# The Agentic Workforce: How One Person Builds a 50-Person Output

**A Master Education Document by BioCreative Strategies**

> *How we designed, built, and operate a fully autonomous AI workforce — 8 specialized agents running 24/7 for ~$125/month — and how others can replicate the pattern.*

**Version:** 1.0 · **Created:** May 18, 2026 · **Author:** Brian Elbert, BioCreative Strategies

## Chapter 1: The Problem — Why AI Agents, Not AI Tools

Every founder of a services company hits the same wall. The work is good. The pipeline is growing. But you can't hire fast enough, can't afford a full team, and can't maintain quality across 15 simultaneous workstreams.

The typical agency response: hire generalists, sacrifice depth. Or hire specialists, sacrifice breadth. Or over-rely on freelancers, sacrifice consistency.

**The BioCreative response:** build a workforce that doesn't sleep, doesn't quit, doesn't forget what you taught it yesterday.

Not a chatbot. Not a copilot that helps you type faster. A *workforce* — specialized agents with defined roles, persistent memory, structured knowledge, and human oversight at the boundaries that matter.

The result: one person with a strategic brain and a 24/7 AI workforce producing the output of a 50-person team — market intelligence reports, outreach campaigns, content pipelines, client onboarding, financial operations, legal review — at a total system cost of ~$125/month.

This document explains exactly how that works.

## Chapter 2: The Mental Model — LLM as a New Kind of Computer

The breakthrough insight — borrowed from Andrej Karpathy's "Software 3.0" framework — is that **a large language model is not a chatbot. It's a CPU.**

Once you see it that way, the architecture writes itself:

| Computer Concept | Agent Equivalent | What We Built |
|---|---|---|
| CPU | The model (Claude Sonnet, Opus, Haiku) | Anthropic API — frontier reasoning |
| RAM | Context window (200K–1M tokens) | Working memory per agent turn |
| L1/L2 Cache | Prompt cache (90% discount on hits) | Persistent context across turns |
| Disk | Vector store + databases + Git | 83K embedded chunks + Neo4j graph + operational DBs |
| Syscalls | Tool calls | MCP tools, Edge Functions, webhooks |
| Device Drivers | MCP servers | Gmail, Calendar, Supabase, Firecrawl, SSH |
| OS / Shell | The harness (runtime wrapper) | Claude Agent SDK, Archon YAML, n8n workflows |
| Processes | Sub-agents | Specialist workers that fork, execute, and return |
| Installed Programs | Skills (structured knowledge) | 155 skill docs across 5 departments |
| Cron / Scheduler | n8n + pg_cron | 25 active workflows, 30+ scheduled |

| | | |
|---|---|---|
| | | jobs |
| Network Bus | HTTP + REST + MCP | Agent-to-agent communication |
| Background Daemon | Memory Dreaming (weekly consolidation) | Agents that learn while the system "sleeps" |

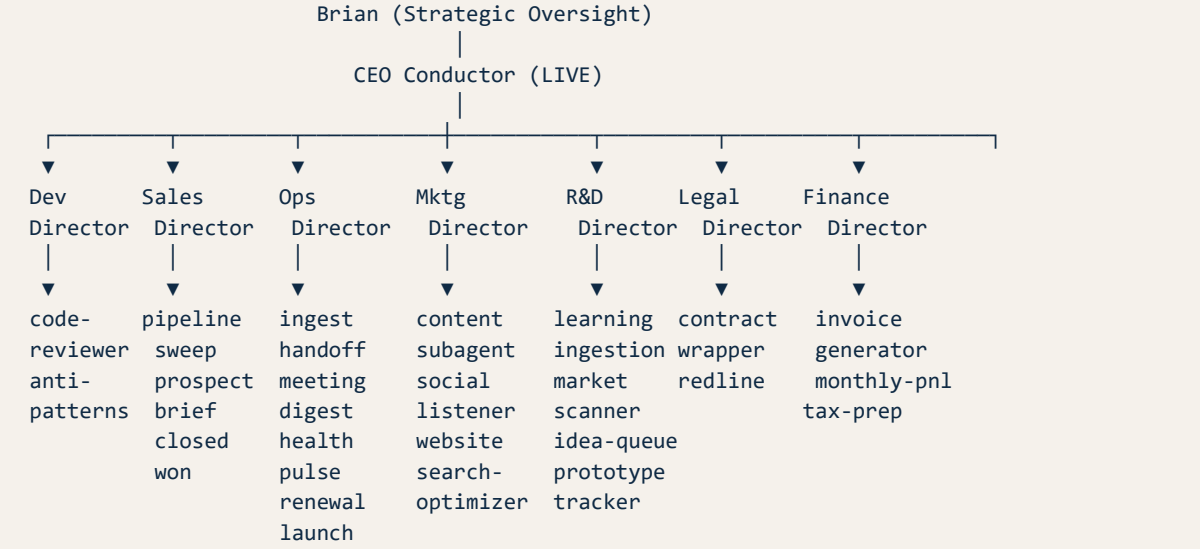
**The shift in mindset:** Stop asking "what prompt should I write?" Start asking "what's the right operating system for this kind of work?"

