Jev (TypeSafe AI) *mémo*

L'essentiel pour utiliser **Jev**, le modèle de décision de TypeSafe AI : on lui envoie un **`state`** et des questions typées, il renvoie des choix, des notes et des probabilités, sans générer de texte. Voir aussi **les avis sur Jev**.

Famille Modèles · **Éditeur** TypeSafe AI · **Licence** Propriétaire (modèle) · MIT (SDK, skill)
Note du wiki 9/15 · **Commandes vérifiées le** 25 septembre 2026
Fiche, notes et avis aitoolswiki.viellard.fr/outils/modeles/jev

Accès

Voie	Identifiant du modèle	Remarque
API TypeSafe (console)	**`jev-latest`**	Clé dans **`TYPESAFE_API_KEY`**. Inscriptions suspendues depuis le 22 septembre 2026.
Playground	–	Pour tester sans code : coller un texte, ajouter des questions.
Vercel AI Gateway	**`typesafe-ai/jev`**	Clé **`AI_GATEWAY_API_KEY`**, base URL **`https://ai-gateway.vercel.sh/typesafe`**.
OpenRouter	**`~typesafe/jev-latest`**	Clé **`OPENROUTER_API_KEY`**, base URL **`https://openrouter.ai/api`**.
Cloudflare Workers AI	**`typesafe/jev`**	**`env.AI.run('typesafe/jev', { state, questions })`**.

Installation

SDK Python (Python ≥ 3.10)	
`pip` `install typesafe-sdk`	ou : uv add typesafe-sdk
`npm` `install @typesafe-ai/sdk`	SDK JavaScript / TypeScript (Node ≥ 20)
`export` `TYPESAFE_API_KEY="..."`	
`claude` `plugin marketplace add typesafe-ai/skills` **`claude`** `plugin install typesafe@typesafe-ai`	Skill pour agents de code : Claude Code
`npx` `skills add typesafe-ai/skills --skill typesafe-ai`	Skill pour les autres agents (Codex, Cursor…)
Intégrations tierces	
`llm` `install llm-typesafe && llm keys set typesafe`	CLI LLM de Simon Willison
`pip` `install "pydantic-ai-slim[typesafe]"`	Pydantic AI

Les trois questions

Type	Question posée	`criteria`	Réponse
`noul`	L'énoncé est-il vrai ?	Optionnel : `{"true": "…", "false": "…"}`	**`noul`** : probabilité de « oui », de 0 à 1 (pas de **`confidence`**)
`choice`	Quelle option parmi une liste ?	Dictionnaire **`option`** → **`description`** (ou **`null`**), 255 options au plus	**`choice`**, **`probabilities`**, **`confidence`**
`score`	Où se situe le texte sur une échelle ?	Liste ordonnée de 2 à 10 niveaux décrits	**`score`** (peut tomber entre deux niveaux), **`legend`**, **`probabilities`**, **`confidence`**

Toutes les questions d'une requête voient le même **`state`** et sont évaluées en parallèle. Les clés choisies pour les questions ne sont pas transmises au modèle.

Appel HTTP

```
curl -X POST https://api.typesafe.ai/v1/systemone \
  -H "Authorization: Bearer $TYPESAFE_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "state": "My Stripe integration has been failing for 3 days and I am losing sales.",
    "model": "jev-latest",
    "questions": {
      "department": {"type": "choice", "instructions": "Which team should handle this?",
        "criteria": {"billing": "Payment issues", "technical": "Bugs or integrations", "sales": "Pricing"}},
      "frustration": {"type": "score", "instructions": "How frustrated is the customer?",
        "criteria": ["Calm", "Frustrated but civil", "Very angry"]},
      "is_urgent": {"type": "noul", "instructions": "The message conveys urgency"}
    }
  }'
```

`curl` `https://api.typesafe.ai/v1/models -H "Authorization: Bearer $TYPESAFE_API_KEY"`	modèles et alias

Python

```
from typesafe_sdk import Choice, Noul, Score, TypeSafeClient

with TypeSafeClient() as client:            # lit TYPESAFE_API_KEY, modèle jev-latest par défaut
    r = client.system_one(
        state={"ticket": "I was charged twice. Please fix this ASAP."},
        questions={
            "billing": Noul(instructions="Is this ticket about billing?"),
            "tone": Choice(instructions="What is the customer's tone?",
                           criteria={"calm": None, "frustrated": None, "angry": None}),
            "urgency": Score(instructions="How urgent is this ticket?",
                             criteria=["can wait", "this week", "today"]),
        },
    )

r.nouls["billing"].noul          # float de 0 à 1
r.choices["tone"].choice         # "calm" | "frustrated" | "angry"
r.choices["tone"].confidence     # float de 0 à 1
r.scores["urgency"].score        # position de 0 à 2, ex. 1.4 entre deux niveaux
```

Option	Usage
`AsyncTypeSafeClient()`	Client asynchrone (**`await`** `client.system_one(...)`).
`TypeSafeClient(model="jev-1.13.0")`	Figer une version plutôt que l'alias.
`system_one(...,` `response_model=MonModele)`	Réponse typée par un modèle Pydantic.
`TypeSafeClient(retry=RetryPolicy(max_retries=3,` `timeout=1.0))`	Politique de retry (aussi par appel).
`TYPESAFE_BASE_URL`, **`TYPESAFE_DEFAULT_MODEL`**	Passer par une passerelle ou changer le modèle par défaut.

