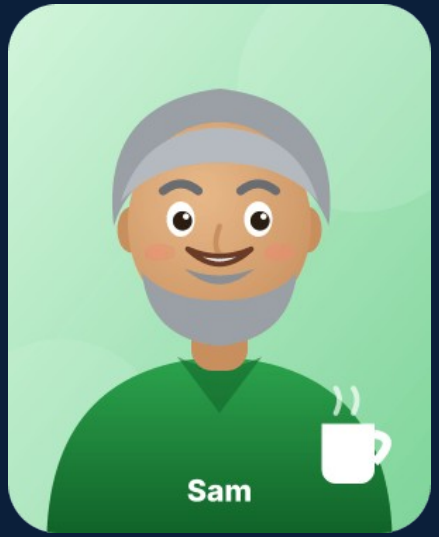
HEF-160 · MISSION 03 / 10

Context engineering



Un modèle ne répond bien que s'il a sous les yeux la bonne information, en quantité raisonnable, bien ordonnée. Composer ce contexte est un métier d'ingénierie.



Badge : Metteur en contexte

105 min · avancé

L'histoire

L'assistant de support de HashBox répond bien aux premières questions d'une conversation, puis se met à contredire ses propres réponses et à oublier le nom du client. Dans les journaux, Kodi découvre pourquoi : l'application envoie à chaque tour *tout* l'historique, les 40 articles d'aide, la politique de remboursement et 12 descriptions d'outils. Au quinzième tour, la fenêtre est pleine ; le début de la conversation est coupé en silence, y compris le nom du client et l'objet du litige. Le coût par conversation a triplé.



« Le prompt, c'est ce qu'on dit au modèle, explique Sam. Le contexte, c'est tout ce qu'il a sous les yeux : consignes, historique, documents, outils, résultats d'outils, mémoire. La qualité d'un système à base d'IA se joue souvent là : quoi mettre, en quelle quantité, dans quel ordre, avec quel résumé — et quoi laisser dehors. »

L'histoire (suite)

Tu apprends à compter les jetons, à budgéter une fenêtre de contexte, à sélectionner l'information la plus utile sous contrainte, à résumer et compresser un historique, à ordonner pour limiter l'oubli du milieu, et à gérer une mémoire qui dure plus qu'une conversation.

Ta mission

À la fin, je serai capable de…

1. décrire les composantes d'un contexte (consignes, historique, connaissances, outils, mémoire) et leurs coûts
2. estimer et budgéter des jetons dans une fenêtre limitée
3. sélectionner l'information la plus pertinente sous contrainte (sac à dos pondéré par la pertinence)
4. résumer et compresser un historique sans perdre les faits critiques
5. ordonner et structurer le contexte, et gérer une mémoire à plusieurs niveaux

Défi d'échauffement

Défi d'échauffement

Tu prépares un dossier pour un collègue qui doit décider en cinq minutes. Tu as 200 pages de documents. Tu peux lui en donner 5. Lesquelles ? Dans quel ordre ? Que fais-tu des 195 autres ? Et si sa décision dépend d'un chiffre caché à la page 87, comment t'assures-tu qu'il le voit ?

Le concept

Qu'est-ce que le contexte ?

Tout ce que le modèle reçoit pour produire sa réponse, dans une **fenêtre de contexte** limitée (mesurée en jetons) qui contient **entrée + sortie**.

Composante	Rôle	Coût typique
Consignes système	rôle, règles, ton, limites	stable, souvent mise en cache
Historique de conversation	continuité	croît à chaque tour
Connaissances récupérées (RAG)	faits à utiliser (mission 4)	variable
Définitions d'outils	ce que le modèle peut appeler (mission 5)	chaque outil coûte des jetons
Résultats d'outils	observations	peuvent être volumineux
Mémoire	faits retenus d'une conversation à l'autre	à gérer

Le concept

Qu'est-ce que le contexte ? (suite)

Composante	Rôle	Coût typique
Entrée de l'utilisateur	la demande	petite

Pourquoi « ingénierie »

- **Plus n'est pas mieux.** Une fenêtre pleine coûte plus cher (jetons), ralentit, et **dilue** l'information utile : un modèle retrouve moins bien un fait noyé dans beaucoup de bruit.
- **L'information utile doit y être.** Si elle n'est pas dans le contexte, le modèle ne la connaît pas (mission 1).
- **Le contexte est un budget.** Chaque jeton est un coût en argent, en temps et en attention.

