AYI NEDJIMI Consultants

Pydantic AI et les Frameworks d'Agents Type-Safe en 2026

Catégorie : Intelligence Artificielle Lecture : 10 min Publié le : 15/02/2026 Auteur : Ayi NEDJIMI

Comparatif Pydantic AI, Instructor, Marvin pour des pipelines IA robustes avec validation structurée. Guide expert avec méthodologies et.

Table des Matières

Le parsing artisanal des réponses LLM via regex ou json.loads() est fragile et non maintenable. Les **frameworks type-safe** résolvent ce problème en imposant une validation structurée des sorties via des modèles Pydantic. Trois bibliothèques dominent cet espace en 2026 : **Pydantic AI**, **Instructor** et **Marvin**. Chacune adopte une philosophie distincte pour atteindre le même objectif : transformer les sorties probabilistes des LLM en données déterministes et typées. Comparatif Pydantic AI, Instructor, Marvin pour des pipelines IA robustes avec validation structurée. Guide expert avec méthodologies et. Dans un contexte où l'intelligence artificielle transforme les pratiques de cybersécurité, la maîtrise de ia pydantic ai agents type devient un avantage stratégique pour les équipes techniques. Nous abordons notamment : table des matières, 2 pydantic ai : agents natifs type-safe et 3 instructor : patching transparent des api llm. Les professionnels y trouveront des recommandations actionnables, des commandes prêtes à l'emploi et des stratégies de mise en œuvre adaptées aux environnements d'entreprise.

Enjeu central : En production, une sortie LLM non conforme au schéma attendu provoque des erreurs silencieuses, des corruptions de données et des comportements imprévisibles. Les frameworks type-safe éliminent cette classe entière de bugs en validant chaque réponse contre un modèle Pydantic avant qu'elle n'atteigne le code métier.

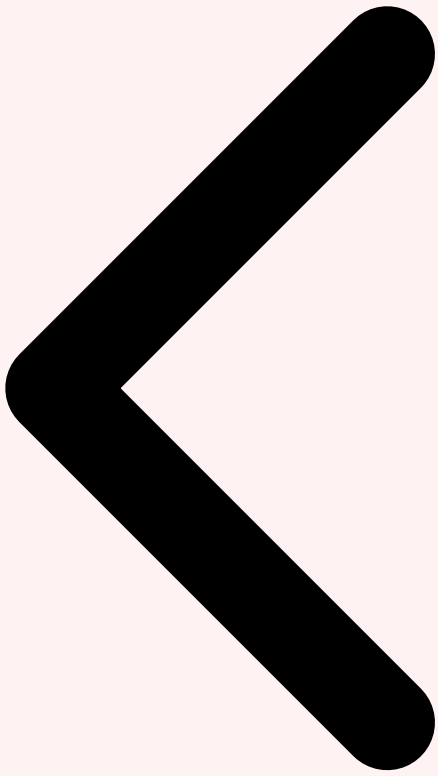
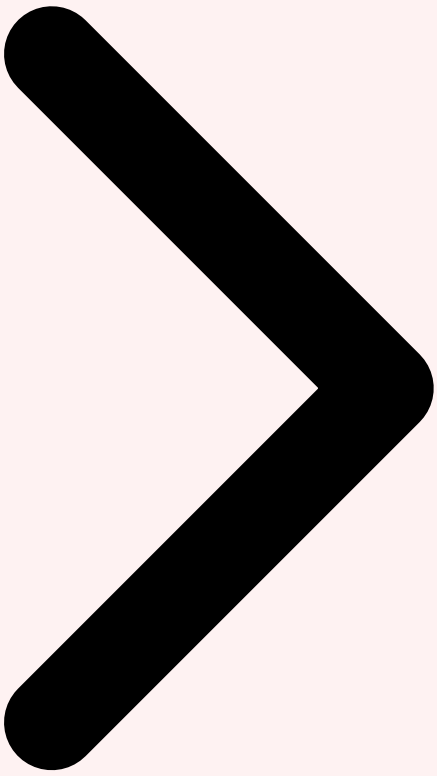


Table des Matières Pydantic AI



Notre avis d'expert
La gouvernance de l'IA est le prochain grand chantier de la cybersécurité. Les attaques par prompt injection, l'empoisonnement de données d'entraînement et l'extraction de modèles sont des menaces concrètes que nous observons de plus en plus lors de nos missions. Ne pas s'y préparer, c'est accepter un risque majeur.

