TAE Week 3 Coaching Guide

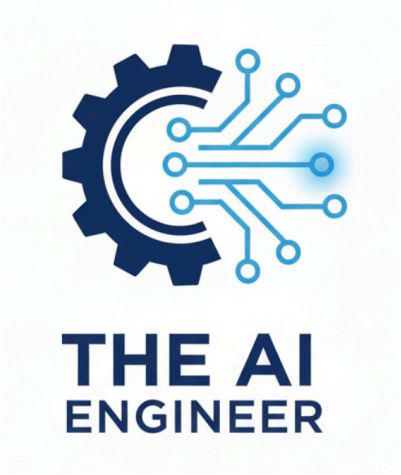
LLMs, Attention & Tiny Transformers Capstone

The AI Engineer Program

November 15, 2025

Abstract

Week 3 connects transformer theory with practical LLM engineering. You will deepen your understanding of attention and tiny decoder-only transformers using the dedicated handout, then embed these ideas into an engineering workflow using Part II of the AI/ML Engineering book and the LLM guide. The capstone centers on a from-scratch tiny transformer language model, built and trained in a Colab notebook, with clear logging, checkpoints, and sampling utilities.



Contents

1 Purpose & Scope (Week 3)

This guide coaches you through Week 3. It focuses on the **LLM book and deck**, the **Engineering** companion (Part II), and the **Attention & Tiny Transformers handout** that defines the capstone: a small decoder-only transformer language model implemented from scratch.

Week 3 Outcomes

By the end of the week you should be able to: (1) explain and implement scaled dot-product attention and (multi-head) self-attention, (2) assemble a tiny decoder-only transformer with positional encodings and causal masking, (3) train and sample from a miniature language model in Colab, and (4) wrap this work in a simple but honest engineering workflow (tracking, checkpoints, and reproducible runs).

2 Core Resources

Keep these open while you work; move between them rather than reading any single one linearly.

- **LLM Book (llm.html)** [1] — mindset, architecture, and training pipeline for building a GPT-style model from scratch.
- **LLM Slides (llm_slides.pdf)** [3] — high-level structure and "why" for each component; skim before and after working in the handout.
- **AI/ML Engineering (eng.html, Part II)** [2] — tracking, configuration, pipelines, and ML lifecycle practices you will apply to the capstone.
- **Engineering Slides (eng_slides.pdf, Part II)** [4] — condensed view of ML engineering in practice; use as a checklist when wiring your training script and logging.
- **Attention & Tiny Transformers Handout** [6] — detailed derivations and implementation roadmap for the tiny transformer LM that anchors the capstone.

- **Program Schedule (schedule.html)** [5] — Week 3 module overview, deadlines, and how this capstone fits into the broader TAE journey.

3 What You Will Learn

The bullets below summarize the core skills and concepts to prioritize during Week 3.

- **Attention mechanics** — content-based addressing, scaled dot-product attention, masking, and the transition from single-head to multi-head attention.
- **Transformer blocks** — self-attention, position-wise feedforward networks, residual connections, and layer normalization.
- **Tiny decoder-only LMs** — token and positional embeddings, stacked transformer blocks, and a projection head for next-token prediction.
- **Training and evaluation** — batching, loss shaping for cross-entropy, tracking training/-validation loss, and basic qualitative sampling.
- **LLM engineering habits** — configuration via code or CLI flags, seeding, logging, checkpoints, and simple run records inspired by Part II of the Engineering book.

4 How to Study

Full-Time Track (3–4 h/day)

If you have several hours per day available, use this structure as a default and adapt it to your context.

- **Morning (90 min)**: LLM book sections on the overall pipeline and transformer architecture [1]; skim the corresponding slides while keeping the big picture in mind.
- **Midday (60–90 min)**: Attention & Tiny Transformers handout [6]; work through scaled dot-product attention, self-attention, and transformer blocks, checking shapes carefully.
- **Afternoon (60 min)**: Capstone notebook work — implement or refine `scaled_dot_product_attention`, a transformer block, and the tiny LM forward pass.
- **End (15–30 min)**: Engineering hygiene (Part II of the Engineering book [2]) — add logging, configuration, and a short run record (what you changed, what happened).

Time-Constrained (60–90 min/day)

If your time is limited, use the following lightweight progression to keep moving the capstone forward.

