CSE 291A/DSC 291: Deep Learning Systems, Spring 2026

Guest Lecture: Building Open Frontier Models End-to-End

Lecturer: Zhengzhong (Hector) Liu

Scribe: Eunbeen Jung, Aleksa Popovic, Ksheer Sagar Agrawal, Jaewon Han, Eyaad Mir, Abhishek Dhaka

1 Introduction

This lecture was delivered by Zhengzhong Liu (also known as Hector), Director of the Institute of Foundation Models (IFM) SV Lab in Sunnyvale, affiliated with the Mohamed bin Zayed University of Artificial Intelligence (MBZUAI) in Abu Dhabi. The lab is approximately one year old and focuses on open foundation model research, having shipped several model series including K2 (65B strong open base, Dec 2024), K2-Think (32B reasoning post-train, Apr 2025), Jais (best open Arabic LLM, Aug 2023), and PAN (foundation world model, Nov 2025).

The central theme of the lecture is an **end-to-end overview of the full pipeline for training a frontier large language model (LLM)**, from raw data collection through pre-training, post-training, and deployment. The speaker emphasized that this pipeline is not only long but increasingly complex, and that many of the practical difficulties lie in seemingly mundane engineering details rather than novel algorithmic ideas.

1.1 What Does "Open" Mean for Foundation Models?

A key motivation for the lab is radical openness. There is an important distinction between three levels of openness:

- **Closed models** (e.g., GPT-4, Claude, Gemini): Only an API is exposed. No weights, no data, no recipe.
- **Open-weights models** (e.g., Llama 3, Mistral, Qwen): Weights are released and the model is fine-tunable, but training data and code stay private.
- **360° Open / open-source models** (e.g., LLM360, Pythia, IFM K2): Weights + data + training code are all released. Full traceability: every checkpoint, data shard, and training log—reproducible end-to-end.

The IFM Lab follows the third philosophy. Only the rightmost level lets you debug what the model learned and why. Intermediate checkpoints and training logs are typically kept secret at commercial labs, but IFM releases them so the community can study model behavior at any point during training.

These intermediate checkpoints and training logs are particularly valuable for academic research because they allow the community to study what the model looks like at, say, 20% or 40% of training — under a specific learning rate or data mix — information that is normally closely guarded at commercial labs. For someone who only cares about the final model this may not matter, but for research reproducibility and understanding model behavior during training, it makes a significant difference

2 The Full Training Pipeline

The end-to-end pipeline consists of the following stages:

1. **Data**: Source, filter, license, and synthesize training data.
2. **Pre-training**: The big run; most of the compute budget.
3. **Mid-training**: Long-context extension and code repair.
4. **SFT** (Supervised Fine-Tuning): Supervised behavior shaping.
5. **RL** (Reinforcement Learning): Preference and verifier feedback.
6. **Evaluation**: A continuous signal throughout training, not a final exam.
7. **Deployment**: Serve and monitor; triggers follow-up training.

Each stage can take several months of effort, and decisions made early (e.g., model architecture, data mix, parallelism strategy) deeply constrain all subsequent stages.

3 Data Creation and Curation

Data is the foundation of everything. With a standard architecture and a high-quality dataset, one is very likely to obtain a good model. Despite appearing tedious, data work is arguably the most critical component of the entire pipeline. Before training, you choose what "the world looks like" to the model. The five key decisions are: (1) **sourcing**—what gets in; (2) **filtering**—quality, dedup, PII, contamination; (3) **balancing**—domains, languages, recency, measured by token count not file count; (4) **synthesis**—generating data when the source distribution falls short; and (5) **licensing**—what you can train on and what you can release. Each is a research problem on its own.

Licensing raises two distinct questions that are easy to conflate: whether you can legally train on a given dataset, and whether you can publicly release it afterward. Sometimes a license permits training but is restrictive enough that you cannot release the data even if training itself was allowed — a critical distinction for a lab committed to full openness.

3.1 Data Composition and Domain Mix

The primary source of pre-training data is large web crawls (e.g., Common Crawl, C4), which typically dominate the token budget. Additional domains include code (GitHub, Stack Overflow), books, scientific papers (arXiv, PubMed), Wikipedia, math, conversations (forums, Reddit, Q&A), and LLM-generated synthetic data. Each domain contributes a different kind of capability to the model.

