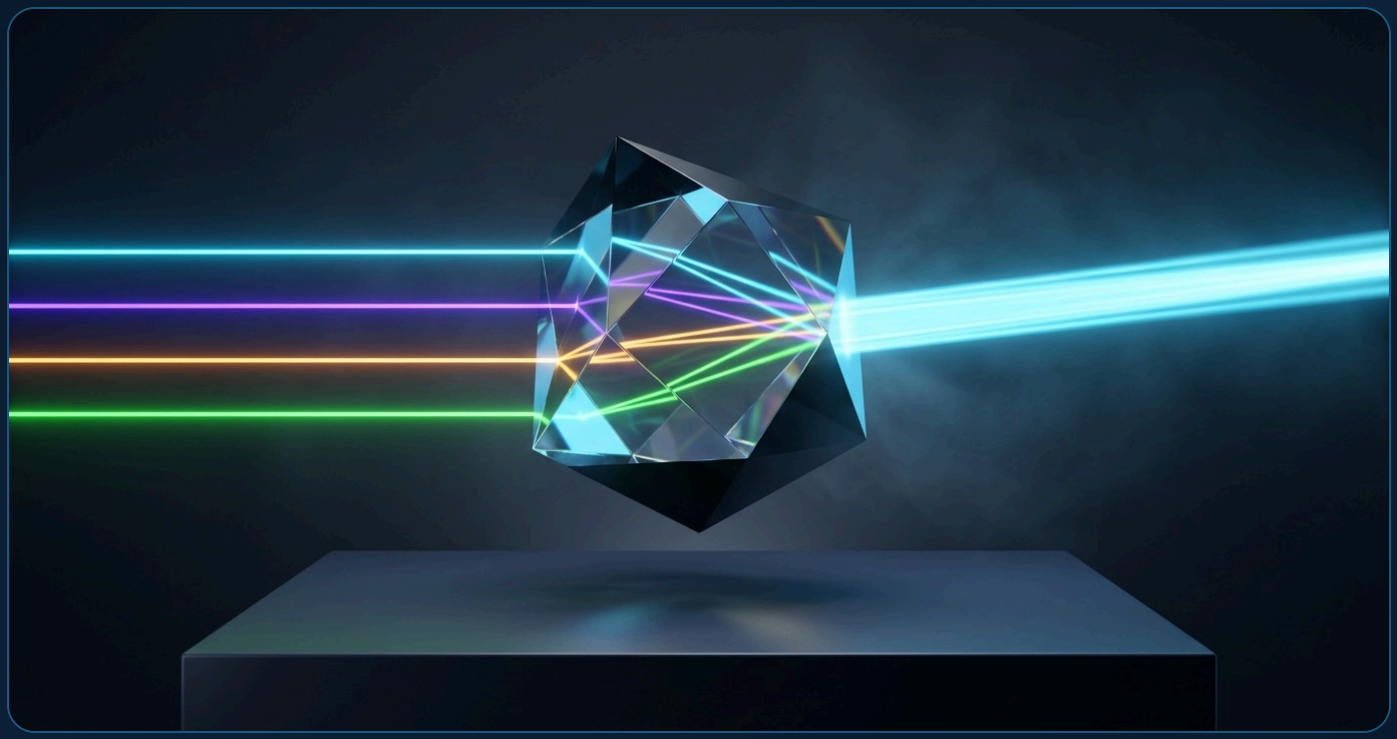
VERCEL & FULL-STACK AI APPLICATIONS • FIELD GUIDE

Vercel AI SDK 7 & AI Gateway

The complete visual handbook for building resilient, streaming TypeScript AI applications on Fluid Compute—uniting `ToolLoopAgent`, typed `Output.object()`, React `message.parts`, and multi-provider AI Gateway failover behind zero-config OIDC authentication.

Node 24 Fluid Compute | AI Gateway Routing | `ToolLoopAgent` | `Output.object()` | `useChat` & `message.parts` | Virtual Models & Budgets



One Unified TypeScript API

Call any foundation model via plain `provider/model` strings with shared streaming, tools, and structured schemas.

Automatic Provider Failover

Route around outages and rate limits across upstream hosts using `VERCEL_OIDC_TOKEN`, Virtual Models, and spend budgets.

Fluid Compute Streaming

Stream tokens and typed UI parts natively on Node.js 24 LTS with 300s default timeouts and Active CPU pricing.

One Streaming TypeScript Runtime, Multi-Provider Intelligence

AI SDK 7 and Vercel AI Gateway split responsibilities cleanly: your React UI and Next.js server speak one strongly typed streaming protocol on Node.js Fluid Compute, while AI Gateway handles provider credentials, automatic failover, and spend governance at the network boundary.

