

LLMs in Practice: A Four-Step Journey from Idea to Implementation.

Anni Chen, [Amazon.com](https://www.amazon.com)
chenanni02@gmail.com



About me

- **Software Engineer at Amazon**, leading **LLM initiatives** in **Personalization** powering Amazon Stores.
- **Co-founder, SpeakUp AI Inc.**
- **Duke University** alum in **CS & Mathematics**.
- **Mentor, speaker, and judge** in AI communities; **published** at **RepL4NLP (ACL)**.
- Talk draws from **enterprise-scale LLMs** and **AI ethics research**.



About this tutorial

1. **Ideation** – Identifying LLM-suitable problems.
2. **Data & Design** – Curating knowledge and defining how the model interacts with context.
3. **Implementation** – Architecting and deploying the system end-to-end.
4. **Evaluation** – Measuring impact and closing the loop for continuous learning.



The Era of Applied LLMs

- LLMs have moved **from research labs to real-world platforms** across retail, healthcare, finance, and education.
- Success isn't just about **access to top models** — it's about **responsible, scalable integration**.
- Building production-grade LLM systems = building an **orchestra** — requires:
 - **Data pipelines**
 - **Feedback loops**
 - **Governance & observability**
- Impact came from **embedding LLMs** into personalization and recommendation systems — connecting them with **real-time data** and **evaluation loops**.
- The magic lies not in the model alone, but in **how it's connected to the ecosystem**.



The Four-Step Journey in Context

1. **Ideation** – Identifying LLM-suitable problems.
2. **Data & Design** – Curating knowledge and defining how the model interacts with context.
3. **Implementation** – Architecting and deploying the system end-to-end.
4. **Evaluation** – Measuring impact and closing the loop for continuous learning.



Ideation:

**From Problem to LLM
Opportunity**

Understanding the “How”



Why Start with the How

- “LLMs are just big next-token predictors.” — *Andrej Karpathy*
- Understanding this reveals both **power and limits** of LLMs.
- Helps teams **pick the right problems** to solve.

Pretraining – Learning the Language

- Model reads vast Internet text → learns to **predict the next token**.
- Every step = “fill in the blank” → statistical mirror of human language.
- Learns **grammar, reasoning, and knowledge** implicitly.
- Best for any task that means “**continue this text sensibly**”:
 - Summarization • Translation • Ideation • Brainstorming • Q&A

Fine-Tuning & Alignment

② Supervised Fine-Tuning (SFT)

- Adds **obedience and structure** to pretrained fluency.
- Human-crafted **instruction** → **ideal response** pairs.
- Works best when “**good output**” is **clearly defined**.
- Example: *“Explain gradient descent like I’m five.”*

🎯 Reinforcement Learning from Human Feedback (RLHF)

- Aligns model with **human preferences**.
- Humans compare answers → reward model learns preference.
- Main model optimized for **helpfulness, honesty, and safety**.
- Excels in **creative, empathetic, and subjective** domains.

Fine-Tuning & Alignment



Stage	Model Learns	Ideal Problem Types
Pretraining	World knowledge & language fluency	Open-domain writing, summarization, concept generation
Supervised Fine-Tuning (SFT)	Task obedience	Clearly defined instruction tasks — classification, QA, reasoning
Reinforcement Learning from Human Feedback (RLHF)	Human alignment	Subjective or conversational tasks — assistants, copilots, personalized content

Prompt Engineering



Zero-Shot Prototyping

“Summarize this policy in one sentence.”

Few-Shot Prompts

“Input: [review] → Output: [sentiment].”

Chain-of-Thought Prompts

“Let’s think step by step.”

Role Prompting

“You are a cybersecurity analyst...”

Three lens



- **Knowledge Lens (Pretraining)** – Does the model already “know” enough about this domain?
→ If not, plan retrieval or fine-tuning.
- **Instruction Lens (SFT)** – Can you describe the task clearly in natural language?
→ If yes, prompt engineering may suffice.
- **Preference Lens (RLHF)** – Is quality subjective (style, empathy, persuasion)?
→ LLMs excel here — trained via preference gradients.

Mini Prompt Lab: Worked Example



Goal: Rewrite product titles for different contexts.

