HOW TRANSFORMERS ACTUALLY WORK

FROM FIRST PRINCIPLES TO PRODUCTION INFERENCE

HAMISH CALLAGHAN

How Transformers Actually Work

From First Principles to Production Inference

Steve Publications

This book is available at https://leanpub.com/howtransformersactuallywork

This version was published on 2026-09-22

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

From First Principles to Production Inference

This book takes you from the ground up through everything you need to know about transformer architectures. We will start with the minimal math required, then build a transformer component by component using NumPy and PyTorch, trace the complete training lifecycle from data preparation to checkpoint saving, and scale up to modern production techniques including distributed training, FlashAttention, KV caching, quantization, and serving systems. Every chapter includes complete, runnable code, concrete numerical examples, and rigorous derivations of the key equations. The goal is not to teach you how to call a library but to give you a genuine understanding of what happens inside a transformer and the ability to implement, inspect, optimize, and deploy one yourself.

Chapter 1: What Problem Are We Solving?

The Sequence Modeling Problem

Natural language is a sequence. You cannot understand a sentence by looking at its words in isolation. The meaning of "bank" changes depending on whether the previous word is "river" or "money". The word "they" refers to something mentioned earlier. Negation moves around ("I don't think she said that"). Language depends on structure at multiple scales: phonemes form syllables, syllables form words, words form phrases, phrases form sentences, sentences form paragraphs, and paragraphs form narratives.

To teach a machine to work with language, you must build a system that can read a sequence of symbols and compute a representation for each position that incorporates information from other positions in the sequence. This is the sequence modeling problem. More formally, given an input sequence of length n represented as vectors x_1, x_2, ..., x_n, a sequence model produces an output sequence z_1, z_2, ..., z_n where each z_i depends on the full context.

This same problem appears across domains: time series prediction, speech recognition, protein folding, video understanding, music generation. A solution that works for language often generalizes to other sequence tasks.

The core challenges are straightforward to state and harder to solve. First, sequences are variable length. A tweet and a book chapter are both sequences of tokens but differ by orders of magnitude in length. Second, relevant information can be far away. In the sentence "The animal didn't cross the street because it was too tired," the pronoun "it" refers to "animal," several words away. Third, the system must be compositional: it should work on entirely new sequences, not just memorize patterns seen during training.

Modern transformers solve all three challenges simultaneously using a single architectural mechanism: attention.

What Came Before: RNNs, LSTMs, and Their Limits

Before transformers became dominant, recurrent neural networks were the standard approach to sequence modeling. An RNN processes a sequence step by step, maintaining a hidden state h_t that accumulates information over time. At each time step, it combines the current input x_t with the previous hidden state h_{t-1} to produce a new hidden state:

h_t = f(W x_t + U h_{t-1} + b)

where W, U, and b are learned parameters and f is an activation function such as tanh or ReLU. The output at each step is computed from h_t.

This simple recurrence captures the intuition of processing language left to right: you read words one at a time and update your internal understanding. The hidden state acts as a compressed memory of everything seen so far.

The problem with this design becomes clear when sequences get long. Gradients flow backward through the same parameters U at every time step during training. After enough steps, repeated multiplication by U either shrinks the gradient toward zero (vanishing gradient) or blows it up (exploding gradient). This was recognized early: Bengio, Simard, and Frascati formalized the issue in 1994 [1]. The practical consequence is that vanilla RNNs cannot learn dependencies that span more than roughly ten to twenty positions.

Long Short-Term Memory networks address this with gating mechanisms. An LSTM maintains a cell state c_t that can pass information unchanged across many steps, controlled by learned gates:

- Forget gate: decides what to discard from c_{t-1}
- Input gate: decides what new information to add
- Output gate: decides what to expose as h_t

Mathematically, the cell state update looks like:

c_t = f_t * c_{t-1} + i_t * g_t

where * denotes element-wise multiplication. If the forget gate f_t outputs values close to one, information flows through unchanged, solving the vanishing gradient problem in principle.