Budgéter

Le concept

Budgéter (suite)

```
fenêtre = consignes + historique + connaissances + outils + réserve de sortie + ma
   rge
```

Règle pratique : **réserver la sortie** (on ne peut pas remplir la fenêtre d'entrée à 100 %), garder une **marge** (estimation des jetons approximative), prévoir ce qui est **fixe** (consignes, outils) et ce qui est **variable** (historique, connaissances).

Estimation grossière : 1 jeton ≈ 0,75 mot en anglais, ≈ 0,6 à 0,75 en français (≈ 4 caractères). Pour compter exactement, utiliser le compteur de jetons du fournisseur.

Sélectionner sous contrainte

Chaque morceau d'information (document, extrait, message passé) a :

- une **pertinence** estimée pour la question (score de récupération, mission 4) ;

Le concept

Sélectionner sous contrainte (suite)

- un **coût** en jetons.

Choisir ce qu'on inclut = problème de **sac à dos** : maximiser la pertinence totale sous le budget. Un glouton par **pertinence / coût** donne souvent un très bon résultat. Ajouter une **diversité** (ne pas inclure cinq fois la même information) améliore la couverture.

Résumer et compresser

Technique	Principe	Risque
Fenêtre glissante	garder les N derniers tours	oublie le début (objet du litige !)
Résumé de l'historique ancien	condenser en quelques lignes	perte de détails, erreurs de résumé

Le concept

Résumer et compresser (suite)

Technique	Principe	Risque
Faits épinglés	garder toujours certains faits critiques (identité, objet, décisions) hors du résumé	à bien choisir
Résumé extractif	garder les phrases les plus informatives	moins fluide mais fidèle
Élagage des résultats d'outils	ne garder que les champs utiles	oubli d'un champ nécessaire plus tard

Règle : **ne jamais résumer ce qui doit rester exact** (chiffres, identifiants, engagements) : l'**épingler** textuellement.

Ordre et structure

- **Instructions importantes au début et rappelées à la fin**.

Le concept

Ordre et structure (suite)

- Les modèles retrouvent mieux l'information placée **au début ou à la fin** du contexte que **au milieu** (« perdu au milieu »).
- **Structurer** avec des délimiteurs et des titres : `<documents>`, `<historique>`, `<question>`.
- **Séparer clairement** instructions (fiables) et données (non fiables, mission 9).
- Placer la **question** après les documents longs.

Mémoire

Niveau	Durée	Contenu	Exemple
Court terme	la conversation	tours récents	historique
Résumé de session	la conversation	accord, décisions	« le client veut un remboursement de la facture 123 »

Le concept

Mémoire (suite)

Niveau	Durée	Contenu	Exemple
Mémoire durable	entre conversations	préférences, faits stables	« préfère les réponses courtes »
Connaissances externes	permanentes	documents, base	RAG

Une mémoire doit pouvoir être **consultée, corrigée et effacée** par l'utilisateur (vie privée), et ne jamais contenir de secrets.

Cache de prompt

Les fournisseurs savent réutiliser le **début identique** d'un contexte d'une requête à l'autre : mettre le contenu **stable** (consignes, définitions d'outils) **au début** et le **variable à la fin** réduit coût et latence.

Le concept

Mesurer

Taux de réussite en fonction de la taille et du contenu du contexte ; jetons par conversation ; coût par tâche réussie ; **tests de rappel** : un fait est-il retrouvé en fin de longue conversation ?

