

SPANFORGE

Complete Ecosystem Build List

The definitive map of every tool, utility, and integration to dominate AI observability

47 Tools • 6 Tiers • AgentOBS foundation • March 2026

Confidential — SpanForge / llm-toolkit project

How to Read This Document

This document is the single master build list for SpanForge — every tool, utility, CLI, integration, and intelligence layer needed to become the dominant platform in AI observability and LLM engineering. It synthesises the existing 17-tool LLM Developer Toolkit roadmap, the AgentOBS event schema standard, the AgentOBS utility ecosystem analysis, and new tools identified across all three sources.

47 entries are organised into six tiers:

Tier	What it is	Count
FOUNDATION	The shared schema and core infrastructure every other tool depends on	2
CORE	The 17 planned tools from the existing LLM Developer Toolkit roadmap	17
POWER	High-value utilities and CLIs that extend the core (AgentOBS-derived + new)	12
ENTERPRISE	Compliance, governance, security, and audit tooling for paying customers	7
VIRAL	Tools designed primarily to drive adoption, GitHub stars, and community	5
INTELLIGENCE	AI-powered analytics, anomaly detection, and insight layers	4

Priority bands indicate when to build:

- **P0 — BUILD NOW:** Blocks everything else. No substitution.
- **P1/P2 — HIGH:** Build within the first 6 months. Direct revenue or adoption path.
- **P3/P4 — MEDIUM:** Months 6–14. Solidifies moat, expands enterprise stickiness.
- **P5 — LONG TERM:** Post product-market fit. Intelligence and platform completeness.

Every tool emits and/or consumes AgentOBS events. The schema is the only architectural decision that is irreversible. Get it right before writing a single line of any other tool.

TIER 1 — FOUNDATION

Two tools. Both must ship before anything else. AgentOBS is the contract. agentobs-validate is the gatekeeper that enforces it. Without these two, the ecosystem has no spine.

01 AgentOBS *The shared event contract*

FOUNDATION P0 — BUILD NOW

The single architectural decision that transforms 17 isolated utilities into a composable platform. OpenTelemetry-compatible JSON envelope emitted by every tool in the ecosystem. If this is designed poorly, every subsequent tool must be refactored.

What it does: Python dataclasses and Pydantic models for all event types. JSON Schema for validation. Deterministic Canonical JSON serialisation (sorted keys, no nulls). ULID generation. HMAC-SHA256 signing primitives. Sensitivity/redaction type system. Published as a standalone PyPI package so any third-party tool can adopt it.

Why it wins: It is the only true zero-dependency foundation. Every competitor tool in this space (LangSmith, Arize, Langfuse) uses proprietary schemas. AgentOBS being open means the ecosystem compounds beyond SpanForge itself — third-party tools adopt it and each adoption makes SpanForge more central.

Depends on: *None*

Emits / consumes: *Defines: llm.trace.*, llm.cost.*, llm.eval.*, llm.guard.*, llm.prompt.*, llm.redact.*, llm.diff.*, llm.audit.*, llm.fence.*, llm.template.*, llm.cache.**

02 agentobs-validate *Schema compliance enforcer*

FOUNDATION P0 — BUILD NOW

Validates any JSON or JSONL file of events against the AgentOBS standard. The gatekeeper that makes the standard real. If teams cannot verify compliance, the schema is aspirational not structural. This is the first tool any new adopter will run.

What it does: Validates required envelope fields, event_type namespace patterns, ULID format, source pattern (name@semver), trace/span ID hex formats, HMAC signatures when present. Outputs per-event pass/fail with structured error detail. --json flag for machine-parseable output. Non-zero exit on failure for CI gates.

Why it wins: Every team adopting AgentOBS will run this first. High GitHub star potential. Positions SpanForge as the authoritative compliance authority for the standard.

Depends on: *AgentOBS*

Emits / consumes: *Consumes: any AgentOBS events. Emits: validation report to stdout/JSON*

TIER 2 — CORE

The 17 tools from the existing LLM Developer Toolkit roadmap. These are already planned with clear build order and revenue signals. Listed here in build-order sequence with the strategic rationale preserved.

03 llm-diff *LLM output comparison engine* DONE

CORE P0 — BUILD NOW

Compares two LLM responses word-level, semantically, or as JSON. The evaluation primitive that evalkit, promptlock, and agentboard all depend on. Already shipped.

What it does: Word-level, semantic, and JSON diff modes. Multi-provider (OpenAI, Anthropic, Groq, Mistral, Ollama). HTML report output — self-contained offline-shareable. Per-diff token counts and latency tracking.

Why it wins: Already the most composable evaluation primitive in the ecosystem. Every tool that needs to compare outputs reuses this rather than reimplementing it.

Depends on: *AgentOBS*

Emits / consumes: *Emits: llm.diff.completed (similarity score, token counts, latency)*

04 promptlock *Prompt version control* NEXT

CORE P1 — HIGH

Git-style versioning for prompts with enterprise RBAC, immutable audit log, approval workflows, and SSO. The first enterprise revenue tool in the stack. Design partner conversations start here.

What it does: CLI: save, log, diff, rollback, tag, push, pull. RBAC with 7 roles. Approval workflows. Environment management (dev/staging/prod). SSO (SAML/OIDC), SCIM provisioning. HashiCorp Vault / AWS SSM integration. Auto-runs llm-diff on promotion requests as eval gate.

Why it wins: No direct competitor offers prompt versioning with enterprise governance. LangSmith has prompt storage but no RBAC, no audit log, no approval workflows. This is the wedge into enterprise accounts.

Depends on: *AgentOBS, llm-diff*

Emits / consumes: *Emits: llm.prompt.saved, llm.prompt.promoted, llm.prompt.rolled_back*

05 promptblock *Prompt template engine* PLANNED

CORE P1 — HIGH

Jinja2-style templating with LLM-specific extensions. Variable management, composition, and inheritance for prompt files. Natural upsell for promptlock users.

What it does: Model routing blocks, conditional sections by environment. Prompt composition via inheritance. Variable type-checking at build time — catch missing variables before runtime. Env-specific value rendering.

Why it wins: Pairs so naturally with promptlock that existing users pull it in immediately. Drives promptlock Pro upsell.