2 Pydantic AI : agents natifs type-safe

Pydantic AI est le framework officiel d'agents développé par l'équipe Pydantic (Samuel Colvin). Lancé fin 2024 et massivement adopté en 2025-2026, il se distingue par son intégration native avec l'écosystème Pydantic v2 et son architecture orientée agents. Contrairement aux alternatives qui se concentrent uniquement sur l'extraction structurée, Pydantic AI propose un framework complet pour construire des **agents conversationnels type-safe** avec gestion de dépendances, outils et historique.

Architecture et concepts clés

L'unité fondamentale est l'**Agent**, qui encapsule un modèle LLM, un type de résultat structuré, des outils et des dépendances injectées. Le système de types est vérifié statiquement par mypy et pyright, garantissant la cohérence à la compilation.

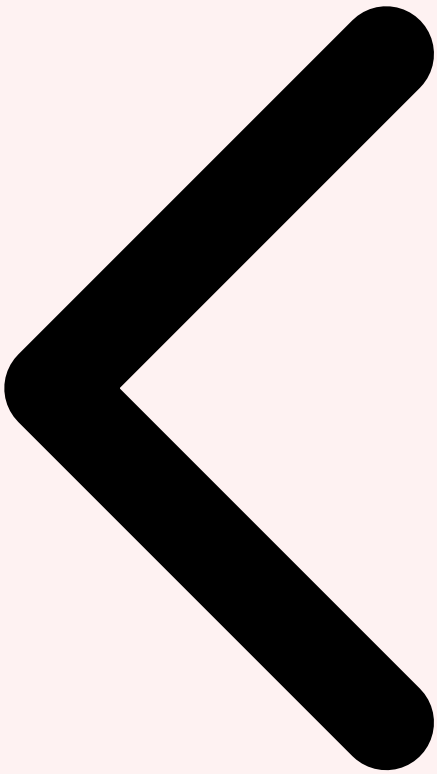
```
from pydantic_ai import Agent
from pydantic import BaseModel

class CityInfo(BaseModel):
    name: str
    country: str
    population: int
    known_for: list[str]

agent = Agent('openai:gpt-4o', result_type=CityInfo)
result = await agent.run('Décris la ville de Lyon')
# result.data est un CityInfo validé par Pydantic
```

Système de dépendances et outils

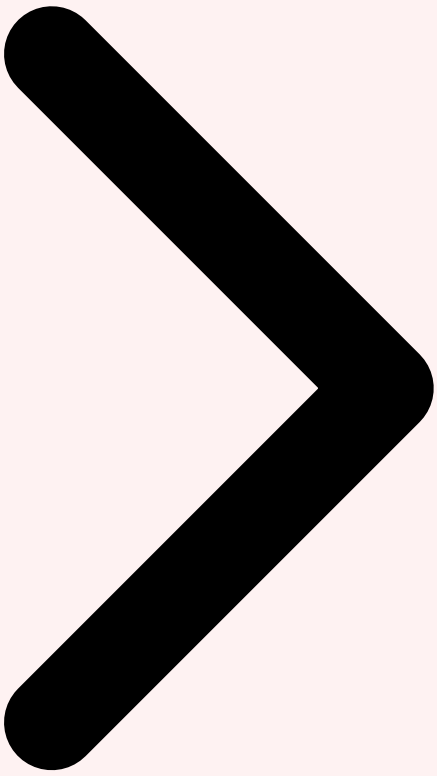
Pydantic AI intègre un système d'**injection de dépendances** inspiré de FastAPI. Les agents reçoivent un contexte typé (connexion base de données, client API, session utilisateur) qui est accessible dans les outils sans couplage global. Ce pattern permet de tester les agents en isolation avec des mocks typés.

```
from dataclasses import dataclass

@dataclass
class Deps:
    db: DatabaseConnection
    user_id: int

agent = Agent('openai:gpt-4o', deps_type=Deps)

@agent.tool
async def get_orders(ctx: RunContext[Deps]) -> list[Order]:
    return await ctx.deps.db.fetch_orders(ctx.deps.user_id)
```

Le support natif du **streaming structuré** permet de recevoir des résultats partiels validés en temps réel, avec validation incrémentale des champs au fur et à mesure de la génération. L'intégration avec **Logfire** (l'outil d'observabilité de Pydantic) offre un tracing complet de chaque appel LLM. Pour approfondir, consultez Apprentissage Fédéré et Privacy-Preserving ML en Cybersécurité.