LSTMs and their simplified variant GRU performed well for many years and remain useful for certain tasks. They captured long-range dependencies

far better than vanilla RNNs and were the backbone of the first major neural language models.

Yet they share a fundamental flaw: sequential computation. An RNN or LSTM must process position 1 before position 2, position 2 before position 3, and so on. This creates two problems. First, training is slow because you cannot parallelize across time steps. A sequence of 1000 tokens requires 1000 sequential operations, regardless of how powerful your GPU is. Second, the maximum path length between any two positions scales linearly with sequence length, making it hard for information to propagate across very long sequences even with gating.

Transformers eliminate both limitations by removing recurrence entirely.

Why Parallelism Matters

Let us be concrete about what parallelism buys you. Modern GPUs are designed to execute thousands of operations simultaneously. A single NVIDIA A100 GPU can perform roughly 312 teraflops of FP16 matrix multiplication [2]. These numbers are meaningless unless you give the GPU work that can actually be parallelized.

With an RNN, your sequence of n tokens requires exactly n sequential steps. Each step waits for the previous one. Your GPU sits idle for most of the time, waiting for the next token. You might parallelize across different examples in a batch, but within each sequence, you are bottlenecked by the recurrence.

A transformer computes all positions simultaneously. Every token attends to every other token in parallel. A single matrix multiplication processes the entire sequence at once. This changes the training time complexity in a way that directly maps to hardware reality. On a sequence of 1024 tokens processed by 8 GPU cores, an RNN takes 1024 steps while a transformer takes roughly 128 steps of parallel work.

The practical impact was immediate when the transformer paper was published. The authors reported training their big model for 3.5 days on 8 P100 GPUs. Comparable models using recurrence would have required weeks or months [3]. Parallelization is not just a performance optimization; it is what made training billion-parameter language models feasible within reasonable time and cost.

The Attention Insight

Attention is not new to machine learning. Bahdanau, Cho, and Bengio introduced attention mechanisms for neural machine translation in 2014 [4], allowing a decoder to selectively look at different parts of the source sentence when generating each output word. Luong et al. simplified and generalized this in 2015 [5]. In these early systems, attention was an add-on to RNN encoders and decoders: the RNN still did the heavy lifting of sequence processing, while attention helped the decoder find relevant source positions.

The breakthrough of the 2017 paper "Attention Is All You Need" was not inventing attention but asking what happens if attention alone does all the work. What if you remove recurrence and convolution entirely and build a sequence model based solely on attention mechanisms?

The answer turned out to be remarkably effective. The key insight can be stated simply: instead of compressing a sequence into a single hidden state and hoping that state carries enough information, why not let each position explicitly look at every other position whenever it needs to?

In self-attention, each token computes a weighted combination of all token representations in the sequence. The weights are learned dynamically based on the content of the tokens, not fixed in advance. If position i is currently thinking about subject-verb agreement, the attention mechanism can learn to focus on the subject position. If position j needs to resolve a pronoun, it can attend back to the antecedent. Each position chooses what to attend to independently, based on its current computational needs.

This is more flexible than a fixed hidden state because it preserves the full information of all positions and lets each position query it as needed. It is also more parallelizable because all queries happen simultaneously. And it generalizes because the attention weights are computed by learned functions of the input, not hardcoded rules.

The paper showed that a model based purely on stacked self-attention layers plus feed-forward networks matched or beat the best RNN-based models on machine translation tasks, with dramatically faster training. The architecture was general enough to work for parsing, question answering, and eventually every major language task.

A Map of This Book

The book is organized as a progressive build. We start with the mathematical concepts you need, then construct a transformer piece by piece, exposing every operation and every tensor shape transformation. Along the way, you will implement everything first in pure NumPy to see what is happening at the computational level, then transition to PyTorch for realistic training runs.

Chapters 2-4 cover foundations: tensors and matrix operations, neural network basics including backpropagation and optimizers, and tokenization with embeddings. These chapters assume basic linear algebra and Python but do not assume prior deep learning experience.

