A Note on Fruitful Research Directions

The purpose of the project is to give students a chance to:

- Bring yourselves to the cutting edge of research and development in this general area.
- Gain experience with using and modifying modern tools and environments for pretraining, post-training, and evaluation. (You don't have to modify or develop all three kinds of tools/environments, but you do need to modify or develop at least one.)
- Intelligently leverage the scale of compute that we have provided to you — and be able to justify the use of compute as well as demonstrate mastery in using it.
- Create concrete and impressive artifacts for your portfolio.

While you have considerable potential flexibility and agency in defining your project, we as the course staff will be involved in shaping the project's scope to make sure that there is both good coverage and non-overlap between all the projects in the class so that the learning experience is maximized for all students collectively. It's a small class and we expect that by the end, you should be able to talk not just in great detail about your own project but also at some level about your classmates' projects as well.

We broadly think about projects through two lenses: a **systems lens** (how we train, serve, and evaluate models efficiently and reliably) and a **model quality / validation lens** (how we design objectives, environments, and evaluation signals that actually move capabilities). But you are absolutely free to combine these in your projects. Some projects might be heavier on one side than the other, and that's ok too — and in practice, many of the most interesting projects sit at the boundary.

To help you see the kinds of ideas we are going for, we're going to list some here. We expect that some/many groups will use these as a jumping off point — and in fact we encourage this. You can also consider mixing-and-matching across them. But you don't have to do any of that. These are intended to get you thinking. (Note: if any of you have friends who are working in Industry, it is perfectly fair to ask them for suggestions. You can bring those to us. We might not agree, but we want to encourage you to bring ideas that we didn't think of but you're excited by.)

Model composition and model merging for improved capabilities

A very interesting direction is combining capabilities without full retraining — merging task-specialized checkpoints, composing "skill vectors," and exploring model combinations. The classical linear sequence perspective of training followed by potentially early stopping followed by fine tuning followed by potentially early stopping no longer reflects the richness of what people can and do do in practice.

Your group would of course start by looking at papers, blog posts, existing folk practices, etc. Starting with, for example, the classic papers on Model Soups (https://arxiv.org/abs/2203.05482) (arXiv), Editing Models with Task Arithmetic (https://arxiv.org/abs/2212.04089) (arXiv), and

TIES-Merging for interference handling (https://arxiv.org/abs/2306.01708) (arXiv); the early survey is also handy for mapping the space (https://arxiv.org/abs/2408.07666) (arXiv). There is also https://developer.nvidia.com/blog/an-introduction-to-model-merging-for-llms/ for practitioners.

But you can and should look more deeply. There is also a lot going on here in connection to MoE models, etc. As well as distillation as a data-oriented perspective on merging. (And recent approaches like RelayLLM (arxiv 2601.05167) that blur the distinction between inference and harness and post-training for model merging.)

Once you've done that, you can start thinking systematically about the existing systems support for doing and evaluating model merges. Treat "merge quality" as something you should be able to measure, reproduce, and iterate on — not something that only works for one magic recipe on one weekend. Think about whether pretraining, midtraining, and/or post-training is the best place for support for model merging, and what that might require (in terms of checkpoint management, tooling, evaluation harness integration, and automation of merge sweeps).

Furthermore, think about how the ideas behind model merging could be supported efficiently at inference if we have concurrent users who might be using different mixtures. What is the serving abstraction here? What needs to change if "the model" is not a single fixed artifact but a family of composed artifacts? How can speculative decoding be properly supported in that world?

Now, formulate some tentative goals and explorations. And then iterate and refine until a project idea comes more crisply into view. It is believed that much of the understanding of model merging is a "black art" that is held closely by the major Frontier Model developers — which is exactly why turning it into systematic engineering plus careful measurement can be so valuable.

Novel architectures and emerging optimizers that win in wall-clock

Keep architecture/optimizer work tightly tied to measurable wall-clock metrics (time/cost to reach a target score) and deployability (serving footprint, decoding speed, comms/memory patterns), with an emphasis on co-design with infrastructure. Ensure to also show that the new architectures you are designing can still retain model quality — "faster" only matters if it is not secretly worse.

Canonical starting points include long-context hybrids like Mamba (https://arxiv.org/abs/2312.00752), Gated Delta Net, etc. On the optimizer side, there is a growing ecosystem of emerging optimizers and training recipes (e.g., Sophia (https://arxiv.org/abs/2305.14342), Lion (https://arxiv.org/abs/2302.06675), as well as newer practitioner-driven directions like Muon, SOAP, etc.). Existing work says that the best hyperparameters needed for each of these can be different, and there is work saying that muP-type scaling isn't exactly correct for state-space models. Getting the mixed-precision right is a part of the challenge here for maximum performance.

