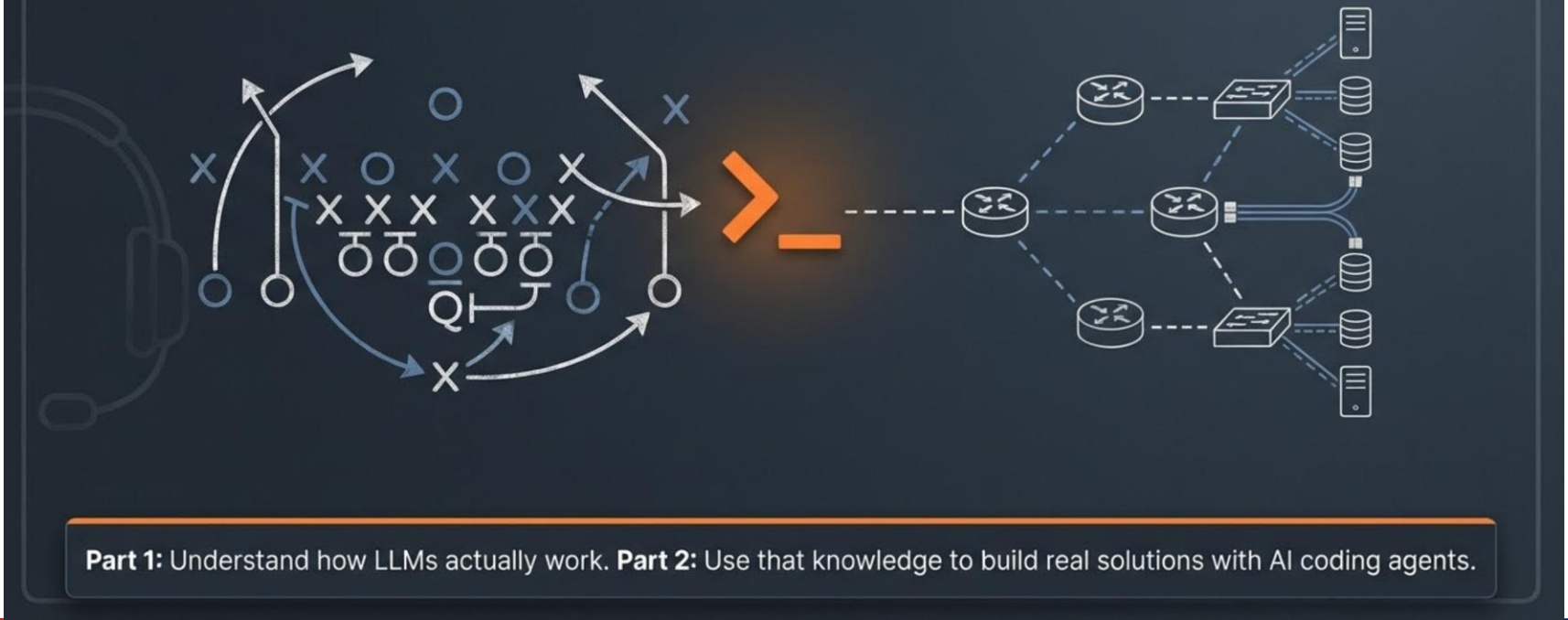
Prompting Is The New Programming

Understanding LLMs Through Football and Network Automation



Part 1: Understand how LLMs actually work. **Part 2:** Use that knowledge to build real solutions with AI coding agents.

THE PREDICTION ENGINE

To understand Large Language Models, you don't need a CS degree.
You just need to understand probability.

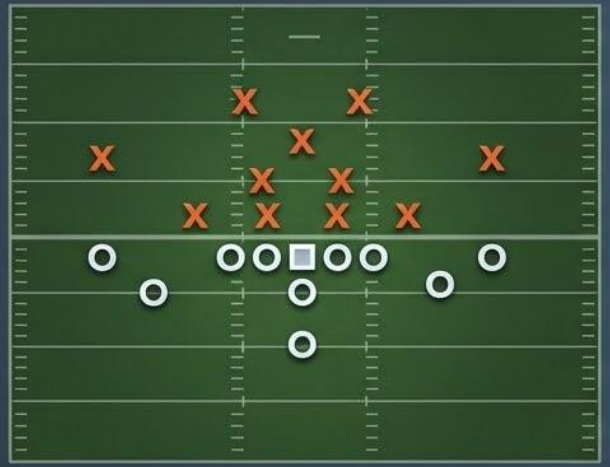
874,643

NFL offensive plays analyzed
from 1999 to 2024.

