ANTERN DATA SOLUTIONS

AI-Native Engineering Sprint

Full 18-Week Curriculum

A systems-first curriculum spanning mathematical foundations, machine learning, LLM internals, backend and data engineering, agents, evaluation, reliability, security, research thinking, production capstones, and proof of work.

COHORT
Cohort 3

COHORT START
1 August 2026

DURATION
18 weeks

DELIVERY
Live instruction, guided builds, independent implementation, evaluation, and public proof of work

Vision: Not an AI course. A system for producing AI-native engineers who can reason from first principles, build production-grade systems, communicate technical decisions, and create credible proof of work.

Philosophy: Distribution over skill. Proof of work over credentials. Systems + Product + Reasoning + Execution — not just models.

Capability Profile

The sprint develops a balanced engineering profile across:

- Software engineering and production systems
- Machine learning and LLM fundamentals
- Agent engineering, retrieval, evaluation, reliability, and security
- Backend, data, infrastructure, deployment, and observability
- Research thinking and careful technical judgment
- Product judgment, ambiguity tolerance, and user-focused execution
- Clear writing, stakeholder communication, and public proof of work
- Fast iteration: idea → prototype → feedback → reliable system

Curriculum Architecture

```
Block A  (Weeks 1–4)   — Math + ML Foundations
Block B  (Weeks 5–7)   — Transformers + LLM Internals
Block C  (Weeks 8–10)  — Systems + Backend + Data Engineering
Block D  (Weeks 11–13) — LLM Engineering + RAG + Agents
Block E  (Weeks 14–15) — Evaluation + Reliability + Security
Block F  (Weeks 16–17) — RLHF + Agentic RL + Frontier Research
Block G  (Week 18)     — Production Capstone + Hiring Sprint
```

Horizontal Tracks (run all 18 weeks in parallel):

- AI Coding Systems Track (Claude Code, Cursor, Codex, Gemini CLI daily)
- Startup Operator Track (ship, talk to users, write, build in public)
- Paper Club (read → critique → reproduce → present weekly)
- Failure Friday (analyze real AI failures every week)
- Proof of Work (LinkedIn/X post every week)

The Learning Loop (Every Single Week, No Exceptions)

Every week follows this exact sequence. The order is not optional.

```
Step 1 — PRODUCTIVE FAILURE TRIGGER (30 min)
Students are thrown into a broken system or unsolvable problem
with only their current tools. They struggle. They feel the gap.
        ↓
Step 2 — RETRIEVAL WARMUP (15 min)
3 questions from 2–3 weeks ago. No notes. Forces spaced recall.
        ↓
Step 3 — THEORY + LIVE DEMONSTRATION (2 hrs)
The invention that solves what they just felt in Step 1.
Instructor builds it live. Students watch the thinking, not just the output.
        ↓
Step 4 — GUIDED BUILD → INDEPENDENT BUILD (4–10 hrs)
First: partial worked example (scaffold). Then: blank — they build from scratch.
        ↓
Step 5 — SELF-EXPLANATION CHECKPOINT
Specific questions about their own project. If they can't explain it, it's not done.
        ↓
Step 6 — INTEGRATE INTO THEIR WORLD
Proof of Work post + Engineering Judgment write-up.
Knowledge becomes public evidence.
```

Why this week exists: Every week's theory block maps directly to a specific capstone deliverable or interview question. This is stated explicitly at the top of each week.

Weekly Deliverable Structure

Deliverable	Time
Productive Failure trigger	30 min
Retrieval Warmup (3 questions from past weeks)	15 min
Theory study + live demonstration	2 hrs
Guided then independent build	4–10 hrs
Self-Explanation Checkpoint	30 min
Weekly paper: read → critique → reproduce	2 hrs
Failure Friday: analyze one real AI failure	1 hr
Proof of Work: LinkedIn or X post	30 min
Engineering Judgment write-up	30 min
Interview mapping	15 min

BLOCK A — Math + ML Foundations

Week 1 — Probability, Statistics & Information Theory

Why this week exists: Every loss function, every training objective, every evaluation metric traces back to probability theory. Without this week, students cargo-cult cross-entropy without knowing why it works.

Capstone connection: Reward model training (Week 16), evaluation system design (Week 14), and LLM-as-judge calibration all require this foundation.

Productive Failure Trigger

Give students a neural network that always predicts the majority class on an imbalanced dataset. Accuracy: 92%. Students declare success. Ask them to evaluate it on a fraud detection task. Watch it fail on every positive case. Question: "What went wrong? How do you measure this?" They struggle. Then introduce proper probability-grounded evaluation.

Retrieval Warmup

(Week 1 — no prior weeks. Students write what they already know about probability from memory. This becomes their baseline.)

Theory

Probability

- Random variables and probability spaces
- Bernoulli, Binomial, Gaussian (Normal) distributions — mean, variance, PDF
- Uniform, Poisson, Exponential distributions
- Expectations E[X] and Variance Var[X]
- Joint, marginal, and conditional probability
- Bayes' Rule and Bayesian updating
- Law of Total Probability
- Markov Chains — states, transition matrices, stationary distributions

Statistics

- Maximum Likelihood Estimation (MLE) — full derivation
- Maximum A Posteriori (MAP) estimation
- Bias-Variance tradeoff
- Confidence intervals
- Hypothesis testing, p-values (what they are and what they are not)
- Frequentist vs Bayesian interpretation
- Central Limit Theorem

Information Theory

- Entropy H(X) — measuring uncertainty
- Joint and conditional entropy
- Cross-entropy — how it connects to loss functions
- KL Divergence — measuring distribution difference
- Jensen-Shannon Divergence — the symmetric version; why it matters for GANs
- Mutual Information I(X;Y)
- Why cross-entropy is the loss function for classification (full derivation from MLE)

Decision Making Under Uncertainty (First Exposure)

- Expected value calculations
- Opportunity cost
- Bayesian reasoning in practice
- Fermi estimation problems
- Calibration — knowing what you don't know
- Exercise: "Should we fine-tune or use RAG?" framed as a decision theory problem

Project

- Build a Bayesian spam classifier from scratch — NumPy only, no sklearn
- Implement MLE parameter estimation manually
- Visualize Gaussian distributions and KL divergence
- Implement Jensen-Shannon Divergence from scratch

Self-Explanation Checkpoint

Answer in writing before submitting:

- Why does your classifier use cross-entropy instead of MSE? Derive it.
- What would happen to your classifier if the prior changed? Walk through it numerically.
- Where does KL divergence appear in your training loop, even if you didn't write it explicitly?

Paper

- Read: "A Few Useful Things to Know About Machine Learning" — Pedro Domingos

Failure Friday

- Analyze: Microsoft Tay (2016) — what probabilistic failure modes caused this?

Interview Mapping

- "Explain KL divergence and where it appears in training"
- "Derive the cross-entropy loss from MLE"
- "What is the bias-variance tradeoff?"
- "What is Jensen-Shannon divergence and how does it differ from KL?"

Week 2 — Linear Algebra, Optimization & Calculus for ML

Why this week exists: Backpropagation is the chain rule applied to a computational graph. Attention is dot products and softmax. Embeddings are vectors in high-dimensional space. All of this is linear algebra. Without this week, students cannot reason about what their models are actually doing.

Capstone connection: Implementing attention from scratch (Week 6), Flash Attention (Week 8), and deriving LoRA's math (Week 7) all require this.

Productive Failure Trigger

Give students a 3-layer network and ask them to manually compute the gradient of the loss with respect to the first layer's weights. No autograd. Watch them struggle with the chain rule across layers. Feel the pain. Then introduce computational graphs and autograd as the invention that solves this.

Retrieval Warmup

- What is the difference between joint and conditional probability?
- State Bayes' Rule and give one real ML application of it.
- What does KL divergence measure? When is it zero?

Theory

Linear Algebra

- Vectors, matrices, tensors
- Matrix multiplication and its geometric interpretation
- Transpose, inverse, rank, determinant
- Eigenvalues and eigenvectors — geometric meaning
- Singular Value Decomposition (SVD) — why it matters for embeddings
- Positive Semi-Definite (PSD) matrices — definition, where they appear (Hessians, covariance, kernel matrices)
- Null space and image space — rank deficiency and its ML implications
- Orthogonality — QR decomposition, Gram-Schmidt
- PCA from scratch
- Dot products and cosine similarity (embedding retrieval foundation)

Calculus

- Derivatives and partial derivatives
- The chain rule — the engine of backpropagation
- Jacobians and Hessians
- Computational graphs
- Automatic differentiation — how PyTorch autograd works under the hood
- Newton's Method — second-order optimization; when and why it's impractical at scale
- Quasi-Newton methods — L-BFGS (concept level; comes up in research interviews)

Optimization

- Gradient descent — derivation and intuition
- Stochastic Gradient Descent (SGD)
- Minibatch gradient descent
- Learning rate schedules — warmup, cosine decay
- Convexity and local vs global minima
- Saddle points

Decision Making Under Uncertainty (Applied)

- Fermi problems: estimate the number of GPU hours needed to train GPT-2
- Cost-benefit framing: "Should we spend $5k on evals or more inference compute?"

- Expected value applied to experiment prioritization

Project

- Implement gradient descent from scratch on a 2D loss surface
- Build a basic autograd engine (scalar-valued, like micrograd)
- Implement Newton's Method on a convex problem — compare convergence to SGD
- Visualize optimization paths: SGD vs Adam vs RMSProp vs Newton on the same loss surface

Self-Explanation Checkpoint

- Walk through backpropagation in your autograd engine line by line. What does each line compute geometrically?
- Why is Newton's Method impractical for neural networks? What specifically breaks down?
- What does a PSD matrix guarantee, and why does that matter for a loss surface?

Paper

- Read: Karpathy's "micrograd" walkthrough + source code

Failure Friday

- Analyze: A real training run that diverged due to learning rate misconfiguration — what happened, what should have been caught earlier

Interview Mapping

- "Walk me through backpropagation from first principles"
- "Why does Adam converge faster than SGD?"
- "What is SVD and where would you use it in an ML system?"
- "What is a positive semi-definite matrix? Where do you encounter them in ML?"

Week 3 — Classical ML & Engineering Judgment

Why this week exists: Classical ML gives you the intuition that deep learning obscures. Overfitting, regularization, evaluation, the bias-variance tradeoff — these concepts are clearer in simple models. Students who skip this can't reason about their neural networks either.

Capstone connection: Evaluation system design, slice testing, and knowing when not to use a neural network all start here.

Productive Failure Trigger

Give students a high-dimensional dataset (500 features, 200 samples) and ask them to train a logistic regression model. It overfits catastrophically. Accuracy: 100% train, 51% test. Question: "What went wrong? How do you fix it without more data?" They suggest more epochs, bigger models. Wrong instinct. Then introduce regularization as the invention.