Introduction Instructor



3 Instructor : patching transparent des API LLM

Instructor, créé par Jason Liu, adopte une approche minimaliste et élégante : plutôt que de construire un framework entier, il **patche les clients LLM existants** (OpenAI, Anthropic, Mistral, Cohere, LiteLLM) pour ajouter la validation Pydantic de manière transparente. L'API reste identique à celle du provider, mais les réponses sont automatiquement validées et retentées en cas d'échec.

```
import instructor
from openai import OpenAI
from pydantic import BaseModel, Field

client = instructor.from_openai(OpenAI())

class ExtractUser(BaseModel):
    name: str
    age: int = Field(ge=0, le=150)
```

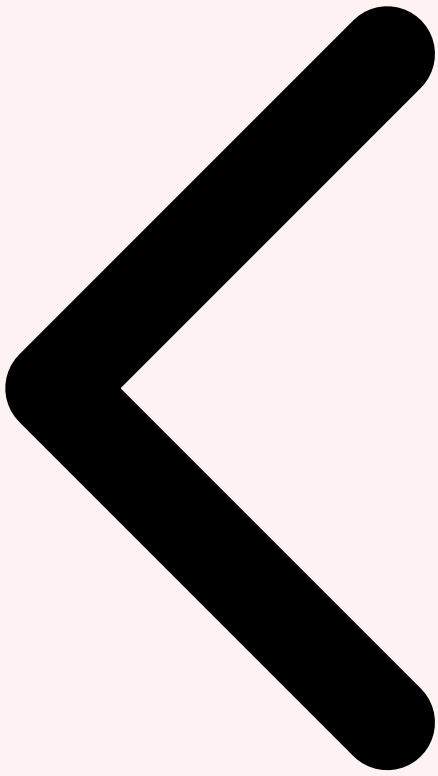
```
user = client.chat.completions.create(
    model="gpt-4o",
    response_model=ExtractUser,
    messages=[{"role": "user", "content": "Jean a 28 ans"}],
)
```

Mécanismes de retry et validation

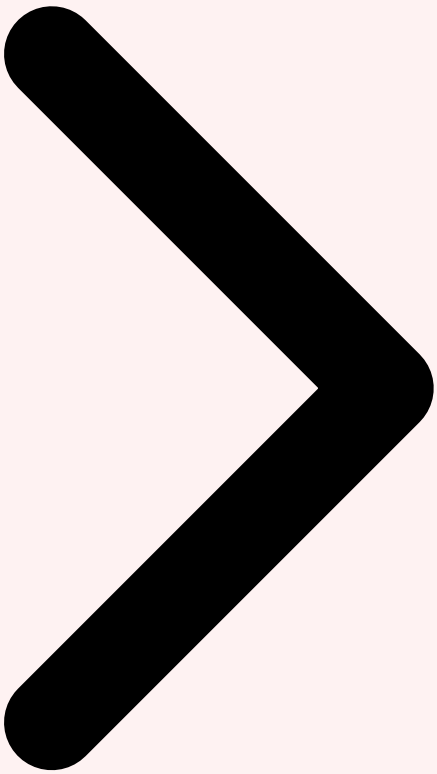
Instructor excelle par son système de **retry automatique avec contexte**. Quand la validation Pydantic échoue, l'erreur de validation est renvoyée au LLM comme contexte additionnel, lui permettant de corriger sa réponse. Le paramètre `max_retries` contrôle le nombre de tentatives. Cette boucle de correction est invisible pour le développeur.

Le framework supporte plusieurs **modes d'extraction** : `TOOLS` (function calling natif), `JSON` (mode JSON du modèle), `MD_JSON` (extraction depuis markdown) et `PARALLEL_TOOLS` pour extraire plusieurs objets simultanément. Le support du streaming partiel permet de valider les champs au fur et à mesure de la génération, idéal pour les interfaces temps réel.

