NULLVECTOR
.monster

PHASE 04 — TRANSFORMER

DAYS 64–84 · ATTENTION MECHANISM · BUILD AN LLM · DEPLOY IT

This is the phase that separates people who use AI from people who build it.

You will read 'Attention Is All You Need' — the 2017 paper that created the transformer architecture and made GPT, Claude, and every other modern AI possible. Then you will implement multi-head self-attention from scratch. Then you will build a small GPT that trains on real text and generates coherent output. By Day 84, you can walk any interviewer through your attention implementation line by line. No other candidate in the room will be able to say the same.

This is the hardest phase. It is also the most important. The math is real and it will take time to sink in. Do not skip the reading. Do not skip the derivation. The people who rush through the theory and jump to the code end up with code they cannot explain — and in an interview, code you cannot explain is code you do not understand.

// PHASE 4 AT A GLANCE

Duration	**21 days · 3.5–5 hours per day**
Milestone	**Day 84 — Trained language model deployed at public URL**
Primary	**Stanford CS224N — youtube.com/playlist?list=PLoROMvodv4rMFqRtEuo6SGjY4XbRIVx76**
Key paper	**Attention Is All You Need — arxiv.org/abs/1706.03762**

Reference	**Andrej Karpathy nanoGPT — github.com/karpathy/nanoGPT**
Video	**Let's build GPT — youtube.com/watch?v=kCc8FmEb1nY**
Library	**HuggingFace Transformers — huggingface.co/docs/transformers**
Your library	**Transformers for NLP 3E (Packt — your Humble Bundle)**

WEEK 1 · DAYS 64–70 · ATTENTION FROM SCRATCH

DAY 064 · READ THE PAPER

The document that changed the entire field.

■ INDUSTRY (15 MIN)

Watch: Yannic Kilcher explains 'Attention Is All You Need' (first 15 min)

This gives you context before you read the paper itself.

→ youtube.com/watch?v=iDulhoQ2pro

■ STUDY (2 HRS) — Attention Is All You Need

Read the paper. All of it.

First read: just read. Note what confuses you. Do not stop to look things up.

Second read: slow down at the equations. Understand each symbol.

Focus on: Section 3 (Model Architecture), specifically 3.2 (Attention).

The scaled dot-product attention formula: Attention(Q,K,V) = softmax(QK^T / √d_k)V

Q = queries, K = keys, V = values. Write down what each means in plain English.

→ arxiv.org/abs/1706.03762

■ BUILD (60 MIN) — Scaled dot-product attention

Implement the attention formula in pure NumPy:

def attention(Q, K, V): return softmax(Q @ K.T / sqrt(d_k)) @ V

Create random Q, K, V matrices. Compute attention. Print the result.

Verify the output shape makes sense: (seq_len, d_v)

✓ DONE WHEN: Attention paper read twice. Scaled dot-product attention implemented in NumPy.

DAY 065 · STANFORD CS224N — NLP FOUNDATIONS

The academic foundation under everything.

■ INDUSTRY (15 MIN)

Watch: Andrej Karpathy 'Intro to Large Language Models' (first 15 min)

→ youtube.com/watch?v=zjkBMFhNj_g

■ STUDY (90 MIN) — CS224N Lectures 1–3

Watch CS224N Lectures 1, 2, and 3.

Lecture 1: introduction to NLP and word vectors.

Lecture 2: word2vec — how words become numbers.

Lecture 3: neural networks for NLP.

These are Stanford's actual graduate NLP lectures, free on YouTube.

→ youtube.com/playlist?list=PLoROMvodv4rMFqRtEuo6SGjY4XbRIVx76

■ BUILD (90 MIN) — Word embeddings

Implement word2vec skip-gram from scratch using NumPy:

Build a vocabulary from a small text corpus

Create one-hot encodings for each word

Train a 2-layer network to predict context words

Extract the weight matrix as word embeddings

Find the 5 nearest neighbors of any word using cosine similarity

✓ DONE WHEN: CS224N lectures 1–3 watched. Word2vec implemented. Nearest neighbors working.

DAY 066 · MULTI-HEAD ATTENTION

The core mechanism of every modern AI.

■ INDUSTRY (15 MIN)

Search: 'transformer architecture explained visually' — any animated explainer.

