

Laravel AI SDK cheat sheet (laravel/ai 1.2)

Agents, prompt/stream/queue, attachments, structured output, tools with approval, Stop middleware, conversations, streaming to Vue, embeddings and vector search, images, audio, transcription, reranking, classification, events and the testing fakes, checked against laravel/ai 1.2.3.



laravel/ai 1.2 · Laravel 13

Install and scaffold

Package, config, migrations

```
composer require laravel/ai
php artisan vendor:publish --
provider="Laravel\Ai\AiServiceProvider"
php artisan migrate          # conversations, messages,
                             # approvals
# .env: OPENAI_API_KEY=... ANTHROPIC_API_KEY=... (defaults in
config/ai.php)
php artisan make:agent SalesCoach # --structured for a
schema() stub
php artisan make:tool RandomNumberGenerator
php artisan make:agent-middleware EnforceBudget
```

An agent

```
use Laravel\Ai\Attributes\{MaxSteps, Model, Provider,
Temperature, Timeout};
use Laravel\Ai\Contracts\{Agent, HasTools};
use Laravel\Ai\Enums\Lab;
use Laravel\Ai\Promptable;

#[Provider([Lab::Anthropic, Lab::OpenAI])] // first wins, next
is the failover
#[Model('claude-sonnet-5'), MaxSteps(8), Temperature(0.2),
Timeout(120)]
class SalesCoach implements Agent, HasTools
{
    use Promptable;

    public function instructions(): string { return 'You are a
sales coach...'; }
    public function tools(): iterable { return [new
RandomNumberGenerator]; }
}
```

Prompt, stream, queue

Three ways to run it

```
$response = (new SalesCoach)->prompt('Analyze this
transcript...');
$response->text; $response->usage->inputTokens; $response-
>steps; $response->meta->provider;

(new SalesCoach)->prompt($text, provider: Lab::OpenAI, model:
'gpt-5', timeout: 60); // one-off override
return (new SalesCoach)->stream($text);          // SSE
response for a route
(new SalesCoach)->queue($text);                  // on
the queue; ->broadcast($text) streams to a channel
Str::of($longText)->summarize();                // one-
liner helpers: summarize(), decide(), toEmbeddings(), toAudio()
```

Attachments

```
use Laravel\Ai\Files\{Document, Image};

(new ImageAnalyzer)->prompt('What is in this image?',
attachments: [
    Image::fromPath('/tmp/photo.jpg'),          // fromUrl,
    fromUpload($request->file('x')), fromStorage('disk', 'path'),
    fromBase64
    Document::fromPath('/tmp/brief.md'),        // also
    fromString('...')
]);
```

Structured output and tools

Structured output

```
use Illuminate\Contracts\JsonSchema\JsonSchema;
use Laravel\Ai\Contracts\HasStructuredOutput;

class SalesCoach implements Agent, HasStructuredOutput
{
    public function schema(JsonSchema $schema): array
    {
        return ['score' => $schema->integer()->min(0)-
>max(100)->required(),
                'summary' => $schema->string()->required()];
    }
}
$response['score']; // array access on the response
```

A tool, with approval

```
use Laravel\Ai\Concerns\InteractsWithApprovals;
use Laravel\Ai\Contracts\{Approvable, Tool};
use Laravel\Ai\Tools\Request;

class DeleteFile implements Tool, Approvable
{
    use InteractsWithApprovals;          // the model must
    get a Decision first

    public function description(): string { return 'Delete a
file by path.'; }
    public function schema(JsonSchema $schema): array { return
['path' => $schema->string()->required()]; }
    public function handle(Request $request): string {
        unlink($request['path']); return 'Deleted.'; }
}
// Resume with the user's answer
(new FileAssistant)->continue($conversationId, as: $user)
->prompt(Decisions::from(['call_abc' =>
Decision::approve(), 'call_def' => Decision::reject('Keep
it.')]));
```

Middleware and conversations

Stop a run early (1.2.1+)

```
use Laravel\Ai\Contracts\HasMiddleware;
use Laravel\Ai\Middleware\Stop;
use Laravel\Ai\PendingStep;

public function middleware(): array          // on an Agent
implementing HasMiddleware
{
    return [
        Stop::when(fn (PendingStep $step) => Cache::has("chat:
{$this->chat->id}:stopped")),
        Stop::unless(fn (PendingStep $step) => $step->usage-
>inputTokens < 50_000),
    ];
}
// Custom: handle(PendingStep $step, Closure $next) { return
$next($step); } or StepResult::stop()
```

Remember the conversation

```
use Laravel\Ai\Concerns\RemembersConversations;
use Laravel\Ai\Contracts\Conversational;

class SupportAgent implements Agent, Conversational { use
Promptable, RemembersConversations; }

$response = (new SupportAgent)->forUser($user)-
>prompt('Hello!');
$id = $response->conversationId;
(new SupportAgent)->continue($id, as: $user)->prompt('Tell me
more. ');
(new SupportAgent)->forUser($user)->continueLastConversation()-
>prompt('And then?');
```

