You said:

Metis Reaction Training Lights - Comprehensive Guide.pdf

PDF

REACT.pdf

PDF

I have created the below generic context engineering framework. I want you to review it and see how we can use it to build an extremely accurate context laden prompt with various steps etc that we can share with cursor ai (it will be using clade code within), build a feature rich and simple to use front end UI for the Queling Reactionx app for the reaction training lights. The Metis Reaction Training lights that I have are exactly the same but just rebranded. The purposed of this exercise is to have a simple UI where a football coach for example, can write in natural language the drill he wants to do and the ui will connect to the back end app and configure the lights and all the required parameters from his or her natural language input. I am not expecting to get the context engineering output correct first time but I would prefer to iterate the context over a number of times to get it correct rather than trying to do that in cursor. I have also compiled a google notebooklm with 102 videos of the same reaction lights from the manufacturer although many have no transcript so I will need to get a detailed description of what is happening in those videos to further the knowledge base of the possibilities of the lights. I also have a doc I have attached that I have built over many hours documenting all the possibilities within the app and all the possible parameters within each section. below is the context engineering framework I have built, and I am also open to improving that framework too if you have anything that notice would improve it and ultimately the output of the final UI . . .

Context Engineering: Framework for Intelligent AI Systems

---------------

A Comprehensive Framework for Context Engineering in Artificial Intelligence

Context Engineering represents a fundamental and critical shift in the development of intelligent Artificial Intelligence (AI) systems, moving beyond the traditional focus on merely crafting effective prompts. It is rapidly becoming the most essential skill for anyone working with advanced AI systems and Large Language Models (LLMs), as it defines how intelligent systems behave and ensures they can accomplish complex tasks reliably.

At its core, Context Engineering is the discipline of designing and building dynamic systems that provide "the right information and tools, in the right format, at the right time" for LLMs and AI agents, enabling them to plausibly accomplish complex tasks. This comprehensive approach is essential because, as AI models become more capable, their failures often stem from a lack of appropriate context rather than inherent model limitations. It's about designing the entire informational ecosystem an AI system utilizes. Context Engineering is considered the "crucial, often invisible force" that makes AI systems genuinely intelligent, useful, and relevant in the real world.

Distinguishing Context Engineering from Prompt Engineering

While "prompt engineering" focuses on crafting initial inputs or questions to guide LLMs, it represents only a small fraction—potentially as little as 0.1% to 5%—of the total context an advanced AI system might process. The AI prompt you write becomes a "tiny drop in an ocean of context" that these models discover and synthesize autonomously.

Context Engineering is widely described as a "step up" from prompt engineering, encompassing the entire informational ecosystem. Prompt engineering is considered a subset of context engineering. The focus shifts from "magic wording" to providing a complete and structured context, especially as AI applications evolve from single prompts to complex, dynamic agentic systems.

Core Principles of Context Engineering

Effective context engineering is built upon several foundational principles that guide the design and management of information flow to AI systems:

1. Relevance and Selectivity: This is considered the most fundamental principle. Every piece of information, or "token," in an LLM's context window is valuable. It is crucial to filter out irrelevant or redundant information and prioritize content based on its importance to the current task. LLMs can struggle and degrade in performance when "flooded with large amounts of unfiltered information" or when relevant information is "buried inside the middle portion," often referred to as "Lost in the Middle" or "context rot". The system must dynamically adapt the context as the conversation or use case evolves.

2. Structure and Formatting: The way information is presented to an LLM is as important as the information itself. This includes strategically ordering information (e.g., placing the most critical details first), using clear formatting like tables or bullet points, and explicitly distinguishing between facts, instructions, and references to effectively guide the model's attention. A well-structured "context schema" acts like a "neatly organized filing cabinet" for the AI, ensuring it only sees what's needed at each turn.

3. Dynamic Adaptation: Context is not static; it must be engineered to change dynamically. In ongoing conversations or systems reacting to real-world changes (like real-time market data or sensor inputs), context needs to be continually updated, incorporating new data streams and managing memory efficiently. Systems should be designed for continuous evolution, learning from user input and self-adaptation. This also encompasses efficient memory management, allowing AI systems to store, save, and retrieve context over extended periods to provide continuity and build on past experiences.

Two-Layer Context Architecture

Nate's Substack introduces a "two-layer AI system architecture" consisting of deterministic and probabilistic context, which fundamentally reshapes how LLMs process information.

1. Deterministic Context: This layer includes information directly controlled by the user or developer. It is the "stuff you control" and forms the initial foundation for the AI's operations. Key components include:

◦ AI Prompts: The initial questions, commands, or queries that kick off an interaction.

◦ Uploaded Documents: Any files, reports, or data directly provided to the AI, such as PDFs or text files. For AI coding assistants like Claude Code, basic Markdown files can store user-specific memories or preferences as part of this layer.

◦ System Instructions: Explicit rules, constraints, or guidelines defining the AI's persona, goals, or how it should behave. These can be general rules, task management guidelines, or style conventions, often set up in global rule files.

◦ Product Requirements Prompts (PRPs): For AI coding assistants, PRPs are comprehensive plans designed to instruct the AI and create an engineered context for implementation, often generated via "slash commands".

◦ Examples: Providing few-shot examples that demonstrate how to perform a task shows the AI the desired pattern rather than just abstractly telling it.

2. Probabilistic Context: This refers to the vast web of information that AI agents autonomously find and integrate on their own, often without direct, conscious input from the user or developer. This layer represents the AI's ability to actively search, pull, and synthesize information from a broad "informational ecosystem" that extends beyond the initial prompt. Examples include an AI searching hundreds of websites, pulling data from Google Drive, connecting to databases, or receiving tool feedback. For instance, Claude was observed searching over 500 sources to research a topic, where the original prompt was only about 0.1% of the total context processed.

Understanding this distinction is crucial because traditional prompt optimization, which focuses on making prompts shorter and cheaper, "completely misses the point" when AI agents are processing massive, autonomously gathered contexts that users cannot directly control.

Key Strategies and Techniques for Context Management

To implement the core principles effectively and leverage the two-layer architecture, various strategies and techniques are employed:

1. Data Gathering and Curation:

◦ Identification of Sources: This initial and crucial step involves identifying all potential sources of relevant information for the AI system. This includes both internal sources (documents, databases, APIs, past user interactions, and proprietary knowledge bases) and external sources (the web, news feeds, scientific literature, real-time market data, and social media sentiment).

◦ Curation: Once identified, these sources must be meticulously curated by filtering out noise and selecting the most authoritative or current content. This prevents the "data dump fallacy," where simply connecting an AI to every available data source without intelligent curation results in "overwhelming noise" and degrades performance.