Determining the optimal domain mixing ratio is a challenging bi-level optimization problem. Formally, given a dataset $D = \{d_1, \ldots, d_N\}$, training finds:

$$\arg\min_{\theta} \mathbb{E}_{d \sim p_\tau(D)}\, \ell(\mathrm{LLM}_\theta,\, d), \quad \text{where} \quad p_\tau(D) = \sum_n w_\tau(d_n)\, \delta_{d_n}$$

The data mixing problem is then to find the token weights τ that maximize downstream evaluation:

$$\arg\max_{\tau}\ \mathrm{Evals}(\mathrm{LLM}_\theta) \quad \text{s.t.} \quad \theta = \arg\min_{\theta} \mathbb{E}_{d \sim p_\tau(D)}\, \ell(\mathrm{LLM}_\theta,\, d)$$

Because $\text{Evals}(\text{LLM}_\theta)$ is expensive to compute, the practical approach is to learn a cheap surrogate f_η such that:

$$f_\eta(\tau) \approx \text{Evals}(\text{LLM}_\theta)$$

The inner problem is training the LLM; the outer problem is finding weights τ; the key challenge is defining a reliable, cheap proxy for the evaluation objective.

In practice, this optimization is very hard because:

- There is no single objective—improving math and code can hurt creative writing ability.
- The optimal mix is sensitive to model size and even the specific model architecture.
- Automated weight-optimization experiments spent **375K GPU-hours** and produced only marginal gains that did not reliably transfer across scales.
- Evaluation benchmarks themselves are noisy, so the outer-loop signal is unreliable.

A concrete illustration is that the best mix at 7B/1T tokens (mix-2) was not the best mix at 1.5B/0.6T tokens (mix-3). MMLU rankings appeared stable across two scales, but BBH rankings flipped, which likely reflects MMLU's tendency to grow monotonically with scale and data, making it a useful indicator of whether one is scaling correctly but a poor signal for capability-specific decisions. Other capabilities such as reasoning and coding require targeted data in the right amounts.

As a result, the team's current approach ("vibe-mixing") relies primarily on human expertise combined with controlled small-scale experiments. An important practical finding is that **data diversity** consistently yields more reliable gains than elaborate data weighting methods, and the benefits are less sensitive to model scale:

Data Mixing	**Data Diversity**
Many nuances. Small gains.	Steady. Reliable.
Many knobs to tune	Broaden sources first
Computationally expensive	Gains compound across capabilities
Optimum shifts with data and model size	Less sensitive to model size
Evals are noisy	Cheaper to validate

Dr. Liu coined their current approach "vibe mixing" — a combination of team intuition and targeted small-scale experiments — and was candid that none of the principled automated methods they tried gave them sufficient confidence to fully automate this process. Data diversity, by contrast, has a simpler knob (broaden sources and content types) and produces more reliable returns without requiring the same level of tuning.

Rather than treating the entire web as one domain, the team uses **WebOrganizer** to segment the web into dozens of fine-grained subdomains, constraining each weight to fall within reasonable bounds and progressively eliminating poor-performing candidates in a tournament-style search.

3.2 Proxy Metrics for Evaluating Pre-Training Data

Choosing the right evaluation signal is critical for the data-selection outer loop. Some lessons learned:

- **MMLU** used to be considered a strong indicator of base model quality, but research from APTBench found only a weak correlation (Pearson $r = 0.38$, $p = 0.22$) between base-model MMLU and post-trained model reasoning ability. Pick base models on data, not on MMLU.

- **Pass@K** is now used as a proxy: sample a base model K times on a problem. Models that have *any* chance of solving it tend to be RL high-performers; models that *never* solve it tend to be RL low-performers. This provides a pre-training-time signal that correlates with post-training outcome—without running RL.
- **SFT in pre-training**: train with chat-formatted SFT data on intermediate checkpoints and measure pass rates. This elicits early signals that correlate with post-training outcomes without running full RL.

3.3 Synthetic Data for Diversity

For a research lab without access to a customer base, synthetic data is essential for increasing diversity. Simple synthetic generation suffers from mode collapse—the generated distribution becomes increasingly repetitive over time, as measured by rising compression ratios. Cascading approaches (curriculum learning, ICL, etc.) do not scale well either.