Chapters 5-8 introduce the core transformer components: scaled dot-product attention, multi-head attention, positional encodings, and feed-forward networks. Each chapter derives the key equations, shows a complete implementation, and includes numerical examples that trace actual values through the computation.

Chapters 9-12 build and train a complete transformer from scratch. Chapter 9 implements a full encoder stack in NumPy. Chapter 10 adds the decoder with causal masking and autoregressive generation. Chapter 11 derives gradients through the transformer by hand and verifies them numerically. Chapter 12 transitions to PyTorch and walks through the complete training loop: data preparation, batching, forward pass, loss computation, backward pass, optimization, checkpointing, and evaluation.

Chapters 13-17 cover modern architectures. Chapter 13 explains GPT-style decoder-only models, the architecture that dominates modern large language models. Chapter 14 covers BERT and encoder-only models. Chapter 15 discusses encoder-decoder models like T5 and instruction-tuned systems. Chapter 16 explains mixture-of-experts and scaling techniques. Chapter 17 covers long-context mechanisms including grouped-query attention, sliding window attention, and context extension methods.

Chapters 18-20 focus on training engineering at scale. Chapter 18 covers initialization, optimizers, learning-rate schedules, and hyperparameter recipes. Chapter 19 explains distributed training strategies: data parallelism, tensor parallelism, pipeline parallelism, and ZeRO. Chapter 20 covers memory management including mixed precision, gradient accumulation, and activation checkpointing.

Chapters 21-24 cover inference and serving. Chapter 21 explains KV caching, continuous batching, and speculative decoding. Chapter 22 covers quantization formats and techniques. Chapter 23 connects everything to GPU hardware, explaining FlashAttention, tensor cores, and memory hierarchy. Chapter 24 discusses production serving architectures and cost modeling.

Chapters 25-26 step back to analyze what we have built. Chapter 25 discusses representational power: what attention can and cannot compute, inductive biases, and systematic failure modes. Chapter 26 traces the historical development of attention-based architectures and surveys post-transformer alternatives. The conclusion synthesizes the key insights into a unified picture.

If you read and work through all the code, you will be able to implement a transformer from scratch, understand the design choices in any modern language model, debug training runs at the tensor level, optimize inference for your hardware, and reason about architectural trade-offs. That is the promise of this book.

Chapter 2: Tensors, Shapes, and Matrix Multiplication

Tensors as n-Dimensional Arrays

A tensor is an n-dimensional array of numbers. That is the complete definition you need for deep learning. You do not need the physics definition involving coordinate transformations. You need to understand how data is organized in memory, how operations combine tensors of different shapes, and how to track dimensions as data flows through a neural network.

The dimensionality is called the rank. A scalar has rank 0. A vector has rank 1. A matrix has rank 2. A 3D array has rank 3, and so on. The size along each dimension is its shape. A vector of length 5 has shape (5,). A matrix with 3 rows and 4 columns has shape (3, 4). A tensor with dimensions 2, 3, and 4 has shape (2, 3, 4).

In transformer code, you will encounter shapes up to rank 4 or occasionally rank 5. The conventions are consistent. The last dimension almost always represents the feature dimension: the size of each vector representation. The second-to-last dimension often represents sequence length or number of attention heads. The first dimension is typically the batch size: the number of examples processed simultaneously.

Here is how NumPy represents tensors of different ranks:

```python
import numpy as np

# Rank 0: scalar
s = np.array(3.14)
print(s.shape)  # ()

# Rank 1: vector
v = np.array([1.0, 2.0, 3.0, 4.0])
print(v.shape)  # (4,)

# Rank 2: matrix
m = np.array([[1, 2, 3],
              [4, 5, 6]])
print(m.shape)  # (2, 3)

# Rank 3: batch of matrices or sequence of vectors
t3 = np.random.randn(2, 4, 8)  # 2 sequences of 4 tokens, each with 8 features
print(t3.shape)  # (2, 4, 8)

# Rank 4: common in transformers: batch, heads, seq_len, head_dim
t4 = np.random.randn(4, 8, 16, 64)
print(t4.shape)  # (4, 8, 16, 64)
```

The key insight for reading transformer code is that higher-dimensional tensors are often just batches of lower-dimensional operations. A rank-4 tensor of shape (batch, heads, seq_len, head_dim) is a batch of batches of matrices: batch groups examples, heads groups independent attention computations, and each head operates on a (seq_len, head_dim) matrix. The underlying operations are always the same: matrix multiplication, element-wise operations, and dimension reshaping.