Retrieval Warmup

- What is the chain rule? Write it out for a 2-layer computation.
- What does SVD decompose a matrix into? Name one ML application.
- What is the difference between MLE and MAP?

Theory

Supervised Learning from Scratch

- Linear Regression — normal equation and gradient descent solution
- Logistic Regression — derivation, sigmoid, binary cross-entropy
- Regularization — L1 (Lasso), L2 (Ridge), ElasticNet — geometric interpretation of each
- Decision Trees — information gain, Gini impurity
- Ensemble Methods — Random Forests, Gradient Boosting, XGBoost intuition
- Support Vector Machines — maximum margin, kernel trick

- k-Nearest Neighbors

Unsupervised Learning

- k-Means clustering
- DBSCAN
- Gaussian Mixture Models
- Dimensionality reduction — PCA, t-SNE, UMAP

Advanced ML Concepts

- Gumbel-Softmax — the reparameterization trick for discrete distributions; used in VAEs and discrete token sampling
- Domain adaptation — when train and test distributions differ
- Transfer learning — how pretrained representations reduce data requirements
- Few-shot and zero-shot learning — the in-context learning precursor

Model Evaluation

- Train/val/test splits
- Cross-validation
- Precision, Recall, F1, AUC-ROC
- Calibration curves
- Confusion matrices
- When accuracy is a misleading metric

Engineering Judgment — First Principles Every week, answer in writing:

1. What would you do?
2. Why?
3. What tradeoff are you making?
4. What alternative did you reject?

This week: "A client wants a churn prediction model. Should you start with logistic regression or XGBoost? Defend your choice with explicit tradeoff analysis."

Project

- Build Logistic Regression from scratch in NumPy — no sklearn
- Train on a real tabular dataset (UCI Adult Income or similar)
- Implement L1 and L2 regularization manually — visualize the effect on weights
- Implement Gumbel-Softmax from scratch — demonstrate discrete sampling
- Full evaluation pipeline: precision, recall, F1, AUC-ROC

Self-Explanation Checkpoint

- Why does L1 regularization produce sparse weights but L2 does not? Explain geometrically.
- What is Gumbel-Softmax solving? Why can't you just use argmax?
- Your model has 95% accuracy. A stakeholder is happy. Why might you not be?

Paper

- Read: "Attention Is All You Need" — Vaswani et al. (preview only; full deep-dive in Week 6)

Failure Friday

- Analyze: A real ML model deployed in production that failed silently — what evaluation gaps caused the failure?

Interview Mapping

- "How does regularization prevent overfitting geometrically?"
- "Walk me through how you'd set up evaluation for a churn model"
- "What is the Gumbel-Softmax trick and where is it used?"

Week 4 — Deep Learning Foundations

Why this week exists: Neural networks are not magic. They are composed matrix multiplications with nonlinearities, trained by backpropagation. Students who don't build one from scratch cannot debug one in production.

Capstone connection: Every project in this curriculum runs on a neural network. Understanding training mechanics — overfitting, normalization, gradient flow — is required to debug every system you will ever build.

Productive Failure Trigger

Give students a 5-layer MLP with no normalization and random weight initialization. Ask them to train it on MNIST. Watch the gradients vanish in the early layers — training stalls at 40% accuracy. Question: "The loss isn't going down. The model looks right. What's wrong?" They inspect the code and find nothing. Then introduce BatchNorm and proper weight initialization as the inventions.

Retrieval Warmup

- What is L2 regularization doing geometrically to the weight vector?
- Write the logistic regression gradient update rule from memory.
- What is the bias-variance tradeoff? Give a concrete example.

Theory

Multilayer Perceptron (MLP)

- Perceptron — the simplest neural network
- Activation functions: Sigmoid, Tanh, ReLU, GELU, SiLU — and why they exist
- Universal approximation theorem — what it says and what it doesn't
- Forward pass as matrix multiplications
- Backpropagation — full derivation through a two-layer network
- Gradient flow — vanishing and exploding gradients
- Weight initialization — Xavier, He, why it matters

Optimization (Deep)

- SGD with momentum
- RMSProp — adaptive learning rates
- Adam and AdamW — combining momentum + adaptive rates; full derivation
- Learning rate warmup and cosine annealing
- Gradient clipping
- Why Adam often outperforms SGD on non-convex landscapes

Training Mechanics

- Overfitting — signs, symptoms, causes
- Regularization: Dropout, Weight Decay, L1/L2
- Batch Normalization — derivation and why it helps training stability
- Layer Normalization — and why transformers use it instead of BatchNorm
- RMSNorm — simplified LayerNorm, computational advantages
- Gradient checkpointing — trading compute for memory; how it works mechanically
- Early stopping
- Data augmentation as regularization

Numerical Precision

- Floating point representation — FP32, FP16, BF16, TF32 at the bit level
- Why BF16 preserves range while FP16 preserves precision
- Overflow and underflow in practice
- Numerical precision tricks: loss scaling, epsilon in Adam denominator

- Why these matter: silent NaN bugs are the hardest to catch

Scientific Thinking (First Exposure)

- Experiment design: hypothesis → independent variables → controls → metrics → confounders
- "How do I know this actually works?" vs "Twitter said it works"
- Ablation studies — isolating one variable at a time

Project

- Build full MLP framework from scratch (forward pass, backward pass, optimizers — no PyTorch autograd)
- Train on MNIST — target >98% accuracy
- Implement BatchNorm and Dropout from scratch
- Ablation study: train 4 variants (no norm, BatchNorm, LayerNorm, RMSNorm) — measure and report difference
- Implement gradient checkpointing and measure memory savings

Self-Explanation Checkpoint

- Your BatchNorm implementation: what exactly are you normalizing, and why does the order of operations matter?
- Explain why gradient checkpointing saves memory. What is the compute cost you pay for it?
- You see a NaN loss after 200 steps. Walk me through your debugging process — where do you look first?

Paper

- Read: LeCun et al. (1998) "Gradient-Based Learning Applied to Document Recognition"

Failure Friday

- Analyze: A case of silent gradient explosion in production — how do you detect it? How do you fix it?

Interview Mapping

- "Derive backpropagation through a two-layer network"
- "Why does BatchNorm help? Why do transformers prefer LayerNorm?"
- "Explain gradient checkpointing — what is the memory-compute tradeoff?"
- "What is BF16 and why do large model training runs prefer it over FP16?"

BLOCK B — Transformers + LLM Internals

Week 5 — Sequence Models, State Spaces & The Road to Transformers

Why this week exists: You cannot appreciate why attention works until you feel what it replaces. RNNs, LSTMs, S4, and Griffin are not historical curiosities — they are live architectural alternatives that come up in interviews at DeepMind, Anthropic, and research-focused labs.

Capstone connection: Understanding architecture tradeoffs is required for the system design and ADR deliverables in Week 18.

Productive Failure Trigger

Give students an LSTM trained on a 1000-token sequence. Show them the gradient magnitudes at each timestep — they decay to near-zero by token 50. Ask: "Your model cannot remember what happened 500 tokens ago. You cannot add more memory. The architecture is the constraint. How do you redesign it?" They propose hacks. Then introduce attention as the invention that removes the bottleneck entirely.

Retrieval Warmup

- What is gradient checkpointing and what does it trade?
- What is BF16 and why does training prefer it over FP16?
- What is the Xavier initialization formula and what problem does it solve?

Theory

Sequence Modeling History

- Recurrent Neural Networks (RNNs) — how they work, why they struggle
- Vanishing gradient problem in sequences — formal analysis
- LSTMs — forget gate, input gate, output gate, cell state
- GRUs — simplified gating
- Sequence-to-sequence models — encoder-decoder architecture
- The attention mechanism — its original form (Bahdanau, 2015)
- Why attention was the breakthrough: alignment without bottleneck

State Space Models (SSMs)

- The S4 model — structured state space sequences
- How SSMs model long-range dependencies differently from attention
- Linear recurrence as the core primitive
- Mamba — selective state spaces; why it's faster than attention at long sequences
- When SSMs outperform transformers (very long sequences, streaming)
- When transformers outperform SSMs (complex reasoning, tool use)

The Griffin Architecture (DeepMind)

- Hybrid RNN-transformer design
- Linear recurrences + local attention
- Why DeepMind built it: memory efficiency at inference
- Architectural tradeoffs vs pure transformer

TransformerXL

- The segment recurrence problem in vanilla transformers
- Relative positional encodings — why they enable longer context
- Memory tokens across segments

Tokenization

- Why we tokenize at all
- Character-level tokenization — pros and cons
- Word-level tokenization — out-of-vocabulary problem
- Byte Pair Encoding (BPE) — algorithm walkthrough
- SentencePiece
- tiktoken — how OpenAI tokenizes
- Vocabulary size tradeoffs
- Tokenization artifacts and failure modes (non-English languages, code, math)

Embeddings

- Word2Vec — CBOW and Skip-gram
- GloVe
- What embeddings actually represent geometrically
- Positional encoding — sinusoidal (original) and learned
- Relative positional encodings — TransformerXL style
- The embedding as a lookup table

Research Taste (First Exposure)

- How to read a paper: abstract → introduction → figures → related work → method → results
- How to distinguish foundational papers from hype
- Paper taxonomy: Foundation / Scaling / Systems / Alignment / Agent papers

- Critique exercise: "What did the Attention paper get wrong? What has changed since 2017?"

Project

- Build a character-level language model (vanilla RNN, then LSTM)
- Implement BPE tokenizer from scratch
- Implement a simplified S4 layer — compare sequence modeling on a long-sequence task vs LSTM
- Visualize attention weights in a seq2seq model
- Benchmark: LSTM vs S4 on sequence length 512 and 2048 — where does each break?

Self-Explanation Checkpoint

- What specific architectural property of S4 allows it to handle longer sequences than an LSTM?
- Why did TransformerXL introduce relative positional encodings? What breaks with absolute encodings at long range?
- Griffin is a hybrid architecture. What does it take from RNNs and what does it take from transformers? Why?

Paper

- Read: "Neural Machine Translation by Jointly Learning to Align and Translate" — Bahdanau et al. (2015)
- Secondary read (skim): "Mamba: Linear-Time Sequence Modeling with Selective State Spaces"

Failure Friday

- Analyze: A production NLP system that failed because of tokenization — common in non-English deployments

Interview Mapping

- "Why did attention replace recurrence?"
- "What is S4 and when would you use it over a transformer?"
- "What is the Griffin architecture and what problem does it solve?"
- "Walk me through BPE tokenization"

Week 6 — Transformers: Architecture Deep Dive

Why this week exists: GPT is the foundation of everything in this curriculum. You cannot engineer systems built on top of something you don't understand from the inside. This week is the most important theory week in the program.

