readme.md for @push.rocks/smartai

A unified provider registry for the Vercel AI SDK □□⚡

npm version TypeScript License: MIT

SmartAI gives you a single `getModel()` function that returns a standard `LanguageModelV3` for **any** supported provider — Anthropic, OpenAI, Google, Groq, Mistral, XAI, Perplexity, or Ollama. Use the returned model with the Vercel AI SDK's `generateText()`, `streamText()`, and tool ecosystem. Specialized capabilities like vision, audio, image generation, document analysis, and web research are available as dedicated subpath imports.

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

□□ Why SmartAI?

- □□ **One function, eight providers** — `getModel()` returns a standard `LanguageModelV3`. Switch providers by changing a string.
- □□ **Built on Vercel AI SDK** — Uses `ai` v6 under the hood. Your model works with `generateText()`, `streamText()`, tool calling, structured output, and everything else in the AI SDK ecosystem.
- □□ **Custom Ollama provider** — A full `LanguageModelV3` implementation for Ollama with support for `think` mode, `num_ctx`, auto-tuned temperature for Qwen models, and native tool calling.
- □□ **Anthropic prompt caching** — Automatic `cacheControl` middleware reduces cost and latency on repeated calls. Enabled by default, opt out with `promptCaching: false`.
- □□ **Modular subpath exports** — Vision, audio, image, document, and research capabilities ship as separate imports. Only import what you need.

- ⚡ **Zero lock-in** — Your code uses standard AI SDK types. Swap providers without touching application logic.

□□ Installation

```
pnpm install @push.rocks/smartai
```

□□ Quick Start

```
import { getModel, generateText, streamText } from '@push.rocks/smartai';

// Get a model for any provider
const model = getModel({
  provider: 'anthropic',
  model: 'claude-sonnet-4-5-20250929',
  apiKey: process.env.ANTHROPIC_TOKEN,
});

// Use it with the standard AI SDK functions
const result = await generateText({
  model,
  prompt: 'Explain quantum computing in simple terms.',
});

console.log(result.text);
```

That's it. Change `provider` to `'openai'` and `model` to `'gpt-4o'` and the rest of your code stays exactly the same.

□□ Core API

`getModel(options): LanguageModelV3`

The primary export. Returns a standard `LanguageModelV3` you can use with any AI SDK function.

```
import { getModel } from '@push.rocks/smartai';
import type { ISmartAiOptions } from '@push.rocks/smartai';

const options: ISmartAiOptions = {
  provider: 'anthropic',  // 'anthropic' | 'openai' | 'google' | 'groq' | 'mistral' | 'xai' |
'perplexity' | 'ollama'
  model: 'claude-sonnet-4-5-20250929',
  apiKey: 'sk-ant-...',
  // Anthropic-only: prompt caching (default: true)
  promptCaching: true,
  // Ollama-only: base URL (default: http://localhost:11434)
  baseUrl: 'http://localhost:11434',
  // Ollama-only: model runtime options
  ollamaOptions: { think: true, num_ctx: 4096 },
};

const model = getModel(options);
```

Re-exported AI SDK Functions

SmartAI re-exports the most commonly used functions from `ai` for convenience:

```
import {
  getModel,
  generateText,
  streamText,
  tool,
  jsonSchema,
} from '@push.rocks/smartai';

import type {
  ModelMessage,
  ToolSet,
  StreamTextResult,
  LanguageModelV3,
} from '@push.rocks/smartai';
```

□□ Supported Providers

Provider	Package	Example Models
Anthropic	`@ai-sdk/anthropic`	`claude-sonnet-4-5-20250929`, `claude-opus-4-5-20250929`
OpenAI	`@ai-sdk/openai`	`gpt-4o`, `gpt-4o-mini`, `o3-mini`
Google	`@ai-sdk/google`	`gemini-2.0-flash`, `gemini-2.5-pro`
Groq	`@ai-sdk/groq`	`llama-3.3-70b-versatile`, `mixtral-8x7b-32768`
Mistral	`@ai-sdk/mistral`	`mistral-large-latest`, `mistral-small-latest`
XAI	`@ai-sdk/xai`	`grok-3`, `grok-3-mini`
Perplexity	`@ai-sdk/perplexity`	`sonar-pro`, `sonar`
Ollama	Custom `LanguageModelV3`	`qwen3:8b`, `llama3:8b`, `deepseek-r1`

□□ Text Generation

Generate Text

```
import { getModel, generateText } from '@push.rocks/smartai';

const model = getModel({
  provider: 'openai',
  model: 'gpt-4o',
  apiKey: process.env.OPENAI_TOKEN,
});

const result = await generateText({
  model,
  system: 'You are a helpful assistant.',
  prompt: 'What is 2 + 2?',
});

console.log(result.text); // "4"
```

Stream Text

```
import { getModel, streamText } from '@push.rocks/smartai';

const model = getModel({
  provider: 'anthropic',
  model: 'claude-sonnet-4-5-20250929',
  apiKey: process.env.ANTHROPIC_TOKEN,
});

const result = await streamText({
  model,
  prompt: 'Count from 1 to 10.',
});

for await (const chunk of result.textStream) {
  process.stdout.write(chunk);
}
```

