Interactive Training 2: Auditable Control Plane for Live Model Training

Wentao Zhang[1,*] **Xuanhe Pan**[2,*] **Han Zhou**[1,*]
Yang Lu[2,†] **Yuntian Deng**[1,†]
[1]University of Waterloo [2]University of Wisconsin-Madison
[*]Equal contribution [†]Equal advising
{w564zhan,h46zhou,yuntian}@uwaterloo.ca {xpan73,ylu97}@wisc.edu

Abstract

Experiment trackers show how training is progressing, but changing a live run still usually requires trainer-specific code. We present **Interactive Training 2**, an open-source control plane for steering training through a shared protocol. Training applications declare which settings and actions they expose, humans and automated controllers submit requests through the same interface, and the training loop validates and applies them at safe control points. A customized Aim workspace combines live metrics and controls with a chronological record of requests and outcomes. We demonstrate the system across five NLP and reinforcement-learning workflows. The released code and traces provide a reusable foundation for auditable human- and agent-guided training.

1 Introduction

Researchers rarely treat model training as a completely passive process. They watch the loss curves, inspect evaluations, and decide whether to lower the learning rate, rebalance the data, save a checkpoint, or stop an unstable run. Modern experiment trackers make these decisions easier by showing what is happening, but they offer no general way to carry them out. Acting on an observation still usually requires custom callbacks or logic written for one trainer and one experiment. In practice, researchers already steer training in response to live behavior (Zhang et al., 2022; Walsh et al., 2025), but the tooling for doing so remains fragmented.

Interactive Training 2 makes a training run steerable through a shared interface. The training application declares which settings may be changed and which actions may be requested. A human, script, heuristic, or LLM agent can then submit a request through the same protocol. The training loop remains in control: it decides when the request can be safely applied, validates it, and records the result before continuing. For example, an application may expose a typed `learning_rate` setting together with `evaluate` and `save_checkpoint` actions. A controller may request a learning rate of 2×10^{-5}, but the optimizer is changed only when the loop reaches a designated control point. We call this layer between training code and its controllers a *control plane*. It defines what can change, when changes take effect, and what is recorded.

Interactive Training v1 showed that a human or constrained LLM could modify a live Hugging Face training job through a control server (Zhang et al., 2025). That system was built around a fixed vocabulary of commands executed through Hugging Face Trainer callbacks. Interactive Training 2 turns this trainer-specific demonstration into a reusable protocol. Training applications can register their own settings and actions, different training loops can choose their own control points, and different kinds of controllers can interact with all of them through the same interface.

Figure 1 previews this interaction in the customized Aim workspace. A researcher can inspect live metrics, request a change, and see whether and when it was applied. The accompanying journal connects plans and controller decisions to requests, results, evaluations, and checkpoints. We call the system *auditable* because it is possible to reconstruct who requested each action, what the request contained, whether it succeeded, and what happened around it. The goal is to let researchers add human or automated steering to different training workflows without rebuilding the monitoring, control, and logging machinery for each experiment.

Our contributions are:

- **A shared protocol for steering live training runs.** Training applications expose typed settings and structured actions, while training loops determine the safe points at which queued requests may take effect.
- **A unified interface for observing, control-**