Capstone connection: Every agent, every RAG system, every fine-tuned model you build later runs on this architecture. Interview question frequency: this week covers ~40% of all LLM engineering interview questions asked at top companies.

Productive Failure Trigger

Show students a vanilla attention implementation. Run it on a 4096-token sequence. Memory usage: 64GB. It OOMs. Question: "The math is correct. The implementation is correct. But it doesn't fit on any GPU you can afford. What do you do?" They suggest batching, chunking. Then introduce Flash Attention as the invention — same math, completely different memory profile.

Retrieval Warmup

- What is the vanishing gradient problem in RNNs? Why does LSTM help?
- What is BPE? Walk through 3 merge steps on a small vocabulary.
- What is the key architectural difference between encoder and decoder transformers?

Theory

The Transformer Architecture (Full)

- Self-attention — query, key, value; scaled dot-product derivation
- Multi-head attention — why multiple heads, what each head learns
- The attention mask — causal masking for autoregressive generation
- Feed-forward network — two linear layers with GELU; why the expansion ratio matters

- Residual connections — and why they're critical for gradient flow
- Layer normalization — pre-norm vs post-norm, why pre-norm is now standard
- Encoder vs Decoder vs Encoder-Decoder architectures

Modern GPT Internals

- RMSNorm — simplified LayerNorm, why it's computationally preferred
- Rotary Positional Embeddings (RoPE) — how position is encoded in Q/K space
- Sinusoidal embeddings — original approach; tradeoffs vs RoPE
- Relative positional embeddings — the family of approaches
- SwiGLU activation — replacing GeLU in the FFN; performance reasons
- Grouped Query Attention (GQA) — reducing KV cache memory
- KV Cache — what it is, why it exists, how it enables fast inference
- Flash Attention — the IO-aware attention algorithm (concept; implementation in Week 8)
- Parameter count derivation — calculate GPT-2 parameters from scratch

Advanced Architectures

- Mixture of Experts (MoE) — sparse activation; routing mechanism; expert capacity
- Why MoE scales parameter count without proportional compute cost
- Load balancing problem in MoE — the auxiliary loss
- Mixtral, DeepSeek MoE — real-world implementations
- When MoE helps and when it hurts (inference complexity, communication overhead)
- The Perceiver — cross-attention to a fixed-size latent array; handling arbitrary input modalities
- How architecture choices affect inference cost, memory, and latency

Scaling

- Scaling laws — Kaplan et al. and Chinchilla
- Compute-optimal training
- Why bigger isn't always better; the Chinchilla insight
- LLM scaling factor — tokens per parameter

Research Taste Applied

- Why "Attention Is All You Need" changed the field
- What it got wrong (positional encodings evolved, pre-norm is better, FFN grew)
- Reading papers critically, not reverently

Project

- Build GPT from scratch (PyTorch)
- Train on Tiny Shakespeare — target coherent text generation
- Implement KV cache and measure inference speedup
- Implement a basic MoE layer — replace one FFN block with 4 experts + router
- Calculate parameter counts for GPT-2 by hand, then verify against HuggingFace config
- Benchmark: dense FFN vs MoE — measure routing distribution and expert utilization

Self-Explanation Checkpoint

- Walk through exactly what happens to a single token as it passes through every component of your GPT. Be specific about shapes at each step.
- Your MoE router is sending 80% of tokens to 1 expert. What's happening? How do you fix it?
- RoPE vs sinusoidal embeddings: what does RoPE encode differently, and why does it extrapolate better to longer sequences?

Paper

- Read + reproduce: "Attention Is All You Need" — Vaswani et al. (2017)

- Secondary: "Language Models are Few-Shot Learners" — GPT-3 paper (skim)

Failure Friday

- Analyze: A production LLM that hallucinated confidently — failure mode analysis at the attention and probability level

Interview Mapping

- "Derive scaled dot-product attention from scratch"
- "What is RoPE and why was it invented?"
- "Explain Mixture of Experts — what is the routing mechanism and what is the load balancing problem?"
- "What does the KV cache actually store and why does it matter for inference cost?"

Week 7 — LLM Internals: Inference, Quantization & Fine-Tuning

Why this week exists: Understanding how to train a model is only half the job. Understanding how to serve it cheaply and adapt it efficiently is what separates engineers who build demos from engineers who build products.

Capstone connection: Every deployed system in this curriculum requires inference optimization decisions. LoRA is used in Week 16 training. Quantization tradeoffs appear in Week 18 cost dashboards.

Productive Failure Trigger

Students are given a Llama 3 8B model and told to serve it on a single A100 80GB GPU for 100 concurrent users. Naive implementation: each request blocks until complete, memory fills up, throughput collapses to 2 requests/second. Question: "The GPU has plenty of memory. The model is fast on one request. What's broken at scale?" Then introduce continuous batching and PagedAttention as the inventions.

Retrieval Warmup

- What is GQA and how does it reduce KV cache memory?
- What is the load balancing problem in MoE? How is it typically addressed?
- What is the difference between pre-norm and post-norm transformers?

Theory

Inference Deep Dive

- Autoregressive generation — token-by-token sampling
- Temperature, top-k, top-p (nucleus) sampling — derivation and effects
- Beam search vs greedy vs sampling
- Repetition penalty
- Speculative decoding — how a draft model + verifier model speeds inference
- Continuous batching — how vLLM serves multiple users simultaneously
- PagedAttention — KV cache memory management (vLLM's key innovation)

Quantization

- Why quantization: memory and compute savings
- Floating point representation recap: FP32 → FP16 → BF16 — tradeoffs
- INT8 quantization — post-training quantization (PTQ)
- 4-bit quantization — GPTQ, AWQ
- QLoRA — fine-tuning quantized models
- Accuracy vs speed tradeoff analysis — when does quantization hurt?
- JIT compiling — torch.compile and JAX JIT; what they do and when to use them

Parameter Efficient Fine-Tuning

- Why full fine-tuning is prohibitively expensive
- LoRA — Low-Rank Adaptation; the math: W = W■ + BA where B∈ R^(d×r), A∈ R^(r×k)
- QLoRA — LoRA on quantized models

- FSDP (Fully Sharded Data Parallel) — how it differs from ZeRO; when to use each
- Prefix tuning and prompt tuning (historical context)
- When to fine-tune vs when to use few-shot prompting — the decision framework

AI Infrastructure (First Exposure)

- vLLM — architecture and key innovations (PagedAttention, continuous batching)
- SGLang — structured generation serving
- TGI (Text Generation Inference)
- GPU memory hierarchy: HBM → L2 cache → SRAM
- Batching strategies: static vs dynamic vs continuous
- Caching KV across requests

Project

- Deploy Llama 3 or Mistral locally using vLLM
- Implement top-p sampling from scratch
- Run 4-bit quantization with GPTQ — measure accuracy/latency tradeoff on a benchmark
- Fine-tune a small model with LoRA on a custom dataset
- Benchmark: naive serving vs vLLM continuous batching — measure throughput improvement

Self-Explanation Checkpoint

- Walk through the LoRA math. Why does low-rank decomposition preserve most of the model's capability?
- PagedAttention: what specifically was fragmented before, and how does paging fix it?
- You need to choose between QLoRA fine-tuning and full fine-tuning on 2 A100s. Walk through the decision.

Paper

- Read: "LoRA: Low-Rank Adaptation of Large Language Models" — Hu et al. (2021)
- Secondary: FlashAttention paper (concept level — full implementation next week)

Failure Friday

- Analyze: Inference cost blowup in production — a team that didn't think about batching or quantization

Interview Mapping

- "How does speculative decoding work?"
- "What is PagedAttention and why does vLLM use it?"
- "Walk me through LoRA — the math, not just the concept"
- "What is torch.compile doing and when does it help?"

BLOCK C — Systems + Backend + Data Engineering

Week 8 — GPU Architecture & AI Systems

Why this week exists: Every performance bottleneck you will ever hit in AI engineering comes from misunderstanding the hardware. This week is where 99% of bootcamps fail. Engineers who understand roofline analysis can diagnose and fix problems that everyone else just accepts.

Capstone connection: The AI Infrastructure capstone track (Track D) starts here. Profiling and optimization skills are directly tested in systems interviews at every top lab.

Productive Failure Trigger

Give students a naive attention implementation and a FlashAttention implementation. Both produce identical outputs. But naive attention OOMs on 8K tokens and Flash Attention doesn't. Question: "The math is the same. The output is the same. Why does one run out of memory?" They look at the code. They don't see it. Then introduce the memory hierarchy — HBM vs SRAM — as the reason, and tiling as the invention.

Retrieval Warmup

- What is FSDP and how does it differ from ZeRO stage 3?
- Walk through speculative decoding. What are the two models doing?
- Why does continuous batching improve GPU utilization vs static batching?

Theory

GPU Architecture

- CPU vs GPU — parallelism philosophy
- CUDA programming model — threads, blocks, grids, warps
- GPU memory hierarchy: global memory (HBM) → shared memory (SRAM) → registers
- Memory bandwidth as the bottleneck (not compute)
- TPU architecture — systolic arrays
- Memory hierarchy: HBM, L2 cache, SRAM — sizes, bandwidths, latencies

Floating Point & Precision (Systems Level)

- FP32, FP16, BF16, TF32 — how each is stored in GPU memory
- Mixed precision training — FP16 forward pass, BF16 master weights, FP32 optimizer state
- Loss scaling — preventing underflow in FP16 gradients
- Detecting and recovering from NaN/Inf in training

Performance Analysis

- Roofline model — compute-bound vs memory-bound operations
- FLOPs — counting operations in attention, FFN, linear layers
- Arithmetic intensity — FLOPs per byte
- How to profile GPU kernels — nsight, torch.profiler
- Where time actually goes in a transformer forward pass

Parallelism Strategies

- Data Parallelism (DP) — replicate model, split batch
- Tensor Parallelism (TP) — split individual operations across GPUs
- Pipeline Parallelism (PP) — split layers across GPUs
- Fully Sharded Data Parallel (FSDP) — shard parameters, gradients, optimizer state
- Sequence Parallelism
- 3D Parallelism — combining DP + TP + PP
- ZeRO stages (DeepSpeed) — stage 1, 2, 3 — what each shards

Distributed Training

- All-reduce, all-gather, reduce-scatter — what each collective does
- NCCL — GPU communication library
- Ring all-reduce — why the ring topology minimizes communication
- Gradient accumulation
- Mixed precision training mechanics

Triton

- Why Triton exists (Python-level GPU programming without CUDA C)
- Writing a simple Triton kernel — fused operations

- When to write custom kernels vs use existing ones

Flash Attention Implementation

- Online softmax — the algorithmic trick that makes tiling possible
- Tiling in SRAM to avoid HBM round-trips
- Flash Attention vs Flash Attention 2 improvements
- IO complexity analysis: $O(N^2)$ memory $\rightarrow$ $O(N)$ memory
- Gradient checkpointing in Flash Attention — recomputing rather than storing

Project

- Write CUDA kernels for matrix multiplication — optimize with shared memory tiling
- Implement Flash Attention (simplified Triton version)
- Profile transformer training with torch.profiler: identify top-3 bottlenecks
- Benchmark throughput improvements from: baseline $\rightarrow$ tiling $\rightarrow$ Flash Attention $\rightarrow$ mixed precision

Self-Explanation Checkpoint

- Walk through why naive attention uses $O(N^2)$ memory. Show the exact tensor that causes this. Then show how tiling avoids storing it.
- You're told your kernel is memory-bound. What does that mean on the roofline model? What do you do about it?
- Explain the difference between ZeRO stage 3 and FSDP. When would you choose each?

