Industry-Ready Applied Generative AI

3-Month Intensive Practical Crash Course Curriculum (72–80 Hours) for Undergraduate Engineers

Target Audience: B.Tech / B.E. Computer Science, IT, AI & Data Science Students	**Total Duration:** 12 Weeks / 36 Lectures / 72 Hours (2 hrs/lecture)
Pedagogy: 75% Hands-on Coding (Colab/Laptop), 25% High-Level Intuition	**Key Modern Focus:** TurboVec, MTP, dFlash, ColBERT, Coding Harnesses, SLMs, VLM OCR

1. Course Philosophy & Pedagogical Strategy

This curriculum is specifically engineered to bridge the gap between academic computer science education and the rapidly evolving requirements of the GenAI job market. Modern production AI relies heavily on practical software harness integration, efficient vector search engines, multi-token prediction architectures, lightweight open-weight models, and continuous fine-tuning.

> **Key Course Design Principles:**
> • Pragmatic Depth: Avoid pure research-level proofs. Teach the intuitive mathematics behind self-attention, loss functions, and tokenization, then immediately transition to code implementation.
> • Code-First After Week 1: From Lecture 4 onwards, 75% of class time is spent in Google Colab / local VS Code environments.
> • Cutting-Edge Modernity: Integrates latest industry breakthroughs including TurboVec, Multi-Token Prediction (MTP), dFlash attention, ColBERT late interaction, Vision-Language OCR, and CLI coding harnesses (Claude Code, OpenCode, Gemini CLI).

2. High-Level Module Matrix

Module #	Module Title	Lectures	Primary Hands-on Outcome
Module 1	Generative AI Foundations & Architecture	L01 - L04 (8h)	Custom BPE Tokenizer & PyTorch Attention Block
Module 2	Building & Training an LLM from Scratch	L05 - L08 (8h)	Working NanoGPT-style Transformer running locally/Colab
Module 3	Advanced RAG, ColBERT & TurboVec	L09 - L12 (8h)	Production RAG Pipeline with TurboVec & Multi-Vector ColBERT
Module 4	Fine-Tuning, SLMs & Vision-Language OCR	L13 - L16 (8h)	Fine-Tuned Qwen/SLM for Structured JSON Extraction
Module 5	Inference Optimization: MTP, dFlash & Quant	L17 - L20 (8h)	Deploying GGML/vLLM engine with Speculative MTP & Quantization

Module 6	Autonomous AI Agents & Tool Calling	L21 - L24 (8h)	Multi-Agent System with ReAct, Function Calling & Memory
Module 7	Developer Workflows & CLI Coding Harnesses	L25 - L28 (8h)	Automated Feature Build using Claude Code / OpenCode / Gemini CLI
Module 8	Reinforcement Learning (RLHF, DPO & GRPO)	L29 - L32 (8h)	Fine-Tuning an LLM with DPO/GRPO for Reasoning
Module 9	Capstone Project & Industry Deployment	L33 - L36 (8h)	Production-Grade AI Application Deployed with Docker & API

3. Detailed Lecture-by-Lecture Syllabus (72 Hours)

Module 1: Generative AI Foundations & Core Architectures

Lecture 01: Introduction to Generative AI & Market Landscape

- **Core Topics Covered:**
  - Evolution from Rules-based NLP to Statistical LM to Generative AI.
  - Overview of LLM Stack: Compute, Base Models, Fine-Tuning, RAG, and Agents.
  - Understanding Open-Weight vs. Closed API Ecosystem (DeepSeek, Qwen, Llama vs OpenAI, Anthropic).
  - Hardware 101: GPU VRAM sizing, FLOPS, memory bandwidth, and quantization necessity.

Hands-on Coding / Lab (Colab / Laptop): Setup Python 3.11+, PyTorch CUDA environment on Google Colab / Local GPU. Run benchmark tests on VRAM memory throughput.

