al., 2018] established the autoregressive pre-training framework. Pre-trained on BookCorpus with 117 million parameters, it achieved strong benchmark results through fine-tuning, demonstrating that left-to-right language model pre-training captures transferable linguistic knowledge. The model had no zero-shot or few-shot capabilities; it required task-specific fine-tuning for downstream performance, following the same paradigm as BERT.

GPT-2 [Radford et al., 2019] scaled to 1.5 billion parameters trained on WebText, a dataset of 40 GB of web pages curated by selecting Reddit posts with at least three upvotes. The scale increase produced a qualitative surprise: GPT-2 exhibited zero-shot task-solving abilities. Without any fine-tuning, provided only with a natural language description of the task, GPT-2 could perform translation (`Translate English to French: The house is beautiful. '' $\to$`La maison est belle.'‘), reading comprehension, and summarization, simply by conditioning on appropriate prompts. The paper's title — "Language Models are Unsupervised Multitask Learners'' [Radford et al., 2019] — made a provocative claim: a model trained only on language modeling had implicitly learned to perform diverse NLP tasks because those tasks are implicit in the structure of natural language. GPT-2 also achieved state-of-the-art perplexity on seven of eight language modeling benchmarks in zero-shot evaluation, without any adaptation.

GPT-3 [Brown et al., 2020] scaled to 175 billion parameters trained on 300 billion tokens from a mixture of Common Crawl, WebText2, Books1, Books2, and Wikipedia. The scale produced a

further qualitative leap: *in-context learning*. Given a task description and k input-output examples in the prompt (without any gradient updates), GPT-3 could perform new tasks from context alone, with performance improving as k increased from 0 (zero-shot) to 1 (one-shot) to 32 (few-shot). Few-shot GPT-3 matched or approached fine-tuned BERT on several benchmarks, including TriviaQA (68.0% few-shot versus 72.5% fine-tuned) and translation tasks, despite requiring no parameter updates. The no-architectural-change point deserves emphasis: GPT-3 is the same decoder-only Transformer as GPT-1. No new attention mechanisms, no new training objectives, no new architectural components. The only changes were scale — more parameters, more data, more compute — and yet the behavioral difference between GPT-1 and GPT-3 is as large as the difference between a narrow domain-specific tool and a general-purpose language system.

9.3.4 Zero-Shot and Few-Shot Generalization

GPT-2 and GPT-3 revealed a capability that the pre-training paradigm had not explicitly designed for: large autoregressive models can perform tasks they were never explicitly trained for, by conditioning on natural language descriptions or examples provided in the prompt. This phenomenon operates in two modes. In *zero-shot generalization*, the model receives only a task description — "Translate the following English sentence to French:" — and must complete the task without any examples of the desired input-output mapping. In *few-shot generalization*, the model receives k labeled examples demonstrating the mapping before the test input, and must extrapolate to the test case. In neither mode are the model's parameters updated; the entire adaptation occurs through the attention mechanism processing the prompt context.

It is tempting to assume that few-shot learning is just gradient-based learning with very few examples. This is incorrect. In few-shot learning via in-context learning (ICL), the model parameters θ are completely frozen during the entire inference procedure. The model processes the k examples as part of its input context through forward-pass computation, with the examples influencing the output through the attention mechanism's soft nearest-neighbor lookup. The model's weights encode the linguistic and factual knowledge learned during pre-training; the in-context examples direct that knowledge toward the specific task through pattern matching in the attention computations. This is fundamentally different from any gradient-based adaptation procedure. The mechanism by which in-context learning actually works remains an active research area, and intellectual honesty requires acknowledging this: we know that ICL works and we have formal characterizations of what the attention mechanism computes over context examples, but the full account of why ICL capabilities emerge from scale and whether they constitute genuine generalization or sophisticated pattern matching is not yet settled [Brown et al., 2020]. Chapter 13 examines in-context learning in detail. What is already clear from the GPT lineage is that decoder-only models at sufficient scale develop remarkable generalization capabilities from autoregressive pre-training alone, without any explicit training for generalization.

Section 9.3 has traced the GPT lineage from GPT-1's demonstration that autoregressive pre-training transfers, through GPT-2's discovery of zero-shot capabilities, to GPT-3's revelation that scale enables in-context learning without parameter updates. The central lesson is that scale does not merely improve performance on existing tasks — it unlocks qualitatively new capabilities. Section 9.4 introduces the third pre-training paradigm, T5, which takes a different approach: rather than specializing toward either understanding (BERT) or generation (GPT), it casts every NLP task as text-to-text and pre-trains an encoder-decoder model with span corruption.