Paper

- Read + reproduce: "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness" — Dao et al.

Failure Friday

- Analyze: A training run that silently ran 10x slower than expected — OOM fragmentation, suboptimal parallelism strategy

Interview Mapping

- "What is the roofline model? Is attention compute-bound or memory-bound?"
- "Walk me through Tensor Parallelism vs Pipeline Parallelism"
- "How does Flash Attention reduce memory from $O(N^2)$ to $O(N)$?"
- "What is FSDP doing at each training step?"

Week 9 — Backend Engineering for AI Products

Why this week exists: AI engineers who cannot build software are unemployable at startups and underperforming at labs. You are not a model trainer. You are a product builder. The model is one component.

Capstone connection: Every capstone track requires a deployed backend. This week is the plumbing that makes everything else reachable.

Productive Failure Trigger

Give students a working FastAPI endpoint that calls an LLM. It handles one request beautifully. Simulate 50 concurrent users with a load test. The server hangs — blocking calls, no connection pooling, synchronous code in an async framework. Latency goes from 200ms to 45 seconds. Question: "The code works. Why doesn't it scale?" Then introduce async Python, connection pooling, and task queues as the inventions.

Retrieval Warmup

- What is NCCL and what does it do in distributed training?
- Explain the difference between data parallelism and tensor parallelism.
- What does arithmetic intensity measure on the roofline model?

Theory

APIs

- REST API design — resource modeling, status codes, idempotency
- FastAPI — async Python, dependency injection, Pydantic validation
- WebSockets — real-time streaming for LLM outputs
- Server-Sent Events (SSE) — streaming tokens to frontend
- Async Python — event loop, async/await, concurrent requests, the GIL
- API versioning and backward compatibility

Databases

- Postgres internals — B-tree indexes, query planner, EXPLAIN ANALYZE
- Indexes — when to use them, when they hurt write performance
- Connection pooling — PgBouncer; why it's essential under load
- Redis — caching patterns: cache-aside, write-through, write-behind
- Redis pub/sub for real-time features
- OLTP vs OLAP — when to use which

Task Queues and Background Jobs

- Celery — distributed task queue
- ARQ — async task queue for Python
- When to use task queues in AI systems (async inference, batch embedding jobs)
- Dead letter queues and retry logic
- Job prioritization

Auth and Security

- JWT — structure, signing, verification, refresh tokens
- RBAC — role-based access control
- API key management
- Rate limiting strategies

Deployment

- Docker — Dockerfile, multi-stage builds, layer caching
- Docker Compose — multi-service local development
- Nginx — reverse proxy, load balancing, SSL termination
- Cloud basics: AWS EC2, S3, Lambda; Vercel; Railway; Render
- Environment management and secrets
- Health checks and graceful shutdown

Distributed Systems (First Block)

- CAP theorem — Consistency, Availability, Partition tolerance
- Consensus algorithms — Raft (concept level)
- Eventual consistency and its implications
- Replication strategies
- Sharding — horizontal partitioning

Product Engineering (First Exposure)

- Every API you build serves a user. Who is the user? What is their job-to-be-done?
- Before writing any code this week: "Who needs this API? What decision does it enable?"
- Product metrics for backend systems: p50/p90/p99 latency, error rate, throughput

Project

- Build a production-grade AI API with FastAPI

- Async streaming LLM responses via WebSockets + SSE
- Postgres backend with proper indexing (verify with EXPLAIN ANALYZE)
- Redis caching layer for repeated queries
- Celery for background embedding generation
- Dockerize and deploy to Railway or Render
- Add JWT auth and rate limiting
- Load test with 50 concurrent users — measure p99 latency before and after optimization

Self-Explanation Checkpoint

- Your FastAPI endpoint is slow under load but fast on a single request. Walk through every place where blocking could occur.
- Why does connection pooling matter? What specifically happens without it at 50 concurrent users?
- A user asks: "Why is my request taking 8 seconds?" Walk through how you'd diagnose this end-to-end.

Paper

- Read: "Designing Data-Intensive Applications" — Chapter 1 (Kleppmann)

Failure Friday

- Analyze: A production AI API that went down under load — no connection pooling, no rate limiting, blocking event loop

Interview Mapping

- "Design an API that streams LLM responses to many concurrent users"
- "When would you choose Redis vs Postgres for storing session context?"
- "How does CAP theorem apply to a distributed agent system?"

Week 10 — Data Engineering

Why this week exists: Most real AI projects fail because of data, not models. A model is only as good as what you feed it. FDEs spend the majority of their time on data pipelines, not model selection.

Capstone connection: The embedding pipeline, vector DB ingestion, and retrieval system in Weeks 11–12 all depend on the infrastructure built here.

Productive Failure Trigger

Give students a RAG system that answers questions about a large document corpus. The retrieval is great. The answers are often wrong. Investigation reveals: the chunking split sentences mid-thought, the embedding model was changed 2 weeks ago and old embeddings weren't recomputed, and the Airflow job that keeps the index fresh has been failing silently for 5 days. Question: "The model is fine. The retrieval looks reasonable. Where is the problem?" They debug for 20 minutes. Then introduce data pipeline observability and embedding versioning as the inventions.

Retrieval Warmup

- What is idempotency and why does it matter in a task queue?
- What does EXPLAIN ANALYZE show you in Postgres?
- What is CAP theorem? Give an AI system example of the tradeoff.

Theory

Data Modeling

- Relational data modeling — normalization, 1NF/2NF/3NF
- OLTP vs OLAP design patterns
- Star schema, snowflake schema
- Data vault modeling for historical tracking

Database Internals

- Postgres internals — storage, MVCC, vacuum, WAL
- Query optimization — indexes, query plans, covering indexes
- Partitioning — range, hash, list
- JSONB in Postgres — when to use it

Data Pipelines

- ETL vs ELT — tradeoffs and when each makes sense
- Apache Airflow — DAGs, operators, scheduling, sensors
- Data quality — great_expectations, dbt testing
- Schema evolution and migration strategies

Streaming Data

- Event-driven architecture — producers, consumers, topics
- Apache Kafka — architecture, partitions, consumer groups, offsets
- Change Data Capture (CDC) — Debezium, tracking changes in source systems
- Stream processing — filtering, aggregation, windowing
- Flink vs Spark Streaming

Feature Engineering & Feature Stores

- Feature engineering for ML — encoding, imputation, scaling
- Feature Store concepts — point-in-time correctness
- Online vs offline feature serving
- Feast, Tecton (concept level)

Embedding Pipelines

- Chunking strategies — fixed size, semantic, recursive
- Embedding model selection — sentence-transformers, OpenAI embeddings
- Batch vs online embedding generation
- Embedding versioning — what happens when you change models? Full recompute strategy.

Vector Database Internals

- HNSW — Hierarchical Navigable Small World (the algorithm)
- IVF-PQ — Inverted File Index with Product Quantization
- Pgvector vs Pinecone vs Weaviate vs Qdrant — architectural tradeoffs
- ANN vs exact search tradeoffs
- Filtering on metadata with vector search

Project

Full pipeline from raw data to retrieval:

```
Raw Data (CSV/JSON/Postgres)
        ↓
Cleaning & Validation (Airflow DAG with great_expectations)
        ↓
Feature Engineering
        ↓
Embedding Generation (batch, versioned)
        ↓
Vector DB Ingestion (Pgvector)
        ↓
Retrieval API
```

- Deploy Kafka locally, implement a CDC pipeline
- Build a feature store for a tabular ML dataset
- Ship an embedding pipeline with semantic chunking + Pgvector
- Add embedding version tracking — simulate a model change and full recompute

Self-Explanation Checkpoint

- Your embedding pipeline ran last night. This morning, retrieval quality dropped. Walk through your debugging process — what do you check first?
- What is HNSW doing algorithmically? Why is it approximate rather than exact?
- Your Airflow DAG failed at step 3 of 5. Step 4 and 5 didn't run. When the DAG reruns, what do you need to ensure about steps 1–3?

Paper

- Read: "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" — Lewis et al. (2020)

Failure Friday

- Analyze: A production RAG system with poor retrieval — traced to silent embedding pipeline failures and chunking decisions

Interview Mapping

- "Design an embedding pipeline that handles 10M documents and supports model versioning"
- "What is Change Data Capture and why does it matter for AI systems?"
- "Explain HNSW — how does approximate nearest neighbor search work?"

BLOCK D — LLM Engineering + RAG + Agents

Week 11 — LLM Engineering & Context Engineering

Why this week exists: Most AI engineers know how to call an API. Very few understand what goes in the context window, why it matters, and how to evaluate whether it worked. This week closes that gap.

Capstone connection: The Perplexity clone project is a direct capstone preview. The evaluation and cost tracking skills built here are used in every subsequent week.

Productive Failure Trigger

Give students a simple RAG chatbot. Ask them to evaluate whether it's good. They say "it seems to answer correctly." Ask them to prove it. They can't. Ask them to measure whether a prompt change made it better or worse. They can't. Question: "How do you know your AI system is working?" Then introduce golden datasets, LLM-as-judge, and systematic evaluation as the inventions.

Retrieval Warmup

- What is HNSW? What does "navigable small world" refer to?
- What is point-in-time correctness in a feature store? Why does it matter?
- What is the difference between ETL and ELT?