- **Zero-shot:** “Rewrite this title to be concise and mobile-friendly.”
- **Iterated:** “You are a copywriter for Amazon. Rewrite each title in under 60 characters.”
Observations:
 - Understands e-commerce phrasing → *Pretraining*
 - Follows instruction → *SFT*
 - Adapts tone via role prompt → *RLHF*
 - ✓ Lightweight, infrastructure-free viability test.

Why Prompt Engineering Works



- Each prompt = **micro fine-tuning in real time**.
- Your text shifts the model's probability space within its **learned manifold**.
- Role prompts (e.g., "You are a lawyer...") move generation toward that subdomain.
- **RLHF amplifies** politeness, structure, safety.
 - ✨ Clear roles & instructions work because they mirror training signals.

Prompting vs. Retraining

Myth: Prompting can fix any domain gap.

Reality: “If the model never saw data like yours, prompting won’t invent it.” — *Karpathy*

Use prompting for **feasibility testing**.

If consistent gaps appear → move to **fine-tuning** or **RAG**.

 *Prompting = stethoscope, not surgery.*

Prompting vs. Retraining

Myth: Prompting can fix any domain gap.

Reality: “If the model never saw data like yours, prompting won’t invent it.” — *Karpathy*

Use prompting for **feasibility testing**.

If consistent gaps appear → move to **fine-tuning** or **RAG**.

 *Prompting = stethoscope, not surgery.*

Mapping Problems to LLM Strengths



Problem Type	Why LLMs Work	Example
Knowledge synthesis	Pretraining stores broad facts	Generate FAQs, explain code
Instruction execution	SFT teaches task following	Convert reviews → sentiment
Preference alignment	RLHF rewards human-liked style	Draft empathetic emails/headlines

Prompt Engineering Tips

- 🎯 **Be explicit** about role and goal.
- 🧱 **Specify output format:** “Answer in JSON with ‘summary’ and ‘tone’ fields.”
- 🛠️ **Add constraints**, not examples, when style matters more than content.
- 🎛️ **Experiment with temperature** 0.2–0.8 for creativity vs. stability.
- 🧩 **Test multiple completions** for behavioral consistency.
 - 💡 Prompts are disposable — use them to **discover**, not finalize.

From Prompt to Problem Definition



After 10–15 prompt tests:

- Identify what inputs led to success.
- Note recurring failures (knowledge gaps, verbosity).
- Draft metrics (fluency, relevance, tone).
 - ➔ Document insights as an **LLM Opportunity Brief** — the bridge to **Data & Design**.

Ethical & Operational Sanity Checks



Data Sensitivity — Avoid sending private info to external APIs.

Bias Sensitivity — Watch for stereotyping in zero-shot outputs.

Cost Realism — Prompt length → token cost; prototype within budget.



Data & Design

From Prompt to Pipeline



1. What does the model already “know”?
2. What should we remind or teach it dynamically?
3. When is it worth *re-teaching its weights* through Supervised Fine-Tuning (SFT)?

The Three Layers of Memory



Parametric Memory – learned in pretraining; changed via SFT.

Context Memory – what's in the prompt window; changed via prompt design.

Retrieval Memory – external indexed docs; changed via RAG/vector DB.

Why Pretraining Is Not Enough



Base models know the Internet - but not your catalog, tone, or updates.

Retrieval pipelines refresh knowledge without retraining.

If gaps persist, consider SFT after understanding cost and purpose.

Building the Retrieval Layer



Most systems reach 80–90% performance with RAG + prompt templating.

Steps: trusted docs → chunking → embedding → retrieval → prompt assembly.

Keeps knowledge fresh without retraining.

Chunking & Embedding Deep Dive



Balance chunk size between context and coherence.

Add metadata for filtering and freshness.

Embeddings form semantic geometry - the model's real-time knowledge graph.

Designing Prompts That Scale



Prompt templates combine retrieved content and instructions.

Use structured templates with context, task, and output schema.

Version them—small wording changes shift accuracy 10–20%.

When RAG Reaches Its Limit



Retrieval can't enforce style, consistency, or logic.

Examples: legal phrasing, empathy tone, multi-turn reasoning.

If prompts become too long—encode rules via SFT.

When RAG Reaches Its Limit



Retrieval can't enforce style, consistency, or logic.

Examples: legal phrasing, empathy tone, multi-turn reasoning.

If prompts become too long—encode rules via SFT.