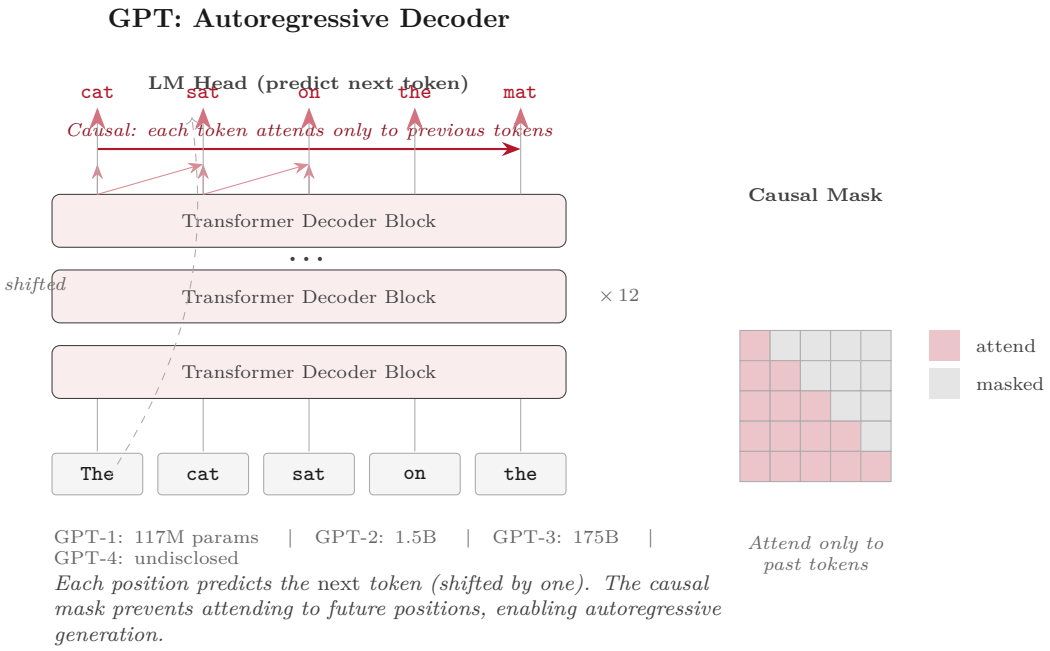

Figure 4: Figure 9.4 – GPT Architecture with Causal Language Modeling

9.4 T5 and Encoder-Decoder Pre-training

9.4.1 The Text-to-Text Framework

What if every NLP task — classification, translation, summarization, question answering — were simply a matter of converting one text string into another?

There is something almost comically simple about the central idea of T5: what if every NLP task were just a matter of taking some text in and producing some text out? Sentiment classification? Input: `classify: This film is extraordinary.'' Output:`positive." Translation? Input: `translate English to German: The cat sat on the mat.'' Output:`Die Katze saß auf der Matte." Question answering? Input: `question: What year was BERT published? context: BERT was published in 2019.'' Output:`2019." Summarization? Input: `summarize: [long article text].'' Output:`[summary text]." The text-to-text framework [Raffel et al., 2020] claims that this reduction is not merely a convenient notation but a genuine unification: one model, one loss function, one decoding procedure, and one training pipeline for all tasks.

The T5 architecture is an encoder-decoder Transformer (Chapter 8, Section 8.6.3). The encoder processes the task-prefixed input text and produces a sequence of contextual representations using bidirectional self-attention. The decoder generates the output text autoregressively, attending to both the decoder's own previously generated tokens through causal self-attention and to the encoder's output through cross-attention. This encoder-decoder design is the original Transformer architecture of Vaswani et al. [2017], adapted and scaled by Raffel et al. [2020] for pre-training. The critical distinction from BERT and GPT is that T5 does not require task-specific output heads. BERT classification requires a linear layer mapping the `[CLS]` representation to class logits; BERT NER requires per-token linear classifiers; BERT QA requires start- and end-position predictors. T5

requires none of these: every task is handled by the same decoder generating text tokens, using the same vocabulary-size softmax as the pre-training objective. This unification dramatically simplifies the training and deployment pipeline and removes the need for task-specific engineering beyond writing the task prefix.

The text-to-text framework is not merely a formatting convention; it is an architectural and computational unification. The encoder-decoder model is trained with the same cross-entropy loss, the same beam search decoding, and the same tokenizer for every task. Multi-task training (training on multiple tasks simultaneously by mixing task-prefixed examples) becomes straightforward, as all tasks share the same input-output interface. The T5 paper [Raffel et al., 2020] demonstrates this unification at scale: T5-11B achieved state-of-the-art results on GLUE (90.3%), SuperGLUE (88.9%), SQuAD (93.7% F1), and CNN/DailyMail summarization (43.5 ROUGE-L) simultaneously, using a single pre-trained model with task prefixes as the only task-specific component. The C4 dataset (Colossal Clean Crawled Corpus) used for T5's pre-training consists of approximately 750 GB of cleaned Common Crawl text, providing the scale of diverse web text necessary to support the model's broad task coverage.

T5 was released in five sizes: T5-small (60 million parameters), T5-base (220 million), T5-large (770 million), T5-3B (3 billion), and T5-11B (11 billion), all sharing the same text-to-text interface and training procedure. The progression across sizes reveals clear scaling behavior: performance on most tasks increases monotonically with model size, with the largest gains observed between T5-base and T5-large for language generation tasks and between T5-large and T5-3B for complex reasoning tasks. This scaling behavior validates the text-to-text unification as a robust framework that benefits from scale across diverse task types, rather than being tightly optimized for any particular task family. The SentencePiece tokenizer [Kudo and Richardson, 2018] used by T5, with a vocabulary of 32,000 subword units, is shared across all languages in the training corpus and provides multilingual coverage that enables cross-lingual transfer without any explicit multilingual training signal. The use of relative position biases rather than absolute position embeddings in T5's attention layers (using the approach of Shaw et al. [2018]) is another architectural departure from BERT and GPT: relative position biases encode the offset between query and key positions rather than absolute positions, making the model's representations more robust to sequence length variation and enabling modest generalization to sequences slightly longer than those seen during training.