■ STUDY (90 MIN) — CS224N Lectures 4–5

Watch CS224N Lectures 4 and 5: dependency parsing and language models.

Then re-read Attention Is All You Need Section 3.2.2: Multi-Head Attention.

Multi-head attention runs attention in parallel across h heads,

each with its own learned Q, K, V projection matrices.

Different heads learn to attend to different types of relationships.

→ youtube.com/playlist?list=PLoROMvodv4rMFqRtEuo6SGjY4XbRIVx76

■ BUILD (90 MIN) — Multi-head attention in PyTorch

Implement MultiHeadAttention as a PyTorch nn.Module:

self.W_q = nn.Linear(d_model, d_model)

self.W_k = nn.Linear(d_model, d_model)

self.W_v = nn.Linear(d_model, d_model)

self.W_o = nn.Linear(d_model, d_model)

Split into h heads. Compute attention for each. Concatenate. Project.

Test: input shape (batch, seq_len, d_model) → output same shape.

✓ DONE WHEN: MultiHeadAttention module implemented in PyTorch. Input/output shapes verified.

DAY 067 · POSITIONAL ENCODING + TRANSFORMER BLOCK

The full architecture emerges.

■ INDUSTRY (15 MIN)

Search: 'why does ChatGPT sometimes get things wrong hallucination'

Understanding limitations is as important as understanding capabilities.

■ STUDY (90 MIN) — Full transformer architecture

CS224N Lecture 8: Transformers and pre-training.

Re-read Attention paper Sections 3.3–3.5: positional encoding, encoder, decoder.

Positional encoding injects sequence order using sine and cosine functions.

Without it the model cannot distinguish 'the dog bit the man' from 'the man bit the dog'.

→ youtube.com/playlist?list=PLoROMvodv4rMFqRtEuo6SGjY4XbRIVx76

■ BUILD (90 MIN) — Full transformer encoder block

Build a complete transformer encoder block:

MultiHeadAttention (your Day 66 module)

Add & Norm (residual connection + layer normalization)

FeedForward (two linear layers with ReLU)

Add & Norm again

Positional encoding (sinusoidal)

Test with random input. Verify output shape.

✓ DONE WHEN: Full transformer encoder block implemented. Positional encoding added. Shape verified.

DAY 068 · TOKENIZATION

How text becomes numbers.

■ INDUSTRY (15 MIN)

Go to the OpenAI tokenizer. Type sentences. See how they become tokens.

This tool makes tokenization concrete and visual.

→ platform.openai.com/tokenizer

■ STUDY (90 MIN) — Tokenization deep dive

Three tokenization approaches: character-level, word-level, subword (BPE).

BPE (Byte Pair Encoding): start with characters, merge frequent pairs repeatedly.

GPT uses BPE. BERT uses WordPiece. Both are subword methods.

Read: HuggingFace tokenizer documentation — huggingface.co/docs/tokenizers

■ BUILD (90 MIN) — Simple BPE tokenizer

Implement a simplified BPE tokenizer from scratch:

Start with character vocabulary

Count all adjacent character pairs in the corpus

Merge the most frequent pair. Add merged token to vocabulary.

Repeat 100 times

Encode a sentence using your final vocabulary

Bonus: compare your tokenization to GPT-2's tokenizer output.

✓ DONE WHEN: BPE tokenizer implemented. Encodes text using learned vocabulary. Compared to GPT-2 output.

DAY 069 · KARPATHY NANOGPT — READ IT

The clearest GPT implementation in existence.

■ INDUSTRY (15 MIN)

Watch: 'Let's build GPT from scratch' — Andrej Karpathy (first 15 min)

The full video is 2 hours — watch at least 45 min today.

→ youtube.com/watch?v=kCc8FmEb1nY

■ STUDY (90 MIN) — nanoGPT codebase

Clone the nanoGPT repo. Read model.py completely.

Every class, every method, every line. Add a comment to each one.

Key classes: CausalSelfAttention, Block, GPT.

Notice: it uses your Day 66 multi-head attention with a causal mask.

The causal mask prevents tokens from attending to future tokens.

