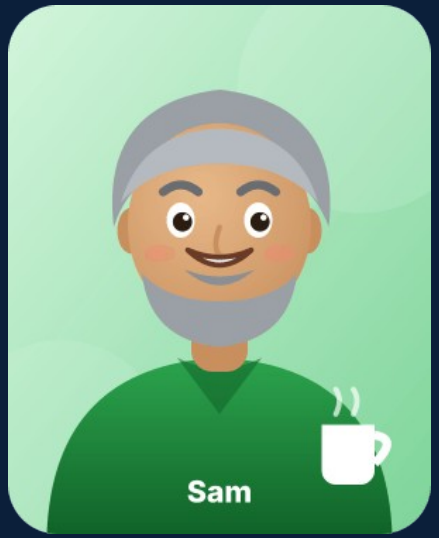
HEF-160 · MISSION 07 / 10

Évaluation et benchmarks

« Ça a l'air de marcher » n'est pas une évaluation. Pour comparer, choisir et surveiller un système d'IA, il faut des cas, des métriques, de la variance et de l'honnêteté.





Badge : Mesureur de systèmes d'IA

110 min · avancé

L'histoire

Deux prompts pour la classification des tickets de HashBox. Kodi les essaie sur cinq exemples : le premier se trompe une fois, le second aucune. Il déploie le second. En production, le second se trompe plus souvent que le premier. Sam examine : les cinq exemples étaient tirés de la documentation du prompt gagnant (il les avait déjà « vus »), l'échantillon était minuscule, et les deux prompts avaient été lancés une seule fois chacun, avec une température non nulle.

« Cinq exemples, une exécution, des cas qui ressemblent à la réponse : ce n'est pas une mesure, c'est une impression, dit Sam. Évaluer un système d'IA, c'est construire un instrument : un jeu de cas représentatif, des critères objectifs, plusieurs exécutions, un intervalle de confiance, une comparaison appariée — et se méfier des fuites entre entraînement et test. »



L'histoire (suite)

Tu apprends à construire un jeu d'évaluation, à choisir des métriques (exactitude, précision/rappel, pass@k), à estimer l'incertitude, à comparer deux systèmes de façon appariée, à mesurer l'accord entre annotateurs (kappa), à détecter la contamination, et à éviter les pièges (Goodhart, juges automatiques complaisants).

Ta mission

À la fin, je serai capable de…

1. construire un jeu d'évaluation représentatif avec des critères de correction objectifs
2. calculer exactitude, précision, rappel, F1 et pass@k (estimateur non biaisé)
3. estimer l'incertitude par bootstrap et comparer deux systèmes sur les mêmes cas
4. mesurer l'accord entre annotateurs avec le kappa de Cohen
5. détecter la contamination et repérer les pièges d'évaluation (Goodhart, juge automatique)

Défi d'échauffement

Défi d'échauffement

Un professeur veut savoir si deux méthodes d'enseignement diffèrent. Il teste la première sur 5 élèves, la seconde sur 5 autres élèves, après avoir donné à la seconde classe les corrigés de l'examen. Cite les erreurs de méthode. Comment procéderais-tu pour que la comparaison soit honnête ?

Le concept

Que veut-on évaluer ?

Avant les métriques : **la tâche**, **qui utilisera le système**, **ce qu'est une erreur grave**. Une évaluation sert à **décider** (déployer ? changer de modèle ? corriger le prompt ?) ou à **surveiller** (régression).

Le jeu d'évaluation

Qualité	Exigence
Représentatif	reflète les vraies entrées (types, difficulté, fréquence)
Divers	cas faciles, difficiles, limites, **cas piégeux**, **hors périmètre**
Étiqueté de façon fiable	critère de correction écrit, accord entre annotateurs mesuré
Séparé	jamais utilisé pour concevoir le prompt ou entraîner (sinon : surajustement)

Le concept

Le jeu d'évaluation (suite)

Qualité	Exigence
Assez grand	l'incertitude doit être acceptable (au moins quelques dizaines à centaines de cas)
Versionné	pour comparer dans le temps

Découper : un jeu de **développement** (on itère dessus) et un jeu de **test** (on ne le regarde qu'à la fin).