2. Contextual Compression and Summarization:

◦ Given the finite size of an LLM's context window (its "RAM" or "working memory"), techniques to maximize the utility of this limited space are essential.

◦ Summarization: This is a powerful and common technique for managing context growth. It involves distilling older parts of a conversation, full agent trajectories (recursive/hierarchical summarization), or tool call feedback (e.g., summarizing 10 pages of web results into a concise overview). Smaller, faster, and potentially cheaper LLMs can be used to prepare context for larger, more expensive LLMs.

◦ Token Optimization/Semantic Density: Techniques like LLMLingua aim to reduce token count while maintaining "maximum information density" by removing unimportant tokens or transforming text into a form challenging for humans but well-understood by LLMs. However, correctness should always trump compression, as context failures can cost exponentially more than token expenses.

◦ Addressing Information Overload ("Lost in the Middle"): LLMs struggle to extract valuable information from very large contexts, especially when relevant information is "buried inside the middle portion". Providing too much unfiltered information can degrade performance. Efficient compression helps mitigate this.

3. Persistence (Memory Management):

◦ For AI systems to provide continuity and build on past experiences, which is vital for personalized and human-like interaction, context engineering includes sophisticated memory management mechanisms to store, save, and retrieve context over extended periods.

◦ Storage Mechanisms: Include basic files (e.g., Markdown files for user-specific memories or preferences like in Claude Code), embedded documents/vector databases (converting text into numerical "embeddings" for semantic search), and Knowledge Graphs (KGs) that explicitly map connections and relationships between facts and entities in data. KGs help eliminate AI hallucinations by rooting outputs in verifiable knowledge.

◦ Memory Creation/Updating: Memories can be user-initiated (e.g., explicitly saving parts of a conversation via Claude Code's shortcut) or autonomous. Advanced agents can generate their own "hints" or "reflections" after each turn, learning from mistakes and refining their approach (e.g., the "reflection paper"). Direct user feedback, like corrections to an email assistant's drafts, can also update stored preferences.

◦ Retrieval: For large collections, selective fetching is crucial, using techniques like embedding-based similarity search or graph traversal queries. Sophisticated scoring functions may combine semantic similarity, recency, and perceived importance to mimic human recall.

4. Isolation:

◦ This strategy involves partitioning information across different agents, logical parts of the system, or environments to prevent information overload and improve the LLM's focus.

◦ Context Schemas: Act as a structured blueprint for the information an agent needs, allowing engineers to precisely control what the LLM sees at each turn and how it's presented.

◦ Multi-Agent Systems: Complex tasks can be split across specialized sub-agents, each with its own specific instructions, focused context window, and specialized tools. While this can lead to dramatic performance improvements (e.g., Anthropic's multi-agent system outperformed single-agent systems by 90.2% on benchmarks), it can also lead to higher overall token usage and requires "incredibly careful prompting and coordination" between sub-agents to avoid disjointed or contradictory outputs. Multi-agent systems work best for "easily parallelizable tasks".

◦ Environment Isolation for Code Agents: For agents interacting with external systems or generating code, the LLM can output code to be executed in a "separate controlled sandbox environment". This sandbox can store large objects generated during code execution, effectively isolating them from the LLM's main context window, freeing up the LLM's "precious mental bandwidth for higher level reasoning".

5. Tooling and Automation:

◦ Manual context engineering is unsustainable for most real-world applications; thus, leveraging tools and automation frameworks is critical.

◦ Retrieval-Augmented Generation (RAG): RAG systems retrieve relevant snippets from a knowledge base in real-time and provide them to the LLM. This enhances accuracy and credibility, especially for knowledge-intensive tasks, and allows for continuous knowledge updates. RAG architectures have evolved through Naive, Advanced, and Modular RAG paradigms.

◦ Vector Databases and Semantic Search: These are crucial for RAG, converting text into numerical "embeddings" to find semantically similar information quickly.

◦ Automated Context Builders: Frameworks like LangChain and LlamaIndex automate much of the context selection and structuring process. Anthropic's Model Context Protocol (MCP) is also emerging as a standard for structured context handling.

◦ Product Requirements Prompts (PRPs) and Slash Commands: For AI coding assistants like Claude Code, PRPs are comprehensive plans designed to instruct the AI, generated via "slash commands" to create an engineered context for implementation.

6. Evaluation and Iteration:

◦ Continuous improvement is essential for context-engineered systems.

◦ Testing and Feedback: This involves A/B testing different context configurations, analyzing LLM output quality and relevance, and collecting user feedback on system performance.

◦ Observability Tools: Tools like LangSmith help by tracing agent calls and showing exact inputs/outputs to the LLM for debugging context quality and formatting.

◦ Rapid Iteration: Rapid iteration and testing are crucial given the fast pace of AI development.

7. Human-AI Collaboration:

◦ Context engineering is not solely a technical endeavor; it requires deep domain knowledge and understanding user needs.

◦ Cross-Functional Teams: Close collaboration between domain experts (business units), AI engineers, and users is essential.

◦ Defined Roles: Business units must identify which context sources matter and define quality standards, while technology teams build the integration architectures and ensure data quality. This collaboration ensures that the AI system truly understands the specific business context, internal rules, customer history, and market position, moving beyond generic responses to genuine business understanding.

Challenges and Pitfalls in Context Engineering

Despite its immense promise, context engineering is not without significant challenges that developers must navigate:

• Information Overload and "Lost in the Middle": LLMs can struggle to extract valuable information from very large contexts, especially when relevant information is "buried inside the middle portion".

• Computational Constraints and Cost: Building, summarizing, and formatting context, especially in real-time, can be computationally intensive, impacting latency, cost, and scalability. Longer contexts generally lead to higher costs per token processed.

• Context Degradation Syndrome (Context Rot): When too much irrelevant information accumulates in the AI's working memory, it becomes harder for the AI to recall specific facts or follow complex instructions accurately, leading to performance degradation.

• Overfitting and Irrelevance: Including too much or the wrong kind of information can distract the model, introduce contradictions, or lead to contexts that work for one specific case but don't generalize well.

• Human-AI Collaboration Issues: Misunderstanding user intent, insufficient transparency about what information the AI is using, and biases in human context selection are common pitfalls.

• The Data Dump Fallacy: Simply connecting an AI to every available data source without intelligent curation and structuring results in "overwhelming noise" and degrades performance.

• The Silo Trap: Building separate, isolated context systems for different departments defeats the purpose of unified intelligence across the organization.