Curated Learning Resources: *3Blue1Brown - But what is a neural network?* | *State of AI Report (Latest Edition)* | *Hugging Face Course: Introduction*

Lecture 02: Neural Network Prerequisite Refresher & Tensor Ops

- **Core Topics Covered:**
  - PyTorch Fundamentals: Tensors, Autograd, Backpropagation intuition.
  - Linear layers, activation functions (ReLU, GELU, SwiGLU), and LayerNorm vs RMSNorm.
  - Matrix multiplication mechanics: Shape matching, batch processing, matrix dimensions.

Hands-on Coding / Lab (Colab / Laptop): Build a simple multi-layer perceptron (MLP) from scratch in PyTorch to classify synthesized embeddings.

Curated Learning Resources: *PyTorch Official Deep Learning with PyTorch Tutorial* | *Andrej Karpathy - Micrograd Walkthrough*

Lecture 03: Tokenization Deep Dive & Modern Embeddings (ColBERT Concept)

- **Core Topics Covered:**
  - How text becomes numbers: Character, Word, Subword Tokenization (BPE, WordPiece, Unigram).
  - Tokenizers in practice: Tiktoken, SentencePiece, HuggingFace Tokenizers.
  - Dense Embeddings vs. Sparse Embeddings.
  - Introduction to ColBERT (Contextualized Late Interaction over BERT): Token-level dense representation vs single-vector pooling.

Hands-on Coding / Lab (Colab / Laptop): Build a basic Byte-Pair Encoding (BPE) tokenizer from scratch in Python. Compare output of OpenAI tiktoken vs Llama tokenizer. Implement a toy ColBERT late-interaction scoring routine (dot-product matrix between query tokens and document tokens).

Curated Learning Resources: *Karpathy - Let's build the GPT Tokenizer | ColBERT Paper: Efficient and Effective Passage Retrieval via Contextualized Late Interaction over BERT*

Lecture 04: The Transformer Architecture: Self-Attention Mechanics

- **Core Topics Covered:**
  - Query, Key, Value (Q, K, V) vector intuitive explanation.
  - Scaled Dot-Product Attention: Math, scaling factor sqrt(d_k), and Softmax.
  - Multi-Head Attention (MHA), Grouped Query Attention (GQA), and Multi-Query Attention (MQA).
  - Positional Encodings: Absolute, Sinusoidal, and Rotary Position Embeddings (RoPE).

Hands-on Coding / Lab (Colab / Laptop): Write Scaled Dot-Product Attention and Multi-Head Attention modules from scratch using raw PyTorch tensor operations.

Curated Learning Resources: *Illustrated Transformer by Jay Alammar | Attention Is All You Need (Vaswani et al.)*

Module 2: Building & Training an LLM from Scratch

Lecture 05: Building the Decoder Block & Causal Masking

- **Core Topics Covered:**
  - Decoder-only vs Encoder-Decoder vs Encoder-only architectures.
  - Causal Masking: Why upper-triangular matrix is filled with -inf.
  - Residual connections and pre-LN vs post-LN layer normalization.
  - Feed-Forward Network (FFN) with SwiGLU activation.

Hands-on Coding / Lab (Colab / Laptop): Assemble the Transformer Decoder Block in PyTorch by combining Masked MHA, RMSNorm, and SwiGLU FFN.

Curated Learning Resources: *Andrej Karpathy - Let's build GPT: from scratch, in code | Llama 2 / Llama 3 PyTorch implementation from scratch*

Lecture 06: Assembling NanoGPT & Pre-training Pipeline

- **Core Topics Covered:**
  - Complete model architecture definition: Combining Token Embedding, Positional Embedding, N-Decoder Blocks, and Final LM Head.
  - Loss Function: Cross-Entropy Loss over target tokens, Perplexity metric.
  - Data Pipeline: Tokenizing datasets (TinyStories / FineWeb-Edu subset) into memory-mapped arrays.

Hands-on Coding / Lab (Colab / Laptop): Complete full PyTorch code for a 10M parameter NanoGPT model. Implement data loader for autoregressive training.

