How to Train Your Model

HtTYM 2026 · Participant Preparation Guide & Worksheet

A single guide to everything you'll bring to the bootcamp. Read it through once, then use it as your working worksheet as you scope your project across the July preparation meetings.

How to use this document

1. **Make your own copy.** Save a copy to the HtTYM Student Folder. Name it **LastName_FirstName** (e.g., Vater_Ashley).

2. **Fill in the worksheet fields as you go.** Blank lines, checkboxes (☐), and writing boxes are for you. The shaded reference panels explain what each piece is and point to examples — you don't edit those.

3. **Bring your drafts to each July meeting.** You'll present a rough draft at the first meeting, a revised version mid/late July, and a finalized version when we convene in person on Sunday 8/2.

Heads up: arriving prepared is required. Consistent effort on this worksheet is what makes the in-person week productive — participants who don't make progress may be unable to take full advantage of the event.

1. The big picture

HtTYM is built around one moment: **hitting "run" on a real training job during the in-person week.** Everything in the preparation period exists so that, when you sit down on with us on Sunday, every piece you need is already in hand and to some degree even tested. The point of this guide is to make sure nothing is missing.

To train a model end-to-end you assemble six things. They fit together like this: your **data** is loaded and turned into features, fed through your **model**, scored by a **loss function** and tracked with **metrics**, while a **logging** system records progress and your **documentation** captures what you did.

The six deliverables you'll assemble

1. **Project scope & training approach** — a well-defined question, and a decision about whether you're fine-tuning, rewriting, or building from scratch.
2. **Dataset** — your data, accessible and organized, in the exact format you'll train on (split into train/validation/test).
3. **Dataloading & featurization script** — code that reads your files and turns raw data into model-ready features (and batches them).
4. **The model** — a block diagram of your architecture plus a working draft of the model code.
5. **Output analysis: loss & metrics** — code implementing your loss function(s) and the metrics you'll use to judge success.
6. **Progress monitoring & logging** — a tested way to record and visualize metrics as training runs.

(Process documentation & note-taking runs alongside all six — see Section 8.)

"Hit run" readiness checklist

By the start of the in-person week (8/2) you should be able to tick every box:

- ☐ Dataset organized and accessible, split into train/validation/test
- ☐ A script that reads the data into Python and produces model-ready, batched features
- ☐ A model (block diagram + code) that runs locally on fake data or a small subset
- ☐ Early draft of loss function(s) and evaluation metrics implemented and giving sane values on a dry run
- ☐ A logging solution that reports the first few batches/epochs correctly
- ☐ A note-taking / documentation plan so your process is reproducible

Before vs. during the event. Some deliverables must be finished and tested *before* you arrive; others are the actual work of the in-person days. The rule of thumb: anything that *feeds* the

training run — your data, your model's design, and your logging — should be locked in beforehand, so that Sunday and Monday go to writing and debugging the **model** and its **loss function**, not assembling inputs. Expect everything to get tweaked once training is underway. "Finalized before" means *ready and tested*, not *frozen*.

Before the event — *finalized & dry-run tested*	**During the event (Sun–Mon) — *primary build***
Project scope & training approach	Model **code** — maybe a rough draft beforehand, but the real work happens here
Dataset — sorted, split, accessible	**Loss function** — written and wired into the training script
Dataloading/featurization runs on a small subset	Evaluation **metrics** — *selected* beforehand, *implemented* now
Model **block diagram(s)**	Verify logging reports correctly on the first real batches/epochs
Progress monitoring & logging set up and tested	*Continuous:* notes/documentation, and refining everything as results come in

** Full featurization may keep firming up during the event; what matters before is that data demonstrably flows into a model-ready tensor.*

2. Choose your training approach BEFORE THE EVENT

Your first decision is **how** you'll train. This sets the scope of everything that follows. Pick the level that matches what you can realistically do this summer — all levels are welcome, and a tightly-scoped fine-tune may be a better "weekend" project than an over-ambitious from-scratch build.

Training-type option	Choose this if right now you can / have…
Fine-tune an off-the-shelf model	Surveyed model options and have ideas about what might suit your project.
Rewrite a model, based on an existing architecture	Done a deep exploration of the starting model's architecture (e.g., you understand and can document its base classes).
Create a model from scratch	Deep knowledge of existing architectures, with an example and pre-identified documentation to use as a starting point.