→ github.com/karpathy/nanoGPT

■ BUILD (90 MIN) — Train nanoGPT on tiny dataset

Run nanoGPT on the tiny Shakespeare dataset it ships with:

```
python data/shakespeare_char/prepare.py
```

```
python train.py config/train_shakespeare_char.py
```

Watch the loss decrease. After training: generate some Shakespeare.

```
python sample.py --out_dir=out-shakespeare-char
```

✓ DONE WHEN: nanoGPT read line by line with comments. Trained on Shakespeare. Generates plausible text.

DAY 070 · WEEK 1 CAPSTONE — ATTENTION VISUALIZATION

See what the model is thinking.

■ INDUSTRY (15 MIN)

Search: 'BERT attention visualization what does attention look at'

Attention visualization is one of the few ways to interpret what transformers learn.

■ STUDY (45 MIN)

Read: Jay Alammar's 'The Illustrated Transformer' — jalammar.github.io/illustrated-transformer

This is the most-cited explanation of transformers written for practitioners.

It is long. Read all of it. Print it if you learn better from paper.

■ BUILD (2.25 HRS) — Attention visualization

Build an attention visualization tool:

Load a pre-trained model with HuggingFace (distilbert-base-uncased)

Run a sentence through it and extract attention weights

Use matplotlib to draw a heatmap: rows=query tokens, columns=key tokens

Brighter = higher attention weight

Try multiple sentences. Notice patterns: what does each head attend to?

Post your most interesting visualization to your GitHub README.

✓ DONE WHEN: Attention visualization working. Multiple sentences analyzed. Best visualization in README.

WEEK 2 · DAYS 71–77 · BUILD YOUR LANGUAGE MODEL

DAY 071 · BUILD YOUR OWN GPT

From scratch. No framework doing the hard parts.

■ INDUSTRY (15 MIN)

Watch: 'Let's build GPT from scratch' — Karpathy (watch 45–90 min section)

→ youtube.com/watch?v=kCc8FmEb1nY

■ STUDY (60 MIN) — GPT architecture review

GPT is a transformer decoder — it only has the decoder stack, no encoder.

Key difference from vanilla transformer: causal self-attention (can't see future).

GPT-2 architecture: token embedding + positional embedding → N decoder blocks → linear.

Review your Days 66–67 modules. You have all the pieces.

■ BUILD (2 HRS) — Start your own GPT

Build from scratch — do NOT copy nanoGPT. Use your own modules:

class GPTConfig: define n_layer, n_head, n_embd, vocab_size, block_size

class CausalSelfAttention(nn.Module): your Day 66 attention + causal mask

class Block(nn.Module): attention + feedforward + layer norms + residuals

class GPT(nn.Module): embedding + positional + N blocks + linear head

Get the model to instantiate without errors. Print parameter count.

✓ DONE WHEN: Custom GPT class instantiates. Parameter count printed. All modules connect correctly.

DAY 072 · TRAIN ON CUSTOM DATA

Your model, your text, your weights.

■ INDUSTRY (15 MIN)

Search: 'what datasets were used to train GPT-3 and GPT-4'

Data quality and quantity are as important as architecture.

■ STUDY (45 MIN) — Training loop for language models

Language model training: predict the next token given all previous tokens.

Loss: cross-entropy between predicted distribution and actual next token.

Perplexity: e^loss. Lower is better. Random guessing = vocab size perplexity.

Good training: perplexity should fall significantly from epoch 1 to epoch N.

■ BUILD (2.25 HRS) — Full training run

Choose your training corpus — something you care about:

A book in the public domain (Project Gutenberg: gutenberg.org)

Wikipedia article dump on a topic you love

Song lyrics, scripts, code — anything with consistent style

Tokenize it. Build a DataLoader. Write the training loop.

Train for as many epochs as your hardware allows.

Generate text every 500 steps to watch the model improve.

✓ DONE WHEN: GPT trained on custom corpus. Loss decreasing. Text generation improving over training.

DAY 073 · TRAINING IMPROVEMENTS

Make your model better.

■ INDUSTRY (15 MIN)

Search: 'learning rate warmup and cosine annealing explained'

These techniques make the difference between training that diverges and training that converges.

■ STUDY (90 MIN) — Advanced training techniques

Gradient clipping: torch.nn.utils.clip_grad_norm_ prevents exploding gradients.

Learning rate warmup: start low, increase linearly, then decay.