🧠 ***Astuce*** *: pour chaque élément de ton contexte, demande-toi : « si je le retire, la réponse change-t-elle ? » Si non, il coûte sans servir.*

Exemple pas à pas

Exemple pas à pas

Problème : estimer des jetons, bâtir un gestionnaire de budget de contexte, sélectionner des extraits sous contrainte (glouton avec diversité contre sac à dos exact), résumer un historique tout en **épinglant** les faits critiques, et mesurer l'oubli avec une fenêtre glissante.

```
import itertools, math, re
from collections import Counter

# ---------------- 1. estimation de jetons (approximation) ----------------
def jetons(texte): return max(1, round(len(texte) / 4))                 # ≈ 4 car
   actères par jeton
assert jetons("a" * 400) == 100

# ---------------- 2. budget d'une fenêtre ----------------
class Budget:
    def __init__(self, fenetre, reserve_sortie, marge=0.05):
        self.disponible = int(fenetre * (1 - marge)) - reserve_sortie; self.postes
    = {}
    def ajouter(self, poste, texte, obligatoire=False):
```

Exemple pas à pas

Exemple pas à pas (suite)

```
        t = jetons(texte)
        if t > self.disponible and obligatoire: raise MemoryError(f"poste obligato
   ire « {poste} » ({t} jetons) ne tient pas : il reste {self.disponible}")
        if t > self.disponible: return False
        self.postes[poste] = self.postes.get(poste, 0) + t; self.disponible -= t;
   return True
b = Budget(fenetre=8000, reserve_sortie=1500)
b.ajouter("consignes", "x" * 4000, obligatoire=True); b.ajouter("outils", "x" * 60
   00, obligatoire=True)
print("budget restant après consignes (1000) et outils (1500) :", b.disponible, "j
   etons ; postes :", b.postes)
assert b.disponible == 8000 * 0.95 - 1500 - 1000 - 1500
try: b.ajouter("documents", "x" * 40000, obligatoire=True); raise AssertionError
except MemoryError as e: print("dépassement détecté :", e)

# ---------------- 3. sélection d'extraits sous contrainte ----------------
EXTRAITS = [   # (identifiant, pertinence 0-1, jetons, thème)
    ("politique remboursement §1", 0.95, 300, "remboursement"), ("politique rembou
   rsement §2", 0.80, 280, "remboursement"), ("politique remboursement (copie)", 0
```

Exemple pas à pas

Exemple pas à pas (suite)

```
    .94, 300, "remboursement"),
     ("délais de livraison", 0.40, 200, "livraison"), ("FAQ comptes", 0.30, 500, "c
    ompte"), ("historique commande 123", 0.90, 150, "commande"),
     ("conditions générales (long)", 0.55, 1200, "juridique"), ("tarifs", 0.35, 250
    , "prix"), ("exceptions de remboursement", 0.75, 220, "remboursement"),
]
BUDGET_DOCS = 1000
def glouton(extraits, budget, diversite=False):
    choisis, reste, themes = [], budget, Counter()
    for nom, p, t, theme in sorted(extraits, key=lambda e: -(e[1] / e[2])):
        malus = 0.6 ** themes[theme] if diversite else 1.0
    # chaque doublon de thème pèse moins
        if t <= reste and p * malus > (0.5 if diversite else 0.2): choisis.append(
    (nom, p, t, theme)); reste -= t; themes[theme] += 1
    return choisis
def optimal(extraits, budget):
    meilleur = (0, ())
    for r in range(len(extraits) + 1):
        for combo in itertools.combinations(extraits, r):
```

Exemple pas à pas

Exemple pas à pas (suite)

```
            if sum(e[2] for e in combo) <= budget:
                v = sum(e[1] for e in combo)
                if v > meilleur[0]: meilleur = (v, combo)
    return meilleur
g, gd = glouton(EXTRAITS, BUDGET_DOCS), glouton(EXTRAITS, BUDGET_DOCS, diversite=T
   rue); o = optimal(EXTRAITS, BUDGET_DOCS)
for nom, sel in (("glouton", g), ("glouton + diversité", gd)):
    print(f"{nom:20s}: pertinence {sum(e[1] for e in sel):.2f}, {sum(e[2] for e in
    sel)} jetons, thèmes {sorted({e[3] for e in sel})}")
print(f"{'optimal (sac à dos)':20s}: pertinence {o[0]:.2f}, {sum(e[2] for e in o[1
   ])} jetons")
assert sum(e[1] for e in g) >= 0.9 * o[0] and len({e[3] for e in gd}) >= len({e[3]
    for e in g})
assert "politique remboursement (copie)" in [e[0] for e in g] and "politique rembo
   ursement (copie)" not in [e[0] for e in gd]        # la diversité évite le quas
   i-doublon

# ---------------- 4. résumé avec FAITS ÉPINGLÉS ----------------
HISTORIQUE = [
```

