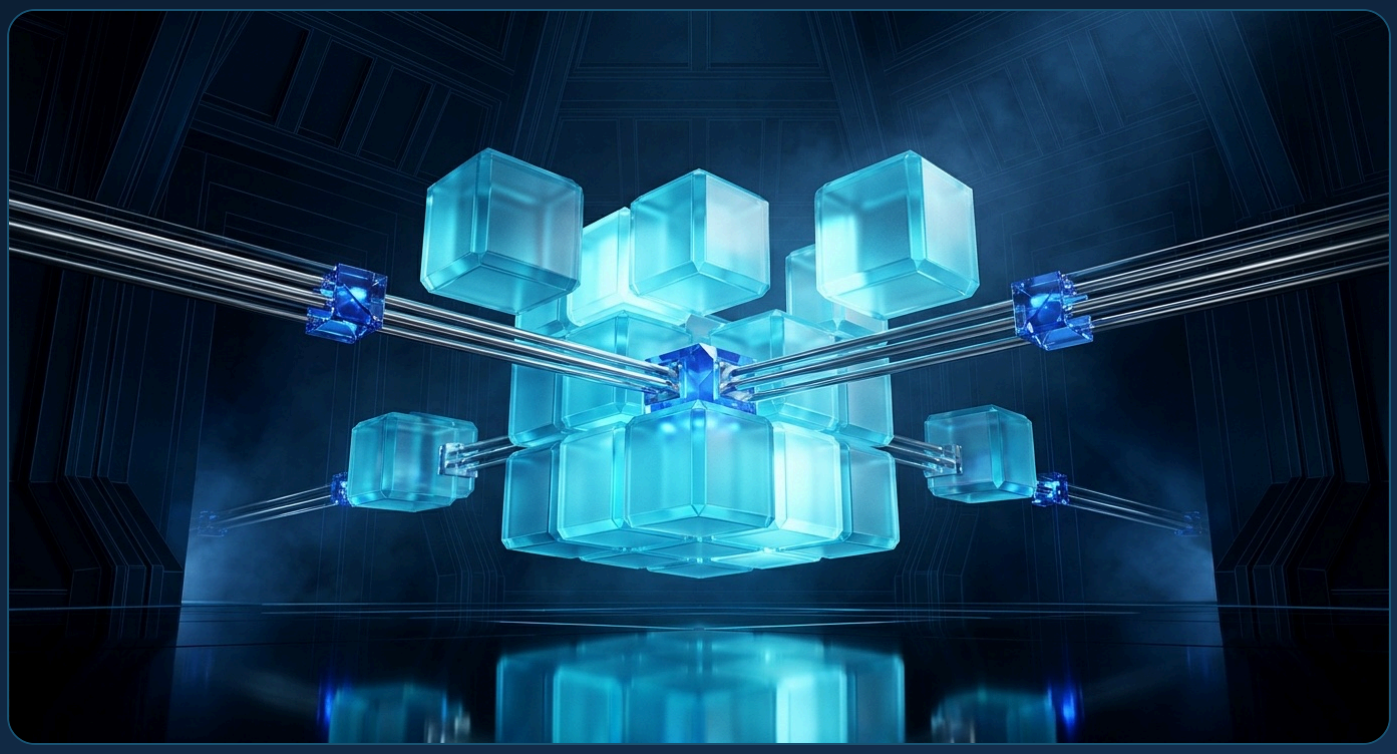
AI AGENT ENGINEERING & ARCHITECTURE • FIELD GUIDE

Structured Outputs & State Machines

The complete 12-page visual handbook to constrained token decoding, Pydantic v2 and Zod v4 contracts, streaming partial JSON, deterministic transition tables, and production approval gates.

CFG & FSM Logit Masking | OpenAI, Claude & Gemini Schemas | Pydantic v2 & Zod v4 | Streaming Partial Objects | Explicit Transition Tables | Idempotency & HITL Gates



Zero-Syntax-Error Decoding
Compile JSON Schemas into token grammars that mask illegal characters to negative infinity before sampling.

Cognition vs. Control
Confine probabilistic model reasoning inside typed step outputs while deterministic tables govern every transition.

Durable Production Loops
Pair bounded semantic repair retries with idempotency ledgers, checkpoints, and human-approval gates.

The Core Mental Model: Separating Cognition from Control

Early agent prototypes let language models steer execution through free-form chat loops and brittle regex parsing. Production systems flip that design: the model acts as a typed cognitive oracle, while a deterministic state machine owns every transition.

