Prompt Engineering: an Evolution of Software Engineering

BuildClub.ai Bali Chapter – 22 March 2025

By: Christoforus Yoga Haryanto
Companion materials: https://github.com/cyharyanto/prompt-collections

Quick Poll: Your AI Experience

Where are you in your AI journey?

- Just starting to explore AI tools for development
- Using AI occasionally for specific coding tasks
- Regularly using AI in your development workflow
- Working on building AI-powered applications
- Other (please share)

This presentation will be very high density. The slide deck and accompanying materials will be available. The goal will be to help anyone to start and explore.

The Constant Evolution of Our Field

Era	Paradigm	Engineer's Focus
1940s-50s	**Hardwired Programming**	Physical wiring and circuits
1960s-70s	**Punch Cards & Assembly**	Machine-level instructions
1980s-90s	**Desktop Applications**	APIs and libraries
2000s-10s	**Cloud & Mobile**	Services and platforms
2020s+	**LLM-Augmented Development**	Intent and system architecture

Engineering Disciplines: Mature vs. Emerging Tools

Profession	Core Mission	Yesterday	Today
Accounting	Financial governance, risk management, and strategic insight	Accountant + Bookkeeper + Data Entry + AP/AR Team	Accountant + Integrated Financial Platforms
Civil Engineering	Creating safe, sustainable infrastructure for human flourishing	Civil Engineer + Drafters + Analysis Team	Civil Engineer + Advanced Simulation Suites
Software Engineering	Building reliable, ethical systems that transform human experience	BA + Developer + QA + DevOps + Tech Writer	Software Engineer + AI

The True Purpose of Software Engineering

- **Beyond functionality:**
  Augmenting human capability and experience
- **Ethical responsibility:**
  Innovation with privacy, security, and societal impact
- **Knowledge integration:**
  Domain expertise with technical implementation
- **Future-proofing:**
  Building systems resilient to evolving needs and contexts

The New Engineering Paradigm

- Other engineering disciplines have mature simulation tools that model entire complex systems
  - Aerospace engineers model complete Boeing 787 behavior before building
  - Electronic engineers work at intent level, not individual connections
  - Civil engineers simulate entire cities and infrastructure networks
- Software engineering is finally catching up
  - Moving from code-level thinking to intent-driven development
  - Shifting from implementation details to architectural patterns
  - Working at higher abstraction levels previously impossible
  - Focusing on what systems should do rather than how they do it

The Critical Paradigm Shift

- **Physical vs. Logical Complexity:** Software lacks physical constraints that made simulation central to other disciplines
- **Intent Translation:** LLMs excel at bridging human intent and machine execution
- **Abstraction Elevation:** We're no longer constrained to thinking at the syntax level
- **Knowledge Leverage:** LLMs encode vast software patterns and practices
- **Where We're Headed:** Not yet at "specify entire system, AI handles details" but rapidly approaching

Prompt Engineering as Intent Specification

For Software Engineers USING LLMs:

- Intent vs. Implementation: Express what you need, not how to do it (some limitations apply – niche know-how)
- System-Level Thinking: Focus on architecture and patterns
- Contextual Framing: Provide relevant project context

For Building LLM-POWERED Software:

- User Intent Extraction: Design systems that understand ambiguous requests (some limitations – at least to clarify)
- Domain Knowledge Encoding: Embed expert knowledge in prompts
- Error Recovery Pathways: Plan for misinterpretations and edge cases

Beyond Coding: Enabling Previously Impossible

Traditional Programming Approach

```
// To transform complex medical notes into
patient-friendly instructions:

// 1. Create NLP preprocessing (thousands of lines)

// 2. Build domain-specific medical terminology database

// 3. Implement context-aware semantic analysis

// 4. Design readability scoring system

// 5. Create grammar/syntax correction system

// RESULT: Years of specialized development with limited
adaptability
```

Prompt Engineering Approach

You are a medical communication specialist. Transform these technical doctor's notes into simple, clear instructions for the patient. Use plain language at an 8th-grade reading level, organize instructions chronologically, highlight critical care information, and include simple explanations for any medical terms that cannot be avoided.

LLM Mechanics: Training Data & Statistical Patterns

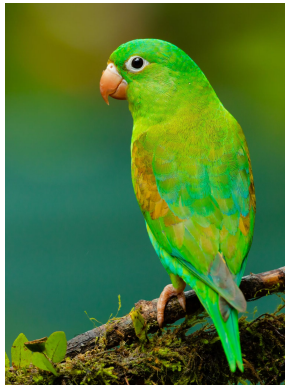
Understanding the Foundation

- **Knowledge Representation:** LLMs encode patterns from billions of texts
- **Statistical Reasoning:** Predictions based on token probability distributions
- **Knowledge Boundaries:** Training cutoff limits and domain coverage

Implications for Engineers

- **Data Mirroring:** Models reflect patterns in their training data
- **Instruction Following:** How explicit directions shape output
- **Uncertainty Handling:** Managing hallucinations and speculation

In machine learning, the term **stochastic parrot** is a metaphor to describe the theory that large language models, though able to generate plausible language, do not understand the meaning of the language they process.