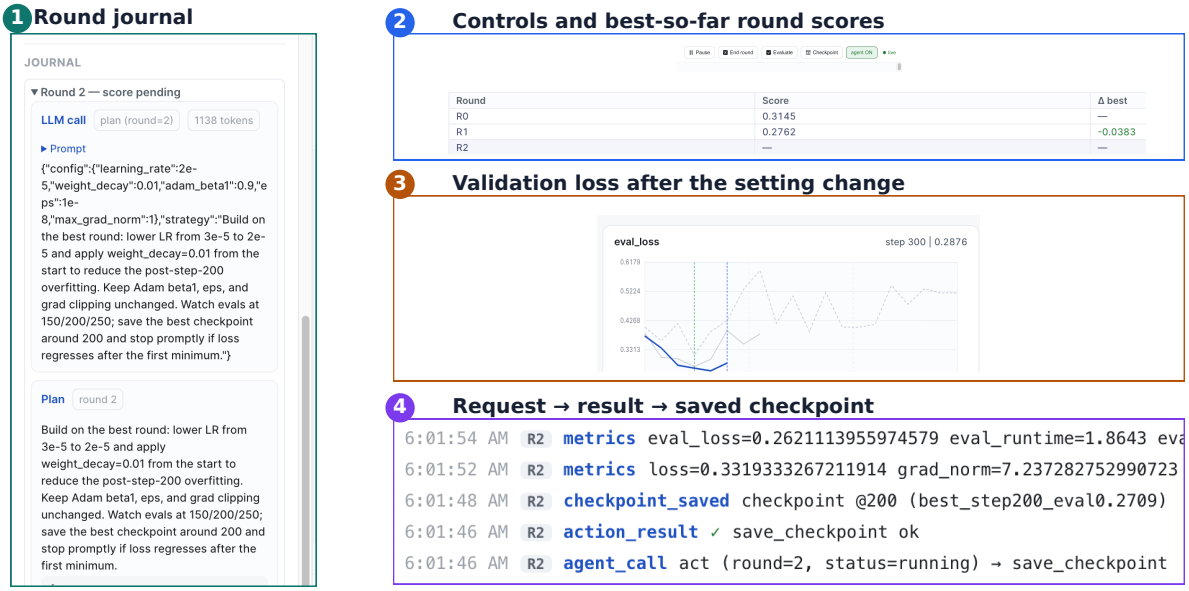

Figure 1: **One request from plan to result in the Aim Live workspace.** Enlarged crops from one illustrative BERT run show (1) the Round 2 journal and proposed change, (2) Controls and best-so-far round scores, (3) Validation loss after the setting change, and (4) Logging area showing the request, result, and saved checkpoint.

ling, and reviewing training. A customized Aim workspace combines live metrics and controls with an ordered journal that connects controller decisions to requests, execution results, evaluations, and checkpoints.

- **An open-source implementation demonstrated across five workflows.** The same protocol supports callbacks, optimizer wrappers, and direct PyTorch and reinforcement-learning loops, and can be used by humans, scripts, and LLM agents. The code is available at github.com/yuntian-group/interactive-training and an interactive demo is available at interactivetraining.ai/live.

2 System at a Glance

Consider a researcher fine-tuning BERT. From the Aim workspace, the researcher inspects the live validation trajectory, lowers the learning rate from 3×10^{-5} to 2×10^{-5}, requests an evaluation and checkpoint, and lets training continue. The training loop applies these requests when it reaches a designated control point, records whether they succeeded, and then proceeds with the updated run.

Figure 1 illustrates this shared interaction pattern: observe the run, submit a request, apply it at a safe point, record the outcome, and continue. Humans, scripts, heuristics, and LLM agents all submit requests through the same protocol. The journal keeps each request together with its source, result, and surrounding training context, making the sequence of decisions and outcomes easier to inspect than when it is scattered across console logs.

3 Control-Plane Design

Training code defines each control. The application registers typed settings, which are called knobs in the Python API, and implements the corresponding getters, setters, and actions. The same mechanism can represent an AdamW (Loshchilov and Hutter, 2019) learning rate, source-sampling weight, Muon (Jordan et al., 2024) momentum, or curriculum difficulty in reinforcement learning without putting task-specific logic in the training session. HTTP clients, the UI, and automated controllers all see the same list.

Figure 2 answers three questions: what the training code exposes, who may submit requests, and when the session applies and records them.

The loop decides when changes apply. Requests may arrive at any time, but the training loop applies them one at a time at explicit control points. Human users, scripts, and LLM agents submit the same request format to one queue. The application chooses the control points, validates task-specific actions, and can hide dangerous actions from the LLM agent.