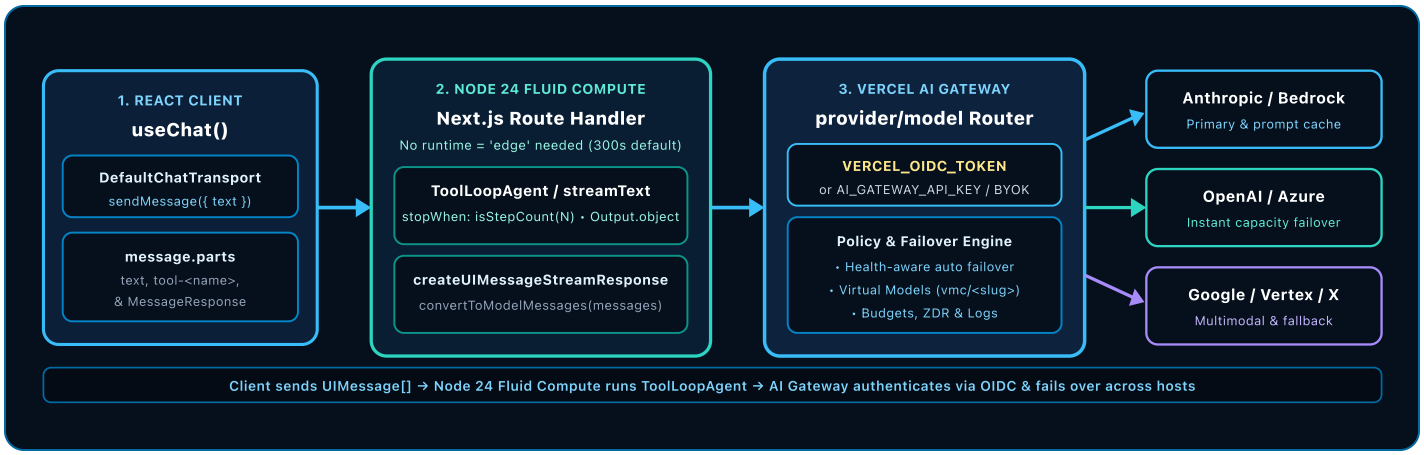
Quick Navigation

1. Streaming Architecture & Mental Model p. 2
2. Fluid Compute Setup & Myths vs. Truths p. 3
3. Annotated DevTools & Gateway Inspector p. 4
4. Vercel AI Gateway Mastery & Decisions p. 5
5. `ToolLoopAgent` & `Output.object()` p. 6
6. Troubleshooting Clinic & Pre-Flight p. 12
7. Streaming React & Multimodal UIs p. 7
8. `vercel.ts`, Embeddings & Coding Agents p. 8
9. 5 Real-World Copy-Paste Recipes p. 9
10. Budgets, Caching, Security & Logs p. 10
11. Cross-Tool Vercel Synergy Pipeline p. 11
Golden Rule: Verify Against `node_modules/ai/docs/` All

Core Mental Model: Decouple App Logic from Model Vendors

Never hardcode vendor-specific SDKs inside your application routes. Passing a plain `provider/model` string (or a centrally managed `vmc/<slug>` Virtual Model) into `ToolLoopAgent` or `streamText` routes traffic automatically through AI Gateway. Your app stays portable across models while Vercel Fluid Compute streams Server-Sent Events directly from Node.js 24 LTS without artificial Edge runtime constraints.

End-to-End Streaming Architecture on Node.js 24 Fluid Compute



Mental Model Takeaway: Keep App Code Agnostic, Keep Routing Central

Install only `ai` (plus `@ai-sdk/react` for browser hooks) to get started. Because plain `provider/model` strings speak directly to AI Gateway, you gain multi-provider redundancy, OIDC security, and unified cost logs without installing five vendor packages.

Fluid Compute Setup & Modern AI SDK 7 Truths

AI SDK 7 requires Node.js 22+ and native ESM, pairing seamlessly with Vercel's default **Node.js 24 LTS** runtime on Fluid Compute. Forget legacy Edge workarounds and outdated v4–v6 habits before writing your first route.

1 Project Setup & Version-Matched Docs

```
# Install AI SDK 7 + React hooks + Zod schemas (Node 22+ / ESM)
pnpm add ai @ai-sdk/react zod

# Link project & pull 12-hour VERCEL_OIDC_TOKEN locally
vercel link && vercel env pull

# Read version-matched docs bundled right inside npm!
ls node_modules/ai/docs/
```

Never trust stale training memory: `node_modules/ai/docs/` ships exact APIs for your installed version. Discover live model IDs with `curl -fsSL https://ai-gateway.vercel.sh/v1/models`.

2 Node.js 24 LTS Fluid Compute Advantages

- **Streams Natively Without Edge:** Standard Node.js routes stream SSE and AI chunks cleanly—never add deprecated `runtime = 'edge'`.
- **300s Default Timeout:** Every plan (including Hobby) gets `300s` default execution (up to `800s` Pro/Enterprise, `1800s` extended beta).
- **Active CPU Pricing:** Fluid Compute reuses instances concurrently and bills only active CPU compute—idle time waiting for tokens costs zero CPU.
- **Typed Project Config:** Use `vercel.ts` via `@vercel/config/v1` instead of hand-editing JSON.

Outdated LLM Myths vs. Modern AI SDK 7 + Fluid Compute Truths (Do Not Use v4–v6 Habits)