Métriques selon la tâche

Tâche	Métriques
Classification	exactitude, **précision**, **rappel**, **F1**, matrice de confusion
Code	**taux de réussite des tests** (`pass@k`)
Extraction	correspondance exacte, F1 sur les champs

Le concept

Métriques selon la tâche (suite)

Tâche	Métriques
Génération libre	critères vérifiables (grille), comparaison par paires, juge humain ou automatique **calibré**
RAG	rappel@k, fidélité aux sources, taux de refus justifié (mission 4)
Agents	taux de tâches résolues, coût, violations (mission 6)

Précision, rappel, F1

```
précision = VP / (VP + FP)     rappel = VP / (VP + FN)     F1 = 2·P·R / (P + R)
```

Selon le coût des erreurs : détecter une fuite de données (rappel d'abord) ou envoyer un avertissement à tort (précision d'abord).

Le concept

pass@k

Pour du code : on génère n solutions, c passent les tests. La probabilité qu'**au moins une** de k solutions tirées passe vaut (estimateur non biaisé) :

```
pass@k = 1 − C(n − c, k) / C(n, k)
```

Variance et incertitude

- Un système échantillonné **varie** d'une exécution à l'autre : lancer **plusieurs fois**, rapporter moyenne et **dispersion**.
- Toute métrique sur n cas a une **incertitude** : intervalle de confiance par **bootstrap** (rééchantillonner les cas) ou de Wilson (proportion).
- « 82 % contre 79 % » sur 50 cas **ne prouve rien**.

Le concept

Comparer deux systèmes

- **Mêmes cas** pour les deux (comparaison **appariée**) : on regarde, cas par cas, qui gagne.
- Test sur les **discordances** (cas où l'un réussit et pas l'autre) : test de McNemar, ou bootstrap de la différence.
- Rapporter la **différence** avec son intervalle, pas deux nombres isolés.

Accord entre annotateurs : kappa de Cohen

```
κ = (accord observé − accord attendu par hasard) / (1 − accord attendu par hasard)
```

`κ > 0,8` : excellent ; `0,6-0,8` : bon ; `< 0,4` : faible — les critères sont flous, et aucun système ne peut être jugé de façon fiable.

Le concept

La contamination

Si des cas du jeu de test figurent dans les **données d'entraînement** (ou dans tes prompts d'exemples), le score est **gonflé**. Détection : recouvrement de n-grammes entre test et corpus connu ; cas « canari » ; jeux de test privés ; renouvellement.

Les pièges

Piège	Remède
Goodhart (métrique optimisée → plus fiable)	plusieurs métriques, jeu de test caché, revue qualitative
Juge automatique (un modèle note un autre)	calibrer sur des notes humaines, mesurer l'accord, biais de longueur / de position
Moyenne qui cache les cas graves	analyser par catégorie et par gravité

Le concept

Les pièges (suite)

Piège	Remède
Évaluer sur un seul exemple ou une seule exécution	échantillon, répétitions
Benchmarks publics	utiles pour comparer des modèles, **pas** pour valider ton usage
Coût ignoré	rapporter latence, coût par tâche réussie

Évaluation continue

Les modèles, les prompts, les données changent : **jouer le banc** à chaque changement (régression) et **surveiller en production** (échantillonnage + relecture humaine, signaux de dérive).

🧠 ***Astuce*** *: avant de lancer une évaluation, écris la décision qu'elle va éclairer et le seuil qui la déclenche. Sinon, tu choisiras le seuil après avoir vu les chiffres.*

Exemple pas à pas

Problème : calculer précision, rappel, F1 et pass@k ; estimer l'incertitude par bootstrap ; comparer deux systèmes de façon appariée ; mesurer l'accord entre annotateurs ; détecter la contamination ; et montrer qu'une évaluation sur cinq exemples est trompeuse.

```
import math, random
from itertools import combinations

# ---------------- 1. précision, rappel, F1 ----------------
def prf(vrais, predits, positif):
    vp = sum(1 for v, p in zip(vrais, predits) if v == positif and p == positif)
    fp = sum(1 for v, p in zip(vrais, predits) if v != positif and p == positif)
    fn = sum(1 for v, p in zip(vrais, predits) if v == positif and p != positif)
    precision = vp / (vp + fp) if vp + fp else 0.0; rappel = vp / (vp + fn) if vp
   + fn else 0.0
    return precision, rappel, (2 * precision * rappel / (precision + rappel) if pr
   ecision + rappel else 0.0)
V = ["fuite"] * 10 + ["ok"] * 90
P_prudent = ["fuite"] * 6 + ["ok"] * 4 + ["ok"] * 90                          # r
```