Instructor brille particulièrement pour les cas d'**extraction d'entités** depuis du texte non structuré : parsing de CV, extraction de données financières, classification multi-label avec justification. Sa légèreté en fait le choix privilégié pour les projets qui utilisent déjà directement les SDK des providers.



Pydantic AI Marvin



Cas concret
L'attaque par prompt injection sur les systèmes GPT documentée par OWASP en 2023 a révélé que des instructions malveillantes dissimulées dans des documents pouvaient détourner le comportement de chatbots d'entreprise, accédant à des données internes sensibles sans aucune authentification supplémentaire.

Vos pipelines de données d'entraînement sont-ils protégés contre l'empoisonnement ?

4 Marvin : IA fonctionnelle et décorateurs

Marvin, développé par l'équipe Prefect, adopte une philosophie radicalement différente : faire de l'IA une **primitive du langage Python**. Plutôt que de manipuler des clients et des messages, Marvin expose des fonctions de haut niveau (`marvin.cast`, `marvin.extract`, `marvin.classify`, `marvin.fn`) qui transforment les appels LLM en opérations Python idiomatiques.

import marvin

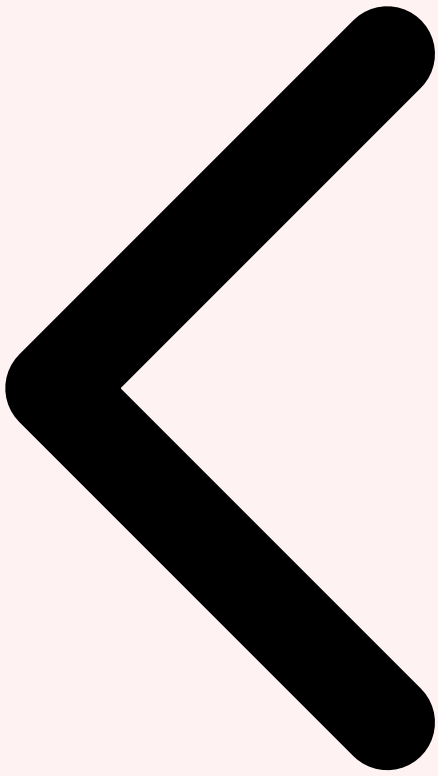
Cast : convertir du texte en type structuré

```
city = marvin.cast("la ville lumière", target=CityInfo)

# Extract : extraire une liste d'entités
entities = marvin.extract("Apple et Google dominent le marché", target=str,
instructions="noms d'entreprises")

# Classify : classification en enum
sentiment = marvin.classify("Ce produit est excellent", labels=["positif", "négatif", "neutre"])

# fn : fonctions IA avec décorateur
@marvin.fn
def summarize(text: str, max_words: int = 50) -> str:
    """Résume le texte en max_words mots."""
```

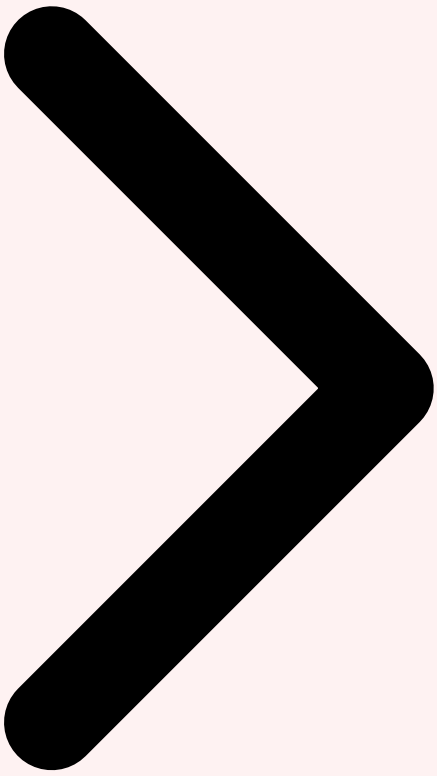
Le décorateur @marvin.fn

Le décorateur `@marvin.fn` est la fonctionnalité signature de Marvin. Il transforme n'importe quelle fonction Python avec une docstring et des annotations de types en un appel LLM. Le corps de la fonction est ignoré -- seules la signature, la docstring et les types de retour sont utilisés pour construire le prompt. Le résultat est validé par Pydantic si le type de retour est un `BaseModel`.