Curated Learning Resources: *Karpathy nanoGPT GitHub repository | FineWeb-Edu Dataset on Hugging Face*

Lecture 07: Training Dynamics, Optimizers & Precision

- **Core Topics Covered:**
  - AdamW Optimizer: Learning rate schedules, warmup, and cosine decay.
  - Mixed Precision Training: FP32 vs FP16 vs BF16, GradScaler usage.
  - Gradient Clipping, Weight Decay, and preventing exploding/vanishing gradients.

Hands-on Coding / Lab (Colab / Laptop): Train the 10M NanoGPT model on TinyStories dataset for 1000 steps in Colab. Track training loss and plot learning rate curve.

Curated Learning Resources: *PyTorch Automatic Mixed Precision (AMP) Tutorial | WandB / TensorBoard Logging Guide*

Lecture 08: Autoregressive Text Generation & Decoding Strategies

- **Core Topics Covered:**
  - Generation loop: KV-Cache intuition (saving computed Keys and Values).
  - Decoding strategies: Greedy search, Beam search.
  - Stochastic sampling: Temperature scaling, Top-K sampling, Top-P (Nucleus) sampling, and Repetition Penalty.

Hands-on Coding / Lab (Colab / Laptop): Write the inference generation script from scratch. Implement KV-Cache in Python. Test text generation with varying Temperature and Top-P values.

Curated Learning Resources: *Hugging Face Blog - How to generate text: Using different decoding methods* | *Transformers KV-Cache Explanation*

Module 3: Advanced RAG, ColBERT & TurboVec

Lecture 09: RAG Architecture & Document Processing Pipeline

- **Core Topics Covered:**
  - Naive RAG vs Advanced RAG vs Modular RAG.
  - Document Ingestion: Chunking strategies (Fixed size, Recursive character, Semantic chunking).
  - Embedding Generation: Bi-encoders (OpenAI, BGE, E5).
  - Vector Databases: Index types (HNSW, IVFFlat) and similarity metrics (Cosine, L2, Inner Product).

Hands-on Coding / Lab (Colab / Laptop): Build a basic RAG pipeline using LangChain / LlamaIndex and ChromaDB to query PDF documents.

Curated Learning Resources: *Pinecone Learning Center - Vector Indexing* | *LlamaIndex Documentation on Chunking Strategies/* https://github.com/GiovanniPasq/agentic-rag-for-dummies /

Lecture 10: Multi-Vector Retrieval & ColBERT Implementation

- **Core Topics Covered:**
  - Limitations of single-vector dense embeddings for long context/complex queries.
  - ColBERT Late Interaction: MaxSim operator mechanics.
  - RAGatouille library: Training/Using ColBERTv2 models easily.
  - Hybrid Search: Combining BM25 keyword search with Dense & ColBERT embeddings via Reciprocal Rank Fusion (RRF).

Hands-on Coding / Lab (Colab / Laptop): Implement RAGatouille (ColBERTv2) in Python for document retrieval. Build a hybrid search pipeline with RRF reranking.

Curated Learning Resources: *RAGatouille GitHub Documentation* | *Stanford Future Data Lab ColBERT Papers & Blog*

Lecture 11: Ultra-Fast Vector Search: TurboVec & Vector Indexing

- **Core Topics Covered:**
  - High-performance vector search: Demystifying SIMD, AVX-512, and GPU-accelerated vector indexing.
  - TurboVec Architecture: High-throughput, low-latency vector indexing techniques for large-scale embeddings.
  - Comparison of vector backends: TurboVec vs Faiss vs Milvus vs Qdrant in latency and recall.

Hands-on Coding / Lab (Colab / Laptop): Benchmark vector retrieval speed across 100,000 vectors comparing standard Python cosine similarity vs Faiss vs TurboVec-enabled vector index.