Weight decay: L2 regularization for transformer weights.

Mixed precision training: torch.cuda.amp for faster training on GPU.

■ BUILD (90 MIN) — Add all four to your training loop

Add to your Day 72 training loop:

Gradient clipping: clip_grad_norm_(model.parameters(), 1.0)

Cosine LR schedule: torch.optim.lr_scheduler.CosineAnnealingLR

Weight decay in AdamW: optimizer = torch.optim.AdamW(params, weight_decay=0.1)

Mixed precision: torch.cuda.amp.autocast() context manager

Measure: does loss improve faster? Is training more stable?

✓ DONE WHEN: All four training improvements added. Training more stable. Loss improves faster.

DAY 074 · HUGGINGFACE ECOSYSTEM

The tools professional LLM engineers use daily.

■ INDUSTRY (15 MIN)

Search: 'HuggingFace how it became the GitHub of AI'

HuggingFace is where the AI open source world lives.

■ STUDY (90 MIN) — HuggingFace Transformers

Work through the HuggingFace Transformers quick tour.

Key concepts: AutoTokenizer, AutoModel, pipeline(), Trainer API.

Pipeline API: one line to run any task (text generation, classification, etc.)

The HuggingFace Hub: over 400,000 pre-trained models. Free to use.

→ huggingface.co/docs/transformers/quicktour

■ BUILD (90 MIN) — Five HuggingFace pipelines

Build and run 5 different HuggingFace pipelines:

1. Text generation with GPT-2

2. Sentiment analysis

3. Named entity recognition

4. Question answering

5. Text summarization

For each: run 3 real examples. Document what works and what doesn't.

✓ DONE WHEN: Five HuggingFace pipelines running. Real examples tested. Strengths and limitations noted.

DAY 075 · EVALUATION + BENCHMARKS

How do you know if your model is any good?

■ INDUSTRY (15 MIN)

Search: 'MMLU benchmark LLM evaluation explained' — how language models are ranked.

■ STUDY (90 MIN) — LLM evaluation

Perplexity: measures how well the model predicts text. Lower = better.

BLEU score: used for translation. Measures n-gram overlap with reference.

ROUGE: used for summarization. Recall-oriented n-gram overlap.

Human evaluation: for generative quality, humans are still the gold standard.

Read: Hugging Face Evaluate library — huggingface.co/docs/evaluate

■ BUILD (90 MIN) — Evaluate your model

Measure your trained GPT on three metrics:

1. Perplexity on a held-out test set

2. Average token prediction accuracy on the test set

3. Human evaluation: generate 10 samples, rate each 1–5 for coherence

Compare your perplexity to GPT-2 small on the same text domain.

Document all results in your log.

✓ DONE WHEN: Model evaluated on perplexity, accuracy, and human rating. Results documented.

DAY 076 · DEPLOY YOUR LANGUAGE MODEL

Your first deployed LLM.

■ INDUSTRY (15 MIN)

Search: 'how to deploy a language model to production 2025'

The gap between a trained model and a deployed product is where many fail.

■ STUDY (45 MIN) — Model deployment options

HuggingFace Spaces: easiest — push to a Space, Gradio handles the UI.

FastAPI: build a REST API, containerize with Docker, deploy anywhere.

Replicate: one-line deployment of PyTorch models — replicate.com

Choose the option that gets you to a public URL fastest.

■ BUILD (2.25 HRS) — Deploy your GPT

Deploy your trained language model to HuggingFace Spaces.

Interface: text input (seed text) → your model generates a continuation.

Make it clear: what was the model trained on? What is it good at?

Add examples. Add a note on its limitations.

This is Portfolio Project #6 — your first live language model.

✓ DONE WHEN: Custom-trained language model deployed at public URL. Portfolio Project #6 live.

DAY 077 · WEEK 2 CAPSTONE + TECHNICAL WRITEUP

Document your work at the level of a professional.

■ INDUSTRY (15 MIN)

Search: 'how to write an AI research blog post' — examples from ML practitioners.

■ STUDY (30 MIN)

Read 2 technical blog posts from the Andrej Karpathy blog — karpathy.github.io

Note structure: problem, approach, results, insights, what's next.