Depends on: *AgentOBS, promptlock*

Emits / consumes: *Emits: llm.prompt.rendered (variable values, redacted if secret)*

06 llm-trace *Framework-agnostic agent span tracing* NEXT

CORE P1 — HIGH

The highest-value gap in the market. Framework-agnostic OpenTelemetry-compatible span tracing for multi-step agent runs. Works with raw OpenAI calls, AutoGen, CrewAI, LangChain, or entirely custom stacks. LangSmith is the competitor but only works inside LangChain.

What it does: Single decorator `@trace()` instruments any Python function as an agent span. Captures span start/end, tool calls, arguments, return values, model calls, branching, retries. Parent-child span relationships for full trace trees. Async-native. Exports to OTLP, Datadog, Grafana Tempo, Honeycomb, Jaeger.

Why it wins: Framework-agnostic is the moat. LangSmith's dependency on LangChain is the single biggest weakness of the incumbent. Every team not using LangChain has no good tracing option today.

Depends on: *AgentOBS*

Emits / consumes: *Emits: llm.trace.span.started, llm.trace.span.completed, llm.trace.span.failed, llm.trace.agent.step, llm.trace.agent.completed, llm.trace.reasoning.step*

07 llm-cost *Cost and latency tracker* PLANNED

CORE P2 — HIGH

Per-call, per-run, per-agent USD and token accounting with budget alerts and trend dashboards. Natural add-on for llm-trace users. Unlocks usage-based billing.

What it does: Per-call and per-trace USD cost calculation across all major providers. Updated pricing tables maintained as a versioned data file. Budget alerts via Slack/email/webhook. Rolling 7-day and 30-day dashboards in the terminal. Cost attribution by team, project, environment, or custom tag.

Why it wins: Engineering managers care intensely about AI cost attribution. No existing tool gives per-step cost breakdown inside a trace. This is a daily-use tool that creates strong retention.

Depends on: *AgentOBS, llm-trace*

Emits / consumes: *Consumes: llm.trace.span.completed. Emits: llm.cost.recorded, llm.cost.attributed*

08 llm-inspect *Tool call inspector* PLANNED

CORE P2 — HIGH

Surfaces every tool call in an agent run: function name, arguments, return value, time taken. Verifies whether the model actually used the tool result or silently discarded it. DevTools for agents.

What it does: Per-tool-call breakdown from agent traces. Flags discarded tool outputs. Interactive terminal replay: step through a recorded agent run call by call. Diff tool calls between two runs to spot what a prompt change caused.

Why it wins: The 'did the model actually use this tool?' question is asked by every agent developer. No existing tool answers it. This becomes a daily debugging tool.

Depends on: *AgentOBS, llm-trace*

Emits / consumes: *Consumes: llm.trace.span.* events. Emits: llm.inspect.report*

09 evalkit *Agent evaluation framework* PLANNED

CORE P2 — HIGH

Scenario-based, framework-agnostic agent evaluation. Define test cases, run against any agent, score outputs. Completes the quality assurance pipeline when combined with llm-diff and llm-trace.

What it does: Test scenarios: prompt + expected tool call sequence + output assertion. Framework-agnostic runner — calls any callable. Deterministic scoring (schema match, tool call match) and model-graded scoring (LLM-as-judge). Regression mode fails if score drops below baseline. CI-native: JSON output, non-zero exit, GitHub Action published.

Why it wins: evalkit + llm-diff + llm-trace = a complete QA pipeline that no single competitor offers. The GitHub Action makes this the default eval tool in every AI project's CI.

Depends on: *AgentOBS, llm-diff, llm-trace*

Emits / consumes: *Consumes: llm.trace.span.* events. Emits: llm.eval.score.recorded, llm.eval.regression.detected*

10 agentping *Agent health checker* PLANNED

CORE P2 — HIGH

Runs a test suite against a live agent endpoint and reports pass/fail per tool call. curl for agents — one command that tells you if your agent is healthy right now.

What it does: Supports HTTP endpoints, Python callables, WebSocket agents. Health checks: latency threshold, response schema, required tool calls, forbidden outputs. Prometheus metrics endpoint, Datadog checks, PagerDuty alerts integration.

Why it wins: The monitoring use case is universal — every team running an agent in production needs a health check. Simple enough to be adopted independently of the rest of the ecosystem.

Depends on: *AgentOBS*

Emits / consumes: *Emits: llm.eval.scenario.started, llm.eval.scenario.completed*

11 agentboard *Local run dashboard* PLANNED

CORE P1 — HIGH

Self-hosted localhost web UI for agent run history, span trees, tool call timeline, and cost. One pip install, zero account required. The OSS viral loop tool — developers who resist SaaS adopt this.

What it does: Span tree visualisation. Tool call timeline. Cost per run. Similarity scores from llm-diff. Filters by model, date, environment, tag, success/failure. Export any run as shareable HTML. SQLite backend by default. No telemetry, no cloud dependency.

Why it wins: Developers who distrust cloud observability platforms adopt this immediately. Once it's part of a team's workflow it becomes the natural gateway to the SaaS offering. The zero-friction entry point.

Depends on: *AgentOBS, llm-trace*

Emits / consumes: *Consumes: all AgentOBS events from local SQLite store*

12 promptguard *Prompt injection firewall* PLANNED

CORE

Detects prompt injection patterns in user input before they reach the model. Output scanning for jailbreak success indicators, policy violations, data exfiltration patterns. Configurable guardrail policies with risk scoring.

What it does: Allowlist/denylist rules, regex patterns, LLM-based detection for sophisticated attacks. Risk score per input and output — block, flag, or log based on configurable thresholds. Audit integration feeds into promptlock audit log.

Why it wins: EU AI Act and enterprise security teams mandate injection protection. This is the enterprise security add-on that enables SOC 2 and HIPAA compliance conversations.

Depends on: *AgentOBS*

Emits / consumes: *Emits: llm.guard.input.blocked, llm.guard.input.passed, llm.guard.output.blocked, llm.guard.output.passed*

13 llm-redact *PII redaction layer* FUTURE

CORE

Detects and redacts PII in prompts and responses before logging. GDPR/HIPAA-ready. Plugs directly into llm-trace. Implements the AgentOBS Privacy Profile requirements.

What it does: Detects: names, emails, phone numbers, SSNs, credit cards, addresses, medical identifiers. Redacts before logging — PII never hits the event store. Configurable entity types per deployment. In-process redaction — no data leaves the environment. Five-level sensitivity scale (LOW to PHI).