Streaming to Vue

Routes

```
Route::post('/chats/{chat}/stream', fn (Chat $chat, Request $r)
=> (new SupportAgent($chat)->stream($r->string('prompt')));
// Vercel AI SDK protocol (useChat):
Route::post('/chat', function (Request $request) {
    $chat = Vercel::chat($request);
    return (new SupportAgent)->withMessages($chat->history())-
->stream($chat)->usingProtocol($chat->protocol());
});
```

Read the SSE in a component

```
const res = await fetch(`/chats/${chatId}/stream`, { method:
'POST', headers: { 'Content-Type': 'application/json', 'X-XSRF-
TOKEN': xsrf() }, body: JSON.stringify({ prompt }) })
const reader = res.body!.pipeThrough(new
TextDecoderStream()).getReader()
let buffer = ''
while (true) {
    const { value, done } = await reader.read(); if (done) break
    buffer += value; const chunks = buffer.split('\n\n'); buffer
= chunks.pop() ?? ''
    for (const chunk of chunks) {
        const data = chunk.replace(/^data: /, ''); if (data ===
'[DONE]') continue
        const event = JSON.parse(data) // text_delta, tool_call,
tool_result, tool_approval_request, stream_end
        if (event.type === 'text_delta') answer.value +=
event.delta
        if (event.type === 'stream_end') status.value =
event.reason === 'cancelled' ? 'stopped' : 'done'
    }
}
```

Embeddings, images, audio, reranking

Embeddings and similarity search

```
$response = Embeddings::for(['Napa Valley has great wine.',
'Laravel is a PHP framework.'])->generate();
$vector = Str::of('Napa Valley has great wine.')->
->toEmbeddings();
Document::query()->whereVectorSimilarTo('embedding', 'best
wineries in Napa')->limit(10)->get(); // Laravel 13 vector
columns
$store = Stores::create('Knowledge Base'); $store-
>add(Document::fromPath('/tmp/handbook.pdf')); // provider file
stores
```

Images, audio, transcription, reranking

```
$image = Image::of('A donut on a kitchen counter')-
>landscape()->quality('high')->generate();
$image->storeAs('donut.jpg'); //
quality(), square(), portrait(), size(), attachments(), queue()
$audio = Audio::of('I love coding with Laravel.')->
>voice('alloy')->generate(); $audio->storeAs('hello.mp3');
$text = Transcription::fromPath('/tmp/call.mp3')-
>language('en')->diarize()->generate();
$ranked = Reranking::of($documents)->limit(3)->rerank('PHP
frameworks');
```

Classification

Probabilities instead of generated text

```
use Laravel\Ai\Classification;
use Laravel\Ai\Classification\{Boolean, Choice, Score};

$result = Classification::of($ticket->body, ['front' =>
Image::fromPath($photo)] // attachments optional
->questions([
    'urgent' => new Boolean('Does this need an immediate
response?'),
    'team' => new Choice('Which team?', ['billing' =>
'Invoices and refunds', 'tech' => 'Bugs and outages']),
    'anger' => new Score('How frustrated is the customer?',
['Calm', 'Frustrated', 'Very angry']),
]);
->classify(provider: 'openai'); // default provider:
config 'default_for_classification' (typesafe)

$result['urgent']->isTrue(threshold: 0.8); $result['urgent']-
>probability;
$result['team']->choice; $result['team']-
>probabilityOf('tech'); $result['anger']->level();
$result['anger']->label();
```

Events and testing

Listen

```
use Laravel\Ai\Events\{AgentPrompted, AgentFailed,
ClassificationFailed, EmbeddingsFailed, ToolInvoked};

Event::listen(AgentPrompted::class, fn (AgentPrompted $e) => /*
usage, provider, model */ null);
Event::listen([EmbeddingsFailed::class,
ClassificationFailed::class, RecordAiFailure::class]; //
1.2.2+: *Failed per attempt
```

Fakes and assertions

```
SalesCoach::fake(['Scored 82/100.']); // canned
answers in order, or a closure
(new SalesCoach)->prompt('Analyze...');
SalesCoach::assertPrompted(fn (AgentPrompt $p) => $p-
>contains('Analyze')); // assertPromptedTimes,
assertNotPrompted, assertQueued

Embeddings::fake(); Image::fake(); Audio::fake();
Transcription::fake(); Reranking::fake();
Classification::fake();
Embeddings::assertGenerated(fn (EmbeddingsPrompt $p) => $p-
>contains('Laravel'));
Image::assertNothingGenerated();
Classification::assertClassified(fn (ClassificationPrompt $p)
=> $p->asks('urgent'));
Http::fake(); // provider HTTP is plain Laravel HTTP client:
fake it to test failover and *Failed events
```

Checked 2026-10-17 against: <https://laravel.com/docs/13.x/ai-sdk> · <https://github.com/laravel/ai/releases> · <https://github.com/laravel/ai/tree/1.x/src>
Online version with copy buttons: [laravel-vuejs.space/cheatsheets/laravel-ai-sdk](#) · More guides and the monthly newsletter at [laravel-vuejs.space](#)