Element-Wise Operations

Element-wise operations apply a function independently to each element of a tensor. Addition, subtraction, multiplication, division, and most activation functions are element-wise. They preserve the shape of their inputs.

```
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])

# Element-wise addition
print(a + b)
# [[ 6  8]
#  [10 12]]

# Element-wise multiplication
print(a * b)
# [[ 5 12]
#  [21 32]]

# Activation function (applied element-wise)
print(np.tanh(a))
# [[0.7616 0.9640]
#  [0.9951 0.9993]]
```

Element-wise operations can combine tensors of different shapes through broadcasting, which we will cover next. They are computationally cheap: each element is independent, so they parallelize perfectly across GPU cores. In a transformer, element-wise operations handle additions of residual connections, bias terms, and activation functions like GELU or ReLU.

Matrix Multiplication and Broadcasting

Matrix multiplication is the single most important operation in deep learning. Every linear layer, every attention score computation, and every embedding lookup can be expressed as matrix multiplication.

The rule is simple but strict. To multiply matrix A of shape (m, n) by matrix B of shape (n, p), the inner dimensions must match. The result has shape (m, p). Each element (i, j) of the output is the dot product of row i of A with column j of B:

C[i, j] = sum_k A[i, k] * B[k, j]

Here is a concrete example:

```
A = np.array([[1, 2, 3],
              [4, 5, 6]])  # shape (2, 3)

B = np.array([[7, 8],
              [9, 10],
              [11, 12]])   # shape (3, 2)

C = A @ B  # @ is the matrix multiplication operator
print(C)
# [[ 58  64]
#  [139 154]]
print(C.shape)  # (2, 2)

# C[0, 0] = 1*7 + 2*9 + 3*11 = 58
# C[0, 1] = 1*8 + 2*10 + 3*12 = 64
# C[1, 0] = 4*7 + 5*9 + 6*11 = 139
# C[1, 1] = 4*8 + 5*10 + 6*12 = 154
```

In NumPy and PyTorch, the @ operator and np.matmul or torch.matmul perform matrix multiplication. The function np.dot has different behavior for higher-dimensional tensors and should be avoided for clarity.

Broadcasting extends element-wise operations to tensors of different shapes. When two tensors have different shapes, NumPy compares dimensions from the last axis backward. Two dimensions are compatible if they are equal or one of them is 1. If incompatible, the operation fails. If compatible, the dimension of size 1 is virtually stretched to match the other.

```
# Add a vector to each row of a matrix
matrix = np.array([[1, 2, 3],
                   [4, 5, 6]])  # (2, 3)
vector = np.array([10, 20, 30])  # (3,)
print(matrix + vector)
# [[11 22 33]
#  [14 25 36]]

# Add a column vector to each column
col = np.array([[100],
                [200]])  # (2, 1)
print(matrix + col)
# [[101 102 103]
#  [204 205 206]]
```

Broadcasting is used throughout transformers. When you add positional encodings to embeddings, you broadcast a (seq_len, d_model) matrix across the batch dimension. When you apply a causal mask, you broadcast a (seq_len,

seq_len) mask across batch and heads. Understanding broadcasting is essential for reading tensor shapes in transformer code without being overwhelmed.

Batch Dimensions and the Last-Dimension Convention

Transformers process data in batches for efficiency. Processing 64 examples simultaneously on a GPU is much faster than processing them one at a time because matrix operations are highly optimized for large inputs and the GPU keeps more cores busy.

The convention across frameworks is consistent: the first dimension is the batch dimension. If you have a single example of shape (seq_len, d_model), batching it gives you (batch_size, seq_len, d_model). All operations are designed to work on batches automatically through broadcasting or batched matrix multiplication.