T5 text-to-text inference: the same model, same interface, and same decoding procedure handle translation, summarization, and classification.

```
from transformers import T5ForConditionalGeneration, T5Tokenizer

model = T5ForConditionalGeneration.from_pretrained("t5-small")
tokenizer = T5Tokenizer.from_pretrained("t5-small")

# Same model, same API, different tasks -- T5's core unification
tasks = [
    "translate English to German: The house is wonderful.",
    "summarize: Pre-training on large corpora enables "
        "transfer learning across NLP tasks.",
    "stsb sentence1: The cat sat. sentence2: The feline rested.",
]
for task in tasks:
    inputs = tokenizer(task, return_tensors="pt",
```

```
                    max_length=128, truncation=True)
outputs = model.generate(**inputs, max_length=50)
print(f"Input:  {task}")
print(f"Output: {tokenizer.decode(outputs[0],
                 skip_special_tokens=True)}\n")
```

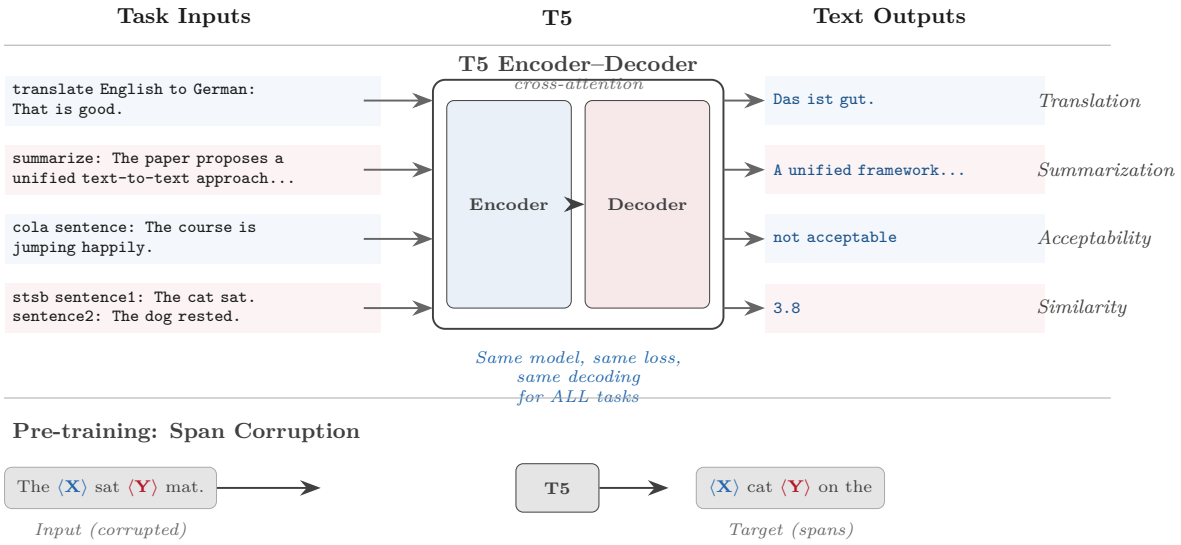

Figure 5: Figure 9.5 – T5 Text-to-Text Framework

9.4.2 Span Corruption Objective

T5's pre-training objective is *span corruption*, a generalization of BERT's masked language modeling that exploits the encoder-decoder architecture to provide richer training signal. Rather than masking individual tokens, span corruption selects contiguous spans of tokens for removal. Each selected span is replaced by a unique sentinel token (`<extra_id_0>`, `<extra_id_1>`, ...) in the encoder input. The decoder is trained to generate the sentinel tokens followed by the original content of each span, in order, terminated by a special end-of-sequence marker.

Formally, let $\mathbf{x} = (x_1, \ldots, x_T)$ be the input token sequence. Span corruption selects S non-overlapping spans $\{(l_s, r_s)\}_{s=1}^{S}$ from positions in $\mathbf{x}$, with span lengths drawn from a geometric distribution with mean length $\bar{L} = 3$ tokens and a total corruption rate of approximately 15% of input tokens. The corrupted encoder input $\mathbf{x}_{\text{corrupt}}$ replaces each span (l_s, r_s) with the sentinel token `<extra_id_(s-1)>`. The decoder target $\mathbf{y}$ is the sequence `<extra_id_0>` $x_{l_1} \cdots x_{r_1}$ `<extra_id_1>` $x_{l_2} \cdots x_{r_2} \cdots$, containing the original spans preceded by their sentinel identifiers. The span corruption loss is:

$$\mathcal{L}_{\text{span}} = -\sum_{t=1}^{|\mathbf{y}|} \log P(y_t \mid y_{<t}, \mathbf{x}_{\text{corrupt}}; \theta) \tag{9.4}$$

The key differences from BERT's MLM are instructive. First, the target is multi-token spans rather than individual tokens: the decoder must generate coherent multi-word phrases, not just classify each position independently. This requires the model to develop generative competence in the decoder, not merely classification competence. Second, the loss is applied autoregressively across the entire target sequence $\mathbf{y}$: if the corrupted spans contain a total of K tokens, the decoder generates a sequence of length approximately $K + S$ (the K original tokens plus S sentinel markers), and every generated token contributes to the loss. Third, the encoder processes a shorter sequence (with spans collapsed to sentinels) while the decoder handles the reconstruction task: the division of labor encourages the encoder to develop rich compressed representations of the uncorrupted context, while the decoder learns to generate coherent text from those representations. The span corruption objective thus simultaneously trains both the encoder (understanding the corrupted context) and the decoder (generating the original spans), making it well-suited for the encoder-decoder architecture in a way that MLM (which has no natural decoder role) is not.