Exemple pas à pas (suite)

```
   ate 4 fuites mais ne fait aucune fausse alerte
P_alarmiste = ["fuite"] * 10 + ["fuite"] * 20 + ["ok"] * 70                    # a
   ttrape tout mais 20 fausses alertes
for nom, p in (("prudent", P_prudent), ("alarmiste", P_alarmiste)):
    pr, ra, f1 = prf(V, p, "fuite"); exact = sum(v == x for v, x in zip(V, p)) / l
   en(V)
    print(f"{nom:10s}: exactitude {exact:.0%} | précision {pr:.0%} | rappel {ra:.0
   %} | F1 {f1:.2f}")
assert prf(V, P_prudent, "fuite")[0] == 1.0 and prf(V, P_alarmiste, "fuite")[1] ==
    1.0
print("→ l'exactitude seule (96 % contre 80 %) cache que le système prudent laisse
    passer 40 % des fuites")

# ---------------- 2. pass@k (estimateur non biaisé) ----------------
def pass_at_k(n, c, k):
    if n - c < k: return 1.0
    return 1 - math.comb(n - c, k) / math.comb(n, k)
print("n = 20 solutions générées, c = 4 passent :", {k: round(pass_at_k(20, 4, k),
    3) for k in (1, 3, 5, 10)})
```

Exemple pas à pas (suite)

```
assert round(pass_at_k(20, 4, 1), 3) == 0.2 and pass_at_k(20, 4, 10) > 0.95

# ---------------- 3. incertitude : 5 exemples contre 200 ----------------
def ic_bootstrap(resultats, b=2000, graine=1):
    rng = random.Random(graine); n = len(resultats)
    moyennes = sorted(sum(rng.choice(resultats) for _ in range(n)) / n for _ in ra
   nge(b)); return moyennes[int(0.025 * b)], moyennes[int(0.975 * b)]
rng = random.Random(3)
vrai_taux_A, vrai_taux_B = 0.80, 0.84                                    # B
   est réellement un peu meilleur
cas_A = [1 if rng.random() < vrai_taux_A else 0 for _ in range(200)]
petit = [1, 1, 0, 1, 1]
print(f"sur 5 cas : A = {sum(petit) / 5:.0%}, IC95 = [{ic_bootstrap(petit)[0]:.0%}
    ; {ic_bootstrap(petit)[1]:.0%}] ; sur 200 cas : A = {sum(cas_A) / 200:.0%}, IC
   95 = [{ic_bootstrap(cas_A)[0]:.0%} ; {ic_bootstrap(cas_A)[1]:.0%}]")
assert ic_bootstrap(cas_A)[1] - ic_bootstrap(cas_A)[0] < 0.14 and ic_bootstrap(pet
   it)[1] - ic_bootstrap(petit)[0] >= 0.4

# ---------------- 4. comparaison appariée de deux systèmes sur les MÊMES cas ----
```