Quick Navigation

1. Core Mental Model: Cognition vs. Control p. 2
2. Provider Schema & Engine Matrix p. 3
3. Annotated Schema & FSM Inspector Tour p. 4
4. Constrained Decoding: CFG & FSM Masking p. 5
5. Pydantic v2, Zod v4 & Union Contracts p. 6
6. Troubleshooting & Pre-Flight Checklist p. 12
7. Streaming Partial JSON & Repair Loops p. 7
8. Deterministic Agent State Machines p. 8
9. Five Copy-Paste Production Recipes p. 9
10. Idempotency, Checkpoints & Rollouts p. 10
11. End-to-End Cross-Tool Synergy Pipeline p. 11

C Probabilistic Cognition

Inside a bounded step, the LLM interprets messy inputs and proposes a typed decision. Constrained decoding guarantees 100% valid syntax.

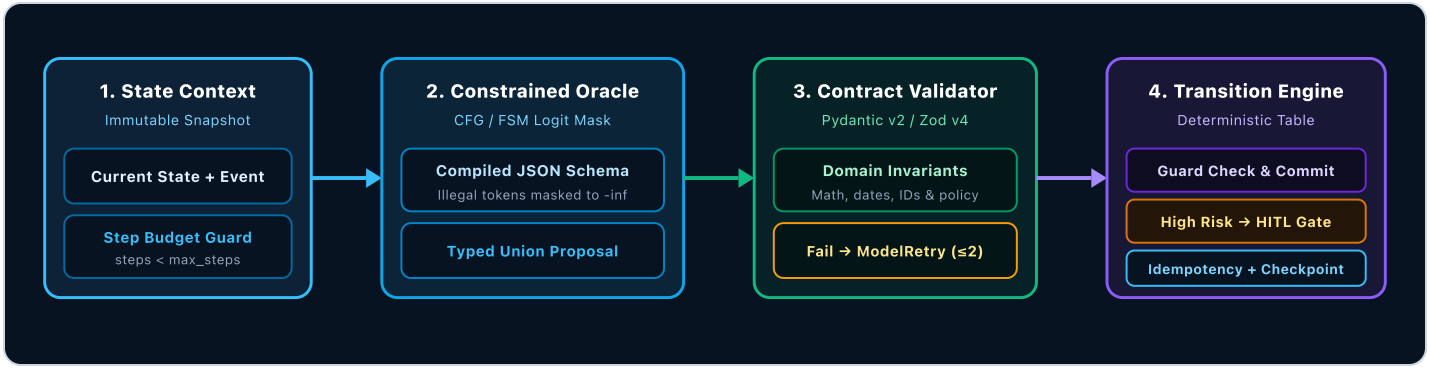
V Semantic Contract Check

Application validators such as Pydantic v2 or Zod v4 check domain invariants, math consistency, and reference integrity before accepting output.

D Deterministic Control

An explicit transition table verifies guards, decrements step budgets, saves durable checkpoints, and runs idempotent side effects.

Architecture: Event → Constrained Oracle → Contract Gate → State Transition



Why This Two-Layer Split Matters

Constrained decoding guarantees that your application receives syntactically valid JSON matching every type and field name. Your deterministic state machine then guarantees that only legal state transitions and authorized tool calls ever touch real customer data.

Provider Schema Support & State Engine Decision Matrix

Every major model API and serving engine supports constrained decoding in 2026, yet each enforces distinct schema rules. Pick the right schema surface and orchestration layer before writing code.

Provider / Engine	API Parameter	Compiler Engine	Strict Rules & Quirks	Best Fit
OpenAI API Responses & Chat	`text.format` or `response_format` (`strict: true`)	Cached CFG token mask	Needs `additionalProperties: false` and every field in `required` (use `| null`).	Strict extraction, typed routing & tool calls.
Anthropic Claude Messages API (GA)	`output_config.format` & `tools[].strict: true`	Compiled grammar (24h cache)	Replaces legacy `structured-outputs-2025-11-13` beta; strict object & tool schemas.	Strict tools plus typed final JSON turn.
Google Gemini google-genai SDK	`responseSchema` or `responseJsonSchema`	Finite-state schema graph	Requires `response_mime_type: "application/json"`; JSON Schema adds `anyOf` & `$ref`.	Key-ordered Pydantic bindings & multimodal JSON.
vLLM & SGLang Self-Hosted Serving	`guided_json` / `response_format`	`XGrammar 2` or `llguidance`	Sub-40µs bitmasks (`XGrammar 2`) or Rust lazy lexers (`llguidance`) at high batch scale.	Private clusters & custom CFG grammars.
Outlines Open-Source Lib	`outlines.generate.json` / regex	FSM / Regex Automaton	Pioneering FSM compiler; heavier upfront mask build on giant vocabularies.	Local models & strict regex strings.