Why it wins: Every enterprise deploying AI in regulated industries needs this. HIPAA BAA conversations require it. Combined with llm-trace it becomes the compliance argument for healthcare and fintech customers.

Depends on: *AgentOBS*

Emits / consumes: *Emits: llm.redact.pii.detected, llm.redact.phi.detected, llm.redact.applied*

14 llm-fence *Output schema enforcement* FUTURE

CORE

Validates LLM output against a defined JSON Schema or Pydantic model. Automatic retry with updated prompt on schema mismatch. Works with any provider — no dependency on native structured output features.

What it does: Validates against JSON Schema Draft 2020-12 or any Pydantic model. Retry on mismatch with configurable budget (max retries, backoff, fallback value). Provider-agnostic — uses native structured output where available, falls back to validation-and-retry.

Why it wins: Structured output is table stakes for production agents. Every team building agents that call APIs or fill databases needs this. Simple to adopt independently.

Depends on: *AgentOBS*

Emits / consumes: *Emits: llm.fence.validated, llm.fence.retry.triggered, llm.fence.max_retries.exceeded*

15 toolsmith *Tool schema builder* *PLANNED*

CORE

Define LLM tools with Python decorators. Auto-generates OpenAI function calling schemas, Anthropic tool use schemas, and LangChain tool definitions from a single decorated function.

What it does: Python type annotations and docstrings become schema properties and descriptions automatically. Validates tool call arguments at runtime before executing. Generates schemas for all major providers from one definition.

Why it wins: Eliminates the most tedious part of building agents — writing and maintaining JSON schemas for every tool. Zero overhead to adopt; immediate day-one value.

Depends on: *AgentOBS*

Emits / consumes: *Emits: no events (schema generation utility); integrates with llm-trace for runtime validation logging*

16 llm-assert *LLM output assertion library* *FUTURE*

CORE

Fluent assertion library for LLM outputs. Plugs into pytest with no runner changes. Soft assertions collect all failures before surfacing them.

What it does: Fluent API:

`assert_that(response).contains('refund').matches_schema(schema).sentiment_above(0.5)`. Soft assertions: collect all failures in a run. Custom matchers: register domain-specific assertions (`is_valid_sql()`, `cites_only_approved_sources()`). pytest fixtures included.

Why it wins: pytest is how Python developers test everything. An LLM assertion library in that idiom will be adopted by every team that uses pytest — which is almost all of them.

Depends on: *AgentOBS*

Emits / consumes: *Emits: llm.eval.score.recorded*

17 llm-retry *Retry and fallback engine* *FUTURE*

CORE

Configurable retry with exponential backoff and cross-provider fallback. Circuit breakers included. Cost-aware routing prefers cheapest provider that meets latency requirements.

What it does: Exponential backoff for rate limits, timeouts, transient errors. Cross-provider fallback: gpt-4o fails → route to claude-3-5-sonnet or local model. Circuit breaker: stop retrying failing provider and route all traffic to fallback. Cost-aware routing.

Why it wins: Production reliability for LLM calls is an unsolved problem for most teams. This is the first tool many teams will grab when gpt-4o rate-limits them in production.

Depends on: *AgentOBS*

Emits / consumes: *Emits: llm.trace.span.failed (on failure), routing decision metadata in payload*

18 llm-cache *Semantic caching layer* 🧠 FUTURE

CORE

Deduplicates near-identical prompts using embedding similarity. A query 95% similar to a cached one returns the cached response. Cuts costs on repeated or similar queries.

What it does: Configurable similarity threshold per use case. Storage backends: in-memory (dev), Redis (production), SQLite (local). TTL-based and manual cache invalidation. promptlock prompt changes can trigger cache invalidation automatically.

Why it wins: Cache hits are pure cost savings with zero latency penalty. Easy to justify to any engineering manager. Tight integration with promptlock creates a virtuous loop where prompt changes automatically invalidate stale cache entries.

Depends on: *AgentOBS, llm-trace*

Emits / consumes: *Emits: llm.cache.hit, llm.cache.miss, llm.cache.evicted, llm.cache.written*

19 llm-cost-prices *Provider pricing data feed* 📋 PLANNED

CORE P2 — HIGH

A continuously maintained, versioned data file of current pricing for every major LLM provider. Used internally by llm-cost but published as a standalone resource. Becomes the authoritative pricing reference for the ecosystem.

What it does: JSON data file covering all major providers and all current models. Updated with every provider price change. Versioned by effective date. CLI to query current and historical pricing for any model. Used by llm-cost and agentobs-cost-forecast.

Why it wins: Provider prices change frequently and teams get caught paying old rates. Being the authoritative source for LLM pricing data drives constant return traffic from developers.

Depends on: *None*

Emits / consumes: *Consumed by: llm-cost, agentobs-cost-forecast, agentobs-replay*

TIER 3 — POWER UTILITIES

Twelve high-value tools derived from AgentOBS ecosystem analysis and gap mapping. These extend the core ecosystem into daily workflow territory. Many have higher viral potential than core tools because they solve immediate, tangible pain without requiring deep platform adoption.

20 **agentobs-view** *Trace viewer CLI*

POWER

The kubectl for SpanForge traces. A rich terminal trace viewer that lets developers explore agent runs, spans, and costs without writing any code. The first tool a developer runs after instrumenting their agent.

What it does: agentobs-view traces — list all recent runs. agentobs-view trace <id> — drill into a specific run showing all spans, token counts, tool calls, cost, and duration. agentobs-view spans <trace_id> — show the span tree. Colour-coded status, duration heatmap, cost per step. Works directly on any AgentOBS JSONL file.

Why it wins: The gap between 'I have traces' and 'I can explore traces' is currently a code-writing exercise. This CLI closes that gap in one install. High Reddit/HN visibility potential because it makes traces tangible immediately.

Depends on: AgentOBS

Emits / consumes: Consumes: any AgentOBS JSONL event file

21 **agentobs-diff** *Trace comparison tool*

POWER

Compares two full agent traces rather than just individual LLM outputs. Shows exactly what changed in behaviour between a prompt change, a model swap, or a code refactor. Fills a critical daily workflow gap.

What it does: Side-by-side comparison of two trace files. Highlights: changed tool call sequences, token count deltas, cost differences, latency changes per step, new or removed spans, changed model parameters. Output as terminal diff, JSON, or HTML report. Integrates with evalkit as a regression primitive.