When you ask "should this agent be a long-running service or a one-shot process?" — you're making the same decision a systems engineer makes between a daemon and a script. When you ask "where does memory live?" — you're designing a disk hierarchy. When you ask "should agents consolidate what they've learned?" — you're designing a defragmentation cycle.

We're not building chatbots. We're booting a new kind of computer and writing its operating system.

## Chapter 3: The Org Chart — A Corporate Hierarchy of AI Agents

Every company has an org chart. Ours happens to be made of AI agents.




**Why organizational structure matters:** Agents work better when they're *specialized*. A Sales Director that only thinks about pipeline doesn't hallucinate marketing strategy. A Legal Director that only sees contract data can't accidentally expose financial information.

### The Wall of Separation

Legal and Finance are **walled off** from all other agents:

- **Finance** holds credentials to the Core DB *only*. Cannot access operational data. Cannot be called by other directors directly.
- **Legal** holds Hub DB access for legal reviews only. Excluded from general queries.
- **Cross-wall communication** happens only through the Conductor — brokered, logged, auditable.

Each container validates its own wall on startup and reports `wall_ok: true` in its health check. If a wrong credential is detected, the container refuses to start.

## Chapter 4: How We Build Agents — The End-to-End Pattern

Building a new agent follows a repeatable 5-step pattern:

### Step 1: Design (in the IDE)

Every agent starts as a conversation. Brian describes what the agent should do. The system produces:

- A **skill doc** — identity, scope, tools, system prompt, decision rules
- A **runtime doc** — topology, lifecycle, cost envelope
- **Sub-agent specs** if the director will delegate work

### Step 2: Implement (in the monorepo)

Every director follows the same folder structure:

```
<director>/
├── server.py              (FastAPI shell — uniform across all)
├── <director>.py          (Core orchestration logic)
├── scopes/                (One handler per scope of work)
│   ├── pipeline_sweep.py
│   └── status_report.py
├── lib/                   (Shared utilities)
│   ├── cost.py            (Budget enforcement)
│   ├── memory.py          (Write to memory_events)
│   ├── heartbeat.py       (Emit to heart-api)
│   └── hub_client.py      (PostgREST helper)
├── Dockerfile
├── docker-compose.yml
└── requirements.txt
```

### Step 3: Deploy (via SSH)

```
git pull → docker compose up -d --build → curl /health
```

Traefik auto-discovers the container via Docker labels. HTTPS endpoint is live in seconds.

### Step 4: Wire to Conductor

Register the new director's endpoint so the CEO Conductor can route tasks to it.

### Step 5: Test

Fire a real task via the uniform contract and verify the full lifecycle: start → execute → heartbeat → memory write → result.

**Total time from concept to live agent:** 2–4 hours for a new director. The pattern is so repeatable that we deployed 7 directors in 4 days.

## Chapter 5: Harness Architectures — Five Ways to Run Intelligence

Not every piece of work needs a full autonomous agent. We use five harness types, each fit for a different shape of work:

### 1. Claude Agent SDK Containers (Directors)

Long-running Docker daemons with a REST contract. The workhorse of the system. Each director implements `POST /run`, `GET /status`, `GET /health`. Runs 24/7, holds state, emits heartbeats.

**Best for:** Domain ownership with continuous responsibilities.

### 2. Pydantic AI Agents (Specialists)

Python-native agents inside the Brain API container. Market Intel, Client Prep, Content, Deep Research. Called by directors or directly. Rich tool surfaces (11–15 tools each).

**Best for:** Reusable specialist capabilities that multiple callers need.

### 3. Archon YAML Harnesses (DAG Workflows)

Explicit node-and-edge workflows. Each node is a step; edges are data flow. The model fills in the judgment at each node, but the *path* is predetermined.