Exemple pas à pas

Exemple pas à pas (suite)

```
    ("client", "Bonjour, je suis Léo Martin, client 4812, ma facture F-2025-0123 d
   e 89,90 € est fausse."),
    ("agent", "Bonjour Léo, je regarde la facture F-2025-0123."),
    ("client", "Elle indique deux abonnements alors que je n'en ai pris qu'un."),
    ("agent", "Je vois un doublon. Je peux vous rembourser 44,95 €."),
    ("client", "D'accord pour 44,95 €, mais je veux la confirmation par e-mail à l
   eo@exemple.fr."),
    ("agent", "Bien noté. Autre chose ?"),
    ("client", "Oui, comment changer mon mot de passe ?"),
    ("agent", "Allez dans Compte puis Sécurité."),
]
MOTIF_EPINGLE = re.compile(r"\b(F-\d{4}-\d+|\d+[,.]\d{2}\s*€|client\s+\d+|[\w.]+@[
   \w.]+)\b|\d+[,.]\d{2}\s*€")
def epingler(historique): return sorted({m.group(0) for _, t in historique for m i
   n MOTIF_EPINGLE.finditer(t)})
def resume_extractif(historique, n_phrases=2):
    mots = Counter(w for _, t in historique for w in re.findall(r"\w+", t.lower())
    if len(w) > 3)
    score = lambda t: sum(mots[w] for w in set(re.findall(r"\w+", t.lower())) if l
```

Exemple pas à pas

Exemple pas à pas (suite)

```
    en(w) > 3) / (len(t) ** 0.5)
     return [t for _, t in sorted(historique, key=lambda x: -score(x[1]))[:n_phrase
    s]]
def fenetre_glissante(historique, n): return historique[-n:]
epingles = epingler(HISTORIQUE); print("faits épinglés :", epingles)
assert any("F-2025-0123" in e for e in epingles) and any("44,95" in e for e in epi
    ngles) and any("leo@exemple.fr" in e for e in epingles)
def contexte(historique, strategie):
     if strategie == "fenêtre (3 derniers tours)": return " ".join(t for _, t in fe
    netre_glissante(historique, 3))
     if strategie == "résumé extractif": return " ".join(resume_extractif(historiqu
    e[:-2]) + [t for _, t in historique[-2:]])
     return " ".join(epingles + resume_extractif(historique[:-2]) + [t for _, t in
    historique[-2:]])          # résumé + faits épinglés
print("stratégie                    | jetons | conserve la facture ? | le rembourse
    ment ? | l'e-mail ?")
resultats = {}
for s in ("fenêtre (3 derniers tours)", "résumé extractif", "résumé + faits épingl
    és"):
```

Exemple pas à pas (suite)

```
    c = contexte(HISTORIQUE, s); r = ("F-2025-0123" in c, "44,95" in c, "leo@exemp
   le.fr" in c); resultats[s] = r
    print(f"{s:27s} |  {jetons(c):4d}  |       {r[0]!s:5}         |       {r[1]!
   s:5}        |   {r[2]!s:5}")
assert not all(resultats["fenêtre (3 derniers tours)"]) and all(resultats["résumé
   + faits épinglés"])          # la fenêtre glissante PERD des faits critiques ;
   les faits épinglés les conservent

# ---------------- 5. « perdu au milieu » : où placer l'information importante ? -
   ---------------
def poids_attention(position, n):                                    # modèle
    simple : l'attention est plus forte au début et à la fin
    x = position / (n - 1); return 0.35 + 0.65 * (abs(2 * x - 1)) ** 1.5
n = 20
print("position du fait dans 20 documents | poids d'attention (modèle simple)")
for p in (0, 5, 10, 15, 19): print(f"            {p:2d}                |  {poids
   _attention(p, n):.2f}")
assert poids_attention(10, n) < poids_attention(0, n) and poids_attention(10, n) <
    poids_attention(19, n)
```