Deterministic State Machine & Contract Layer Comparison

Tool / Runtime	Language	Core Superpower	When to Choose
Pydantic v2 + PydanticAI	Python 3.10+	Rust-fast `model_json_schema()` and automatic `ModelRetry` error feedback loops.	Python services, ETL pipelines & validation-heavy agents.
Zod v4 + Vercel AI SDK	TypeScript / JS	Built-in `z.toJSONSchema()` (replaces old plugin) plus streaming `DeepPartial` objects.	Full-stack TypeScript, React streaming UIs & edge APIs.
LangGraph	Python / TS	Cyclical agent graphs, Postgres checkpoints & native `interrupt()` pauses.	Multi-step reasoning loops needing pause and resume.
Temporal + XState	Polyglot / TS	Append-only durable replay across crashes paired with formal statechart guards.	Multi-day human approvals & mission-critical writes.

Syntax Is Not Semantics: The Most Common Beginner Trap

Token logit masking guarantees balanced braces and valid types, yet cannot verify whether a refund exceeds an invoice balance or whether extracted dates are valid. Always pair grammar constraints with application-level semantic validators.

Annotated Inspector: Schema & State-Transition HUD

Debuggable agent platforms combine a live grammar bitmask trace with an explicit finite-state inspector so engineers can see why a token passed, why a semantic repair ran, and which guard halted execution.



Control Legend & Inspection Points

1 **Active State, Step Budget & Key:** Tracks the exact enum state, hard step ceiling preventing infinite loops, and deterministic idempotency key.

2 **Grammar Cache & Bitmask Meter:** Confirms the compiled CFG or FSM is cached in memory and shows illegal token suppression at each step.

3 **Semantic Repair Trace:** Logs domain validation failures caught by Pydantic or Zod and the targeted field hint fed into bounded retries.

4 **Discriminated-Union Output:** Displays the strictly typed payload whose literal `kind` tag routes the state machine deterministically.

5 **Transition Guard Evaluator:** Shows the exact lookup-table rule and boolean guard condition evaluated by host code, never by prompt text.

6 **Durable Checkpoint Store:** Records the serialized machine state after every transition so crashed workers replay or fork cleanly.

7 **Cryptographic Approval Gate:** Suspends execution before irreversible mutations until a human signs the exact proposal hash.

How CFG & FSM Logit Masking Guarantee Zero Syntax Errors

Prompts cannot enforce formal syntax reliably across thousands of runs. Constrained decoding solves formatting at the token-sampling layer by masking illegal vocabulary choices before a token is ever picked.