A concrete example clarifies the span corruption procedure. Consider the input sentence `The Transformer architecture was introduced in 2017 by a team at Google Brain.''` `Span corruption might select the span`introduced in 2017" and the span `Google Brain,''` `producing the corrupted encoder input:`The Transformer architecture was `<extra_id_0>` by a team at `<extra_id_1>`." The decoder target would then be: "`<extra_id_0>` introduced in 2017 `<extra_id_1>` Google Brain `</s>`." The encoder must represent the full context surrounding each sentinel, while the decoder must reconstruct the missing spans from those representations in the correct order. Note that the encoder input is shorter than the original (two multi-token spans have been collapsed to two single sentinel tokens), while the decoder input/output is shorter than the original sentence but requires generating coherent phrase-level text. This asymmetric compression-and-reconstruction structure is a deliberate design choice: it forces the encoder to develop informationally dense representations that can guide accurate span reconstruction, rather than simply copying contextual information into the decoder. The T5 ablation [Raffel et al., 2020] compared span corruption against token-level corruption (replacing individual tokens rather than spans), prefix language modeling, and deshuffling (reordering randomly shuffled sentences), finding span corruption consistently superior across task types.

9.4.3 Lessons from Systematic Ablations

The T5 ablation results are, frankly, overwhelming in their scope — the paper reports hundreds of experiments — and we necessarily present only the conclusions most relevant to the practitioner. The value of the T5 paper [Raffel et al., 2020] extends beyond the T5 model itself: it is the field's first large-scale, controlled comparison of pre-training design choices, providing empirical grounding for decisions that had previously been made by intuition or analogy to computer vision. Understanding these findings is essential for any practitioner who must choose a pre-training strategy.

The most directly actionable findings concern pre-training objectives. Span corruption consistently outperforms both BERT-style token-level MLM and causal language modeling as a pre-training objective for encoder-decoder models across a wide range of tasks, with the advantage largest on generation tasks (summarization, translation) and smaller but persistent on understanding tasks (classification, QA). The key variable appears to be the multi-token generation requirement: objectives that require generating coherent token sequences develop stronger decoder representations than objectives that classify positions independently. The corruption rate of 15% and mean span length of 3 tokens were also found to be near-optimal: higher corruption rates hurt performance by destroying too much context, while longer mean span lengths reduced performance by making the

reconstruction task too difficult in early training.

On architectural choices, the T5 ablation found that encoder-decoder models slightly outperform decoder-only models at the same total parameter count across most tasks, with the advantage most pronounced on tasks requiring structured transformation (translation, summarization) rather than open-ended generation. Prefix language models — a hybrid where the encoder portion of a decoder-only model sees bidirectional context over a prefix while the decoder portion generates autoregressively — performed between the two extremes, confirming that bidirectional processing of the input is genuinely valuable for understanding-heavy tasks. These findings provide important context for the decoder-only dominance discussed in Section 9.5.2: the T5 ablations suggested encoder-decoder is slightly superior at fixed parameter count, yet the field subsequently converged on decoder-only models for large-scale deployments.

The data findings are equally clear: larger datasets consistently improve performance, with no evidence of saturation at the scales studied. The C4 dataset (750 GB) substantially outperforms smaller web corpora, and mixing multiple data sources (web, books, Wikipedia) outperforms any single source for general-purpose models. The T5 paper also found that multi-task training (jointly training on all tasks simultaneously using task prefixes) matches single-task fine-tuned performance on most tasks, validating the text-to-text unification as a practical training strategy and not merely a conceptual convenience.

One of the T5 paper's most practically useful findings concerns the relationship between model size and the number of training steps. Raffel et al. [2020] found that for a given compute budget, it is generally better to train a larger model for fewer steps than to train a smaller model for more steps, provided that the model has not been undertrained relative to its parameter count. This finding anticipates the Chinchilla scaling law [Hoffmann et al., 2022] (discussed in Chapter 11), which formalizes the optimal parameter-to-token ratio for a given compute budget. The T5 ablation did not have the scale to fully characterize this trade-off, but its directional finding — more parameters, fewer steps, within a fixed compute budget, is generally preferable — proved influential. Another important finding concerns regularization: dropout rates of 0.1 consistently improved performance across model sizes, and no weight decay was found to be needed when training with the Adafactor optimizer [Shazeer and Stern, 2018], which replaces the Adam optimizer's full second-moment estimate with a factorized approximation that dramatically reduces the optimizer's memory footprint. Adafactor is one of the reasons that T5-11B was trainable at all with the hardware available in 2019, and it has influenced optimizer design choices in subsequent large-scale pre-training efforts. Section 9.5 synthesizes these findings alongside the characteristics of BERT and GPT into a practical framework for paradigm selection.

9.5 Comparing Paradigms

9.5.1 Architecture-Task Suitability Matrix

When a practitioner faces a new NLP task, which of the three paradigms — encoder-only, decoder-only, or encoder-decoder — provides the most appropriate starting point?

The three pre-training paradigms are not interchangeable. Each has structural properties that confer advantages and limitations for specific task families, and understanding these properties enables principled model selection rather than defaulting to the largest available model. Figure~**??** presents a systematic comparison; we discuss the key dimensions here.