Theory

Prompt Engineering

- Zero-shot, few-shot, many-shot prompting
- Chain-of-thought (CoT) prompting — why it works (process vs answer)
- Tree-of-thought (ToT)
- Self-consistency — majority voting over multiple CoT paths
- Role prompting and system prompt design
- Prompt chaining — breaking complex tasks into steps
- Prompt compression — getting the same output with fewer tokens
- Negative prompting — specifying what not to do

Context Engineering

- The context window as the environment — not a chat interface
- What to put in context: tools, memory, retrieved docs, history, examples
- Context prioritization — what goes first matters (primacy and recency effects)
- Lost-in-the-middle problem — models underperform on information in the middle
- Long context strategies — summarization, compression, rolling window
- System prompt design for production
- Context window size vs cost tradeoffs

Structured Outputs

- JSON mode and structured output APIs
- Pydantic integration with LLM outputs
- Function calling — how it works at the API level
- Tool calling — definitions, arguments, result injection
- Instructor library patterns
- Handling schema validation failures gracefully

Memory Systems

- In-context memory (conversation history)
- External memory (vector DB retrieval)
- Episodic memory — storing past interactions
- Semantic memory — storing facts and knowledge
- Procedural memory — storing skills and instructions
- Memory compression and summarization

Evaluation (First Exposure)

- Why "vibes" aren't enough
- Golden datasets — curating ground truth examples
- LLM-as-judge — using a model to evaluate a model
- Unit tests for LLM behavior
- Regression testing — don't break what works

Economics of AI (First Block)

- Token costs — input vs output pricing
- Latency economics — P50/P90/P99 and user experience
- GPU economics — spot vs on-demand, reserved capacity
- Open-source vs closed-source tradeoffs at scale
- Build vs buy — when to use the API vs fine-tune vs host your own

Product Engineering

- Before building: "Who needs this? What decision does it help them make?"
- JTBD (Jobs to Be Done) applied to AI product design
- User observation — what do users actually do vs what they say they want?
- The difference between a working demo and a useful product

Project

- Build a Perplexity clone with streaming, citations, and source attribution
- Implement LLM-as-judge evaluation system with calibration
- Build a memory system that persists across conversations
- Cost dashboard tracking per-query token spend
- Before starting: interview 2 potential users — document what they actually need

Self-Explanation Checkpoint

- Your LLM-as-judge gives a response a 7/10. What does that number mean? What are three ways it could be wrong?
- You have a 128K context window. How do you decide what to put in it?
- A user says the chatbot "feels worse" after your prompt change. How do you prove or disprove this?

Paper

- Read: "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models" — Wei et al.

Failure Friday

- Analyze: A production chatbot that lost context in long conversations — root cause in context management

Interview Mapping

- "How would you design a memory system for a long-running AI assistant?"
- "What is the lost-in-the-middle problem and how do you mitigate it?"
- "How do you evaluate an LLM system without a ground truth dataset?"

Week 12 — Retrieval Science & Advanced RAG

Why this week exists: Most RAG courses teach vector databases. That is the shallow version. Production RAG fails at retrieval, not generation. This week teaches the full retrieval science that separates engineers who build RAG demos from engineers who build RAG products.

Capstone connection: Every capstone track that involves document search, knowledge agents, or research agents depends directly on this week.

Productive Failure Trigger

Give students a RAG system with dense retrieval. It works well on simple queries. Give it: "What was the revenue growth rate in Q3 2022 compared to the same period the previous year, adjusted for the acquisition?" It returns the wrong chunk. The answer exists in the corpus. Question: "The document is there. The embedding model is good. Why didn't we get it?" Then introduce hybrid retrieval, query decomposition, and HyDE as the inventions.

Retrieval Warmup

- What is the lost-in-the-middle problem? How does context prioritization address it?
- What is self-consistency prompting and when does it help?
- What is LLM-as-judge? Name two failure modes.

Theory

Retrieval Theory

- Why sparse retrieval still matters
- TF-IDF — derivation and intuition
- BM25 — the gold standard sparse retrieval algorithm; full formula derivation
- Dense retrieval — bi-encoder models (DPR, E5, BGE)
- Cross-encoders — reranking; why they're more accurate but slower
- Hybrid retrieval — combining sparse + dense scores (RRF, weighted fusion)
- ColBERT — late interaction for efficient but accurate retrieval
- Query expansion — HyDE (Hypothetical Document Embeddings)
- Query decomposition — breaking complex queries into sub-queries
- Multi-hop retrieval — chaining retrievals for complex questions

Advanced RAG Architectures

- Naive RAG — retrieve once, generate (and its failures)
- Advanced RAG — pre-retrieval, retrieval, post-retrieval processing

- Modular RAG — treating retrieval as configurable modules
- Self-RAG — model decides when to retrieve and how to use results
- Corrective RAG (CRAG) — evaluating and correcting retrieval quality
- Iterative RAG — retrieve → generate → critique → re-retrieve

Knowledge Graphs

- Graph-based knowledge representation
- GraphRAG — Microsoft's approach
- Combining structured knowledge with vector retrieval
- When to use knowledge graphs vs vector DBs

Reranking

- Cross-encoder reranking — full query-document attention
- Cohere Rerank, Jina Reranker
- Reciprocal Rank Fusion (RRF)
- Diversity in reranking — MMR (Maximum Marginal Relevance)

Chunking Science

- Fixed-size chunking — when it fails
- Semantic chunking — sentence transformers + similarity threshold
- Recursive character chunking
- Document-level vs passage-level vs sentence-level
- Chunk overlap and its effects
- Parent-child chunk indexing

Project

- Build full RAG pipeline with BM25 + dense retrieval hybrid
- Implement and benchmark: naive RAG vs HyDE vs reranking
- Evaluate retrieval quality: NDCG, MRR, Hit@K
- Build a simple GraphRAG pipeline on a knowledge-rich dataset

Self-Explanation Checkpoint

- BM25 gives high scores to documents with rare terms that appear many times. Why is that the right behavior for retrieval? When does it fail?
- HyDE generates a hypothetical answer before searching. Why does this help dense retrieval specifically?
- Your hybrid retrieval system uses RRF to combine BM25 and dense scores. Walk through the RRF formula and explain what it's doing.

Paper

- Read + reproduce: "HyDE: Precise Zero-Shot Dense Retrieval without Relevance Labels" — Gao et al.
- Secondary: "REALM: Retrieval-Augmented Language Model Pre-Training" (concept level)

Failure Friday

- Analyze: A production RAG failure traced to chunking decisions — hallucinations caused by split context

Interview Mapping

- "Compare BM25 and dense retrieval — when would you use each?"
- "What is HyDE and why does it improve retrieval?"
- "How would you evaluate your retrieval system without labeled data?"

Week 13 — Agent Engineering

Why this week exists: Agents are where AI engineering gets hard. A single LLM call is easy. An agent that operates over multiple steps, uses tools, recovers from failures, and knows when to ask a human — that is a systems problem. This week is proprietary to this curriculum.

Capstone connection: The Tern Coding Agent built this week is directly productionized in Week 14 and red-teamed in Week 15.

Productive Failure Trigger

Show students the original AutoGPT running on a simple task: "Research the top 5 competitors of Company X and write a report." Watch it loop. It searches the same query 7 times. It hallucinates a result it can't find. It writes a report based on its own previous (wrong) output. It runs for 40 minutes and produces nothing useful. Question: "The model is capable. The tools work. What's wrong with the architecture?" Then introduce the PRAOR loop, error detection, and HITL as the inventions.

Retrieval Warmup

- What is HyDE? Walk through the full retrieval flow with it.
- What is ColBERT? How does late interaction differ from bi-encoder retrieval?
- What is Corrective RAG and when does it help?

Theory

Agent Foundations

- What is an agent? (Perceive → Reason → Act → Observe → Repeat)
- The PRAOR loop in detail
- Environment types — fully observable vs partial, deterministic vs stochastic
- Tool use as the fundamental primitive
- Why agents fail: compounding errors, tool misuse, context loss

Agent Architectures

- Single Agent — one model, one loop
- ReAct (Reasoning + Acting) — interleaving thought and action
- Planner-Executor — separate planning and execution models
- Multi-agent systems — why and when to use them
- Hierarchical agents — manager/worker patterns
- Critic agents — separate model for self-evaluation
- Debate frameworks — adversarial agent pairs

Dynamic Workflows

- Static vs dynamic task decomposition
- Conditional branching in agent loops
- Parallel agent execution
- Agent handoff patterns
- Event-driven agent triggers

Human-in-the-Loop (HITL)

- When to require human approval — the permission model
- Interrupt and resume patterns
- Audit trails for agent actions
- Trust and escalation design
- UX for AI products — how users actually interact with agents

Long-Running Agents

- Durable execution — surviving process restarts

- Temporal workflows — durable task execution engine
- Checkpointing agent state
- Idempotency in agent actions

Skills Architecture

- Skill abstraction — composable, reusable agent capabilities
- /detective — information gathering skill
- /scout — research and exploration skill
- /council — multi-perspective reasoning skill
- Building your own skill library

Human-Computer Interaction for AI

- Trust in AI systems — how users build and lose trust
- Explainability — can you explain what the agent did and why?
- Escalation design — when and how to surface agent uncertainty
- Studying Cursor, Claude Code, Perplexity — why users trust them

Product Engineering

- "Who is the user of this agent? What job are they trying to do?"
- Activation: when does the user first get value?
- Retention: why do they come back?
- The hardest product question for agents: how do you know it worked?

Project — Tern Coding Agent

- ReAct loop with tool use (file read/write, bash execution, web search)
- HITL approval for destructive actions (file deletion, git push)
- Durable execution with Temporal
- Full audit trail — every action logged with timestamp and reasoning
- Skill system: /planner, /executor, /critic
- Interview 2 engineers who would use this tool — document what they need

Self-Explanation Checkpoint

- AutoGPT collapsed because of compounding errors. Walk through exactly how an error in step 3 caused step 7 to fail in the original system.
- Your agent called the wrong tool 3 times in a row. What does your PRAOR loop do about it? Be specific about the Observe step.
- Why does your agent need durable execution? What specific failure mode does Temporal prevent?

Paper

- Read + reproduce: "ReAct: Synergizing Reasoning and Acting in Language Models" — Yao et al.

Failure Friday

- Analyze: AutoGPT — why it collapsed in production; compounding error analysis

Interview Mapping

- "Design a multi-agent system for code review — how do you prevent cascading failures?"
- "What is durable execution and when do you need it?"
- "Walk me through the PRAOR loop and where agents fail at each stage"

BLOCK E — Evaluation + Reliability + Security

Week 14 — Evaluation & Reliability Engineering

Why this week exists: Most AI engineers can build. Very few can make it reliable. This is the skill that gets you senior roles and keeps you employed. A system that works in a demo but fails in production is not a system.

Capstone connection: Every capstone track must have a deployed eval harness, cost dashboard, and monitoring system. This week builds all of it.

Productive Failure Trigger

Students deployed their Week 13 agent to production (simulated). Two days later, a prompt change was pushed. Response quality dropped 30%. Nobody noticed for 48 hours because there was no monitoring. Question: "You pushed a change. Users got worse results. You didn't know for 2 days. What should have caught this?" They list manual testing. Then introduce regression evals, LLM-as-judge pipelines, and canary deployments as the inventions.

Retrieval Warmup

- What is the PRAOR loop? Where does it most commonly fail?
- What is durable execution? What does Temporal guarantee?
- What is HITL and when is it required?

