Assignment 1B: Learning Reflection

Name: Biao Wu **Date:** May 29, 2026 **Subject:** 42069 Research Methods — Autumn 2026

Introduction: Learning Intentions for Research Methods

My PhD looks at reinforcement learning (RL) for Multimodal Large Language Models in sequential decision tasks, mostly inside simulator and GUI environments. As I wrote in Assignment 1A, this first year isn't meant to produce research output. The actual model building, a custom interactive decision model on top of simulator APIs and GUI interfaces, is scheduled for year two. Year one is for getting the prerequisites in place: advanced RL, memory and generalization in large models, and distributed training.

So my goal this semester was deliberately practical: bring my RL knowledge up from the undergrad level (Q-learning, REINFORCE) to something usable on LLMs, get hands-on with distributed training and the experience-replay / external-memory infrastructure I'll need on the cluster, and do enough GUI and simulator work to know what I'm getting into next year.

The structure below loosely follows Gibbs' reflective cycle. Each section says what the activity was, what I took away, and what I'd do differently.

Checklist: Planned vs Actual Activities

Table 1. Planned activities from Assignment 1A vs what was actually completed this semester.

#	Planned (Assignment 1A) — hrs / cps	Actual — date / hrs / cps	Status
1	Safety & cluster rules workshop — 10h / 1.0	5 Mar 2026 — 10h / 1.0	Completed
2	Advanced RL II (audit) — 12h / 1.2	Mar–Apr 2026 — 12h / 1.2	Completed (centrepiece, see §2)
3	Memory Architectures III (audit) — 12h / 1.2	Mar–May 2026 — 12h / 1.2	Completed
4	GUI Environments & Simulator Design — 5h / 0.5	15–20 Apr 2026 — 8h / 0.8	Extended (+3h)
5	Distributed computing (DeepSpeed/vLLM) — 9h / 0.9	Apr–May 2026 — 10h / 1.0	Completed + colleague collab
6	Large-scale cluster industry workshop — 4h / 0.4	—	Deferred to next semester
7	Guided study: Sutton + literature — 5h / 0.5	—	Folded into §2 / §3 (pure reading insufficient per rubric)
8	Hands-on lab (cluster + GUI + logging) — 10h / 1.0	Throughout semester — 8h / 0.8	Partial (overlap with §5 / §6)

#	Planned (Assignment 1A) — hrs / cps	Actual — date / hrs / cps	Status
9	Research Writing Workshop — 4h / 0.4	—	Deferred to next semester
10	MIT xPro Systems Thinking — 5h / 0.5	—	Deferred to next semester
Total	**76h / 7.6 cps**	**60h / 6.0 cps**	**6 cps requirement met ✓**

Justifications for the deferred items and scope adjustments are given in the Reflection on Deviations section.

Section 1: Compulsory Safety and Cluster Rules Workshop

Date: 5 March 2026 | **Credit Points:** 1.0

Learning Objectives

To understand the safety rules, Slurm scheduling, and resource management I'd need in order to use our HPC cluster responsibly on shared infrastructure.

What I Learned

Before the workshop my mental model of the cluster was basically "submit an sbatch script, wait, hope it runs." The workshop gave me a much clearer picture of how Slurm's partition, QoS, and time-limit settings drive queue priority, the lifecycle and quota differences between `$SCRATCH`, `$HOME`, and shared `/project`, and why checkpointing, job arrays, and requeue matter once a job runs longer than a few hours.

The most useful change was how I request resources. I used to ask for the max GPUs and longest wall-time "just in case," not realising fair-share punishes overasking — it pushed me further back in the queue, not forward. After this I started profiling jobs before submitting them.

Learning Artifact

I put together a personal Slurm submission template plus a short cheatsheet: standard sbatch headers for single-GPU, multi-GPU, and multi-node jobs, auto-checkpoint and requeue snippets I keep needing, and the queue-state and historical-usage commands I always end up looking up. I now start every cluster experiment from this template, and have passed it on to two newer Masters students in the group.

Section 2: Advanced Reinforcement Learning II

Date: March–April 2026 | **Credit Points:** 1.2

Learning Objectives

To upgrade my RL knowledge from the undergraduate level (Q-learning, REINFORCE) to something usable on large language models, and to understand why the classical algorithms fail in the LLM setting and what people are actually using instead.