Exemple pas à pas (suite)

```
    ------------
def simuler_paire(n, graine):
    r = random.Random(graine); difficulte = [r.random() for _ in range(n)]
       # chaque cas a une difficulté commune aux deux systèmes
    A = [1 if r.random() < 0.92 - 0.3 * d else 0 for d in difficulte]
    B = [1 if r.random() < 1.0 - 0.3 * d else 0 for d in difficulte]
    return A, B
A, B = simuler_paire(300, 5)
def diff_bootstrap(A, B, b=3000, graine=2):
    rng = random.Random(graine); n = len(A); idx = range(n)
    d = sorted(sum(B[i] - A[i] for i in (rng.choice(idx) for _ in range(n))) / n f
   or _ in range(b)); return d[int(0.025 * b)], d[int(0.975 * b)]
bas, haut = diff_bootstrap(A, B)
b01 = sum(1 for a, b in zip(A, B) if a == 0 and b == 1); b10 = sum(1 for a, b in z
   ip(A, B) if a == 1 and b == 0)
chi2 = (abs(b01 - b10) - 1) ** 2 / (b01 + b10) if b01 + b10 else 0
print(f"A = {sum(A) / len(A):.1%}, B = {sum(B) / len(B):.1%} ; discordances : B se
   ul réussit {b01}, A seul réussit {b10} ; différence B − A = {(sum(B) - sum(A))
   / len(A):+.1%}, IC95 = [{bas:+.1%} ; {haut:+.1%}] ; McNemar χ² = {chi2:.1f} (>
```

Exemple pas à pas

Exemple pas à pas (suite)

```
   3,84 : significatif à 5 %)")
assert (bas > 0) == (chi2 > 3.84)

# ---------------- 5. accord entre annotateurs : kappa de Cohen ----------------
def kappa(a, b):
    n = len(a); po = sum(x == y for x, y in zip(a, b)) / n
    cats = set(a) | set(b); pe = sum((a.count(c) / n) * (b.count(c) / n) for c in
   cats)
    return (po - pe) / (1 - pe)
r = random.Random(7); verite = [r.choice(["bon", "moyen", "mauvais"]) for _ in ran
   ge(120)]
def annotateur(taux): return [v if r.random() < taux else r.choice(["bon", "moyen"
   , "mauvais"]) for v in verite]
for nom, taux in (("critères clairs", 0.95), ("critères flous", 0.55)):
    print(f"annotateurs avec {nom:16s}: κ = {kappa(annotateur(taux), annotateur(ta
   ux)):.2f}")
assert kappa(annotateur(0.95), annotateur(0.95)) > 0.7 > kappa(annotateur(0.5), an
   notateur(0.5))
```

Exemple pas à pas

Exemple pas à pas (suite)

```
# ---------------- 6. contamination : recouvrement de n-grammes ----------------
def ngrammes(texte, n=5):
    mots = texte.lower().split(); return {tuple(mots[i:i + n]) for i in range(len(
   mots) - n + 1)}
CORPUS_CONNU = "pour obtenir un remboursement il faut en faire la demande dans les
    quatorze jours suivant l achat du cours avec le moyen de paiement d origine"
TEST = ["pour obtenir un remboursement il faut en faire la demande dans les quator
   ze jours suivant l achat", "comment changer le moteur d un avion de ligne", "il
    faut en faire la demande dans les quatorze jours"]
connus = ngrammes(CORPUS_CONNU)
for t in TEST:
    g = ngrammes(t); rec = len(g & connus) / max(len(g), 1); print(f"  recouvremen
   t {rec:4.0%} {'← SUSPECT (contamination)' if rec > 0.5 else ''}: {t[:60]}")
assert len(ngrammes(TEST[0]) & connus) / len(ngrammes(TEST[0])) == 1.0 and len(ngr
   ammes(TEST[1]) & connus) == 0

# ---------------- 7. cinq exemples : une décision trompeuse ----------------
def decision_sur_n(n, graine):
    r = random.Random(graine); a = sum(r.random() < 0.80 for _ in range(n)); b = s
```

Exemple pas à pas

Exemple pas à pas (suite)

```
   um(r.random() < 0.84 for _ in range(n)); return b > a
for n in (5, 30, 200, 1000):
    bonne = sum(decision_sur_n(n, g) for g in range(1000)) / 1000
    print(f"n = {n:4d} cas : le meilleur système (84 % contre 80 %) gagne dans {bo
   nne:.0%} des évaluations")
assert sum(decision_sur_n(5, g) for g in range(1000)) / 1000 < sum(decision_sur_n(
   1000, g) for g in range(1000)) / 1000
```

Lecture :

1. L'**exactitude** peut tromper : un système « prudent » à 96 % laisse passer 40 % des fuites ; le système « alarmiste » à 80 % les trouve toutes. Le bon choix dépend du **coût des erreurs**, que précision et rappel rendent visible.
2. **pass@k** mesure la chance d'obtenir au moins une solution correcte parmi `k` essais ; avec 4 bonnes solutions sur 20, `pass@1 = 20 %` mais `pass@10 > 95 %` : très utile **si** un vérificateur automatique peut sélectionner la bonne (mission 8).