The journal links rounds. Within a round, metrics, requests, results, checkpoints, and controller decisions share one ordered journal. Across rounds,

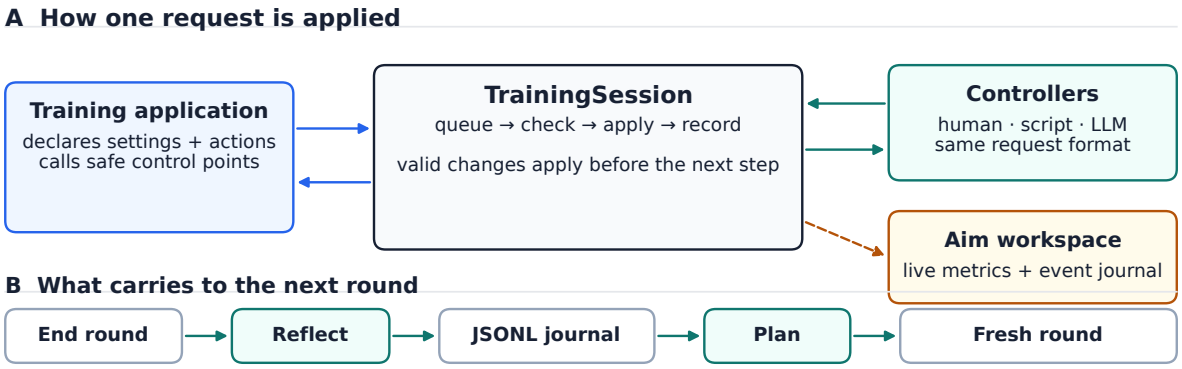

Figure 2: **The shared control protocol.** Training code declares settings and actions and calls `TrainingSession` at safe control points. Human users, scripts, and the optional LLM agent submit the same typed actions; the session validates, applies, and records them before the next training step, while Aim presents live metrics and the event journal. The lower panel shows what carries across rounds; each new round creates a fresh model and optimizer.

the demo scripts start a new model and optimizer, then pass a text summary of configurations, actions, scores, and reflections into the next plan.

4 How the System Works

Figure 2 shows the training loop, the long-lived `TrainingSession`, and the interfaces used by controllers. The training code still owns every model update. Human users, scripts, and the optional LLM agent send requests through the session.

4.1 Registered Settings and Actions

A typed **setting** binds a name to a getter and setter, with a data type, optional range and step size, and a natural-language description. The API method is named `register_knob`. For numerical settings, the session converts and clamps requested values before calling training-owned code. An **action** defines a command such as evaluation or checkpointing. Each submitted request records the action type, its arguments, its source, an identifier, and a timestamp. The HTTP API and LLM agent both receive the same list of registered actions. Examples include `set_knob`, `evaluate`, `save_checkpoint`, `load_checkpoint`, `pause`, and `stop`. Applications can add domain-specific actions, such as freezing a language model embedding component.

A **goal** names the metric used to score a round, such as the best validation loss observed. The current implementation either minimizes or maximizes that value. An **event** records an increasing sequence number, event type, payload, round, and time. Metric reports, plans, LLM calls, setting changes, action results, checkpoints, and reflections all enter the same journal as typed events.

4.2 Applying Requests at Control Points

At each `session.step(metrics)` call, the training process:

1. Records new metrics and snapshots the initial setting configuration.
2. Runs attached controllers on their configured schedule, such as every 100 steps.
3. Drains human and automated requests from one action queue.
4. Runs the matching handler, clamps setting values, and records a result for every request.
5. Continues to service resume and stop requests while paused.

When a loop is idle, `session.pump()` processes pending requests without recording a training metric. The call returns a `StepControl` object that tells the integration whether to stop, evaluate, save or load a checkpoint, reset a module, or update a setting. The training integration translates these decisions into the local framework's semantics.

Table 1 summarizes the five released workflows, how each one integrates the session, what it exposes, and the scores recorded in its journal.

The control point guarantees that when the call returns, every accepted change is already in place before the next step. An HTTP request does not wait for that point. `POST /actions` returns an identifier immediately, and the later `action_result` event says whether the request was applied or rejected. After reconnecting, a client can ask for recent missed events with a `since` cursor.