Marvin excelle pour le **prototypage rapide** et les tâches ponctuelles d'extraction, classification ou transformation. Son API fonctionnelle réduit le boilerplate au minimum. Cependant, il offre moins de contrôle sur les prompts, les retries et la gestion multi-provider que Instructor ou Pydantic AI. Pour approfondir, consultez Human-AI Collaboration 2026 : Travailler avec des Agents.



Instructor Comparatif

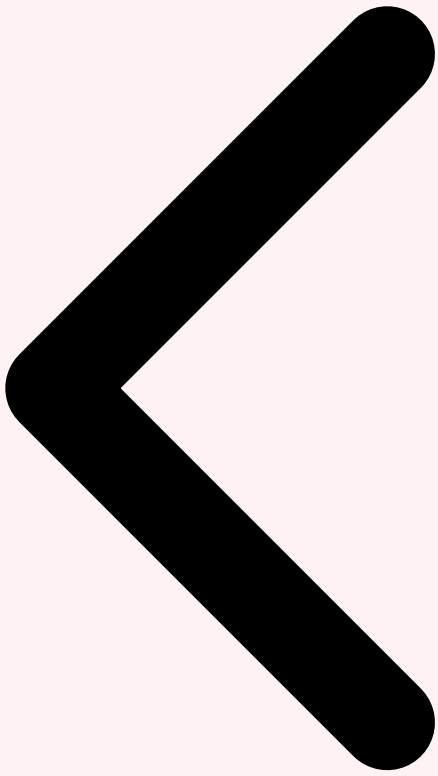


5 Comparatif détaillé

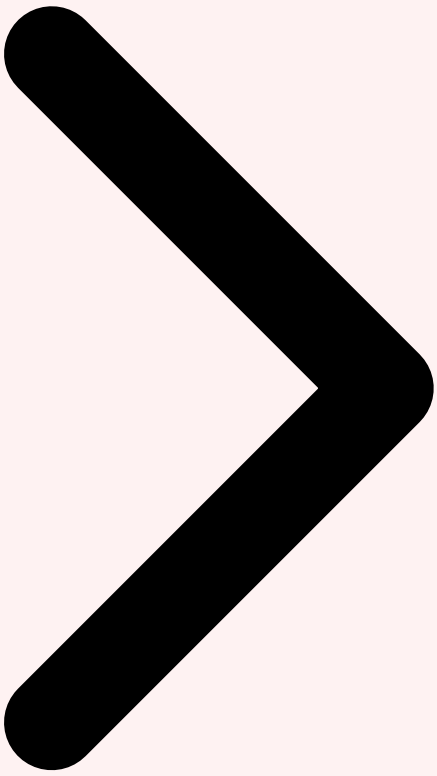
Les trois frameworks partagent une base commune -- Pydantic v2 pour la validation -- mais divergent radicalement dans leur approche architecturale, leur surface d'API et leurs cas d'usage cibles.

Critère	Pydantic AI	Instructor	Marvin
Philosophie	Framework d'agents complet	Patch minimal des SDK	IA comme primitive Python
Courbe d'apprentissage	Moyenne	Faible	Très faible
Multi-provider	OpenAI, Anthropic, Gemini, Groq, Mistral, Ollama	20+ providers via LiteLLM	OpenAI principalement
Agents / Outils	Natif (Agent, tools, deps)	Non (extraction uniquement)	Basique (marvin.fn)
Streaming structuré	Oui, natif	Oui (Partial)	Non
Retry intelligent	Oui	Oui (avec contexte d'erreur)	Limité
Observabilité	Logfire natif	Via hooks	Prefect intégration
Cas d'usage idéal	Agents de production	Extraction structurée	Prototypage rapide

Règle de choix : Si vous construisez des **agents autonomes** avec outils et état, choisissez Pydantic AI. Si vous ajoutez de la **validation structurée à un projet existant** utilisant directement les SDK, choisissez Instructor. Si vous avez besoin de **fonctions IA ponctuelles** avec un minimum de code, choisissez Marvin.



Marvin Intégration agents



6 Intégration avec les systèmes d'agents

En 2026, les architectures **multi-agents** se généralisent. Les frameworks type-safe s'intègrent à différents niveaux dans ces architectures, depuis la validation des sorties individuelles jusqu'à l'orchestration complète des agents.