• The Static Context Mistake: Business contexts are dynamic; context engineering systems must be continuously updated, not rigid snapshots.

• Security Risks: Context-aware systems face documented vulnerabilities, including prompt injection through Model Context Protocol (MCP) channels, cross-tenant contamination risks, model poisoning, and data leakage. "Bad actors are already figuring out how to manipulate AI model behavior through poisoned web content and compromised data sources". Implementing VPC deployments, role-based access controls, and audit logging are crucial defensive measures. Security, privacy, and governance "cannot be bolted on at the end" but must be "foundational elements baked into the architecture and processes from day one".

• Initial Complexity and Upfront Investment: Setting up a context engineering framework can be complex and demanding initially. It requires "a significant upfront investment in data integration and knowledge graph creation" and "a lot of time up front creating this context".

Real-World Impact and Future Prospects of Context Engineering

Context Engineering is already delivering tangible business value across diverse industries by enabling AI systems to operate with deeper understanding, greater accuracy, and enhanced relevance in complex, real-world scenarios.

Real-World Impact:

• Enhanced Accuracy and Reduced Hallucinations: A primary benefit is the ability to significantly reduce hallucinations by connecting AI systems to "governed facts" and verifiable, real-world knowledge, often through Retrieval-Augmented Generation (RAG) and Knowledge Graphs. This is crucial for building user trust and ensuring the AI provides accurate and credible information.

• Access to Up-to-Date and Proprietary Knowledge: Context engineering, particularly through RAG, enables LLMs to access real-time, specific, or even proprietary information that was not part of their initial training data. This is vital for applications in dynamic fields like financial services or highly specialized sectors like healthcare.

• Tangible Business Outcomes Across Industries: Examples include financial services (40% reduction in client meeting prep time), healthcare (integrating patient records with research for accurate diagnoses), e-commerce (35% increase in click-through rates), smart cities (20% reduction in average commute times), legal research, and AI coding assistants like Claude Code, which use context engineering to generate comprehensive project plans and implement full projects end-to-end with just a few prompts.

• Cost and Efficiency: While LLM context windows are expanding (e.g., Google's Gemini 1.5 Pro allows 2 million tokens), simply "context stuffing" can lead to decreased answer quality, increased hallucination risk, and higher costs. Context engineering techniques like chunked retrieval and on-demand input significantly improve operational efficiency and inference speed by presenting only the most relevant and valuable information.

• Transparency and Traceability: RAG-based generation allows users to quickly locate the original references for LLMs, enabling verification of generated answers and making the retrieval and reasoning process observable, unlike relying solely on long context, which often remains a "black box".

• Scalability: For complex, multi-step, data-rich, or dynamic tasks, context engineering offers a viable way to build robust, enterprise-grade AI systems. It enables multi-agent systems to function effectively by providing focused context windows.

• Human-Centered AI Development: Context engineering inherently aligns with human-centered design principles by focusing on providing comprehensive, relevant, and dynamically managed information needed to understand and respond to nuanced human needs and real-world scenarios.

Future Prospects: The future of context engineering is dynamic and closely interwoven with the continuous evolution of Large Language Models and the increasing sophistication of autonomous AI agents. Key future prospects include:

• Evolving LLM Capabilities and Expanding Context Windows: As LLM context windows continue to grow (e.g., Google's Gemini 1.5 Pro allows 2 million tokens), the potential for richer and more nuanced context engineering expands. However, context engineering remains vital because simply providing massive amounts of context can still impact inference speed, lead to the "Lost in the Middle" problem, and lack traceability. Developing new RAG methods specifically for these "super-long contexts" is a future research trend.

• Advancements in RAG Paradigms: There is an ongoing shift towards "Modular RAG," which offers enhanced adaptability and versatility by integrating new specialized components like Search, Memory, Routing, and Predict modules.

• Hybrid Approaches (RAG + Fine-tuning): Combining RAG with fine-tuning is emerging as a leading strategy to optimize model performance, allowing for real-time knowledge updates from RAG and deep customization from fine-tuning.

• Multi-modal RAG: RAG is expanding its application scope into multi-modal domains, interpreting and processing diverse data forms such as images, audio, video, and code.

• Production-Ready RAG: Efforts are intensifying to enhance retrieval efficiency, improve document recall in large knowledge bases, and ensure data security, privacy, and governance for real-world deployment. The RAG ecosystem is growing, with specialized tools and frameworks like LangChain and LlamaIndex providing extensive RAG-related APIs.

• AI Anticipating User Needs: The ultimate long-term vision of context engineering is to enable AI systems to become so intelligent and attuned to human needs that they can genuinely anticipate information needs even before they are consciously articulated by users. This transforms AI from a simple query-response machine into a truly knowledgeable and powerful partner that deeply understands specific business contexts, personal needs, history, and goals.

Conclusion

Context engineering marks the maturation of enterprise AI, moving it from exciting experimental technology to a core strategic operational capability. It is the "crucial, often invisible force" that makes AI systems genuinely intelligent, useful, and relevant in the real world. By mastering context engineering, organizations can ensure their AI systems profoundly understand their unique businesses, anticipate needs, maintain institutional memory, apply company-specific logic, respect governance, and scale seamlessly, ultimately leading to faster, more confident decision-making, reduced operational costs, improved compliance, and the ability to identify market opportunities. It's about designing an AI that doesn't just respond generically but genuinely understands and operates within your specific business context, unique data, internal rules, customer history, and market position.

--------------------------------------------------------------------------------

ChatGPT said:

Request for o3 pro

Reasoned for 3m 27s

Below is Version 0.1 of a working plan that links your Context‑Engineering framework to the practical goal: “a coach types a drill → the UI configures the Queling / Metis lights automatically.”

It is deliberately modular so we can iterate quickly; feel free to tell me which chunk you’d like to deepen next.

🧭 Executive snapshot

What we already have What we still need This plan delivers

• A rich Context‑Engineering manifesto (your text)

• Two authoritative PDFs enumerating every game‑mode & setting of the lights • Machine‑readable catalogue of those modes

• Transcripts / descriptions of 102 reference videos

• A deterministic ⇄ probabilistic context pipeline that Cursor AI (Claude Code) can consume • Concrete upgrades to your framework

• A prompt‑schema + JSON output spec that turns NL drills → device parameters

• A thin‑yet‑powerful front‑end concept

• A step‑by‑step build & evaluation loop

1 🧠 Quick review of your Context‑Engineering framework

Strengths

Comprehensive taxonomy – covers relevance, formatting, adaptation, persistence, isolation, tooling, security, human factors.

Two‑layer architecture – deterministic vs. probabilistic context gives a clean mental model.