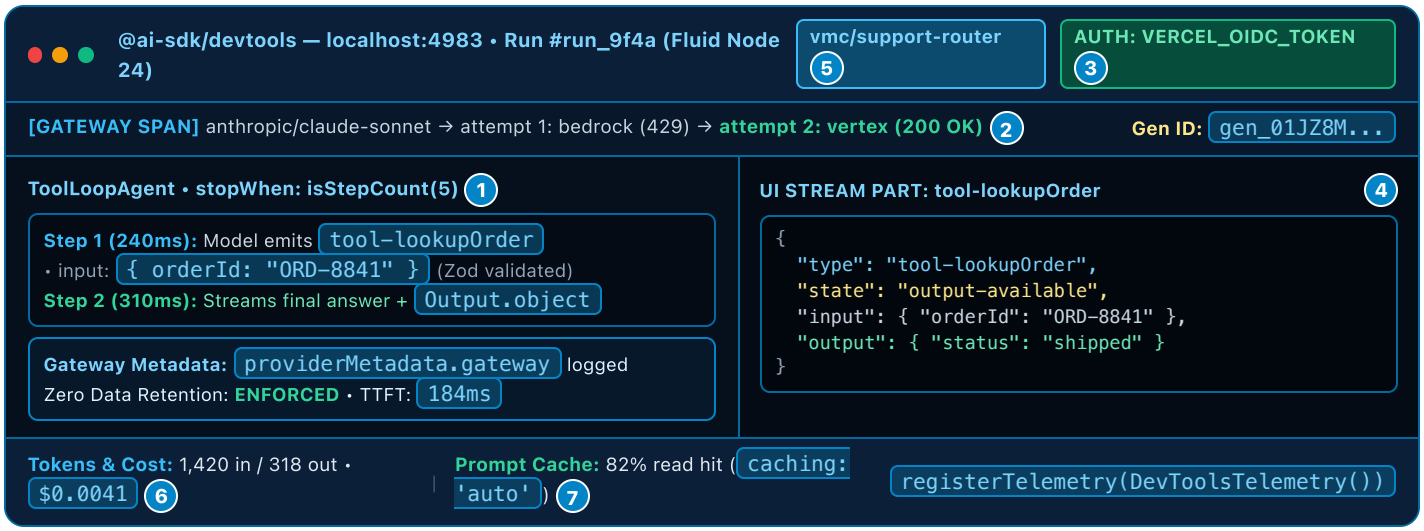
Domain	Outdated Myth / Obsolete v4–v6 Pattern	Modern AI SDK 7 + Fluid Compute Truth
Streaming Runtime	Adding `export const runtime = 'edge'` so SSE and AI token streaming work.	**Drop Edge completely:** Node.js 24 LTS on Fluid Compute streams cleanly with full Node APIs and 300s+ timeouts.
Agent Abstraction	`Experimental_Agent` or hand-rolled `while` loops around `generateText`.	Use `ToolLoopAgent` from `ai` with `agent.generate()` and `agent.stream()`.
Loop Step Limits	`maxSteps: 5` (v4) or `stepCountIs(5)` (v5–v6).	Use `stopWhen: isStepCount(5)` imported directly from `ai`.
Structured Output	`generateObject()` or `streamObject()` (both deprecated since v6).	Use `generateText` or `streamText` with `output: Output.object({ schema })`.
Tool Definitions	`tool({ parameters: z.object(...) })`.	MCP-aligned `tool({ inputSchema, outputSchema, execute })`.
Message Types & Streams	`CoreMessage` and `toDataStreamResponse()`.	`ModelMessage` + `convertToModelMessages()`; `createUIMessageStreamResponse({ stream: result.toUIMessageStream() })`.
Client Chat Hook	`useChat({ api })`, `handleSubmit`, `isLoading`, and `message.content`.	`useChat({ transport: new DefaultChatTransport({ api }) })`, `sendMessage({ text })`, `status`, and `message.parts`.

Live Catalog Rule: Fetch Before Selecting Any Model Slug

Models ship and retire rapidly. Run `curl -fsSL https://ai-gateway.vercel.sh/v1/models | jq -r '.data[].id'` (or `vercel ai-gateway models list`) to inspect exact `provider/model` IDs, context windows, and `zdr` flags rather than guessing from memory.

Annotated Tour: `@ai-sdk/devtools` & Gateway Inspector

Pair local `npx @ai-sdk/devtools` (`http://localhost:4983`) with AI Gateway telemetry to trace every multi-step `ToolLoopAgent` turn, provider failover hop, typed UI part, and prompt cache hit in real time.



Inspector Legend: The 7 Live Diagnostic Touchpoints

1. **Multi-Step `ToolLoopAgent` Timeline:** Shows every reasoning step, tool invocation, and `stopWhen: isStepCount(N)` boundary in chronological order.
2. **`provider/model` Failover Span:** Reveals upstream attempts (such as a `429` on Bedrock automatically failing over to Vertex) without app retries.
3. **`VERCEL_OIDC_TOKEN` Auth Badge:** Confirms zero-secret OIDC authentication on Vercel deployments or locally via `vercel env pull`.
4. **Typed `tool-<toolName>` Part View:** Inspects exact AI SDK 7 `UIMessage` parts (replacing legacy `tool-invocation`) consumed by `useChat`.
5. **Virtual Model Routing Pill:** Identifies centrally managed `vmc/<slug>` slugs that inject server-side model policy and fallbacks.
6. **Token & Generation Cost Telemetry:** Captures exact input/output tokens, TTFT, and `providerMetadata.gateway.generationId` per request.
7. **Prompt Cache Hit Ratio:** Tracks cached prefix reads from `caching: 'auto'` (for Anthropic/MiniMax explicit markers) and implicit caching on OpenAI/Google so multi-turn agent loops stay fast and inexpensive.

Vercel AI Gateway Mastery: Routing, OIDC & Decisions

AI Gateway turns any `provider/model` string into a fault-tolerant, observable endpoint with zero markup on provider token rates, automatic health-based failover, and unified CLI management.

