Report on the GPT-OSS-20B Transformer Architecture

Cross Entropy Loss

Cross entropy loss is applied by reshaping the model output logits from shape `[batch, seq_len, vocab_size]` and the target labels from `[batch, seq_len]` into flattenedtensors, then computing `torch.nn.CrossEntropyLoss()` between the predicted probability distribution over the vocabulary and the ground-truth next-token indices at each position.

Token Generation

The `TokenGenerator` class (`src/execution/TokenGenerator.py`) generates tokens autoregressively by repeatedly feeding the current sequence through the model, selecting the next token using a configurable `token_choice_fn`(either greedy argmax or nucleus sampling), appending it to the sequence, and iterating until the desiredgeneration length is reached.

Nucleus Sampling vs. Greedy DecodingComparing

Generation results with and without nucleus sampling reveals significant differences in output qualityand diversity. Greedy decoding (`TokenGenerator.token_choice_greedy`) always selects the highest-probabilitytoken at each step, which produces deterministic, often repetitive text that can get stuck in loops or producegeneric outputs. In contrast, nucleus sampling (`TokenChoiceAdvanced` in `src/execution/TokenChoiceStrategy.py`)restricts sampling to the smallest set of tokens whose cumulative probability exceeds the threshold `top_p`,then samples from this "nucleus" according to the renormalized distribution. This approach balances coherence with diversity: by excluding the long tail of unlikely tokens, it avoids nonsensical outputs while still allowing creative variation. In our experiments, nucleus sampling with `top_p=0.9` produced more natural, varied text compared to greedy decoding, which tended to repeat common phrases.

Impact of Temperature

The temperature parameter `t` in `TokenChoiceAdvanced` controls the sharpness of the probability distribution before sampling. Dividing logits by temperature (`scaled_logits = model_logits / self.t`) has the following effects: when `t < 1.0`, the distribution becomes more peaked, making high-probability tokens even more likely and reducing randomness—this produces more conservative, predictable outputs. When `t > 1.0`, the distribution flattens, giving lower-probability tokens a better chance of being selected—this increases creativity but risks incoherence. At `t = 1.0`, the original model distribution is preserved. In practice, combining moderate temperature (e.g., `t=0.8`) with nucleus sampling (`top_p=0.9`) yields the best balance between fluency and diversity.

Per-Token Accuracy Analysis

The per-token accuracy figure generated by `PerTokenAccuracy` (`src/execution/PerTokenAccuracy.py`) reveals howprediction difficulty varies across sequence positions. Typically, accuracy is lowest for the first few tokens where the model has minimal context, then increases as more context becomes available. However, accuracy may plateau or slightly decrease at very long positions due to the model's limited capacity to leverage distant context effectively. This pattern confirms the importance of context length in autoregressive language modeling and suggests that techniques like sliding window attention (used in every other layer of GPT-OSS-20B) help maintain reasonable performance across longer sequences while managing computational costs.