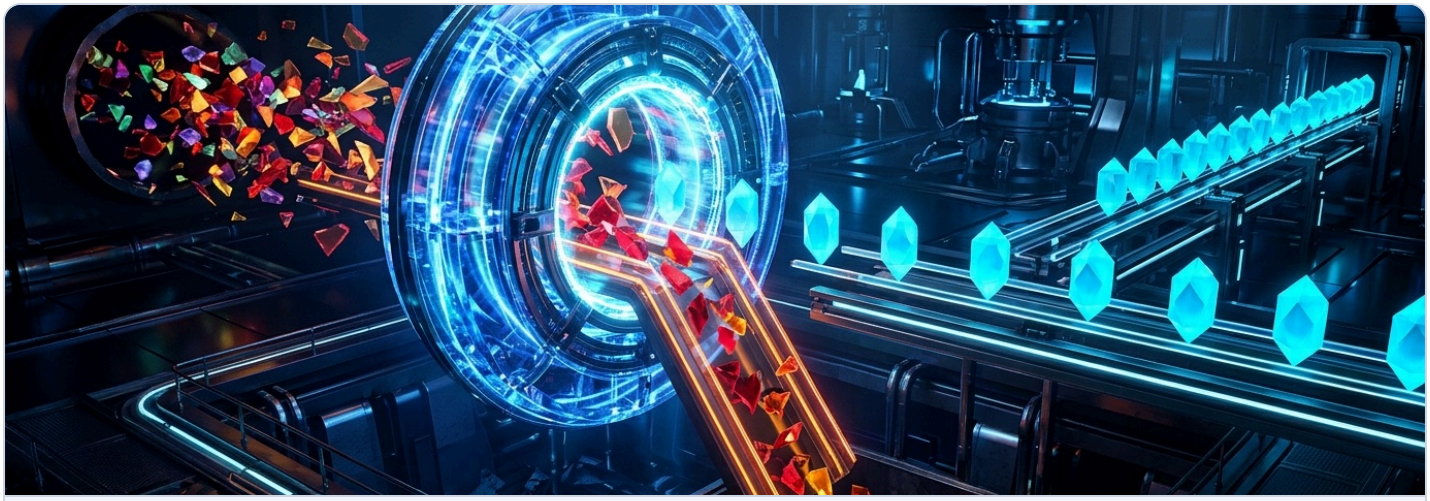


Figure 5.1 • Token-Level Grammar Filtering: Candidate vocabulary tokens pass through the active grammar state; invalid characters and keys are deflected to zero probability while schema-compliant tokens lock cleanly onto the output track.

1 Schema-to-Grammar Compile

Your JSON Schema compiles into a Finite State Machine (regular patterns) or Context-Free Grammar (recursive objects) and caches by schema hash.

2 Microsecond Bitmasks

Engines such as `XGrammar 2` and `llguidance` pre-classify context-independent tokens so per-step bitmasks compute in under 40 microseconds.

3 Logit -Infinity Masking

Disallowed tokens receive a `-inf` logit bias prior to softmax. Even with creative temperatures, malformed JSON becomes mathematically impossible.

API Anatomy: OpenAI `text.format` & Anthropic `output_config`

`Schema Wrapper` (`response_format` / `output_config.format`) + `Reasoning First` (`reasoning_summary` key) + `Strict Guard` (`additionalProperties: false`) + `Typed Decision` (`enum` / union)

```
// OpenAI Responses API (text.format) & Anthropic Messages API (output_config.format)
{
  "text": { "format": { "type": "json_schema", "name": "triage_step", "strict": true,
    "schema": {
      "type": "object",
      "properties": {
        "reasoning_summary": { "type": "string" }, // Keep reasoning before decision enum!
        "route": { "type": "string", "enum": ["search_kb", "issue_refund", "escalate_human"] },
        "ticket_tag": { "type": ["string", "null"] }
      },
      "required": ["reasoning_summary", "route", "ticket_tag"],
      "additionalProperties": false
    }
  }}
}
```

Pydantic v2, Zod v4 & Discriminated-Union Contracts

Maintain a single source of truth for schemas, runtime checks, and compiler types. Modern Zod v4 and Pydantic v2 export strict JSON Schemas natively while validating domain math and discriminated-union tags.

TS Zod v4 Native `z.toJSONSchema()`

Zod v4 replaces the deprecated `zod-to-json-schema` package with built-in `z.toJSONSchema(schema)`, targeting `draft-2020-12` or `draft-07` cleanly.

Use `z.strictObject()` and `z.discriminatedUnion("kind", [...])` so every variant automatically emits closed objects.

PY Pydantic v2 `model_json_schema()`

Powered by a Rust core, Pydantic v2 compiles complex union models via `Model.model_json_schema()` and powers `google-genai` and OpenAI SDKs directly.

Combine `ConfigDict(extra="forbid")` with `@model_validator` to enforce cross-field business rules after decoding.

Union Formula: `Discriminator Literal` (`kind`) + `Variant Fields` (`tool_name` , `args`) + `Nullable Optional` (`.nullable()`) + `Closed Object` (`extra="forbid"`)

```typescript
// TypeScript (Zod v4): Single source of truth for LLM Schema + Switch Exhaustiveness
import * as z from "zod";

const StepDecision = z.discriminatedUnion("kind", [
  z.strictObject({ kind: z.literal("call_tool"), tool: z.enum(["lookup_order", "refund"]), orderId: z.string() }),
  z.strictObject({ kind: z.literal("ask_user"), question: z.string(), missingField: z.string().nullable() }),
  z.strictObject({ kind: z.literal("complete"), summary: z.string() }),
]);

export const providerSchema = z.toJSONSchema(StepDecision, { target: "draft-2020-12" });
```

```python
# Python (Pydantic v2 + google-genai): Structural schema + semantic cross-field invariant
from typing import Literal
from pydantic import BaseModel, ConfigDict, Field, model_validator

class RefundProposal(BaseModel):
    model_config = ConfigDict(extra="forbid")
    kind: Literal["refund_proposal"]
    invoice_cents: int = Field(gt=0)
    refund_cents: int = Field(gt=0)

    @model_validator(mode="after")
    def verify_cap(self) -> "RefundProposal":
        if self.refund_cents > self.invoice_cents:
            raise ValueError(f"refund_cents ({self.refund_cents}) exceeds invoice_cents ({self.invoice_cents})")
        return self
```

Cross-Provider Strict Schema Portability Checklist

To make one schema work smoothly across OpenAI, Anthropic, and Gemini: avoid optional properties (use `.nullable()` instead of `.optional()`), set `additionalProperties: false` on every nested object, and keep complex regex lookarounds in your Pydantic or Zod post-validator rather than in the token grammar.