LLM Attention Limit

"Lost in the Middle" Phenomenon

- **$O(n^2)$ Attention Distribution:** Fixed "attention budget" spreads thinner as context grows
- **Positional Dilution:** Self-attention prioritizes tokens at boundaries (beginning frames context, end is generation point)
- **Attention Mechanism Design:** Each token's influence diminishes with distance, creating a U-shaped attention curve

Long Context Computational Challenges

- **Quadratic Scaling:** Self-attention complexity is $O(n^2)$ – doubling context creates 4× compute load
- **KV Cache Growth:** Memory usage increases linearly with context as each token's representation must be stored
- **Layer Multiplicative Effect:** Computational burden multiplies across each transformer layer and attention head

Context Sensitivity & Management

For Software Engineers USING LLMs:

- **Codebase Representation:** Effectively describing your project's architecture
- **Development History:** Providing relevant background and evolution
- **Technical Requirements:** Highlighting critical constraints and standards
- **Knowledge Prioritization:** Focusing on what matters most for the task

Just think like working with someone super-smart yet oblivious to the context unless you told (or can be “seen”).

For Building LLM-POWERED Software:

- **System Context Design:** Creating robust guardrails in system/meta-prompts
- **User Context Management:** Maintaining conversation state and history
- **Memory Architecture:** Designing what information persists and how
- **Information Hierarchy:** Structuring prompts for optimal attention

Imagine you are developing an operating procedure to be understood by someone highly knowledgeable yet oblivious to the surrounding.

Goal-oriented & Constraint-based Prompting

For Software Engineers USING LLMs:

- **Explicit Objectives:** Clearly state what you're trying to accomplish
- **Technical Constraints:** Specify languages, frameworks, and standards
- **Quality Attributes:** Emphasize security, maintainability, testability

For Building LLM-POWERED Software:

- **System Boundaries:** Define what the AI should and shouldn't do
- **Failsafe Mechanisms:** Plan for graceful handling of edge cases
- **Evaluation Criteria:** Establish how outputs will be judged

Anatomy of Effective Prompts

Core Components:

- **Context Setting:** Establishes background and framing ("You are a senior security engineer reviewing code...")
- **Role Definition:** Specifies the expertise or perspective to adopt ("Act as a database optimization expert...")
- **Task Description:** Defines the specific action required ("Refactor this code to improve performance...")
- **Constraints:** Establishes boundaries and requirements ("Must maintain backward compatibility...")
- **Output Format:** Specifies how results should be presented ("Provide the solution as a numbered list...")

Hierarchical Structure:

- **System-level instructions:** Global rules and constraints that apply throughout
- **Task-level instructions:** Specific requirements for the current request
- **Example patterns:** Few-shot demonstrations of desired inputs and outputs
- **Evaluation criteria:** How success will be measured or validated

The Fundamental Constraint of LLMs

No Context, No Intelligence

- **No Inherent Goals:** Unlike humans with internalized motivations, LLMs have no intrinsic reward functions
- **Context Boundary:** Even the most advanced models cannot reason about information they haven't seen
- **Pre-training Limitations:** Training objectives shape capabilities but don't translate to specific application goals
- **Agentic Illusion:** Self-planning "agentic" models still operate within the context boundary - they cannot "want" things

Practical Implications

- Models cannot magically intuit your unstated requirements
- Explicit is better than implicit in all prompt engineering
- The quality of your context directly determines the ceiling of possible outputs
- Human judgment essential for appropriateness and alignment with unstated goals

Remember: the context window is limited.

Challenge:

- How we can express everything?
- How we can make sure the right emphasis?

The P.R.O.M.P.T Framework for Engineers

Purpose: Define clear objective and success criteria

Reference: Provide relevant context and examples

Organize: Structure information logically and prioritize

Model: Understand the LLM's capabilities and limitations

Precision: Use specific, unambiguous language

Test: Iterate and refine based on outputs

Key Takeaways

- Software engineering is evolving toward higher-level abstraction
- For personal use: master intent specification and constraint articulation
- For product development: focus on robustness, domain knowledge integration, and user safety

Remember: The skills differ between USING LLMs as a tool vs. building LLM-POWERED systems

Hands-on Challenges & Demo

For Software Engineers USING LLMs:

- **Code Architecture Transformation:** Convert a monolithic class into a microservices design
- **Test Suite Generation:** Create comprehensive tests for an existing method
- **Documentation Automation:** Generate clear documentation from undocumented code

For LLM-POWERED Software Builders:

- **Intent Parser Design:** Create a prompt that accurately extracts structured data from user requests
- **Jailbreak Detection:** Design a prompt that can detect when users attempt to bypass AI safeguards
- **Domain Expert Collaboration:** Transform expert knowledge into an effective prompting system

Knowledge Boundaries & Advanced Topics

- **Hallucination Management:** Detecting and correcting speculative outputs
- **Vulnerabilities:** Understanding security implications in both use cases
- **RAG Integration:** Combining LLMs with domain knowledge
- **Evaluation Frameworks:** Systematically assessing output quality and safety

Thank You!

Let's move to the hands-on session