- **Day 1**: Skim LLM overview (book + slides) for global structure; read the handout's motivation and goal and "How to Use This Capstone" [1, 6].
- **Day 2**: Study scaled dot-product attention and implement `scaled_dot_product_attention` in Colab with the numeric example from the handout [6].
- **Day 3**: Implement single-head self-attention and a minimal transformer block (attention + feedforward + residual + norm) [6].
- **Day 4**: Assemble the tiny decoder-only LM, wire up the loss and training loop, and run a short training session [6, 1].
- **Day 5**: Add simple logging and checkpoints inspired by Part II of the Engineering book; run sampling experiments at different temperatures and document the results [2, 6].

5 Focus First (Priorities)

When in doubt, return to these priorities before adding additional features or experiments.

- **Shapes and masking**: whenever you get lost, write down tensor shapes for Q, K, V, the attention scores, and outputs; verify causal masks by hand on tiny examples [6].
- **Single forward path**: get one clean forward pass from token indices to logits working before adding bells and whistles [1, 6].
- **Training loop basics**: stick to a simple, well-tested pattern (batch $\rightarrow$ forward $\rightarrow$ loss $\rightarrow$ backward $\rightarrow$ step); log training and validation loss periodically [1, 2].
- **Engineering hygiene**: keep configuration in one place, seed randomness, and save at least one checkpoint per working run; a small run log is more valuable than an extra experiment [2].
- **Capstone alignment**: periodically cross-check your notebook against the handout's "Colab Implementation Roadmap" and capstone description so you do not drift from the intended scope [6].

6 Key Concepts — Concise Recap

Attention and Self-Attention

Use these bullets as a quick recall checklist while implementing or debugging attention.

- Attention as content-based addressing: queries, keys, values, softmax weights, and weighted sums [6].
- Scaled dot-product attention with masking: score matrix, softmax over keys, and causal masks that prevent looking into the future [6].
- Self-attention: Q, K, V derived from the same sequence of token embeddings, applied in parallel across positions [6].

Transformer Blocks

These points summarize what each transformer block is responsible for in your tiny LM.

- Multi-head attention: split heads, compute attention per head, then concatenate and project back to the model dimension [6].
- Position-wise feedforward nets: simple MLP applied independently at each position, sandwiched between residual connections and layer normalization [6].
- Pre-LN vs. Post-LN: for this capstone, a pre-layer normalization structure improves stability at small scales [6].

Tiny Decoder-Only LM

Keep the following structure in mind when you reason about the end-to-end model.

- Token and positional embeddings combined into a sequence of hidden states of shape (B, T, d_{model}) [6, 1].
- A stack of transformer blocks followed by a projection head to vocabulary logits; reshape logits and targets for cross-entropy [6, 1].

- Autoregressive sampling: repeatedly feed back generated tokens, using temperature or top-k sampling to control diversity [6, 1].

Engineering and Tooling

Ground your capstone in these minimal but important engineering practices.

- Configuration objects or CLI flags for hyperparameters; keep them in version control alongside code [2].
- Lightweight tracking (CSV/JSON logs, simple run IDs) to compare experiments; avoid opaque, one-off notebook states [2].
- Checkpoints via `state_dict()` and clear naming conventions so you can resume or resample later [2, 1].

7 Review Questions

Answer in code or 1–3 sentences; use them as prompts for tiny experiments.

- Write down the shapes of Q, K, V, the attention score matrix, and the attention output for a batch of size B and sequence length T. Where does the scaling factor $1/\sqrt{d_k}$ appear?
- Explain why causal masking is required for autoregressive language models. How would you test that your mask is implemented correctly?
- In your tiny transformer, where do residual connections appear and why do we use layer normalization around them?
- Show how you reshape logits and targets before applying `nn.CrossEntropyLoss`. What happens if you forget to reshape?
- Describe one engineering decision you made inspired by Part II of the Engineering book (e.g., how you log metrics, how you structure configuration, or how you name checkpoints).
- After training, compare samples at two different temperatures. What changes in the generated text and why?

8 Practice Checklist

Use this checklist to confirm that you have covered the essential implementation and engineering pieces for the Week 3 capstone.