■ BUILD (2.75 HRS) — Technical writeup

Write a 600-word technical writeup about your language model. Structure:

Architecture: describe your model in precise terms (layers, parameters, dimensions)

Dataset: what you trained on, how much, why

Training: key hyperparameters, number of epochs, hardware used

Results: perplexity, accuracy, sample outputs (good and bad)

Insights: what surprised you? What would you do differently?

Commit to GitHub as LLM_WRITEUP.md

This writeup is what separates candidates who understand from candidates who copied.

✓ DONE WHEN: 600-word technical writeup committed. Architecture, training, and results all documented.

WEEK 3 · DAYS 78–84 · ADVANCED LLM + PORTFOLIO

DAY 078 · BERT AND MASKED LANGUAGE MODELING

The other side of the transformer.

■ INDUSTRY (15 MIN)

Search: 'what is BERT and how is it different from GPT'

→ youtube.com — search 'Jay Alammar BERT illustrated'

■ STUDY (90 MIN) — BERT architecture

Read: 'The Illustrated BERT' by Jay Alammar.

GPT: decoder-only, predicts next token (autoregressive).

BERT: encoder-only, predicts masked tokens (bidirectional).

GPT is better for generation. BERT is better for understanding tasks.

Both are transformers. The difference is training objective and attention masking.

→ jalammar.github.io/illustrated-bert/

■ BUILD (90 MIN) — Fine-tune BERT for classification

Fine-tune BERT for a classification task using HuggingFace:

from transformers import BertForSequenceClassification, Trainer

Choose a dataset: news categorization, intent detection, anything.

Fine-tune for 3 epochs. Evaluate on test set.

Compare: BERT fine-tuned vs your Day 52 LSTM sentiment model.

BERT should win significantly with less training.

✓ DONE WHEN: BERT fine-tuned for classification. Accuracy compared with LSTM. BERT wins.

DAY 079 · MODERN LLM LANDSCAPE

Understanding what exists and why.

■ INDUSTRY (15 MIN)

Watch: Andrej Karpathy 'Intro to Large Language Models' (full version — 1 hour)

You now have enough context to understand everything in this video.

→ youtube.com/watch?v=zjkBMFhNj_g

■ STUDY (90 MIN) — LLM landscape survey

GPT family: GPT-2, GPT-3, GPT-4 — OpenAI, decoder-only, scaled up.

LLaMA family: Meta's open weights models — LLaMA 2, LLaMA 3.

Mistral: efficient open models — Mistral 7B, Mixtral 8x7B.

Claude: Anthropic, Constitutional AI, harmlessness focus.

Gemini: Google DeepMind, multimodal from the ground up.

Open vs closed: what does open weights mean for builders?

Read: Hugging Face Open LLM Leaderboard — huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard

■ BUILD (60 MIN) — Run an open-source LLM locally

Install Ollama: ollama.ai — run open-source LLMs on your laptop.

ollama pull llama3.2

ollama run llama3.2

Chat with it. Compare outputs to GPT-4 or Claude on the same prompts.

Notice: quality difference, speed difference, context window behavior.

→ ollama.ai

✓ DONE WHEN: LLM landscape documented. LLaMA 3 running locally. Comparison with closed models noted.

DAY 080 · PROMPTING AND PROMPT ENGINEERING

Getting the most from any LLM.

■ INDUSTRY (15 MIN)

Search: 'prompt engineering techniques 2025 best practices'

■ STUDY (90 MIN) — Prompt engineering

Zero-shot: just ask. Works for simple tasks.

Few-shot: provide examples. Works for formatting and style.

Chain of thought: 'let's think step by step'. Works for reasoning.

System prompts: set the context and role for the entire conversation.

Read: Anthropic's prompt engineering guide — docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/overview

■ BUILD (90 MIN) — Prompt engineering experiment

Pick a task that requires reasoning (math problem, logic puzzle, coding challenge).

Test 5 different prompting strategies on the same task:

1. Zero-shot

2. Few-shot (3 examples)

3. Chain of thought

4. System prompt + zero-shot

5. System prompt + chain of thought

Rate accuracy and quality for each. Document which works best and why.

✓ DONE WHEN: 5 prompting strategies tested on same task. Results documented. Best approach identified.

DAY 081 · TOKENIZATION DEEP DIVE + CONTEXT WINDOWS

The practical limits of LLMs.