Pydantic AI comme orchestrateur d'agents

Pydantic AI permet de construire des **graphes d'agents** où chaque agent possède un type de résultat distinct. Un agent de routage peut déléguer à des agents spécialisés, chacun avec ses propres outils et dépendances. Le résultat est un pipeline entièrement typé de bout en bout, où les erreurs de type sont détectées avant l'exécution.

```
# Agent de routage avec délégation type-safe
from pydantic_ai import Agent
from pydantic import BaseModel
from typing import Union
```

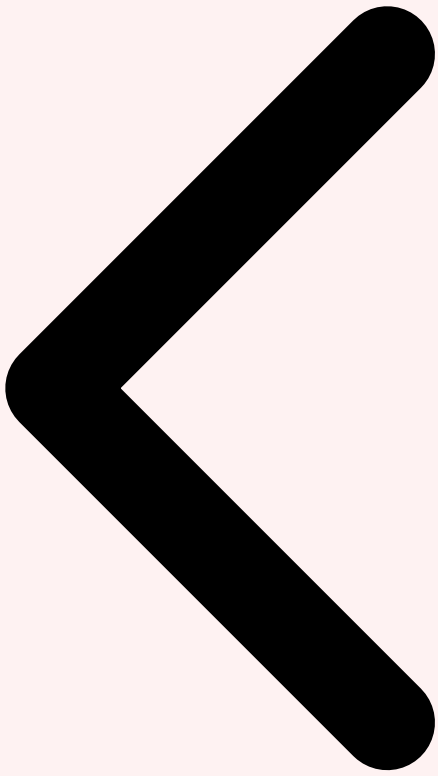
```
class TechQuery(BaseModel):
    domain: str
    question: str

class SecurityReport(BaseModel):
    severity: str
    findings: list[str]
    recommendations: list[str]

router = Agent('openai:gpt-4o', result_type=TechQuery)
security_agent = Agent('anthropic:claude-sonnet', result_type=SecurityReport)
```

Instructor dans les pipelines LangGraph

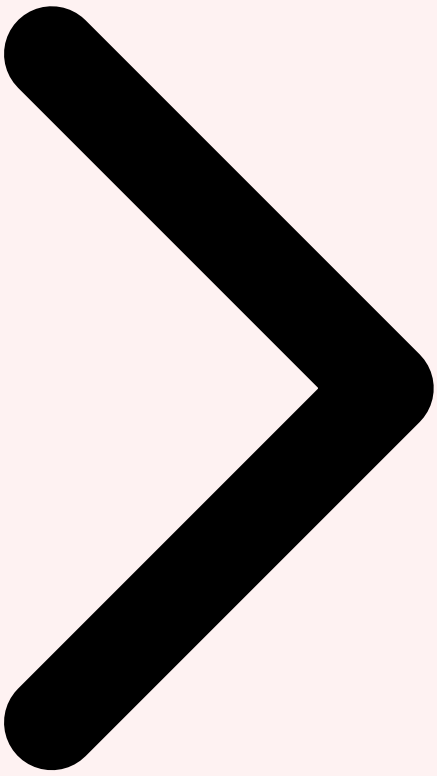
Instructor s'intègre naturellement dans les **pipelines LangGraph** comme couche de validation. Chaque noeud du graphe peut utiliser un client patché par Instructor pour garantir la structure de ses sorties. Cette approche permet de combiner l'orchestration de LangGraph avec la rigueur de validation d'Instructor, sans imposer un framework d'agents spécifique.

Marvin et les workflows Prefect

Marvin, issu de l'écosystème **Prefect**, s'intègre nativement dans les workflows d'orchestration de données. Les fonctions `@marvin.fn` peuvent être encapsulées dans des `@task` Prefect, bénéficiant automatiquement du retry, du logging et du monitoring de la plateforme. Cette synergie est particulièrement adaptée aux pipelines ETL augmentés par l'IA.



Comparatif Patterns production



7 Patterns de production

Déployer des pipelines IA type-safe en production exige des patterns spécifiques pour gérer la latence, les erreurs et la montée en charge. Voici les pratiques éprouvées en 2026. Pour approfondir, consultez Architectures Multi-Agents et Orchestration LLM en Production.