Theory

Evaluation Systems

- Why evaluation is the hardest part of AI engineering
- Golden dataset construction — what makes a good eval set
- Slice testing — evaluating on demographic, topic, and difficulty slices
- LLM-as-judge — design, calibration, bias sources, when it fails
- Reference-based vs reference-free evaluation
- BLEU, ROUGE, BERTScore — and their limitations
- Regression testing — never break what works
- Eval-driven development — write evals before features

Observability

- Tracing — LangFuse, Braintrust, custom implementations
- What to log: inputs, outputs, latency, cost, model version, retrieval docs, tool calls
- Structured logging for AI systems
- Distributed tracing for multi-step agent pipelines

Cost Tracking

- Per-query cost breakdown: embedding, retrieval, generation
- Cost anomaly detection
- Budget guardrails
- Cost vs quality Pareto frontier

Deployment Strategies

- Canary deployments — send X% of traffic to new model
- Shadow mode — run new model in parallel without serving output
- A/B testing LLM responses — statistical significance
- Feature flags for AI behavior
- Blue-green deployments

Guardrails

- Input validation — prompt injection detection
- Output validation — format, length, content
- Topical guardrails — keeping the model on-task

- Toxicity and safety classifiers
- Structured output validation with Pydantic
- Guardrails AI, NeMo Guardrails (concept level)

Reliability Patterns

- Retry logic with exponential backoff
- Circuit breakers
- Timeout management
- Fallback strategies — degraded but functional
- SLA definition for AI systems

Project

- Productionize the Week 13 agent:
- Build full eval harness with 50-item golden dataset
- LLM-as-judge with calibration (measure judge agreement with humans)
- Canary deployment with traffic splitting
- Cost dashboard per request
- Tracing with LangFuse
- Guardrails for prompt injection and off-topic detection
- Alerting on quality regression

Self-Explanation Checkpoint

- Your golden dataset has 50 examples. A stakeholder asks "is 50 enough?" What do you say and why?
- Your LLM-as-judge disagrees with human raters 30% of the time. What do you do?
- A canary deployment shows worse performance. At what threshold do you roll back? How did you decide?

Paper

- Read: "Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference" — Zheng et al.

Failure Friday

- Analyze: A prompt injection attack on a production RAG system — how it worked, how to prevent it

Interview Mapping

- "Design an evaluation system for a multi-agent coding assistant"
- "How would you implement canary deployment for an LLM-based feature?"
- "What is LLM-as-judge? What are its failure modes?"

Week 15 — Security & Distributed Systems

Why this week exists: Enterprise AI is blocked on security. Scalable AI is blocked on distributed systems. Both are required for senior roles. Neither is taught in most AI courses.

Capstone connection: The red team eval suite built this week becomes part of the capstone deliverable. Distributed systems knowledge is tested in system design interviews at every top company.

Productive Failure Trigger

Give students their own RAG agent from Week 13. Tell them a "user" has been interacting with it. Show them the conversation: the user gradually convinced the agent to ignore its system prompt, exfiltrate documents from the knowledge base, and summarize private data to an external URL — all through normal-looking conversational messages. Question: "The system prompt said not to do this. The guardrails were on. How did this happen?" Then introduce indirect prompt injection and defense-in-depth as the inventions.

Retrieval Warmup

- What is a canary deployment? At what point do you promote or roll back?
- What is shadow mode testing and when do you use it over canary?
- What does eval-driven development mean in practice?

Theory

AI Security

- Threat modeling for AI systems — attack surfaces
- Prompt injection — direct, indirect, multi-turn
- Jailbreaking — techniques and defenses
- Data exfiltration via prompt injection
- RAG security — poisoning the knowledge base
- MCP (Model Context Protocol) security — tool permission boundaries
- Agent permission systems — principle of least privilege
- Adversarial examples (concept level)
- Model extraction attacks

Red Teaming

- Red team methodology for AI systems
- Automated red teaming — using a model to attack a model
- Test cases: role confusion, instruction override, context manipulation
- Building a red team eval suite
- Responsible disclosure in AI

Distributed Systems (Full Block)

- CAP theorem — deeper: network partitions are not optional
- Consistency models: strong, eventual, causal, read-your-writes
- Consensus — the problem of agreement in distributed systems
- Raft — leader election, log replication, safety guarantees
- Replication — master-replica, multi-master
- Sharding — consistent hashing, hot spots
- Event-driven architecture — events as the source of truth
- Message queues — Kafka, RabbitMQ, SQS — tradeoffs
- Saga pattern — distributed transactions in microservices
- Idempotency — designing operations that are safe to retry

Case Studies

- OpenAI serving architecture — how they handle millions of requests
- Stripe payments — idempotency keys, exactly-once semantics
- Uber dispatch — real-time matching at scale

Project

- Red-team your own agent from Week 13:
- Document 5+ successful attacks (prompt injection, jailbreak, data exfiltration)
- Implement defenses for each attack
- Build automated red team eval suite that runs on every deploy
- Implement a simplified distributed key-value store using Raft
- Design (on paper + ADR) the architecture for an AI system handling 10M daily requests

Self-Explanation Checkpoint

- Walk through exactly how the indirect prompt injection attack on your system worked. What chain of events led to the exfiltration?
- In Raft, what happens when a leader fails? Walk through the election process step by step.
- Your AI system needs to handle 10M daily requests. Walk through every architectural decision you'd make and the tradeoff at each.

Paper

- Read: "Universal and Transferable Adversarial Attacks on Aligned Language Models" — Zou et al.

Failure Friday

- Analyze: A real prompt injection attack in production — how the system was compromised, full post-mortem

Interview Mapping

- "What is indirect prompt injection and how do you defend against it?"
- "Explain the Raft consensus algorithm"
- "Design the serving infrastructure for an LLM API handling 1M requests/day"

BLOCK F — RLHF + Agentic RL + Frontier Research

Week 16 — Foundational RL + Post-Training: RLHF, DPO & Alignment

Why this week exists: PPO, GRPO, and DPO are built on top of foundational RL concepts. Engineers who don't know Bellman equations, the Policy Gradient Theorem, and Q-learning cannot reason about why RLHF pipelines fail or succeed. This week teaches both the foundation and the application.

Capstone connection: Research Engineer track students will be asked Bellman equations and Policy Gradient derivations in interviews. Applied AI students need RLHF intuition to reason about model behavior.

Productive Failure Trigger

Show students an RLHF-trained model that has been reward hacked. It learned to generate very long, confident-sounding responses that score high with the reward model — even when the answer is wrong. The reward model gave it 9.2/10. Humans give it 4/10. Question: "The reward model is a trained neural network. The agent optimized it. Why did this produce bad outputs?" Then introduce KL penalties, reward model limitations, and RLVR as the inventions.

Retrieval Warmup

- What is the Saga pattern in distributed systems?
- Walk through a prompt injection attack. What made your defense work?
- What is the difference between strong consistency and eventual consistency?

Theory

Foundational Reinforcement Learning

- What is the RL problem: agent, environment, state, action, reward, policy
- Markov Decision Process (MDP) — formal definition
- Markov Property — why it matters for tractability
- Bellman Equations — the recursive value decomposition (full derivation)
- Bellman expectation equation: $V(s) = E[r + \gamma V(s')]$
- Bellman optimality equation
- Q-Learning — off-policy TD learning; the full update rule
- TD Learning vs Monte Carlo — variance-bias tradeoff
- SARSA — on-policy TD; how it differs from Q-learning

- Policy Gradient Theorem — deriving the gradient of expected return
- Actor-Critic methods — combining value estimation and policy optimization
- A2C/A3C — advantage actor-critic; the advantage function
- Soft Actor-Critic (SAC) — maximum entropy RL; why entropy regularization helps exploration
- Generalized Advantage Estimation (GAE) — reducing variance in policy gradients
- On-Policy vs Off-Policy — formal definition and implications
- Exploration vs Exploitation — epsilon-greedy, UCB, Thompson sampling
- Credit assignment problem — how do you attribute a reward to an action 50 steps earlier?
- Model-Based vs Model-Free RL — tradeoffs
- World Models and Dreamer — learning a model of the environment
- MuZero and AlphaGo — planning in learned models
- Curriculum learning — ordering training examples by difficulty
- Importance Sampling — correcting for distribution shift in off-policy learning

Supervised Fine-Tuning (SFT)

- What SFT does: teaching format, instruction following, style
- Dataset curation for SFT — quality over quantity
- Instruction datasets: Alpaca, ShareGPT, OpenHermes
- Catastrophic forgetting and how to mitigate it
- Evaluation of SFT models

Reinforcement Learning from Human Feedback (RLHF)

- Why pretrained LLMs aren't aligned by default
- Reward model training — Bradley-Terry model, pairwise comparisons
- PPO (Proximal Policy Optimization) for language models — connecting to Policy Gradient Theorem
- The InstructGPT pipeline: SFT → RM → PPO
- KL penalty — why you need it, what happens without it
- Reward hacking — how models game the reward signal

Direct Preference Optimization (DPO)

- The insight: remove the reward model from the loop
- DPO derivation from RLHF objective
- DPO vs RLHF — pros, cons, when to use which
- SimPO, IPO — DPO variants

Group Relative Policy Optimization (GRPO)

- The key innovation: group-relative advantage estimation
- Connection to GAE and Actor-Critic methods
- Why it's more stable than PPO for reasoning tasks
- DeepSeek's use of GRPO — implementation walkthrough

Reinforcement Learning from Verifiable Rewards (RLVR)

- Using rule-based rewards instead of human labels
- Math and code as verifiable domains
- DeepSeek R1 approach — chain-of-thought as a learned behavior
- Why this might scale better than human feedback

Constitutional AI & RLAIF

- Anthropic's Constitutional AI (CAI) approach
- RLAIF — using AI feedback instead of human feedback
- Self-critique and revision loops

Project

- Implement Q-learning from scratch on a simple grid environment
- Implement Policy Gradient (REINFORCE) on CartPole
- Train a small reward model on pairwise preference data
- Implement GRPO on a math reasoning dataset
- Fine-tune with DPO on a preference dataset
- Compare: base model vs SFT vs DPO on alignment benchmark

Self-Explanation Checkpoint

- Derive the Bellman equation from first principles. What assumption makes it recursive?
- Why does PPO need a KL penalty? Walk through what happens to the policy without it over 1000 steps.
- GRPO vs PPO: what specifically does GRPO change and why does that matter for language model training?

Papers

- Read: "Training Language Models to Follow Instructions with Human Feedback" — InstructGPT
- Read: "Direct Preference Optimization" — Rafailov et al.
- Read: "DeepSeek R1" — GRPO section in detail

Failure Friday

- Analyze: A reward-hacked model — what behaviors emerged when the reward model was gamed?

