Large Language Models for Research

Tools, Techniques, and Best Practices

Sam Davenport

UC San Diego

April 2026

Roadmap

Part 1: Foundations

- LLM fundamentals
- Prompt engineering

Part 2: Advanced Topics

- DSPy for prompt optimization
- LLMs for mathematical reasoning

Part 3: Practical Tools

- Claude Code: AI in the terminal
- Multi-agent workflows
- Real-world applications

Section 1

LLM Fundamentals

What are Large Language Models?

Definition

Neural networks trained on massive text corpora to predict and generate human-like text

Key characteristics:

- **Scale**: Billions of parameters (GPT-4: $\sim$1.7T, Claude 3: undisclosed)
- **Training**: Self-supervised on internet-scale text
- **Capabilities**: Emerge from scale + diverse training data

Foundation models — general-purpose, adaptable to many tasks

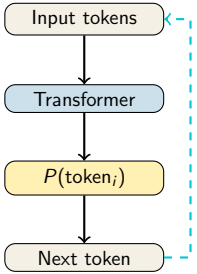
How LLMs Work: The Basics

1. Tokenization

- Text $\rightarrow$ discrete tokens
- "statistical" $\rightarrow$ ["stat", "ist", "ical"]
- Vocabulary: 30K–100K tokens

2. Next-Token Prediction

- Core objective: $P(\text{next token} \mid \text{context})$
- Transformer architecture
- Autoregressive generation



Next-Token Prediction: Examples

Given context, predict the most likely next token:

Context	Likely next tokens
"The capital of France is"	**Paris** (very high prob)
"In Bayesian inference, the posterior is"	proportional, computed, ...
"import numpy as"	**np** (very high prob)
"The p-value was 0.03, so we"	reject, concluded, ...
"Let $X \sim \mathcal{N}(0,$ "	1), σ^2), ...

Key insight: LLMs learn *patterns* from training data

- Common phrases $\rightarrow$ high confidence predictions
- Domain knowledge encoded in token probabilities
- Generation = repeated next-token sampling

Context Windows

Context Window

Maximum number of tokens the model can process at once (input + output)

Model	Context	Approx. Pages
GPT-3.5	16K	∼25 pages
GPT-4o	128K	∼200 pages
Claude 3.5 Sonnet	200K	∼300 pages
Gemini 1.5 Pro	2M	∼3000 pages

Implications for research:

- Analyze entire papers or datasets in context

Key Parameters: Temperature & Top-p

Temperature (τ)

- Controls randomness
- $\tau \to 0$: deterministic
- $\tau \to 1$: more creative

$$P(x_i) = \frac{\exp(z_i/\tau)}{\sum_j \exp(z_j/\tau)}$$

Research use: Low temp (0.0–0.3) for factual tasks

Top-p (Nucleus Sampling)

- Sample from smallest set where $\sum P \geq p$
- $p = 0.9$: top 90% probability mass
- Complements temperature

Typical settings:

- Coding: $\tau = 0$, $p = 1$
- Writing: $\tau = 0.7$, $p = 0.9$

Temperature in Action

Prompt: "Write a one-sentence description of Bayesian inference."

Temperature = 0.0 (deterministic):

> *"Bayesian inference is a statistical method that updates the probability of a hypothesis as more evidence becomes available."*

Temperature = 0.7 (balanced):

> *"Bayesian inference provides a principled framework for updating our beliefs about unknown quantities by combining prior knowledge with observed data through Bayes' theorem."*

Temperature = 1.2 (creative):

> *"Like a detective weighing clues against hunches, Bayesian inference elegantly fuses what we suspected beforehand with fresh evidence to refine our grasp of uncertain truths."*

Higher temperature → more varied, creative (but potentially less precise)

System Prompt

Special instructions that set the model's behavior, role, and constraints

```
messages = [
    {"role": "system", "content": """You are a statistical consultant.
    - Be precise and cite sources
    - Use mathematical notation when appropriate
    - If uncertain, say so explicitly"""},
    {"role": "user", "content": "Explain MCMC convergence diagnostics"}
]
```

Use cases:

- Define expertise domain and communication style
- Set output format requirements
- Establish safety guardrails

Section 2

Prompt Engineering

Prompt Engineering: Why It Matters

Prompt Engineering

The practice of designing inputs to elicit desired outputs from LLMs

Same model, different prompts:

- "Summarize this paper" $\rightarrow$ Generic summary
- "List the 3 main contributions and their limitations" $\rightarrow$ Structured analysis

Quality of output $\propto$ Quality of prompt

The gap between a naive prompt and an optimized prompt can be larger than the gap between model generations

Basic Prompt Structure

1. **Context**: Background information, role definition
2. **Task**: Clear, specific instruction
3. **Format**: Desired output structure
4. **Examples**: (Optional) Demonstrations
5. **Constraints**: Limitations, edge cases

Example

You are a statistics expert reviewing a methods section. [Context]
Identify any statistical errors or questionable practices. [Task]
Format as a numbered list with severity ratings (minor/major/critical). [Format]
Focus only on statistical methodology, not writing style. [Constraint]

Few-Shot Prompting

Provide examples to guide the model's behavior:

```
prompt = """Convert statistical notation to LaTeX.

Input: X ~ N(mu, sigma^2)
Output: $X \sim \mathcal{N}(\mu, \sigma^2)$

Input: E[X] = integral from -inf to inf of x f(x) dx
Output: $\mathbb{E}[X] = \int_{-\infty}^{\infty} x f(x) \, dx$

Input: P(A|B) = P(B|A)P(A) / P(B)
Output:"""
```

Guidelines:

- 2–5 examples usually sufficient
- Cover edge cases in examples
- Ensure examples are correct (models learn from errors too!)

Chain-of-Thought (CoT)

Prompting the model to show its reasoning step-by-step before giving a final answer

Simple trigger: "Let's think step by step..."

Why it works:

- Breaks complex problems into manageable steps
- Reduces errors in multi-step reasoning
- Makes reasoning auditable

Best for:

- Mathematical derivations
- Multi-step analysis
- Debugging code

Chain-of-Thought: Example

Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ✗

Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

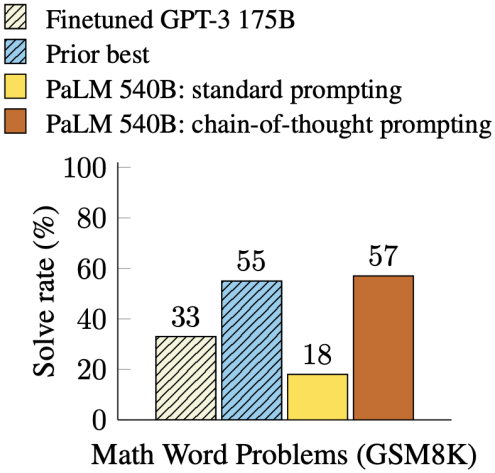
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

Chain-of-Thought: Example



Iteration Strategies

Prompting is an iterative process:

1. **Start simple** — Get a baseline response
2. **Analyze failures** — Where does it go wrong?
3. **Add specificity** — Address the failure modes
4. **Test variations** — Try different phrasings
5. **Document what works** — Build a prompt library

Useful techniques:

- A/B test prompts on held-out examples
- Version control your prompts
- Track metrics: accuracy, consistency, cost

Section 3

Claude Code

What is Claude Code?

Definition

An AI assistant that lives in your terminal, with full access to your filesystem, shell, and development tools

Key capabilities:

- **Read & write files** — Navigate codebases, edit source files
- **Execute commands** — Run tests, compile code, manage git
- **Iterative development** — Fix errors, refactor, and verify

Why this matters:

- Browser-based LLMs are isolated from your actual work environment
- Terminal integration enables *agentic* workflows
- The AI can verify its own outputs by running code

CLAUDE.md: Project-Specific Instructions

CLAUDE.md

A markdown file at the project root that provides context and instructions to Claude Code

```
# My Research Project

## Codebase Overview
- `src/` - Main source code (Python 3.10+)
- `tests/` - Pytest test suite
- `data/` - Raw and processed datasets (do not commit)

## Conventions
- Use numpy docstrings for all functions
- Run `pytest` before committing
- Type hints required for public APIs

## Current Focus
Working on implementing the bootstrap confidence intervals
for the estimator in src/estimators.py
```