Explicit pitfalls – “Lost in the Middle”, context‑rot, data‑dump fallacy, etc.

Improvement opportunities (v0.1)

Gap Enhancement

No formal metadata spec for documents / tools Adopt a light Context Schema Manifest (CSM): YAML describing each asset with id, type, modality, recency, quality, security\_tag.

Evaluation section is high‑level Add quantitative metrics: Factuality @k, Parameter‑Hit‑Rate, Coach‑override‑rate, Latency (ms).

Role responsibilities not mapped Use RACI inside the framework (Responsible, Accountable, Consulted, Informed) for business, AI‑engineering, coaches, QA.

Security talks about risks, but not mitigations Embed Context Boundary Tests (CBTs) – red‑team prompts automatically injected during CI to detect leakage / injection.

No explicit versioning Add semantic version tags (CONTEXT\_SPEC: 0.1.0) to every deterministic asset so Cursor agents can diff & migrate.

2 🔧 Apply the framework to the Reaction‑Lights product

2.1 Knowledge‑asset inventory

Asset Deterministic? Ingestion task

Metis “Comprehensive Guide” PDF (22 modes, full parameters) ✅ PDF → Markdown → JSON catalogue.

React user manual (setup, offline, connectivity limits) ✅ Same as above.

102 product videos ↗ Probabilistic Whisper + lightweight human QA → description & timestamped actions.

Future coach feedback logs ↗ Probabilistic Append to vector store for retrieval.

2.2 Data pipeline outline

Parsing & Normalisation

pdf\_parser.py → table rows (mode, field, default, range, description).

Embedding + Vector DB (e.g., Weaviate) keyed by mode and motion‑cue.

Video transcription & action tagging

Whisper auto‑transcribes.

Small LLM (e.g., gpt‑3.5‑turbo) classifies clips by action\_type, skill\_goal, lighting\_pattern.

Periodic summarisation to reduce token load – persisted as “knowledge snapshots”.

3 🏗️ Prompt & Agent design (first iteration)

3.1 System prompt (deterministic layer)

text

Copy

Edit

You are ReactionX‑Config‑Agent v0.1.

You receive:

• COACH\_REQUEST: unstructured natural‑language drill description

• LIGHTS\_CATALOGUE: JSON of 22 modes & parameters (latest version hash …)

You must:

1. Extract intent: primary objective, player count, surface size, constraints.

2. Select the most suitable MODE or MODE\_COMBINATION.

3. Fill all parameters. Use defaults when unspecified; optimise others to maximise objective.

4. Return a strictly valid JSON conforming to SCHEMA v0.1 (below).

5. Provide a human‑readable one‑paragraph rationale.

(Place SCHEMA here – see §3.3)

3.2 Few‑shot examples (seed two good drills + one tricky edge‑case).

These live in deterministic context and are rotated as we gather real coach prompts.

3.3 Structured output schema (draft)

jsonc

Copy

Edit

{

"mode": "Standard | AllAtOnce | Sequence | …",

"num\_lamps": 8,

"colours": ["red","green"],

"targets": 2,

"delay": {"min": 0.3, "max": 0.7},

"timeout": 1.5,

"sensor\_activation": "close",

"flash": false,

"timer": 30,

"start\_delay": 0,

"notes": "Return to base cone after each touch",

"confidence": 0.84

}

NB: All field names match those in the Metis PDFs to avoid translation errors.Metis Reaction Training…

3.4 Sub‑agent isolation (optional but recommended)

Agent Focus Context window

NL‑to‑Intent parse drill COACH\_REQUEST only

Parameter‑Synthesiser map intent→mode→params Intent + LIGHTS\_CATALOGUE

Validator JSON schema + safety checks Synthesised JSON

Explainer rationale for UI final JSON

This keeps each window small and debuggable.

4 🖥️ Front‑end concept (“Coach Console”)

Plain‑language textbox – “Describe your drill…”.

Live Draft Panel – renders parsed JSON + rationale. Coach can toggle Advanced view to tweak any parameter.

Simulate button – flashes virtual lights on screen so the coach visualises sequence.

Deploy button – sends JSON to backend / Bluetooth.

Templates library – saved drills indexed by tags (age group, skill, duration).

Minimal stack: React + Tailwind, call Cursor API endpoint /generate\_config.

5 📊 Evaluation & iteration loop

Stage Metric Tool

Offline unit tests 100% JSON schema validity Jest + Ajv

Simulated drills Parameter‑Hit‑Rate ≥ 95 % Cypress

Field feedback Coach override count per session Mixpanel

LLM quality BLEU between suggested & accepted configs Custom script

Logs feed back into vector store → automatic fine‑tuning or reranking.

6 🔒 Security checkpoints

Input sanitiser strips or encodes suspicious content before feeding LLM.

Context Boundary Tests run in CI (malicious coach prompt trying to expose manuals, etc.).

Bluetooth auth – use device‑id white‑list from manual’s limits table REACT.

7 🚀 Next actions

Horizon Task Owner

48 h Parse both PDFs → JSON catalogue (I can supply parsing script) AI eng.

48 h Whisper transcribe first 10 videos, review tags Data ops

72 h Implement prompt & schema in Cursor sandbox Prompt eng.

96 h Build React textbox + live draft component Front‑end

1 wk On‑pitch beta with two coaches, collect overrides Sports science

Where would you like to go deeper in Version 0.2?

Examples: refine the JSON schema, design the multi‑agent prompts, or map out the video‑annotation taxonomy.

I’m ready when you are.

You said:

Below is an improve framework, so can you run that agian using the new improved framework

Context Engineering: Framework for Intelligent AI Systems

---------------

A Comprehensive Framework for Context Engineering in Artificial Intelligence

Context Engineering represents a fundamental and critical shift in the development of intelligent Artificial Intelligence (AI) systems, moving beyond the traditional focus on merely crafting effective prompts. It is rapidly becoming the most essential skill for anyone working with advanced AI systems and Large Language Models (LLMs), as it defines how intelligent systems behave and ensures they can accomplish complex tasks reliably.

At its core, Context Engineering is the discipline of designing and building dynamic systems that provide "the right information and tools, in the right format, at the right time" for LLMs and AI agents, enabling them to plausibly accomplish complex tasks. This comprehensive approach is essential because, as AI models become more capable, their failures often stem from a lack of appropriate context rather than inherent model limitations. It's about designing the entire informational ecosystem an AI system utilizes. Context Engineering is considered the "crucial, often invisible force" that makes AI systems genuinely intelligent, useful, and relevant in the real world.