We are going to explain the architecture of modern AI using a real dataset of 874,643 NFL offensive plays. Football is inherently probabilistic. So are LLMs. If you can understand why a coordinator calls a pass on 3rd & Long, you can understand how an LLM writes code.

WHAT IS A LARGE LANGUAGE MODEL?

Situation: 1st & 10



Raw Probability: 50.4% Run / 49.6% Pass

If an offensive coordinator learned from 874,643 play calls in every situation imaginable — what would he call on 1st & 10?

WHAT IS A LARGE LANGUAGE MODEL?

Raw Probability: 50.4% Run / 49.6% Pass

Context Window —
Added Context:
Own Territory + Down by 3
+ 2 Minutes Left

New Prediction:
~90% Pass

LLM — Predicts the Next Token based on Text History.

LLMs don't "think." They predict the next most likely token based on patterns learned from training data.

CAPACITY MATRIX: SLMs vs. LLMs

The Rookie / SLM

Small Language Model (< 7B parameters)

Profile: The Rookie Coordinator

Context Considered: Down, Distance (16 total situational buckets).

Prediction Output: 50% chance of a Run.

The Veteran / LLM

Large Language Model (100B+ parameters)

Profile: The 30-Year Veteran

Context Considered: Down, distance, field position, score differential, time remaining, weather, turf type, historical matchups... and their interactions (960+ unique game situations).

Prediction Output: 68% Play Action — it's raining, cornerbacks struggle with cuts on wet grass, play action averages 7.4 YPA here.

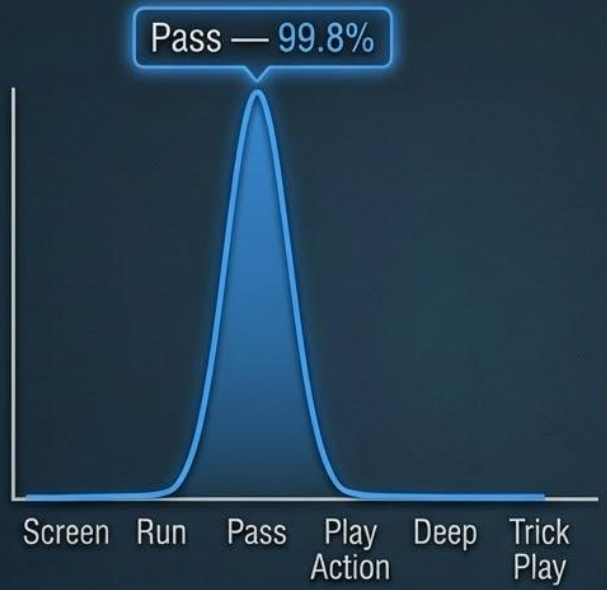
A parameter is a single learned numerical value — one tiny piece of the model's "gut instinct." Parameters aren't data. They are the model's capacity to recognize nuance and weigh complex patterns simultaneously.

TEMPERATURE: REGULATING RISK TOLERANCE

Low Temp (0.0 - 0.3):
"The Conservative Coordinator"

The model sharpens the distribution — the most likely token becomes almost certain.

Reliable. Predictable. Use for Configs.



The model scored every play. Low temperature sharpens those scores — the obvious call becomes nearly guaranteed.

TEMPERATURE: REGULATING RISK TOLERANCE

Low Temp (0.0 - 0.3):
"The Conservative Coordinator"

The model sharpens the distribution — the most likely token becomes almost certain.

Reliable. Predictable. Use for Configs.

High Temp (0.7 - 1.0):
"The Gunslinger"

The model flattens the distribution — lower-probability plays now have a real chance.

Creative. Risky. Hallucinations/interceptions happen here.
Use for Brainstorming.

Same model. Same scores. Temperature reshapes the probability distribution before the model picks. Low temp sharpens it. High temp flattens it.

