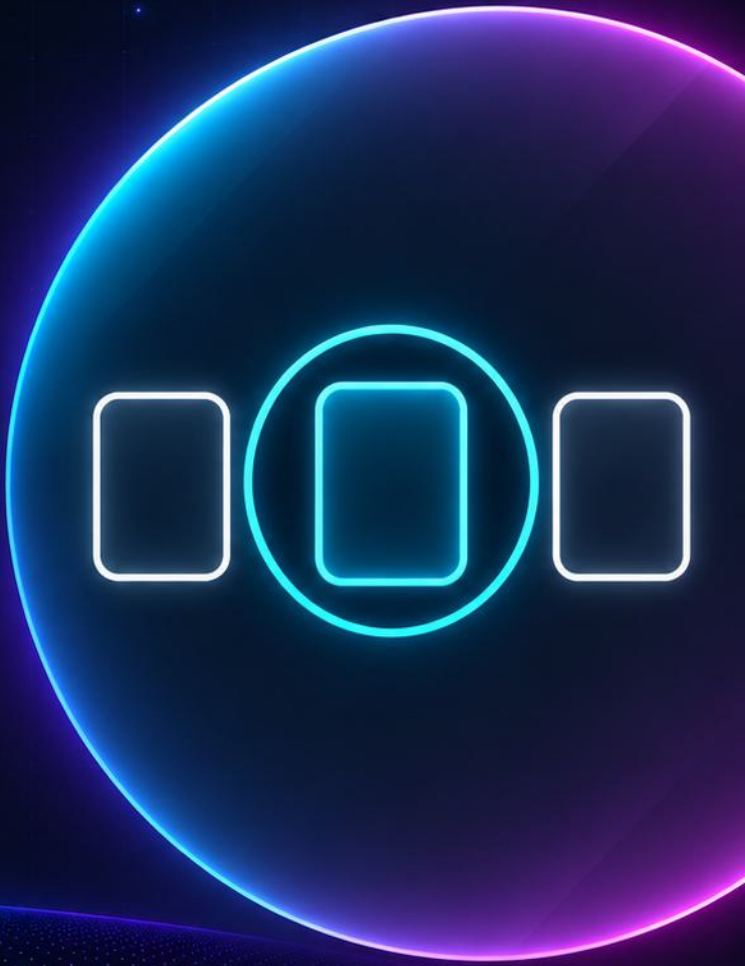
10

PROJET 2 · POINT DE THÉORIE · SÉANCE 2

Choisir sa solution

Thibault Six · IFOSUP Wavre · 2026-2027



Trois familles de solutions : les harness utilisables depuis votre code, les SDK d'agents, les harness des grands laboratoires

Famille	Ce qu'elle apporte	Pour le projet 2
 **Harness utilisables depuis votre code** (pi, OpenCode)	Un agent complet que votre serveur appelle comme une bibliothèque	Une voie possible en JavaScript
 **SDK d'agents** (`laravel/ai`, Vercel AI SDK, OpenAI Agents SDK, PydanticAI)	Boucle d'outils, sortie structurée, approbation, parfois conversations et faux fournisseur	**Le choix conseillé**
 **Harness des grands laboratoires** (Claude Code, Codex)	Les agents les plus capables, en ligne de commande	En démonstration seulement

Certains harness de programmation s'utilisent depuis votre code : pi offre un SDK, et le jeu du cours est construit ainsi

Dans votre serveur, vous lui retirez ses outils de lecture, d'écriture et de shell, vous remplacez son prompt d'agent de code par le vôtre, et vous lui donnez vos outils.





Un SDK d'agent est une bibliothèque dans votre serveur : c'est la famille faite pour votre cas

- **`laravel/ai`** (Laravel 12 ou 13) : le plus complet, conversations en base sans code, **`Agent::fake()`** pour les tests
- **Vercel AI SDK** (paquet **`ai`**, version 7) : boucle, sortie structurée, approbation ; les conversations à stocker vous-même
- **Python :** OpenAI Agents SDK, PydanticAI, Google ADK
- **PHP sans Laravel :** Prism, Neuron AI



Claude Code et Codex restent en démonstration : jamais derrière un produit, jamais avec votre abonnement

- Un processus lancé à chaque requête
- Un agent qui a par défaut le shell et les fichiers de votre serveur
- Une facture que vous plafonnez mal, et un abonnement personnel que les éditeurs interdisent derrière un produit



Dans le projet 2, ils servent à programmer votre application.

