Core Task Progress Notes

Team: _____

November 13, 2025

Sanity-Check Run

- **Command.** `OMP_NUM_THREADS=1 python3 pico-llm.py --tinystories_weight 0.0 --input_files 3seqs.txt --block_size 32 --max_steps_per_epoch 1 --kgram_k 2 --embed_size 64 --device_id cpu`
- **Environment tweaks.** Installed the missing `tiktoken` package and pinned OpenMP threads to avoid shared-memory warnings in the sandboxed environment.
- **Outcome.** Training loop completed three abbreviated epochs for the LSTM baseline on `3seqs.txt`, producing loss values ($10.82 \rightarrow 10.80 \rightarrow 10.77$) and sample generations at greedy, $p = 0.95$, and $p = 1.0$ settings. This confirms data loading, tokenization, model forward/backward, and text generation pathways function end-to-end on CPU-only hardware.

Key Observations

- The current nucleus sampling placeholder yields identical outputs across different p values, reinforcing the need to implement true top-p sampling before the presentation.
- LSTM outputs trained on the minimal dataset remain nonsensical, which is expected given the toy corpus and tiny training budget; future sanity checks on TinyStories or richer data should improve qualitative quality.
- Installing dependencies on the target machine (e.g., `tiktoken`) should be part of setup instructions to prevent runtime failures during the live demo.

Sample Interview Questions & Answers

1. **What did you verify in the sanity-check run, and why did you choose the custom dataset?**
   We confirmed that the default training loop, tokenizer, and generation routines run without crashing on CPU. The `3seqs.txt` data ships with the repo, so it avoids the network dependency of downloading TinyStories and shortens iteration time.
2. **Why did the generated samples look incoherent, and is that a concern?**
   The dataset contains synthetic numeric sequences and we limited training to one gradient step per epoch. With minimal data and budget, the LSTM cannot learn meaningful structure; the purpose of this run was functionality, not quality. Larger datasets and more steps will address coherence.

3. **You pinned `OMP_NUM_THREADS` to 1—will that hurt performance later?**
   For this quick CPU smoke test it eliminated shared-memory errors. On a full training environment we can remove or raise the cap once OpenMP shared-memory permissions are available, restoring multithreaded BLAS performance.

4. **What additional checks will you run before the presentation?**
   We plan to repeat the sanity check on TinyStories once Transformer and k-gram models are implemented, capture training curves for the required figures, and verify qualitative outputs under true top-p sampling to demonstrate improved diversity.

KGramMLPSeqModel Design Options

1. **Baseline One-Hot MLP.**
   Structure: Flatten the k one-hot context vectors into a $(k \times |V|)$ input and feed through stacked Linear $\rightarrow$ SiLU blocks ending in a Linear layer to logits.
   Pros: Direct translation of the docstring; simplest to reason about; no extra embeddings needed.
   Cons: Extremely high input dimensionality (tens of thousands) makes each forward/backward pass slow and memory hungry; scales poorly with larger vocab or k.
   Performance expectations: Works for tiny vocab toy data but may be impractical for full TinyStories unless chunking is aggressive.

2. **Token Embedding + MLP (Shared Projection).**
   Structure: Map each context token through a learned embedding matrix ($|V| \rightarrow d$), concatenate the k embeddings (size $k \times d$), then apply a compact MLP that projects back to vocabulary logits.
   Pros: Input dimensionality drops dramatically; parameters scale with d instead of $|V|$; embeddings can be reused or initialized from LSTM/Transformer values.
   Cons: Slightly more complex; must ensure embeddings are trained jointly; loses the "pure" one-hot formulation.
   Performance expectations: Much faster and fits GPU memory; still expressive enough for k-gram patterns when d is moderate.

3. **Embedding + Depthwise 1D Convolution Hybrid.**
   Structure: Embed tokens as above, stack along temporal dimension, then use 1D convolution(s) over the k context positions followed by a projection to logits.
   Pros: Weight sharing across positions reduces parameters; convolutional kernels capture local order efficiently; fits well with chunked inference.
   Cons: Deviates from strict MLP requirement; requires reshaping logic in the provided forward loop; debugging convolutional behavior may be harder.
   Performance expectations: Efficient on GPU and handles longer k gracefully; offers inductive bias similar to CNNs for n-gram features.

Action Plan: Implementing KGramMLPSeqModel

1. **Select architecture.** Choose between one-hot MLP, embedding-based MLP, or convolution hybrid based on desired trade-offs.

2. **Define network layers.** Instantiate embedding (if needed), hidden Linear layers with activation, dropout if desired, and final Linear layer to $|V|$ logits.

3. **Integrate into forward loop.** Replace the `self.net = None` stub and ensure tensors are moved to the correct device, handling chunked processing without altering the provided outer loops.

4. **Add safety checks.** Validate input shapes, raise informative errors if `self.net` is misconfigured, and confirm gradients flow through embeddings.