The key idea: **ground synthetic data in real data.** The pipeline is:

Index pre-training data $\rightarrow$ retrieve relevant context for seed query $\rightarrow$ add randomness to prompt $\rightarrow$ generate

Diversity inherits from the source distribution, not from the generator's priors. The team released this as **TxT360-MIDAS** on Hugging Face (`huggingface.co/datasets/LLM360/TxT360-Midas`), containing over 2 million synthetic documents each grounded in a real corpus chunk.

This approach maintains a much healthier compression ratio compared to pure synthetic generation and scales more gracefully because the external documents anchor generation in novel contexts.

4 Pre-Training Systems

The pre-training system appears straightforward in principle—load data, run distributed training—but many practical problems arise from seemingly simple components.

4.1 Data Loading: Packing Strategies

Pre-training batches contain a large number of tokens, and individual documents are short, so multiple documents must be packed into a single sequence. There are four common strategies, each with different trade-offs:

- **Naive doc-level** (one doc per slot): Long docs are over-represented; short docs dominate by count.
- **Token-budget batching** (pack until token budget is hit): Effective batch size in samples drifts over time.
- **Random truncation + concat** (cut mid-doc, concatenate): Cheap, but breaks long-range document structure.
- **Length-bucketed** (group similar-length docs, sample per bucket): Boundary choice becomes a hyperparameter.

If packing is done with insufficient randomness (e.g., always grouping similar-length documents), the non-i.i.d. ordering introduces spurious patterns. The team observed training loss fluctuations caused by such packing artifacts and had to revise their randomization algorithm multiple times mid-training.

4.2 Tokenization: Details Have Outsize Effects

A basic sanity check is: tokenize a document, decode back to text, and re-tokenize. If the token sequence changes, the tokenizer has a consistency bug. This was found in almost all publicly released tokenizers. Key design choices that cause real bugs:

- **BOS/EOS placement**: System prompts prepend BOS at training but omit at eval. Same model, different scores. The IFM Lab retains BOS to support unconditional generation; most modern models use EOS-only.
- **Chat-template drift**: Llama 3, Llama 3.1, Qwen each use different role markers. A single character mismatch between training and inference can degrade reasoning by 5+ points.
- **Tokenizer mismatch**: Pre-train tokenizer $\neq$ post-train tokenizer. New special tokens (e.g., `<think>`, tool-call tokens) get random embeddings; old special tokens may now retrain from noise.
- **Whitespace handling**: BPE merges depend on adjacent whitespace. `"hello"` vs. `" hello"` tokenize differently. Code data is especially sensitive.

An illustrative anecdote: the team registered `<ThinkHard>` as a special token to trigger long chain-of-thought. An engineer later typed the plain phrase "think extra hard" (not a registered token). The tokenizer decomposed it into `<think>`, `extra`, and `hard`, and began exhibiting extended reasoning behavior—semantic meaning emerged from sub-token composition without any special-token registration.

4.3 Distributed Training and Parallelism

There is no single parallelism strategy that is universally optimal; the right choice depends on model architecture, size, and cluster hardware. The four main strategies are:

- **FSDP** (Fully Sharded Data Parallel): Compute-optimal; fewest duplicate compute, fewest recompute. Communication grows quasi-quadratically with sharding count—past a knee in GPU count, all-gather and reduce-scatter dominate step time.
- **TP** (Tensor Parallel): Splits each layer's matrix multiplications across ranks. Local, low compute cost. Heavy intra-layer communications.
- **PP** (Pipeline Parallel): Splits the model across stages. Light communications but introduces bubble loss and activation checkpointing pain. The team could not make PP work reliably in their paradigm.
- **EP** (Expert Parallel): Distributes MoE experts. Light communications when routing balances; pathological when it doesn't.

The fix for FSDP's communication bottleneck is to layer in TP and PP to localize communications, co-designed with the cluster topology. One practical layout discussed was to colocate tensor parallelism and expert parallelism on the same rank when possible, place expert parallelism across data-parallel ranks,

and keep data parallelism as the outermost dimension, although the speaker emphasized that this is not a universal recipe. There is no single setting that scales forever—the right combination shifts as the cluster grows. These decisions must be locked in before training begins, as a 3–6 month run cannot be reorganized mid-way.