1. Plain String Routing, Ordering & Typed Decisions

```
import { generateText, experimental_decide as decide } from
'ai';

// Zero-config gateway routing with fallback models & ZDR
const { text } = await generateText({
  model: 'anthropic/claude-sonnet-4.6',
  prompt: 'Summarize incident metrics.',
  providerOptions: {
    gateway: {
      order: ['bedrock', 'anthropic'], // Prefer Bedrock;
keep others
      models: ['openai/gpt-5.4'],       // Cross-provider
fallback
      zeroDataRetention: true,
    },
  },
});

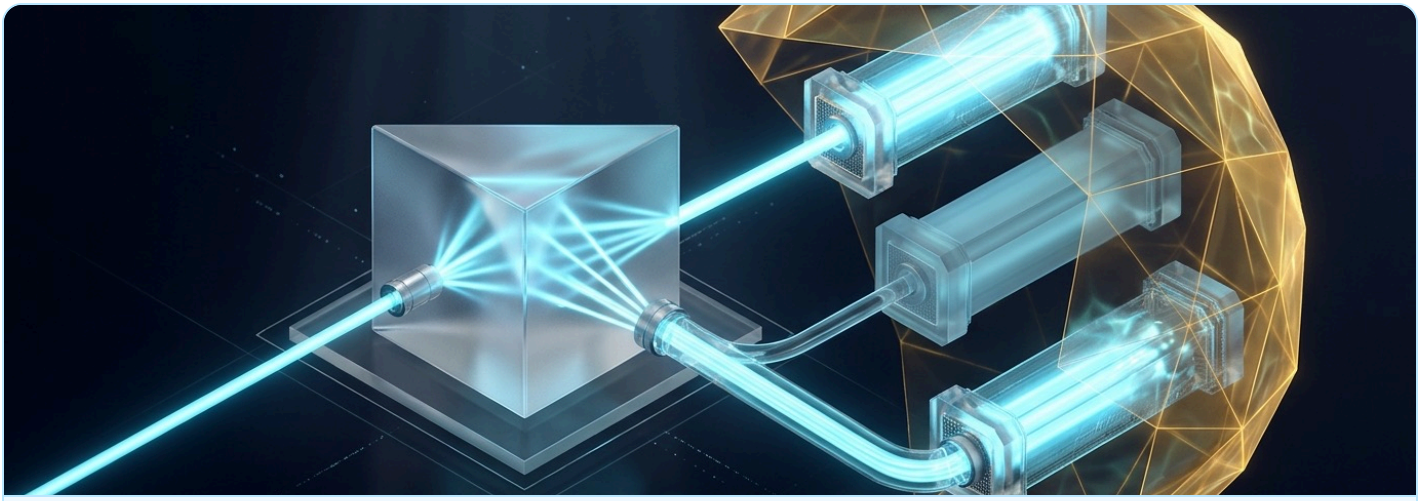
// Typed Decisions (replaces deprecated evaluate / POST
/v1/evaluate)
const verdict = await decide({
  model: 'typesafe-ai/jev',
  state: text,
  questions: { resolved: { type: 'boolean', instructions:
'Fixed?' } },
});
```

2. Authentication Tiers & `vercel ai-gateway` CLI

1. **`VERCEL_OIDC_TOKEN`:** Automatic on Vercel deployments; run `vercel link && vercel env pull` locally (12-hour expiry).
2. **`AI_GATEWAY_API_KEY`:** For CI pipelines, Node scripts, external servers, and coding agents.
3. **BYOK Provider Keys:** Authenticate Gateway to vendors via dashboard or `providerOptions.gateway.byok` (still requires OIDC or Gateway key).

```
vercel ai-gateway models list          # Inspect live catalog
vercel ai-gateway virtual-models create # Manage vmc/<slug>
vercel ai-gateway budgets set team --amount 500
vercel ai-gateway api-keys create --amount 100
vercel ai-gateway rules list           # Routing rules (beta)
vercel ai-gateway setup                # Wire coding agents
```

`order` vs. `only` vs. `sort`: Leave routing unset by default for maximum resilience. Use `order` to prefer hosts without blocking fallbacks, `sort` for `cost` / `ttft` / `tps`, and reserve `only` strictly for contractual vendor allowlists (since `only` fails if listed hosts go down). HTTP clients can also call `POST /v1/decisions` directly.



Health-Aware Multi-Provider Failover: Protected behind OIDC or API key authentication, AI Gateway evaluates live upstream latency and capacity on every call—transparently rerouting optical streams from a degraded provider pillar to healthy fallbacks with zero application code changes.

Reliable Agents with `ToolLoopAgent` & `Output.object()`

Stop hand-writing fragile `while` loops or calling deprecated `generateObject()`. AI SDK 7 unifies autonomous multi-step tool execution inside `ToolLoopAgent` and strict JSON validation inside `Output.object()`.

1. `ToolLoopAgent` + MCP-Aligned Tool Contract

```
import { ToolLoopAgent, isStepCount, tool } from 'ai';
import { z } from 'zod';

export const supportAgent = new ToolLoopAgent({
  model: 'anthropic/claude-sonnet-4.6',
  instructions: 'Resolve order tickets using verified
tools.',
  tools: {
    lookupOrder: tool({
      description: 'Fetch order shipping state by ID.',
      inputSchema: z.object({ orderId: z.string() }),
      outputSchema: z.object({ status: z.string() }),
      execute: async ({ orderId }) => fetchOrder(orderId),
    }),
  },
  stopWhen: isStepCount(6), // Hard safety ceiling
});

// Call agent.generate() or agent.stream() (NEVER
generateText!)
const res = await supportAgent.generate({ prompt: 'Check
ORD-8841' });
```

2. Typed `Output.object()` & `ModelMessage`

```
import { generateText, Output, convertToModelMessages, type
UIMessage } from 'ai';
import { z } from 'zod';

const TicketSchema = z.object({
  severity: z.enum(['low', 'high', 'critical']),
  category: z.string(),
  actionRequired: z.boolean(),
});

// Convert UIMessage[] from React into ModelMessage[] safely
const modelMessages = await
convertToModelMessages(uiMessages);

const { output } = await generateText({
  model: 'openai/gpt-5.4',
  messages: modelMessages,
  output: Output.object({ schema: TicketSchema }),
});
// output is strictly typed: { severity, category,
actionRequired }
```

Color-Coded Anatomy: Dissecting a Production Agent Contract

[1. Gateway Model ID] [2. System Instructions] [3. inputSchema / outputSchema] [4. Deterministic Execute] [5. stopWhen: isStepCount(N)]

model: 'vmc/triage-agent' or 'anthropic/claude-sonnet-4.6' instructions: 'Never invent order IDs; cite tool outputs verbatim.' tool({ inputSchema: z.object(...), outputSchema: z.object(...) }) execute: async (args) => db.query(...) // Runs on Node 24 Fluid Compute stopWhen: isStepCount(6) // Prevents runaway loops & token burn