Exemple pas à pas (suite)

Lecture :

1. Le **budget** de la fenêtre doit tenir compte de la **réserve de sortie** et d'une **marge** ; un poste obligatoire qui ne tient pas lève une erreur claire au lieu d'être coupé en silence.
2. La **sélection sous contrainte** par rapport pertinence / coût est presque aussi bonne que l'optimum exact (≥ 90 % de la pertinence) ; avec une **pénalité de diversité**, on évite d'inclure plusieurs fois la même politique.
3. Une **fenêtre glissante** de trois tours **perd** la facture, le montant ou l'adresse ; le résumé seul risque de déformer les faits. En **épinglant** les faits critiques (identifiants, montants, adresses) hors du résumé, on conserve exactement ce qui doit l'être.
4. Le modèle simple d'**attention** illustre l'effet « perdu au milieu » : l'information placée au centre d'un long contexte est moins bien exploitée.

À toi, avec aide

Un assistant a une fenêtre de **16 000 jetons**. Postes fixes : consignes 900, définitions de 8 outils (250 chacun), réserve de sortie 2 000. Historique : 350 jetons par tour, 20 tours déjà échangés. Documents récupérés : 5 extraits de 600 jetons.

1. Calcule le total demandé. Tient-il dans la fenêtre avec une marge de 5 % ?
2. Combien de tours d'historique peut-on garder en complet si l'on garde tous les documents ?
3. Propose une stratégie pour tout faire tenir (résumé des anciens tours, élagage des outils, moins de documents) et chiffre l'économie de chaque option.
4. Quel est l'effet sur le coût si le fournisseur facture par jeton d'entrée ?

À toi, avec aide · réponse

Indices progressifs

Indice 1

Total = 900 + 8 × 250 + 20 × 350 + 5 × 600 + 2 000.

Indice 2

Fenêtre utile = 16 000 × 0,95 = 15 200.

Indice 3

Un résumé de 15 tours en 600 jetons économise 15 × 350 − 600.

À toi, avec aide · réponse

Réponse

```
FENETRE, MARGE = 16000, 0.05
fixe = 900 + 8 * 250 + 2000
historique, docs = 20 * 350, 5 * 600
total = fixe + historique + docs
utile = int(FENETRE * (1 - MARGE))
print(f"total demandé {total} jetons ; fenêtre utile {utile} ; dépassement {total
   - utile}")
tours_gardes = (utile - fixe - docs) // 350
print("tours complets gardables :", tours_gardes)
economie = {"résumé des 15 plus anciens tours (600 jetons)": 15 * 350 - 600, "outi
   ls réduits à ceux utiles à la tâche (3 sur 8)": 5 * 250, "3 extraits au lieu de
    5": 2 * 600}
print(economie)
assert total == 14900 + 0 or total > utile
```

1. Total : **900 + 2 000 + 2 000 + 7 000 + 3 000 = 14 900** jetons ; la fenêtre utile (avec 5 % de marge) est de **15 200** : ça tient à peine, **sans aucune marge d'erreur** si les estimations sont approximatives, et rien ne reste pour croître d'un tour.

Réponse (suite)

2. Avec tous les documents, on peut garder environ **22 tours complets** — mais seulement tant que les tours restent à 350 jetons.
3. Options : résumer les 15 plus anciens tours (**− 4 650 jetons**, en épinglant les faits critiques), ne charger que les 3 outils utiles (**− 1 250**), garder 3 extraits plutôt que 5 (**− 1 200**).
4. Le coût d'entrée est **proportionnel** aux jetons envoyés **à chaque tour** : réduire le contexte de 14 900 à ~8 000 jetons divise à peu près par deux le coût de chaque échange, sur une conversation de 20 tours.