Validation multi-couche

La validation ne doit pas reposer uniquement sur Pydantic. Un pattern robuste combine trois couches : les **validators Pydantic** pour la structure et les types, les **validators métier** (via `@field_validator` et `@model_validator`) pour les règles business, et une **validation sémantique** optionnelle par un second LLM pour vérifier la cohérence du contenu.

```
from pydantic import BaseModel, field_validator

class InvoiceExtract(BaseModel):
    vendor: str
    total_ht: float
```

```
    tva_rate: float
    total_ttc: float

    @field_validator('tva_rate')
    def validate_tva(cls, v):
        if v not in [5.5, 10.0, 20.0]:
            raise ValueError(f"Taux TVA invalide: {v}")
        return v
```

Gestion du fallback multi-modèle

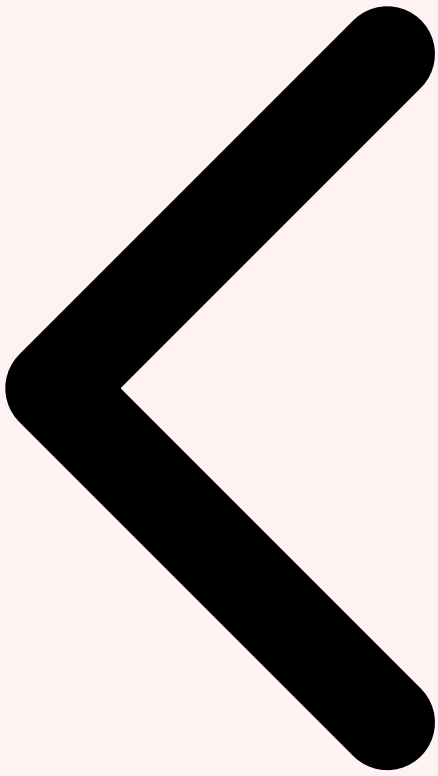
En production, un seul provider ne suffit pas. Le pattern **fallback chaîné** tente l'extraction avec un modèle principal (GPT-4o), bascule sur un modèle secondaire (Claude Sonnet) en cas d'échec de validation, puis sur un modèle local (Ollama/Llama) en dernier recours. Pydantic AI et Instructor supportent ce pattern via la configuration de modèles multiples.

Cache et déduplication

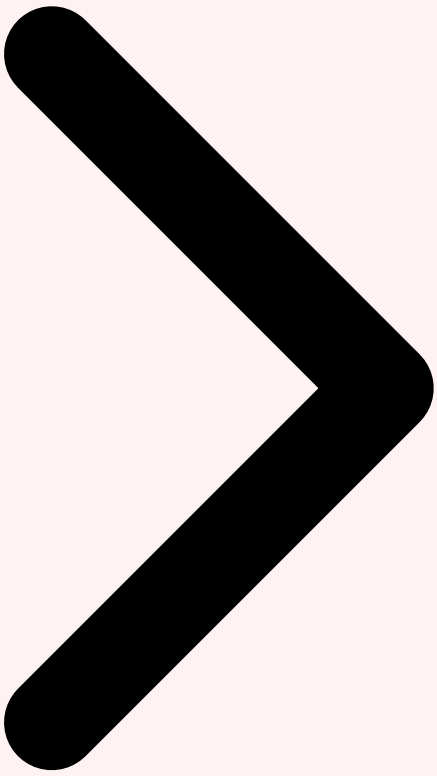
Les appels LLM sont coûteux. Un cache basé sur le **hash du prompt + modèle + schéma** élimine les appels redondants. Redis ou DiskCache stockent les résultats validés avec un TTL adapté au cas d'usage. Pour les extractions déterministes (parsing de factures, extraction d'entités), un TTL long (24h+) réduit drastiquement les coûts.

Tests et CI/CD

Le typage statique des trois frameworks permet d'intégrer **mypy/pyright** dans la CI. Les tests unitaires utilisent des réponses mockées pour valider la logique de parsing sans appels LLM. Les tests d'intégration vérifient la compatibilité avec les API réelles sur un sous-ensemble représentatif de cas. Ce double niveau garantit à la fois la rapidité de la CI et la fiabilité en production.