Streaming Partial JSON & Bounded Validation-Repair Loops

Waiting for a complete JSON payload makes interactive apps feel frozen. Incremental parsers render partial objects live as tokens arrive, while capped validation-feedback loops self-heal semantic mistakes cleanly.

1 Progressive Partial JSON Streaming

OpenAI streams raw structured tokens, while Anthropic emits `input_json_delta` events carrying `partial_json` slices. Libraries such as Vercel AI SDK `Output.object` and Python `jiter` auto-close unfinished brackets mid-stream.

Key rule: Expose streamed chunks as `DeepPartial<T>` for UI rendering only; run strict validation and trigger state transitions only after the final token arrives.

2 Bounded Semantic Repair Loops

When PydanticAI raises `ModelRetry` or a Zod validator rejects a value, inject the exact field path and violation message back into the conversation for a targeted correction turn.

Keep error payloads surgical: cite only the failing JSON pointer (for example `/refund_cents`) and the invariant rule, not an entire raw stack trace.

```
// TypeScript (Vercel AI SDK): Stream typed DeepPartial objects safely before final commit
import { streamText, Output } from "ai";
import * as z from "zod";

const PlanSchema = z.strictObject({
  headline: z.string(),
  steps: z.array(z.strictObject({ title: z.string(), risk: z.enum(["low", "high"]) })),
});

const result = streamText({
  model: providerModel,
  output: Output.object({ schema: PlanSchema }),
  prompt: "Draft a database index migration plan.",
});

for await (const partial of result.partialOutputStream) {
  renderLiveDraftCard(partial); // Safe DeepPartial: fields populate progressively as tokens arrive
}
const finalVerifiedPlan = await result.output; // Fully validated against PlanSchema on stream end
```

The 4-Stage Bounded Repair Protocol

Attempt Stage	Trigger Condition	Runtime Action	Next State
Initial Call (0)	Constrained decoding enabled	Stream tokens; verify schema + semantic rules on EOF.	`TRANSITION_OK` or Retry 1
Repair Try 1	Semantic check fails (e.g., bad ID)	Append concise JSON-pointer error via `ModelRetry`.	`TRANSITION_OK` or Retry 2
Repair Try 2	Second invariant mismatch	Lower temperature to `0.0`; include allowed enum/ID hints.	`TRANSITION_OK` or Escalate
Circuit Open (3)	`retry_count >= max_retries`	Halt generation immediately; preserve failed trace in DLQ.	`DEAD_LETTER` / Human

Why You Must Cap Repair Loops at 2 or 3 Attempts

Unbounded self-healing loops burn tokens rapidly when a model lacks a required fact in context and keeps guessing invalid IDs. Treating repair exhaustion as a deterministic transition to `DEAD_LETTER` turns silent hangs into observable alerts.

Explicit Transition Tables, Loop Breakers & Approval Gates

An agent loop is safest when every state, allowed event, guard condition, and exit ceiling lives in an explicit transition table. The language model proposes events; deterministic code decides whether they execute.