WEIGHTS: THE COORDINATOR'S INSTINCTS

Rain on Grass = Play Action Success +12%

3rd & Short + Nickel Defense = Lean Run

Down by 8 in Q4 = Pass Heavy

The model does not "remember" the 874,643 plays. It learned PATTERNS — distributed across billions of weights.

No single weight stores "rain = play action." It's millions of tiny numerical values that collectively produce the tendency.

The coordinator can't explain WHY he feels play action is right — he just knows. That feeling IS the weights.

Weights are static after training. The model doesn't know about today's game or your current network outage. The only way to change weights is fine-tuning.

VECTOR DATABASES: FILING BY MEANING

Vectors are how we store documents so that AI can search by meaning instead of keywords.

1. Every document gets split into chunks (pages from the scouting binder).
2. Each chunk gets converted to a vector — a list of numbers that captures how the language is used, not just the words.
3. Similar chunks end up near each other in vector space — clustered by meaning.
4. Your question also becomes a vector. Find the nearest chunks. Those are your matches.

Query: "injured defensive line" → finds "DL missing 2 starters" and "Run defense ranked 28th" — even though the exact words don't match.

Network example: "BGP peer flapping" → finds "Route oscillation in eBGP multihop" and "Peer stability "Peer stability timers".

Vectors capture how language is used, not just what it says. Similar usage = similar vectors = nearby in the space. This happens across 4,096 dimensions simultaneously.

RAG: THE SCOUTING REPORT

The coordinator's weights are frozen. He doesn't know what's happening in TODAY's game. But the booth analyst does.

Stage 1 — The Binder: 300 pages of scouting reports. The coordinator can't read all of it before the play clock runs out.

Stage 2 — The Booth Analyst: Someone upstairs searches the vector database — finds the 2-3 pages relevant to THIS situation.

Stage 3 — The Handoff: Only those pages get sent down to the sideline. The coordinator never sees the other 297 pages.

Stage 4 — The Call: The coordinator reads the relevant pages, factors them into his prediction, and makes a better-informed play call. His brain didn't change — he just had better context.

RAG doesn't change the model. The booth analyst searches the vector database, finds the relevant pages, and adds them to the prompt. The model just sees a longer input with better context.

FINE-TUNING: THE TRAINING CAMP

Changing *how* the model thinks, not just what it knows.

BASE MODEL

"I know what NFL teams typically do."

FINE-TUNED MODEL

"I know what NFL teams do, AND I know *your* playbook, *your* terminology, and *your* risk tolerance."

RAG (Scouting Report)	Temporary context.	Best for real-time data.
Fine-Tuning (Training Camp)	Permanent brain change.	Best for company voice & coding standards.

THE PLAYBOOK — SUMMARY

Concept	The Football Version
LLM	A prediction engine — situation in, play call out
Parameters	The coordinator's capacity for nuance — more = deeper reads
Temperature	Risk tolerance — conservative or gunslinger. Reshapes the distribution.
Weights	Gut instincts — learned patterns, frozen after training
Vector Database	The scouting binder, filed by meaning instead of keywords
RAG	The booth analyst hands down the 2-3 relevant pages
Fine-Tuning	Training camp — permanently changes how the coordinator thinks

You are the **Head Coach**. The AI is the **Coordinator**. You choose the play, you set the risk tolerance, and you provide the scouting report. They run the numbers.

IF IT'S JUST PREDICTING, WHY DOES IT THINK?

Some coordinators were trained to stop and talk through their reads before calling the play.

STANDARD MODEL

"Run left."

One prediction. Immediate answer.
Like a gut call.

40 SECONDS

X|O X|O

THINKING MODEL

<think> "They're in nickel..." → "Safety cheating right..." → "DL is gassed..." → "Run left behind the guard..." </think>

"Run left."

01 SECONDS

Same call — but every thinking step added context for the next prediction, making it more informed.

Thinking models were trained to generate intermediate tokens — **<think>...</think>** — before answering. Each step becomes context that shapes the next prediction. More thinking = more context = more accurate. But there's a play clock.