À chaque solution, posez deux questions : qui fait tourner la boucle, et où vivent les conversations ?

Question : classez laravel/ai, pi par son SDK et Codex. Qui tient la boucle ? Où sont les messages ?



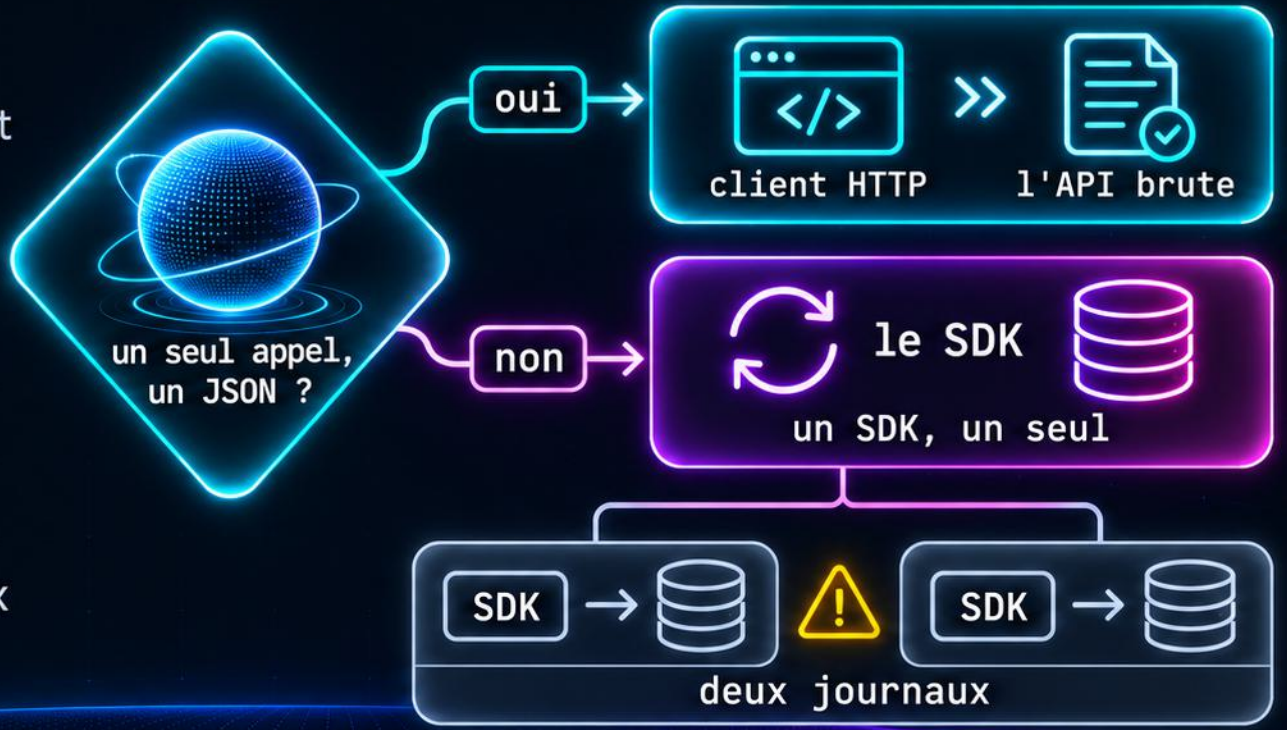
Votre stack décide : deux options par stack, et ce qu'il vous reste à écrire

à écrire vous-même

Stack	Option 1	Option 2	Ce que vous écrivez vous-même
PHP, Laravel	`laravel/ai`	Prism	Le journal (O4) et les quotas ; avec Prism, les tables de conversation
JavaScript, TypeScript	Vercel AI SDK et `@openrouter/ai-sdk-provider`	pi par son SDK	Les tables de conversation, le journal, les quotas
Python	OpenAI Agents SDK	PydanticAI ou Google ADK	Le journal, les quotas, la persistance selon le SDK

Un seul SDK par projet, et l'API brute reste permise pour un seul appel qui rend un JSON

- **Un seul appel, pas d'outil, pas de conversation :** un client HTTP et un schéma suffisent, et vous écrivez la validation, le journal et le faux fournisseur
- **Plusieurs outils et un historique :** un SDK
- **Deux bibliothèques d'agent,** c'est deux façons de stocker les messages et deux journaux à réconcilier