Why it wins: Before/after comparison of agent behaviour is the most common debugging workflow for AI teams. Today they do this manually. agentobs-diff automates it completely and is the kind of tool that gets shared in Slack with 'you need this'.

Depends on: AgentOBS, llm-diff

Emits / consumes: Consumes: two AgentOBS JSONL files. Emits: llm.diff.computed

22 **agentobs-replay** *Trace replay and regression engine*

POWER

Re-runs a recorded agent trace against a new model or modified prompt and diffs the output against the original. Makes regression testing of agents as easy as running a command. The most novel tool in this tier.

What it does: Takes a recorded JSONL trace as a fixture. Re-executes all model calls with updated parameters (model, prompt, temperature). Diffs new outputs against originals using llm-diff. Reports: cost delta, latency delta, output similarity, tool call sequence changes. Can be run in CI against a suite of golden traces.

Why it wins: Agent regression testing is completely unsolved today. Teams manually eyeball outputs after changes. agentobs-replay makes this systematic and automatable. A tool with no direct competitor that solves a genuinely painful problem.

Depends on: *AgentOBS, llm-diff, llm-trace*

Emits / consumes: *Consumes: AgentOBS JSONL trace. Emits: llm.eval.regression.detected, llm.diff.computed*

23 agentobs-cost-forecast *Cost projection tool*

POWER

Given a sample of recent traces, projects monthly and annual costs at scale. Answers the engineering manager question: 'If this agent runs 10,000 times a day what does it cost?' before shipping to production.

What it does: Analyses a sample of actual traces. Extrapolates to user-defined scale (calls/day, calls/month). Breaks down cost by: model, step, token type (input/output/cached/reasoning). Compares cost across models for the same workload. Outputs a cost report as terminal table, JSON, or PDF.

Why it wins: The 'what will this cost?' question is asked before every AI feature ships. Today teams estimate manually. agentobs-cost-forecast uses real trace data to answer it precisely. Engineering managers will share this tool with their teams.

Depends on: *AgentOBS, llm-cost, llm-cost-prices*

Emits / consumes: *Consumes: AgentOBS JSONL traces. Emits: cost projection report*

24 agentobs-stats *Trace analytics CLI*

POWER

Simple aggregate analytics over a trace file or directory of traces. Total traces, spans, tokens, cost, top models, error rates, average latency — in one command.

What it does: agentobs-stats events.jsonl — instant summary. Breakdowns by model, provider, event type, error type, tag. Rolling trend over time if multiple files provided. --json flag for piping into other tools or dashboards. Weekly/monthly summary email mode.

Why it wins: The 'give me a summary of what happened' question is universal. One command that answers it becomes a daily tool. High PyPI download potential from casual users who do not need the full platform.

Depends on: *AgentOBS*

Emits / consumes: *Consumes: AgentOBS JSONL files. Outputs to stdout or JSON*

25 agentobs-anomaly *Trace anomaly detector*

POWER

Flags agent runs that deviate from baseline behaviour. Unusually high token counts, unexpected tool call patterns, latency spikes, cost outliers, error rate increases. Requires no ML — z-score and IQR methods catch 80% of real production issues.

What it does: Builds a statistical baseline from a set of reference traces. Flags deviations: token count anomalies, unusual tool call sequences, latency outliers, cost spikes. Configurable sensitivity. Integrates with alerting (Slack, PagerDuty). Can run as a daemon watching a live event stream.

Why it wins: Production AI teams have no equivalent of traditional APM anomaly detection. This is that tool for agents. Positions SpanForge as an active monitoring platform not just a passive log viewer.

Depends on: *AgentOBS, llm-trace*

Emits / consumes: *Consumes: AgentOBS event streams. Emits: llm.eval.regression.detected, alert events*

26 agentobs-export *Format converter*

POWER P3 — MEDIUM

Converts AgentOBS JSONL event files to other formats for downstream analysis. CSV, Parquet, SQLite, DuckDB. Bridges SpanForge traces to existing data science and analytics workflows.

What it does: `agentobs-export events.jsonl --format csv`. Supported: CSV, Parquet (columnar, efficient for large datasets), SQLite (queryable), DuckDB (in-process analytics). Schema-aware: flattens nested payload fields into columns intelligently. Configurable field selection.

Why it wins: Data scientists and analysts do not live in the terminal. They work in Jupyter, Excel, Tableau. `agentobs-export` is the bridge. Every data team at a company using SpanForge will use this.

Depends on: *AgentOBS*

Emits / consumes: *Consumes: AgentOBS JSONL. Outputs: CSV, Parquet, SQLite, DuckDB files*

27 agentobs-sample *Production log sampler*

POWER P3 — MEDIUM

Samples large production trace logs for analysis, testing, or sharing. Supports random sampling, stratified sampling by model/status/cost, and head/tail windowing. Essential for teams with high-volume production logs.

What it does: `agentobs-sample events.jsonl --rate 10%`. Stratified sampling: sample proportionally by model, provider, error type, or custom tag. Head/tail windows: take the first N or last N events. Reservoir sampling for streaming input. Outputs a valid AgentOBS JSONL file.

Why it wins: Production logs at scale are too large for most analysis tools. Sampling is infrastructure plumbing that no one wants to write from scratch. Making it first-class in SpanForge means every other tool works seamlessly on large datasets.

Depends on: *AgentOBS*

Emits / consumes: *Consumes and emits: AgentOBS JSONL files (subset)*

28 agentobs-merge *Multi-source trace merger*

POWER P3 — MEDIUM

Merges trace files from multiple agents, services, or time windows into a single coherent AgentOBS JSONL file. Essential for distributed systems and parallel agent architectures.

What it does: `agentobs-merge run1.jsonl run2.jsonl run3.jsonl`. Deduplicates by `event_id`. Sorts by timestamp. Handles overlapping trace IDs across sources. Validates merged output against AgentOBS. Optional `--by-trace-id` mode preserves trace groupings.

Why it wins: Multi-agent and distributed AI systems produce fragmented trace files. Merging them manually is error-prone. This is a necessary utility for any team running agents in parallel.

Depends on: *AgentOBS*

Emits / consumes: *Consumes and emits: AgentOBS JSONL files*

29 agentobs-gen *Synthetic trace generator*

POWER

Generates realistic synthetic agent traces conforming to AgentOBS. Used for tutorials, testing, demos, and bootstrapping agentboard with data before a team has real production traffic.