**Best for:** Multi-stage pipelines where the path is known but each step needs LLM reasoning. Our market intelligence pipeline is an 18-node DAG.

### 4. MCP Servers (Tool Surfaces)

Universal tool bus. The Model Context Protocol connects any agent to external services — Gmail, Google Calendar, Supabase, web scraping, SSH. Each server is a "device driver."

**Best for:** Giving agents access to the outside world through a standardized interface.

### 5. Edge Functions + n8n (Lightweight Automation)

Serverless compute for sync HTTP tools (Edge Functions) and no-code orchestration for scheduled workflows (n8n). 45+ Edge Functions, 25 active n8n workflows.

**Best for:** Scheduled tasks, webhook receivers, and lightweight operations that don't need a full agent loop.

### The Composition Pattern

These aren't alternatives — they *compose*:

```
CEO Conductor (SDK container)
  → dispatches to Sales Director (SDK container)
    → which calls Market Intel Agent (Pydantic AI)
      → which uses Firecrawl + PubMed tools (MCP)
        → results stored via Edge Function (Supabase)
          → notification sent via n8n workflow
```

## Chapter 6: The Knowledge Architecture — The Agent's Disk Hierarchy

Agents are only as good as what they know. Our knowledge architecture gives every agent access to exactly the information it needs — no more, no less.

### Four Knowledge Stores

| Store | What It Holds | Size |
|---|---|---|
| Brain DB (Vector + FTS) | Unified knowledge base — articles, skills, docs, embedded chunks | 83,000+ chunks |
| Neo4j Graph | Entity-relationship reasoning — companies, people, trials, grants | 52K nodes, 31K relationships |
| Hub DB | Operational state — accounts, pipeline, campaigns, content | 17K+ accounts, 265K+ contacts |
| Core DB | Financial data — walled off, Finance Director only | Sequestered |

### The RAG Pipeline

Every query flows through:

```
Query → Brain Router (Haiku classifier, 5 routes)
  → Hybrid Search (FTS + vector + Reciprocal Rank Fusion)
    → Top-K chunks returned with relevance scores
      → Injected into agent's context window
```

### Multi-Model Routing

A LiteLLM proxy routes model calls to the best-fit model:

- **Claude Sonnet 4.6** — default reasoning (fast, capable)
- **Claude Opus 4.7** — hard problems, council judge
- **Claude Haiku** — classification, tagging, extraction
- **Qwen 2.5** (local Ollama) — zero-cost classification tasks

## Chapter 7: The Skills System — Context Engineering at Scale

This is the "secret sauce." Skills are the difference between an agent that hallucinates and one that executes with precision.

### What a Skill Is

A skill is a structured markdown document (<6,000 characters) that gives an agent:

- **Identity** — who it is in this context
- **Scope** — what it's allowed to do (and NOT do)
- **Tools** — what capabilities it has access to
- **Rules** — hard constraints and decision logic
- **Examples** — patterns to follow

### Skills ≠ Knowledge

This distinction is critical:

- **Knowledge** = "facts about the world" (stored in Brain DB, retrieved on demand)
- **Skills** = "how to do things" (loaded into agent context at task start)

An agent might *know* that CARR Biosystems is a CRO in the cell therapy space (knowledge). But it needs a *skill* to know how to write an outreach message that matches CARR's messaging matrix, uses the correct persona voice, and follows the 5-part email structure.

### 155 Skills Across 5 Departments

| Department | Skills | Focus |
|---|---|---|
| Internal Ops | 50 | Infrastructure, data pipelines, deployment, monitoring |
| Pre-Client | 29 | Prospect research, ICP scoring, outreach, targeting |
| R&D | 35 | Prototypes, media generation, tool evaluation, intelligence |
| Agent Brain | 28 | RAG, memory, agents, VPS services, Conductor |
| Client Mgmt | 13 | Campaign execution, Miller Heiman, onboarding |

### Context Engineering: The Loading Pattern

```
Root CLAUDE.md (entry point, ~280 lines)
    ↓
Department CLAUDE.md (department context)
    ↓
Skill doc (quick-load, <6K chars)
    ↓
Context doc (deep reference, unlimited)
```

Each agent gets **only the skills relevant to its task**. A Sales Director loading for a "pipeline sweep" gets the pipeline management skill, the ICP scoring skill, and the Miller Heiman methodology — not the 150 other skills that would pollute its context.

This is the Karpathy frame applied directly: we're managing the agent's *page table* — what's loaded into working memory at any given moment.