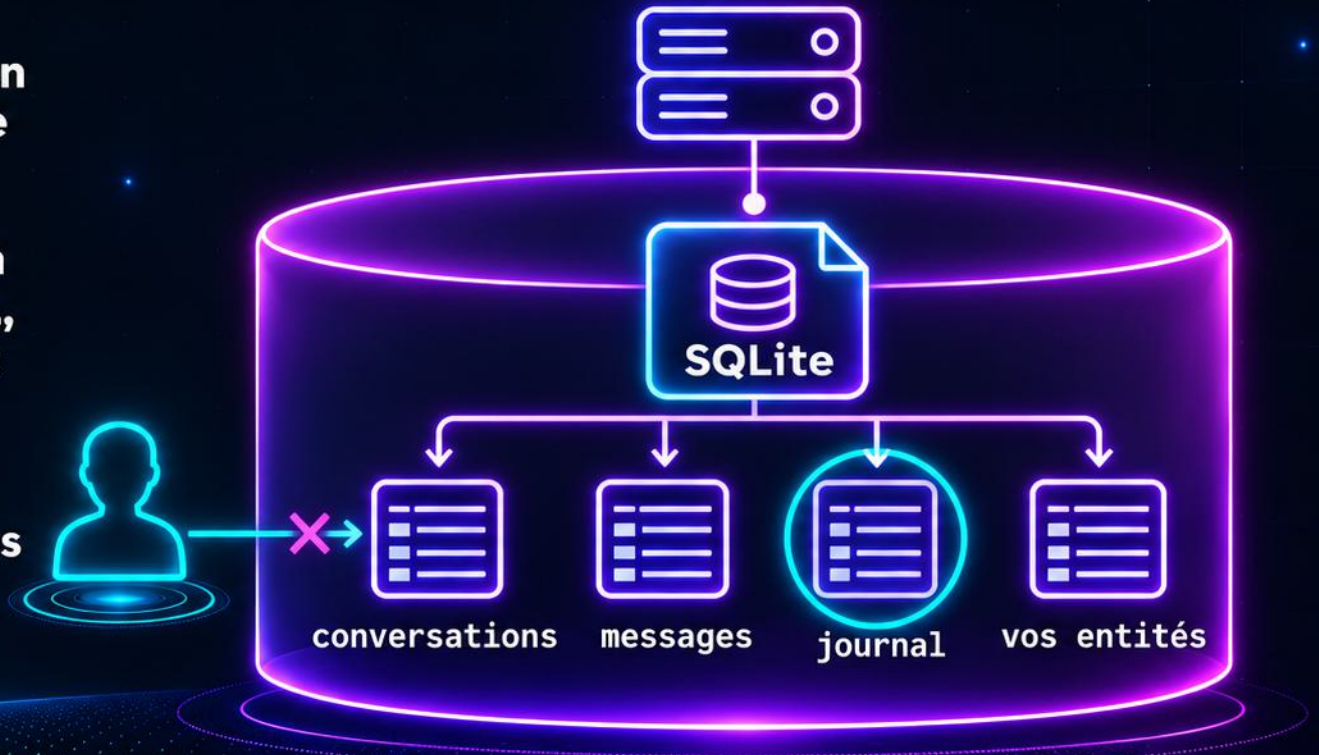


Deux POC, un par famille conseillée : l'expédition avec pi, le gîte avec laravel/ai

	L'expédition	Le gîte
Famille	Harness utilisé depuis le code (pi par son SDK)	SDK d'agent (`laravel/ai`)
Stack	Nuxt, SQLite	Laravel, SQLite
Ce que fait le modèle	Écrit le scénario, les événements et les répliques d'un personnage	Rédige une réponse à un voyageur et en extrait une demande de réservation
Ce que fait le code	Applique les règles du jeu, borne les effets, plafonne la dépense	Recalcule prix et disponibilité, attend le clic du propriétaire

SQLite suffit : un fichier dans un volume persistant, pour les conversations, les messages et le journal

- **Un fichier, aucun service en plus, des transactions, une sauvegarde par copie**
- **Si votre projet tourne déjà sur MySQL ou PostgreSQL, gardez-les :** les tables sont les mêmes
- **Supprimer un compte supprime ses conversations (O7) ; le journal garde ses lignes pour vos totaux**



Trois pièges de persistance : le volume oublié, plusieurs serveurs, l'identifiant qui vient de l'URL