Curated Learning Resources: *Faiss Official Documentation & Benchmarks* | *TurboVec High-Performance Indexing Whitepaper*

Lecture 12: Contextual Retrieval, GraphRAG & Evaluation

- **Core Topics Covered:**

- Advanced RAG Techniques: Contextual Compression, Parent Document Retriever, HyDE (Hypothetical Document Embeddings).
- GraphRAG: Knowledge Graphs combined with Vector Search for multi-hop reasoning.
- RAG Evaluation: Ragas framework (Faithfulness, Answer Relevance, Context Recall, Context Precision).

Hands-on Coding / Lab (Colab / Laptop): Build a production RAG system with HyDE and evaluate its performance score using Ragas framework on a synthetic QA dataset.

Curated Learning Resources: *Ragas Evaluation Framework Documentation* | *Microsoft GraphRAG Documentation & GitHub*

Module 4: Fine-Tuning, SLMs & Vision-Language OCR

Lecture 13: Fine-Tuning Fundamentals & Parameter-Efficient Fine-Tuning (PEFT)

- **Core Topics Covered:**
  - Full Fine-Tuning vs PEFT: Why full tuning is computationally expensive.
  - LoRA (Low-Rank Adaptation): Math behind rank matrices (A and B), alpha parameter, and target modules.
  - QLoRA: 4-bit NormalFloat (NF4) quantization, Double Quantization, Paged Optimizers.

Hands-on Coding / Lab (Colab / Laptop): Configure LoRA parameters in PyTorch. Implement a micro-LoRA layer from scratch on a linear matrix.

Curated Learning Resources: *LoRA Paper (Hu et al.)* | *QLoRA Paper (Dettmers et al.)* | *Hugging Face PEFT Library Docs*

Lecture 14: Hands-on Fine-Tuning Open SLMs (Qwen / Llama / Phi)

- **Core Topics Covered:**
  - Small Language Models (SLMs): Why 1B-3B models rule edge computing.
  - Data Preparation: Instruction formatting (ChatML, Alpaca format, System prompts).
  - SFT (Supervised Fine-Tuning) using Hugging Face TRL (`SFTTrainer`) and Unsloth for 2x-5x speedup.

Hands-on Coding / Lab (Colab / Laptop): Use Unsloth / Hugging Face SFTTrainer on Google Colab GPU to fine-tune Qwen 2.5 / Llama 3 3B on a custom instruction dataset.

Curated Learning Resources: *Unsloth AI Documentation & Notebooks* | *Hugging Face TRL (Transformer Reinforcement Learning) Library*

Lecture 15: Vision-Language Models (VLMs) & Multimodal OCR

- **Core Topics Covered:**
  - Multimodal Architectures: Vision Encoder (ViT / CLIP) + Projection Layer + Language Model.
  - Multimodal Projector: Linear layers vs Cross-Attention vs MLP connectors.
  - Document Intelligence: Using VLMs (Qwen-VL, GLM-Vision) for structured data extraction from financial/identity images (Invoices, Receipts, Cards).

Hands-on Coding / Lab (Colab / Laptop): Load Qwen-VL / Phi-3-Vision in Colab. Pass invoice/cheque images and extract structured JSON data matching a Pydantic schema.

Curated Learning Resources: *Vision Transformers (ViT) Paper* | *Qwen-VL Architecture Overview* | *Hugging Face Transformers VLM Docs*

Lecture 16: Fine-Tuning VLMs for Structured JSON Extraction

- **Core Topics Covered:**
  - Preparing Multimodal Datasets: Image-Text pairs with structured target JSON schemas.
  - Enforcing valid JSON output: Outlines, Instructor, and JSON-Mode generation parameters.
  - Handling noisy scanned OCR images and low-resolution inputs.

Hands-on Coding / Lab (Colab / Laptop): Fine-tune a small Vision-Language model (e.g., Qwen-2.5-VL-3B or Florence-2) to extract custom fields from unstructured scanned documents into clean JSON.