What it does: `agentobs-gen --agent research_agent --steps 5 --model gpt-4o`. Configurable: number of steps, tool call patterns, cost ranges, error rates, model mix. Produces valid AgentOBS JSONL with all required fields. Used in all SpanForge tutorials and documentation examples.

Why it wins: Without `agentobs-gen`, every tutorial requires the reader to instrument their own agent before seeing any output. `agentobs-gen` removes that barrier entirely. It is the onboarding accelerator for the entire ecosystem.

Depends on: *AgentOBS*

Emits / consumes: *Emits: valid AgentOBS JSONL trace files*

30 agentobs-lint *SDK instrumentation linter*

POWER P3 — MEDIUM

Static analysis for codebases using the AgentOBS SDK. Checks that all required fields are being populated, PII-sensitive strings are typed as Redactable, cost is being tracked, event types are valid. Catches instrumentation gaps before they reach production.

What it does: Integrates with standard Python linters (flake8, ruff). Checks: missing required payload fields, bare strings in PII-sensitive positions (should be Redactable), unregistered event type strings, missing trace context propagation. Emits warnings for incomplete instrumentation.

Why it wins: Instrumentation gaps are invisible until an event is missing from a dashboard. `agentobs-lint` surfaces them at development time. This is the tool that makes SchemaConformance claims meaningful — you can verify you're conformant before running tests.

Depends on: *AgentOBS*

Emits / consumes: *Static analysis tool — no runtime events*

31 agentobs-guard-report *Security event summariser*

POWER

Summarises guard and fence events across a time window. 'Your agent blocked 23 inputs today — here are the top 3 trigger patterns.' The security dashboard in a CLI that compliance teams ask for the moment they start using promptguard.

What it does: Aggregates `llm.guard.*` events over configurable time windows. Groups by: block reason, risk score range, input pattern clusters. Top N trigger patterns. Trend over time. Export as PDF for compliance reporting. Slack/email scheduled delivery.

Why it wins: Security and compliance teams need reports, not event streams. `agentobs-guard-report` bridges the raw event data to the format those teams consume. Direct sales enabler for enterprise deals.

Depends on: *AgentOBS, promptguard*

Emits / consumes: *Consumes: llm.guard.* events. Outputs: compliance report (terminal, JSON, PDF)*

TIER 4 — ENTERPRISE

Seven tools that turn SpanForge from a developer favourite into an enterprise platform. These generate the highest per-seat revenue and unlock regulated industries. None are necessary for OSS adoption but all are necessary for \$1M+ ARR.

32 spanforge-audit *Compliance-grade audit chain*

ENTERPRISE P2 — HIGH

Implements HMAC-SHA256 tamper-evident audit chains across all AgentOBS events. Every event is signed. Any modification, deletion, or insertion is detectable. SOC 2, ISO 27001, and EU AI Act compliance argument.

What it does: Single-event signing: SHA-256 payload checksum + HMAC-SHA256 sig over event_id + checksum + prev_id. Chain construction: each event's prev_id links to the previous event_id. verify_chain() returns: valid bool, first_tampered event_id, gaps list, tampered count. Key rotation without breaking chain continuity.

Why it wins: Compliance teams at enterprise customers ask 'can you prove this log hasn't been tampered with?' constantly. The answer today is always no. spanforge-audit makes it yes. This is the difference between a demo and a signed enterprise contract.

Depends on: AgentOBS

Emits / consumes: Emits: `llm.audit.key.rotated`; adds signature and checksum fields to all events

33 spanforge-governance *Schema lifecycle governance*

ENTERPRISE P3 — MEDIUM

EventGovernancePolicy, ConsumerRegistry, and DeprecationRegistry in one package. Operators configure blocking and warning rules on event types at runtime. Multi-team schema evolution managed programmatically.

What it does: blocked_types: hard-block event types (raises GovernanceViolationError). warn_deprecated: soft-block with DeprecationWarning. custom_rules: arbitrary cross-field governance constraints. ConsumerRegistry: tools declare schema version dependencies and fail fast if incompatible. DeprecationRegistry: sunset tracking with NEXT_MAJOR/MINOR/LONG_TERM/UNSCHEDULED policies.

Why it wins: Large engineering organisations managing dozens of teams all emitting events need governance. Without it, schema drift reaches production silently. spanforge-governance is the tool that makes multi-team SchemaConformance sustainable.

Depends on: AgentOBS

Emits / consumes: Emits: governance violation events, deprecation warnings via standard logging

34 spanforge-siem *SIEM and log pipeline exporter*

ENTERPRISE P3 — MEDIUM

Exports SpanForge events to enterprise SIEM and log management platforms. Splunk, Elastic SIEM, IBM QRadar, Microsoft Sentinel. Maps AgentOBS events to native platform schemas with no adapter code required on the customer side.

What it does: Pre-built connectors for Splunk HEC, Elasticsearch index API, QRadar syslog, Sentinel workspace API. Field mapping configuration per platform. Batching, retry, dead-letter queue. TLS mutual authentication. Audit events routed separately from observability events.

Why it wins: Enterprise security teams route everything through their SIEM. If SpanForge cannot feed the SIEM, it will not be approved for enterprise deployment. This unblocks the security review.

Depends on: *AgentOBS, spanforge-audit*

Emits / consumes: *Consumes: all AgentOBS events. Pushes to external SIEM platforms*

35 spanforge-rbac *Multi-tenant RBAC and isolation*

ENTERPRISE P3 — MEDIUM

Role-based access control and tenant isolation for multi-team and multi-org SpanForge deployments. Controls who can read, write, and export which event namespaces. Verifies that events from different orgs never mix in the same stream.

What it does: Role definitions: viewer, analyst, operator, admin, auditor, security, superadmin. Namespace-level permissions: team A can see `llm.trace.*` but not `llm.audit.*`. Multi-tenant isolation verification: `verify_tenant_isolation()` and `verify_events_scoped()`. SSO integration (SAML/OIDC).

Why it wins: Enterprise procurement requires documented access controls. No AI observability platform today has serious RBAC — they all assume single-tenant. SpanForge's multi-tenant design with real RBAC is a direct enterprise procurement differentiator.

Depends on: *AgentOBS, spanforge-audit*

Emits / consumes: *Emits: access control events to `llm.audit.*` namespace*

36 spanforge-dataretention *Data retention and erasure manager*

ENTERPRISE P3 — MEDIUM

Manages event data lifecycle: configurable retention periods per namespace, automated expiry, GDPR right-to-erasure handling for HMAC-chained audit logs. The tool that makes Right to Erasure legally defensible.