A key question that often gets skipped but really matters: do these considerations change for RL vs pretraining? Optimizers and scaling rules that are “obviously good” for (self-)supervised learning may behave very differently when the objective is an RL post-training signal with on-policy data, reward-model noise, or an online distribution shift.

One way to make this direction crisp is to pick a small set of architectural variants and a small set of optimizers, and then be ruthless about what you measure: not just final benchmark numbers, but the full story of stability, sensitivity to hyperparameters, throughput, memory use, and how quickly you can get to a target score in practice.

Note: here, there is a very easy path for impact.

LLM RL post-training: scaling laws, reliability, and algorithmic breadth

While there has been research into scaling laws for LLM pre-training that gives us some insight into how to trade off model size and compute, how can we have scaling laws for RL post-training? How much data should one collect for RL training given a model size? If you have a fixed data collection budget, what model size is optimal? Some starting points would be recent work on non-LLM RL scaling (https://arxiv.org/abs/2508.14881) and LLM RL scaling (https://compute-optimal-rl-llm-scaling.github.io/).

But in practice, “RL scaling” is not just an abstract curve-fitting exercise. It forces you to answer what the right balance of RL, and SFT is — and what your post-training mix should look like when you account for cost, reliability, and deployment constraints.

Along the way, there are multiple systems issues as well as algorithmic issues to engage with:

- **RL reliability and reproducibility: closing inference/training divergence.** Closing inference engine/training backend logit divergence on popular frameworks without sacrificing too much performance is not just a “bugfix” — it determines whether your entire post-training loop is stable and trustworthy. (See https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/ and think about how to bring these ideas into the toolchains that we are using.)
- **On-policy distillation as a first-class primitive.** Supporting on-policy distillation (https://thinkingmachines.ai/blog/on-policy-distillation/) efficiently when we have access to strong teacher models — and in particular, how to do this when the “strong teacher” is actually an ensemble of some sort including best-of-N but also including ideas like debate, etc. Treat this as both a modeling question and a pipeline design question: if you can’t run the ablations, you can’t learn the rule.
- **Going beyond the current on-policy default: off-policy RL for LLMs.** While on-policy RL (e.g., PPO-style approaches, GRPO-style approaches, and relatives) has been hugely successful in LLM post-training, off-policy RL methods (e.g., DDPG, SAC) have been largely unused by the community. These off-policy methods have achieved state-of-the-art performance in robotics simulation benchmarks, and can lead to better sample complexity because they can re-use “stale” data. What breaks when you try to

apply them to LLMs? What are the real bottlenecks — credit assignment, reward noise, exploration, replay bias, distribution drift, infrastructure, or something else? Understanding how to use off-policy RL methods (or adaptations thereof) in the LLM setting is a deep and fertile direction.

There is also a closely related sub-direction that deserves explicit attention: **RL with emerging optimizers.** There has been work on new optimizers (see the architecture/optimizer discussion above) that work better than Adam for (self-)supervised training. However, this is less explored for RL. Do these new optimizers also work better for RL training? Since RL training follows a fundamentally different objective, what properties should RL optimizers have? Even a careful negative result here can be very informative, because it forces clarity about what is actually hard about RL optimization in practice.

End-to-end efficiency for training and serving: kernels, scheduling, and toolchains

A coherent way to think about “efficiency” is not as a grab-bag of micro-optimizations, but as an end-to-end effort to move real service and training metrics: throughput, cost to reach a target quality, and tail latency under realistic workloads. This direction can be tackled at multiple layers, and the best projects often connect at least two layers.

At the kernel/toolchain level, with the advent of DSLs like Thunderkittens and CuteDSL, it is now realistically possible to optimize performance for novel architectures rather than being locked into whatever kernels happen to exist. Representative starting points include FlashAttention and follow-ons — but you should also treat the broader transformer block as a target, especially where models are bottlenecked by memory movement (attention/KV-cache, normalization, activation, and common linear patterns). One pragmatic strategy that often pays off is to focus on a single widely used stack (e.g., PyTorch + Triton, or CUDA/CUTLASS) and repeatedly land improvements where those bottlenecks occur so that small wins compound.

At the inference/systems level, don’t treat serving as “just batching.” In addition to kernel improvements, consider scheduling as LLM-specific co-design: batching policy, KV-cache memory policy, and decoding algorithm choice (vanilla vs speculative/multi-token), with a focused optimization around service metrics like p99 time-to-first-token (TTFT) and p99 tokens/s/request under realistic prompt-length and concurrency traces. The point is to optimize the system you actually run, not the system you wish you ran.