The second convention is the last-dimension convention: the rightmost dimension is the feature dimension. A token embedding has shape (d_model,). A batch of embeddings has shape (batch_size, seq_len, d_model). When you apply a linear transformation, you multiply the last dimension by a weight matrix.

```
# Single sequence of 10 tokens, each with embedding dimension 512
x = np.random.randn(10, 512)

# Linear layer: weight matrix maps 512-dim inputs to 256-dim outputs
W = np.random.randn(512, 256)

# Output: each token's 512-dim vector is transformed to 256-dim
out = x @ W  # shape (10, 256)
```

With batching:

```
# Batch of 4 sequences, each 10 tokens, embedding dim 512
x_batch = np.random.randn(4, 10, 512)

# Same weight matrix (512, 256)
# Batched matrix multiply: applies W to the last dimension of each example
out_batch = np.matmul(x_batch, W)  # shape (4, 10, 256)
```

Notice that the weight matrix W does not have a batch dimension. It is shared across all examples and all positions. The matmul operation applies the same W to every token in every sequence by multiplying along the last dimension.

This convention matters because it determines how you reshape tensors for attention. Multi-head attention splits the feature dimension into multiple heads and rearranges axes so that each head operates independently. The typical reshaping looks like this:

```
# Input: (batch, seq_len, d_model)
x = np.random.randn(4, 16, 512)
num_heads = 8
head_dim = 512 // 8  # 64

# Reshape: (batch, seq_len, num_heads, head_dim)
x = x.reshape(4, 16, num_heads, head_dim)

# Transpose: (batch, num_heads, seq_len, head_dim)
x = x.transpose(0, 2, 1, 3)
print(x.shape)  # (4, 8, 16, 64)
```

The transpose moves the heads axis next to the batch axis so that all heads for all sequences can be processed in a single batched matrix multiply. This pattern appears constantly in transformer implementations. Memorizing it saves hours of debugging.

A Running Example of Shape Tracking

Let us work through a concrete sequence of operations that mirror what happens inside a transformer encoder block. We will track shapes at every step. This is the skill that separates people who can read transformer code from those who cannot.

Start with a batch of input tokens. Suppose we have 8 examples in the batch, each is a sequence of 32 tokens, and our embedding dimension is 256:

```
x = np.random.randn(8, 32, 256)  # (batch, seq_len, d_model)
```

First, we compute query, key, and value matrices for self-attention. Each is obtained by a linear transformation: multiplying by a weight matrix of shape (d_model, d_model).

```
W_q = np.random.randn(256, 256)
W_k = np.random.randn(256, 256)
W_v = np.random.randn(256, 256)

Q = x @ W_q  # (8, 32, 256)
K = x @ W_k  # (8, 32, 256)
V = x @ W_v  # (8, 32, 256)
```

For multi-head attention with 8 heads, we reshape Q, K, V to separate the heads:

```
num_heads = 8
head_dim = 256 // 8  # 32

Q = Q.reshape(8, 32, num_heads, head_dim).transpose(0, 2, 1, 3)  # (8, 8, 32,
↪  32)
K = K.reshape(8, 32, num_heads, head_dim).transpose(0, 2, 1, 3)  # (8, 8, 32,
↪  32)
V = V.reshape(8, 32, num_heads, head_dim).transpose(0, 2, 1, 3)  # (8, 8, 32,
↪  32)
```

Now each element Q[b, h, i, :] is the query vector for batch b, head h, position i.

Attention scores are computed by multiplying Q with the transpose of K along the feature dimension. We need K transposed so that positions become columns:

```
# K.transpose(..., -2, -1) swaps seq_len and head_dim dimensions
# Shape: (8, 8, 32, 32)
scores = np.matmul(Q, K.transpose(0, 1, 3, 2))  # (8, 8, 32, 32)
```

Each scores[b, h, i, j] is the dot product of query at position i with key at position j, for head h and batch b. This measures how relevant position j is to position i.