5. **Test in notebook.** Run the reduced configuration from the sanity check, compare loss trajectories, and profile runtime to confirm expected improvements.

Forward Pass Rationale

- **Responsibility.** The `forward()` method ingests a tensor of token IDs with shape (sequence length, batch size), walks through the sequence in micro-batches, extracts the preceding k tokens for each position, pads when history is short, one-hot encodes that context, and ships the flattened vector through the chosen sub-network to obtain logits.

- **Chunking.** Iterating in blocks of size `chunk_size` lowers peak memory usage, making the naive Python loop feasible even for long sequences.

- **Why it matters.** Without this logic, the k-gram model would never transform context into next-token distributions—no logits means no loss, no gradients, and no generation. This function therefore defines the core behavior we train and evaluate.

- **Postconditions.** Concatenating the outputs from each chunk restores the expected (seq_len, batch, $|V|$) tensor that downstream code (loss, sampling) depends on.

Action Plan: Implementing Nucleus Sampling

1. **Normalize logits.** Convert logits to probabilities via `softmax` to obtain a stable distribution over the vocabulary.

2. **Rank tokens.** Sort probabilities in descending order while keeping token indices to identify the highest-mass prefix.

3. **Cumulative cutoff.** Compute the running probability mass and retain the smallest slice whose sum exceeds the user-specified p threshold.

4. **Renormalize.** Scale the retained probabilities so they sum to one, ensuring a valid categorical distribution.

5. **Sample token.** Draw the next token from the truncated distribution (fall back to the full distribution if p never triggers).

6. **Surface API.** Expose the sampler through `generate_text` so notebook experiments can vary p without extra wiring.

Nucleus Sampling Observations

- **Setup.** Used the quick LSTM sanity-check run (one optimization step on the synthetic corpus) and generated 20 new tokens from the prompt "Once upon a" with different top-p thresholds.
- **Greedy (p = None).** *"Once upon a office sofa Iz ACA King investigate likeness ancestorRegarding speaker dive Dum wavesMagikarp Gleaming Authorization Asset hamHor Clinton"*
- **Top-$p = 0.8$.** *"Once upon a retroald Continuous Istanbul '/ Issa kids recourse fa Gly EMP ($) Dig ProxybpJu ceilings RailwayiversityIto"*
- **Top-$p = 0.95$.** *"Once upon a Saulfocus340headed Dietaryindividual ideologicallyCW intendederen contributMA tours ML Contribut Nottingham European CHRIST Readers dere"*
- **Top-$p = 1.0$.** *"Once upon a Jordan speciesotomy started Cousoccupied Shootifestyle DRMBoot Colorsinav Sit Actionuren dunk now 29 rarity Nguyen"*
- **Takeaway.** Higher p values admit a larger tail of low-probability tokens, yielding more diverse (but noisier) completions. Greedy decoding sticks to the modal choice and repeats motifs, while $p = 1.0$ allows maximum randomness.

Performance Comparison Strategy

- **Feasibility.** It is practical to compare all three architectures visually: log training loss and wall-clock time per step for each variant while running identical hyperparameters, then plot the metrics using `matplotlib` inside the standalone notebook. GPU execution is optional but accelerates experimentation.
- **Instrumentation plan.**
  - Wrap the training loop to record (loss, step, elapsed seconds) tuples for each KGramMLPSeqModel option via separate helper functions.
  - Store results in a shared list or `pandas` DataFrame, noting architecture label and configuration (e.g., embedding dimension).
  - After each run, generate line plots for loss vs. step and bar charts for tokens/sec to highlight efficiency differences.
- **Caveats.** Full TinyStories runs may be slow in Colab CPU mode; expect to subsample the dataset or limit training steps for a fair yet tractable comparison. Ensure random seeds are fixed for reproducibility.

Empirical Results (Synthetic Corpus)

- **Setup.** CPU only; k=2, embedding size 64, two inner MLP layers, Adam with learning rate 1 $times 10^{-3}$, 20 mini-batches per variant (batch size 32) over the `3seqs.txt` dataset.
- **Metrics.**

  - One-hot MLP: average loss $approx$10.63, throughput $approx$17.6 tokens/s, runtime $approx$1161 seconds.
  - Embedding MLP: average loss $approx$9.49, throughput $approx$42.3 tokens/s, runtime $approx$485 seconds.
  - Conv Hybrid: average loss $approx$10.23, throughput $approx$32.3 tokens/s, runtime $approx$634 seconds.

- **Decision.** The embedding-based MLP achieves the best trade-off (lowest loss and highest throughput), so we will proceed with this architecture for the full implementation and upcoming experiments.