Prompting Is The New Programming
Part 2
Effective Prompting → VibeOps → Spec-Driven DevOps

WHY LLMs ARE GOOD AT CODE

Strict Syntax

Python, YAML, JSON, CLI commands have rigid rules. Fewer valid next tokens = higher confidence predictions.

Repetitive Patterns

Function definitions, import statements, error handling — these patterns repeat millions of times in training data.

Structured Training Data

GitHub repos, official docs, Stack Overflow — the training corpus for code is more internally consistent than conversation.

Testable Output

Code either runs or it doesn't. The training signal is cleaner — the model gets clear feedback on what's correct.

HOW DO YOU ACTUALLY USE THIS?

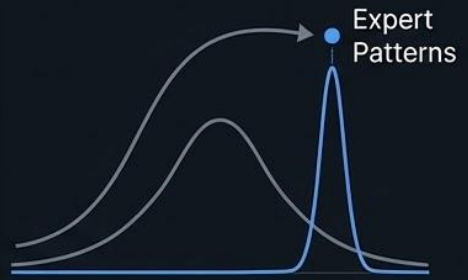
Number	Method	Note
1	~~Copy-paste from a web chat~~	(this is so 2024...)
2	Claude Code desktop app	(I personally haven't used this)
3	Claude in your IDE	(This is how I use Claude for VibeOps — saved .md prompts for quick scripts)
4	`CLI: claude --dangerously-skip-permissions`	(This is how I do Spec-Driven DevOps)

`--dangerously-skip-permissions` is optional. It allows chaining multiple agents to avoid context window bloat.

THE ART OF PROMPTING: PERSONAS

Tell the AI who it is — and it becomes that version of itself.

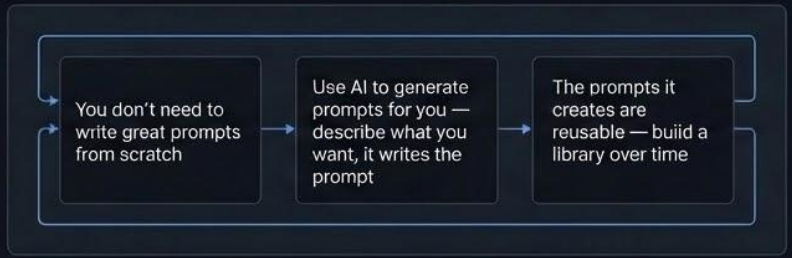
```
You are an expert Cisco network automation engineer.
You prioritize idempotent scripts, robust error handling,
and PEP-8 compliant Python.
```



Setting a persona steers the model's probability distribution. It biases predictions toward expert-level patterns for that domain — like tuning which playbook the coordinator reaches for first.

THE PROMPT THAT WRITES PROMPTS

Your best tool when getting started.



The One Prompt — A single reusable prompt that generates specialized prompts for any task. You describe what you want to build. It asks clarifying questions, then outputs a structured prompt with persona, constraints, output format, and edge cases you wouldn't think to include. One prompt to bootstrap everything else.

DEMO: 30 MINUTES IN NOVEMBER → 5 MINUTES TODAY

Using the prompt generator to recreate a network automation script

```
From netmiko hello-world to concurrent multi-device automation — same
script, new workflow.
```

BEYOND VibeOps

VibeOps is powerful. But for complex systems, context is the real challenge.

Challenge	Detail
Token Cost	Larger context windows consume more tokens — and more money.
Compaction	As conversations grow, the model compresses earlier context. Details get lost.
Lost in the Middle	Research shows LLMs struggle with information buried in the middle of long contexts.The beginning and end get attention; the middle gets forgotten.

Breaking the process into subagents keeps each context window small, focused, and efficient.