We scale the scores and apply softmax along the last dimension (the key positions):

```
scores = scores / np.sqrt(head_dim)  # still (8, 8, 32, 32)
attn_weights = np.exp(scores) / np.exp(scores).sum(axis=-1, keepdims=True)  #
↪  (8, 8, 32, 32)
```

Now attn_weights[b, h, i, :] is a probability distribution over key positions for query at position i. Each row sums to 1.

We weight the value vectors by these attention weights:

```
context = np.matmul(attn_weights, V)  # (8, 8, 32, 32)
```

Each context[b, h, i, :] is a weighted sum of all value vectors, with weights determined by attention. This is the self-attention output: each position now carries information from all other positions.

Finally, we reshape back to combine the heads:

```
context = context.transpose(0, 2, 1, 3).reshape(8, 32, 256)  # (8, 32, 256)
```

And project the result through another linear layer:

```
W_o = np.random.randn(256, 256)
output = context @ W_o  # (8, 32, 256)
```

The output has the same shape as the input. Every token still has a 256-dimensional representation, but now each representation incorporates information from every other token in the sequence. This entire sequence of operations is what happens inside one multi-head attention layer. Understanding the shape transformations is the first step to understanding the computation.

The computational cost is worth noting. The attention score matrix has shape (batch, heads, seq_len, seq_len), so memory scales as O(seq_len^2). For 8 heads, batch size 8, and sequence length 2048, this is 8 * 8 * 2048 * 2048 * 4 bytes ≈ 107 megabytes for a single attention score tensor, stored in FP32. This quadratic memory is the primary bottleneck for long sequences and motivates many of the optimizations we will cover later, including FlashAttention and sparse attention variants.

Chapter 3: Neural Networks From First Principles

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Linear Layers and Affine Transformations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Activation Functions: Why Non-Linearity Is Necessary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Loss Functions: Cross-Entropy and What It Measures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Backpropagation: The Chain Rule in Practice

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Optimizers: Gradient Descent and Its Variants

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 4: From Text to Numbers: Tokenization and Embeddings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Character-Level versus Token-Level Representations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Byte-Pair Encoding and Subword Tokenization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Building a Vocabulary and Handling Unknown Tokens

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

The Embedding Layer as a Lookup Table

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Embeddings as Learned Vector Representations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 5: The Attention Mechanism

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

What Is Attention, Intuitively?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Query, Key, and Value: The Retrieval Analogy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Dot-Product Similarity and the Scaling Problem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Softmax as a Probability Distribution Over Positions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Computing Attention: A Step-by-Step Numerical Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 6: Multi-Head Attention

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Why One Head Is Not Enough

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Projecting Into Subspaces

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Concatenating and Mixing Head Outputs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

The Tensor Algebra of Multi-Head Attention

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

What Different Heads Learn

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 7: Positional Information

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

The Permutation Invariance Problem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Sinusoidal Positional Encodings: Derivation and Properties

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Learned Positional Embeddings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Relative Position and Distance-Aware Attention

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Masking: Why and When to Zero Out Positions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 8: Feed-Forward Networks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

The Position-Wise Feed-Forward Network

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Why We Need a Separate Transformation After Attention

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Activation Functions: ReLU, GELU, and SwiGLU

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Width, Depth, and Computational Trade-Offs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

The Complete Transformer Block

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 9: Implementing a Transformer in NumPy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Designing the Data Structures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

The Embedding and Positional Encoding Layer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Implementing Scaled Dot-Product Attention From Scratch

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Building Multi-Head Attention

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

The Complete Encoder Stack

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Testing and Verification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 10: Building a Decoder and Encoder-Decoder Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Causal Masking and Autoregressive Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

The Decoder Self-Attention Layer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Cross-Attention: Attending to the Encoder

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Output Projection and Logits

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Generating Text One Token at a Time

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 11: Automatic Differentiation and Backpropagation Through a Transformer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Gradients of Matrix Multiplication

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Gradients Through Softmax and Attention

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Gradients of the Feed-Forward Network