## Chapter 8: Memory & Persistence — How Agents Remember and Learn

### Six Memory Stores

| Store | Scope | Purpose |
|---|---|---|
| Brain DB chunks | Global RAG | Unified knowledge base — the "hard drive" |
| Neo4j graph | Global relationships | Entity connections — "who knows whom" |
| Qdrant vectors | Semantic similarity | Cognee-internal for graph-RAG |
| agent_memory | Typed entries | Facts, decisions, preferences, episodes, insights |
| Mem0 | User/session | Semantic user memory for conversational context |
| memory_events | Append-only log | Every agent action leaves a trace — the audit trail |

### Memory Dreaming — The Weekly Consolidation Cycle

This is what turns a database into a *learning system*.

Every Sunday at 2am, the Memory Dreaming workflow:

1. Scans all `memory_events` from the past week
2. An LLM summarizes patterns across episodes
3. Promotes salient observations into durable graph relationships
4. Flags contradictions for human resolution
5. Writes consolidation events back to the log

**What this means in practice:** If the system notices that "CARR meetings tend to focus on scalability concerns" across 5 separate call transcripts, that pattern gets promoted from buried log entries into a retrievable relationship. Next time the Sales Director prepares for a CARR meeting, that insight surfaces automatically.

Agents don't just remember individual runs — the system synthesizes *cross-run patterns* into durable knowledge. It learns.

## Chapter 9: Inter-Agent Communication — How the Workforce Collaborates

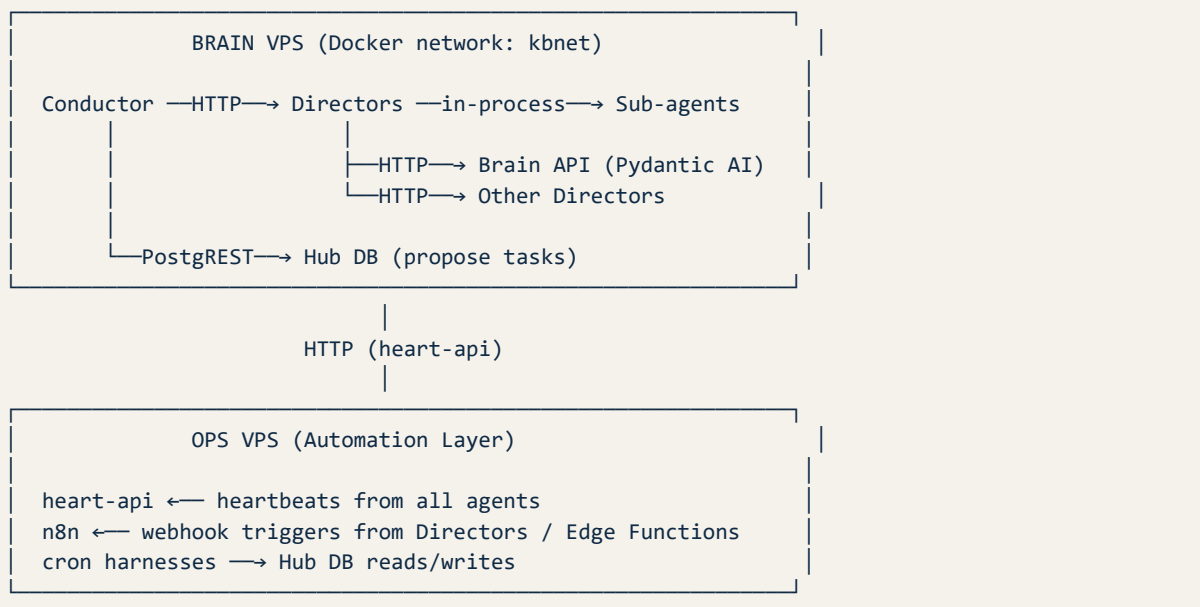
### The Uniform Director Contract

Every director implements the same interface:

```
POST /run     {"scope": "...", "inputs": {...}, "task_id": "...", "requested_by": "..."}
GET  /status/{task_id}   → {"status": "running|done|failed", "result": {...}}
GET  /health             → {"ok": true, "version": "...", "runs_in_flight": 0}
```

This uniformity means the Conductor doesn't need to know *how* a director works — only *what* it can do.

