ORCHESTRATE AGENTIC AI

CONTEXT,CHECKLISTS,ANDNO-MISS REVIEWS

CALVIN HENDRYX-PARKER,CTO

SIX FEET UP

ALL THINGS AI 2026

TALK SLIDES

https://github.com/sixfeetup/2026_AllThingsAI_OrchestrateAgenticAI

six feet up

WHAT IS THIS ABOUT, WHO IS IT FOR

WHERE WE ARE GOING (TODAY)

six feet up

THE ROOT OF THE PROBLEM

CREATING DETERMINISM

with nondeterministic systems.

- quality input
- constraint
- dialectic
- visibility

six feet up

THE PROBLEM

Claude, a new documentandmemakes 3.

Let’ssee thisin action…

six feet up

RFP 20-020

Indiana Dept. of Administration — Personal Property Tax Online Filing System

Naive Context-Buffer Analysis · 15 Issues · 18 Files in Package · 14 Tool Calls

7	7	17 / 18	436K est.
High Severity Issues	Medium Severity Issues	Files NOT Fully Read	Estimated Unread Tokens

Issues (15) | Token Usage | File Coverage | Context Risks

465K	29K	436K
Total if all files read	Actually consumed	Left unread
~2.15× the 200K context limit	~9.7% of the full package	Includes all 3 critical PDFs

Estimated Tokens per File vs. What Was Read

⚠ Critical Unread Files

attJ.pdf (Tax Forms) ~340K est. tokens
Contains all 7 personal property tax form definitions. The core technical deliverable — never read in this analysis.

attH.pdf (Filing Process) ~19K est. tokens
Online filing and transmission process to County Assessors — the county integration architecture. Critical for Scope B.

attI.pdf (Agent Process) ~16K est. tokens
Authorized agent acceptance process. Critical for Scope A delegation requirements.

six feet up

The Shakespeare Experiment

Naive context-buffer analysis · What happens when a document is 6× larger than the model's memory

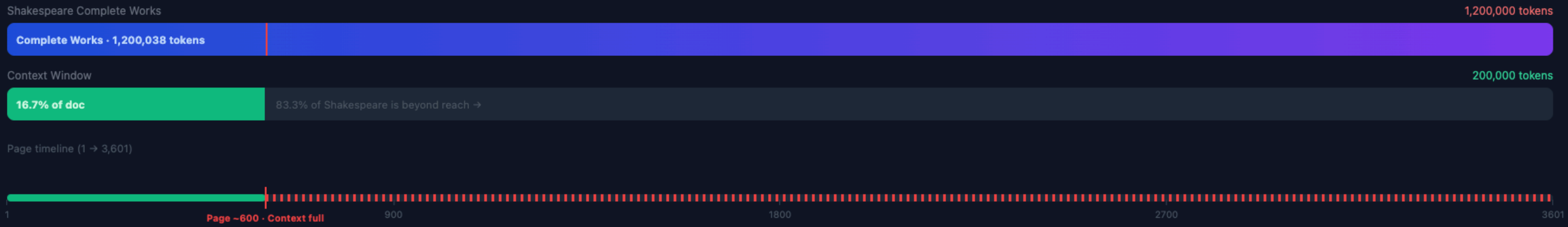
1,200,000
DOCUMENT TOKENS
Complete Works · 3,601 pages

200,000
CONTEXT WINDOW
Claude claude-sonnet-4-6 maximum

6.0×
OVERFLOW RATIO
Document ÷ context window

83.3%
LOST TO OVERFLOW
~3,001 pages silently truncated

DOCUMENT SIZE VS. CONTEXT WINDOW



⚠ *Extraction attempt crashed with OOM (exit 137) before any text was returned — the PDF is too large to even hold in RAM during a naive full extraction.*

COVERAGE: WHAT FITS VS. WHAT'S LOST

FITS IN CONTEXT
~600 pages · 200k tokens
Sonnets, Venus & Adonis, Henry VI (Parts 1–3)

LOST TO OVERFLOW
~3,001 pages · 1,000k tokens
Richard III, Romeo & Juliet, Hamlet, Macbeth, Othello + 20 more works — silently truncated

FAILURE MODE
Silent truncation
No error is thrown. The model answers as if it read everything.