Distinguishing Context Engineering from Prompt Engineering

While "prompt engineering" focuses on crafting initial inputs or questions to guide LLMs, it represents only a small fraction—potentially as little as 0.1% to 5%—of the total context an advanced AI system might process. The AI prompt you write becomes a "tiny drop in an ocean of context" that these models discover and synthesize autonomously.

Context Engineering is widely described as a "step up" from prompt engineering, encompassing the entire informational ecosystem. Prompt engineering is considered a subset of context engineering. The focus shifts from "magic wording" to providing a complete and structured context, especially as AI applications evolve from single prompts to complex, dynamic agentic systems.

Core Principles of Context Engineering

Effective context engineering is built upon several foundational principles that guide the design and management of information flow to AI systems:

1. Relevance and Selectivity: This is considered the most fundamental principle. Every piece of information, or "token," in an LLM's context window is valuable. It is crucial to filter out irrelevant or redundant information and prioritize content based on its importance to the current task. LLMs can struggle and degrade in performance when "flooded with large amounts of unfiltered information" or when relevant information is "buried inside the middle portion," often referred to as "Lost in the Middle" or "context rot". The system must dynamically adapt the context as the conversation or use case evolves.

2. Structure and Formatting: The way information is presented to an LLM is as important as the information itself. This includes strategically ordering information (e.g., placing the most critical details first), using clear formatting like tables or bullet points, and explicitly distinguishing between facts, instructions, and references to effectively guide the model's attention. A well-structured "context schema" acts like a "neatly organized filing cabinet" for the AI, ensuring it only sees what's needed at each turn.

3. Dynamic Adaptation: Context is not static; it must be engineered to change dynamically. In ongoing conversations or systems reacting to real-world changes (like real-time market data or sensor inputs), context needs to be continually updated, incorporating new data streams and managing memory efficiently. Systems should be designed for continuous evolution, learning from user input and self-adaptation. This also encompasses efficient memory management, allowing AI systems to store, save, and retrieve context over extended periods to provide continuity and build on past experiences.

Two-Layer Context Architecture

Nate's Substack introduces a "two-layer AI system architecture" consisting of deterministic and probabilistic context, which fundamentally reshapes how LLMs process information.

1. Deterministic Context: This layer includes information directly controlled by the user or developer. It is the "stuff you control" and forms the initial foundation for the AI's operations. Key components include:

◦ AI Prompts: The initial questions, commands, or queries that kick off an interaction.

◦ Uploaded Documents: Any files, reports, or data directly provided to the AI, such as PDFs or text files. For AI coding assistants like Claude Code, basic Markdown files can store user-specific memories or preferences as part of this layer.

◦ System Instructions: Explicit rules, constraints, or guidelines defining the AI's persona, goals, or how it should behave. These can be general rules, task management guidelines, or style conventions, often set up in global rule files.

◦ Product Requirements Prompts (PRPs): For AI coding assistants, PRPs are comprehensive plans designed to instruct the AI and create an engineered context for implementation, often generated via "slash commands".

◦ Examples: Providing few-shot examples that demonstrate how to perform a task shows the AI the desired pattern rather than just abstractly telling it.

2. Probabilistic Context: This refers to the vast web of information that AI agents autonomously find and integrate on their own, often without direct, conscious input from the user or developer. This layer represents the AI's ability to actively search, pull, and synthesize information from a broad "informational ecosystem" that extends beyond the initial prompt. Examples include an AI searching hundreds of websites, pulling data from Google Drive, connecting to databases, or receiving tool feedback. For instance, Claude was observed searching over 500 sources to research a topic, where the original prompt was only about 0.1% of the total context processed.

Understanding this distinction is crucial because traditional prompt optimization, which focuses on making prompts shorter and cheaper, "completely misses the point" when AI agents are processing massive, autonomously gathered contexts that users cannot directly control.

Key Strategies and Techniques for Context Management

To implement the core principles effectively and leverage the two-layer architecture, various strategies and techniques are employed:

1. Data Gathering and Curation:

◦ Identification of Sources: This initial and crucial step involves identifying all potential sources of relevant information for the AI system. This includes both internal sources (documents, databases, APIs, past user interactions, and proprietary knowledge bases) and external sources (the web, news feeds, scientific literature, real-time market data, and social media sentiment).

◦ Curation: Once identified, these sources must be meticulously curated by filtering out noise and selecting the most authoritative or current content. This prevents the "data dump fallacy," where simply connecting an AI to every available data source without intelligent curation results in "overwhelming noise" and degrades performance.

◦ Multi-Modal Context Handling: Incorporate non-text data by extracting descriptions (e.g., video frames/subtitles via tools like view\_x\_video) and converting them into structured text summaries to enrich the deterministic layer. This is crucial for applications involving visual demonstrations, like training drills.

2. Contextual Compression and Summarization:

◦ Given the finite size of an LLM's context window (its "RAM" or "working memory"), techniques to maximize the utility of this limited space are essential.

◦ Summarization: This is a powerful and common technique for managing context growth. It involves distilling older parts of a conversation, full agent trajectories (recursive/hierarchical summarization), or tool call feedback (e.g., summarizing 10 pages of web results into a concise overview). Smaller, faster, and potentially cheaper LLMs can be used to prepare context for larger, more expensive LLMs.

◦ Token Optimization/Semantic Density: Techniques like LLMLingua aim to reduce token count while maintaining "maximum information density" by removing unimportant tokens or transforming text into a form challenging for humans but well-understood by LLMs. However, correctness should always trump compression, as context failures can cost exponentially more than token expenses.

◦ Addressing Information Overload ("Lost in the Middle"): LLMs struggle to extract valuable information from very large contexts, especially when relevant information is "buried inside the middle portion". Efficient compression helps mitigate this.

3. Persistence (Memory Management):

◦ For AI systems to provide continuity and build on past experiences, which is vital for personalized and human-like interaction, context engineering includes sophisticated memory management mechanisms to store, save, and retrieve context over extended periods.

◦ Storage Mechanisms: Include basic files (e.g., Markdown files for user-specific memories or preferences like in Claude Code), embedded documents/vector databases (converting text into numerical "embeddings" for semantic search), and Knowledge Graphs (KGs) that explicitly map connections and relationships between facts and entities in data. KGs help eliminate AI hallucinations by rooting outputs in verifiable knowledge.

◦ Memory Creation/Updating: Memories can be user-initiated (e.g., explicitly saving parts of a conversation via Claude Code's shortcut) or autonomous. Advanced agents can generate their own "hints" or "reflections" after each turn, learning from mistakes and refining their approach (e.g., the "reflection paper"). Direct user feedback, like corrections to an email assistant's drafts, can also update stored preferences.