What it does: Retention policies per namespace (`llm.trace.*` for 90 days, `llm.audit.*` for 7 years). Automated expiry with cryptographic proof of deletion. Right-to-erasure: produces successor chain excluding deleted events while preserving chain integrity for remaining events. Data residency configuration (EU/US/APAC).

Why it wins: GDPR Right to Erasure + HMAC audit chains is a genuine technical challenge — deleting an event breaks the chain. `spanforge-dataretention` solves this with successor chains. It is the tool that unlocks the European enterprise market.

Depends on: *AgentOBS, spanforge-audit*

Emits / consumes: *Emits: erasure confirmation events to `llm.audit.*` namespace*

37 spanforge-compliance-cli *CI/CD compliance gate*

A CLI and GitHub Action that runs the full AgentOBS/SpanForge compliance check suite against any event file and exits non-zero on failure. The tool that turns schema conformance from aspirational to enforceable.

What it does: CHK-1: all required envelope fields present. CHK-2: all event_type values are registered. CHK-3: all source fields match name@semver. CHK-4: all event_id values are valid ULIDs. Chain integrity check. Tenant isolation check. Profile conformance claim verification (Core/Security/Privacy/Enterprise). --json flag for machine-parseable output.

Why it wins: CI gates make standards stick. A GitHub Action that marks a PR as failing if new events do not conform is how the standard spreads virally across engineering teams. This is also the enterprise audit evidence tool.

Depends on: *AgentOBS, spanforge-audit, agentobs-validate*

Emits / consumes: *Outputs: compliance report (terminal, JSON). No event emission — static analysis*

38 spanforge-tenant-dashboard *Enterprise multi-tenant dashboard*

Web dashboard for enterprise operators managing multiple teams and projects. Org-level cost rollup, cross-team trace comparison, policy enforcement visibility, audit trail viewer. The tool that justifies the enterprise licence fee.

What it does: Org-level cost attribution and chargeback reports. Cross-team model usage comparison. Policy violation dashboard. Audit chain integrity status. User access audit log. Multi-environment (dev/staging/prod) aggregated views. Export to PDF for executive reporting.

Why it wins: Enterprise buyers need to justify the platform cost to non-technical stakeholders. A dashboard that shows org-wide AI cost, model usage, and compliance status in one screen is that justification.

Depends on: *AgentOBS, llm-cost, spanforge-audit, spanforge-rbac*

Emits / consumes: *Consumes all AgentOBS events; no new event emission*

TIER 5 — VIRAL

Five tools designed primarily to generate adoption, GitHub stars, and community. Each is independently useful but their strategic purpose is to bring developers into the SpanForge ecosystem at zero friction.

39 spanforge-github-action *CI/CD schema validation action*

VIRAL P1 — HIGH

A published GitHub Action that runs agentobs-validate on every PR. One YAML block in .github/workflows/ and every event file change is validated automatically. The fastest path to the standard appearing in thousands of repositories.

What it does: Runs agentobs-validate + spanforge-compliance-cli on any JSONL event files changed in the PR. Annotates the PR with per-event validation errors. Configurable: warn-only mode or fail mode. Badges: 'SpanForge Validated' for repo READMEs. Published to GitHub Actions Marketplace.

Why it wins: GitHub Actions Marketplace is the most efficient distribution channel for developer tools. Once this action is in a project it is never removed. Every fork and clone carries it. This is the viral adoption mechanism.

Depends on: agentobs-validate, spanforge-compliance-cli

Emits / consumes: No event emission — CI integration only

40 spanforge-vscode *VS Code extension*

VIRAL P2 — HIGH

VS Code extension that provides inline schema validation, autocomplete for AgentOBS event types, hover documentation for all fields, and a local trace viewer panel. Makes the standard discoverable at the point of development.

What it does: IntelliSense for AgentOBS Pydantic models. Inline error highlights for invalid field values. Hover docs for every envelope field and payload type. Embedded agentobs-view panel showing recent traces from the workspace. 'Open in agentboard' link from any trace file.

Why it wins: VS Code extensions get installed by word of mouth faster than any other tool category. A developer who installs it shows it to their team. The trace viewer panel alone will keep people in the extension for hours.

Depends on: AgentOBS

Emits / consumes: No event emission — development tool

41 spanforge-mcp *Model Context Protocol server*

VIRAL P2 — HIGH

An MCP server that exposes SpanForge trace data, cost analytics, and compliance reports to any MCP-compatible AI assistant. Ask Claude or Cursor: 'What did my agent spend last week?' or 'Show me the 3 slowest traces from yesterday.'

What it does: MCP tools: `get_recent_traces`, `get_trace_by_id`, `get_cost_summary`, `get_anomaly_alerts`, `get_compliance_status`, `run_validation`, `generate_cost_forecast`. Works with any MCP client (Claude Desktop, Cursor, Windsurf). Read-only by default; write mode for triggering replays.

Why it wins: MCP is the distribution channel of 2026 for developer tools. Every AI-native developer uses an MCP-compatible editor. An MCP server makes SpanForge data queryable in natural language from the tools developers already have open all day.

Depends on: *AgentOBS, llm-trace, llm-cost, agentobs-stats*

Emits / consumes: *Exposes AgentOBS event data via MCP protocol; no new events*

42 spanforge-playground *Hosted schema playground*

VIRAL P2 — HIGH

A hosted web UI at playground.spanforge.dev where developers can paste any JSON, validate it against AgentOBS, see the field errors inline, and generate a corrected event with one click. Zero install friction.

What it does: Paste any JSON → instant validation against current AgentOBS. Error annotations inline on the JSON. 'Fix it' button generates a schema-compliant version. Shareable URL for each validation result. 'Download as JSONL' output. Links to relevant documentation for each error.

Why it wins: Hosted playgrounds are the fastest way to make a standard approachable. The shareable URL means developers post links in Slack and GitHub Issues, which drives organic discovery. This is the top-of-funnel for everyone who has never heard of SpanForge.

Depends on: *AgentOBS*

Emits / consumes: *No event emission — hosted web tool*

43 spanforge-badge *Schema conformance badge system*

VIRAL P3 — MEDIUM

A badge service that lets any OSS project display their AgentOBS/SpanForge conformance profile (Core, Security, Privacy, Enterprise) in their README. 'AGENTOBS-Core-2.0 Certified' with a link to the validation report.