Tip: Update CLAUDE.md as your project evolves

Plan Mode: Think Before Acting

Plan Mode

A workflow mode where Claude designs an implementation strategy for user approval before making any changes

How it works:

1. Invoke with `/plan` or let Claude suggest it for complex tasks
2. Claude explores the codebase and designs an approach
3. Review the plan — ask questions, request changes
4. Approve to begin implementation

When to use:

- Multi-file refactoring or architectural changes
- Unfamiliar codebases where you want to understand the approach
- Complex features with multiple valid implementations

Key benefit: Catch design issues before any code is written

Example: Rapid Prototyping

Creating applications with no prior experience:

Traditional approach:

- Learn the framework
- Set up build tools
- Write boilerplate
- Debug configuration
- *Finally* write logic

A hex game I coded at 18 took 2 months

With Claude Code:

- Describe what you want
- Review generated code
- Iterate on feedback

Same hex game: **10 minutes**

Examples: `https://sjdavenport.github.io/games/hex/`

Example: Paper → Presentation

Converting a paper to slides with minimal effort:

Iterative Prompting

1. *"Create a presentation for this paper. I've included an example presentation in the example_slides folder."*
2. *"Include the key parts of the proofs and explain them."*
3. *"Some content overflows the page — fix the formatting."*

Key insight: Providing examples helps the LLM learn your style

- Include an existing presentation as a template
- The paper provides content; the example provides format

Example: Academic Website

Prompt:

> *I am a statistics researcher at UC San Diego. Here is my Google Scholar page: [URL]. Create an academic website with a homepage introducing me and my research, plus a publications page.*

What Claude Code generated:

- Responsive HTML/CSS layout
- Parsed publication list from Scholar
- Professional academic styling
- Ready for GitHub Pages deployment

Time: A few minutes (vs. hours of manual HTML/CSS)

Key Idea

Run multiple Claude instances simultaneously on different tasks

Setup:

1. Use a modern terminal (e.g., Warp, iTerm2 with tabs/panes)
2. Launch Claude Code in multiple directories
3. Each instance has its own context via CLAUDE.md

Example workflow:

- **Agent 1:** Writing simulation code in `src/`
- **Agent 2:** Drafting documentation in `docs/`
- **Agent 3:** Creating visualizations in `figures/`

Agents in subfolders inherit parent CLAUDE.md context

Best Practices for Claude Code

1. **Start with clear context**
   - Write a good CLAUDE.md
   - Describe your goal before asking for code
2. **Iterate incrementally**
   - Ask for small changes, review, then continue
   - Easier to catch errors early
3. **Verify outputs**
   - Run tests after each change
   - Check generated code before committing
4. **Use version control**
   - Commit before major AI-assisted changes
   - Easy to revert if needed

Section 4

LLMs for Mathematics

The Challenge

Mathematical reasoning is a hallmark of intelligence — can LLMs truly reason mathematically?

Hierarchy of math benchmarks:

1. **GSM8K** — Grade-school word problems (Solved)
2. **MATH** — High-school competition math (Solved)
3. **AIME** — American Invitational Math Exam (Near saturation)
4. **USAMO/IMO** — Olympiad-level proofs (Frontier)

Key shift: IMO requires *rigorous proofs*, not just final answers

The International Mathematical Olympiad

- World championship of high-school mathematics (since 1959)
- 100+ countries, 6 problems over 2 days (4.5 hours each)
- Problems selected for **novelty** — no pattern matching
- Gold medal: top $\sim$8% of contestants

Why IMO matters for AI:

- Demands creativity, multi-step reasoning, and rigor
- Novel problems prevent memorization
- Proofs must be logically sound — no "close enough"