Curated Learning Resources: *Instructor Python Library Documentation | Outlines Structured Generation Library*

Module 5: Inference Optimization: MTP, dFlash & Quantization

Lecture 17: Model Quantization Techniques (GGUF, AWQ, GPTQ, EXL2)

- **Core Topics Covered:**
  - Quantization Fundamentals: Post-Training Quantization (PTQ) vs Quantization-Aware Training (QAT).
  - Symmetric vs Asymmetric Quantization, INT8, INT4, FP4.
  - Formats: GGUF (llama.cpp format), AWQ (Activation-aware Weight Quantization), GPTQ, EXL2.

Hands-on Coding / Lab (Colab / Laptop): Quantize an FP16 open-weight model to GGUF (Q4_K_M) using llama.cpp scripts. Run benchmark tests comparing latency and memory footprints.

Curated Learning Resources: *llama.cpp GitHub Repository | AWQ: Activation-aware Weight Quantization Paper | Hugging Face Quantization Guide*

Lecture 18: Speculative Decoding & Multi-Token Prediction (MTP)

- **Core Topics Covered:**
  - Inference Bottlenecks: Memory-bandwidth bound vs Compute-bound generation.
  - Speculative Decoding: Using a small Draft Model to verify tokens with a Target Model.
  - Multi-Token Prediction (MTP): DeepSeek V3 / modern LLM innovation of predicting N future tokens simultaneously instead of single next token.
  - MTP architecture, training objectives, and speculative acceptance rates.

Hands-on Coding / Lab (Colab / Laptop): Implement a toy Speculative Decoding loop in PyTorch with a 135M draft model and 1B target model. Measure latency speedup factor.

Curated Learning Resources: *DeepSeek V3 Technical Report (MTP Section) | Fast Inference via Speculative Decoding (Leviathan et al.)*

Lecture 19: dFlash Attention & Modern Attention Acceleration

- **Core Topics Covered:**
  - Attention Complexity Revisited: O(N^2) memory bottleneck in long contexts.
  - FlashAttention-1, 2, 3: Tiling, IO-awareness, SRAM vs HBM memory management.
  - dFlash Attention (Dynamic / Block-sparse Flash Attention): Skipping non-essential attention blocks for ultra-fast long-context processing.

Hands-on Coding / Lab (Colab / Laptop): Profile standard PyTorch attention vs `torch.nn.functional.scaled_dot_product_attention` (FlashAttention) vs dynamic sparse attention. Measure VRAM and speed across context lengths (1k to 32k).

Curated Learning Resources: *FlashAttention Paper (Tri Dao) | dFlash & Sparse Attention Optimization Papers*

Lecture 20: Production Inference Servers (vLLM, Ollama, TGI)

- **Core Topics Covered:**
  - PagedAttention & vLLM: Virtual memory allocation for KV-Cache.
  - Continuous Batching vs Static Batching.
  - Deploying local REST API endpoints with vLLM, Ollama, and TensorRT-LLM.

Hands-on Coding / Lab (Colab / Laptop): Spin up a local vLLM / Ollama server, configure API endpoints, and run load testing using Python `asyncio` and `httpx` for parallel requests.

Curated Learning Resources: *vLLM Official Documentation | Ollama GitHub & API Reference*

Module 6: Autonomous AI Agents & Tool Calling

Lecture 21: Agentic Architectures & ReAct Framework

- **Core Topics Covered:**
  - What makes an Agent? Model + System Prompt + Tools + Memory + Loop.
  - ReAct Pattern: Reason -> Act -> Observe -> Repeat loop.
  - Planning & Reflection techniques (Tree of Thoughts, Self-Refine).

Hands-on Coding / Lab (Colab / Laptop): Build a complete ReAct Agent loop from scratch in pure Python without using any external framework.