What it does: Developers submit their package name and conformance claims.

`spanforge-compliance-cli` runs against their published event schema. On pass: a dynamic badge SVG is served from badge.spanforge.dev. Badge updates automatically on new releases via GitHub webhook.

Why it wins: README badges create social proof at the exact moment another developer is evaluating a library. Every OSS project that earns a badge is a billboard for SpanForge. This is the mechanism by which the standard spreads to third-party tools.

Depends on: *spanforge-compliance-cli*

Emits / consumes: *No event emission — hosted badge service*

TIER 6 — INTELLIGENCE

Four AI-powered tools that transform SpanForge from an observability platform into an intelligence platform. These are post-PMF investments — build them when there is enough trace data to make them genuinely useful.

44 spanforge-insights *AI-powered trace analyst*

INTELLIGENCE P5 — LONG TERM

An AI layer over trace data that answers natural language questions about agent behaviour, cost patterns, and quality trends. 'Why did costs spike on Tuesday?' 'Which tool call is causing the most failures?' 'Is my new prompt actually better?'

What it does: Powered by the Anthropic API with tool use. Tools: `query_traces`, `compute_stats`, `run_diff`, `get_cost_breakdown`, `run_anomaly_detection`. Maintains conversation context across multiple questions. Produces cited answers with links to specific trace events that support the conclusion.

Why it wins: The jump from 'here is your data' to 'here is what your data means' is where platforms capture massive value. Every observability platform is racing to build this. SpanForge's structured event schema makes it dramatically more accurate than competitors working with unstructured logs.

Depends on: *AgentOBS, llm-trace, llm-cost, agentobs-anomaly, llm-diff*

Emits / consumes: *Consumes: AgentOBS event data. Emits: insight reports*

45 spanforge-optimizer *Prompt and model optimiser*

INTELLIGENCE P5 — LONG TERM

Analyses trace history to suggest specific optimisations: 'Switching step 3 from gpt-4o to gpt-4o-mini would save 60% cost with <5% quality drop based on your eval scores.' Turns observability data into actionable engineering decisions.

What it does: Analyses token usage patterns to suggest caching opportunities. Identifies steps where a cheaper model would match quality based on evalkit scores. Detects over-prompted steps (high token count, low information density). Suggests retry budget reductions based on actual failure rates. Every suggestion includes cost/quality impact estimate.

Why it wins: Optimisation advice grounded in a team's own production data is worth far more than generic best-practice guides. This tool justifies the entire platform investment in a single report.

Depends on: *AgentOBS, llm-cost, evalkit, llm-cache, llm-trace*

Emits / consumes: *Consumes: production trace history. Emits: optimisation recommendation report*

46 spanforge-benchmark *Cross-provider benchmark runner*

INTELLIGENCE P4 — MEDIUM

Runs a standardised benchmark suite against any set of LLM providers and models, producing a SpanForge-formatted comparison report. Quality scores, cost per task, latency, and consistency across providers.

What it does: Pre-built task suites: reasoning, code generation, summarisation, tool use, structured output. Custom task suite definition. Runs each task N times per model for statistical significance. Output: ranked comparison table, per-task breakdowns, cost efficiency scores, consistency (variance) analysis. Shareable HTML report.

Why it wins: Model selection is one of the hardest decisions AI teams make. A benchmark runner that uses your actual task types — not generic benchmarks — gives results that are directly actionable. Published benchmark reports also drive organic traffic.

Depends on: *AgentOBS, llm-diff, evalkit, llm-cost*

Emits / consumes: *Emits: llm.eval.score.recorded events. Outputs: benchmark report HTML*

47 spanforge-forecaster *Capacity and budget forecaster*

INTELLIGENCE PS — LONG TERM

Time-series forecasting of token usage, cost, and latency trends. Predicts when a team will hit budget limits, when latency will degrade under load, and what capacity to provision. Uses actual production trace history as the training signal.

What it does: Time-series models fitted to rolling trace data. Forecasts: monthly cost trajectory, token volume growth, latency trends under load. Budget alert predictions: 'At current growth rate you will exceed \$10k/month in 6 weeks.' Scenario modelling: 'If you add this new feature running 500 times/day, here is the cost impact.' Export as chart or PDF.

Why it wins: Finance and engineering leadership both need forward-looking cost projections for AI features. Today these are manual spreadsheet exercises. spanforge-forecaster makes them automatic and accurate because they are based on real production data.

Depends on: *AgentOBS, llm-cost, agentobs-cost-forecast*

Emits / consumes: *Consumes: production trace history. Emits: forecast report*

Master Build Order

All 47 tools ordered by the sequence in which they should be built, grouped by phase. The critical path is AgentOBS → agentobs-validate → llm-trace. Nothing else matters until these three are solid.

#	Tool	Tier	Phase	Revenue / Adoption Signal
01	AgentOBS	FOUNDATION	Phase 0 — Now	Zero revenue. 100% blocking dependency.
02	agentobs-validate	FOUNDATION	Phase 0 — Now	High GitHub star potential. Standards credibility.
03	llm-diff	CORE	✅ Done	Community tool. evalkit dependency.
04	agentobs-gen	POWER	Phase 0 — Now	Unblocks all tutorials and onboarding.
05	promptlock	CORE	Phase 1 — Wks 1–6	First enterprise revenue. Design partner ARR.
06	agentobs-view	POWER	Phase 1 — Wks 1–6	High OSS adoption. Organic discovery.
07	llm-trace	CORE	Phase 1 — Wks 7–12	Highest market value gap. LangSmith switchers.
08	agentobs-diff	POWER	Phase 1 — Wks 7–12	Daily workflow tool. Viral share potential.
09	promptblock	CORE	Phase 2 — Wks 7–10	promptlock Pro upsell. Immediate for existing users.
10	spanforge-github-action	VIRAL	Phase 2 — Wks 7–10	Viral distribution. Appears in repos automatically.
11	llm-cost	CORE	Phase 2 — Wks 17–20	Usage-based billing unlocked.
12	llm-cost-prices	CORE	Phase 2 — Wks 17–20	Authoritative pricing reference. Return traffic.
13	agentobs-cost-forecast	POWER	Phase 2 — Wks 17–20	Engineering manager tool. Justifies investment.