11 of 34 Fields Medalists (since 1990) are IMO gold medalists

LLM Performance on IMO 2025

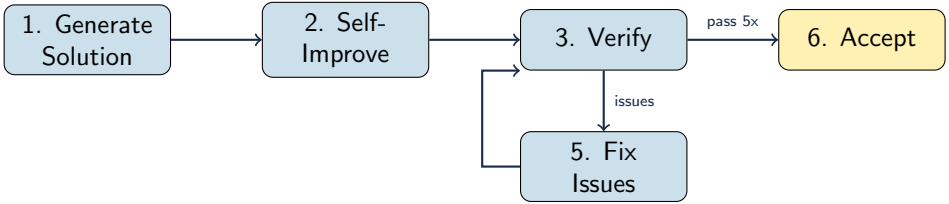
Model	Baseline (Best of 32)	With Pipeline
Gemini 2.5 Pro	31.6%	**85.7%**
Grok-4	21.4%	**85.7%**
GPT-5	38.1%	**85.7%**

Key insight: Same models, dramatically different results

The verification-and-refinement pipeline (Huang & Yang, 2025) achieved **5/6 problems** — gold medal level

Source: "Winning Gold at IMO 2025 with a Model-Agnostic Verification-and-Refinement Pipeline"

The Verification-and-Refinement Pipeline



Core idea: Iterate until solution passes verification 5 consecutive times

Why it works:

- Breaks reasoning into manageable steps (finite token budget)
- Verifier catches errors that solver misses
- Iterates until success — analogous to peer review

Solver Prompt: Emphasizing Rigor

```
### Core Instructions ###

* **Rigor is Paramount:** Your primary goal is to produce a
  complete and rigorously justified solution. Every step must
  be logically sound and clearly explained. A correct final
  answer derived from flawed reasoning is considered a failure.

* **Honesty About Completeness:** If you cannot find a complete
  solution, you must **not** guess or create a solution that
  appears correct but contains hidden flaws. Instead, present
  only significant partial results that you can rigorously prove.

### Output Format ###
**1. Summary**
  * **Verdict:** State whether you have a complete or partial solution
  * **Method Sketch:** High-level outline of your approach

**2. Detailed Solution**
  Present the full, step-by-step mathematical proof. Each step
  must be logically justified.
```

Verifier Prompt: Acting as IMO Grader

```
You are an expert mathematician and meticulous grader for an
International Mathematical Olympiad (IMO) level exam. Your task
is to rigorously verify the provided solution.

A solution is judged correct **only if every step is rigorously
justified.** A solution that arrives at a correct answer through
flawed reasoning must be flagged as incorrect.

**How to Handle Issues:**
* **Critical Error:** Breaks the logical chain (factual errors,
  logical fallacies). Invalidates the current line of reasoning.

* **Justification Gap:** Conclusion may be correct but argument
  is incomplete or lacks rigor. Assume true and continue checking.

**Output Format:**
* **Final Verdict:** Single sentence on validity
* **List of Findings:** Location + Issue for each problem found
* **Detailed Verification Log:** Step-by-step analysis
```

Why Verification Matters

Single-pass generation is insufficient:

- Models have finite reasoning budgets (e.g., 32K thinking tokens)
- Complex proofs exceed single-pass capacity
- Initial solutions often contain subtle errors

The verifier catches:

- **Critical errors** — logical fallacies, calculation mistakes
- **Justification gaps** — hand-wavy arguments, missing steps

Key Finding

Strong LLMs already possess powerful mathematical reasoning capabilities — but verification pipelines are essential to convert latent ability into rigorous proofs

Historical Progress: DeepMind at IMO

IMO 2024: AlphaProof + AlphaGeometry 2

- Silver medal: 28/42 points (4/6 problems)
- Required translating problems to formal language (Lean)
- 2–3 days of computation per problem

IMO 2025: Gemini Deep Think

- Gold medal: 35/42 points (5/6 problems)
- End-to-end natural language proofs
- Within 4.5-hour competition time limit

Parallel thinking: Explore multiple solution paths simultaneously, then combine insights — not just linear chain-of-thought

Implications for Statisticians

What this means for research:

1. **Proof assistance** — LLMs can help verify derivations
2. **Exploration** — Generate candidate approaches to try
3. **Verification is key** — Never trust without checking

Current limitations:

- Still struggles with the hardest problems (Problem 6)
- Combinatorial reasoning remains challenging
- Novel techniques may be beyond training distribution

Best practice: Use LLMs for exploration and drafting, but verify rigorously yourself

Section 5

DSPy

The Problem: Prompt Sensitivity

Same task, different prompts, wildly different results:

Prompt for sentiment classification	Accuracy
"Is this review positive or negative?"	72%
"Classify the sentiment of this text."	78%
"You are a sentiment analysis expert. Carefully analyze..."	81%
"Given the review, output POSITIVE or NEGATIVE."	85%
+ 3 well-chosen examples	**91%**

The gap is huge

Prompt choice can matter more than model choice! But finding the right prompt is tedious, task-specific, and doesn't transfer.

What is DSPy?

DSPy

A framework for *programmatic* prompt optimization — treat prompts as learnable parameters

The philosophy:

- Manual prompt engineering is brittle and doesn't scale
- Prompts should be *optimized*, not hand-crafted
- Define *what* you want, let DSPy figure out *how*

Key analogy:

Manual prompting : DSPy :: Hand-tuned features : Deep learning

You provide: Task definition, training examples, evaluation metric
DSPy returns: Optimized prompt (instructions + examples)

Method 1: Bootstrap Few-Shot

Core Idea

Automatically discover which examples work best as few-shot demonstrations

How it works:

1. Run the LLM on training examples with a basic prompt
2. Identify which examples the model solved correctly
3. Use successful input-output pairs as few-shot demonstrations
4. Optionally: include the model's reasoning traces (chain-of-thought)

Why it works:

- Examples the model "understands" transfer better
- Diverse examples cover more edge cases
- Self-generated traces match the model's reasoning style

Best for: Tasks where examples matter more than instructions

Method 2: MIPROv2

MIPRO = Multi-prompt Instruction Proposal Optimizer

Jointly optimize *both* instructions and few-shot examples using Bayesian optimization

How it works:

1. Generate candidate instructions by prompting an LLM to propose variations
2. Use Bayesian optimization (surrogate model) to explore the space efficiently
3. Jointly search over: instruction text, which examples, example ordering
4. Evaluate candidates on a validation set with your metric

Key insight: Instructions and examples interact — optimizing them jointly outperforms optimizing separately

Best for: Complex tasks where both instructions and examples matter

Method 3: GEPA

GEPA = Greedy Example and Prompt Augmentation

Augment prompts with task-specific information extracted from examples

How it works:

1. Analyze training examples to extract patterns and edge cases
2. Generate "tips" or guidelines based on common failure modes
3. Greedily add tips that improve validation performance
4. Result: instruction prompt augmented with learned heuristics

Example tip discovered automatically:

> *"When the text contains negation words like 'not' or 'never', pay extra attention to the overall sentiment direction."*

Best for: Tasks with subtle patterns that benefit from explicit guidance

DSPy: Comparison of Methods

	Bootstrap	MIPROv2	GEPA
Optimizes examples	✓	✓	✓
Optimizes instructions		✓	✓
Learns task-specific tips			✓
Compute cost	Low	High	Medium
Data needed	Few	Moderate	Moderate

Typical workflow:

1. Start with Bootstrap Few-Shot (quick baseline)
2. If not good enough, try MIPROv2 (more thorough)
3. For specialized domains, consider GEPA (learns explicit rules)

Rule of thumb

If you're manually iterating on prompts for >1 hour, consider DSPy

Section 6

Conclusion

Key Takeaways

1. **LLMs are powerful tools, not magic**
   - ▷ Understand the basics: tokens, temperature, context windows
   - ▷ Prompt quality matters as much as model choice

2. **Systematic approaches outperform ad-hoc prompting**
   - ▷ DSPy: automate prompt optimization
   - ▷ Verification pipelines: iterate until correct

3. **Integration into workflows is the real multiplier**
   - ▷ Claude Code: AI meets your actual development environment
   - ▷ Multi-agent workflows for parallel productivity

Questions?

Thank you!

Sam Davenport
UC San Diego

Slides and examples available at:
`https://github.com/sjdavenport/llm-presentation`