Curated Learning Resources: *ReAct: Synergizing Reasoning and Acting in Language Models (Yao et al.)* | *LangChain Agent Conceptual Documentation*

Lecture 22: Function Calling & JSON Tool Schema Integration

- **Core Topics Covered:**
  - Native Function Calling: How LLMs output structured tool invocation calls.
  - OpenAI / Anthropic / Ollama Tool Calling Spec: JSON Schemas, parameter validation.
  - Handling tool call execution results and feeding responses back to the model context.

Hands-on Coding / Lab (Colab / Laptop): Define custom Python functions (e.g., Live Weather API, Database Query, Calculator). Bind them via JSON schemas to an open LLM and execute automated tool loops.

Curated Learning Resources: *OpenAI Function Calling Guide* | *Pydantic Documentation for Schema Generation*

Lecture 23: Multi-Agent Frameworks & Collaborative Workflows

- **Core Topics Covered:**
  - Multi-Agent Patterns: Hierarchical, Sequential, Network, and Supervisor setups.
  - Frameworks overview: LangGraph, CrewAI, AutoGen.
  - State Management, Persistence, and Human-in-the-Loop approval workflows.

Hands-on Coding / Lab (Colab / Laptop): Build a 3-agent research system in LangGraph / CrewAI (Agent 1: Web Researcher, Agent 2: Technical Summarizer, Agent 3: Code Reviewer).

Curated Learning Resources: *LangGraph Official Tutorials* | *CrewAI Documentation*

Lecture 24: Agent Memory, Vector Persistence & Sandboxing

- **Core Topics Covered:**
  - Short-term memory (Context Window) vs Long-term memory (Vector Store / Key-Value).
  - Agent state persistence across sessions.
  - Execution Safety: Sandboxing code execution environment (Docker / E2B code interpreter).

Hands-on Coding / Lab (Colab / Laptop): Integrate E2B / Docker code execution sandbox into your agent so it can safely execute arbitrary Python code generated during problem solving.

Curated Learning Resources: *E2B Code Interpreter SDK Docs* | *MemGPT / Letta Autonomous Agent Memory Papers*

Module 7: Developer Workflows & CLI Coding Harnesses

Lecture 25: Introduction to AI Coding Harnesses & Agentic CLI Tools

- **Core Topics Covered:**
  - The Shift in Software Engineering: From writing code to orchestrating AI coding agents.
  - Overview of CLI Harnesses: Claude Code, OpenCode, Gemini CLI, Cursor, Aider.
  - How coding harnesses index codebases: ASTs, Tree-sitter, Git diffs, repo-maps.

Hands-on Coding / Lab (Colab / Laptop): Install and set up Claude Code / OpenCode / Aider CLI environments. Index a sample 5,000-line GitHub repository.

Curated Learning Resources: *Claude Code / Anthropic Developer Documentation | Aider AI Coding in Terminal Docs*

Lecture 26: Prompt Engineering & System Directives for Code Bases

- **Core Topics Covered:**
  - Writing effective `.clauderc`, `AGENTS.md`, and custom system instructions for repository context.
  - Context window management: Token budgets, file pruning, and repo mapping.
  - Architectural guidance to prevent model hallucination in large codebases.

Hands-on Coding / Lab (Colab / Laptop): Create a full system harness definition (`AGENTS.md`) for an existing Flask/FastAPI project and instruct the CLI harness to implement unit test suites autonomously.

Curated Learning Resources: *Anthropic System Prompting Guidelines | Aider Repo Map Architecture Overview*

Lecture 27: Automated Code Generation, Refactoring & Bug Fixing

- **Core Topics Covered:**
  - Automated PR Generation, multi-file refactoring, and test-driven agent loops.
  - Tool integration with Git: Automated branch creation, committing, and conflict resolution via agent harness.
  - Benchmarking Coding Agents: HumanEval, SWE-bench evaluation methodology.

Hands-on Coding / Lab (Colab / Laptop): Use a CLI coding harness (Claude Code / OpenCode / Gemini CLI) to automatically debug a broken open-source repository, pass failing test suites, and draft a pull request.