4.3 Three Ways to Connect Training Code

Trainer. `make_interactive(Trainer)` installs a callback that registers optimizer controls, reports

Workflow	Integration	Settings and actions	Rounds, steps, call frequency	Score (R0 → best)
Sentiment classification	Hugging Face callback	Schedule, AdamW, clipping, evaluation, checkpoint, stop	11 rounds; 1,000 steps; eval/controller 50/100	Validation loss $0.354 \rightarrow 0.220$
Sentiment mixing	Direct PyTorch loop	Source weights, class weights, learning rate	11 rounds; 2,000 steps; eval/controller 100/100	Macro-F1 $0.568 \rightarrow 0.656$
Layerwise GPT	Direct PyTorch loop	27 embedding, block, head, and residual-group LRs	11 rounds; 3,000 steps; eval/controller 100/300	Validation loss $5.346 \rightarrow 5.055$
Muon–AdamW GPT	Direct PyTorch loop	Both LRs, Muon momentum, decay, gradient diagnostics	11 rounds; 3,000 steps; eval/controller 100/300	Validation loss $4.797 \rightarrow 4.429$
RLVR Countdown	Direct RL loop	Policy, clipping, and curriculum controls	8 rounds; 100 iterations; eval/controller 10/10	Hardest accuracy $0.032 \rightarrow 0.232$

Table 1: **What the five released workflows expose.** Each row summarizes one multi-round run. Round 0 uses the original configuration without the LLM agent, and every later round starts with a fresh model and optimizer. Eval/controller frequencies are in steps (iterations for RLVR Countdown). Scores give context for the associated journals.

training and evaluation metrics, and maps session decisions to Trainer control flags (Wolf et al., 2020). When evaluation is scheduled, the callback waits to invoke the LLM agent until fresh evaluation metrics are available. This callback path underlies the BERT (Devlin et al., 2019) sentiment classification demonstrations and requires only replacing the Trainer class and passing a session.

Optimizer patching. For loops that are difficult to edit, an optional wrapper turns `optimizer.step()` into a control point. It saves the returned `StepControl` on the session for the loop to read.

Direct integration. Custom pretraining loops and reinforcement learning with verifiable rewards (RLVR) loops call `session.step` directly. The application chooses which diagnostics to report and how to implement checkpoint or evaluation requests.

4.4 Optional LLM Agent

Any controller can use the shared protocol. The supplied LLM agent demonstrates automated use in three phases. **Plan** receives the task, goal, available settings, remaining rounds, and a short summary of earlier configurations, scores, actions, and reflections. It returns an initial configuration and strategy. **Act** receives recent metric values, current settings, earlier decisions in the round, prior reflections, and tools generated from the registered actions. It may issue several tool calls or intentionally take no action. **Reflect** receives the completed trajectory, action history, score, and earlier lessons, then writes a short actionable lesson and a distinct next variation.

All recorded runs use GPT-5.5 with high reasoning effort as the agent. When the agent is attached, the multi-round driver first runs one reference round without it. Each later round starts a fresh model and optimizer, creates a plan, runs training, computes a score, and writes a reflection to the JSONL journal. Each entry includes the round's starting configuration, best score and step, successful actions, and reflection. Prompts use at most the ten latest entries to bound context growth.

Figure 3 plots the score recorded for every fresh round. Its purpose is to keep unsuccessful rounds visible alongside rounds that set a new best.

Table 2 condenses three cases in which a failed or stalled round changed the next plan.

4.5 Aim and HTTP Interfaces

A thin FastAPI service exposes `/state`, `/actions`, `/events`, and a WebSocket event stream. It owns no training logic. An Aim transport records events on a background thread so writing metrics does not pause training. It creates one Aim run per training round, stores the goal and control endpoint as run metadata, records numerical metrics with round and branch context, and stores controller-related events as text sequences. The customized Aim workspace uses that endpoint both to display metrics and to submit requests.

5 System Validation

Validation scope. We test three things. First, can one protocol represent many kinds of changes? Second, does each request produce a recorded result?