Hardware matters. DeepSeek adopted complex pipeline parallelism partly because their custom accelerators (with weaker interconnects) necessitated it. The IFM Lab, having stronger machines but weaker networking, takes a different approach closer to PyTorch FSDP/TorchTitan.

4.4 Kernel Optimization with Coding Agents

The team uses coding agents to profile, benchmark, and optimize training kernels. Their workflow:

1. Agents benchmark every line of training code in isolation: 10 warmups, 100 iterations, full sweep over batch size, sequence length, and 7B / 29B / 375B / 1T model configs.
2. This takes 5 hours sequentially but only **20 minutes on 32 GPUs in parallel**.
3. Output: per-line forward-pass time map, cross-referenced to source location.
4. Inspect compute graphs (solid edges = sequential; dotted = conditional; independent branches → CUDA stream candidates) to identify fusion candidates.

One optimization on a limiting operator (fused into a custom kernel) yielded a **∼8× speedup** on that hotspot op, resulting in a **∼30% end-to-end forward+backward speedup** for the full model.

5 Post-Training in the Agentic Era

Post-training encompasses SFT followed by reinforcement learning (RL). The core insight: post-training in the agentic era is still about scaling and data diversity—but the system around training is far more complex.

What's the Same	What's Different
Gradient descent on a loss	The system around training
Data and scale do the work	Inference, sandboxes, reward functions, judges—all coupled to the loop
Diverse data beats clever loss functions	The model produces the data it trains on

5.1 System Components

The post-training loop involves multiple interacting distributed systems. The loop runs as fast as its slowest component:

- **Inference engine** (sglang, vLLM, TRT-LLM): Generates rollouts; must be accurate and efficient.
- **Sandboxes**: Safely executes model-generated code and tool calls in an isolated environment.
- **Reward functions**: Verifier- and rule-based for math/code; judge LLMs for open-ended tasks.

- **Replay buffers**: On-policy or off-policy storage of past rollouts.
- **Eval harness**: On-policy + held-out evaluation; modern models call tools rather than just predict tokens.
- **Judge LLMs**: Preference rating for tasks without ground-truth verifiers.

5.2 Stability: Pre-Training vs. Post-Training

Pre-training: More Stable Now	**Post-training (RL): Not Stable**
Stationary objective	Model produces the data it trains on
Gradient noise dominates randomness	Distribution shifts on every step
Failures are usually infra failures	Reward hacking, judge/sandbox/latency cascades

Teams routinely give up MFU to gain stability—sync RL or 1-step async beats fully async in practice. Reward hacking can occur not just in the model but in the evaluation harness itself: a widely used agentic harness (*Forge Code*) was found to have been optimized via its own RL loop, causing it to inadvertently hint the model toward correct benchmark answers—reward hacking at the framework level.

On closer inspection, the harness had been developed with an RL loop in which both the harness and the model were being co-optimized simultaneously. As a result, the harness had inadvertently learned to provide hints to the model pointing toward correct benchmark answers — effectively hacking the very benchmarks it was supposed to evaluate. The harness authors almost certainly did not intend this; it was an artifact of the RL optimization loop. The broader lesson is that reward hacking can happen anywhere in the system stack, not just inside the model itself.

5.3 Scaling Laws for RL Post-Training (ISO-Compute Playbook)

Unlike pre-training, RL post-training lacks a clean Chinchilla-style scaling law. The fundamental reason: the data distribution depends on the policy being optimized. There is no fixed dataset; exploration and optimization are mutually dependent (exploration gives data for optimization; optimization affects exploration).

In collaboration with researchers from UCSD and CMU, the team developed an *iso-compute playbook* for RL. Key findings from math reasoning tasks:

Finding 1: More Compute → More Parallel Rollouts. The optimal number of parallel rollouts per problem (n) grows with compute budget—on both easy and hard problems. High compute → increase n aggressively (up to 512 on frontier); low compute → fewer rollouts.

Finding 2: Same Trend, Different Mechanisms.

- *Easy problems*: More rollouts sharpen probability mass on correct tokens (sharpening). Diagnosed with worst@K.
- *Hard problems*: More rollouts increase coverage—finding successful traces at all. Diagnosed with best@K.

For easy problems, higher rollout counts sharpen the model's probability concentration on correct tokens — helping it converge quickly on what already works. For hard problems, higher rollout counts increase

coverage, raising the chance that at least one rollout succeeds and provides a valid reward signal to train on. These are mechanistically different phenomena, but both lead to the same practical recommendation: allocate more rollouts as compute scales up.

