

Tutoriel

Évaluation · LLM · Qualité

Évaluer les hallucinations d'un LLM

Construire un mini-jeu d'évaluation avec réponses de référence, questions hors contexte et critères explicites de factualité.

Auteur **Youness Boutyour**

**DURÉE**

55 minutes

NIVEAU

Intermédiaire

FORMAT

Tutoriel pratique

Évaluation · LLM · Qualité · publié le 2026-08-11 · mock

Objectifs et livrables

Une réponse peut paraître convaincante tout en s'éloignant des documents fournis. Pour repérer ce problème et comparer les versions d'un assistant, nous allons construire un petit jeu d'évaluation que vous pourrez relancer après une modification du prompt ou du modèle.

Nous préparerons dix questions : six cas directement couverts par le corpus, deux paraphrases et deux questions hors contexte. Vous produirez les réponses, écrirez les règles de comparaison et examinerez les verdicts. Cette variété permet de regarder à la fois les réponses retrouvées, la sensibilité aux formulations et la capacité à reconnaître une information absente.

Nous lirons ensuite les échecs un par un. Une réponse manquante, une formulation différente de la référence et une affirmation sans source appellent des analyses différentes. Le score sera donc accompagné des questions, des réponses et des raisons du verdict, afin de pouvoir comprendre ce qui l'explique.

Vous obtiendrez [outputs/evaluation.json](#), avec un résultat global et le détail des cas. Ce premier outil de régression restera lié au corpus et aux règles du tutoriel ; il vous donnera une base concrète pour enrichir vos tests et suivre les changements de comportement.

Passer d'une impression à un verdict

Schéma de l'évaluation qui compare une réponse à une référence, distingue les cas hors contexte et calcule un score.



● référence explicite · verdict détaillé · régression

evaluer-hallucinations

Un score n'est utile que s'il reste relié aux questions, aux réponses et aux raisons du verdict.

Lire le protocole d'évaluation

Le schéma distingue le cas de test, la réponse obtenue et le verdict. Une réponse peut être grammaticalement correcte mais infidèle à la référence ; à l'inverse, une paraphrase peut employer des mots différents tout en conservant le même fait. Les règles doivent donc comparer le sens attendu, pas seulement les chaînes de caractères.

Les deux questions hors contexte vérifient un autre comportement : l'assistant doit reconnaître l'absence d'information au lieu de compléter avec une connaissance non vérifiée. Le rapport conserve le détail de chaque décision afin qu'un score global puisse être expliqué et corrigé.

1. Préparer l'environnement

Conservez séparément les cas, les réponses produites et les rapports d'évaluation. Vous pourrez ainsi relancer le score sans régénérer les réponses, puis comparer deux versions de la règle sur les mêmes sorties. Le mode mock sert à vérifier ce circuit local ; il ne fournit pas, à lui seul, une mesure des hallucinations d'un modèle interrogé en conditions réelles.

Le projet sépare les cas d'évaluation dans `data/`, les fonctions de calcul dans `src/` et les rapports dans `outputs/`. Cette organisation facilite la comparaison de plusieurs exécutions : les questions restent stables pendant que les réponses du modèle peuvent changer.

L'exemple JSON présente deux situations à traiter différemment. Pour un cas documenté, `expected` contient une réponse de référence et `support` le texte disponible. Pour un cas hors contexte, `must_abstain` vaut vrai. Comprendre cette différence dès le départ évite de compter une abstention correcte comme une absence de réponse.

Terminal · Bash

```
mkdir -p evaluer-hallucinations/{data,src,outputs}
cd evaluer-hallucinations
python3 -m venv .venv && source .venv/bin/activate
python -m pip install pytest
```

Sortie obtenue

Sortie / exemple

Successfully installed pytest

Commande de vérification

Terminal · Bash

```
python -c "import pytest; print('evaluation stack ok')"
```

Code de l'étape

Un cas porte à la fois sur la réponse de référence et sur l'obligation éventuelle de s'abstenir :

JSON

```
[
  {"question": "q1", "expected": "a1", "support": "a1", "must_abstain": false},
  {"question": "q7", "expected": "", "support": "", "must_abstain": true}
]
```

2. Écrire les cas de référence

Les cas documentés et les cas hors périmètre ne mesurent pas la même chose. Pour les premiers, indiquez l'information qui doit être retrouvée dans le support ; pour les seconds, indiquez explicitement l'abstention requise. Relisez chaque référence sans regarder la réponse générée : adapter les attentes après coup pour faire passer une sortie rendrait la comparaison entre versions trompeuse.