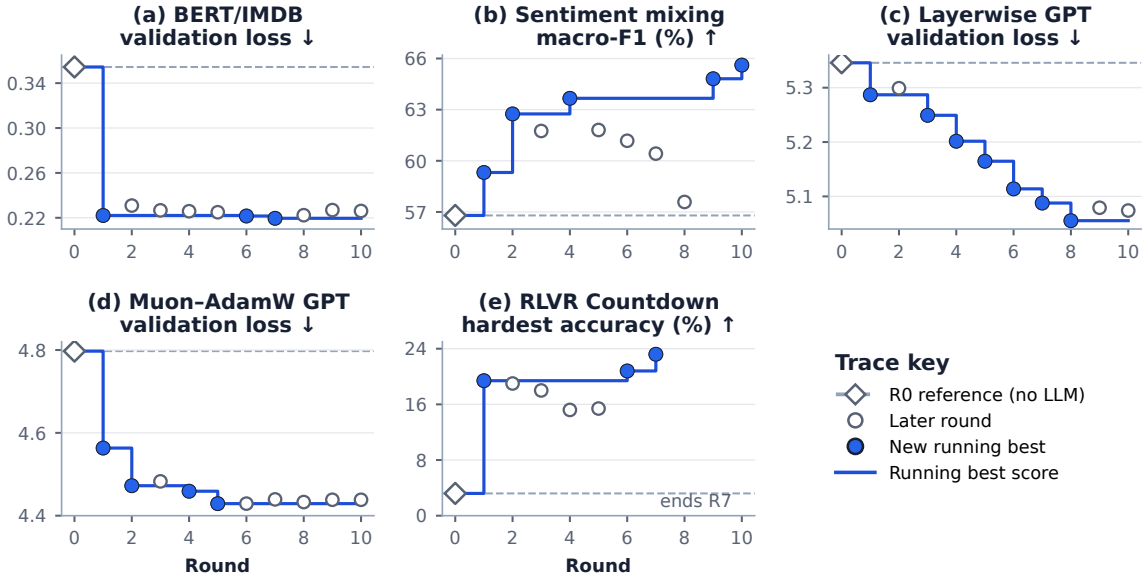

Figure 3: **Scores recorded during five example runs.** Each point is the best score recorded in one fresh round. The solid line tracks the best score seen so far, the diamond and dashed line mark the Round 0 reference without the LLM agent, and filled circles mark a new best. Y-axes use task-specific raw units and fixed limits. All panels share the Round 0–10 x-scale, with Countdown ending at Round 7. Sentiment and Countdown use percentages here. Table 1 reports their decimal values.

Workflow	Scores and next-plan evidence
Sentiment mixing	R8: 0.57586 → R9: 0.64812 → R10: 0.65612. Round 8 used a product-free (McAuley and Leskovec, 2013), tweet-dominant (Barbieri et al., 2020) mixture and regressed. Its reflection proposed balanced tweet and finance (Malo et al., 2014) weights and asymmetric class weights. Round 9 records those choices explicitly.
Muon–AdamW GPT	R4: 4.4591 → R5: 4.4291. The Round 4 reflection names momentum as a setting it has not tried. The Round 5 plan lowers Muon momentum from 0.95 to 0.90.
RLVR Countdown	R5: 0.154 → R6: 0.208 → R7: 0.232. Round 5 starts at the hardest curriculum level and stalls. Its reflection proposes wider clipping and more early exploration; the Round 6 and 7 plans record those changes.

Table 2: Failed rounds appear in the next plan.

Third, can a later round visibly use an earlier reflection? Scores provide context for the recorded decisions; they are not controlled comparisons of optimization algorithms. The artifact includes the five JSONL journals and the fields derived from them. Figure 3 provides the score context, and Table 1 summarizes what each workflow exposes.

5.1 Many Kinds of Changes

The workflows range from optimizer and checkpoint settings to source and class mixtures, 27 embedding, block, head, and residual-group learning rates, two optimizer groups, and an RL difficulty schedule. The same protocol works across these workflows: each application publishes its settings and actions, and controllers read that list. Appendix C and Table 4 give model, data, budget, call frequency, and diagnostic details.

Table 3 compares this shared protocol with Interactive Training v1. Section 6 returns to that distinction.

5.2 From Request to Recorded Result

Figure 1 exposes a complete request path in the deployed interface. At 6:01:46, the LLM agent requests `save_checkpoint`, the shared queue assigns the request an identifier, the next control point runs the application handler, and an `action_result` records success. Two seconds later, `checkpoint_saved` records step 200 and evaluation loss 0.2709. The same journal would record an explicit failure if validation or the handler rejected the request. Beyond this capture, the released JSONL journals record each round's executed actions across all five workflows, so the request-to-result path can be checked against the released traces.