Your one-sentence problem statement

By the end of this worksheet you should be able to write a statement as specific as this:

> *"I want to **fine-tune** the gradient-boosted decision tree model RealKcat, with a **glycosyl-hydrolase-specific** dataset (d2dcure.com), such that my new model is predictive of **designs/variants with increased kinetic activity.**"*

Your draft problem statement (1–2 sentences, refine it over July):

3. Your model BEFORE THE EVENT DURING THE EVENT

What this deliverable is

Two things: a **block diagram** of your model (a schematic of the components and how they connect — inputs, outputs, labels, and connectivity for each block) and a **draft of the model code**. The diagram can capture the whole model at a high level, or break a larger model into component diagrams. You'll develop it iteratively: high-level first, then more detail (subcomponents, internal loops like cycling), then a final version at the right level of detail.

Worked examples to study: The Illustrated Transformer (high → low level), RFdiffusion (Fig 1a), ProteinMPNN (Fig 1), AlphaFold2 (Fig 1e), Chai-1 (Fig 1), ThermoMPNN (Fig 1A).

block diagram before · model code during

Define your model

Starting / template model's name: _____

Confirm:

- ☐ My starting model is open source, or I have access to the code.

Model type

- ☐ LLMs / Protein Language Models
- ☐ GNNs
- ☐ Diffusion model
- ☐ CNNs
- ☐ Other (describe): _____

Relevant language / framework

- ☐ PyTorch
- ☐ TensorFlow
- ☐ JAX
- ☐ Other: _____

Relevant publication(s) you've already read *(list + links):*

Model's GitHub / HuggingFace page *(if relevant):* _____

Block diagram of your model

Sketch the layers and the flow of data through them. You'll build this up across the three drafts below; it's fine to be unsure of many elements early on.

- **First draft (first July meeting):** the model at a very high level — inputs, outputs, and the main blocks.
- **Working draft (second July meeting):** more detail — subcomponents and any internal data loops (e.g., cycling). Totally dependent on your model and training type.
- **Final draft (first in-person day, Sun 8/2):** the architecture at the most appropriate level of detail. If fine-tuning, describe the changes you're making to the base model (e.g., a new head) and anything else worth discussing.

Draft your high-level block diagram here *(or paste / link an image):*

4. Your data BEFORE THE EVENT

What this deliverable is

Your dataset, plus the **code that turns it into something the model can consume.** This breaks into three jobs:

- **Dataloading** — loading information from your dataset files into your script. Often includes loading specific splits, filtering invalid samples, and multithreading/multiprocessing.
- **Featurization** — transforming raw information into "model-ready" features. Features are model-specific (geometric, chemical, physical, evolutionary…) and usually involve normalizations/transforms (e.g., one-hot), returning Arrays/Tensors.
- **Collation** — stacking individual examples into a batch for the model.

Code examples to learn from: ThermoMPNN, PIPPack, Boltz-2, and EvoDiff all have readable dataloading, featurization, and collation code (CSV, PDB, NPZ, and FASTA inputs).

Describe your dataset

Aim for the level of detail in these examples:

Example 1: "My dataset is publicly available, ~1300 records of numeric enzyme-variant kinetic (Kcat, KM) and thermal stability data (T50, TM), in CSV. I may need to compute delta values from WT-matched records to prepare it for training."

Example 2: "My dataset was created by downloading embeddings for 2.2M bacterial DNA-binding protein sequences from UniProt. Each embedding is dimension 2040, saved as gzipped .pt."

Your dataset description:

Link to dataset *(if available):* _____

Confirm:

☐ My dataset is open source, or I have access to use it for this purpose.

Dataset format

☐ CSV
☐ JSON
☐ PT

☐ Other (describe): _____

Exactly how many records does your dataset contain? _____

Exactly how many fields / variables does it contain? _____

What data processing / munging will you need? *(e.g., train/val/test splitting, dedup, tokenize…)*

On a scale of **1–10**, how confident are you that you can write a Python script to load your input data into model-ready features? _____

All comfort levels are welcome — this just tells us how much support to plan for.

5. Predicted output — what you're trying to predict DURING THE EVENT

This section pins down what your model produces and how you'll know it worked.

Protein system you're working with

- ☐ Enzyme structure / function
- ☐ Antibody structure / function
- ☐ Small-molecule docking
- ☐ Binders
- ☐ Glycosylation / glycan structures
- ☐ Multimeric complex structure prediction
- ☐ Peptides
- ☐ DNA / protein interactions
- ☐ Other (describe): _____

Briefly: your input features, and what you're trying to predict. *(e.g., "Using sequences only, I want to predict optimal enzyme temperature in °C.")*

Your "Output of Interest" (your "Y-axis" variable):

_____

Sketch a results figure (≤4 panels) that would summarize your trained model. *Propose at least one panel addressing your output of interest — e.g., Panel A loss vs. epoch, Panel B AU-ROC, Panel C UMAP of features, Panel D ablation box-plot.*

Your model's inputs and how many. *(e.g., "list of 100 FASTA files of beta-glucosidase single-point variants")*

Where will you get those inputs? *(e.g., "we have a ready-to-go list of these FASTA files")*

Loss & evaluation metrics

What this deliverable is

Loss is your measure of how well (or poorly) your model's predictions match the true target values. The **loss function** quantifies error for a batch of examples; training **minimizes** it via an optimization algorithm.