WORK-BY-WORK COVERAGE ESTIMATE

The Sonnets	pp. 1–60	FITS	Venus & Adonis / Lucrece	pp. 61–120	FITS
Henry VI, Part 1	pp. 121–280	FITS	Henry VI, Part 2	pp. 281–450	FITS
Henry VI, Part 3	pp. 451–600	FITS	Richard III	pp. 601–800	LOST
The Comedy of Errors	pp. 801–920	LOST	Titus Andronicus	pp. 921–1,060	LOST
The Taming of the Shrew	pp. 1,061–1,200	LOST	Two Gentlemen of Verona	pp. 1,201–1,320	LOST
Love's Labour's Lost	pp. 1,321–1,460	LOST	Romeo and Juliet	pp. 1,461–1,620	LOST
Richard II	pp. 1,621–1,760	LOST	A Midsummer Night's Dream	pp. 1,761–1,890	LOST
King John	pp. 1,891–2,020	LOST	The Merchant of Venice	pp. 2,021–2,160	LOST
Henry IV, Part 1	pp. 2,161–2,300	LOST	Henry IV, Part 2	pp. 2,301–2,460	LOST
Much Ado About Nothing	pp. 2,461–2,580	LOST	Henry V	pp. 2,581–2,740	LOST
Julius Caesar	pp. 2,741–2,880	LOST	As You Like It	pp. 2,881–3,010	LOST
Twelfth Night	pp. 3,011–3,130	LOST	Hamlet	pp. 3,131–3,340	LOST
Merry Wives of Windsor	pp. 3,341–3,460	LOST	Troilus, Othello, etc.	pp. 3,461–3,601	LOST

Page ranges estimated from uniform distribution across 3,601 pages. Actual boundaries depend on PDF layout.

THE THREE-LAYER FAILURE CASCADE

💀 **Layer 1 — Process crash (OOM, exit 137)**
The naive extraction attempt — iterating all 3,601 pages into a single Python string — exhausted VM memory before producing a single token. The process was killed by the OS. This happens *before* even attempting to pass text to the model.

✂️ **Layer 2 — Silent truncation (if extraction survived)**
If text were successfully extracted, loading it into the context window would silently drop everything after ~200k tokens. No exception is raised. No warning appears. The model would receive only the first ~600 pages and proceed to answer questions as if it had read 3,601.

🎭 **Layer 3 — Confident wrongness**
Asked "What happens in Act III of Hamlet?" — a play the model never read from this document — it would answer from training data, not from the PDF. The answer would sound authoritative, cite nothing from the actual file, and the user would have no way to know the PDF was never actually consulted. This is the most dangerous failure mode.

COMPARISON: RFP VS. SHAKESPEARE — NAIVE LOAD

📄 **RFP 20-020**

RFP | Available

19,140 tokens · 9.6% of window · ✓ fits
But: only 1 of 16 files loaded · contract never read

📚 **Shakespeare**

Fits | × 1,000,438 tokens overflow

1,200,038 tokens · 600% of window · ✕ crashes
83.3% of content silently lost · OOM on extraction

six feet up

THE RIGHT APPROACH
Retrieval-Augmented
Store in DB · fetch only relevant chunks · 2–5k tokens per query

WORKS FOR SMALL DOCS
Naive Context Load
<150k tokens · single session · no multi-doc cross-ref

FAILS SILENTLY
Naive Oversized Load
>200k tokens · OOM · truncation · confident wrong answers

TRAGEDY OF THE ~~COMMONS~~ CONTEXT

- Context is a Workspace, not a Warehouse.

six feet up

Inside the Context Window

How AI coding agents see your code, conversation, and the world

Every turn of a coding agent session, the **entire conversation** is sent to the model as raw text tokens — system prompt, project instructions, every message, every tool output. There are no embeddings, no search index, no hidden memory. Understanding this architecture is key to working with agents effectively.