Interview Mapping

- "Derive the Bellman equation from scratch"
- "What is the Policy Gradient Theorem?"
- "Derive DPO from first principles"
- "What is reward hacking and how do you detect it?"
- "Compare PPO vs GRPO — when would you choose each?"
- "What is GAE and why does it reduce variance?"

Week 17 — Reasoning Models, Agentic RL, Generative Modeling & Research Thinking

Why this week exists: Frontier knowledge. This week covers the areas where interviews at DeepMind, Anthropic, and OpenAI go deepest — reasoning models, generative modeling (GANs, VAEs, diffusion), and research thinking. Not everyone needs all of it equally, but everyone needs exposure.

Capstone connection: Frontier Labs project starts here and delivers in Week 18.

Productive Failure Trigger

Show students a base language model (before RLHF) attempting a math problem. It gets the answer wrong in a single step. Show the same model after GRPO training on math — it now writes out a long reasoning chain and gets the right answer. Question: "The weights changed, but the architecture is identical. What specifically did training teach it to do differently?" Then introduce process vs outcome rewards and test-time compute scaling as the inventions.

Retrieval Warmup

- What is the credit assignment problem in RL? How does GAE address it?
- What is the difference between on-policy and off-policy learning?
- What is Constitutional AI? How does it differ from RLHF?

Theory

Reasoning Models

- Chain-of-Thought as emergent behavior vs trained behavior

- Trained CoT vs prompted CoT — the difference
- Process reward models (PRMs) vs outcome reward models (ORMs)
- o1-style reasoning — search over thought space
- Test-time compute scaling — why more inference compute helps
- Majority voting, best-of-N sampling
- Monte Carlo Tree Search (MCTS) for language models

Tool Use RL

- Training agents to use tools through RL
- Tool use as a learnable skill — not just prompted
- Code execution as a verifiable environment
- Toolformer — the original tool use paper

Agentic RL

- Multi-step RL in agent environments
- Credit assignment over long trajectories
- Exploration in agent RL — UCB adapted for LLMs
- Curriculum learning for agents
- Multi-agent RL — cooperation and competition

Generative Modeling *(Required for Research Engineer track; important exposure for all students)*

- Generative Adversarial Networks (GANs)
- Generator and discriminator — the minimax game
- Training instability and mode collapse
- WGAN — Wasserstein distance as a more stable objective
- StyleGAN, BigGAN — modern GAN architectures
- Variational Autoencoders (VAEs)
- The VAE objective: reconstruction + KL penalty
- ELBO — Evidence Lower Bound; full derivation
- The reparameterization trick — why it enables backprop through sampling
- Latent space structure and interpolation
- VQ-VAE — discrete latent spaces
- Score Functions and Diffusion Models
- Score function — the gradient of the log-density
- Score matching — training a network to estimate the score
- Diffusion forward process — adding noise (DDPM)
- Diffusion reverse process — denoising (DDPM vs DDIM)
- Diffusion as SDEs — stochastic differential equations formulation
- Flow Matching — ODE-based generative models; cleaner than diffusion
- Classifier-Free Guidance (CFG) — steering generation without a separate classifier
- Stable Diffusion architecture — VAE + UNet + CLIP

Scientific Thinking (Full Block)

- Experiment design for ML research: hypothesis, IV, DV, controls, confounders
- Ablation studies — isolating variables
- Statistical significance in ML experiments
- Reading results tables critically
- The replication crisis in ML — what it means for your work
- "How do I know this works?" as the fundamental research question

Research Taste (Advanced)

- Which papers actually changed the field:
- Attention Is All You Need — architecture revolution
- Chinchilla — scaling law revision
- FlashAttention — systems innovation
- LoRA — efficiency revolution
- InstructGPT — alignment breakthrough
- DPO — simplifying alignment
- DeepSeek R1 — reasoning via RL
- DDPM — making diffusion practical
- Papers that were hyped but didn't stick — and why
- How to read results tables critically — benchmark overfitting

Frontier Research: Emerging Areas

- World models — predicting future states of the environment
- Multimodal models — vision-language, video understanding
- Long context — 1M token models and their applications
- Mechanistic interpretability — understanding what models actually do
- Agent foundations — formal theories of agency

Building Original Things

- Observe a real problem → invent an abstraction → build a system
- Examples: ReAct, Toolformer, DSPy, Cursor Tab, Claude Artifacts — original abstractions
- The Frontier Labs project below is this in practice

Project — Frontier Labs

Students must:

1. Find a real problem no tutorial addresses
2. Read 10 relevant papers
3. Propose a new abstraction or approach
4. Build a prototype
5. Write a research memo
6. Present findings with a live demo

Project options:

- Train a math reasoning model using GRPO — implement from scratch
- Build a process reward model for code generation
- Implement a simplified o1-style reasoning loop with tree search
- Implement VAE + reparameterization trick; explore latent space structure
- Implement a simplified DDPM — train on MNIST, visualize denoising steps
- Implement Flow Matching — compare to DDPM on the same task
- Novel evaluation methodology for multi-step agents

Self-Explanation Checkpoint

- What is the reparameterization trick in VAEs? Why can't you just sample directly and backpropagate?
- What is the difference between DDPM and DDIM? What does DDIM sacrifice and what does it gain?
- PRMs vs ORMs: when does each fail? Give a concrete example of a task where ORMs are not enough.

Papers

- Read: "DeepSeek R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning"
- Read: "Denoising Diffusion Probabilistic Models" — Ho et al.

- Read: "Toolformer: Language Models Can Teach Themselves to Use Tools"
- Secondary: "Flow Matching for Generative Modeling" — Lipman et al.

Failure Friday

- Analyze: A reasoning model that developed reward hacking in chain-of-thought — generating plausible-sounding but wrong reasoning steps

Interview Mapping

- "What is test-time compute scaling and why does it matter?"
- "Derive the VAE ELBO from scratch"
- "What is the reparameterization trick?"
- "Explain the diffusion forward and reverse processes"
- "What is classifier-free guidance and how does it work?"
- "Compare process reward models vs outcome reward models"

BLOCK G — Production Capstone + Hiring Sprint

Week 18 — Capstone: Production AI System + Hiring Sprint

Why this week exists: Everything in the previous 17 weeks was building toward this. The capstone is not a school project. It is a production system that must be deployed, evaluated, monitored, and demoed live. It is the proof of work that gets you hired.

Productive Failure Trigger

Week 18 opens with each student demoing their Week 17 Frontier Labs prototype to the cohort — in its current broken, incomplete state. Publicly. The point: shipping imperfect work and defending it is a skill. Most engineers never practice this before their first job.

Retrieval Warmup

- Derive the ELBO in a VAE.
- What is classifier-free guidance? How does it steer generation?
- What is the Bellman optimality equation?

Theory

AI-Native Systems Design

- Architecture Decision Records (ADRs) — how to document decisions that will be read by engineers who weren't in the room
- System design for AI products — unique considerations vs traditional systems
- AI product design — onboarding, trust, explainability, escalation
- Capstone system design principles

Product Engineering (Full Block)

- Customer discovery — problem interviews, JTBD (Jobs to Be Done)
- User observation — watching someone use your product tells you more than asking them
- Product metrics — activation, retention, engagement, NPS
- Unit economics for AI products — cost per query, margin
- A/B testing AI features — measurement and statistical significance
- Feature prioritization — ICE, RICE, impact vs effort
- The question before every feature: "Who needs this? What decision does it help them make? How will we know it worked?"

AI Product Intuition

- Why Cursor won — UX, latency, product-led trust, tab completion as a paradigm
- Why Perplexity won — search UX, citation trust, answer confidence calibration
- Why Claude Code won — ergonomics, safety, tool use depth, memory
- Why Linear beats Jira — speed, keyboard-first, opinionated defaults
- Critique framework: What problem does it solve? What tradeoff did they make? What would you do differently?

Distribution (Full Block)

- Technical writing — blog posts that demonstrate depth, not just breadth
- Building in public — GitHub, X, LinkedIn as proof-of-work portfolios
- Demos that sell — the art of showing a system to someone who doesn't understand it
- OSS contribution as a hiring signal — one merged PR teaches more than 20 certificates

Full-Stack AI Development

- End-to-end product: Frontend + Backend + DB + Agent + Deployment
- AI coding systems at full speed: Claude Code + Cursor + Codex as daily defaults
- Code review of AI-generated code — the skill of knowing what to trust
- Architecture with AI — prompting for system design, not just implementation

Hiring Sprint — Logistics

- **Timing strategy:** Do not start company A's test until company B has scheduled their first interview. The goal is offers landing in the same 2-week window — this is what creates real leverage.
- **Start with companies you care less about.** Calibrate your confidence and salary expectations before negotiating for your top choices.
- **Tell every company about your other processes.** This is normal and expected. It speeds up timelines and signals you are a serious candidate.
- **Mock interview methodology:** Before each interview, paste the role description + company into Claude and ask it to interview you. The overlap with real questions is significant. Do this for every single interview.
- **Flashcard discipline:** Write your own cards — writing them is half the learning. Do not download someone else's deck. When reviewing, ask yourself questions, don't just read answers.
- **Negotiation mechanics:**
- RSUs vs stock options — understand the difference before you sign anything
- Companies can move their numbers significantly if they want you. Always ask.
- Recruiters read you. Small signals — how often you mention a company, your tone — get noted. Be careful.
- Competing offers from actual peers (top labs, top-funded startups) carry weight. An offer from an unknown startup does not move a frontier lab's number.
- Companies have historical data on which offers candidates actually take. Your bluff doesn't work if they know the base rate.
- **Emotional preparation:**
- This process is stochastic. Getting rejected does not mean you are a bad engineer.
- Messing up a question you know in an interview does not mean you don't know it.
- Design a pre-interview ritual. Consistent routine reduces anxiety.
- Your worth as a person is not being decided by these interviews.

Capstone Tracks (Student picks one)

Track A — AI SWE

- PR Review Agent
- Full repo analysis, multi-file changes, diff comprehension, comment generation

Track B — FDE (Forward Deployed Engineer)

- Hospital Agent: patient intake, diagnosis support, document processing

- Legal Agent: contract review, clause extraction, compliance checking
- Supply Chain Agent: anomaly detection, root cause analysis, remediation suggestions

Track C — Applied AI

- Research Agent: paper summarization, hypothesis generation, experiment tracking
- Sales Agent: lead research, outreach personalization, CRM integration
- Knowledge Agent: internal knowledge base with multi-hop reasoning

Track D — AI Infrastructure

- Evaluation Platform: golden datasets, LLM-as-judge, regression testing, dashboards
- Agent Observability Platform: tracing, cost tracking, failure analysis
- LLM Gateway: routing, caching, cost optimization, fallbacks

Deliverables (All Tracks)

Every student must deliver:

- ADRs documenting every major technical decision (minimum 5)
- Full system design document
- Eval harness with golden dataset and LLM-as-judge
- Deployed production system — not localhost; must have a public URL
- Monitoring and cost dashboard
- Postmortem: what failed during development and why
- Live demo — 20 minutes: problem → who needs it → solution → architecture → demo → eval results → cost
- Public write-up: LinkedIn post, technical blog, or X thread

Self-Explanation Checkpoint (Final)

Before the demo, answer in writing:

- What is the user's job-to-be-done? How do you know? What evidence do you have?
- What are the three most important architectural decisions you made? What did you reject and why?
- How do you know the system works? Walk through your eval methodology.
- What failed during development? What would you do differently?