### Communication Patterns



### The Lifecycle of Every Agent Run

6. **Start** → writes `memory_events` row (run_started)
7. **Execute** → scope handler does the work
8. **Heartbeat** → emits to heart-api for observability
9. **Complete** → writes `memory_events` row (run_succeeded) with cost + latency
10. **Fail** → writes `memory_events` row (run_failed) with error details

Every action is traceable. Every cost is tracked. Nothing runs in the dark.

## Chapter 10: Human-in-the-Loop Governance — Not an Afterthought

The most common failure mode of AI agent systems: they auto-execute before a human can catch errors. Our system is designed with the opposite principle: **propose freely, confirm once.**

### The Cortex Confirm Gate

Every agent — every sub-agent — can only *propose* actions. Nothing auto-executes at the outbound boundary (anything that touches clients, sends emails, publishes content, or commits code).

**Task lifecycle:**

```
Signal (system detects something)
  → Propose (agent writes to Cortex Inbox)
    → Confirm (Brian reviews and approves)
      → Dispatch (agent receives confirmation)
        → Execute (agent does the work)
          → Report (result logged, task closed)
```

### What Runs Without Human Gates

Everything *internal*:

- Brain consolidation and memory writes
- Director-to-Director coordination
- Sub-agent spawns within a director's scope
- Health checks, heartbeats, sync operations

### What Requires Human Confirmation

Everything *outbound*:

- Sending emails to prospects or clients
- Publishing content to any channel
- Creating or modifying data that affects billing
- Deploying code changes
- Any action that touches the outside world

### The Design Principle

> *"AI agents over-capture. The plan must stay high signal-to-noise."*

One friction step — asking first — is cheap insurance. It maintains trust in the system while allowing agents to operate autonomously within safe boundaries.

## Chapter 11: Results & Economics — The Proof

### Cost Structure

| Component | Monthly Cost |
|---|---|
| CEO Conductor | ~$5–10 |
| Dev Director (worker = $0 via subscription) | ~$11–22 |
| Sales Director | ~$15–25 |
| Operations Director | ~$10–20 |
| Marketing Director | ~$15–25 |
| R&D Director | ~$10–15 |
| Legal Director | ~$5–15 |
| Finance Director | ~$5–10 |
| Always-on learning crons | ~$12 |
| n8n workflow automation | ~$10–15 |
| Total | ~$115–125/month |

Hard cap: $200/month. The Dev Director's heavy compute is $0 marginal cost via a flat-rate subscription.

### Time Economics

| Task | Before (Human) | After (Agent) | Savings |
|---|---|---|---|
| Market intelligence report | 4–8 hours | 45–90 minutes (autonomous) | 80–90% |
| Outreach message generation (50 messages) | 3–4 hours | 15 minutes | 93% |
| Meeting prep briefing | 45 minutes | 5 minutes | 89% |
| Content draft (blog/social) | 2 hours | 10 minutes + review | 90% |
| Client onboarding setup | 6 hours | 30 minutes | 92% |

### Scale Properties

- **24/7 operation** — agents don't sleep, don't take PTO, don't burn out
- **Consistent quality** — same methodology, same frameworks, every time
- **Parallel execution** — multiple directors working simultaneously
- **Compound learning** — the system gets smarter every week via Memory Dreaming
- **Zero marginal cost for additional capacity** — adding a new director costs deployment time, not headcount

### The Positioning

One person with:

- 155 structured skills loaded on demand
- 83,000+ knowledge chunks retrievable in milliseconds
- 8 autonomous agents running continuously
- 25 automated workflows handling routine operations
- Human judgment applied only where it matters most

Producing the output of a 50-person team. For $125/month.

## Chapter 12: How Others Can Build This — Replicable Patterns

### Start Small: The Minimum Viable Agentic Workforce

You need exactly three components:

11. **A Brain** — embeddings + skills + persistent memory (Supabase + pgvector is enough to start)
12. **An Orchestrator** — something that routes tasks and manages state (n8n or a simple Claude Code loop)
13. **One Specialist Agent** — the Market Intel pattern is the simplest standalone agent to build first

### The Build Sequence

14. **Week 1:** Set up your knowledge layer (embed your docs, build 5-10 skills)
15. **Week 2:** Build one agent that can retrieve knowledge and execute one workflow
16. **Week 3:** Add a scheduler (n8n cron) so the agent runs without you triggering it
17. **Week 4:** Add memory persistence so the agent learns from its runs
18. **Month 2:** Add a second specialist. Wire them through an orchestrator.
19. **Month 3:** Formalize the director pattern. Add human governance.