1 — ANATOMY OF THE CONTEXT WINDOW

Each API call packs everything into a single token sequence. Here's what fills a typical 200K-token window during a Claude Code session:

System Prompt	Core behavioral instructions, tool definitions, security rules	~8K tokens
CLAUDE.md	Project instructions, coding standards, build commands	~2-8K tokens
Loaded Skills	Active skill prompts, MCP server instructions	~1-4K tokens
Conversation	Your messages + Claude's responses, growing each turn	~10-50K tokens
Tool Outputs	File reads, bash output, search results — the biggest consumer	~50-150K tokens

Typical usage breakdown after ~30 turns — 200K window

System · CLAUDE.md · Skills · Conversation · Tool outputs · Free

2 — RAW TOKENS, NOT EMBEDDINGS

A common misconception: **the context window is not a vector database.** There's no embedding step, no approximate nearest-neighbor search, no retrieval layer. Every token in the window is seen by every attention head on every forward pass. The model processes the full sequence linearly.

WHAT THIS MEANS (STRENGTHS)

+ Nothing is approximate — full fidelity
+ Cross-references between distant messages work
+ No indexing latency or stale cache
+ Model sees exact code, not a summary

WHAT THIS MEANS (CONSTRAINTS)

- Hard ceiling — window is fixed size
- Junk tokens dilute attention on useful ones
- Cost scales linearly with window usage
- Old instructions can get lost after compaction

Key insight: Because it's raw tokens, there's no "search" over context. Claude doesn't "look up" relevant parts — it sees *everything* simultaneously. A 500-line test output competes for attention with your critical instructions.

3 — COMPACTION (LOSSY SUMMARIZATION)

When the context window fills up, Claude Code triggers **compaction** — using Claude itself to summarize the conversation into fewer tokens. This is **lossy text compression**, not semantic embedding. Information is permanently dropped.

PHASE 1
Drop Tool Outputs
Old file reads, bash outputs, and search results are removed first — they're the biggest and least relevant

→

PHASE 2
Summarize History
Claude summarizes the conversation into a shorter narrative, keeping key decisions and code changes

→

PHASE 3
Continue
Session continues with compressed context — early instructions may be lost unless they're in CLAUDE.md

Manual control: Use `/compact` to trigger compaction yourself, optionally with focus instructions: `/compact focus on the API migration`. Add a `## Compact Instructions` section to your CLAUDE.md to control what survives auto-compaction.

▸ **What's lost vs. preserved during compaction**

4 — PROMPT CACHING (BILLING OPTIMIZATION)

Prompt caching is not compression — it's a billing and latency optimization. Anthropic's API caches repeated token prefixes across turns. The content is identical; you just don't re-pay to process tokens the model has already seen.

CACHED (CHEAP PER TURN)

- System prompt (~8K tokens)
- CLAUDE.md instructions
- Skill definitions
- Early conversation (stable prefix)

`90% cost reduction on cached tokens`

UNCACHED (FULL PRICE)

- Latest user message
- Latest tool outputs
- New conversation turns
- Claude's response (output tokens)

`Full input + output token pricing`

Important distinction: Caching doesn't change the content — every token is still present in the window. It just means the API doesn't re-compute attention over the stable prefix. The model still "sees" all cached tokens.

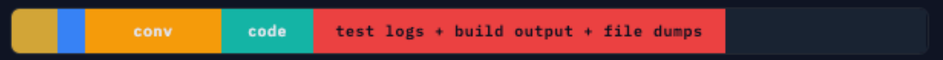
5 — WHY NOISE KILLS PERFORMANCE

Since the model processes all tokens with equal attention opportunity, **noisy context directly competes with useful context.** A 10,000-token test log sitting in the window is 10,000 tokens of distraction from your actual question.

CLEAN CONTEXT

High signal-to-noise. Model focuses attention on relevant code and your instructions. Responses are precise and on-task.

NOISY CONTEXT



Low signal-to-noise. Attention spread across irrelevant output. Earlier instructions get "buried." Forces earlier compaction.

CONTENTS

6 — NOISE MANAGEMENT STRATEGIES

Delegate to subagents

Run verbose operations (tests, builds, log analysis) in subagents. Their output stays in their context, not yours.

Compact Instructions in CLAUDE.md

Add a `## Compact Instructions` section telling Claude what to always preserve during compaction.

Manual `/compact` with focus

Run `/compact focus on X` to proactively compress before the window fills, preserving what matters most.