Exemple pas à pas (suite)

3. Sur 5 cas, l'incertitude est énorme ; sur 200 elle reste de plusieurs points.
4. La **comparaison appariée** s'appuie sur les discordances et sur l'intervalle de la différence. Ici, B fait +4 points sur 300 cas, mais l'intervalle contient 0 et le test de McNemar n'est pas significatif : **on ne peut pas conclure** — mieux vaut le savoir que décider au hasard.
5. Le **kappa** montre que des critères flous (κ faible) rendent toute évaluation peu fiable.
6. Le **recouvrement de n-grammes** repère un cas de test déjà présent dans un corpus connu : le score serait gonflé.
7. Avec 5 cas, le meilleur système ne « gagne » que dans 41 % des évaluations (les égalités comptent comme non-victoire) ; il faut des centaines de cas pour détecter une différence de 4 points : 80 % des évaluations à 200 cas, 99 % à 1 000.

À toi, avec aide

À toi, avec aide

Un classificateur de tickets a produit ces résultats sur 200 tickets (classe « urgent » = positive) :

	Prédit urgent	Prédit non urgent
Vraiment urgent	38	12
Vraiment non urgent	22	128

1. Calcule exactitude, précision, rappel, F1.
2. Un ticket urgent non détecté coûte en moyenne 400 € ; une fausse alerte coûte 20 €. Quel est le coût total sur ces 200 tickets ? Quel serait le coût si l'on détectait 10 urgents de plus au prix de 30 fausses alertes de plus ?
3. Que préfères-tu optimiser, précision ou rappel ? Justifie avec les coûts.
4. Calcule un intervalle de Wilson sur le rappel (38/50).

À toi, avec aide · réponse

Indices progressifs

Indice 1

Précision = VP / (VP + FP) ; rappel = VP / (VP + FN).

Indice 2

Coût = FN × 400 + FP × 20.

Indice 3

Wilson : voir mission HEF-150.

À toi, avec aide · réponse

Réponse

```
import math
vp, fn, fp, vn = 38, 12, 22, 128
precision, rappel = vp / (vp + fp), vp / (vp + fn)
f1 = 2 * precision * rappel / (precision + rappel)
print(f"exactitude {(vp + vn) / 200:.1%} | précision {precision:.1%} | rappel {rap
   pel:.1%} | F1 {f1:.2f}")
cout = lambda fn, fp: fn * 400 + fp * 20
print("coût actuel :", cout(fn, fp), "€ ; après +10 détections et +30 fausses aler
   tes :", cout(fn - 10, fp + 30), "€")
def wilson(s, n, z=1.96):
    p = s / n; c = (p + z * z / (2 * n)) / (1 + z * z / n); d = z * math.sqrt(p *
   (1 - p) / n + z * z / (4 * n * n)) / (1 + z * z / n); return c - d, c + d
print("IC95 du rappel : [%.0f%% ; %.0f%%]" % tuple(100 * x for x in wilson(38, 50)
   ))
assert cout(12, 22) == 5240 and cout(2, 52) == 1840
```

1. Exactitude **83 %**, précision **63,3 %**, rappel **76 %**, F1 **0,69**.

Réponse (suite)

2. Coût actuel : 12 × 400 + 22 × 20 = **5 240 €**. En détectant 10 urgents de plus au prix de 30 fausses alertes : 2 × 400 + 52 × 20 = **1 840 €** : on économise 3 400 €.
3. Le **rappel**, car manquer un urgent coûte vingt fois plus qu'une fausse alerte. Le point de fonctionnement optimal dépend de ce rapport de coûts, pas d'un score unique.
4. Rappel de 76 % avec un intervalle de Wilson d'environ **[62 % ; 86 %]** : sur 50 urgents seulement, l'incertitude reste forte.