Key Migration & Design Rules for Agent Loops

Capability	Correct AI SDK 7 API	Why It Matters in Production
Agent Execution	`agent.generate()` / `agent.stream()`	Encapsulates `instructions`, `tools`, and `stopWhen` once so UI routes and background jobs share identical agent logic.
MCP-Aligned Tools	`inputSchema` + optional `outputSchema`	Matches the Model Context Protocol spec and rejects malformed model arguments before your `execute` function runs.
Message Conversion	`convertToModelMessages(uiMessages)`	Safely strips client UI metadata and converts `UIMessage[]` (with `parts`) into `ModelMessage[]` (never `CoreMessage`).

Pro Tip: Infer Client Types Straight from Your Agent

Export `type SupportAgentUIMessage = InferAgentUIMessage<typeof supportAgent>` and pass it to `useChat<SupportAgentUIMessage>()` so every `tool-lookupOrder` input and output property autocompletes end-to-end in React.

Streaming React & Multimodal UIs with `message.parts`

AI SDK 7 replaces flat text strings with structured `UIMessage.parts`. Stream text deltas, typed tool cards, and custom data chunks cleanly from Fluid Compute routes into React `useChat` hooks.

1. Server Stream & Multimodal Image Generation

```
// app/api/chat/route.ts (Node 24 Fluid Compute — no edge
runtime!)
import { createUIMessageStreamResponse,
createUIMessageStream,
         streamText, convertToModelMessages, generateText }
from 'ai';

export async function POST(req: Request) {
  const { messages } = await req.json();
  const result = streamText({
    model: 'anthropic/claude-sonnet-4.6',
    messages: await convertToModelMessages(messages),
  });
  return createUIMessageStreamResponse({ stream:
result.toUIMessageStream() });
}

// Custom data parts: use writer.write(), NEVER
stream.write()!
const stream = createUIMessageStream({
  execute: async ({ writer }) => { writer.write({ type:
'data-status', data: 'ok' }); },
});

// Multimodal images via Gateway (reads result.files, no
DALL-E!)
const img = await generateText({
  model: 'google/gemini-3.1-flash-image-preview',
  prompt: 'Minimalist glass prism studio render',
});
// img.files[0].uint8Array / base64 ready for Vercel Blob
```

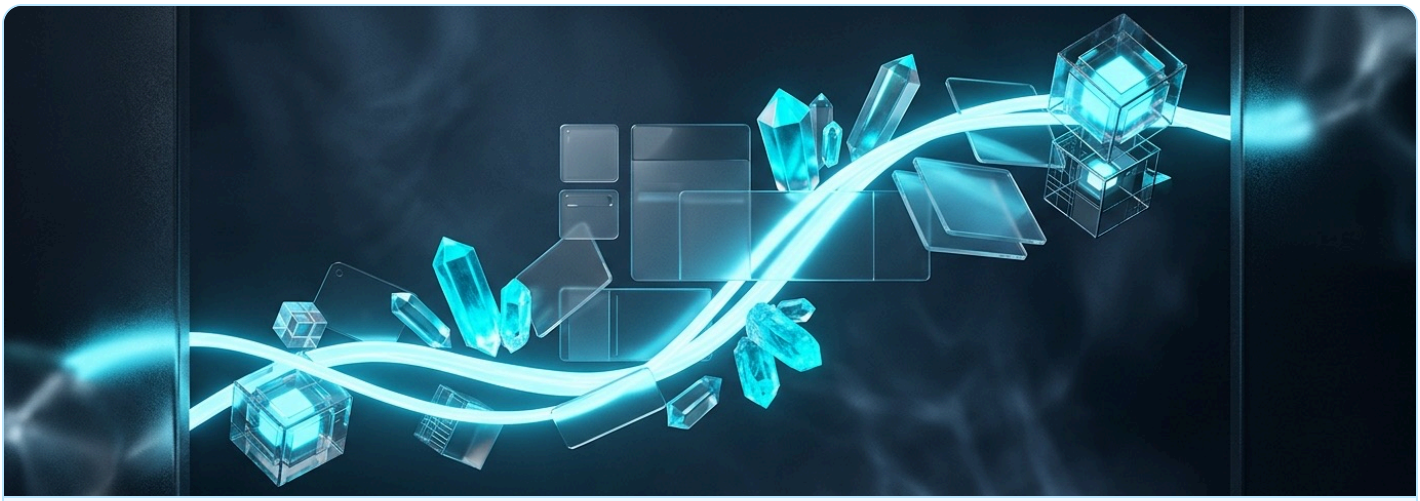
2. React Client: `DefaultChatTransport` & AI Elements

```
'use client';
import { useChat } from '@ai-sdk/react';
import { DefaultChatTransport } from 'ai';
import { MessageResponse } from '@/components/ai-
elements/message';

export function ChatPanel() {
  const { messages, sendMessage, status } = useChat({
    transport: new DefaultChatTransport({ api: '/api/chat'
}),
  });

  const busy = status === 'streaming' || status ===
'submitted';
  return (
    <>
      {messages.map((m) => m.parts.map((part, i) => {
        if (part.type === 'text')
          return <MessageResponse key={i}>{part.text}
</MessageResponse>;
        if (part.type === 'tool-lookupOrder')
          return <OrderCard key={i} state={part.state} data=
{part.output} />;
        return null;
      }))}
      <button disabled={busy} onClick={() => sendMessage({
text: 'Hi' })}>Send</button>
    </>
  );
}
```

Safe Streaming Rendering: Never dump raw model strings into `dangerouslySetInnerHTML` or unguarded Markdown parsers. Render `text` parts with AI Elements (`MessageResponse`) so partial markdown tokens, code blocks, and math formulas stream cleanly without XSS or layout flicker.