Dim	Encoder-Only (BERT)	Decoder-Only (GPT)	Enc-Dec (T5)
Directionality/Bidirectional/Left-to-right/Bi (enc) + L2R (dec)			
Pre-train objective/Masked LM (MLM)/Causal LM (CLM)/Span corruption			
Tokens trained per step/15% (masked only)/100% (all tokens)/15–50% (spans)			
Native generation/No/Yes/Yes			
Zero- and few-shot ICL/Weak/Strong at scale/Moderate			
Classification/Excellent/Good (via prompt)/Good			
NER and token tasks/Excellent/Good/Good			
Extractive QA/Excellent/Good/Good			
Summarization/Poor/Good/Excellent			
Translation/Poor/Good/Excellent			
Open generation/Poor/Excellent/Good			
Dominant scale/<1B params/1B–1T+ params/1–11B params			
Key models/BERT RoBERTa/GPT-2 GPT-4 LLaMA/T5 BART Flan-T5			

Figure 6: Figure 9.6 – Paradigm Comparison Table

Encoder-only models (BERT and its variants) are best suited for tasks where the primary requirement is producing a representation of the input, rather than generating output text. For classification tasks — sentiment analysis, textual entailment, document categorization — the bidirectional `[CLS]` representation provides rich context from the full input sequence, and a single linear layer suffices as the task head. For token-level understanding tasks such as NER, the bidirectional per-token representations capture both left and right linguistic context for each entity span. For extractive question answering, bidirectional context enables the model to ground answer spans in their surrounding passage context effectively. Encoder-only models are computationally efficient for these understanding tasks: at inference time, a single forward pass through the encoder produces the needed representation, without the sequential token-by-token generation required by decoder-based models.

The computational profile of encoder-only models at inference deserves attention. Because the entire input sequence is processed in a single forward pass with no autoregressive loop, encoding is embarrassingly parallelizable: all T positions are computed simultaneously. For a sequence of length T and model dimension d, the encoding cost is $\mathcal{O}(T^2 d)$ for the attention layers (quadratic in sequence length due to the $T \times T$ attention matrix) and $\mathcal{O}(Td^2)$ for the feed-forward layers, with the latter dominating for typical model dimensions where $d > T$. This fixed-cost forward pass makes encoder-only models highly suitable for production environments where throughput is critical: batches of classification queries can be processed in parallel with predictable latency. Decoder-only models, by contrast, generate each token sequentially, with each generation step requiring a full forward pass over the growing KV cache: the total generation cost for a sequence of T tokens is $\mathcal{O}(T \cdot Td) = \mathcal{O}(T^2 d)$, but this cost is sequential rather than parallel, imposing a minimum latency proportional to the output length. For applications where response latency is critical — real-time

content moderation, high-throughput document classification in search pipelines — encoder-only models remain the architecture of choice regardless of the superior capabilities of large decoder-only models on other dimensions.

Decoder-only models (GPT and its successors) are essential for any task that requires generating free-form text: language generation, story continuation, code synthesis, dialogue, and open-ended question answering. At sufficient scale, decoder-only models can perform understanding tasks through in-context learning without any fine-tuning, making them general-purpose systems. The alignment between the CLM training objective and the generation inference procedure is a fundamental advantage: the model is never asked to do anything during deployment that it was not trained to do during pre-training. You might expect this alignment advantage to be less important at large scale, but empirically the opposite is true: the alignment becomes more valuable as scale increases, because the model's generative capability — and therefore its ability to reformulate understanding problems as generation problems — improves with scale in ways that classification capability does not.

Encoder-decoder models (T5, BART) are particularly well suited for *structured transformation* tasks: tasks where the input and output have distinct structures and the output is not simply a classification of the input but a substantive textual transformation. Translation, summarization, paraphrase generation, and code-to-documentation tasks all fall in this category. The encoder processes the input bidirectionally, building rich representations of its structure, while the decoder generates the output with access to those full-context representations through cross-attention. This division of labor is especially valuable when the input is long and structured (a news article, a source code function, a legal document) and the output is a concise, structured transformation of it.

9.5.2 Why Decoder-Only Models Won the Scaling Race

If we had to identify the single most consequential technical observation of the 2020–2025 period, it would be this: decoder-only models scale better than encoder-decoder models for the same compute budget. The dominance of decoder-only architectures in frontier AI systems — GPT-4, Claude, LLaMA, Gemini, Mistral, and virtually every large-scale system deployed commercially since 2022 — represents a decisive empirical verdict. Understanding why this happened requires examining three interacting factors.

The first factor is training efficiency. CLM trains on every token in the input sequence, providing T gradient signals per forward pass. MLM provides only $0.15T$ signals per forward pass, meaning that for the same number of gradient updates, CLM trains on approximately 6.7 times as much information. At the scale of hundreds of billions of tokens, this efficiency difference compounds dramatically: a model trained with CLM for C compute units sees roughly the same number of training examples as an MLM model trained for $6.7C$ compute units. As total training compute has grown from GPU-days to GPU-years, the efficiency advantage of CLM has translated into substantial capability differences at fixed compute budget. The T5 ablation [Raffel et al., 2020] found that encoder-decoder slightly outperforms decoder-only at the same parameter count, but this advantage shrinks when compute rather than parameters is held constant, because CLM's training efficiency compensates for the parameter-count disadvantage.

The second factor is capability generality. A model that can generate coherent, factually grounded text about a topic necessarily understands that topic at some level. Generation subsumes understanding in a profound sense: to produce a grammatically and semantically appropriate continuation of `The Treaty of Versailles was signed in \_\_\_'' as`1919, ending the First World War,"

the model must have encoded the relevant historical facts in its representations. Conversely, a model that can only classify inputs into categories has no internal pressure to develop generative representations of factual content. The generation objective thus provides a richer, more demanding training signal that incidentally develops understanding as a prerequisite.