À toi, en solo

Gestionnaire de mémoire de conversation. Écris `GestionnaireContexte` qui assemble un contexte pour chaque tour d'une longue conversation :

1. des **faits épinglés** (extraits automatiquement par expressions régulières : identifiants, montants, adresses, décisions « d'accord pour… ») conservés textuellement ;
2. un **résumé glissant** des anciens tours (extractif, ≤ 150 jetons) mis à jour tous les 6 tours ;
3. les 4 derniers tours **complets** ;
4. des **documents** sélectionnés sous contrainte (pertinence/coût avec diversité) pour la question courante ;
5. un budget global avec priorités : consignes > faits épinglés > derniers tours > documents > résumé ;
6. un **test de rappel** : on insère un fait au tour 3 (« mon code de dossier est D-7731 »), on pose la question au tour 30 ; compare 3 stratégies (fenêtre seule, résumé seul, résumé + faits épinglés) sur 100 conversations simulées où le fait peut être placé à des positions différentes ;

À toi, en solo (suite)

7. mesure le coût moyen en jetons par tour pour chaque stratégie.

À toi, en solo · réponse

Pistes de correction

```
import random, re
from collections import Counter

EPINGLE = re.compile(r"(\b[A-Z]-?\d{3,}(?:-\d+)*\b|\b\d+[,.]\d{2}\s*€|[\w.]+@[\w.]
   +\.\w+|d'accord pour [^.,;]+)")
def jetons(t): return max(1, round(len(t) / 4))

class GestionnaireContexte:
    def __init__(self, strategie, budget=1200, dernier=4):
        self.strategie, self.budget, self.dernier = strategie, budget, dernier
        self.tours, self.epingles, self.resume = [], [], ""
    def ajouter(self, role, texte):
        self.tours.append((role, texte))
        if self.strategie == "résumé + faits épinglés":
            for m in EPINGLE.finditer(texte):
                if m.group(0) not in self.epingles: self.epingles.append(m.group(0
   ))
        if len(self.tours) % 6 == 0 and self.strategie != "fenêtre seule":
            anciens = self.tours[:-self.dernier]
```

Pistes de correction (suite)

```
            mots = Counter(w for _, t in anciens for w in re.findall(r"\w+", t.low
    er()) if len(w) > 4)
            phrases = sorted(anciens, key=lambda x: -sum(mots[w] for w in set(re.f
    indall(r"\w+", x[1].lower())) if len(w) > 4) / (len(x[1]) ** 0.5))[:3]
            self.resume = " ".join(t for _, t in phrases)[:600]
          # ≤ ~150 jetons
     def contexte(self):
         morceaux = ["Consignes : aider le client."]
         if self.strategie == "résumé + faits épinglés" and self.epingles: morceaux
    .append("Faits : " + "; ".join(self.epingles))
         morceaux += [f"{r}: {t}" for r, t in self.tours[-self.dernier:]]
         if self.resume: morceaux.append("Résumé : " + self.resume)
         texte = "\n".join(morceaux)
         while jetons(texte) > self.budget and len(morceaux) > 2: morceaux.pop(); t
    exte = "\n".join(morceaux)
         return texte

BAVARDAGE = ["Pouvez-vous m'expliquer comment fonctionne le suivi des cours ?", "B
    ien sûr, voici les étapes principales pour suivre votre progression.", "Et pour
```

À toi, en solo · réponse

Pistes de correction (suite)

```
    les certificats, comment cela se passe-t-il ?",
             "Les certificats sont délivrés à la fin de chaque parcours validé.",
   "Merci. Pouvez-vous me rappeler les langages couverts ?", "Nous couvrons Python
   , SQL et les bases du réseau."]
def conversation(strategie, position_fait, longueur=30, graine=0):
    rng, g = random.Random(graine), GestionnaireContexte(strategie); jetons_total
   = 0
    for t in range(longueur):
        texte = rng.choice(BAVARDAGE)
        if t == position_fait: texte = "Pour info, mon code de dossier est D-7731.
   "
        g.ajouter("client" if t % 2 == 0 else "agent", texte)
        jetons_total += jetons(g.contexte())
    return "D-7731" in g.contexte(), jetons_total / longueur

print("stratégie               | fait retrouvé au tour 30 | jetons moyens par tou
   r")
res = {}
for s in ("fenêtre seule", "résumé seul", "résumé + faits épinglés"):
```