What I Learned — the biggest shift this semester

I came into this course with the two RL methods I knew best from undergrad: Q-learning and REINFORCE. I assumed I just needed to wrap them in some extra engineering and they'd carry over to LLMs.

It didn't take long to realise that was wrong. Both algorithms are basically out of date for LLMs. Q-learning needs a value estimate for every (state, action) pair, but at LLM scale the action space is the whole vocabulary; a Q-table is hopeless and a Q-network can't be trained reliably at that size. REINFORCE skips the value function, but its variance in long-sequence generation gets so high that training doesn't converge. PPO replaced both as the standard, but it needs a critic roughly the same size as the policy, and at 7B+ parameters that critic alone nearly doubles the GPU memory needed.

So my undergrad toolbox was already out of date for the problems I actually want to work on, and I had to start looking at things I hadn't even heard of a few months earlier.

Pivoting to GRPO (Group Relative Policy Optimization)

GRPO is the algorithm I ended up spending most of my time on. It was introduced in DeepSeekMath and then scaled up in DeepSeek-R1. The reason it works in this setting is that it gets around almost all of the issues above at once. For each prompt, instead of estimating an absolute value function, GRPO samples several rollouts in the same mini-batch and uses their rewards relative to each other as the advantage signal. No critic to train, no value function to fit, and the memory that critic would have taken is freed up. That's what makes 7B-scale RL realistic on the 8-GPU cluster we have.

Table 2 below summarises how the four algorithms compare on the dimensions that matter for the LLM setting.

Algorithm	Critic required?	Action space at LLM scale	Variance	Feasible at 7B+?
Q-learning	N/A (value-based)	Infeasible (vocab-sized)	Low	✗
REINFORCE	No	Feasible	Very high	✗
PPO	Yes, same size as policy	Feasible	Medium	Costly
GRPO	**No, group-relative**	**Feasible**	**Low–medium**	✓

To actually understand it rather than just have a vague picture, I worked through GRPO from a few angles. I read the GRPO sections in both DeepSeek papers and re-derived the objective from PPO on paper, which pushed me back through Sutton's actor-critic chapter to remember why we ever bothered with a critic. I then went through the GRPO trainer code in two open-source frameworks (verl and OpenRLHF) to see how group sampling, advantage normalisation, and the KL penalty are wired up in practice. Finally I ran a small GRPO experiment on a math reasoning task. Reward curves, KL, and how group size affects stability are easier to think about once you've watched them move on a real run.

Learning Artifact

- A short GRPO note with the derivation, going from the PPO objective to how GRPO replaces GAE with a group-relative advantage, plus a variance comparison.
- The comparison table above (Table 2), which I keep coming back to whenever someone in the group asks why we picked GRPO over PPO.

- The toy experiment notebook with reward, KL, and loss curves, plus the 30-minute walk-through I gave the other students in the course.

Reflection

The thing I'll take away here isn't really any single algorithm. It's that I need to stay more skeptical of what I already know. Going from Q-learning to GRPO showed me how fast RL moves and how little of my undergrad knowledge transfers cleanly to LLMs. Next semester I want to keep tracking what comes after GRPO (DAPO, ReMax, RLOO and so on) so I don't get caught flat-footed again.

Section 3: Memory Architectures and Generalization in Neural Networks III

Date: March–May 2026 | **Credit Points:** 1.2

Learning Objectives

To understand how explicit memory mechanisms address the long-sequence forgetting problem of Transformers, which is exactly the bottleneck I'd flagged in 1A as the core pain point of my PhD topic.

What I Learned

The course moved through a few angles on memory. On the architecture side we covered Differentiable Neural Computers (DNCs) and memory-augmented Transformers, and the trade-off between giving the model explicit read/write memory and relying entirely on attention. On the retrieval side, RAG and vector databases, and the part I cared most about: what happens to gradients when the retriever isn't fixed but is trained jointly with the policy. We also covered episodic memory in the RL sense, treating successful trajectories as a retrievable experience pool — basically the theoretical version of the experience-replay plus external-memory framework I'd committed to in 1A.

What actually shifted for me wasn't DNC itself. It was moving away from thinking of memory as a fixed lookup and toward thinking of it as a learnable component. That changes how I plan to design the year-two system: I want the retriever co-trained with the policy, not bolted on as a LangChain-style external piece.