Papers

- Read: "LLM-as-a-Judge" — Zheng et al. (for capstone eval harness)

Failure Friday (Final)

- Each student analyzes one failure from their own capstone build — postmortem format: what happened, why, what they'd do differently

Interview Mapping

- "Walk me through your capstone system design"
- "How did you evaluate your system? How do you know it works?"
- "What failed? What would you do differently?"
- "Show me a production deployment"
- "Who is the user? What is their JTBD?"

Horizontal Tracks (All 18 Weeks)

AI Coding Systems Track

Every week, every student must use these tools — not as a bonus, but as the default:

- **Claude Code** — agentic coding, architecture with AI, debugging
- **Cursor** — AI-native IDE, tab completion, multi-file context
- **Codex / GitHub Copilot** — completion and explanation
- **Gemini CLI** — command line AI workflows
- **OpenHands** — autonomous coding agent

Skills built across 18 weeks:

- Prompting for code architecture, not just snippets
- Reviewing and debugging AI-generated code
- Skill systems — writing reusable prompts for recurring tasks
- Prompt-to-production workflows
- Knowing when to trust AI output and when to verify independently

Startup Operator Track

Every week, every student must:

Activity	Cadence
Ship one working thing	Weekly
Talk to one real user about a problem	Weekly
Write one technical post (blog, LinkedIn, X)	Weekly
Contribute to an OSS project	Monthly
Demo publicly	Every two weeks
Maintain a portfolio of deployed AI products	Ongoing

Target OSS contributions:

- LangGraph
- vLLM
- LiteLLM
- OpenHands
- LlamaIndex
- DSPy
- PydanticAI
- Haystack
- CrewAI

Paper Club (All 18 Weeks)

The canonical reading list:

Week	Paper
1	A Few Useful Things to Know About ML — Domingos
2	micrograd — Karpathy
3	Deep Learning — LeCun, Bengio, Hinton (Nature, 2015)
4	Gradient-Based Learning Applied to Document Recognition — LeCun (1998)
5	Neural Machine Translation by Jointly Learning to Align and Translate — Bahdanau
5	Mamba: Linear-Time Sequence Modeling with Selective State Spaces
6	Attention Is All You Need — Vaswani et al.
7	LoRA: Low-Rank Adaptation of Large Language Models — Hu et al.

Week	Paper
8	FlashAttention — Dao et al.
9	Designing Data-Intensive Applications Ch. 1 — Kleppmann
10	Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks — Lewis et al.
11	Chain-of-Thought Prompting Elicits Reasoning — Wei et al.
12	HyDE: Precise Zero-Shot Dense Retrieval — Gao et al.
13	ReAct: Synergizing Reasoning and Acting — Yao et al.
14	Chatbot Arena: Evaluating LLMs by Human Preference — Zheng et al.
15	Universal Adversarial Attacks on Aligned LLMs — Zou et al.
16	Training Language Models to Follow Instructions — InstructGPT
16	Direct Preference Optimization — Rafailov et al.
17	DeepSeek R1
17	Denoising Diffusion Probabilistic Models — Ho et al.
17	Toolformer — Schick et al.
18	LLM-as-a-Judge — Zheng et al.

Every paper follows this loop:

```
Read
  ↓
Critique (what did they get right? what's missing? what aged poorly?)
  ↓
Reproduce key idea (not the full paper — the core insight, in code)
  ↓
Present (5 minutes to the cohort — defend your critique)
```

Failure Friday (All 18 Weeks)

Every Friday, analyze one real AI failure:

Week	Failure
1	Microsoft Tay
2	Training divergence due to LR misconfiguration
3	Silent model failure in production — evaluation gap
4	Gradient explosion — undetected in production
5	Tokenization failure in non-English deployment
6	LLM hallucination — failure mode at the probability level
7	Inference cost blowup — no batching or quantization
8	10x slower training run — OOM fragmentation, wrong parallelism
9	Production AI API down under load
10	RAG failure traced to silent embedding pipeline failures
11	Chatbot losing context in long conversations
12	RAG hallucination from split context
13	AutoGPT collapse — compounding error analysis
14	Prompt injection attack on production RAG
15	Real prompt injection exploit — full post-mortem
16	Reward hacked model — unexpected behavior in production
17	Reasoning model gaming chain-of-thought

Week	Failure
18	Student's own capstone failure — postmortem format

Analysis format (every week):

1. What happened?
2. What failed technically?
3. What monitoring would have caught it?
4. What would you do differently?

Engineering Judgment (All 18 Weeks)

Every week, answer these four questions in writing about one real decision:

1. What would you do?
2. Why?
3. What tradeoff are you making?
4. What alternative did you reject and why?

Week	Judgment Question
1	Frequentist or Bayesian approach for this classification problem?
2	Adam or SGD for this task? Defend it with the math.
3	XGBoost or neural network for this tabular dataset?
4	BatchNorm or LayerNorm? RMSNorm? Why?
5	S4 or transformer for a 10K-token sequence task?
6	Dense model or MoE for this use case? What are the inference tradeoffs?
7	Fine-tune or few-shot prompt? Walk through the decision framework.
8	FSDP or ZeRO stage 3 for this model size and GPU count?
9	Postgres or Redis for session storage in this system?
10	Kafka or SQS for this pipeline?
11	Fine-tune or RAG for this use case?
12	BM25, dense, or hybrid retrieval?
13	Multi-agent or single-agent architecture for this task?
14	LLM-as-judge or golden dataset eval for this system?
15	Canary or blue-green for this deployment?
16	DPO or RLHF for this preference task?
17	PRMs or ORMs for this reasoning task?
18	Build vs buy: host your own model or use the API at this scale?

Curriculum Map

```
BLOCK A — Math + ML Foundations (Weeks 1–4)
■■■ Week 1: Probability, Statistics, Information Theory (+ Jensen-Shannon Divergence)
■■■ Week 2: Linear Algebra, Calculus, Optimization (+ PSD matrices, Newton's Method)
■■■ Week 3: Classical ML from Scratch (+ Gumbel-Softmax, Domain Adaptation)
■■■ Week 4: Deep Learning Foundations (+ Gradient Checkpointing, Numerical Precision)

BLOCK B — Transformers + LLM Internals (Weeks 5–7)
■■■ Week 5: Sequence Models + S4/SSMs + Griffin + TransformerXL + Road to Transformers
■■■ Week 6: Transformer Architecture Deep Dive (+ MoE, Perceiver, Sinusoidal vs RoPE)
■■■ Week 7: Inference, Quantization, Fine-Tuning (+ FSDP, JIT, torch.compile)

BLOCK C — Systems + Backend + Data Engineering (Weeks 8–10)
■■■ Week 8: GPU Architecture + FSDP + Numerical Precision + Flash Attention
■■■ Week 9: Backend Engineering for AI Products (+ Product Engineering intro)
■■■ Week 10: Data Engineering (+ Embedding Versioning, Vector DB Internals)

BLOCK D — LLM Engineering + RAG + Agents (Weeks 11–13)
■■■ Week 11: LLM Engineering + Context Engineering (+ JTBD, Product Engineering)
■■■ Week 12: Retrieval Science + Advanced RAG (+ ColBERT, GraphRAG)
■■■ Week 13: Agent Engineering (+ Product Engineering, User Research)

BLOCK E — Evaluation + Reliability + Security (Weeks 14–15)
■■■ Week 14: Evaluation + Reliability Engineering
■■■ Week 15: Security + Red Teaming + Distributed Systems (full block)

BLOCK F — RLHF + Agentic RL + Frontier Research (Weeks 16–17)
■■■ Week 16: Foundational RL (Bellman, Q-Learning, Policy Gradient, SAC, GAE)
■            + Post-Training (RLHF, DPO, GRPO, RLVR, Constitutional AI)
■■■ Week 17: Reasoning Models + Agentic RL
             + Generative Modeling (GANs, VAEs, Diffusion, Flow Matching)
             + Research Thinking

BLOCK G — Production Capstone (Week 18)
■■■ Week 18: Capstone System + Hiring Sprint (+ Full Hiring Logistics Framework)

HORIZONTAL TRACKS (All 18 Weeks)
■■■ AI Coding Systems (Claude Code, Cursor, Codex daily)
■■■ Startup Operator Track (ship, talk to users, write, OSS)
■■■ Paper Club (read → critique → reproduce → present weekly)
■■■ Failure Friday (analyze real AI failures)
■■■ Engineering Judgment (4 questions every week)
```

What Good Looks Like at Graduation

On a whiteboard:

- Derive attention from scratch
- Explain KV cache, RoPE, SwiGLU, GQA without notes
- Explain MoE routing and the load balancing problem
- Design a production RAG system with proper retrieval science
- Architect a multi-agent system and explain failure modes at each stage
- Explain RLHF → DPO → GRPO progression and tradeoffs
- Derive the Bellman equation; explain the Policy Gradient Theorem
- Explain VAE ELBO derivation and the reparameterization trick
- Explain diffusion forward and reverse processes
- Roofline analysis for a transformer workload
- CAP theorem applied to a real AI system

In code:

- Ship a full-stack AI product solo (frontend + backend + DB + agent + deployment)

- Build and evaluate a RAG pipeline that measurably outperforms baseline
- Run a LoRA fine-tune and evaluate it with a proper eval harness
- Write a working GRPO training loop on a math dataset
- Implement Flash Attention in Triton
- Deploy and monitor an AI system in production with cost tracking

As a portfolio:

- 18 weeks of LinkedIn/X posts documenting learning in public
- Deployed capstone with live demo link
- At least one merged OSS PR
- 3–5 technical blog posts or architecture write-ups
- GitHub with clean, documented, deployed projects

In an interview:

- Answer "design a system like Perplexity" end-to-end with tradeoff analysis
- Defend every architectural choice with explicit alternatives rejected
- Demonstrate research taste: "This paper mattered because… and here's what it got wrong"
- Show failure analysis from their own builds — postmortem format

This curriculum is designed for 2026–2030 relevance. Models will change. Frameworks will change. The underlying skills — systems thinking, engineering judgment, retrieval science, evaluation discipline, research taste, and the ability to ship — will not.