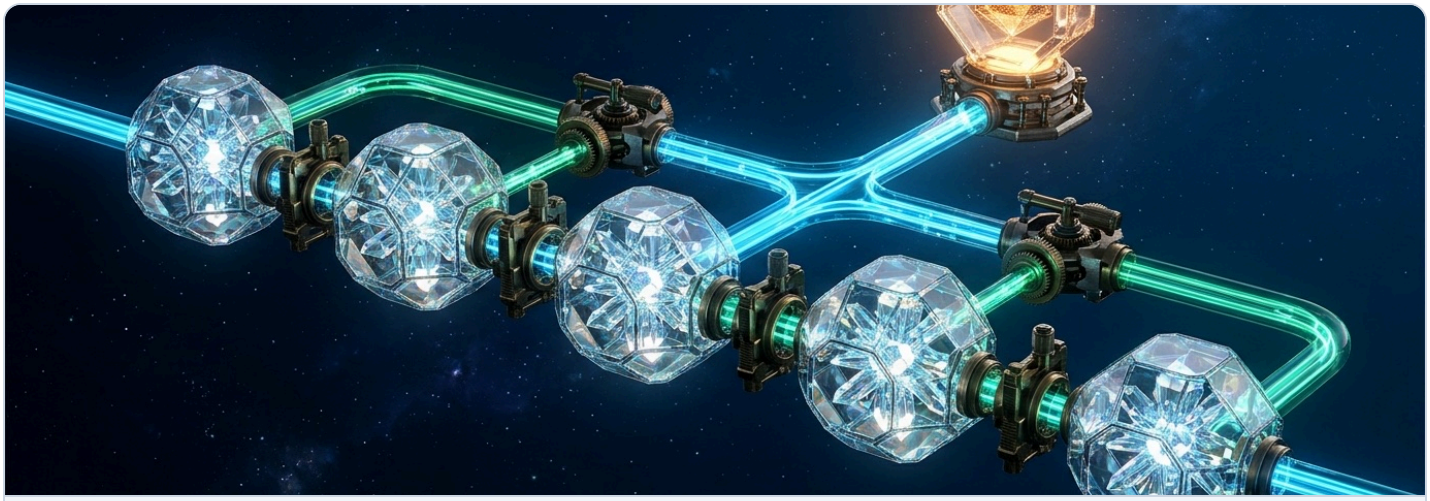


Figure 8.1 • Explicit Statechart Topology: Execution flows strictly through locked chambers along verified fiber bridges, routing high-consequence mutations up through the amber human-approval pedestal before final commit.

Production Agent Transition Table

Current State	Typed Event Trigger	Deterministic Programmatic Guard	Target State
`IDLE`	`EV_TASK_SUBMITTED`	Input sanitized & `step_budget > 0`	`PLANNING`
`PLANNING`	`EV_PROPOSE_READ`	Read-only tool allowlisted & args valid	`EXECUTING_READ`
`PLANNING`	`EV_PROPOSE_WRITE`	Mutation flag true or `amount > threshold`	`AWAITING_APPROVAL`
`AWAITING_APPROVAL`	`EV_APPROVE`	Signed `approval_token` matches diff hash	`EXECUTING_WRITE`
`AWAITING_APPROVAL`	`EV_REJECT`	Reviewer feedback attached & `steps < max`	`PLANNING`
`ANY_ACTIVE`	`EV_BUDGET_EXHAUSTED`	`steps >= max_steps` or wall timeout hit	`DEAD_LETTER`

B Max-Step Loop Breakers

Enforce a hard global `max_steps` counter plus per-tool repeat limits so an agent cannot call the same failing search tool endlessly.

G Cryptographic HITL Gates

Keep write credentials out of LLM reach. Freeze a serialized diff, pause via `interrupt()`, and require a version-locked signature to resume.

T Illegal Pair Rejection

Any `(state, event)` missing from `TRANSITION_TABLE` throws immediately before tool dispatch, proving safety even under adversarial inputs.

Five Copy-Paste Production Recipes

Proven implementation blueprints covering strict extraction, discriminated-union routing, streaming partial interfaces, self-healing validators, and durable human-approved tool execution.

Recipe & Goal	Stack	Core Pattern	Deterministic Guard
1. Strict Extraction Oracle Parse invoices or tickets cleanly	OpenAI `text.format` or Gemini `responseJsonSchema`	Compile Pydantic/Zod contract with `strict: true` and nullable optional keys.	`additionalProperties: false` plus post-parse math check.
2. Typed Intent Router Eliminate fragile if/else parsing	Claude `output_config` + TypeScript switch	Use `z.discriminatedUnion("kind", [...])` for mutually exclusive branches.	Compiler exhaustiveness check (`assertNever`) on `kind`.
3. Live Streaming Form Populate UI cards token by token	Vercel AI SDK `streamText` + `Output.object`	Iterate `partialOutputStream` into UI state; await final `output`.	Disable submit buttons until final strict validation resolves.
4. Self-Healing SQL / API Fix bad IDs or constraints	PydanticAI `ModelRetry` / Instructor	Validate foreign keys inside `@model_validator`; return concise pointer hint.	Hard cap `retries=2` before routing to `DEAD_LETTER`.
5. Gated Mutation Agent Approve refunds, deploys, emails	LangGraph `interrupt()` + Postgres	Halt inside policy node, write checkpoint, await signed UI resume webhook.	Bind `approval_token` to SHA-256 hash of tool payload.