Tool Calling

```
import { getModel, generateText, tool, jsonSchema } from '@push.rocks/smartai';

const model = getModel({
  provider: 'anthropic',
  model: 'claude-sonnet-4-5-20250929',
  apiKey: process.env.ANTHROPIC_TOKEN,
});

const result = await generateText({
  model,
  prompt: 'What is the weather in London?',
  tools: {
    getWeather: tool({
      description: 'Get weather for a location',
      parameters: jsonSchema({
        type: 'object',
```

```
        properties: {
          location: { type: 'string' },
        },
        required: ['location'],
      }),
      execute: async ({ location }) => {
        return { temperature: 18, condition: 'cloudy' };
      },
    }),
  },
});
```

Ollama (Local Models)

The custom Ollama provider implements `LanguageModelV3` directly, calling Ollama's native `/api/chat` endpoint. This gives you features that generic OpenAI-compatible wrappers miss:

```
import { getModel, generateText } from '@push.rocks/smartai';

const model = getModel({
  provider: 'ollama',
  model: 'qwen3:8b',
  baseUrl: 'http://localhost:11434', // default
  ollamaOptions: {
    think: true,      // Enable thinking/reasoning mode
    num_ctx: 8192,     // Context window size
    temperature: 0.7,  // Override default (Qwen models auto-default to 0.55)
  },
});

const result = await generateText({
  model,
  prompt: 'Solve this step by step: what is 15% of 340?',
});

console.log(result.text);
```

Ollama Features

- **`think` mode** — Enables reasoning for models that support it (Qwen3, QwQ, DeepSeek-R1). The `think` parameter is sent at the top level of the request body as required by the Ollama API.
- **Auto-tuned temperature** — Qwen models automatically get `temperature: 0.55` when no explicit temperature is set, matching the recommended inference setting.
- **Native tool calling** — Full tool call support via Ollama's native format (not shimmed through OpenAI-compatible endpoints).
- **Streaming with reasoning** — `doStream()` emits proper `reasoning-start`, `reasoning-delta`, `reasoning-end` parts alongside text.
- **All Ollama options** — `num_ctx`, `top_k`, `top_p`, `repeat_penalty`, `num_predict`, `stop`, `seed`.

□□ Anthropic Prompt Caching

When using the Anthropic provider, SmartAI automatically wraps the model with caching middleware that adds `cacheControl: { type: 'ephemeral' }` to the last system message and last user message. This can significantly reduce cost and latency for repeated calls with the same system prompt.

```
// Caching enabled by default
const model = getModel({
  provider: 'anthropic',
  model: 'claude-sonnet-4-5-20250929',
  apiKey: process.env.ANTHROPIC_TOKEN,
});

// Opt out of caching
const modelNoCaching = getModel({
  provider: 'anthropic',
  model: 'claude-sonnet-4-5-20250929',
  apiKey: process.env.ANTHROPIC_TOKEN,
  promptCaching: false,
});
```

You can also use the middleware directly:

```
import { createAnthropicCachingMiddleware } from '@push.rocks/smartai';
import { wrapLanguageModel } from 'ai';
```

```
const middleware = createAnthropicCachingMiddleware();
const cachedModel = wrapLanguageModel({ model: baseModel, middleware });
```

□□ Subpath Exports

SmartAI provides specialized capabilities as separate subpath imports. Each one is a focused utility that takes a model (or API key) and does one thing well.

□□ Vision — `@push.rocks/smartai/vision`

Analyze images using any vision-capable model.

```
import { analyzeImage } from '@push.rocks/smartai/vision';
import { getModel } from '@push.rocks/smartai';
import * as fs from 'fs';

const model = getModel({
  provider: 'anthropic',
  model: 'claude-sonnet-4-5-20250929',
  apiKey: process.env.ANTHROPIC_TOKEN,
});

const description = await analyzeImage({
  model,
  image: fs.readFileSync('photo.jpg'),
  prompt: 'Describe this image in detail.',
  mediaType: 'image/jpeg', // optional, defaults to 'image/jpeg'
});

console.log(description);
```

`analyzeImage(options)` accepts:

- `model` — Any `LanguageModelV3` with vision support
- `image` — `Buffer` or `Uint8Array`
- `prompt` — What to ask about the image
- `mediaType` — `'image/jpeg'` | `'image/png'` | `'image/webp'` | `'image/gif'`

Audio — `@push.rocks/smartai/audio`

Text-to-speech using OpenAI's TTS models.

```
import { textToSpeech } from '@push.rocks/smartai/audio';
import * as fs from 'fs';

const stream = await textToSpeech({
  apiKey: process.env.OPENAI_TOKEN,
  text: 'Welcome to the future of AI development!',
  voice: 'nova',     // 'alloy' | 'echo' | 'fable' | 'onyx' | 'nova' | 'shimmer'
  model: 'tts-1-hd', // 'tts-1' | 'tts-1-hd'
  responseFormat: 'mp3', // 'mp3' | 'opus' | 'aac' | 'flac'
  speed: 1.0,         // 0.25 to 4.0
});

stream.pipe(fs.createWriteStream('welcome.mp3'));
```