Intégration agents Conclusion



8 Conclusion et recommandations

L'écosystème des frameworks type-safe pour LLM a atteint une maturité significative en 2026. La validation structurée n'est plus optionnelle -- c'est un **prérequis pour tout déploiement en production**. Les trois frameworks analysés offrent des approches complémentaires pour résoudre le même problème fondamental : transformer les sorties probabilistes des LLM en données fiables et typées.

Pydantic AI s'impose comme le choix de référence pour les applications nécessitant des agents complets avec outils, dépendances et observabilité. Son intégration native avec l'écosystème Pydantic et Logfire en fait la solution la plus cohérente pour les projets greenfield. **Instructor** reste incontournable pour les projets existants qui souhaitent ajouter la validation structurée sans refactoring majeur -- son approche par patching est élégante et non invasive. **Marvin** excelle dans le prototypage et les tâches d'extraction ponctuelles où la concision du code prime.

Les trois frameworks convergent vers un avenir où chaque interaction avec un LLM sera automatiquement validée, typée et traçable. Les recommandations concrètes : Pour approfondir, consultez ROI de l'IA Générative : Mesurer l'Impact Réel.

- 1.**Adopter la validation structurée dès le premier prototype** -- le coût de migration ultérieure est toujours supérieur au coût d'intégration initiale
- 2.**Utiliser des validators métier** dans les modèles Pydantic pour encoder les invariants business directement dans le schéma
- 3.**Implémenter le fallback multi-modèle** pour garantir la disponibilité, en particulier sur les chemins critiques
- 4.**Activer le typage statique** (mypy strict) dans la CI pour détecter les incompatibilités de schéma avant le déploiement
- 5.**Monitorer les taux de retry** -- un taux élevé signale un prompt mal calibré ou un schéma trop contraignant pour le modèle utilisé

Besoin d'un accompagnement expert ?

Nos consultants en cybersécurité et IA vous accompagnent dans l'implémentation de pipelines IA type-safe avec Pydantic AI, Instructor et Marvin. Devis personnalisé sous 24h.

Références et ressources externes

- OWASP LLM Top 10 — Les 10 risques majeurs pour les applications LLM
- MITRE ATLAS — Framework de menaces pour les systèmes d'intelligence artificielle
- NIST AI RMF — AI Risk Management Framework du NIST
- arXiv — Archive ouverte de publications scientifiques en IA
- HuggingFace Docs — Documentation de référence pour les modèles de ML

Pour approfondir ce sujet, consultez notre outil open-source ai-prompt-injection-detector qui facilite la détection des injections de prompt.

Sources et références : ArXiv IA · Hugging Face Papers

FAQ

Qu'est-ce que Pydantic AI et les Frameworks d'Agents Type-Safe en 2026 ?

Le concept de Pydantic AI et les Frameworks d'Agents Type-Safe en 2026 est détaillé dans les premières sections de cet article, qui couvrent les fondamentaux, les enjeux et le contexte opérationnel. Pour un accompagnement sur ce sujet, contactez nos experts.

Pourquoi Pydantic AI et les Frameworks d'Agents Type-Safe en 2026 est-il important en cybersécurité ?

La compréhension de Pydantic AI et les Frameworks d'Agents Type-Safe en 2026 permet aux équipes de sécurité d'améliorer leur posture défensive. Les sections « Table des Matières » et « 2 Pydantic AI : agents natifs type-safe » détaillent les raisons de cette importance. Pour un accompagnement sur ce sujet, contactez nos experts.

Comment mettre en œuvre les recommandations de cet article ?

Les recommandations pratiques sont détaillées tout au long de l'article, avec des commandes, des outils et des méthodologies éprouvées. La section « Conclusion » fournit une synthèse actionnable. Pour un accompagnement sur ce sujet, contactez nos experts.

Conclusion

Cet article a couvert les aspects essentiels de Table des Matières, 1 Introduction : le problème des sorties non structurées, 2 Pydantic AI : agents natifs type-safe. La mise en pratique de ces recommandations permet de renforcer significativement la posture de securite de votre organisation.

Ayi NEDJIMI Consultants — Expert cybersécurité offensive & intelligence artificielle

ayinedjimi-consultants.fr · ayi@ayinedjimi-consultants.fr