Calvin AI/ML Objectives Reflection | *Apr 27, 2026* Name: _____

For each objective, mark your confidence: **Met** (I can do this), **Developing** (I'm getting there), or **Not yet**. Then pick 1–2 where you're unsure and jot a brief note about what evidence you have or what you still need to work on.

Tuneable Machines

TM-LLM-Embeddings I can identify various types of embeddings (tokens, hidden states, output, key, and query) in a language model and explain their purpose. I can distinguish between token embeddings (input lookup table) and context embeddings (output of transformer layers); I can explain how context embeddings incorporate information from other tokens via attention; I can describe how the final context embedding is used to produce next-token logits (dot product with token embeddings).

TM-SelfAttention I can explain the purpose and components of a self-attention layer (key, query, value; multi-head attention; positional encodings). I can explain what queries, keys, and values represent and how they interact (dot product → softmax → weighted sum); I can compute a simple attention calculation by hand given Q, K, V vectors; I can explain why multi-head attention is useful (different heads can attend to different relationship types); I can explain why causal masking is needed for autoregressive models.

TM-TransformerDataFlow I can identify the shapes of data flowing through a Transformer-style language model. Given model hyperparameters (layers, heads, hidden dim, vocab size, seq length), I can state the shape of key tensors (embeddings, Q/K/V, attention weights, logits); I can trace how a single token's representation changes from input embedding through attention and MLP layers to output logits; I can explain the role of residual connections in preserving information across transformer layers.

TM-LLM-Generation I can extract and interpret model outputs (token logits) and use them to generate text. I can explain the autoregressive generation loop (prompt → logits → sample → append → repeat); I can explain how temperature affects the sampling distribution (higher = more random); I can write pseudocode for a basic text generation algorithm given a model and tokenizer.

TM-LLM-Compute I can analyze how computational requirements scale with model size and context length, and reason about the feasibility of training and running generative AI systems. I can estimate memory requirements for a model given its parameter count and numerical precision; I can explain how compute scales with parameters and sequence length; I can describe at least two optimization techniques (e.g., quantization, KV caching) and their trade-offs; I can evaluate whether a given model can run on specific hardware (e.g., 12GB GPU, laptop CPU, cloud API).

Optimization Games

OG-Eval-Experiment I can design and execute valid experiments to evaluate model performance. I partition data appropriately (train/val/test) before any model fitting; I can explain why we need held-out data and what goes wrong without it; I can select metrics appropriate to the task and stakeholder needs; I can interpret learning curves (loss/metric vs epoch) to understand training dynamics.

OG-LLM-Tokenization I can explain how inputs get chunked into tokens, how outputs are generated token by token, and how this affects usage of the model. I can describe the tokenization pipeline (text → token IDs → embeddings) and its reverse; I can explain why subword tokenization is used instead of character-level or word-level approaches; I can identify at least one consequence of tokenization choices (e.g., multilingual bias, difficulty counting syllables, spelling quirks).

OG-LLM-ConversationAsDocument I can explain how a conversation with an LLM can be represented as a carefully structured document, including system messages, tool calls, and multimodal inputs and outputs. I can describe how system, user, and assistant turns are serialized into a single token sequence; I can explain how function/tool calls fit into the conversation document format; I can explain how this framing allows a next-token predictor to behave as a dialogue agent.

OG-LLM-Prompting I can critique and refine prompts to improve the quality of responses from an LLM. I can identify why a given prompt produces poor results (ambiguity, missing context, wrong framing); I can apply at least two prompting strategies (e.g., role assignment, few-shot examples, chain-of-thought, structured output constraints); I can explain the difference between system, user, and assistant messages and when to use each.

OG-LLM-ContextAndTools I can construct effective inputs for LLM-powered systems and use tool calling to connect models to external information. I can trace how a prompt is assembled from components (system message, examples, tool results, conversation history) and explain why each part is there; I can build an LLM-powered system that uses structured outputs and at least one tool call; I can identify when adding context (examples, retrieved docs) helps vs. when it wastes the context window or distracts the model; I can diagnose failures in an LLM-powered system (e.g., hallucinating instead of using retrieved context, irrelevant tool results, prompt injection).

OG-LLM-Eval I can apply and critically analyze evaluation strategies for generative models. I can explain why evaluating generative systems is harder than evaluating classifiers; I can describe at least two evaluation approaches (e.g., perplexity, human preference, task-specific metrics, LLM-as-judge); I can identify limitations of automatic metrics for open-ended generation; I can design a basic evaluation strategy for a specific LLM application.

OG-SelfSupervised I can explain how self-supervised learning can be used to train foundation models on massive datasets without labeled data. I can explain next-token prediction as a self-supervised task (the "labels" come from the data itself); I can connect cross-entropy loss and perplexity to the idea of prediction quality and "surprise."; I can explain why self-supervised pretraining enables capabilities that weren't explicitly trained for.

OG-LLM-Train I can describe the overall process of training a state-of-the-art dialogue LLM such as Llama or OLMo. I can identify the three main stages of training (pretraining, supervised fine-tuning, RLHF/RLVR) and what each accomplishes; I can explain what data is used at each stage and where human input enters the process; I can describe how the model's behavior changes across stages (mimicry → instruction following → aligned responses); I can explain the basic insight of scaling laws (more data + more compute → predictably better models) and its practical implications.

OG-Theory-Feedback I can explain how feedback tuning can improve the performance and reliability of a model or agent. I can explain the basic RLHF loop (generate samples → collect preferences → train reward model → optimize policy); I can explain what a reward signal is and give examples for different tasks (human preference, code correctness, factual accuracy); I can articulate why the reward signal is the hard part (reward hacking, specification gaming, proxy objectives); I can describe how RLVR (RL with verifiable rewards) simplifies the reward problem for certain tasks.

Overall

Overall-Impact I can analyze real-world situations to identify potential negative impacts of AI systems. Given a scenario, I can identify at least two distinct stakeholder groups who might be affected differently; I can articulate how training data distribution might differ from deployment conditions; I can identify feedback loops where model outputs might affect future training data or user behavior; I can flag concerns that warrant careful analysis before deployment.

Overall-LLM-Failures I can identify common types of failures in LLMs, such as hallucination (confabulation) and bias. I can explain what hallucination/confabulation is and why LLMs are prone to it; I can identify at least two other failure modes (e.g., bias amplification, prompt injection, sycophancy, inconsistency across turns); I can describe strategies for mitigating specific failure modes in a given application context.