■ INDUSTRY (15 MIN)

Search: 'context window explained LLM why it matters'

Context window size is one of the most practically important LLM properties.

■ STUDY (90 MIN) — Context windows and token limits

Context window: how much text an LLM can 'see' at once.

GPT-4: up to 128k tokens. Claude 3.5: up to 200k tokens.

Why it matters: longer documents, longer conversations, more few-shot examples.

Cost: attention computation is $O(n^2)$ in sequence length — longer = more expensive.

Read: how Anthropic's long context works — anthropic.com/research

■ BUILD (90 MIN) — Context window experiments

Using the HuggingFace API or Anthropic API:

1. Find the exact point where your chosen model loses information (fill context)

2. Implement a sliding window approach for processing long documents

3. Implement chunking + summarization for documents exceeding context

These are real production engineering problems.

✓ DONE WHEN: Context window limits tested. Sliding window and chunking both implemented.

DAY 082 · MODEL IMPROVEMENTS

Make your mini-LLM better.

■ INDUSTRY (15 MIN)

Search: 'scaling laws for language models Chinchilla' — how model size and data interact.

■ STUDY (60 MIN) — Pick one improvement to implement

Choose one improvement from this list and study it deeply:

A. Rotary Position Embedding (RoPE) — better positional encoding

B. Grouped Query Attention (GQA) — more efficient attention

C. SwiGLU activation function — used in LLaMA

D. RMS Norm instead of Layer Norm — simpler and works as well

Read the original paper or blog post for whichever you choose.

■ BUILD (2 HRS) — Add your chosen improvement

Add the improvement you studied to your existing GPT architecture.

Retrain for a few epochs. Compare loss curves to your baseline.

Did the improvement help? By how much? Document the result.

This is a tiny version of what ML researchers do every day.

✓ DONE WHEN: Architecture improvement implemented and evaluated. Loss comparison documented.

DAY 083 · PORTFOLIO DOCUMENTATION

Your transformer work presented at professional level.

■ INDUSTRY (15 MIN)

Search: 'machine learning portfolio what employers actually look for 2025'

■ STUDY (30 MIN)

Look at the GitHub profiles of 3 ML researchers you admire.

What do their READMEs look like? How do they explain their projects?

■ BUILD (2.75 HRS) — Phase 4 portfolio update

Update your GitHub README with Phase 4 section:

Link to your live language model demo

Link to your attention visualization

Link to your BERT classification project

Key skills: attention mechanism, transformer architecture, LLM training,

tokenization, model evaluation, HuggingFace ecosystem

Record a 3-minute demo video: walk through your language model live.

Upload to YouTube (unlisted is fine). Add link to README.

✓ DONE WHEN: Phase 4 README section complete. Demo video recorded and linked. All projects documented.

DAY 084 · PHASE 4 COMPLETE

You are now someone who builds language models.

■ REVIEW (60 MIN)

Open your language model demo. Generate some text.

Think about what you know now: attention mechanism, transformer architecture,

tokenization, training loops, evaluation metrics, prompting, the LLM landscape.

Three weeks ago you were reading a paper you barely understood.

Today you have implemented what that paper describes.

■ PHASE 5 PREP (90 MIN) — Set up for Fine-Tuning and RAG

Phase 5 is about making existing LLMs useful for specific tasks.

Install the tools you'll need:

pip install peft bitsandbytes transformers accelerate

pip install llama-index chromadb

Read the PEFT (Parameter-Efficient Fine-Tuning) quickstart.

Read the LlamaIndex 5-minute quickstart.

Phase 5 starts tomorrow.

→ huggingface.co/docs/peft

→ docs.llamaindex.ai

✓ DONE WHEN: Phase 4 complete. Demo video live. Phase 5 libraries installed. Ready.

DAY 84 MILESTONE · TRAINED LANGUAGE MODEL — DEPLOYED AT PUBLIC URL

A Language Model You Built From Scratch

You implemented multi-head self-attention from the equations in a research paper. You built a GPT architecture from scratch. You trained it on real text. You deployed it. You can explain every line of code because you wrote every line of code. Phase 5 — OPERATOR — begins Day 85. You will learn to make existing LLMs useful for specific real-world tasks through fine-tuning, RAG, and production deployment.