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

The Residual Connection and Gradient Flow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Verification Against Numerical Gradients

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 12: Training a Transformer From Scratch With PyTorch

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Why PyTorch: Autograd and Dynamic Graphs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Preparing a Tiny Training Dataset

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

The Complete Training Loop

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Monitoring Loss and Generating Samples

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Saving, Loading, and Resuming Checkpoints

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 13: GPT-Style Decoder-Only Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Why Decoder-Only Works for Language Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Architectural Choices in the GPT Family

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Layer Normalization Placement and RMSNorm

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

RoPE: Rotary Positional Embeddings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

SwiGLU and the Modern Feed-Forward Design

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 14: BERT and Encoder-Only Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Bidirectional Attention and Pretraining Objectives

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Masked Language Modeling in Detail

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Next Sentence Prediction

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Fine-Tuning for Downstream Tasks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

The Encoder-Only Limitations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 15: Encoder-Decoder and Instruction-Tuned Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

The T5 Architecture and Text-to-Text Pretraining

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Prefix-LM and Hybrid Designs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Instruction Tuning and Alignment

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Parameter-Efficient Fine-Tuning: LoRA and Friends

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Architectural Trade-Offs in Practice

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 16: Scaling Techniques and Mixture of Experts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

The Scaling Laws: Data, Parameters, and Compute

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Sparse Expert Routing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Mixture-of-Experts Training Dynamics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Load Balancing and Expert Capacity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

MoE in Production Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 17: Long-Context Techniques

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

The $O(n^2)$ Attention Bottleneck

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Grouped-Query and Multi-Query Attention

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Sliding Window Attention

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Memory Compression and Token Pruning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Extending Context Through Data and Training

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 18: Initialization, Optimization, and Scheduling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Initialization Strategies and Variance Control

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

AdamW and Why It Works for Transformers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Learning Rate Warmup and Cosine Decay

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Gradient Clipping and Stability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Hyperparameter Recipes That Work

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 19: Distributed Training: Parallelism Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Data Parallelism and Gradient Synchronization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Model Parallelism: Tensor and Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

ZeRO and Optimizer State Sharding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Communication Overhead and Topology

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Real-World Training Runs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 20: Memory and Precision

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

The Memory Budget of a Transformer Training Run

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Mixed Precision: FP16, BF16, and Loss Scaling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Gradient Accumulation for Effective Batching

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Activation Checkpointing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Memory Profiling and Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 21: Inference Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Pre-Fill Versus Decode: Two Different Regimes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

KV Caching: Why and How

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Continuous Batching and Scheduling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Speculative Decoding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Speculative Decoding Correctness and Sampling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 22: Quantization and Efficient Formats

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Why Quantization Works for Neural Networks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Post-Training Quantization: INT8 and Below

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Quantization-Aware Training

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Grouped Quantization and Outlier Handling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

GGUF, GGML, and the Open-Source Ecosystem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 23: FlashAttention and GPU Hardware

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

GPU Memory Hierarchy and Tensor Cores

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

The Attention Kernel: Why It Is Slow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

FlashAttention: IO-Aware Recomputation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

FlashAttention-2 and Beyond

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Measuring and Profiling GPU Utilization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 24: Serving Architectures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Throughput Versus Latency Trade-Offs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Model Serving Frameworks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Multi-GPU Inference and Expert Parallelism

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Caching, Load Balancing, and Autoscaling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Cost Modeling and Efficiency

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 25: What Transformers Can and Cannot Do

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

What Attention Can Represent

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Transformers as Approximate Algorithm Implementers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Inductive Biases: What the Architecture Assumes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Positional Encoding Limits and Extrapolation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Failure Modes and Systematic Errors

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Chapter 26: The History and Future of Attention

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

The Pre-Transformer Landscape: 2014-2017

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

"Attention Is All You Need": Context and Impact

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

The Architecture Arms Race: 2018-Present

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Post-Transformer Alternatives

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Open Questions and Where to Look Next

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

The Transformer in One Page

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

What You Can Do Now

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

The Next Decade of Sequence Modeling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/howtransformersactuallywork.