The third factor is in-context learning (ICL). As demonstrated by GPT-3 [Brown et al., 2020], decoder-only models at sufficient scale can perform new tasks from examples provided in the prompt without any parameter updates. This emergent capability effectively eliminates the need for task-specific fine-tuning for many applications, reducing the engineering effort for deploying a new application from `collect labeled data, fine-tune, evaluate, deploy'' to`write a prompt, evaluate, deploy." Encoder-only models do not exhibit comparable ICL: because BERT processes input bidirectionally and produces representations rather than text, it cannot naturally reformulate task demonstrations as input-output examples for the model to continue. The decoder-only generation mechanism, which simply extends the prompt with appropriate continuation text, provides the natural substrate for ICL at scale.

There is also a more subtle argument for decoder-only dominance related to the loss function's coverage of the input. Consider two models of identical parameter count: one encoder-only trained with MLM on a corpus of N tokens, and one decoder-only trained with CLM on the same corpus. The CLM model receives gradient signal from all N tokens; the MLM model receives signal from only $0.15N$ tokens. Over a typical pre-training run of hundreds of billions of tokens, this difference is enormous. The CLM model has, in effect, seen the same training corpus 6.7 times more densely in terms of learning signal, which translates into more thoroughly trained representations at every parameter. This does not mean CLM learns more per token — the quality of each gradient signal is different (unidirectional versus bidirectional context) — but it does mean that at fixed compute, CLM tends to produce models that generalize better across the broad distribution of tasks that constitute modern language AI benchmarks. The combination of training efficiency, generative capability, and emergent in-context learning has made decoder-only models the dominant architecture for frontier AI systems, a dominance that shows no sign of reversing as of the time of writing.

The practical consequence of these three factors is that the field has converged on decoder-only architectures for large-scale models. Encoder-only models remain competitive for understanding-only tasks at small to moderate scale, where their bidirectional context advantage and computational efficiency are not overwhelmed by the decoder-only models' scaling advantages. But the trajectory is clear: as scale increases and compute costs allow, decoder-only models' versatility advantage grows relative to encoder-only models' efficiency advantage on specific task families.

9.5.3 Practical Guidelines for Model Selection

A natural but incorrect intuition is that one should always use the largest available model for any NLP task. The right model selection depends on the deployment scenario, the task requirements, the available labeled data, and the computational budget, and the decision is rarely simply "largest model wins." We offer a structured set of guidelines for practitioners navigating the paradigm landscape.

For understanding-only tasks with tight latency or cost requirements, fine-tuned BERT-size encoder models (110–340 million parameters) are frequently the best practical choice. A fine-tuned BERT-base model achieves competitive performance on sentiment classification, NER, and extractive QA while requiring a single forward pass at inference time, with no autoregressive generation loop. Compared to prompting a 70-billion-parameter decoder-only model, a fine-tuned BERT-base is

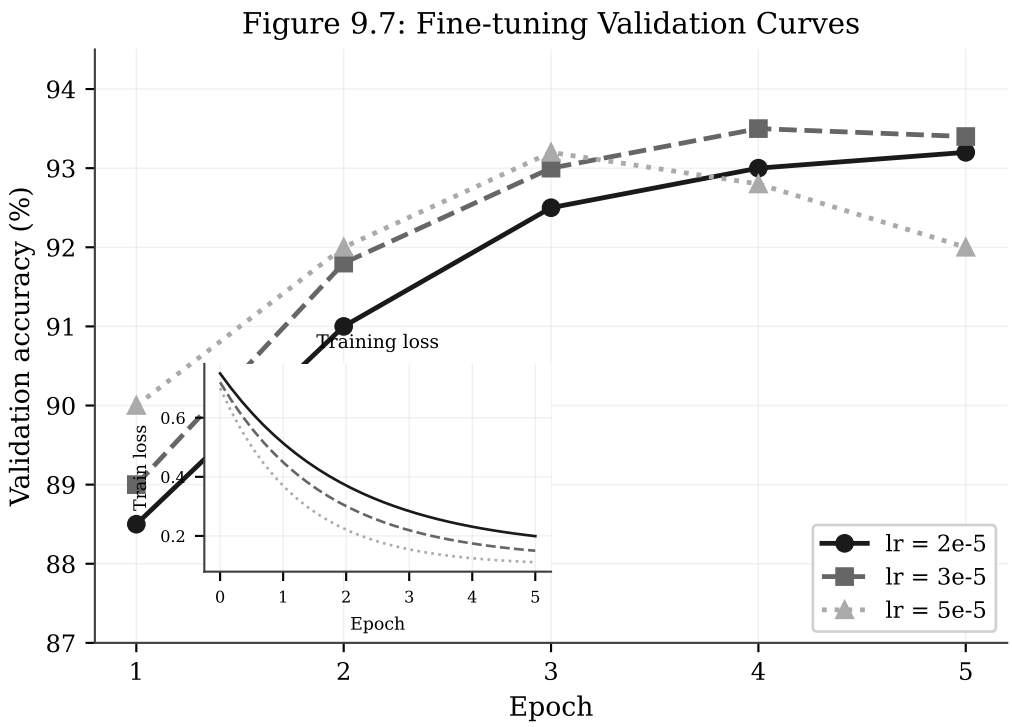

Figure 7: Figure 9.7 – Fine-tuning Validation Accuracy Curves on SST-2

approximately 600 times cheaper to serve at equivalent throughput. For organizations deploying NLP at scale, this cost differential is decisive. The appropriate use of encoder-only models has diminished somewhat as model efficiency has improved, but they remain the right choice for high-throughput classification applications, real-time NER in production systems, and any scenario where the latency of autoregressive generation is unacceptable.