Learning Artifact

- A short design note for a learnable retriever inside the RL loop, with a minimal diagram showing how the retriever's gradient interacts with the policy update.
- A small RAG demo: a library of a few hundred trajectories encoded with sentence-transformers, indexed, queried for top-k, and the results spliced into the prompt as in-context examples. It's a small end-to-end pipeline. I'll re-use it as the starting point for the year-two project.

Reflection

This course turned "memory" from an abstract research keyword into an object I can actually build with. Realistically though, what I have right now is the ability to implement existing methods. I can't yet show that any specific memory mechanism actually beats a plain long-context Transformer on my own task. That controlled comparison is on the list for next semester.

Section 4: GUI Environments and Simulator Design (extended)

Date: 15–20 April 2026 | **Credit Points:** 0.8

Learning Objectives

To learn how to build reproducible, controllable, low-noise GUI interaction environments for RL agents, and to close the gap between toy environments and real GUIs.

What I Learned

I'd assumed this would be a "take a screenshot, simulate a click" exercise. In reality the workshop took 1.6× the planned hours (8h instead of 5h) because the engineering details the lectures glossed over were where most of the time actually goes.

Reproducibility was the first surprise. Real GUIs aren't deterministic — network latency, render jitter, and pop-ups all leak into the observation stream — so getting an episode to actually repeat required a mock server in front of network calls plus a normalisation layer on screenshots. State representation was the second: raw screenshots are too high-dimensional and noisy for direct learning, so I started pulling semantic features from the DOM and accessibility tree before feeding them to the policy. Timing was the least obvious. Actions aren't instantaneous and pages render gradually, so without an explicit "wait until the GUI is stable" check the agent reads half-rendered states and acts on garbage.

Learning Artifact

- A small GUI sandbox wrapper built on Playwright, exposing a Gym-style `reset() / step(action) / observation()` interface with dual-channel observations (DOM snapshot + screenshot).
- A short lessons-learned note covering the difficulties above and the conventions I've adopted. It will act as a checklist for me when I build production-grade environments next year.

Reflection

The three-hour overrun was worth it. The edge cases that don't appear in the lecture notes are exactly where engineering time disappears. I've started budgeting GUI-related work at 1.5× whatever my first estimate is.

Section 5: Distributed Computing (DeepSpeed/vLLM) + Colleague Collaboration

Date: April–May 2026 | **Credit Points:** 1.0

Learning Objectives

To turn the 1A goal of "an early focus on a distributed framework for experience replay and external memory retrieval" into concrete technical knowledge: understanding the trade-offs across ZeRO, pipeline parallel, and tensor parallel, and learning how vLLM accelerates inference, which is particularly important for the GRPO rollout phase from Section 2.

What I Learned

A senior PhD colleague and I worked together to move an experiment that had been taking days on a single GPU onto an 8-GPU ZeRO setup. Higher ZeRO stages aren't automatically better. At 7B on 8 GPUs, we ended up on Stage 2 because Stage 3's all-gather frequency wasn't a win on our network — that had to be measured, not assumed. Mixed precision is also less plug-and-play than it sounds: naive fp16 produced loss spikes mid-training, so we used dynamic loss scaling plus selective fp32 master weights for the more sensitive layers. The thing I didn't appreciate at all before was how much vLLM's PagedAttention matters for the rollout phase — for an algorithm like GRPO that rolls out multiple samples per prompt online, vLLM raises throughput a lot compared to HuggingFace `generate`.

Learning Artifact

- A working DeepSpeed config and launch script for training a 7B model on our 8-GPU cluster, with the values we settled on for ZeRO Stage 2, bf16, and gradient checkpointing.
- A profiling report using PyTorch Profiler and Nsight that compares communication versus computation breakdowns. The first-pass configuration was spending a disproportionate share of time on all-reduce, and tuning the gradient accumulation steps brought that ratio down.

Reflection

The most counter-intuitive thing in distributed training is that a lot of what looks like an algorithm problem is actually a configuration problem. The same code with a different ZeRO stage or batch size can behave completely differently. That pushed me into a "profile first, tune second" habit instead of tuning from intuition.

Section 6: Hands-on Lab — Cluster, Vector DB, Simulator Integration

Date: throughout the semester | **Credit Points:** 0.8

Learning Objectives

To take the 1A threads (experience replay, external memory retrieval, multi-node GPU cluster) and stitch them together at the engineering level, not just on paper.