Copy-Paste Python Deterministic Transition Runner with Loop Breaker

```
from dataclasses import dataclass

TRANSITIONS = {
    ("IDLE", "EV_START"): "PLANNING",
    ("PLANNING", "EV_CALL_READ"): "EXECUTING_READ",
    ("EXECUTING_READ", "EV_OBS_READY"): "PLANNING",
    ("PLANNING", "EV_PROPOSE_WRITE"): "AWAITING_APPROVAL",
    ("AWAITING_APPROVAL", "EV_APPROVE"): "EXECUTING_WRITE",
    ("AWAITING_APPROVAL", "EV_REJECT"): "PLANNING",
    ("EXECUTING_WRITE", "EV_COMMITTED"): "COMPLETED",
}

@dataclass
class MachineContext:
    state: str = "IDLE"
    steps: int = 0
    max_steps: int = 12

def step_machine(ctx: MachineContext, event: str, guard_ok: bool = True) -> str:
    if ctx.steps >= ctx.max_steps:
        ctx.state = "DEAD_LETTER"
        return ctx.state
    nxt = TRANSITIONS.get((ctx.state, event))
    if nxt is None or not guard_ok:
        raise ValueError(f"Blocked transition from {ctx.state} on {event}")
    ctx.steps += 1
    ctx.state = nxt
    save_checkpoint(ctx)  # Persist every step to Postgres or Temporal
    return ctx.state
```

Idempotency, Checkpointing & Dead-Letter Transitions

Structured outputs protect what enters your state machine; idempotency keys, durable checkpoints, and dead-letter queues protect what happens when workers crash mid-flight or upstream APIs time out.

1 Deterministic Idempotency Keys

If an agent sends a payment or creates a ticket and the HTTP response drops, a naive retry executes the side effect twice. Derive an `idempotency_key` from `sha256(run_id + step_index + action + payload_hash)`.

Check a durable execution ledger before firing external APIs; if that key already recorded `SUCCEEDED`, return the cached receipt immediately.

2 Durable Checkpoints & Replay

Commit state snapshots (enum state, message cursor, remaining `max_steps`, and pending tool intent) to Postgres or Temporal event histories after every transition.

Long-lived human approvals can sleep for hours or days at zero compute cost until an external webhook signals resumption.

D Dead-Letter Transitions

Route exhausted repairs or step-budget overruns into `DEAD_LETTER` with full inputs, schema version, and validator diagnostics for human triage.

V Schema Versioning

Stamp every checkpoint with `schema_version`. When deploying breaking union changes, run pure upcasters before replaying paused runs.

R Shadow Canary Rollouts

Run new schema versions in shadow audit mode first to measure repair rates, grammar compile latency, and enum drift before gating traffic.

Idempotent Side-Effect Ledger Guard (TypeScript)

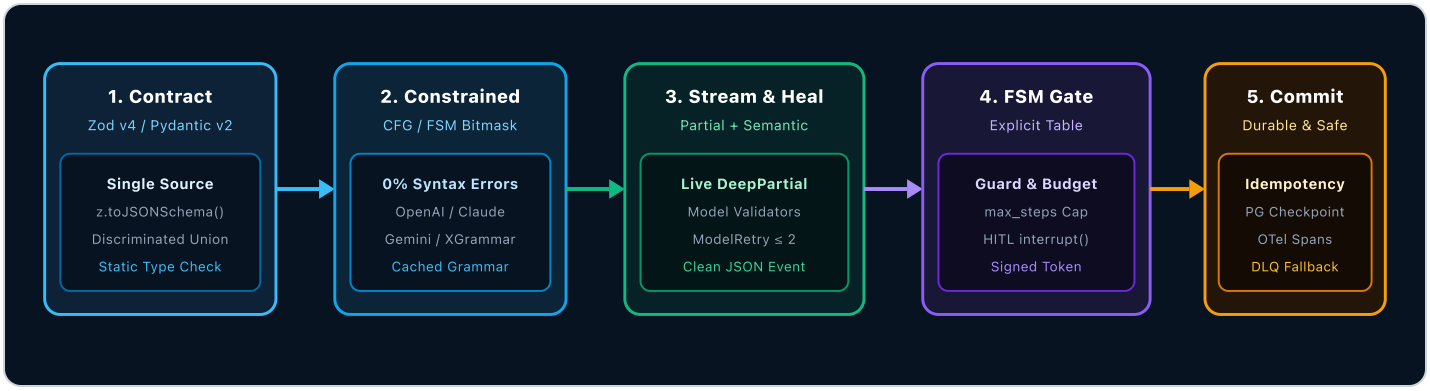
```
export async function executeIdempotentTool(runId: string, step: number, proposal: ToolCall) {
  const key = sha256(runId + ":" + step + ":" + proposal.name + ":" + canonicalJson(proposal.args));
  const existing = await ledgerRepo.findByKey(key);
  if (existing?.status === "SUCCEEDED") {
    return existing.cachedReceipt; // Short-circuit duplicate side effect safely
  }
  await ledgerRepo.markInFlight(key, proposal);
  const receipt = await dispatchExternalTool(proposal, { idempotencyKey: key });
  await ledgerRepo.markSucceeded(key, receipt);
  return receipt;
}
```

Separate Speculative Reads from Commit Writes

Let agents run read-only queries freely inside `PLANNING` with fast retries, yet funnel every state-mutating tool through an idempotent `EXECUTING_WRITE` state backed by a ledger and an optional human gate.

