Calvin AI/ML Objectives Reflection | *Mar 2, 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-MLPParts I can compute the forward pass through a two-layer classification neural network by hand (or in simple code) and explain the purpose and operation of each part. I can identify the sequence of operations in a two-layer classification neural network; I can compute the output of the matrix multiplication and the activation function for each layer; I can define the following terms in the context of a neural network - input, output, weights, biases, activation function, activation value, logit, probability, and loss.

TM-LinearLayers I can implement linear (fully-connected) layers using efficient parallel code. I can write a correct linear layer computation (matrix multiply plus bias) given weight and bias tensors; I can explain why linear layers are called "fully connected."; I can predict the shape of the output given input and weight dimensions.

TM-ActivationFunctions I can implement and explain elementwise nonlinear activation functions. I can implement ReLU and explain what it does to negative vs positive values; I can explain why networks need nonlinear activations between layers.

TM-Softmax I can implement softmax and explain its role in classification networks. I can implement softmax and explain why it produces a valid probability distribution; I can explain when and why we use softmax in neural networks; I can identify the relationship between softmax and cross-entropy loss.

TM-DataFlow I can draw clear diagrams of the data flow through a neural network, labeling each layer and the tensor shapes at each step. I can draw a diagram showing layers, activations, and the flow of data through an MLP; I can label the shapes of tensors at each point in the network (including batch dimension); I can distinguish between parameters (weights, biases) and activations in my diagrams; I can trace how a single input becomes a prediction and then a loss value.

TM-DotProduct I can compute and reason about dot products of vectors. I can compute the dot product of two vectors by hand; I can explain that dot products measure similarity or alignment between vectors; I can explain what it means geometrically when two vectors have a dot product of zero.

TM-TensorOps I can reason about matrix multiplication and multi-dimensional tensor shapes. Given tensor shapes, I can predict whether a matrix multiplication is valid and what shape results; I can interpret what each dimension of a tensor represents in context (e.g., batch, features, classes); I can diagnose and fix shape mismatch errors by tracing dimensions through operations.

TM-Embeddings I can explain how neural networks represent data as vectors (embeddings) where geometric relationships encode meaning. I can explain that similar items should have similar embeddings (high dot product or small distance); I can interpret a 2D visualization of embeddings and explain why clusters form; I can explain why learned embeddings can be more useful than hand-crafted features; I can describe how pretrained embeddings enable transfer learning.

TM-RepresentationLearning I can explain how a neural network learns useful internal representations through training. I can explain that hidden layers transform data to make the task easier for subsequent layers; I can give an example of a representation that would make classification easier (e.g., linearly separable); I can explain why deeper networks can learn more complex representations.

TM-Autograd I can explain the purpose of automatic differentiation and identify how it is used in PyTorch code. I can explain that autograd computes gradients of the loss with respect to all parameters; I can identify which tensors require gradients and why (requires_grad=True); I can explain what loss.backward() and optimizer.step() do in a training loop.

TM-Implement-TrainingLoop I can implement a basic training loop in PyTorch. I can write a training loop that iterates over batches and updates parameters; I can correctly order the steps - forward pass, loss computation, backward pass, optimizer step, zero gradients; I can add validation evaluation at appropriate points in training; I can track and plot training and validation metrics over epochs.

Optimization Games

OG-ProblemFraming-Supervised I can frame a problem as a supervised learning task with appropriate inputs, targets, and loss function. I can identify what the input features and target variable should be for a given problem; I can determine whether the task is regression or classification; I can select an appropriate loss function for the task type (MSE for regression, cross-entropy for classification); I can articulate what “success” means for the task and how to measure it.

OG-ProblemFraming-Paradigms I can distinguish between supervised learning, self-supervised learning, and reinforcement learning. I can identify which paradigm applies to a given learning scenario; I can explain what provides the learning signal in each paradigm (labels, prediction task, rewards); I can give an example problem suited to each paradigm; I can explain why imitation (supervised) differs from exploration (RL) in what can be learned.

OG-LossFunctions I can select and compute appropriate loss functions for regression and classification tasks. I can compute MSE loss given predictions and targets; I can compute categorical cross-entropy loss given predicted probabilities and true class; I can explain why we use cross-entropy rather than accuracy as a training loss; I can identify which loss function is appropriate for a given task type.

OG-DataDistribution I can reason about how the distribution of training data shapes what a model learns. I can identify ways a training distribution might differ from deployment (selection bias, domain shift); I can explain how data augmentation expands the effective training distribution; I can contrast supervised learning (distribution given) with RL (distribution shaped by exploration); I can give an example of how a model might succeed on training data but fail in practice.

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-Generalization I can diagnose and address generalization problems in trained models. I can identify overfitting from learning curves (train loss decreasing while val loss increases); I can identify underfitting (both train and val loss remain high); I can propose appropriate interventions (more data, augmentation, regularization, adjust model capacity); I can explain the bias-variance tradeoff at an intuitive level.

OG-Implement-Validate I apply validation techniques correctly and proactively. I split data into train/validation before fitting, without being reminded; I evaluate on validation data to select hyperparameters, not training data; I spot-check model predictions on specific examples to build intuition; I can explain why validation performance is a better estimate of real-world performance than training performance.

OG-LLM-APIs I can apply LLM APIs (such as the Chat Completions API) to build AI-powered applications. I can construct appropriate API calls with system and user messages; I can process and use the model’s response in an application; I can identify tasks where an LLM API is and is not appropriate.

OG-Pretrained I can explain the benefits and risks of using pretrained models. I can explain how pretrained models provide useful features without task-specific training data; I can describe the “body + head” pattern (pretrained feature extractor with task-specific classifier); I can identify risks of pretrained models (bias, domain mismatch, licensing); I can explain when fine-tuning vs feature extraction is appropriate.

OG-Theory-SGD I can explain how stochastic gradient descent uses gradients to improve model performance. I can explain that the gradient points in the direction of steepest increase of the loss; I can explain why we move in the negative gradient direction to reduce loss; I can explain why we use batches (stochastic) rather than the full dataset; I can identify the role of learning rate and what happens if it’s too large or too small.