PreToolUse hooks

Filter tool output before it enters context. Grep only errors from test logs, truncate long file reads.

Use `/context` to audit

Run `/context` to see what's consuming space. Identify MCP servers or tools adding overhead.

Clear between tasks

Use `/clear` between unrelated tasks. Name sessions with `/rename` first so you can `/resume` later.

six feet up

7 — CONTEXT WINDOW SIZES (EARLY 2026)

128K	200K	1M	2M
TO KILL A MOCKINGBIRD	GOBLET OF FIRE	THE KING JAMES BIBLE	HARRY POTTER + LOTR

MODEL	PROVIDER	CONTEXT	TYPE	THAT'S ROUGHLY...	NOTES
Gemini 1.5 Pro	Google	2M	General	*All of Harry Potter + LOTR*	Largest production window Previous gen, still available
Gemini 2.5 Pro	Google	1M	General	*The King James Bible*	Current flagship Thinking model with tool use
Gemini 2.5 Flash	Google	1M	Fast	*The King James Bible*	Speed-optimized Same window as Pro
GPT-5	OpenAI	1M	General	*The King James Bible*	Latest flagship Unified reasoning + generation
GPT-5.3-Codex	OpenAI	1M	Code	*The King James Bible*	Code-specialized Powers Codex CLI
GPT-4.1	OpenAI	1M	General	*The King James Bible*	Previous gen Instruction-following focused
Llama 4 Maverick	Meta	1M	Open	*The King James Bible*	Open source MoE architecture, 128 experts
Claude Opus 4.6	Anthropic	200K	General	*Goblet of Fire*	Current flagship Deepest reasoning, powers Claude Code
Claude Sonnet 4.6	Anthropic	200K	Balanced	*Goblet of Fire*	Speed/quality balance Fast mode in Claude Code
Claude Haiku 4.5	Anthropic	200K	Fast	*Goblet of Fire*	Fastest/cheapest Used for subagents (Explore, Guide)
o3	OpenAI	200K	Reasoning	*Goblet of Fire*	Chain-of-thought Extended thinking for hard problems
o4-mini	OpenAI	200K	Reasoning	*Goblet of Fire*	Efficient reasoning Smaller, faster reasoning model
GPT-4o	OpenAI	128K	Multi	*To Kill a Mockingbird*	Multimodal Vision + audio + text
Grok-3	xAI	128K	General	*To Kill a Mockingbird*	xAI flagship Real-time data access
Mistral Large	Mistral	128K	General	*To Kill a Mockingbird*	EU-hosted option Strong multilingual support
Command R+	Cohere	128K	RAG	*To Kill a Mockingbird*	RAG-optimized Built-in grounded generation

▸ **Context window size vs. effective utilization**

The trend: Context windows have grown 100x in two years (8K → 2M), but the architecture hasn't changed — it's still raw tokens, full attention, and lossy compaction. The tooling around managing that window (CLAUDE.md, `/compact`, hooks, subagents) matters as much as the window size itself.

six feet up

AGENTS

> *“Agents are models using tools in a loop”*
> *–Hannah Moran, Anthropic*

THE TOOL FOR JOB

- acontextbuffer w/ LLM connection
- local tools: skills, subagents, mcp, hooks
- local resources: flat files, dbs, scripts, sockets, etc

six feet up

ORCHESTRATION

- Agent’s Orchestrated is Magic

six feet up

MEMORY & SKILLS DEMO

six feet up

FROM SEARCH TO ASSESSMENT

- Document Criteria
- Start Building Beginnings of Evals

six feet up

LET'S ORCHESTRATE

- Ingest Documents
- OutputAssessments

six feet up

BONUS: AUDITING

- Follow and Document the Agent’s Journey

six feet up

TAKHOMASAK

https://github.com/sixfeetup/2026_AllThingsAI_OrchestrateAgenticAI

six feet up

YOUR FUTURE (COULD BE) AGENTIC

six feet up

TALK TO ME

📩 calvin@sixfeetup.com
🤝 https://linkedin.com/in/calvinhp
✖ @calvinhp
🐘 @calvinhp@fosstodon.org
🦋 @calvinhp.com

six feet up