Dimension	Interactive Training v1	This work
Training integration	Hugging Face Trainer callbacks	One `TrainingSession` used by Hugging Face callbacks, optimizer wrappers, and direct loops
What can change	Predetermined command types	Application-registered typed settings and structured actions
When changes apply	Execution at trainer callback boundaries	Applied at developer-chosen control points before the next training step
Controllers	Human dashboard and a limited learning-rate agent	Humans, scripts, heuristics, and agents use one typed protocol
Interface	Separate React dashboard	Monitoring and control embedded in Aim
What is recorded	Commands and checkpoint branches	Ordered requests, results, metrics, checkpoints, plans, and reflections
What reaches the next round	No explicit cross-round journal	Explicit text journal across fresh rounds

Table 3: **From a trainer-specific demonstration to a reusable control protocol.** Live intervention, an HTTP API, human control, checkpoint management, and within-run LLM action are inherited from v1.

5.3 Using Earlier Reflections

Table 2 gives three examples of a later plan using an earlier reflection, while every round still starts a fresh model and optimizer. After Sentiment R8 regresses, its reflection proposes asymmetric class weights; Rounds 9 and 10 record those choices. In Muon–AdamW, the R4 reflection says that momentum has not yet been tried. The R5 plan lowers it from 0.95 to 0.90, preceding the score of 4.4291. After Countdown R5 begins at the hardest curriculum level and stalls at 0.154, the next plans widen clipping, increase early exploration, and sustain a mid-difficulty curriculum before R6 and R7 reach 0.208 and 0.232.

6 Related Work

Experiment trackers such as Aim and Weights & Biases persist metrics (Arakelyan et al., 2020; Biewald, 2020), while orchestration systems manage jobs and resources. Interactive Training 2 uses a tracker as its monitoring and control UI. The training application, not the tracker, decides what may change. Trainer callbacks can make similar changes inside one framework (Wolf et al., 2020). Our system instead gives every controller the same external protocol, records a result for each request, and lets the training loop choose where changes apply.

Hyperparameter optimization, population-based training, and dynamic algorithm configuration choose or adapt configurations according to a search policy (Bergstra and Bengio, 2012; Jaderberg et al., 2017; Akiba et al., 2019; Adriaensen et al., 2022; Shabgahi et al., 2024). They can act as controllers on top of this system. The control plane supplies the available actions, the places where changes apply, and an audit trail; the search algorithm remains separate.

Recent LLM systems plan experiments or change training recipes (Rao and Advani, 2026; Radianis, 2026; Rodrigues et al., 2026). In our system, the LLM agent is optional and uses the same protocol as humans, scripts, and heuristics. Interactive Training v1 is the direct predecessor; Table 3 isolates the new shared protocol from the inherited live-intervention demonstration.

7 Conclusion

Interactive Training 2 turns live training intervention from experiment-specific logic into a reusable interface. Training applications define what may change and when changes may safely take effect; humans and automated controllers decide what to request through a shared protocol, and the journal records the resulting sequence of decisions and outcomes. This separation makes training runs easier to steer, reuse, and inspect across different frameworks and workflows. The released system provides a foundation for interactive training tools and for more systematic studies of how humans and automated agents supervise learning in progress.

Limitations

The training loop waits while the LLM responds. The system also does not yet handle a response that arrives after training has moved on. Applications remain responsible for checking custom actions. Value bounds, type checks, permissions, and human controls limit what an agent may do, but they cannot guarantee safe or optimal actions. High-stakes use needs approval gates, resource budgets, rollback policies, and task-specific safety checks.

References

Steven Adriaensen, André Biedenkapp, Gresa Shala, Noor Awad, Theresa Eimer, Marius Lindauer, and Frank Hutter. 2022. Automated dynamic algorithm configuration. *Journal of Artificial Intelligence Research*, 75:1633–1699.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631.

Gor Arakelyan, Gevorg Soghomonyan, and The Aim team. 2020. Aim. Computer software. Initially released 18 June 2020.

Francesco Barbieri, Jose Camacho-Collados, Luis Espinosa-Anke, and Leonardo Neves. 2020. TweetEval:Unified Benchmark and Comparative Evaluation for Tweet Classification. In *Proceedings of Findings of EMNLP*.

James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305.