Curated Learning Resources: *SWE-bench Benchmark Leaderboard & Paper | OpenCode / Gemini CLI Repository Guides*

Lecture 28: Custom Harness Construction & Model Context Protocol (MCP)

- **Core Topics Covered:**
  - Model Context Protocol (MCP): Open standard for connecting AI models to local data sources and tools.
  - Building custom MCP servers in Python/TypeScript.
  - Integrating local IDEs and terminal harnesses with external APIs and databases via MCP.

Hands-on Coding / Lab (Colab / Laptop): Build a custom Model Context Protocol (MCP) server in Python that connects a CLI harness directly to a local PostgreSQL database for database schema analysis.

Curated Learning Resources: *Model Context Protocol (MCP) Official Spec & SDK | Anthropic MCP Developer Guides*

Module 8: Reinforcement Learning (RLHF, DPO & GRPO)

Lecture 29: Post-Training Pipeline: SFT vs Alignment RL

- **Core Topics Covered:**
  - Why Base Models aren't Assistant Models: Pre-training vs Post-training.
  - Alignment Problem: Helpful, Honest, and Harmless (HHH).
  - RLHF Workflow: SFT -> Reward Model Training -> PPO Reinforcement Learning.

Hands-on Coding / Lab (Colab / Laptop): Analyze response preference datasets (Anthropic HH-RLHF). Train a simple Reward Model in PyTorch using binary cross-entropy loss over preferred vs rejected pairs.

Curated Learning Resources: *Illustrating Reinforcement Learning from Human Feedback (Hugging Face Blog) | InstructGPT Paper (Ouyang et al.)*

Lecture 30: Direct Preference Optimization (DPO) & Direct Alignment

- **Core Topics Covered:**

- Mathematical simplification of RLHF: Eliminating the explicit Reward Model.
- DPO Loss Function: Implicit reward derivation from reference model and policy model.
- Variants: KTO (Kahneman-Tversky Optimization), ORPO, IPO.

Hands-on Coding / Lab (Colab / Laptop): Fine-tune a 1B-3B SLM using DPO with Hugging Face TRL `DPOTrainer` to align conversational tone and eliminate refusal hallucinations.

Curated Learning Resources: *DPO Paper (Rafailov et al.)* | *Hugging Face TRL DPO Script Documentation*

Lecture 31: Group Relative Policy Optimization (GRPO) & Reasoning LLMs

- **Core Topics Covered:**
  - The DeepSeek-R1 Revolution: Rule-based reinforcement learning without reliance on traditional reward models.
  - GRPO Algorithm: Group sampling of trajectories, relative advantage normalization.
  - Reward Functions: Enforcing chain-of-thought `<think>` tags, mathematical accuracy, and formatting compliance.

Hands-on Coding / Lab (Colab / Laptop): Implement a toy GRPO reward function in Python to reward model outputs for correctly solving math logic puzzles step-by-step.

Curated Learning Resources: *DeepSeek-R1 Technical Report (GRPO Explanation)* | *Unsloth / Hugging Face GRPO Training Tutorials*

Lecture 32: Practical RL Alignment Workshop

- **Core Topics Covered:**
  - Setting up an end-to-end alignment pipeline on Colab.
  - Evaluating aligned models: Win-rate calculation using LLM-as-a-Judge (GPT-4 / Claude as evaluator).
  - Preventing reward hacking and mode collapse.

Hands-on Coding / Lab (Colab / Laptop): Execute a full DPO/GRPO alignment notebook. Evaluate pre-aligned vs post-aligned model outputs using an automated LLM-as-a-Judge script.

Curated Learning Resources: *AlpacaEval Framework* | *LLM-as-a-Judge Evaluation Methodology (Zheng et al.)*

Module 9: Capstone Project & Industry Deployment

Lecture 33: Production System Architecture & API Design

- **Core Topics Covered:**
  - Designing enterprise-grade GenAI systems: Fast API wrapper, queue systems (Celery/Redis), rate limiting.
  - Streaming responses: Server-Sent Events (SSE) and WebSockets for real-time token streaming.
  - Cost calculation & latency optimization strategies.