Un **volume oublié** dans Coolify : toutes les conversations disparaissent au déploiement suivant

SQLite suppose **un seul serveur**

L'identifiant d'une conversation ne vient **jamais** de l'URL ni du prompt : il est en base, lié à l'utilisateur, et le serveur vérifie (O3)



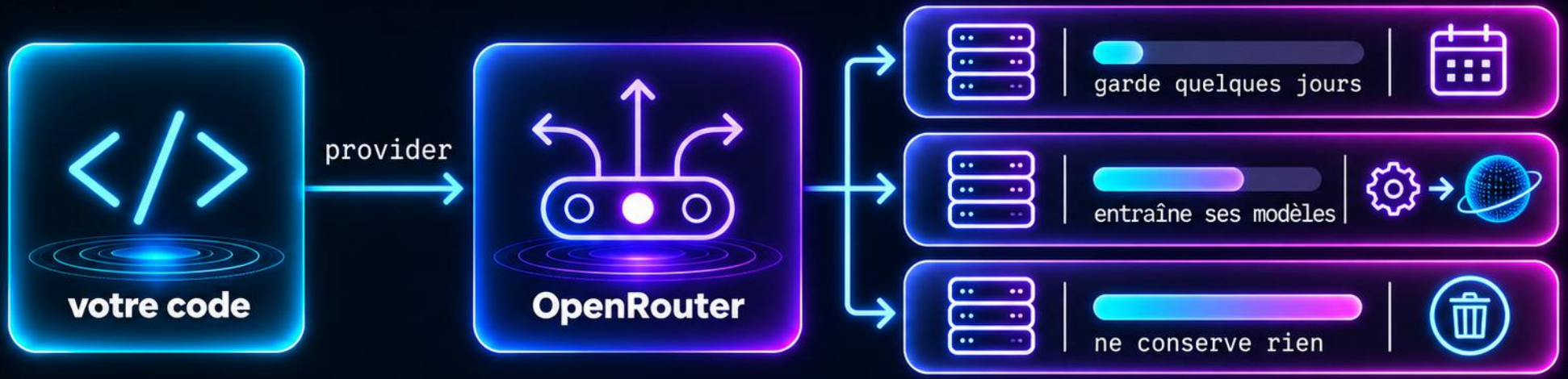
Deux modèles conseillés : DeepSeek V4.1 Flash, moins cher au catalogue, et GPT-6 Luna, qui combine outils et sortie structurée



	DeepSeek V4.1 Flash	GPT-6 Luna
Prix catalogue, $ par million de jetons	0,045 en entrée, 0,50 en sortie	0,10 en entrée, 0,50 en sortie
Outils et sortie structurée **dans la même requête**	⚠ non : le modèle n'a jamais appelé son outil	✓ oui
Raisonnement réglable	oui	oui

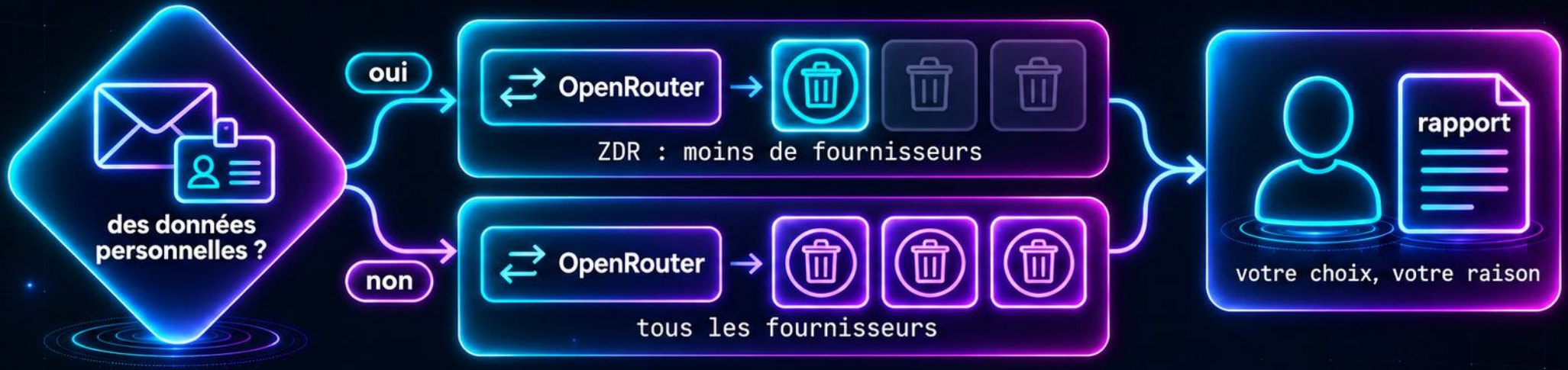
À l'usage, DeepSeek compte deux fois plus de jetons pour la même requête : mesurez avec votre prompt avant de choisir.