- Implement `scaled_dot_product_attention` and test it against the numeric example in the handout [6].
- Build a minimal transformer block with single-head self-attention and a position-wise feedforward network; verify shape invariants end-to-end [6].
- Assemble a tiny decoder-only LM (`TinyTransformerLM`) with embeddings, a small stack of blocks, and a projection head; run a short training loop on a tiny corpus [6, 1].
- Add configuration, logging, and checkpoints following the Engineering book (Part II) and verify that you can reproduce at least one run [2].
- Implement greedy and temperature-based sampling; save a small "gallery" of prompts and completions as part of your capstone artifact [6, 1].

9 Tiny Study Loop

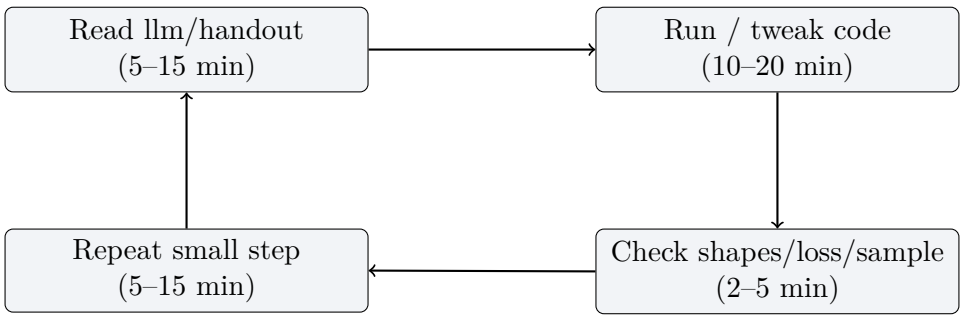


Figure 1: Short, repeatable loop for Week 3: read a small piece, run it, inspect shapes/loss/samples, repeat.

10 Daily Cadence (Example)

Block	Focus	Resource
45–60 min	Attention/transformer theory	Handout (`att`)
20–30 min	LLM pipeline context	`llm.html`, llm_slides
10–15 min	Engineering hygiene	`eng.html` (Part II)

11 Capstone Project (Week 3)

Build a **Colab-ready tiny transformer language model** that follows the Attention & Tiny Transformers handout and the LLM book patterns. Your model should implement scaled dot-product attention, self-attention, transformer blocks, and a decoder-only architecture that performs next-token prediction on a small text corpus. Host your notebook and any supporting code on **GitHub**; keep the model intentionally small but complete, with seeds, checkpoints, and simple sampling utilities.

- **Execution**: one notebook (plus optional small module) that runs top-to-bottom on Colab; deterministic seeds; clear configuration and device handling.
- **Scope**: tiny decoder-only transformer LM on a simple character- or token-level dataset, with at least one training run that produces non-trivial samples; single or few transformer blocks are sufficient.
- **Deliverables**: GitHub link to the notebook/code; a short run log (what configuration you used, how training behaved, and example samples); optional exported PDF of the handout-aligned results.
- **Schedule**: Reserve **3** × **90 min** focused sessions to (1) build the attention and block components, (2) assemble and train the tiny LM, and (3) refine engineering hygiene and sampling, aligning with the handout's implementation roadmap.

References

[1] Building a Large Language Model from Scratch. llm.html

[2] AI, ML & Software Engineering — Building Intelligent Systems in Practice. eng.html

[3] Building a Large Language Model from Scratch — Slide Deck. llm_slides.pdf

[4] AI, ML & Software Engineering — Slide Deck (Part II). eng_slides.pdf

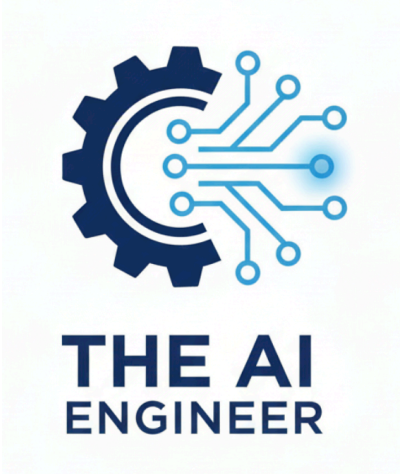
[5] Program Schedule. schedule.html

[6] Attention & Tiny Transformers Handout (From Scaled Dot-Product Attention to a Mini LLM). att_transformers.pdf

Contact

TAE Week 3 Coaching Guide

LLMs, Attention & Tiny Transformers Capstone



Get in touch:

https://linktr.ee/dyjh