A common regression loss is **mean squared error (MSE)** — the average squared difference between predicted and target values: MSE = $(1/N)\ \Sigma\ (y_i - \hat{y}_i)^2$ over a batch of N examples. Other losses suit binary/multiclass classification or specific data types. See PyTorch loss functions.

An evaluation metric measures overall performance on a dataset (usually a test set). Classification often uses **accuracy, precision, recall**; regression may use MSE; protein-design tasks often use **sequence recovery** and **RMSD**. Ready-made metrics: TorchMetrics (PyTorch) or the HuggingFace evaluate module.

This may overlap with your logging choices (Section 7).

What loss function(s) and accuracy measures are you considering? *Provide rationale, and the equations where you can. (e.g., "Binary Cross-Entropy loss; measure regression performance with RMSD.")*

6. Progress monitoring & logging BEFORE THE EVENT DURING THE EVENT

What this deliverable is

A logging system lets you **track and visualize** training/evaluation metrics, hyperparameters, artifacts, and system stats as your model runs. It pays off in four ways: **reproducibility** (re-run under the same conditions), **debugging** (spot bad runs early), **comparison** (evaluate configs side-by-side), and **collaboration** (share dashboards).

set up before · verified during

Common approaches

Approach / tool	What it gives you
Local file logging *CSV, JSON, logs*	Most basic: write metrics with print() or structured files. No built-in UI, versioning, or artifact logging — minimal overhead, limited utility.
TensorBoard	Native to TF/PyTorch (SummaryWriter). Logs scalars, histograms, images, graphs, with visualization. Get started
Weights & Biases	Cloud or self-hosted. Auto-logs metrics, config, artifacts, system metrics; rich UI, comparison, sweeps. Docs
Neptune.ai	Tracking server + UI; projects, models, registry; logs metrics, params, artifacts, code. Docs
Comet	Auto-logs code, metrics, images, system stats; good framework integration. Docs
Others	ClearML, Polyaxon, Guild AI, Sacred + Omniboard.

Your logging plan

Pick an approach that fits your background and your project's complexity — deeper models (e.g., large language models) generally call for more capable logging.

Logging solution you'll use: _____

Why this choice *(fit for your background + project complexity):*

What “done” looks like before the event

- Offline, run a basic dry run of your logging on fake data or a simple task; confirm every metric is reported accurately.
- Aim to log: initialization (if applicable), batch-level metrics (loss), and epoch-level metrics (loss + accuracy / RMSE / whatever fits your objective).
- If the dry run works, there should be little new code needed during the event. Reach out to the instructor team early if anything misbehaves.

7. Process documentation & note-taking THROUGHOUT

Why this matters

Good documentation has two payoffs: **you** will remember what you did two months from now, and **someone in your lab** could reproduce and build on what you learned here. This worksheet is itself the start of your process documentation.

Good practice

- Keep notes throughout prep and the event — what you tried, what broke, how you troubleshot it, and results.
- Be active on the dedicated **Slack** channel: post issues using the help-post template, and help resolve others' issues.
- Capture your dry-run write-up: how you set it up, problems hit, troubleshooting done, results, and any hyperparameters you'd like to tune.
- During the event, consider short screen recordings or a running log of what you did each day.

Your note-taking / documentation plan: *(where will notes live, and how will you keep them?)*

8. Final readiness check

Before the in-person week, walk back through and confirm you can tick all six:

- ☐ Project scoped: clear problem statement + chosen training approach (Sections 2–3)
- ☐ Dataset accessible, organized, and split (Section 4)
- ☐ Dataloading / featurization script produces batched, model-ready features (Section 4)
- ☐ Model block diagram + code that runs locally on a small subset (Section 3)
- ☐ Loss + metrics drafted, bonus points if you can do sanity check, trial dry run (Section 5)
- ☐ Logging tested on a dry run (Section 6)
- ☐ Documentation / note-taking plan in place, Slack access confirmed (Section 7)

Questions as you work through this? Bring them to your breakout group, post them in Slack, or reach out to the instructor team. That's what the July meetings are for.