Spec-Driven DevOps: The Process

Phase	Role	What Happens
1. Vision	Vision Assistant	Explore the idea, ask questions, shape the spec
2. Architecture	Lead Architect	Design the system, define components and interfaces
3. Planning	Project Planner	Break the build into ordered stages with dependencies
4. Building	Stage Managers	Execute each stage — code, test, iterate
5. Testing & Security	Testers + Security Auditor	Verify functionality, audit for vulnerabilities

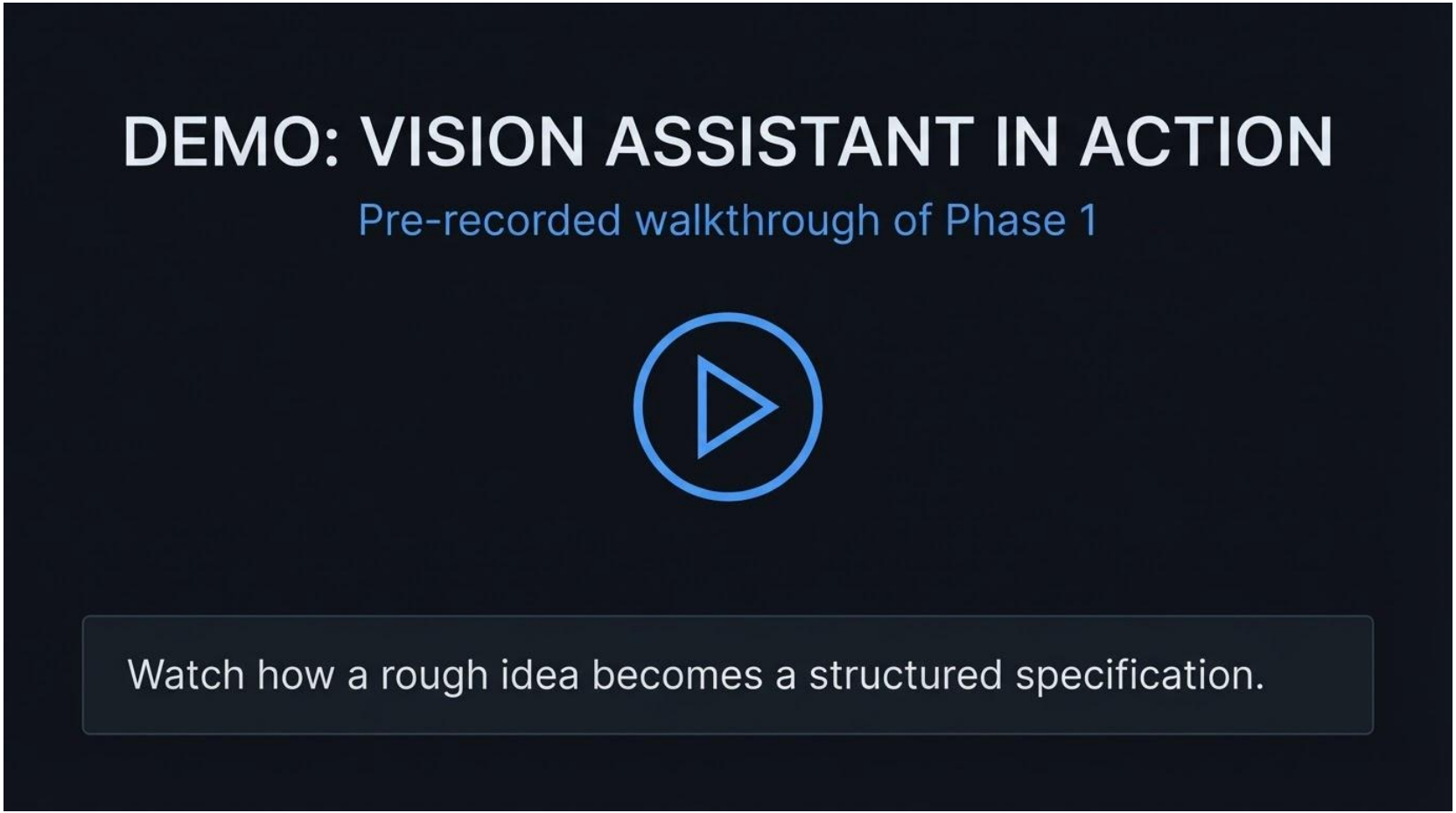
Human-in-the-loop: Active during Vision (you shape the spec) and Testing (you verify the result). Other phases run autonomously unless you choose to intervene.