For generation tasks of any kind — text completion, summarization with abstraction, dialogue, question answering requiring synthesis, code generation — decoder-only models are the only architecturally appropriate choice. The CLM training objective aligns perfectly with the generation inference procedure, and the resulting fluency and coherence of large decoder-only models is consistently superior to BERT-style models adapted for generation through awkward constrained decoding schemes. For structured input-to-output tasks such as translation, abstractive summarization, and document transformation at moderate model sizes, encoder-decoder models like T5 and BART remain competitive and offer the advantage of efficient encoder processing of long inputs combined with quality decoder generation.

For general-purpose applications where the task requirements are not fully specified in advance, or where a single model must serve multiple diverse task types, large decoder-only models with in-context learning are the most practical choice. The ability to specify task behavior through prompts rather than labeled data and fine-tuning pipelines reduces deployment complexity substantially, and the broad coverage of large pre-trained decoder-only models means that most tasks can be handled at acceptable quality without any fine-tuning. The cost of this flexibility is higher per-query compute and the inability to guarantee the same level of performance on specific tasks that a fine-tuned specialist model provides. For practitioners who must choose between these poles, a

reasonable heuristic is: if the task is well-defined, the volume is high, and latency matters, fine-tune a task-specific encoder model; if the task is evolving, the volume is modest, or multi-task flexibility is required, use a large decoder-only model with prompting.

A practical consideration that often goes unexamined in academic treatments is the cost of model updates. Fine-tuned task-specific models — whether encoder-only or encoder-decoder — must be re-trained whenever the underlying pre-trained model is updated or when the task distribution shifts. A system that serves 20 different NLP tasks using 20 fine-tuned BERT models requires re-training 20 models whenever a better pre-trained checkpoint becomes available. A system that serves the same 20 tasks using a single large decoder-only model with prompts requires updating only the prompts. As organizations have scaled their NLP deployments, the maintenance cost of managing large portfolios of fine-tuned models has become a significant practical concern, further shifting the balance toward large general-purpose decoder-only models. The emergence of parameter-efficient fine-tuning methods — LoRA [Hu et al., 2021], prefix tuning [Li and Liang, 2021], and adapter layers [Houlsby et al., 2019] — has partially addressed this concern by reducing the cost of fine-tuning to updating a small number of additional parameters, but the fundamental maintenance trade-off between specialist fine-tuned models and general-purpose prompted models remains an active engineering consideration. Chapter 12 discusses alignment fine-tuning, which applies parameter-efficient methods at scale to adapt large decoder-only models to specific behavioral requirements without full parameter updates.

The selection decision is ultimately an engineering tradeoff, and the right answer changes as model capabilities, hardware efficiency, and serving infrastructure evolve. What does not change is the fundamental insight: the three pre-training paradigms — encoder-only MLM, decoder-only CLM, and encoder-decoder span corruption — each encode different assumptions about what a language model should be able to do, and those assumptions translate into concrete performance advantages for specific task families. Understanding these assumptions, rather than treating model selection as an empirical search over a checklist, is what enables principled deployment decisions.

Chapter 9 has traced the arc of the pre-training revolution through three foundational paradigms. BERT established that bidirectional language model pre-training produces transferable representations of extraordinary quality, enabling state-of-the-art performance across understanding tasks with minimal labeled data. GPT demonstrated that autoregressive language model pre-training scales more efficiently, with quality emergent capabilities — zero-shot task solving, in-context learning — appearing as scale increases. T5 unified diverse task families under the text-to-text framework, provided systematic empirical grounding for pre-training design choices, and demonstrated that encoder-decoder models excel at structured transformation tasks. Together, these three paradigms define the landscape within which all subsequent work in large language models operates.

The thread connecting all three paradigms is the one established in Chapter 1: all three are, at their core, about predicting tokens. BERT predicts masked tokens from bidirectional context. GPT predicts the next token from left context. T5 predicts corrupted spans from the encoder representation. The differences in which tokens are predicted, and what context is available when predicting them, produce radically different models with distinct strengths and scaling properties. Pre-training harnesses the prediction objective at scale to develop representations that encode linguistic structure, world knowledge, and reasoning capacity — all from the simple imperative to predict the next word.

We have, however, glossed over a critical question throughout this chapter: what does "token"

actually mean? When BERT masks a token, or GPT predicts the next token, or T5 corrupts a span, what is the fundamental unit of text being predicted? A word? A character? A subword fragment? The answer turns out to matter enormously for vocabulary size, sequence length, handling of rare words, and cross-lingual transfer. Word-level tokenization creates vocabulary explosion and cannot handle novel words. Character-level tokenization produces sequences too long for practical Transformer training. The solution — subword tokenization via Byte-Pair Encoding, WordPiece, and SentencePiece — is the subject of Chapter 10. Alongside tokenization, Chapter 10 examines the data curation pipelines that produce the massive corpora on which BERT, GPT, and T5 were trained. The choice of tokenizer and the quality of the training data are the hidden infrastructure that determines, ultimately, what any language model can learn.

The success of pre-trained models raises practical questions that provide a natural transition to Chapter 10, where we examine how tokenization choices and data curation affect model quality at scale.

Exercises

Exercise 9.1 (Theory — Basic). Derive the gradient of the MLM loss with respect to the logits at a single masked position m. Let $\mathbf{z}_m \in \mathbb{R}^{|V|}$ denote the pre-softmax logit vector at position m. Show that $\partial \mathcal{L}_{\text{MLM}} / \partial z_{m,v} = P(v \mid \mathbf{w}_{\setminus \mathcal{M}}; \theta) - \mathbf{1}[v = w_m]$, where w_m is the true token. Explain why only masked positions contribute gradients, and quantify the per-example gradient efficiency of MLM compared to CLM for a sequence of T tokens.