◦ Retrieval: For large collections, selective fetching is crucial, using techniques like embedding-based similarity search or graph traversal queries. Sophisticated scoring functions may combine semantic similarity, recency, and perceived importance to mimic human recall.

4. Isolation:

◦ This strategy involves partitioning information across different agents, logical parts of the system, or environments to prevent information overload and improve the LLM's focus.

◦ Context Schemas: Act as a structured blueprint for the information an agent needs, allowing engineers to precisely control what the LLM sees at each turn and how it's presented.

◦ Multi-Agent Systems: Complex tasks can be split across specialized sub-agents, each with its own specific instructions, focused context window, and specialized tools. While this can lead to dramatic performance improvements (e.g., Anthropic's multi-agent system outperformed single-agent systems by 90.2% on benchmarks), it can also lead to higher overall token usage and requires "incredibly careful prompting and coordination" between sub-agents to avoid disjointed or contradictory outputs. Multi-agent systems work best for "easily parallelizable tasks".

◦ Environment Isolation for Code Agents: For agents interacting with external systems or generating code, the LLM can output code to be executed in a "separate controlled sandbox environment". This sandbox can store large objects generated during code execution, effectively isolating them from the LLM's main context window, freeing up the LLM's "precious mental bandwidth for higher level reasoning".

5. Tooling and Automation:

◦ Manual context engineering is unsustainable for most real-world applications; thus, leveraging tools and automation frameworks is critical.

◦ Retrieval-Augmented Generation (RAG): RAG systems retrieve relevant snippets from a knowledge base in real-time and provide them to the LLM. This enhances accuracy and credibility, especially for knowledge-intensive tasks, and allows for continuous knowledge updates. RAG architectures have evolved through Naive, Advanced, and Modular RAG paradigms.

◦ Vector Databases and Semantic Search: These are crucial for RAG, converting text into numerical "embeddings" to find semantically similar information quickly.

◦ Automated Context Builders: Frameworks like LangChain and LlamaIndex automate much of the context selection and structuring process. Anthropic's Model Context Protocol (MCP) is also emerging as a standard for structured context handling.

◦ Product Requirements Prompts (PRPs) and Slash Commands: For AI coding assistants like Claude Code, PRPs are comprehensive plans designed to instruct the AI, generated via "slash commands" to create an engineered context for implementation.

◦ Tool Integration for Probabilistic Layer: Leverage external tools (e.g., web\_search, view\_x\_video) to dynamically populate probabilistic context, ensuring the AI can fetch real-time examples without over-relying on static data.

6. Evaluation and Iteration:

◦ Continuous improvement is essential for context-engineered systems.

◦ Testing and Feedback: This involves A/B testing different context configurations, analyzing LLM output quality and relevance, and collecting user feedback on system performance.

◦ Observability Tools: Tools like LangSmith help by tracing agent calls and showing exact inputs/outputs to the LLM for debugging context quality and formatting.

◦ Rapid Iteration: Rapid iteration and testing are crucial given the fast pace of AI development.

◦ Versioning and Iteration Mechanics: Implement context versioning (e.g., using Git-like tracking for context schemas) to log changes and A/B test versions. For iterative refinement, define feedback loops where AI outputs are evaluated against ground truth (e.g., user validation of generated configs) and used to update memory stores.

7. Human-AI Collaboration:

◦ Context engineering is not solely a technical endeavor; it requires deep domain knowledge and understanding user needs.

◦ Cross-Functional Teams: Close collaboration between domain experts (business units), AI engineers, and users is essential.

◦ Defined Roles: Business units must identify which context sources matter and define quality standards, while technology teams build the integration architectures and ensure data quality. This collaboration ensures that the AI system truly understands the specific business context, internal rules, customer history, and market position, moving beyond generic responses to genuine business understanding.

Challenges and Pitfalls in Context Engineering

Despite its immense promise, context engineering is not without significant challenges that developers must navigate:

• Information Overload and "Lost in the Middle": LLMs can struggle to extract valuable information from very large contexts, especially when relevant information is "buried inside the middle portion".

• Computational Constraints and Cost: Building, summarizing, and formatting context, especially in real-time, can be computationally intensive, impacting latency, cost, and scalability. Longer contexts generally lead to higher costs per token processed. Monitor token usage and latency; aim for <10% overhead from compression while maintaining >95% accuracy in task completion.

• Context Degradation Syndrome (Context Rot): When too much irrelevant information accumulates in the AI's working memory, it becomes harder for the AI to recall specific facts or follow complex instructions accurately, leading to performance degradation.

• Overfitting and Irrelevance: Including too much or the wrong kind of information can distract the model, introduce contradictions, or lead to contexts that work for one specific case but don't generalize well.

• Human-AI Collaboration Issues: Misunderstanding user intent, insufficient transparency about what information the AI is using, and biases in human context selection are common pitfalls.

• The Data Dump Fallacy: Simply connecting an AI to every available data source without intelligent curation and structuring results in "overwhelming noise" and degrades performance.

• The Silo Trap: Building separate, isolated context systems for different departments defeats the purpose of unified intelligence across the organization.

• The Static Context Mistake: Business contexts are dynamic; context engineering systems must be continuously updated, not rigid snapshots.

• Security Risks: Context-aware systems face documented vulnerabilities, including prompt injection through Model Context Protocol (MCP) channels, cross-tenant contamination risks, model poisoning, and data leakage. "Bad actors are already figuring out how to manipulate AI model behavior through poisoned web content and compromised data sources". Implementing VPC deployments, role-based access controls, and audit logging are crucial defensive measures. Security, privacy, and governance "cannot be bolted on at the end" but must be "foundational elements baked into the architecture and processes from day one".

• Initial Complexity and Upfront Investment: Setting up a context engineering framework can be complex and demanding initially. It requires "a significant upfront investment in data integration and knowledge graph creation" and "a lot of time up front creating this context".

• Bias and Ethical Curation: Curate diverse, representative data to avoid amplifying biases (e.g., ensure drill examples cover various sports and skill levels). Include ethical guidelines in curation, such as privacy for user-input drills and fairness in natural language interpretation.

Real-World Impact and Future Prospects of Context Engineering

Context Engineering is already delivering tangible business value across diverse industries by enabling AI systems to operate with deeper understanding, greater accuracy, and enhanced relevance in complex, real-world scenarios.

Real-World Impact:

• Enhanced Accuracy and Reduced Hallucinations: A primary benefit is the ability to significantly reduce hallucinations by connecting AI systems to "governed facts" and verifiable, real-world knowledge, often through Retrieval-Augmented Generation (RAG) and Knowledge Graphs. This is crucial for building user trust and ensuring the AI provides accurate and credible information.