Finding 3: Optimal n Follows a Predictable Sigmoid.

- **Stop-sequence handling**: Generation stops "too early" or "too late." Pass rate looks worse or better than reality. Differs across harnesses for the same model. The optimal rollout count n^* as a function of compute C follows:

$$\log_2(n^*) = \frac{L}{1 + \exp\big(-k(\log C - \log C_0)\big)} + b$$

  A sigmoid you can fit means you can plan: at any compute point, the optimal n is predictable from earlier runs.

Finding 4: B_p Barely Matters—n Is the Lever. B_p (unique problems per batch) has a smaller effect than expected. For a fixed n, varying B_p yields negligible change in validation reward. Practical implication:

- Low compute → Large B_p, Small n
- High compute → Large n, Small B_p
- Total iterations: $M = C/(n \times B_p)$

5.4 Practical RL Recipe

The "healthy recipe" differs between easy and hard problem regimes:

	KL Loss	**Entropy**	**Zero-Var**	**LR Scale**
Easy (reward ≈ 0.3–0.6)	Yes	Yes	No	Sqrt
Hard (reward ≈ 0–0.06)	No	No	No	Sqrt

The practical five-step workflow:

1. **Evaluate Difficulty**: Measure base model avg@16 to classify problems.
2. **Pick Recipe**: Easy → KL + entropy regularization; Hard → neither.
3. **Tune n First**: Scale n with compute C via the sigmoid fit.
4. **Set B_p**: Keep moderate (32–1024).
5. **Derive M**: $M = C/(n \times B_p)$.

5.5 Towards Agentic Post-Training

The ISO-compute results were obtained on mathematical reasoning. The team is now extending beyond math and beyond GRPO:

- **Cross-domain reasoning**: expanding RL training data to language, science, planning, and agents.

- **Async RL infrastructure**: decouple generation from training. Generation runs continuously while training catches up—up to **2.77× speedup** demonstrated in recent work.
- **Multi-turn, tool-use, long-horizon**: agentic capabilities pull tool use and multi-turn reasoning into the loop; reward design and rollout cost both grow significantly.

For long-horizon tasks, the team's current approach to off-policy drift is cross-stage distillation: train specialized models for different task types and distill back into one model, rather than implementing complex policy-staleness correction.

6 Evaluation

The team treats evaluation as a continuous cycle, not a final exam.

Reward hacking is the central pattern. The proxy reward (training signal) goes up while the true eval (held-out) goes flat or down. Running held-out evals continuously is the only way to detect this early.

Benchmarks are a scarce resource. A good benchmark takes years to design, validate, and stabilize. Optimizing directly to the benchmark score ("benchmaxxing") destroys the very signal the benchmark was built to give. For example, MMLU is a good benchmark for pre-training below an 80-score (80%) ceiling, but once models approach saturation it loses diagnostic value. Dr. Liu noted that neither OpenAI nor Anthropic appears particularly aggressive about optimizing scores on every public benchmark, which he takes as implicit validation of this philosophy among frontier labs.

Where evals quietly break—four failure modes:

- **Tokenizer mismatch**: Eval uses a slightly different tokenizer. Scores diverge from training scores. Looks like a model regression—isn't.
- **Chat-template drift**: System prompt or BOS/EOS differs between training and eval harness. Same model, different number on the leaderboard.
- **Contamination**: Eval set leaked into training. Looks great. Generalizes badly. Reproducibility via open data is the only real defense.

Two evals run at the same time should be byte-for-byte identical to the training stack. Train-eval parity is the cheapest debugging in the whole pipeline.

7 Productizing the Pipeline

The fifth and final section covered the lab's longer-term vision: moving from a production pipeline to a *factory*.

7.1 Continuous vs. Discrete Training

Current practice (discrete training) follows a "plan, train, ship, repeat" cycle: each release is a new run, with a long lull between deploys (6+ months between user-visible improvements), and every stage redone from scratch.

The target (continuous training) is "always-on":

- New data shards continuously rebuild the pipeline.
- Releases become a stream, not a milestone.
- Eval is the gate, not the calendar.
- Failure modes show up earlier and smaller.