On the architecture side, aside from the hybrid architectures (state-space + attention) deployed at scale by next-gen models from NVIDIA, Qwen, Falcon, Granite, and others, there are even more variants being explored in the literature including recent approaches like MHLA (arxiv:2601.07832) and Deep Delta Learning that reinterpret the residual block. There are also attention variants like Differential Attention and Gated Attention. There are also modifications like metacontroller steering (arxiv 2512.20605) that leverage interpretability ideas and are intended to be deployed during post-training RL.

In this vein, there are two recent papers from DeepSeek that are particularly intriguing and should reward careful implementation, optimization, and exploration: “Conditional Memory via Scalable Lookup: A new axis of sparsity for large language models” (arxiv:2601.07372) and mHC: Manifold-Constrained Hyper-Connections (https://arxiv.org/abs/2512.24880).

Here, there is also a very clear path towards impact.

Better “Phantoms” for LLM training: training proxies and distilled mini-models

In areas like imaging, there is the concept of “Phantoms” — namely standard synthetic objects that stand in for the human body when one needs to rapidly evaluate and iterate on experiments and architectures. It’s an open question whether it is possible to assemble together an ensemble of toy problems — either by viewing these as separately trained models or combining different toys together for a single trained model or a combination of both approaches — that can reliably allow the exploration of hyperparameters, optimizers, and training strategies in a way that is far cheaper than training an LLM from scratch.

The lack of such Phantoms makes people conservative in adopting novel ideas when millions of dollars are at risk. Whereas muP suggests training a smaller proxy model on the same data to get the learning rate right, the idea here would be to use an ensemble of proxies that could get many more hyperparameters and architectural questions approximately right.

The idea of “training proxies” (aka Phantoms) spans a continuum from toy models like least-squares and formal languages and subsets of mathematics to very small language models themselves. For very small language models, one of the most exciting/promising developments is rapid distillation from a larger model. (Important for edge deployments, but we can explore a potentially different use.) Often, larger models can train faster in a data-efficiency sense as compared to smaller models. But having access to a (partially) trained language model is more than an input-output black box — we have not only the final logits, but also all the intermediate activations.

Those activations (the approach is called layer-wise distillation) can represent a much richer source of supervision and this is an active area of research. The big question is whether such distillation can be made cheap enough and effective enough in providing a smaller proxy that will allow learning hyperparameters to be adapted during a long training run.

This is an example of a higher-risk/higher-reward family of projects that still have a clear path to impact, even if the results are negative.

RL environment scaling: building better worlds and better signals

A distinct and very compelling direction is to push on the environment side of RL: design new RL environments that models can explore in, including for tasks that are long-horizon and

difficult to solve. The goal is not simply to create something that “works,” but to create something that produces a high-quality learning signal and a validation story you actually believe.

A strong project here should show clear training reward improvement and validation reward improvement in your crafted environment, and then connect that to external benchmark improvements (or at least to robust transfer into a meaningful downstream capability). In other words, the environment should not be a sealed box where progress is only visible inside the box.

Ideally, environments should be oriented toward tasks with real economic value — not because you need a business plan, but because “economic value” tends to be a forcing function for realism: long horizons, hard constraints, tool use, partial observability, and outcomes that are difficult to game. In practice, this direction often benefits from tight integration with systems work (throughput, parallel rollouts, logging, scalable evaluation) and with validation work (how do you know your environment is eliciting the behavior you care about rather than a shortcut?).

Evaluation and benchmark construction: validation signal you can trust

Another direction that should be treated as first-class is evaluation. Similar to RL environment curation, the goal is to push benchmarks toward end-to-end work: multi-step tasks, tool use, long-context retrieval, and hard constraints like latency/cost. The focus should be on benchmarks that are hard to game, and on judging methodology as a way of enhancing validation signals.

One important meta-point: evaluation often doesn’t consume enough GPUs relative to training — and that can be a mistake. It can be worth scaling eval ablations and prioritizing better curated eval data (coverage, gold signals, adversarial cases) as the main thing.

For both evaluation and RL environment construction, complexity and the quality of the signal are the key. If the signal is shallow, the entire project becomes fragile. If the signal is rich and reliable, small model-side changes become legible and you can iterate much faster.

This is a tricky one where it really needs students to (a) have access to some insight into tasks that correspond to actual value (economic or otherwise); (b) figure out how the compute requirement for the environment can be handled with what you have access to. Consequently, students wanting to move in this direction need to be able to make their case very clearly.