THE VISION ASSISTANT

Phase 1: From idea to specification

Step	What Happens
Listen	You describe your idea — rough, incomplete, that's fine
Reflect	The AI mirrors back what it heard — "Here's what I think you're building"
Explore	The AI asks probing questions — edge cases, scale, constraints
Shape	Together you refine the idea into a clear vision
Document	The AI generates a structured specification document

Steering: explore an idea ("Where do you think I'm going with this?" / "Thinking about this leads to…"). Use trailing "…" to help the AI generate more thinking.	Driving: execute once the ideas are solidified.


DEMO: VISION ASSISTANT IN ACTION
Pre-recorded walkthrough of Phase 1
Watch how a rough idea becomes a structured specification.

THE BUILD PIPELINE

Role	Responsibility	Analogy
Lead Architect	Designs the system — components, interfaces, data flow, tech stack	HOW it's built
Project Planner	Breaks the build into ordered stages with dependencies and acceptance criteria	WHAT gets built when
Stage Managers	Execute each stage — write code, run tests, iterate until acceptance criteria pass	BUILD it

Flow: Architect → Planner → Stage Manager 1 → Stage Manager 2 → … → Stage Manager N

TRUST BUT VERIFY

Testing & Security

Layer	What	Detail
Automated Tests	Stage Managers write their own tests	Unit tests, integration tests — built into each stage
Human Testing	You drive the application	Click every button, test every edge case. This is critical — AI can miss UX issues. Test it thoroughly.
Cross-Model Security Audit	The built-in security subagent already treats code as third-party	Use a different model (Codex, Gemini, a local model) for an independent review. Defense in depth — the same principle you apply to your network security.

AI IN YOUR APPLICATIONS

MCP, A2A & Skills

MCP (Model Context Protocol)

Connects AI agents to external tools and services — databases, APIs, docs. A 1:1 client-server relationship.

The agent's hands — reaching out to touch the real world.

A2A (Agent-to-Agent)

Connects AI agents to other AI agents. One agent delegates tasks to specialized agents, which return results. Peer-to-peer collaboration.

The nervous system — agents coordinating with each other.

Skills

Pre-packaged instructions — curated prompts + tool patterns that tell an agent how to accomplish a specific task well.

Not a protocol — institutional knowledge handed to an agent before it acts.

The agent's training — what it knows how to do.

Hierarchy: Skills inform Agents → who communicate via A2A → who use tools via MCP

MCP is the hands that touch the real world. A2A is the nervous system between agents. Skills are the institutional knowledge. Together, they're the unified AI system.

DEMO: CML WEBEX CHATBOT

Draw a network. Watch it come to life.

What you'll see:

- A network topology is drawn on the whiteboard
- Photo sent to a Webex bot
- AI reads the topology from the image
- Bot provisions the network in Cisco CML — live, running devices

HOW IT WORKS: CML WEBEX CHATBOT

Architecture Flow:

1. **Webex Webhook** – Bot receives the photo message
2. **Vision Model** – AI analyzes the whiteboard image, extracts topology (devices, links, IPs)
3. **Topology Generator** – Converts extracted data into CML-compatible topology definition
4. **CML API** – Provisions the lab: creates devices, configures interfaces, starts nodes
5. **Config Push** – SSHes into each device and pushes the running config
6. **Credential Return** – Bot sends SSH credentials back to the Webex room

DEMO: PROJECT HERBIE

The Blind Network Engineer

Hand it SSH credentials and a single IP. No documentation. No diagrams. No prior knowledge. Watch what happens.

Phase	What Herbie Does
Discover	Crawls the network via CDP/LLDP, SSHes into every device, builds the full topology
Confirm	AI analyzes the topology, runs reachability tests, operator confirms intended state
Plan	AI generates a per-device monitoring plan (SNMP traps, syslog, SSH polling)
Instrument	Pushes monitoring config to every device, starts all receivers and pollers
Defend	Detects faults in real-time, diagnoses root cause with AI, pushes fixes via SSH

RESOURCES & Q&A

You already think like an engineer. AI doesn't replace that — it removes the syntax barrier.

The One Prompt
github.com/seanorama/the-one-prompt

Spec-Driven DevOps
npmjs.com/package/spec-driven-devops

Network Python Toolbox
github.com/seanorama/networkpythontoolbox

'VibeOps Forum - Slack'
slack.com/invite/VibeOps_Forum

CML Webex Bot
github.com/seanorama/CML-Webex-Bot