7.2 Stage Reduction

Mid-training and SFT are nearly combined already (both use supervised objectives on similar data). The remaining challenges to merging them fully are: (1) variable-length training support, and (2) understanding downstream needs before choosing data. The ultimate research goal is **Reinforcement Pre-Training**—applying RL signals from the very beginning of training, combining all four stages (Pre-train, Mid-train, SFT, RL) into one continuous process.

IFM currently implements this as SFT-in-pretraining: chat-formatted data is added to the pre-training mix with a proper loss mask, allowing the model to continue in a pre training style while absorbing instruction-following signal simultaneously. Merging mid training and pre-training is the next step, which requires variable-length training support so the model does not wastefully over-compute on short sequences while also learning to handle the long context lengths that mid-training normally introduces.

7.3 Self-Evolve Pipelines

The long-term vision: the current generation of models trains the next generation. The kernel analysis agent (which profiles code and proposes optimizations) is a precursor. The goal is a model that can analyze training runs end-to-end—identify bottlenecks, generate improved recipes, and fix bugs automatically—across all phases. Combined with continuous training, this points toward a fully self-improving training loop for LLMs.

All models, data, code, and training logs are openly available at `huggingface.co/LLM360`, `github.com/LLM360`, `wandb.ai/llm360`, and `ifm.ai`.

8 Q&A Highlights

MoE and creative writing. In a MoE architecture, does one expert specialize in creative writing? The speaker noted this is an interpretability question the team has not yet studied formally. More likely, the creative writing advantage comes from capacity allocation: IFM did not deprioritize creative writing data, while other labs allocated more capacity to coding and agentic tasks. The speaker also noted that the open-source community provides useful qualitative feedback on capabilities such as creative writing, which can reveal regressions that may not be captured by standard coding or reasoning benchmarks.

Arabic tokenization challenges. Key issues: (1) vocabulary trade-off between languages—larger vocab helps but is finite; (2) fidelity score (tokens per word) must be balanced across Arabic and English; (3) Standard Modern Arabic (MSA) has ample data but dialects (Egyptian, Gulf, Levantine) are underrepresented; (4) translation from English can introduce cultural misalignment.

Async RL and policy staleness. For long rollouts where weights may update before a rollout completes, the team's current solution is cross-stage distillation—train separate models per task type and distill back—rather than logical clocks or trust-region re-weighting.

Easy vs. hard problem classification. Run multiple models on all tasks and sort by pass rate for an objective ordering. More practically, use the model's own Pass@K as a quick signal. Two approaches were described: running multiple models on all tasks and sorting by aggregate success rate to get an objective ordering, or more practically, computing Pass@K on the model being trained. Problems solved in low K are easy; problems requiring high K or that consistently fail are hard. Pass@K is simple to compute and provides a reasonable empirical signal without requiring additional models.

Multimodality plans. Native multimodal training (vision + text together from pre-training) is the target. Challenges include: balancing modalities without hurting text; data-loader throughput (disk speed is already a bottleneck for text; video makes it far worse); and the absence of a single next-token-prediction objective for vision pre-training.

9 Summary

This lecture provided a practitioner's end-to-end view of frontier LLM training, emphasizing the following themes:

1. **Data is the foundation.** Diversity matters more than elaborate weighting schemes. Search-grounded synthetic data (TxT360-MIDAS) is a scalable path to improving diversity.
2. **Simple things break at scale.** Data packing randomness, tokenizer round-trip consistency, whitespace handling, and special-token management all cause real training failures.
3. **Parallelism is a deeply hardware-specific choice.** FSDP communication grows quasi-quadratically; the right TP/EP/DP combination shifts as the cluster grows; decisions must be locked in before the multi-month training run.
4. **Post-training is still an art.** RL is unstable; reward hacking can occur even in the evaluation harness; the iso-compute playbook offers partial predictability via the sigmoid fit of n^*.
5. **Eval parity is cheap insurance.** Four silent failure modes (tokenizer mismatch, chat-template drift, stop-sequence handling, contamination) can make a good model look bad or a bad model look good.
6. **The factory is the goal.** Continuous training, stage reduction, and self-evolved pipelines point toward a model that trains its own successor.
7. **Openness enables academic impact.** Full release of weights, data, code, checkpoints, and logs allows the community to reproduce, study, and build on frontier model training.