JavaScript / TypeScript

```
import { choice, noul, score, TypeSafeClient } from "@typesafe-ai/sdk";

const client = new TypeSafeClient();
const r = await client.systemOne({
  state: { ticket: "I was charged twice. Please fix this ASAP." },
  questions: {
    category: choice("What is this ticket about?", { billing: null, technical: null, other: null }),
    urgency: score("How urgent is it?", ["can wait", "this week", "today"]),
    refund: noul("Does the customer ask for a refund?"),
  },
});
console.log(r.answers.category.choice);   // types inférés depuis les questions
```

Modèle et limites (jev-1.13)

Élément	Valeur
Alias	`jev-latest` et `jev-preview` → `jev-1.13.0`
Prix	0,042 $ par million de tokens en entrée, sortie gratuite
Latence annoncée	70 à 500 ms de bout en bout
Contexte	64k tokens par requête ; 32k pour le `state` plus la plus longue question
Débit	250 000 tokens/s et 1 200 requêtes/min, ajustables sans préavis (erreur `429`)
Entrées	Texte uniquement : chaîne, objet JSON ou tableau. Meilleure précision en anglais.
Personnalisation	Aucun fine-tuning : tout passe par le `state`, les `instructions` et les `criteria`

Patterns recommandés

Pattern	Principe
Fan-out spéculatif	Poser toutes les questions utiles en une requête, y compris celles qui ne servent que dans certaines branches, puis trier dans le code.
Routage par confiance	L'`answer` dit quoi faire, la `confidence` dit s'il faut le faire. Seuils plus hauts pour les actions risquées.
Score composite	Découper un jugement complexe en scores atomiques et les pondérer dans le code.
Routage d'intention	Classer chaque requête puis l'envoyer vers du code déterministe, un LLM spécialisé ou un humain.
Cascade	Jev tranche les cas nets, les cas à faible confiance remontent vers un LLM de raisonnement.

```
a = r.choices["action"]                  # une question Choice nommée
"action"
if a.confidence < 0.6:
    route_to_human()                     # le modèle doute : ne pas
deviner
elif a.choice == "approve_transfer" and a.confidence < 0.85:
    ask_user_to_confirm()                # action risquée, confiance
moyenne
else:
    execute(a.choice)
```

Ce qu'il ne faut pas demander à Jev

Piège connu	Faire plutôt
Consignes implicites (Jev lit au pied de la lettre)	Écrire la condition exacte et décrire les cas limites dans les `criteria`.
Calculs, comptages, valeurs numériques	Calculer dans le code ; poser une question par élément puis additionner.
Comparer des dates	Extraire jour, mois et année par des `choice`, comparer dans le code.
Raisonnements à plusieurs étapes, doubles négations	Réduire les étapes et désigner la partie du `state` concernée.
Gros `state` plein de détails inutiles	Filtrer d'abord (au besoin avec un `noul` de pertinence).
Contenu adverse ou instructions injectées	Critères explicites, tests des cas limites avant déploiement.
Consigne et critères contradictoires	Aligner `instructions` et `criteria`.
Supposer que `noul` et `1 - noul(négation)` concordent	Formuler directement la question voulue ; ne pas réutiliser un seuil d'un type de question à l'autre.
Générer du texte	Utiliser un LLM ; pour l'extraction, proposer des candidats (regex, LLM) et faire choisir Jev.

Intégrations tierces

Outil	Utilisation
LiteLLM Auto Router	`classifier_type: jev` avec une clé TypeSafe, pour choisir le niveau de modèle de chaque requête.
Pydantic AI	`Agent('typesafe:jev-latest', output_type=bool, instructions='Is this request harmful?')`
LLM (CLI)	`llm -m jev 'Please refund my last payment.' -s 'Does this message request a refund?' ; -o answer_type choice -o criteria '{…}'`
Cloudflare Workers AI	`await env.AI.run('typesafe/jev', { state, questions })`
`system-one-adapter-python`	Même interface que `TypeSafeClient` mais adossée à OpenAI, Anthropic ou Gemini, pour comparer sur ses propres données.

Conseils d'utilisation

- **Une question = un jugement.** Décomposer : dans un test de phishing, Jev passe de 62,6 % avec une seule question à 95 % avec cinq questions pondérées.
- **Figer la version** (`jev-1.13.0`) dès que des seuils de confiance ont été réglés : l'alias `jev-latest` peut changer de modèle sans prévenir.
- **Mesurer sa propre calibration** en faisant tourner Jev en parallèle de l'existant avant d'automatiser quoi que ce soit.
- **Décrire des situations, pas des degrés** dans les niveaux d'un `score` : Jev évalue chaque niveau séparément et ne voit pas leur numéro.
- **Garder un fournisseur de repli** (Vercel, OpenRouter, Cloudflare) : l'accès direct et les limites de débit ont beaucoup varié depuis le lancement.
- **Éviter les décisions sensibles sur des personnes** (recrutement, notation d'individus) : les réponses ne s'expliquent pas et des biais ont été observés.

Sources (documentation officielle)

- Documentation – quick start
- Documentation – référence de l'API
- Documentation – modèles et limites
- Documentation – modes d'échec de jev-1.13
- Documentation – SDK Python, usage
- Documentation – SDK JavaScript
- Documentation – skill pour agents
- Documentation – routage par confiance
- Cloudflare Workers AI – typesafe/jev
- Pydantic AI – modèle TypeSafe
- LiteLLM – Jev dans l'Auto Router
- Simon Willison – llm-typesafe 0.1a0