Chaque cas doit être compréhensible sans connaître les autres. Écrivez une question, un support et une référence cohérents, puis relisez-les ensemble : un évaluateur ne peut pas considérer une réponse comme incorrecte si l'information nécessaire ne se trouve pas dans le support fourni.

Le jeu présenté contient huit cas avec support et deux cas d'abstention. Le contrôle vérifie leur structure et leur nombre, mais l'équilibre des thèmes doit être examiné manuellement. Gardez une copie stable de ce fichier pour comparer les versions du système ; modifier simultanément les cas et le prompt rendrait les différences de score difficiles à interpréter.

Créez `data/cases.json` avec `question`, `expected`, `support` et `must_abstain`. Une réponse à une question hors contexte doit contenir une abstention explicite.

Code de l'étape

Python

```
import json
from pathlib import Path

cases = [
```

```

{"question": "Quel est le délai de conservation des logs ?", "expected": "Les logs sont conservés 30 jours.", "support": "Les logs sont conservés 30 jours.", "must_abstain": False},
{"question": "Quelle métrique est suivie après le déploiement ?", "expected": "La latence et le taux d'erreur sont suivis.", "support": "La latence et le taux d'erreur sont suivis.", "must_abstain": False},
{"question": "Qui valide une action de mitigation ?", "expected": "Une validation humaine est requise.", "support": "Une validation humaine est requise.", "must_abstain": False},
{"question": "Quel seuil bloque la promotion ?", "expected": "Un seuil de qualité non respecté bloque la promotion.", "support": "Un seuil de qualité non respecté bloque la promotion.", "must_abstain": False},
{"question": "Combien de sources sont citées dans la réponse ?", "expected": "Chaque réponse cite au moins une source.", "support": "Chaque réponse cite au moins une source.", "must_abstain": False},
{"question": "Quel outil conserve les paramètres du run ?", "expected": "MLflow conserve les paramètres du run.", "support": "MLflow conserve les paramètres du run.", "must_abstain": False},
{"question": "Quel est le chiffre d'affaires de l'entreprise en 2030 ?", "expected": "", "support": "", "must_abstain": True},
{"question": "Quelle est la recette du couscous ?", "expected": "", "support": "", "must_abstain": True},
{"question": "Quel fichier conserve les passages récupérés ?", "expected": "outputs/last-retrieval.json conserve les passages.", "support": "outputs/last-retrieval.json conserve les passages.", "must_abstain": False},
{"question": "Le modèle est-il garanti sans erreur ?", "expected": "La qualité doit être mesurée, aucune garantie absolue n'est donnée.", "support": "La qualité doit être mesurée, aucune garantie absolue n'est donnée.", "must_abstain": False},
]
Path("data/cases.json").write_text(json.dumps(cases, ensure_ascii=False, indent=2), encoding="utf-8")
assert len(json.loads(Path("data/cases.json").read_text())) == 10

```

Sortie obtenue

Le fichier contient dix cas équilibrés entre réponses et abstentions.

■ Sortie / exemple

```
cases=10 supported=8 out_of_context=2 schema_errors=0
```

Commande de vérification

Terminal · Bash

```
python src/check_cases.py data/cases.json
```

3. Produire les réponses et les règles

Gardez l'association entre chaque cas et sa réponse lors de la sauvegarde. Si l'évaluation utilise deux listes avec `zip`, contrôlez aussi leurs longueurs et leur ordre : une réponse manquante ne doit pas disparaître du dénominateur. La recherche du mot « abstention » est un indicateur lexical ; une négation ou une phrase qui cite ce mot sans refuser demande une lecture humaine.

Le prompt est reconstruit séparément pour chaque cas. La question et son support sont ainsi transmis ensemble, et la consigne d'abstention reste identique d'une exécution à l'autre. Le mode mock lit des réponses connues pour vérifier le fonctionnement du calcul sans attendre un modèle.