• Access to Up-to-Date and Proprietary Knowledge: Context engineering, particularly through RAG, enables LLMs to access real-time, specific, or even proprietary information that was not part of their initial training data. This is vital for applications in dynamic fields like financial services or highly specialized sectors like healthcare.

• Tangible Business Outcomes Across Industries: Examples include financial services (40% reduction in client meeting prep time), healthcare (integrating patient records with research for accurate diagnoses), e-commerce (35% increase in click-through rates), smart cities (20% reduction in average commute times), legal research, and AI coding assistants like Claude Code, which use context engineering to generate comprehensive project plans and implement full projects end-to-end with just a few prompts.

• Cost and Efficiency: While LLM context windows are expanding (e.g., Google's Gemini 1.5 Pro allows 2 million tokens), simply "context stuffing" can lead to decreased answer quality, increased hallucination risk, and higher costs. Context engineering techniques like chunked retrieval and on-demand input significantly improve operational efficiency and inference speed by presenting only the most relevant and valuable information.

• Transparency and Traceability: RAG-based generation allows users to quickly locate the original references for LLMs, enabling verification of generated answers and making the retrieval and reasoning process observable, unlike relying solely on long context, which often remains a "black box".

• Scalability: For complex, multi-step, data-rich, or dynamic tasks, context engineering offers a viable way to build robust, enterprise-grade AI systems. It enables multi-agent systems to function effectively by providing focused context windows.

• Human-Centered AI Development: Context engineering inherently aligns with human-centered design principles by focusing on providing comprehensive, relevant, and dynamically managed information needed to understand and respond to nuanced human needs and real-world scenarios.

Future Prospects: The future of context engineering is dynamic and closely interwoven with the continuous evolution of Large Language Models and the increasing sophistication of autonomous AI agents. Key future prospects include:

• Evolving LLM Capabilities and Expanding Context Windows: As LLM context windows continue to grow (e.g., Google's Gemini 1.5 Pro allows 2 million tokens), the potential for richer and more nuanced context engineering expands. However, context engineering remains vital because simply providing massive amounts of context can still impact inference speed, lead to the "Lost in the Middle" problem, and lack traceability. Developing new RAG methods specifically for these "super-long contexts" is a future research trend.

• Advancements in RAG Paradigms: There is an ongoing shift towards "Modular RAG," which offers enhanced adaptability and versatility by integrating new specialized components like Search, Memory, Routing, and Predict modules.

• Hybrid Approaches (RAG + Fine-tuning): Combining RAG with fine-tuning is emerging as a leading strategy to optimize model performance, allowing for real-time knowledge updates from RAG and deep customization from fine-tuning.

• Multi-modal RAG: RAG is expanding its application scope into multi-modal domains, interpreting and processing diverse data forms such as images, audio, video, and code.

• Production-Ready RAG: Efforts are intensifying to enhance retrieval efficiency, improve document recall in large knowledge bases, and ensure data security, privacy, and governance for real-world deployment. The RAG ecosystem is growing, with specialized tools and frameworks like LangChain and LlamaIndex providing extensive RAG-related APIs.

• AI Anticipating User Needs: The ultimate long-term vision of context engineering is to enable AI systems to become so intelligent and attuned to human needs that they can genuinely anticipate information needs even before they are consciously articulated by users. This transforms AI from a simple query-response machine into a truly knowledgeable and powerful partner that deeply understands specific business contexts, personal needs, history, and goals.

Conclusion

Context engineering marks the maturation of enterprise AI, moving it from exciting experimental technology to a core strategic operational capability. It is the "crucial, often invisible force" that makes AI systems genuinely intelligent, useful, and relevant in the real world. By mastering context engineering, organizations can ensure their AI systems profoundly understand their unique businesses, anticipate needs, maintain institutional memory, apply company-specific logic, respect governance, and scale seamlessly, ultimately leading to faster, more confident decision-making, reduced operational costs, improved compliance, and the ability to identify market opportunities. It's about designing an AI that doesn't just respond generically but genuinely understands and operates within your specific business context, unique data, internal rules, customer history, and market position.

--------------------------------------------------------------------------------

ChatGPT said:

Request for o3 pro

Reasoned for 2m 41s

Metis Reaction X Coach‑Console — Solution Blueprint v0.2

(aligned with your “Improved Context‑Engineering Framework”)

🧭 1. Executive delta (what changed from v0.1 → v0.2)

Area v0.1 v0.2 upgrade

Context manifest Implicit list CSM‑YAML with id, modality, version, security\_tag, bias\_flag

Multi‑modal ingestion Planned Whisper only Adds frame‑sampling + clip tags for each of 102 videos; summary → deterministic layer

Versioning / observability Basic test loop Git‑style context‑schema repo + LangSmith traces + token cost dashboard

Evaluation metrics Schema validity, hit‑rate Adds Token‑Overhead %, Context‑Recall @k, Coach‑Override‑Rate, Latency (ms)

Security Input sanitiser, CBTs Adds role‑based masking of personal drills + supply‑chain scan of npm dependencies

Bias & ethics Not explicit Data‑curation rubric; coverage matrix by age, gender, sport

Tool chain Cursor API only Adds web\_search, view\_x\_video, vector\_store.search to probabilistic layer

UI Static draft panel Live “Confidence ribbon” + token cost tooltip for transparency

🧠 2. Knowledge‑asset inventory (v0.2)

id Asset Modality Version Governance

guide\_v1 Metis Comprehensive PDF (22 modes) Metis Reaction Training… text+images 1.0.0 public, read‑only

react\_manual React quick‑start PDF REACT text+images 1.0.0 internal

videos\_2025‑06 102 manufacturer demos video 0.0.1 CC‑BY; re‑tag required

coach\_logs Future drill requests & overrides text dynamic pii‑masked, role‑restricted

ui\_repo Front‑end source code semver signed commits

🔄 3. Data & memory pipeline

mermaid

Copy

Edit

graph TD

A[PDFs] -->|parse→md| B(JSON‑catalogue)

V[Videos] -->|whisper + frame‑sample| C(Clip‑summaries)

B & C --> D(Vector DB)

D -->|RAG| LLM

LLM -->|accepted config| E(Coach Logs)

E -->|nightly summariser| D

Key new stages

Frame‑sampling – every 2 s; Vision model → labels (action\_type, player\_count, drill\_goal).

Token‑aware summariser – compresses each clip summary to ≤ 256 tokens with ≥ 0.95 ROUGE‑L vs. full transcript.