What I Learned

Most of what I learned here came from things going wrong. I started by stashing all trajectories in RAM, which was fine at toy scale and broke the moment we scaled up. Switching to Weaviate (and later comparing it against Milvus and Qdrant) forced indexing strategy, distance metric, and bulk-insert throughput into the conversation as actual decisions. Experiment tracking went the same way: moving from `print` to W&B became necessary once the group started running tens of experiments in parallel. I set up W&B alerts for loss spikes and gradient explosions, which has rescued a few runs that would otherwise have completed as silent garbage. The most painful lesson was distributed debugging: a run hung after roughly ten hours, the usual checks gave nothing, and I had to use py-spy plus NCCL debug logs to isolate a collective-communication deadlock that only appeared at a specific batch size.

Learning Artifact

- A personal MLOps template repository: the minimum skeleton of Slurm launch + DeepSpeed config + W&B integration + checkpoint recovery + alerting. Every later experiment I run forks this template as a starting point.
- A short vector-database selection note covering throughput, index build time, and API ergonomics across the three databases at our current scale.

Reflection

Engineering competence acts as a multiplier on research velocity. About 30% of my time this semester went into fixing tools rather than doing research, which sounds inefficient. But these tools should keep paying back once the year-two project starts.

Reflection on Deviations from the Original Plan

Execution diverged from 1A in a few places, each with a reason.

The standalone "Sutton + literature" reading (originally 0.5 cp) was folded into Sections 2 and 3. The 1B rubric is explicit that "reading articles and books ... are insufficient as activities to increasing your research abilities," so pure reading can't stand on its own credit. The value is still there (Sutton's actor-critic chapter underpins my GRPO understanding in Section 2, and a RAG survey underpins Section 3), but the credit isn't double-counted.

The Research Writing Workshop and the MIT xPro visualisation workshop were deferred to next semester. I worried briefly that I was avoiding hard skills, but on reflection it's a sequencing issue. I don't yet have a substantial piece of research to write up. A writing workshop after I have the artifacts from Sections 2 to 6 in hand will return more value than one taken before.

GUI Environments expanded from 5h to 8h while the Hands-on Lab contracted from 10h to 8h. The engineering time didn't match the lecture's estimate. GUI edge cases consumed more than I'd budgeted, and parts of the original Hands-on Lab (vector DB selection, profiling) had already happened inside Sections 5 and 6 and couldn't be counted twice.

Critical Appraisal and Future Learning Pathway

Capabilities Built

- Algorithmic: end-to-end understanding of modern LLM-era RL methods (GRPO and its family), from equations to source code to a toy run.
- Engineering: I can independently configure, profile, and debug a 7B-scale training experiment on an 8-GPU cluster.
- Methodological: I've started checking whether my own tools are still current rather than defaulting to what I learned as an undergrad.

Still Weak

- Theoretical depth. I can run GRPO but I can't prove why it converges or predict when it'll diverge. That caps my contribution at "apply and tune" rather than "design new algorithms".
- Scale. All my experiments are at 8-GPU / 7B level. At 100B+ qualitatively different bottlenecks appear (inter-node communication, fault tolerance, parameter sharding) that I haven't dealt with.
- Research-to-production distance. My GUI sandbox works under controlled conditions but hasn't been exposed to the chaos of real deployment.

Next Semester Action Plan

1. Theory catch-up: audit an optimisation or convergence-analysis course to fill in the hand-wavy parts of my understanding.
2. Actually start the year-two project. Per the 1A roadmap, build the first version of the custom interactive decision model on simulator APIs and GUI interfaces, threading together the toolchain from Sections 2 to 6.
3. Take the Research Writing Workshop. I now have real material to write up, so it should land much better than it would have earlier.
4. Larger-scale experiments. Apply for external collaboration or public compute to push GRPO training to at least 13B–32B and see what real distributed engineering problems appear.

Conclusion

Looking back, the biggest change isn't a new tool I picked up. It's that I've become noticeably more skeptical of what I already know. Going from treating Q-learning as a mature method to admitting it's out of date for LLMs, and then going to find GRPO, is a pattern I expect to repeat through the PhD. Next time it might be GRPO itself that gets superseded.

The 6 credit points didn't produce a finished research result, but they assembled the toolbox and working habits I'll need for real research in year two. That matches 1A's framing of the first year as groundwork rather than output. Next semester I want to use these foundations to actually build one thing, instead of continuing to learn new things in isolation.

Supervisor's signature: _____ Date: _____