Avant de calculer un score, vérifiez que le nombre de réponses correspond au nombre de cas et que leur ordre est conservé. Une réponse associée à la mauvaise question donne un verdict sans signification. Pour le mode réel, notez également les paramètres utilisés : un changement de température ou de modèle peut expliquer une variation que vous auriez attribuée au prompt.

En mode mock, chargez `examples/responses.json`. Pour le modèle réel, consignez le modèle, le prompt et la température. Une réponse est fidèle si elle reprend le support et n'invente pas une valeur absente.

Code de l'étape

```
Python

def build_prompt(case):
    return (
        "Réponds uniquement à partir du support fourni. "
        "Si le support ne permet pas de répondre, réponds exactement ABSTENTION.\n\n"
        f"Question: {case['question']}\nSupport: {case['support']}"
    )

if mock:
    answers = json.loads(Path("examples/responses.json").read_text())
else:
    answers = [chat([{"role": "user", "content": build_prompt(case)}])
               for case in cases]
```

Sortie obtenue

Chaque cas reçoit `faithful`, `abstained` et `notes`.

```
Sortie / exemple

case=1 faithful=true abstained=false
case=9 faithful=false abstained=true
```

Commande de vérification

```
Terminal · Bash

python src/evaluate.py --mock --cases data/cases.json --responses examples/responses.json
```

Interpréter le score

4.

Examinez le score global avec le détail des deux groupes : réponses documentées et demandes auxquelles il faut s'abstenir. Un système qui refuse tout peut réussir les cas inconnus tout en échouant sur les questions utiles. Publiez les comptes avec leurs dénominateurs, puis ouvrez les échecs un par un pour distinguer un défaut de récupération, une affirmation ajoutée et une règle d'évaluation trop grossière.

Lisez la règle `score_case` avant d'interpréter la moyenne. Elle recherche le mot « abstention » ou la présence de la référence dans la réponse, sans analyser le raisonnement complet. Une réponse qui contient la bonne phrase puis ajoute une information fausse peut donc passer ce contrôle lexical.

Le rapport doit être lu avec ses cas détaillés. Examinez séparément une abstention attendue, une abstention alors qu'une réponse existe et une réponse non soutenue. Le couple `faithful` / `abstained` aide à faire cette distinction. Le score sert à repérer des régressions sur ce jeu ; il ne constitue pas une mesure générale du taux d'hallucination du modèle.

Calculez la moyenne de fidélité et la couverture des abstentions. Exportez les cas échoués ; ne masquez pas un mauvais score par une moyenne seule.

Sortie obtenue

Le rapport donne un score sur 10, la liste des échecs et la cause de chaque échec.

Sortie / exemple

```
total=10 faithful=8 score=0.80 failed_cases=[7, 10]
```

Commande de vérification

Terminal · Bash

```
python -m json.tool outputs/evaluation.json >/dev/null
pytest -q
```

Code de l'étape

La règle minimale sépare fidélité et abstention :

Python

```
def score_case(case, answer):
    abstained = "abstention" in answer.casefold()
    faithful = ((case["must_abstain"] and abstained) or
                (not case["must_abstain"] and
                 case["expected"].casefold() in answer.casefold()))
    return {"faithful": faithful, "abstained": abstained}
```

```
records = [
    {**case, "answer": answer, **score_case(case, answer)}
    for case, answer in zip(cases, answers)
]
report = {
    "total": len(records),
    "faithful": sum(item["faithful"] for item in records),
    "failed_cases": [i + 1 for i, item in enumerate(records)
                     if not item["faithful"]],
    "responses": records,
}
```

Exemples de sorties

Le jeu contient dix cas, le mode mock identifie les réponses fidèles et signale au moins une abstention manquée si la fixture est modifiée.

Dépannage et diagnostics

Une métrique automatique ne prouve pas la vérité. Faites relire les cas critiques et conservez les sorties brutes pour permettre un audit.

Vérification finale

- ☐ dix cas équilibrés ;
- ☐ modèle et prompt enregistrés ;
- ☐ score et échecs exportés ;
- ☐ question hors contexte évaluée.

Limites et prolongements

Les critères lexicaux ratent les paraphrases. Ajoutez une évaluation humaine ou une métrique sémantique calibrée sur des exemples annotés.

Références

- [Ragas — documentation d'évaluation RAG](#)
- [NIST AI Risk Management Framework](#)