À toi, en solo

Banc d'évaluation de bout en bout. Construis un petit cadre d'évaluation réutilisable :

1. un jeu de 60 cas de classification de tickets en 4 catégories (avec 8 cas piégeux et 6 hors périmètre) généré avec graine fixée, et deux « systèmes » simulés (taux d'erreur différents par catégorie) ;
2. séparation développement / test ; le banc **refuse** de calculer un score sur le jeu de test plus d'une fois par système (compteur d'accès) pour décourager le surajustement ;
3. métriques par catégorie (précision, rappel, F1), matrice de confusion, exactitude globale avec IC ;
4. exécutions multiples (20 graines) pour un système échantillonné : moyenne et dispersion ;
5. comparaison appariée avec IC de la différence et test de McNemar ;
6. détection de contamination entre exemples du prompt (few-shot) et jeu de test (recouvrement) ;

À toi, en solo (suite)

7. règle de décision écrite d'avance (« déployer B si l'IC bas de la différence > 0 ET le rappel de la catégorie critique ≥ 90 % »), puis application ;
8. rapport texte reproductible.

À toi, en solo · réponse

Pistes de correction

```
import random, math
from collections import Counter

CATS = ["paiement", "compte", "technique", "autre"]
def jeu(graine=1, n=60):
    r = random.Random(graine); cas = []
    for i in range(n):
        v = r.choice(CATS); cas.append(dict(id=i, texte=f"ticket {i} sur {v}", ver
   ite=v, piege=i < 8, hors=i >= n - 6))
    for c in cas:
        if c["hors"]: c["verite"] = "autre"
    return cas
class Banc:
    def __init__(self, cas, part_dev=0.5, graine=0):
        r = random.Random(graine); idx = list(range(len(cas))); r.shuffle(idx); k
   = int(len(cas) * part_dev)
        self.dev, self.test = [cas[i] for i in idx[:k]], [cas[i] for i in idx[k:]]
   ; self.acces_test = Counter()
    def scorer_test(self, nom_systeme, systeme):
```

À toi, en solo · réponse

Pistes de correction (suite)

```
        self.acces_test[nom_systeme] += 1
        if self.acces_test[nom_systeme] > 1: raise PermissionError(f"le jeu de tes
   t a déjà été utilisé pour {nom_systeme} : risque de surajustement")
        return [systeme(c) for c in self.test]
def systeme(taux_base, taux_critique, graine):
    r = random.Random(graine)
    def f(c):
        e = taux_critique if c["verite"] == "paiement" else taux_base
        if c["piege"]: e *= 1.8
        return r.choice([x for x in CATS if x != c["verite"]]) if r.random() < e e
   lse c["verite"]
    return f
cas = jeu(); banc = Banc(cas)
sA, sB = systeme(0.15, 0.12, 1), systeme(0.10, 0.04, 1)
pA, pB = banc.scorer_test("A", sA), banc.scorer_test("B", sB)
verites = [c["verite"] for c in banc.test]
def prf(cat, pred):
    vp = sum(v == cat == p for v, p in zip(verites, pred)); fp = sum(v != cat and
   p == cat for v, p in zip(verites, pred)); fn = sum(v == cat and p != cat for v,
```

À toi, en solo · réponse

Pistes de correction (suite)

```
    p in zip(verites, pred))
    return (vp / (vp + fp) if vp + fp else 0.0), (vp / (vp + fn) if vp + fn else 0
   .0)
for nom, p in (("A", pA), ("B", pB)):
    print(nom, "exactitude", f"{sum(v == x for v, x in zip(verites, p)) / len(p):.
   0%}", "| paiement : précision/rappel =", tuple(round(x, 2) for x in prf("paieme
   nt", p)))
try: banc.scorer_test("B", sB); raise AssertionError
except PermissionError as e: print("garde-fou :", e)
A_ok = [int(v == x) for v, x in zip(verites, pA)]; B_ok = [int(v == x) for v, x in
    zip(verites, pB)]
rng = random.Random(4); n = len(A_ok)
d = sorted(sum(B_ok[i] - A_ok[i] for i in (rng.randrange(n) for _ in range(n))) /
   n for _ in range(2000))
bas, haut = d[50], d[1950]
rappel_critique = prf("paiement", pB)[1]
decision = "déployer B" if bas > 0 and rappel_critique >= 0.90 else "ne pas conclu
   re : plus de cas ou meilleur système"
print(f"différence B − A : IC95 [{bas:+.0%} ; {haut:+.0%}], rappel paiement de B :
```