Composable `message.parts` Stream: Every response flows down a single Server-Sent Events ribbon as discrete, strongly typed parts—separating prose tokens, `tool-<toolName>` cards, and generated media files cleanly for React UI components.

`vercel.ts`, Embeddings, Reranking & Coding Agents

Beyond chat routes, AI SDK 7 and AI Gateway power typed project configuration via `vercel.ts`, high-accuracy retrieval pipelines with `embedMany` and `rerank`, and centralized coding-agent traffic.

1. Type-Safe Project Config: `vercel.ts`

```
// vercel.ts — replaces legacy vercel.json with full
TypeScript types
import { routes, deploymentEnv, type VercelConfig } from
'@vercel/config/v1';

export const config: VercelConfig = {
  framework: 'nextjs',
  buildCommand: 'pnpm build',
  rewrites: [
    routes.rewrite('/ai/(.*)',
'https://api.example.com/$1'),
  ],
  headers: [
    routes.cacheControl('/assets/(.*)', {
      public: true, maxAge: '1 week', immutable: true,
    }),
  ],
};
```

Install `@vercel/config` to unlock IDE autocomplete, dynamic environment logic via `deploymentEnv`, and compile-time route validation.

2. Two-Stage RAG: `embedMany` + `rerank`

```
import { embed, embedMany, rerank } from 'ai';

// Vectorize chunks in batches through AI Gateway
const { embeddings } = await embedMany({
  model: 'openai/text-embedding-3-small',
  values: chunks,
});

// Cross-encoder reranking boosts top-K retrieval precision
const { ranking } = await rerank({
  model: 'cohere/rerank-v3.5',
  documents: candidateDocs,
  query: userQuestion,
  topN: 5,
});
```

Use `embed()` for single search queries, `embedMany()` for ingestion batches, and `rerank()` before passing context to `ToolLoopAgent`.

Connecting Coding Agents & External Clients via AI Gateway

Route developer tools like Claude Code, Codex CLI, Cursor, or OpenCode through your team's AI Gateway to unify spend caps, model discovery, and audit logs. Run `vercel ai-gateway setup` to detect installed agents, store keys securely in macOS Keychain when supported, and configure protocol-matched URLs automatically.

1 OpenAI-Compatible

Point any OpenAI SDK, Responses client, or Codex agent to:

`https://ai-gateway.vercel.sh/v1`

2 Anthropic-Compatible

Point Anthropic SDKs to base URL below (and `/claude-code` for Claude Code):

`https://ai-gateway.vercel.sh`

3 Virtual Models

Give coding agents `vmc/<slug>` so server-side routing, fallbacks & reasoning options apply without client flags.

```
# Connect supported coding agents interactively (or preview with --dry-run)
vercel ai-gateway setup --dry-run
vercel ai-gateway setup --agent claude-code --amount 100 --refresh-period monthly

# Manual Claude Code wiring: ANTHROPIC_API_KEY MUST stay empty!
export ANTHROPIC_BASE_URL="https://ai-gateway.vercel.sh/claude-code"
export ANTHROPIC_API_KEY=""
export ANTHROPIC_AUTH_TOKEN="$AI_GATEWAY_API_KEY"
export CLAUDE_CODE_ENABLE_GATEWAY_MODEL_DISCOVERY=1
```

Centralize Agent Policy with `vmc/<slug>` Virtual Models

Most coding agents only expose a model ID picker and cannot pass `providerOptions.gateway`. Create a Virtual Model via `vercel ai-gateway virtual-models create` to pin provider order, ZDR compliance, and fallbacks server-side, then select `vmc/<slug>` in your agent.

5 Real-World Copy-Paste Recipes

Drop these five production-ready patterns straight into Next.js App Router or Node.js 24 LTS services. Every recipe uses verified AI SDK 7 exports, plain `provider/model` routing, and Fluid Compute defaults.

Recipe & Use Case	Stack & Auth	Verified AI SDK 7 Copy-Paste Pattern

1. Streaming App Router Chatbot
Interactive Chat UI
	Node 24 Fluid Compute
	`VERCEL_OIDC_TOKEN`
	`DefaultChatTransport`
		`// app/api/chat/route.ts — NO runtime = 'edge'!`
		`const result = streamText({`
		`  model: 'anthropic/claude-sonnet-4.6',`
		`  messages: await convertToModelMessages(messages),`
		`  providerOptions: { gateway: { caching: 'auto' } },`
		`});`
		`return createUIMessageStreamResponse({ stream:`
		`result.toUIMessageStream() });`

2. Strictly Typed JSON Extraction
Invoice / Ticket Parser
	`Output.object()`
	`zod` schema
	Zero deprecated APIs
		`const { output } = await generateText({`
		`  model: 'openai/gpt-5.4',`
		`  output: Output.object({`
		`  schema: z.object({ totalUsd: z.number(), vendor: z.string(),`
		`lineItems: z.array(z.string()) }),`
		`  }),`
		`  prompt: 'Extract structured fields from this invoice text...',`
		`});`

3. Resilient Virtual Model Router
Multi-Provider Support
	`vmc/<slug>`
	Team Budget Cap
	Fallback Chain
		`// CLI: vercel ai-gateway budgets set project support-app --amount 200`
		`const { text, providerMetadata } = await generateText({`
		`  model: 'vmc/support-router', // Central order, ZDR & fallback models`
		`  prompt: customerTicket,`
		`  providerOptions: { gateway: { user: customerId, tags: ['tier:pro'] }`
		`},`
		`});`