What SFT Actually Changes



SFT doesn't add facts—it changes expression and tone.
Improves efficiency and consistency across outputs.

The Cost and Risk Equation



Data curation: high labor cost

Compute: \$1K–\$10K per run

Maintenance: ongoing retraining

Principle: Prototype → Validate → Commit after stability.

Decision Framework: Do We Need SFT?



Knowledge Gap → Retrieval / RAG

Instruction Gap → Light SFT

Preference Gap → SFT or RLHF

Retrieve if data missing; fine-tune if style misaligned.

LoRA & Lightweight Alternatives



Fine-tune small parameter blocks (1–5% cost).

Swap per domain—marketing, legal, support.

Adapters act as plug-ins for behavior.

Practical Judgment Checklist



Stable prompt + RAG outputs

High-quality labeled data

Defined metrics & ROI >10–15%

Retraining/version plan ready.

Iterative Workflow



Prototype → Log → Curate → Fine-tune.

Iterate gradually to maintain agility and prevent overfitting.

Architectural Perspective



SFT only affects the base model block.

Retrieval and templating remain modular and reusable.

Observability & Cost Metrics



Monitor: token cost, retrieval precision, latency, adapter hit rate.

Measure both RAG and SFT impact before scaling spend.

Ethical & Operational Design



Ensure data provenance and audit trails.

Re-train periodically (3–6 months).

Treat fine-tuning as versioned software.

The Perspective Shift



Training is no longer central—data & design are.

Context is the differentiator; SFT is a late-stage optimization.

Key Takeaways



Context is the new training.

SFT adds polish, not purpose.

Decide with evidence, not enthusiasm.

Keep architecture modular and future-proof.



Implementations & Deployment

Implementation Overview



- Implementation is not one-size-fits-all.
- Varies by latency tolerance, data sensitivity, and interaction mode.
- Three archetypes: Offline Batch, Interactive RAG, Real-Time Systems.

Use Case A – Offline Batch Generation



- High-latency-tolerant workloads: marketing, SEO, documentation.
- Pipeline: dataset prep → batch generation → S3 storage.
- Focus: throughput and cost efficiency.
- Optimization: GPU batching, caching, async orchestration.

Use Case B – Interactive RAG Assistants



- Mid-latency (~2s) systems: support bots, internal assistants.
- Architecture: nightly index + online retrieval.
- Combine offline prep and online generation.
- Focus: retrieval speed, streaming responses.

Use Case C – Real-Time Recommendations



- Sub-500ms response time.
- Precompute outputs, cache results, or use small models.
- Architecture: SageMaker endpoints, Redis cache.
- Key: pre-thinking over real-time creativity.

System Archetypes Comparison



Dimension	Offline Batch	Interactive RAG	Real-Time
Latency goal	Minutes–hours	1–3 s	< 500 ms
Compute	GPU clusters	On-demand API	Cached micro-models
Freshness	Low	Daily refresh	Continuous
Best optimization	Token efficiency	Retrieval speed	Pre-computation
Typical deployment	Step Functions + S3	API Gateway + Bedrock	Lambda + Redis cache

Core Building Blocks



1. Orchestration – Step Functions, Airflow.
2. Model Serving – SageMaker, Bedrock.
3. Retrieval – FAISS, Pinecone.
4. Validation – Guardrails, filters.
5. Observability – Metrics and feedback.

Offline Preparation



- Deduplication and normalization.
- Embedding pre-computation.
- Prompt pre-assembly.
- Offline efficiency → online savings.

Latency & Data Design



- Smaller context → faster response.
- Retrieval quality > quantity.
- Measure 'tokens per correct answer'.

Prompt Templates



- System, context, instruction, format.
- Enables parameterization and A/B testing.
- Prompts = SQL queries for language.

System role: you are a [persona].

Context: {retrieved_text}

Instruction: {task}

Format: {schema}

Caching & Memory Patterns



- Prompt-response cache (hash match).
- Semantic cache (vector similarity).
- Cuts token bills by up to 40%.

Guardrails & Validation



- Schema, grounding, safety, tone checks.
- Treat guardrails as unit tests.

Observability & Feedback



- Monitor: tokens/request, latency p95, cache hits, grounding.
- Collect user feedback for retraining.
- Deployment → new training loop.

Latency Optimization Toolkit



- Parallelize steps.
- Use streaming and caching.
- Reduce context window.
- Prefer smaller models when possible.