À toi, en solo · réponse

Pistes de correction (suite)

```
    {rappel_critique:.0%} → {decision}")
FEW_SHOT = ["ticket 12 sur paiement", "ticket 40 sur compte"]
contamines = [c["id"] for c in banc.test if any(c["texte"] == e for e in FEW_SHOT)
   ]
print("cas de test contaminés par les exemples du prompt :", contamines or "aucun"
   )
```

La **règle de décision écrite avant** les chiffres évite de choisir un seuil complaisant ; le **compteur d'accès** au jeu de test empêche l'itération indéfinie sur la même mesure ; la contamination par les exemples du prompt se détecte par simple comparaison. Avec 30 cas de test, l'intervalle est large : la décision honnête est souvent « on ne sait pas encore ».

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Évaluer sur quelques exemples	Incertitude énorme, décisions au hasard	Centaines de cas, intervalles de confiance
Itérer sur le jeu de test	Surajustement : le score ne vaut plus rien	Séparer développement / test ; jeu caché
Une seule exécution d'un système échantillonné	Variance ignorée	Plusieurs exécutions, moyenne et dispersion
Faire confiance à un juge automatique sans le calibrer	Biais de longueur, complaisance	Comparer à des notes humaines, mesurer l'accord

Mini-boss

Mini-boss

Rapport d'évaluation d'un assistant de HashBox. Évalue un assistant (simulé) qui répond à des questions de support avec RAG :

1. jeu de 100 cas (dont 15 hors corpus, 10 pièges, 10 sur documents contradictoires) avec critères de correction écrits ; deux annotateurs simulés ; kappa ;
2. séparation développement / test ; protocole d'accès unique au test ;
3. métriques : exactitude factuelle, fidélité aux sources, taux de refus justifié, taux de réponse à tort, rappel@k de la récupération ;
4. 20 exécutions par système ; intervalles ; comparaison appariée de deux versions du prompt et de deux tailles de morceaux ;
5. détection de contamination ;
6. analyse par catégorie et par gravité : lister les 5 erreurs les plus graves ;
7. coût et latence par réponse correcte ;

Mini-boss

Mini-boss (suite)

8. décision écrite d'avance et application ; recommandations ; ce que l'évaluation ne dit pas.

Mini-boss · réponse

Pistes de correction

Un bon rapport : critères de correction écrits et kappa > 0,7, protocole d'accès unique au test, tableau de métriques avec intervalles, comparaison appariée avec McNemar et IC de la différence, analyse des erreurs graves (réponse à tort sur des questions hors corpus), coût par réponse correcte, décision conforme à la règle écrite et section « limites » honnête (jeu simulé, taille limitée, juge non indépendant).

Quiz éclair

Quiz éclair

1. Pour comparer deux systèmes, on évalue de préférence :

A. les deux sur les mêmes cas, en regardant la différence et son intervalle
B. chacun sur des cas différents
C. un seul exemple chacun

2. Le kappa de Cohen mesure :

A. la vitesse du modèle
B. l'accord entre annotateurs au-delà du hasard
C. la taille du jeu de test

3. La contamination d'un benchmark signifie que :

A. le jeu de test est trop grand

Quiz éclair

Quiz éclair (suite)

B. le modèle est trop lent

C. des cas de test figurent dans les données d'entraînement ou les exemples, gonflant le score

4. 🔁 *Révision mission 06* : Pour empêcher un agent de « réussir » en modifiant les tests, on :

A. met les fichiers de mesure en lecture seule et vérifie de façon indépendante

B. lui demande poliment de ne pas le faire

C. augmente la température

5. 🔁 *Révision mission 06* : Si chaque étape réussit à 90 %, la réussite globale sur 10 étapes est environ :

A. 90 %

B. 35 %

Quiz éclair

Quiz éclair (suite)

C. 99 %

Quiz éclair · réponse

Réponses

1. **A**.
2. **B**.
3. **C**.
4. **A**.
5. **B**.