Lukas Biewald. 2020. Experiment tracking with weights and biases. Software available from wandb.com.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. 2017. Population based training of neural networks. *Preprint*, arXiv:1711.09846.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. 2024. Muon: An optimizer for hidden layers in neural networks.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. 2024. Fineweb-edu: the finest collection of educational content.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

P. Malo, A. Sinha, P. Korhonen, J. Wallenius, and P. Takala. 2014. Good debt or bad debt: Detecting semantic orientations in economic texts. *Journal of the Association for Information Science and Technology*, 65.

Julian McAuley and Jure Leskovec. 2013. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, page 165–172, New York, NY, USA. Association for Computing Machinery.

Anis Radianis. 2026. Learn-by-wire training control governance: Bounded autonomous training under stress for stability and efficiency. *Preprint*, arXiv:2605.19008.

Anjali Rao and Nikhil Kamalkumar Advani. 2026. AI training manager: Bounded closed-loop control of adaptive training recipes. *Preprint*, arXiv:2606.29871.

Carson Rodrigues, Oysturn Vas, Isaiah Abner DCosta, and Nithish Kumar Prabhakaran. 2026. When is an LLM worth it for hyperparameter optimization? a budget-matched study on tabular data finds the warm-start is a default configuration, not the model. *Preprint*, arXiv:2606.21641.

Soheil Zibakhsh Shabgahi, Nojan Sheybani, Aiden Tabrizi, and Farinaz Koushanfar. 2024. Livetune: Dynamic parameter tuning for feedback-driven optimization. *Preprint*, arXiv:2311.17279.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Evan Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Allyson Ettinger, and 23 others. 2025. 2 OLMo 2 furious (COLM's version). In *Second Conference on Language Modeling*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Transformers: State-of-the-art natural language processing.

In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45. Association for Computational Linguistics.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. Opt: Open pre-trained transformer language models. *Preprint*, arXiv:2205.01068.

Wentao Zhang, Yang Young Lu, and Yuntian Deng. 2025. Interactive training: Feedback-driven neural network optimization. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 851–861, Suzhou, China. Association for Computational Linguistics.

A Minimal Integration Example

The direct integration path requires a session, registered controls, and one call at a safe point in the loop:

```
# 1. Create the training session.
session = TrainingSession(
    goal="val_loss",
    agent=LLMAgent(every=100),
    memory="memory.jsonl",
)
# 2. Register application-owned controls.
session.register_knob(
    "lr", get=get_lr, set=set_lr,
    min=0.0, max=1e-2,
)

with session.run():
    for step, batch in enumerate(loader):
        loss = update_model(batch)
        # 3. Apply at a safe boundary.
        control = session.step(
            {"loss": float(loss)},
            step=step,
        )
        if control.stop:
            break

# HF: make_interactive(Trainer)
```

The artifact includes the complete 3842×1856 unannotated Aim capture used for Figure 1. It preserves the full journal, controls, metrics, and event-stream context at native resolution; the public workspace is at https://interactivetraining.ai/live.

B Control and Journal Records

Every submitted request contains an action type, arguments, identifier, timestamp, and source. Every result records success or failure, optional returned data, and an error message. Unknown actions and handler exceptions therefore appear as recorded failures. Events add an increasing sequence number and round number, allowing reconnecting clients to recover recent missed events; Aim keeps a permanent copy. One cross-round journal row records whether the round is the Round 0 no-LLM reference, its initial configuration and plan, best score and step, successful actions, reflection, and cumulative token usage. Prompts include at most ten recent rows.

`GET /state` returns the current settings, actions, and session metadata. `POST /actions` enqueues a request whose later `action_result` reports success or failure; GET/WS `/events?since=` returns events after a given sequence number. Built-ins cover settings, evaluation, checkpoints, execution control, module reset, annotations, context, and agent configuration. Only humans may invoke destructive actions or actions that change the agent's own configuration.

C Task Configurations and Controls

The journals provide round-level plans, actions, reflections, scores, and usage. Table 4 lists the model, data, budget, evaluation frequency, controller call frequency, and registered controls for each trace. RLVR denotes reinforcement learning with verifiable rewards.

D Released Evidence and Demonstration

Journal evidence. Table 5 reports execution and usage for each workflow. The SHA-256 manifest links each row to its source journal.