Security & Privacy



- Input sanitization.
- Tenant data segregation.
- VPC endpoints, encryption, audit trails.
- Trust = scalability.

Deployment & Rollout



Testing Before Go-Live

- Golden sets, regression, auto-judge, stress tests.
- Measure risk before release.

- Canary rollouts: v1–v3 evolution.
- Compare metrics before promotion.
- Apply CD principles to model behavior.

Cost Governance



8B + good prompts > 70B + poor design.

	Metric	Meaning
	Tokens / output	Efficiency indicator
	Cache hit rate	Reuse percentage
	Cost / approved answer	ROI proxy
	Model mix ratio	% small vs large models

Human-in-the-Loop



- Human review for sensitive domains.
- System learns from edits and ratings.
- Accountability + improvement.

Takeaways



- Match latency to architecture.
- Optimize offline.
- Deploy with PromptOps discipline.
- Fine-tune strategically.
- Monitor semantics and economics.
- Close loop with human feedback.



Evaluation & Continuous Learning

Three Moments of Evaluation



- Offline – before deployment: test safety, relevance.
- Online – after launch: measure user or business impact.
- Ongoing – continuous: detect drift, cost, hallucination.
- Truth types: synthetic, user, operational.

Offline Evaluation



- Safety net before users.
- Golden sets – human-labeled examples with rubrics.
- LLM-as-judge – scalable auto-rating (~ 0.8 correlation with humans).
- Automatic metrics – BLEU, ROUGE, similarity, schema checks.

Hallucination Problem



- Fluency without factuality.
- Checks: Context grounding, citation, contradiction.
- Metric: Grounding Score = supported tokens / total tokens.
- Hallucination = data alignment issue, not moral failure.

Online Evaluation



- Test real-world impact.
- A/B tests, bandit optimization, surveys, implicit signals.
- Measure behavior change, not only text quality.

Drift Detection



- Model degradation sources: data shifts, retriever drift, base model changes.
- Detect via grounding score drops, latency spikes, embedding drift.
- Trigger nightly alerts, re-embedding, re-tests.

Automated Learning Cycle



- 1. Collect feedback.
- 2. Filter strong examples.
- 3. Add to SFT set.
- 4. Retrain monthly.
- 5. Canary redeploy.
- Self-improving LLM pipeline.

Evaluation Dashboards



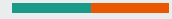
Unified dashboard:

- Quality & Grounding.
- Cost & Latency.
- Feedback & Cache hit.
- Alerts on metric deviation.

Key takeaways



1. Define good early.
2. Mix human + automation.
3. Monitor drift.
4. Close feedback loops.
5. Build evaluation culture across teams.



Conclusion



The Four-Step Journey in Context

1. **Ideation** – Identifying LLM-suitable problems.
2. **Data & Design** – Curating knowledge and defining how the model interacts with context.
3. **Implementation** – Architecting and deploying the system end-to-end.
4. **Evaluation** – Measuring impact and closing the loop for continuous learning.

Resources



Topic	Recommended Resource
Foundation Models & Theory	<i>On the Opportunities and Risks of Foundation Models</i> (Bommasani et al., Stanford 2022)
Prompt Engineering	<i>OpenAI Cookbook, Prompt Engineering Guide</i> (Burns et al.)
Retrieval & Vector Databases	Pinecone docs; <i>RAG From Scratch</i> (Andrej Karpathy video); <i>Haystack 2.0</i> framework
Fine-Tuning & LoRA	Hugging Face PEFT tutorials; <i>Efficient Fine-Tuning of Large Language Models</i> (Li et al., 2023)
Evaluation & Metrics	<i>Beyond Accuracy: Evaluating Text Generation Models</i> (Callison-Burch & Webber, 2021); Anthropic's <i>Constitutional AI</i> papers
Ethics & Responsible AI	Partnership on AI's <i>Responsible Practices for Synthetic Media</i> ; NIST AI Risk Management Framework
Infrastructure & MLOps	AWS Bedrock and SageMaker workshops; <i>LLMOps: Managing LLM Lifecycles</i> (O'Reilly 2024)
Multimodal & Agents	<i>Hugging GPT</i> (Pan et al., 2023); <i>LangGraph</i> and <i>OpenDevin</i> agent frameworks



Thank you!