Je réfléchis

1. Quelle décision d'IA a été prise dans ton équipe sur la base de très peu d'exemples ?
2. Quel cas grave ton jeu d'évaluation actuel ne contient-il pas ?

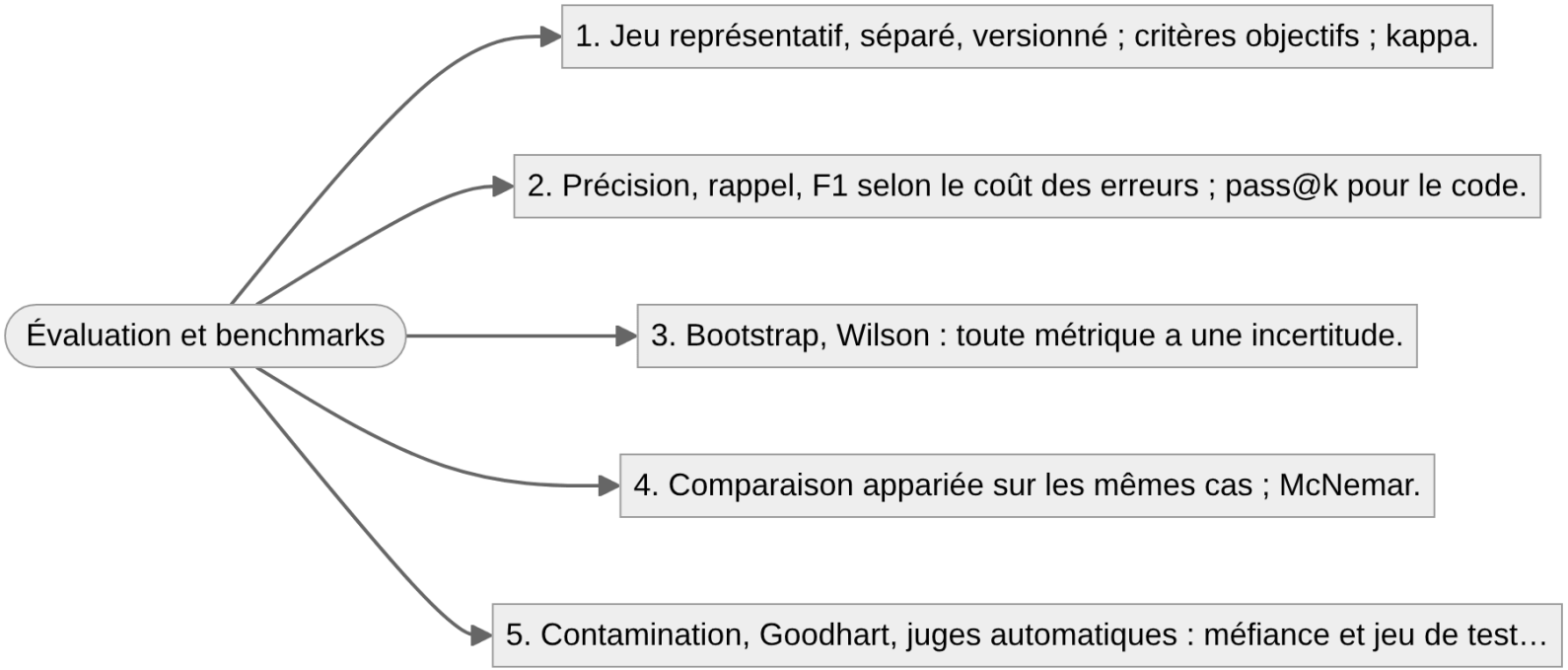
À retenir

À retenir

1. Jeu représentatif, séparé, versionné ; critères objectifs ; kappa.
2. Précision, rappel, F1 selon le coût des erreurs ; pass@k pour le code.
3. Bootstrap, Wilson : toute métrique a une incertitude.
4. Comparaison appariée sur les mêmes cas ; McNemar.
5. Contamination, Goodhart, juges automatiques : méfiance et jeu de test caché.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Mesureur de systèmes d'IA

Prochaine mission : Vérification humaine et tests