Hands-on Coding / Lab (Colab / Laptop): Build a FastAPI backend service that streams LLM tokens via Server-Sent Events (SSE) and exposes endpoints for RAG and tool execution.

Curated Learning Resources: *FastAPI Official Streaming Response Guide* | *AsyncIO Python Documentation*

Lecture 34: Containerization & Cloud Infrastructure (Docker & GPU)

- **Core Topics Covered:**
  - Dockerizing AI applications: CUDA base images, dependency management, weight mounting.
  - Deploying to Cloud GPU providers (RunPod, Modal, AWS EC2 g5, GCP Vertex).
  - Security: Environment variables, API key management, static public IP whitelisting.

Hands-on Coding / Lab (Colab / Laptop): Write a Dockerfile for your FastAPI + vLLM / RAG app. Package container and run it locally, verifying GPU pass-through (`--gpus all`).

Curated Learning Resources: *NVIDIA Container Toolkit Docs* | *Docker for AI Developers Tutorial*

Lecture 35: Observability, Evaluation & Security (LLMOps)

- **Core Topics Covered:**
  - LLMOps stack: Monitoring latency, token usage, drift, and hallucination metrics.
  - Observability tools: LangSmith, Phoenix (Arize), OpenInference.
  - AI Security: Prompt Injection mitigation, PII masking, guardrails (NeMo Guardrails, Llama Guard).

Hands-on Coding / Lab (Colab / Laptop): Integrate LangSmith / Phoenix observability into your application. Add a Llama Guard safety filter layer to block prompt injection attacks.

Curated Learning Resources: *LangSmith Documentation* | *NVIDIA NeMo Guardrails* | *Llama Guard Safety Paper*

Lecture 36: Capstone Project Showcase & Industry Readiness

- **Core Topics Covered:**
  - Final Capstone Presentation: Demonstration of end-to-end industry-ready GenAI product.
  - Resume Building for GenAI Roles: Key project bullet points, GitHub portfolio curation.
  - Industry Technical Interview Prep: Architecture design whiteboarding, coding round expectations.

Hands-on Coding / Lab (Colab / Laptop): Live code demonstration of capstone projects to industry mentors / instructors. Receive feedback and portfolio reviews.

Curated Learning Resources: *GitHub AI Portfolio Best Practices* | *GenAI Engineering Interview Guide*

4. Assessment Architecture & Student Capstone Projects

To ensure students leave the course with a job-ready portfolio, grading is heavily weighted toward practical software engineering outputs rather than traditional examinations.

Assessment Component	Weight	Deliverable Description
Weekly Lab Coding Notebooks	30%	Submission of completed Google Colab notebooks from Lectures 4 through 32.
Mid-Term Milestone: Custom RAG Engine	20%	Build and evaluate an advanced RAG engine using ColBERT, TurboVec, and Ragas metrics.
Fine-Tuned VLM Document Extractor	20%	Fine-tune a 3B Vision SLM to convert unstructured PDF/Image documents into validated JSON.
Final Capstone Project	30%	Full-stack production GenAI product deployed with Docker, FastAPI, tool agents, and CLI coding harness integration.

5. Hardware, Software & Environment Requirements

Recommended Student Setup:
• Student Laptop: Any modern PC/Mac/Linux machine with at least 16GB RAM and VS Code installed.
• Compute Infrastructure: Free Tier Google Colab (T4 GPU) is sufficient for all core coding sessions. Optional Colab Pro or RunPod credit ($10-$15 per student) for faster fine-tuning runs in Modules 4 and 8.
• Python Stack: Python 3.11+, PyTorch 2.x, Transformers, PEFT, TRL, Unsloth, LangChain/LangGraph, llama.cpp, vLLM, TurboVec, RAGatouille, E2B SDK.