Image — `@push.rocks/smartai/image`

Generate and edit images using OpenAI's image models.

```
import { generateImage, editImage } from '@push.rocks/smartai/image';

// Generate an image
const result = await generateImage({
  apiKey: process.env.OPENAI_TOKEN,
  prompt: 'A futuristic cityscape at sunset, digital art',
  model: 'gpt-image-1',     // 'gpt-image-1' | 'dall-e-3' | 'dall-e-2'
  quality: 'high',           // 'low' | 'medium' | 'high' | 'auto'
  size: '1024x1024',
  background: 'transparent', // gpt-image-1 only
  outputFormat: 'png',       // 'png' | 'jpeg' | 'webp'
  n: 1,
});

// result.images[0].b64_json — base64-encoded image data
const imageBuffer = Buffer.from(result.images[0].b64_json!, 'base64');
```

```
// Edit an existing image
const edited = await editImage({
  apiKey: process.env.OPENAI_TOKEN,
  image: imageBuffer,
  prompt: 'Add a rainbow in the sky',
  model: 'gpt-image-1',
});
```

□□ Document — `@push.rocks/smartai/document`

Analyze PDF documents by converting them to images and using a vision model. Uses `@push.rocks/smartpdf` for PDF-to-PNG conversion (requires Chromium/Puppeteer).

```
import { analyzeDocuments, stopSmartpdf } from '@push.rocks/smartai/document';
import { getModel } from '@push.rocks/smartai';
import * as fs from 'fs';

const model = getModel({
  provider: 'anthropic',
  model: 'claude-sonnet-4-5-20250929',
  apiKey: process.env.ANTHROPIC_TOKEN,
});

const analysis = await analyzeDocuments({
  model,
  systemMessage: 'You are a legal document analyst.',
  userMessage: 'Summarize the key terms and conditions.',
  pdfDocuments: [fs.readFileSync('contract.pdf')],
  messageHistory: [],  // optional: prior conversation context
});

console.log(analysis);

// Clean up the SmartPdf instance when done
await stopSmartpdf();
```

□□ Research — `@push.rocks/smartai/research`

Perform web-search-powered research using Anthropic's `web_search_20250305` tool.

```
import { research } from '@push.rocks/smartai/research';

const result = await research({
  apiKey: process.env.ANTHROPIC_TOKEN,
  query: 'What are the latest developments in quantum computing?',
  searchDepth: 'basic',     // 'basic' | 'advanced' | 'deep'
  maxSources: 10,           // optional: limit number of search results
  allowedDomains: ['nature.com', 'arxiv.org'],  // optional: restrict to domains
  blockedDomains: ['reddit.com'],               // optional: exclude domains
});

console.log(result.answer);
console.log('Sources:', result.sources);       // Array<{ url, title, snippet }>
console.log('Queries:', result.searchQueries); // search queries the model used
```

□□ Testing

```
# All tests
pnpm test

# Individual test files
tstest test/test.smartai.ts --verbose    # Core getModel + generateText + streamText
tstest test/test.ollama.ts --verbose     # Ollama provider (mocked, no API needed)
tstest test/test.vision.ts --verbose     # Vision analysis
tstest test/test.image.ts --verbose      # Image generation
tstest test/test.research.ts --verbose   # Web research
tstest test/test.audio.ts --verbose      # Text-to-speech
tstest test/test.document.ts --verbose   # Document analysis (needs Chromium)
```

Most tests skip gracefully when API keys are not set. The Ollama tests are fully mocked and require no external services.

□□ Architecture

```
@push.rocks/smartai
├── ts/                            # Core package
│   ├── index.ts                   # Re-exports getModel, AI SDK functions, types
│   ├── smartai.classes.smartai.ts  # getModel() — provider switch
│   ├── smartai.interfaces.ts      # ISmartAiOptions, TProvider, IOllamaModelOptions
│   ├── smartai.provider.ollama.ts  # Custom LanguageModelV3 for Ollama
│   ├── smartai.middleware.anthropic.ts  # Prompt caching middleware
│   └── plugins.ts                 # AI SDK provider factories
├── ts_vision/                     # @push.rocks/smartai/vision
├── ts_audio/                      # @push.rocks/smartai/audio
├── ts_image/                      # @push.rocks/smartai/image
├── ts_document/                   # @push.rocks/smartai/document
└── ts_research/                   # @push.rocks/smartai/research
```

The core package is a thin registry. `getModel()` creates the appropriate `@ai-sdk/*` provider, calls it with the model ID, and returns the resulting `LanguageModelV3`. For Anthropic, it optionally wraps the model with prompt caching middleware. For Ollama, it returns a custom `LanguageModelV3` implementation that talks directly to Ollama's `/api/chat` endpoint.

Subpath modules are independent — they import `ai` and provider SDKs directly, not through the core package. This keeps the dependency graph clean and allows tree-shaking.

License and Legal Information

Trademarks

license granted herein.

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #3
Created 2026-03-28 11:13:35 UTC by foss.global Team
Updated 2026-03-28 12:20:20 UTC by foss.global Team