TransformerModel Implementation Outline

1. **Surface Configuration.**
   - Expose `d_model`, `n_heads`, `n_blocks` (capped at ten), feed-forward width multiplier, dropout rates, and rotary-frequency toggles so we can sweep sizes without rewriting layers.
   - Share a `transformer_variant` flag to map onto the presets listed below (GPT2, GPT-oss, Llama3-lite, minGPT).
2. **Token and Positional Embeddings.**
   - Reuse the shared token embedding matrix for the output projection (weight tying).
   - Implement learned positional embeddings by default; include a rotary positional embedding helper so GPT-oss and Llama-style presets can opt in.
3. **RMSNorm and Block Skeleton.**
   - Finish the RMSNorm stub: compute the mean square per hidden vector, scale by `weight`, and broadcast the epsilon guard.
   - Each Transformer block follows Pre-Norm layout: `RMSNorm` → Multi-Head Attention + residual → `RMSNorm` → Feed-Forward + residual.
4. **Multi-Head Attention.**
   - Compute query, key, value tensors via shared Linear projections; reshape to `(seq, batch, heads, head_dim)`.
   - Apply rotary or additive position bias if enabled, mask with causal upper-triangular mask, scale by $\frac{1}{\sqrt{\text{head\_dim}}}$, and reuse the cache-friendly loop already provided in the stub comments.
5. **Feed-Forward Network.**
   - Default to GELU two-layer MLP with width multiplier 4×; allow toggling to SwiGLU for the Llama3-lite preset.

- Insert dropout hooks that become no-ops during evaluation to match the starter code style.

6. **Forward Pass and Output Head.**
   - Accept input as `(seq, batch)`, emit logits `(seq, batch, vocab)`.
   - Tie the embedding weights for the output linear layer unless a variant explicitly disables sharing.

7. **Initialization and Guard Rails.**
   - Adopt variant-specific initializers (GPT2 uses scaled normal, Llama-style uses Xavier uniform).
   - Assert `n_heads` divides `d_model` and `n_blocks` ≤ 10 to keep runs within the assignment budget.

8. **Testing Hooks.**
   - Add unit checks that random inputs produce finite logits and that the KV-cache path matches the dense attention for small sequences.
   - Benchmark the synthetic dataset with three quick batches to record loss and throughput per variant.

Transformer Variants and Feasibility

- **minGPT-inspired (Feasible).**
  - Architecture: `d_model` 384, `n_heads` 6, `n_blocks` 6, GELU feed-forward width 1536, learned positional embeddings, tied output weights.
  - Implementation status: fully achievable with the outline above; requires no external dependencies.
  - Metrics plan: reuse the synthetic benchmark harness (20 batches, batch size 32) and report loss/tokens-per-second next to the KGram results.

- **GPT2-small style (Feasible with scaling).**
  - Architecture: adapt GPT-2 small to assignment limits by using `d_model` 768, `n_heads` 12, `n_blocks` 8 (reduced from 12 to stay ≤ 10), learned absolute positions, MLP width 3072, GELU activation, tied output head.
  - Implementation status: feasible once RMSNorm is replaced with LayerNorm for this preset or we supply a LayerNorm toggle; compute cost is higher but manageable on the synthetic corpus.
  - Metrics plan: same synthetic harness; for TinyStories we will downscale batch size to keep runtimes under the cap.

- **GPT-oss (Feasible, requires rotary).**
  - Architecture: `d_model` 512, `n_heads` 8, `n_blocks` 8, rotary positional embeddings, SwiGLU feed-forward width 2048, untied output head per reference design.

  - Implementation status: feasible after we add a light-weight rotary embedding helper; no external kernels needed.
  - Metrics plan: record synthetic benchmark figures and optionally a short TinyStories run (max steps per epoch = 1) to stay within time limits.

- **Llama3-lite (Partially feasible).**
  - Architecture: distilled from the published spec with `d_model` 1024, `n_heads` 16, `n_blocks` 8, rotary positions, SwiGLU feed-forward width 8192, RMSNorm everywhere, untied output head.
  - Implementation status: core building blocks (rotary, SwiGLU, RMSNorm, attention) fit our codebase; full Llama3 features (grouped-query attention, sliding window) exceed current scope. We propose implementing a simplified variant without the extra attention optimisations.
  - Metrics plan: we can capture synthetic benchmark numbers; collecting full TinyStories metrics would demand GPU time beyond the assignment budget, so we will document only CPU synthetic runs unless instructed otherwise.

- **Performance Collection Summary.**
  - All small-batch synthetic metrics are attainable once the Transformer is implemented.
  - Full-scale TinyStories training for GPT2-small or Llama3-lite is out of scope without extensive runtime; we will note this limitation in results to keep expectations clear.

- **Recorded Metrics (Synthetic Corpus, 3 batches, batch size 16).**
  - GPT-oss: average loss 11.05, throughput 11.94 tokens/s, elapsed 1.26 s.
  - Benchmarks are gated behind `--collect_transformer_metrics`; the number above matches the `RECORDED_TRANSFORMER_BENCHMARK` entry in `pico-llm.py`.