4. Multi-Step Research Pipeline
Autonomous Search Agent
	`ToolLoopAgent`
	`isStepCount(8)`
	`inputSchema`
		`const researcher = new ToolLoopAgent({`
		`  model: 'google/gemini-2.5-pro',`
		`  instructions: 'Search docs and synthesize cited findings.',`
		`  tools: { searchDocs: tool({ description: 'Query KB', inputSchema:`
		`z.object({ q: z.string() }), execute }) },`
		`  stopWhen: isStepCount(8),`
		`});`
		`const result = await researcher.generate({ prompt: topic });`

5. Gateway Image Gen → Vercel Blob
Asset Generation API
	`gemini-3.1-flash-image-preview`
	`@vercel/blob`
		`const { files } = await generateText({`
		`  model: 'google/gemini-3.1-flash-image-preview',`
		`  prompt: 'Clean isometric product icon on slate background',`
		`});`
		`const blob = await put('renders/icon.png',`
		`Buffer.from(files[0].uint8Array), { access: 'public' });`

Always Inspect `providerMetadata.gateway` in Critical Workflows

Whenever a recipe handles billing, compliance, or fallback routing, capture `providerMetadata?.gateway?.generationId` and log it alongside your app trace so you can correlate exact token cost and provider attempts in AI Gateway Logs.

Cost Controls, Observability, Auth & Security Hygiene

Production AI apps fail expensively when credentials leak or loops run uncapped. AI Gateway stacks team, project, key, and user budgets with Zero Data Retention routing and OpenTelemetry-ready request logs.

Security Rule: Never Commit Keys or Print OIDC Tokens in Logs

Keep `AI_GATEWAY_API_KEY` and BYOK provider keys strictly in Vercel Environment Variables or uncommitted `.env.local` files. Locally pulled `VERCEL_OIDC_TOKEN` values expire every **12 hours**—refresh them with `vercel env pull` rather than pasting long-lived keys into source.

Stacked Budgets & Credit Controls

- **Four Stacking Scopes:** Set limits across `team`, `project` (OIDC), `api-key`, and `user`. Every active scope must pass or Gateway returns HTTP `402` (`quota_for_entity_exceeded`).
- **Per-Scope Defaults:** Use `vercel ai-gateway budgets set api-key --default --amount 50 --refresh-period monthly` so new keys inherit caps automatically.
- **Proactive Alerts:** Custom budgets email at 50%, 75%, and 100% thresholds; changes propagate within tens of seconds (up to ~5 minutes for hot keys).

Privacy, Prompt Caching & Safety IDs

- **Zero Data Retention & No Training:** Pass `zeroDataRetention: true` and `disallowPromptTraining: true` under `providerOptions.gateway`, or check `zdr` / `no_training` fields in `/v1/models`.
- **Prompt Caching (`caching: 'auto'`):** Injects provider prompt-cache markers on explicit-caching hosts (like Anthropic) for multi-turn prefixes—never use non-existent `cacheControl` response caching.
- **`safetyIdentifier`:** Send an opaque hashed end-user ID (≤64 chars) so provider abuse filters isolate one user instead of your whole team.

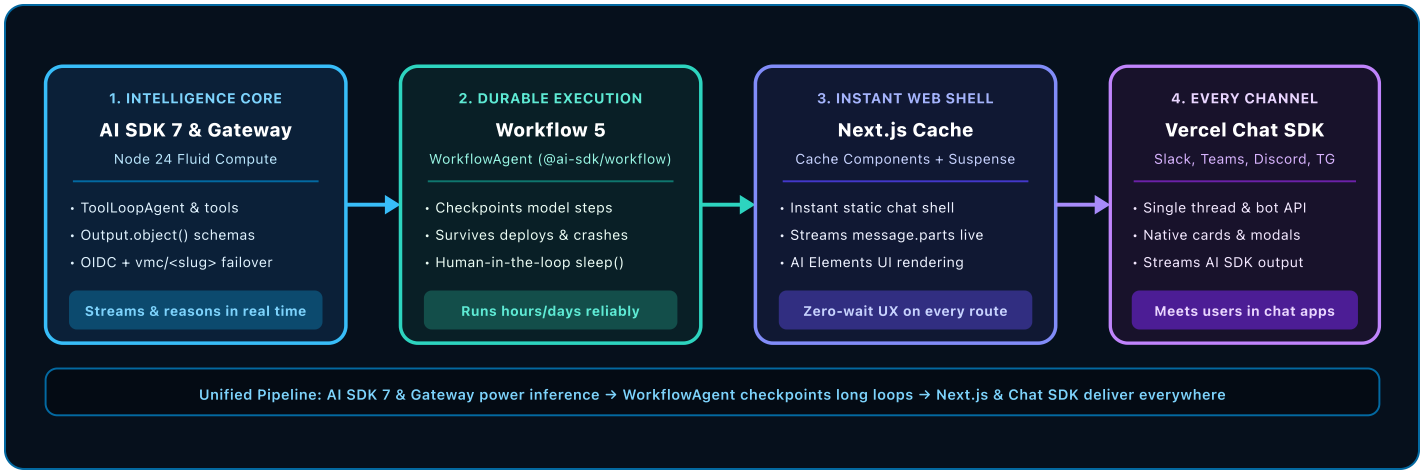
AI Gateway Observability, Logs & Billing REST Endpoints