Version pinning – catalogue & summaries are hashed; hash injected into deterministic prompt to guarantee traceability.

🏗️ 4. Agent & prompt design (v0.2)

4.1 System prompt (excerpt)

“Context‑Spec: 0.2.0

You are RX‑Config‑Agent. Use LIGHTS\_CATALOGUE@1.0.0, CLIP\_MEMORY@0.0.1 and the coach’s NL request.

Respect Bias‑Rubric v1 and Security‑Mask.

Output JSON → SCHEMA 0.2.0 plus rationale, confidence, token\_budget.”

4.2 Sub‑agents (isolation)

Agent Context size cap Tools Output

NL‑Parser 2 K — intent.json

Retriever 4 K vector\_store.search, web\_search, view\_x\_video top‑k snippets

Synthesiser 8 K — config.json + rationale

Validator 1 K JSON‑schema pass/fail + patch

Explainer 2 K — coach‑facing summary

(Parallelisable; keeps each window < 25 % of model limit to avoid “Lost‑in‑Middle”)

4.3 Output SCHEMA 0.2.0 (diff‑friendly)

jsonc

Copy

Edit

{

"schema\_version": "0.2.0",

"mode": "Standard | AllAtOnce | …",

"parameters": {

"num\_lamps": 8,

"colours": ["red","green"],

"targets": 2,

"delay\_s": {"min": 0.3, "max": 0.7},

"timeout\_s": 1.5,

"sensor": "close",

"flash": false,

"timer\_s": 30,

"start\_delay\_s": 0

},

"notes": "Return to base cone after each touch",

"confidence": 0.84,

"source\_hashes": ["guide\_v1@1.0.0#p1", "videos\_2025‑06#a17"],

"token\_budget": 142

}

🖥️ 5. Front‑end UX highlights

Natural‑language box with live token cost indicator (driven by token\_budget).

Confidence ribbon: green > 0.8, amber 0.6‑0.8, red < 0.6.

Advanced drawer: JSON diff editor with schema auto‑complete + version selector.

Simulate overlay uses canvas animation matching parameter timings.

Audit tab: shows source snippets & hashes for transparency (fulfils traceability principle).

📊 6. Evaluation matrix (v0.2)

Metric Target Collection

JSON‑schema validity 100 % Validator agent

Coach‑override‑rate < 20 % UI diff logs

Context‑Recall @5 ≥ 0.9 Synthetic tests

Token‑Overhead % (compression cost) < 10 % LangSmith traces

Latency p95 < 1500 ms API gateway logs

Bias‑coverage score ≥ 0.8 across rubric Nightly audit job

All metrics version‑stamped; dashboards auto‑publish to /observability.

🔒 7. Security additions

Role‑based access token masks PII in coach\_logs.

Dependency audit in CI (Snyk + SBOM).

Prompt injection suite extended to test Model Context Protocol channels.

BLE‑device white‑list from manual’s limitations table REACT.

🌐 8. Bias & ethical curation checklist

Coverage axis Minimum set

Sport types football, basketball, netball, para‑sport drills

Age groups U10, U16, adult

Gender representation ≥ 40 % female examples

Skill level novice, intermediate, elite

Continuous sampler flags imbalance; curator resolves weekly.

🚀 9. Next sprint (1 week cadence)

Day Deliverable Owner

D+1 Parse PDFs → guide\_v1@1.0.0.json AI Eng.

D+2 Video batch‑transcribe & tag (30 %) Data Ops

D+3 Implement multi‑agent chain in Cursor Prompt Eng.

D+4 Token dashboard MVP MLOps

D+5 UX confidence ribbon + tooltip FE Dev

D+7 Field beta; log overrides & latency Sports Sci.

Where to iterate next?

Fine‑tune the Bias‑Rubric, expand multi‑modal cue extraction, or deep‑dive into secure offline‑mode deployment. Let me know what slice you’d like to refine for v0.3.

You said:

Can you recap and save this entire chat to memory?

ChatGPT said:

Got it — here’s your recap for saving:

Date: 2025-08-10

Purpose:

To apply your Improved Context-Engineering Framework to the design of a Queling ReactionX / Metis Reaction Training Lights Coach-Console UI, capable of taking a coach’s natural-language drill request and generating a valid, parameter-filled configuration for the lights, ready to deploy via the backend app.

Key Actions & Outputs:

Framework Upgrade Review

Compared v0.1 and your improved framework, noting key new additions: multi-modal ingestion, explicit bias & ethics guidelines, versioning mechanics, detailed evaluation metrics, and role-based governance.

Integrated these improvements into a v0.2 solution blueprint.

Knowledge Asset Inventory

Identified PDFs (Comprehensive Guide, React manual) as deterministic context.

102 product videos for multi-modal ingestion (probabilistic context).

Future coach logs as dynamic, PII-masked deterministic data.

Data & Memory Pipeline

PDFs parsed to Markdown → JSON catalogue.

Videos processed via Whisper + frame-sampling → structured clip summaries.

All ingested into vector store with hashes for version tracking.

Agent & Prompt Design

Multi-agent chain: NL-Parser → Retriever → Synthesiser → Validator → Explainer.

Output schema v0.2.0 includes confidence, source\_hashes, token\_budget.

Front-End UI Features

Natural-language box with live token cost indicator.

Confidence ribbon (green/amber/red).

Advanced JSON editor with schema auto-complete.

Simulation overlay to preview light sequence.

Audit tab for source traceability.

Evaluation Matrix

Metrics: JSON validity 100%, Coach-override-rate < 20%, Context-Recall @5 ≥ 0.9, Token overhead < 10%, p95 latency < 1500 ms, Bias-coverage ≥ 0.8.

Security Additions

Role-based masking of personal drills.

Dependency audits and prompt-injection tests.

BLE device white-list.

Bias & Ethics Coverage

Defined rubric for sport type, age group, gender representation, skill level.

Next-Sprint Plan (7 days)

D+1 parse PDFs, D+2 transcribe/tag 30% of videos, D+3 implement multi-agent chain, D+4 token dashboard MVP, D+5 UI confidence ribbon, D+7 field beta.

Status:

v0.2 blueprint now documented; ready for iteration toward v0.3.

PDFs from earlier sessions are expired in system storage; if you want them reloaded into active memory for future reference, you’ll need to re-upload them.

Tags:

#ContextEngineering #ReactionX #MetisLights #NaturalLanguageToConfig #MultiModalRAG #BiasEthics #Security #UIBlueprint

Do you want me to go ahead and store this recap now and also prompt you to re-upload the two PDF manuals so they remain in active memory for later work?