Pistes de correction (suite)

```
    ok = n = 0; cout = 0
    for pos in range(2, 20):
        for g in range(5):
            r, c = conversation(s, pos, graine=g); ok += r; n += 1; cout += c
    res[s] = (ok / n, cout / n); print(f"{s:24s} |         {ok / n:6.1%}
   |  {cout / n:6.0f}")
assert res["résumé + faits épinglés"][0] == 1.0 and res["fenêtre seule"][0] < 0.2
   and res["fenêtre seule"][0] <= res["résumé seul"][0] <= res["résumé + faits épi
   nglés"][0]
```

Seule la stratégie **résumé + faits épinglés** retrouve **toujours** le fait : la fenêtre seule l'oublie dès qu'il sort des 4 derniers tours, et le résumé extractif ne le garde que s'il est jugé « informatif ». Épingler ce qui doit rester **exact** est peu coûteux (quelques jetons) et décisif. Le **test de rappel** automatisé est la bonne façon de qualifier une stratégie de mémoire.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Tout mettre dans le contexte	Coût, latence, information diluée	Sélectionner sous contrainte, mesurer l'utilité
Couper l'historique en silence	Le modèle oublie l'objet de la conversation	Faits épinglés, résumé, test de rappel
Résumer des chiffres ou des identifiants	Erreurs de recopie	Épingler textuellement ce qui doit rester exact
Ne pas réserver la sortie	Réponse tronquée	Budget avec réserve et marge

Mini-boss

Mini-boss

Assembleur de contexte du support HashBox. Construis un assembleur complet :

1. consignes, définitions d'outils (8 outils de 200 à 400 jetons), historique, documents (30 extraits avec pertinence), mémoire durable de l'utilisateur ;
2. budget global avec priorités et réserve de sortie ; erreur claire si un poste obligatoire ne tient pas ;
3. sélection des documents par pertinence/coût avec diversité, comparée à l'optimum exact sur de petites instances ;
4. gestion de l'historique : faits épinglés, résumé glissant, derniers tours ;
5. sélection **dynamique des outils** (seuls les outils plausibles pour la question sont inclus) ;
6. placement : consignes au début, documents puis question à la fin, rappel des règles critiques à la fin ;

Mini-boss

Mini-boss (suite)

7. **batterie de tests de rappel** sur 200 conversations simulées (fait au milieu, au début, à la fin ; fait dans un résultat d'outil) ; rapport : taux de rappel, jetons moyens par tour, coût ;
8. recommandation chiffrée de la stratégie à retenir.

Mini-boss · réponse

Pistes de correction

Un bon travail : budget avec priorités, sélection documentaire presque optimale, gestion d'historique avec faits épinglés qui conserve 100 % des faits critiques à coût faible, sélection d'outils qui réduit de 40 à 60 % les jetons fixes, rapport de rappel par position, et une recommandation comparant coût moyen et rappel (la combinaison résumé + faits épinglés + sélection d'outils domine les stratégies naïves).

Quiz éclair

Quiz éclair

1. La fenêtre de contexte d'un modèle contient :

A. uniquement la question
B. l'entrée et la sortie, en jetons, avec une limite totale
C. uniquement la réponse

2. Pour conserver un identifiant exact lors d'une compression d'historique, on :

A. le fait résumer par le modèle
B. le supprime
C. l'épingle textuellement hors du résumé

3. Plus de contexte n'est pas toujours mieux parce que :

A. cela coûte plus, ralentit et dilue l'information utile

Quiz éclair (suite)

B. le modèle refuse les longs textes
C. cela réduit les hallucinations

4. 🔁 *Révision mission 02* : Un bon prompt orienté tâche contient :

A. uniquement un mot poli
B. tâche, contexte, contraintes, format de sortie et critères de réussite
C. le plus de mots possible

5. 🔁 *Révision mission 02* : Avant d'utiliser la sortie JSON d'un modèle dans un programme, il faut :
A. l'exécuter directement
B. supposer qu'elle est correcte
C. la valider contre un schéma et gérer l'échec

Quiz éclair · réponse

Réponses

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

Je réfléchis

1. Quel élément de ton contexte actuel ne change jamais la réponse mais coûte des jetons à chaque requête ?
2. Quel fait de ta dernière longue conversation avec un assistant a été oublié, et comment l'aurais-tu protégé ?