Derrière un même modèle, OpenRouter choisit un fournisseur : le prix peut tripler, et la conservation des données change



- Certains fournisseurs gardent vos prompts quelques jours, d'autres s'en servent pour entraîner, d'autres ne conservent rien
- Le champ `provider` de la requête restreint le choix

La ZDR garantit qu'un fournisseur ne conserve rien : à chaque groupe de décider s'il en a besoin



- **ZDR** (*zero data retention*) : rien n'est gardé une fois la réponse renvoyée
- `"zdr": true` et `"data_collection": "deny"` : utiles quand vos utilisateurs écrivent des données personnelles ou des messages de clients
- En contrepartie, moins de fournisseurs : le prix, la vitesse et les sorties structurées peuvent changer
- Écrivez votre choix et sa raison dans le rapport

Une requête réglée : un raisonnement bas, un plafond de jetons, et le fournisseur seulement si vous avez choisi la ZDR

chat/completions

```
{
  "model": "openai/gpt-6-luna",
  "messages": [ … ],
  "tools": [ … ],
  "response_format": { "type": "json_schema", "json_schema": { … } },
  (1) "reasoning_effort": "low",
  (2) "max_tokens": 800,
  (3) "provider": { "zdr": true, "data_collection": "deny" }
}
```



Votre SDK expose ces réglages sous ses propres noms. Dans la réponse, `usage` donne les jetons et le coût facturé.

Six pièges, six parades : à chaque piège répond une règle du code, jamais une consigne au modèle

Piège	La parade
⚠ **Laisser le modèle calculer**	✓ Il propose des valeurs ; votre code calcule, compare, arrondit, avec la base
⚠ **Un prompt système trop long,** renvoyé à chaque appel	✓ Des consignes courtes et stables, que le cache rend moins chères
⚠ **Une boucle d'outils sans limite,** qui vide le crédit en une soirée	✓ Un plafond d'étapes, un plafond de jetons, un quota par utilisateur
⚠ **Pas de délai :** l'utilisateur recharge, vous payez deux fois	✓ Un délai par appel, un message clair, la saisie conservée (O5)
⚠ **Pas de faux fournisseur :** chaque test coûte	✓ `Agent::fake()`, une fausse session pi, ou un faux client derrière une interface
⚠ **Des secrets dans le dépôt**	✓ Une variable d'environnement, un `.env.example` sans valeur, la clé révoquée au moindre doute

Le faux fournisseur sert vos tests et vos démonstrations : les quatre cas d'erreur de O5 à la demande

Un réglage de l'administrateur le branche à la place du vrai, et chaque cas se montre : délai dépassé, panne du fournisseur, crédit épuisé, réponse inexploitable.



Décidez en quinze minutes, en binôme : neuf cases à cocher dans votre cahier des charges

- ☐ **Phrase O1 :** Y et Z séparés
- ☐ **Famille et solution :** le paquet, sa version, sa page sur OpenRouter
- ☐ **Outils :** lit-il ou agit-il ? Brouillon ou approbation pour ceux qui agissent
- ☐ **Persistance :** où vivent les conversations, dans quel volume
- ☐ **Modèle :** testé avec **votre** combinaison
- ☐ **Fournisseur :** ZDR ou non, et pourquoi
- ☐ **Secrets :** la clé dans une variable d'environnement
- ☐ **Journal :** la table de O4, une ligne par requête, coût compris
- ☐ **Budget et tests :** le coût d'une action type, les plafonds, le faux fournisseur



Le livrable avant la revue n°1 : un appel réel au modèle, depuis votre serveur en production, journalisé

Une route de votre application appelle OpenRouter avec la solution choisie, affiche une sortie validée, et écrit une ligne dans le journal : jetons, coût, durée, succès. Le 24 novembre, vous déclenchez l'appel en direct et vous ouvrez le journal juste après.