Trace shape. Figure 3 shows these patterns across all five workflows. BERT makes its largest score change in the first LLM-guided round, whereas Layerwise GPT continues setting new running bests through R8 and Muon–AdamW sets none after R5. Sentiment includes five later rounds that do not improve the current best before its R9–R10 recovery; Countdown's R1 jump is followed by four such rounds before R6–R7. The released plots retain these unsuccessful rounds rather than showing only the best configuration.

Execution volume. Across the five sessions, 47 LLM-guided rounds record 1,207 summarized successful actions, 3.23M/0.68M input/output tokens, and $36.54 at the configured GPT-5.5 prices. These fields quantify exposure and cost, not decision quality.

Live demonstration. The public site separates the five-round, 1,000-step-per-round Muon video ($5.44 \rightarrow 5.13$; https://youtu.be/SwEHORh6h0k), the distinct seed-42 11-round, 3,000-step-per-round paper trace, and a queued tiny-BERT CPU sandbox driven by a deterministic no-LLM policy. In the sandbox, reviewers can change a setting or request an evaluation and observe the matching `action_result`, privately hosted LLM demos add the plan/act/reflect path.

Workflow	Model and data	Budget	Evaluate / controller (steps)	Settings, actions, and diagnostics
Sentiment classification	BERT-base; 16k/1k IMDB (Maas et al., 2011) train/eval examples	1,000 steps	50/100	Schedule, AdamW LR/β_1/ϵ, clipping, evaluation, checkpoint, stop
Sentiment mixing	Tweet, finance, and product sources; 3k-example SST-5 (Socher et al., 2013) movie sentiment classification	2,000 steps	100/100	Three source weights, three class weights, LR
Layerwise GPT	24×512 Qwen3-style model; FineWeb-Edu (Lozhkov et al., 2024)	3,000 steps	100/300	27 embedding, block, head, and residual-group LRs; layerwise gradient norms
Muon–AdamW GPT	12×768 Qwen3-style model; FineWeb-Edu	3,000 steps	100/300	Muon and AdamW LRs, momentum, decay; gradient norm and weight RMS
RLVR Countdown	Qwen3-0.6B LoRA; 16×8 rollouts per iteration	100 iterations	10/10	LR, KL, temperature, clipping, entropy, difficulty

Table 4: **Setup details for the five recorded runs.** All recorded runs use GPT-5.5 with high reasoning effort as the LLM agent. The public Muon video is a separate five-round run with 1,000 steps per round. For RLVR Countdown, the evaluate and controller frequencies are in iterations. The released journals do not include hardware, wall-clock time, or per-step telemetry.

Workflow	Rounds	LLM rounds	New bests	Actions	Input/output tokens	Cost
BERT/IMDB	11	10	3	88	251.6k / 86.1k	$3.84
Sentiment mixing	11	10	5	399	905.5k / 185.7k	$10.10
Layerwise GPT	11	10	7	355	1191.9k / 200.8k	$11.98
Muon–AdamW GPT	11	10	4	137	449.8k / 138.4k	$6.40
RLVR Countdown	8	7	3	228	429.3k / 69.0k	$4.22
Total	52	47	22	1207	3228.2k / 680.0k	$36.54

Table 5: **Per-workflow journal and usage summary.** Token and cost totals come from the final row of each journal and are computed from unrounded values. "Actions" counts successful setting changes and other control actions, the count does not measure decision quality.

E Implementation and Reviewer Paths

The optional LLM client supports OpenAI's Responses API and compatible chat endpoints. Provider, model, reasoning effort, and controller call frequency are runtime settings; the recorded runs use GPT-5.5 with high reasoning effort. API keys are write-only and are redacted from state and built-in action events.

Zero install. At `https://interactivetraining.ai/live`, reviewers can inspect a recorded trace or submit a limited set of setting changes and evaluation requests to the queued CPU sandbox. The separate Muon video is supplementary.

Core smoke test. Tag `v2.0.2` installs without the Aim fork; `tests/run_tests.py` exercises the session, action protocol, HTTP interfaces, built-in integrations, journal, and scripted controllers without a GPU or provider key.

Full Aim workspace. To reproduce the full Aim interface, use the companion Aim branch and commit in `demo/aim.lock.json`. The main README gives the corresponding clone, `AIM_SRC`, and BERT frontend commands.