Observability Surface	What You Inspect & Ingestion Window	Recommended Production Practice
AI Gateway Logs	Every attempt, status, TTFT, auth method, cost, and provider response. **~90s ingestion window**; routing cards retained 30 days.	Filter by `status`, `model`, or shareable URL params; export CSV/JSON or tail Live mode during incident triage.
Custom Reporting	Aggregates spend by `user`, `tags` (up to 10 per call), `model`, and `zdr` status via `GET /v1/report` or `gateway.getSpendReport()`.	Tag requests with `['feature:chat', 'env:prod']` (note: `user` and `tags` are reporting dimensions, not rate limits).
Credits & Generation API	`GET /v1/credits` returns remaining credit balance; `GET /v1/generation?id=...` returns exact token cost & latency.	Save `providerMetadata.gateway.generationId` in DB audit logs to reconcile per-tenant usage programmatically.

```
# Inspect team budgets, credit balance & usage report directly from terminal
vercel ai-gateway budgets list && vercel ai-gateway budgets list --defaults
curl -fsSL "https://ai-gateway.vercel.sh/v1/credits" -H "Authorization: Bearer $AI_GATEWAY_API_KEY"
```

Cross-Tool Synergy: The Full Vercel AI Stack

AI SDK 7 and AI Gateway form the intelligence core of Vercel's platform. Pair them cleanly with **Workflow 5** for crash-safe multi-minute runs, **Next.js Cache Components** for instant UI shells, and **Vercel Chat SDK** for multi-platform bots.



When to Reach for Each Adjacent Vercel Building Block

1 Upgrading `ToolLoopAgent` to `WorkflowAgent`

When an agent job exceeds a single HTTP request (waiting on external webhooks, human approvals, or multi-minute scraping), swap to `WorkflowAgent` from `@ai-sdk/workflow` on Workflow 5 (replacing legacy `DurableAgent`). Plain `provider/model` Gateway strings and `isStepCount` keep working identically.

2 Delivering Across Web & Messaging Channels

Pair **Next.js Cache Components** with `useChat` so static layouts paint instantly while AI streams hydrate inside Suspense boundaries, then reuse the exact same agent backend inside **Vercel Chat SDK** adapters for Slack, Teams, Discord, and Telegram.

One Agent Definition, Four Delivery Surfaces

By encapsulating instructions, Zod tool schemas, and `vmc/<slug>` routing inside a shared TypeScript module, the exact same intelligence serves interactive web chats, durable Workflow 5 pipelines, and Slack bots simultaneously.

Troubleshooting Clinic & Pre-Flight Checklist

Almost every AI SDK 7 or AI Gateway snag stems from a handful of outdated v4–v6 habits or environment mismatches. Match your symptom below for a ten-second fix, then clear the pre-flight checklist before deploying.

Symptom / Error	Root Cause	10-Second Fix
1. `export const runtime = 'edge'` breaks Node deps	Outdated habit added Edge runtime to a route using Node crypto, FS, or Next.js 16.3+ APIs.	Delete `export const runtime = 'edge'` entirely. Node.js 24 LTS on Fluid Compute streams SSE and AI tokens natively.
2. `useChat({ api })`, `handleSubmit`, or `isLoading` Type Error	Client code uses removed AI SDK v4 hook signatures instead of v5–v7 transport APIs.	Use `useChat({ transport: new DefaultChatTransport({ api }) })`, `sendMessage({ text })`, `status`, and `message.parts`.
3. `generateObject`, `CoreMessage`, or `maxSteps` Fails	Deprecated or removed v4–v6 symbols copied from stale snippets.	Switch to `generateText` + `Output.object({ schema })`, `ModelMessage` + `convertToModelMessages()`, and `stopWhen: isStepCount(N)`.
4. `gateway('gpt-5.4')` Missing Provider Error	Model ID omitted its required vendor prefix (or used hyphenated Claude slugs).	Always pass `provider/model` IDs from `/v1/models` (e.g., `'openai/gpt-5.4'` or `'anthropic/claude-sonnet-4.6'`).
5. Claude Code Ignores Gateway Key (`401`)	A non-empty `ANTHROPIC_API_KEY` in your shell overrides `ANTHROPIC_AUTH_TOKEN`.	Explicitly export `ANTHROPIC_API_KEY=""` alongside `ANTHROPIC_BASE_URL="https://ai-gateway.vercel.sh/claude-code"` and `ANTHROPIC_AUTH_TOKEN`.
6. Missing or Expired Local `VERCEL_OIDC_TOKEN`	App runs locally without linking the project or the 12-hour OIDC token expired.	Run `vercel link && vercel env pull` to refresh `.env.local`, or use `AI_GATEWAY_API_KEY` for standalone scripts.
7. `stream.write is not a function` in `createUIMessageStream`	Code calls `stream.write()` on the returned `ReadableStream` instead of the writer.	Destructure `{ writer }` inside `execute: async ({ writer }) =>` and call `writer.write(...)`.

6-Point Pre-Flight Checklist Before Shipping to Production

- [x] **Node 24 Fluid Compute:** Confirmed zero `runtime = 'edge'` exports and verified Node 22+ ESM compatibility.
- [x] **Live Catalog Verified:** Checked `provider/model` IDs (or `vmc/<slug>`) against `https://ai-gateway.vercel.sh/v1/models`.
- [x] **Version-Matched APIs:** Verified `ToolLoopAgent`, `Output.object()`, `isStepCount()`, and `message.parts` against `node_modules/ai/docs/`.
- [x] **OIDC & Budget Guardrails:** Confirmed `VERCEL_OIDC_TOKEN` or scoped `AI_GATEWAY_API_KEY` plus team/project spend caps via `budgets set`.
- [x] **Safe UI & ZDR Hygiene:** Rendered prose via AI Elements (`MessageResponse`) and applied `zeroDataRetention` where policy requires.
- [x] **Trace Verified in Logs:** Ran a live check via `npx @ai-sdk/devtools` and confirmed routing, usage, and cost in AI Gateway Logs (~90s).