Exercise 9.2 (Theory — Intermediate). Compare the information-theoretic efficiency of MLM and CLM. For a sequence of $T = 512$ tokens, compute the expected number of bits of training signal per forward pass for each objective, assuming that each correctly predicted token contributes $H(P_{\text{true}})$ bits of information on average. Argue whether the bidirectional context advantage of MLM (richer information per prediction) outweighs the efficiency advantage of CLM (more predictions per forward pass) for a fixed compute budget. State the assumptions under which each objective is preferable.

Exercise 9.3 (Theory — Intermediate). Explain why Next Sentence Prediction (NSP) was dropped in RoBERTa. Design an ablation experiment that would cleanly isolate NSP's effect on downstream performance, identifying the confounds that make such isolation difficult in practice. What do the RoBERTa results [Liu et al., 2019] actually demonstrate, and what do they leave uncertain?

Exercise 9.4 (Theory — Advanced). Prove that in-context learning as performed by GPT-3 does not involve gradient-based learning. Formalize the distinction between in-context learning (ICL) and fine-tuning: specify what changes during each procedure (parameters θ, input context, or both), and describe how the attention mechanism over in-context examples implements a form of soft nearest-neighbor lookup. What theoretical properties of self-attention enable a frozen model to condition its outputs on in-context demonstrations?

Exercise 9.5 (Programming — Basic). Using the Hugging Face Transformers library, compare BERT and GPT-2 on a fill-in-the-blank task. For the sentence `The [MASK] of France is Paris,'' use BERT's \texttt{fill-mask} pipeline to predict the masked token. For GPT-2, provide the prefix`The" and measure the probability assigned to various completions (`capital,''`president," "language"). Which model performs better and why? Write code that runs on CPU only (no GPU required) and reports the top-3 predictions for each model.

Exercise 9.6 (Programming — Intermediate). Fine-tune `bert-base-uncased` and `gpt2` on the SST-2 sentiment classification dataset using 2,000 training examples. For GPT-2, adapt the model by using the last token's representation as the classification feature. Compare: (a) validation accuracy at convergence, (b) epochs required to reach 85% accuracy, and (c) training time per epoch. Plot the fine-tuning loss curves for both models on the same axes and interpret the convergence patterns.

Exercise 9.7 (Programming — Intermediate). Run the `t5-small`, `t5-base`, and `t5-large` models on three text-to-text tasks (translation, sentiment classification via textual output, and summarization) using the Hugging Face pipeline with appropriate task prefixes. Measure the quality of each model's output (use BLEU for translation, accuracy for classification, ROUGE-L for summarization). Plot performance versus model size for each task. What does the scaling pattern reveal about the relative task difficulty for the T5 architecture?

Exercise 9.8 (Programming — Advanced). Implement the BERT 80/10/10 masking strategy from scratch. Given a tokenized sequence of 128 tokens, select 15% of positions uniformly at random, apply the 80/10/10 strategy (80% `[MASK]`, 10% random vocabulary token, 10% unchanged), and compute the MLM loss using a pre-trained `BertForMaskedLM`. Verify your implementation against Hugging Face's built-in `DataCollatorForLanguageModeling` by checking that both produce the same expected loss (within numerical tolerance) over multiple random seeds.

Exercise 9.9 (Programming — Advanced). Replicate a small-scale in-context learning experiment using GPT-2. On the SST-2 sentiment task, test zero-shot (task description only), one-shot (one labeled example), and five-shot (five labeled examples) performance. Format prompts as: "Review: [text] Sentiment: [positive/negative]." Measure accuracy as a function of the number of in-context examples. Is the ICL effect consistent across multiple prompt orderings? Compare your results to those of Brown et al. [2020] for GPT-3 and discuss the effect of model scale on ICL quality.

Exercise 9.10 (Theory — Advanced). Construct a comprehensive comparison table of the three paradigms across the following eight dimensions: (1) pre-training objective, (2) attention directionality, (3) FLOPs per token during training, (4) FLOPs per token during inference, (5) memory footprint relative to model size, (6) generation capability, (7) understanding capability at fixed parameter count, and (8) dominant scale regime. For each cell, provide a one-sentence justification. Based on your table, identify two specific scenarios where a fine-tuned BERT-base model would be a better practical choice than a 7-billion-parameter decoder-only model.

References

- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT*, 4171–4186.
- Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving Language Understanding by Generative Pre-Training. OpenAI Technical Report.
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners. OpenAI Technical Report.
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., et al. (2020). Language Models are Few-Shot Learners. In *Advances in NeurIPS*, 1877–1901.

- Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., et al. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *JMLR*, 21(140), 1–67.
- Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv:1907.11692.
- Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep Contextualized Word Representations. In *Proceedings of NAACL-HLT*, 2227–2237.
- Howard, J. & Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification. In *Proceedings of ACL*, 328–339.
- Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2020). ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *Proceedings of ICLR*.
- Clark, K., Luong, M.-T., Le, Q. V., & Manning, C. D. (2020). ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In *Proceedings of ICLR*.
- He, P., Liu, X., Gao, J., & Chen, W. (2021). DeBERTa: Decoding-enhanced BERT with Disentangled Attention. In *Proceedings of ICLR*.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. In *Advances in NeurIPS*, 5998–6008.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in NeurIPS*, 1097–1105.