#	Tool	Tier	Phase	Revenue / Adoption Signal
14	llm-inspect	CORE	Phase 2 — Wks 17–20	Daily debugging tool. llm-trace add-on.
15	agentobs-stats	POWER	Phase 2 — Wks 17–20	High download volume. One-command analytics.
16	agentobs-replay	POWER	Phase 2 — Wks 17–20	Novel, no direct competitor. CI regression fixture.
17	spanforge-audit	ENTERPRISE	Phase 3 — Mo 4–6	SOC 2 / ISO 27001 unlock. First compliance deals.
18	spanforge-compliance-cli	ENTERPRISE	Phase 3 — Mo 4–6	CI gate. Schema adoption viral mechanism.
19	evalkit	CORE	Phase 3 — Mo 6–8	Evaluation SaaS tier. Complete QA pipeline.
20	agentping	CORE	Phase 3 — Mo 6–8	Production monitoring. Prometheus integration.
21	agentobs-anomaly	POWER	Phase 3 — Mo 6–8	Active monitoring. APM-equivalent for AI.
22	agentobs-guard-report	POWER	Phase 3 — Mo 6–8	Security report. Direct enterprise sales enabler.
23	promptguard	CORE	Phase 4 — Mo 8–10	Enterprise security add-on. EU AI Act argument.
24	agentboard	CORE	Phase 4 — Mo 8–10	OSS viral loop. SaaS gateway tool.
25	spanforge-vscode	VIRAL	Phase 4 — Mo 8–10	Point-of-development discovery. Word of mouth.
26	spanforge-mcp	VIRAL	Phase 4 — Mo 8–10	2026 distribution channel. AI editor integration.
27	spanforge-playground	VIRAL	Phase 4 — Mo 8–10	Zero-friction top of funnel. Shareable URL links.

#	Tool	Tier	Phase	Revenue / Adoption Signal
28	toolsmith	CORE	Phase 5 — Mo 10–12	Zero-overhead adoption. Agent developer daily tool.
29	llm-assert	CORE	Phase 5 — Mo 10–12	pytest ecosystem integration. Broad OSS uptake.
30	agentobs-export	POWER	Phase 5 — Mo 10–12	Data science bridge. Excel/Jupyter workflows.
31	agentobs-lint	POWER	Phase 5 — Mo 10–12	Conformance verification. CI integration.
32	spanforge-governance	ENTERPRISE	Phase 5 — Mo 10–12	Multi-team schema evolution. Enterprise stickiness.
33	spanforge-rbac	ENTERPRISE	Phase 5 — Mo 10–12	Enterprise procurement requirement.
34	spanforge-siem	ENTERPRISE	Phase 5 — Mo 10–12	Security team approval requirement.
35	llm-redact	CORE	Phase 6 — Mo 12–14	GDPR/HIPAA unlock. Healthcare and fintech.
36	llm-fence	CORE	Phase 6 — Mo 12–14	Structured output. Production agent reliability.
37	llm-retry	CORE	Phase 6 — Mo 12–14	Production reliability. Rate limit handling.
38	llm-cache	CORE	Phase 6 — Mo 12–14	Cost reduction. promptlock cache invalidation.
39	spanforge-dataretention	ENTERPRISE	Phase 6 — Mo 12–14	GDPR Right to Erasure. EU enterprise market unlock.
40	agentobs-merge	POWER	Phase 6 — Mo 12–14	Distributed agents. Multi-source trace assembly.
41	agentobs-sample	POWER	Phase 6 — Mo 12–14	Large-scale production log handling.

#	Tool	Tier	Phase	Revenue / Adoption Signal
42	spanforge-tenant-dashboard	ENTERPRISE	Phase 6 — Mo 14+	Enterprise deal justification. Executive reporting.
43	spanforge-badge	VIRAL	Phase 6 — Mo 14+	Third-party adoption. README billboard effect.
44	spanforge-benchmark	INTELLIGENCE	Phase 6 — Mo 14+	Model selection tool. Published reports = traffic.
45	spanforge-insights	INTELLIGENCE	Post-PMF	Platform intelligence layer. Premium SaaS tier.
46	spanforge-optimizer	INTELLIGENCE	Post-PMF	Justifies full platform investment. Premium tier.
47	spanforge-forecaster	INTELLIGENCE	Post-PMF	Finance and leadership reporting. Premium tier.

What This List Adds to the Existing Roadmap

The 17-tool LLM Developer Toolkit roadmap is a strong core plan. This master list adds 30 additional tools across five categories of gaps that the original roadmap did not address.

Gap 1: Developer Experience Tools (Tier 3 & 5)

The original roadmap builds the tools but does not build the tooling around the tools. `agentobs-view`, `agentobs-diff`, `agentobs-stats`, `agentobs-gen`, and `agentobs-lint` are the daily workflow layer that makes the schema tangible. Without these, developers instrument their agents but have nothing convenient to do with the data. These are the tools that get shared in Slack, posted on HN, and drive organic star growth.

Gap 2: Active Monitoring (`agentobs-anomaly`)

Every tool in the original roadmap is passive — it records and displays what happened. `agentobs-anomaly` is the first active tool: it watches what is happening and alerts when behaviour deviates from baseline. This is the jump from log viewer to monitoring platform. No existing AI observability tool has credible anomaly detection. Being first here is significant.

Gap 3: Viral Distribution Mechanisms (Tier 5)

The original roadmap has no explicit viral mechanisms. The GitHub Action, VS Code extension, MCP server, hosted playground, and badge system are the five channels through which SpanForge spreads beyond the teams that consciously choose to adopt it. The MCP server in particular is the 2026 distribution channel — every AI-native developer has an MCP-compatible editor open all day.

Gap 4: Compliance and Enterprise Completeness (Tier 4)

The original roadmap has security tools (`promptguard`, `llm-redact`) but no compliance infrastructure. `spanforge-audit`, `spanforge-compliance-cli`, `spanforge-siem`, `spanforge-rbac`, and `spanforge-dataretention` are the five tools that unlock regulated enterprise deals. Without them, SpanForge is a great developer tool but cannot pass a security review at a bank, hospital, or government contractor.

Gap 5: Intelligence Layer (Tier 6)

The original roadmap stores and displays data. The intelligence layer interprets it. `spanforge-insights`, `spanforge-optimizer`, `spanforge-benchmark`, and `spanforge-forecaster` turn SpanForge from a passive record into an active advisor. This is the platform premium tier — the reason to pay 10x the base subscription.