### Key Decisions to Make Early

| Decision | Our Choice | Why |
|---|---|---|
| Model provider | Anthropic (Claude) | Best tool use, best instruction following, MCP standard |
| Database | Supabase (Postgres + pgvector) | Free tier is generous, RLS for security, real SQL |
| Hosting | VPS (Docker + Traefik) | $10–20/month, full control, no vendor lock-in |
| Automation | n8n (self-hosted) | Open source, visual workflows, webhook support |
| Memory | Postgres + Neo4j | Hybrid vector + graph for relationship reasoning |
| Embedding | Gemini (768-dim) | $0.00 cost for embedding, good quality |

### Common Pitfalls

20. **Over-engineering before proving value** — start with one agent that saves you 2 hours/week
21. **Insufficient context** — agents without skills hallucinate; write the skill docs first
22. **No human gate** — AI that auto-executes will eventually embarrass you publicly
23. **Trying to share context across agents** — context isolation is a *feature*, not a bug
24. **Ignoring memory** — an agent that forgets what it learned yesterday will never compound

### The Tools We Use

- **Anthropic Claude** — reasoning (Sonnet for daily work, Opus for hard problems, Haiku for cheap tasks)
- **Supabase** — database + vector search + Edge Functions + auth
- **Docker + Traefik** — containerized deployment with auto-TLS
- **n8n** — workflow automation and scheduling
- **Neo4j** — knowledge graph for entity relationships
- **LiteLLM** — multi-model routing proxy
- **MCP (Model Context Protocol)** — universal tool interface standard
- **Git** — version control for all agent code, skills, and knowledge

## Appendix A: Glossary

| Term | Definition |
|---|---|
| Agent | An LLM that decides what to do next (vs. a workflow where the path is predetermined) |
| Harness | The runtime wrapper that gives an agent tools, memory, and a control loop |
| Skill | A structured markdown document that teaches an agent how to do one thing well |
| Context engineering | Managing what information is loaded into an agent's working memory |
| Sub-agent | A specialist agent spawned by a parent for one task, then destroyed (context isolation) |
| Director | A long-running agent that owns a domain (Sales, Ops, Marketing, etc.) |
| Conductor | The CEO agent that reads system state and proposes task dispatches |
| Memory Dreaming | Weekly LLM consolidation of episodic memories into durable knowledge |
| MCP | Model Context Protocol — a standard for connecting AI to external tools |
| RAG | Retrieval-Augmented Generation — fetching relevant knowledge before generating |
| Brain | The knowledge + memory layer (not a single agent — a substrate all agents use) |
| Heart | The automation + observability layer (n8n, heartbeats, workflow triggers) |
| Cortex | The human governance layer (inbox, confirm gate, task lifecycle) |

## Appendix B: Architecture at a Glance

```
Infrastructure:
  Brain VPS (4 vCPU, 16 GB) → 15 containers (8 agents + services)
  Ops VPS (2 CPU, 8 GB) → n8n, heart-api, learning pipelines, landing pages
  KB VPS (4 vCPU, 16 GB) → Claude Code worker, Ollama (local LLM)

Agent Fleet:
  CEO Conductor (1) + Directors (7) + Pydantic AI Specialists (4) = 12 agents
  + 25 n8n workflows + 45 Edge Functions + 30 pg_cron jobs

Knowledge:
  83K+ embedded chunks | 155 skills | 52K graph nodes | 17K accounts | 265K contacts

Cost: ~$125/month total system | $200/month hard cap
```

## Appendix C: Content Repurposing Map

| Format | Source Chapters | Notes |
|---|---|---|
| Gamma Deck (8–10 slides) | Ch 1, 2, 3, 5, 7, 10, 11 | Visual-first, minimal text |
| Blog Series (5–7 posts) | One post per chapter cluster | Deep dives with code examples |
| LinkedIn Carousel (10 slides) | Ch 2 + Ch 3 + Ch 11 | Hook → Framework → Proof |
| Video Script (10–15 min) | Ch 1–5 | Screen recordings + narration |
| Client Pitch (3 min) | Ch 1 + Ch 11 + Ch 3 | Problem → Results → How |
| Podcast Talking Points | All chapters, conversational | 30-min episode outline |

| | | |
|---|---|---|
| Lead Magnet PDF | Ch 12 expanded | Actionable guide with checklists |

*Built on verified live system state as of May 18, 2026. All agent statuses, costs, and capabilities confirmed via direct infrastructure inspection.*