End-to-End Synergy: Typed Ingress to Durable Execution

When Zod v4 or Pydantic v2 contracts drive both provider token masking and your statechart transitions, every layer reinforces the next—turning stochastic text generation into verifiable software engineering.



1 Streaming UI Sync: Vercel AI SDK + XState

Pair Vercel AI SDK `Output.object` streams with client-side XState statecharts so UI panels transition smoothly from `streaming_draft` to `awaiting_confirmation` without race conditions, duplicate button clicks, or half-parsed JSON glitches.

2 Durable Backend: LangGraph + Temporal + OTel

Use LangGraph to manage cyclical reasoning nodes while Temporal persists durable activity replays across server restarts. Emit OpenTelemetry spans for `schema.compile`, `repair.retry`, `fsm.transition`, and `gate.approve`.

Observability & Governance Signals Across the Pipeline

C Grammar Cache Hit Rate

Track first-request compile latency versus cached CFG hits to catch accidental dynamic schema generation per turn.

R Repair Loop Frequency

Alert when `ModelRetry` spikes above 5% on a route—an immediate signal that field descriptions or prompts need clarity.

A Transition Audit Trail

Record immutable `(state, event, guard_result, actor)` tuples for compliance reviews and deterministic post-incident replay.

Single Source of Truth Design Principle

Never hand-maintain separate JSON Schema files, TypeScript or Python dataclasses, and prompt descriptions. Author one canonical Zod v4 or Pydantic v2 module with descriptive field metadata so every boundary stays strictly synchronized across deploys.

Troubleshooting Clinic & Production Pre-Flight Checklist

Use this field clinic to diagnose the seven most frequent structured-output and state-machine failures in seconds, followed by six readiness gates every agent must clear before production traffic.

Symptom & Error	Root Cause	10-Second Production Fix
`400: Invalid strict schema`	Object missing `additionalProperties: false` or optional key omitted from `required`.	Use `z.strictObject()` or `extra="forbid"`; declare optional fields as `["string", "null"]`.
First-call latency spike	Cold CFG/FSM compilation for large or deeply nested schemas.	Warm common schemas on server boot and reuse identical schema strings to hit provider caches.
Accuracy drops in strict mode	Schema forces final enum classification before any reasoning tokens.	Add a `reasoning_summary` field ahead of the `decision` field in property declaration order.
Infinite tool or repair loops	Missing global `max_steps` cap or ambiguous repair error prompts.	Enforce hard step ceilings, cap `ModelRetry` at 2, and inject exact JSON-pointer paths.
Duplicate writes or charges	Worker retried timed-out tool execution without ledger lookup.	Compute deterministic `idempotency_key` from `run_id + step` and check ledger before calling APIs.
Stale HITL approval resume	Proposal changed after human inspection or old resume event replayed.	Bind cryptographic `approval_token` to SHA-256 of proposal diff and checkpoint version.
Streaming UI throws JSON error	Calling strict `.parse()` on incomplete partial JSON chunks.	Bind UI directly to `DeepPartial<T>`; run full schema validator only after final stream completion.

6-Point Pre-Flight Production Readiness Checklist

☐ **1. Strict Mode Verified:** Confirm every provider schema sets `additionalProperties: false`, places all fields in `required`, and passes provider dry-run validation.

☐ **2. Reasoning Order Tested:** Verify complex routing schemas place scratchpad reasoning keys ahead of strict classification enums so quality stays high.

☐ **3. Bounded Loops Active:** Test that hitting `max_steps` or exhausting 2 semantic retries transitions cleanly into `DEAD_LETTER` rather than hanging.

☐ **4. Idempotency Ledger Proven:** Simulate a network timeout during `EXECUTING_WRITE` and verify the replay returns the cached receipt with zero duplicate writes.

☐ **5. HITL Token Bound to Hash:** Verify modifying any parameter of a paused proposal invalidates prior approval signatures and forces fresh review.

☐ **6. Checkpoint Migration Safe:** Confirm persisted checkpoints carry `schema_version` tags and replay smoothly across rolling deploys.