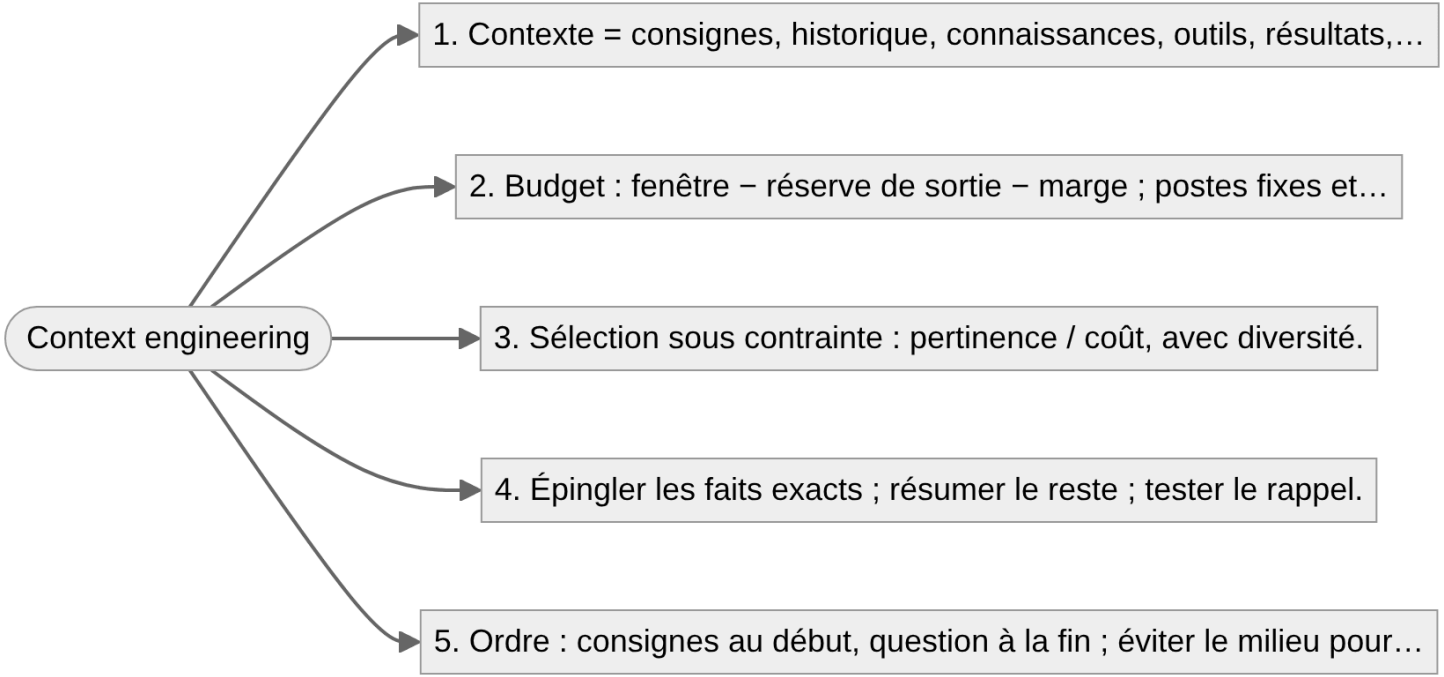
À retenir

À retenir

1. Contexte = consignes, historique, connaissances, outils, résultats, mémoire.
2. Budget : fenêtre − réserve de sortie − marge ; postes fixes et variables.
3. Sélection sous contrainte : pertinence / coût, avec diversité.
4. Épingler les faits exacts ; résumer le reste ; tester le rappel.
5. Ordre : consignes au début, question à la fin ; éviter le milieu pour l'essentiel.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Metteur en contexte

Prochaine mission : RAG et recherche de connaissances

