Pipeline RAG

Mode deckbuilding

Mode deckbuilding

> *Dernière mise à jour : 2026-05-10*

Depuis 2026-04-24 (Axe 2 Phase 1), un 4e intent `'deckbuilding'` s'ajoute à `'rules' | 'synergy' | 'meta'`. Il déclenche un pipeline dédié pour produire des **decklists structurées exécutables** (60 cartes MTG, 40+12+3 Riftbound, 60+sideboard FAB) là où le RAG classique produisait de la prose d'analyse.

Flux (`src/services/rag/deckbuilding/`)

1. Classification (`classify.ts`)

Haiku retourne `'deckbuilding'` quand la question exige une **liste exhaustive avec un nombre de cartes précis** ("decklist complète", "40 + 12 runes", "avec sideboard").

À distinguer de `synergy` qui reste une discussion d'archétype en prose.

Ordre de priorité : `deckbuilding > meta > synergy > rules`.

2. Dispatch (`answer.ts`)

Early-return vers `askDeckbuilding()` juste après résolution de l'intent. Le retrieval standard est jeté, askDeckbuilding relance son propre retrieval avec `intent='synergy'` forcé.

3. Spec Haiku (`deckbuilding/spec.ts`)

Un appel Haiku produit un `DeckSpec` (Zod) structuré :

- format
- tailles (mainboard, sideboard, runes, battlefields, equipment)
- max copies (généralement 4 MTG, 3 Riftbound, etc.)
- rôles cibles (`creatures`, `spells`, `lands`, `equipment`...)
- anchor cards (cartes obligatoires)

Fallback defaults par jeu si Haiku renvoie un JSON invalide :

- MTG Standard : `60 + 15, max 4`
- Riftbound : `40 + 0 + 3 + 12r + 3bf`
- FAB CC : `60 + 0, max 3`

4. Anchor cards

Les fresh + sticky mentions (@-cartes de l'Axe 1) deviennent **obligatoires** dans la decklist. Le prompt Opus a une règle stricte « tu DOIS inclure ces cartes ». Log warning serveur si une anchor est absente de la réponse finale.

5. Retrieval synergy

`retrieveChunks()` est appelé avec `intent='synergy'` → déclenche l'expansion mécanique (`retrieve/synergy-expansion.ts`). `retrieveChunks` accepte `'deckbuilding'` mais le mappe vers `'synergy'` en interne (fallback safe).

6. Prompt Opus spécialisé (`prompt-deckbuilding.ts`)

`DECKBUILDING_SYSTEM_PROMPT` impose :

- total exact par section
- max copies non-basiques (basic lands FR/EN whitelistés)
- anchor cards obligatoires
- pool strict (uniquement cartes du contexte)
- format markdown par rôles : `### Créatures (24)` + `4× [[card:Nom]]`

7. userPrompt assemblage

Ordre des blocs :

1. `SPEC DU DECK` (JSON Haiku)
2. `CARTES OBLIGATOIRES (ANCHORS)` (anchors fresh + sticky)
3. `CARTES CITÉES PAR LE JOUEUR (ce tour)` (full)
4. `CARTES ÉVOQUÉES DANS LA CONVERSATION` (sticky abrégées)
5. `DECKLISTS DE TOURNOI` (mainboard + sideboard complets, jusqu'à 5)
6. `POOL DE CARTES ÉLIGIBLES` (organisé par rôle, cap 30 par rôle)
7. `CARTES CANDIDATES` (chunks `[CARD]` + `[META]`, pas `[BASE]`/`[EXT]`)

8. Génération Opus

`promptStream(DECKBUILDING_SYSTEM_PROMPT, userPrompt, 'opus')` streamé. Post-processing : `autoWrapCardMentions` + `buildWrapAliases` + enrichissement extraCards.

9. Pool exhaustif + decklists d'inspiration (Phase 2, 2026-04-24)

- `fetchEligibleCards()` (`deckbuilding/pool.ts`) scroll Qdrant avec filtre structurel :
  - MTG : `card_mtg_legal_formats` + post-filtre `card_mtg_color_identity ⊆ spec.colors`
  - FAB : `card_legal_heroes`
  - Riftbound : sans filtre Qdrant (collection mono-format) + post-filtre `card_domains ⊆ spec.domains`
- `fetchInspirationDecklists()` : hybrid search (dense+sparse) sur `meta_type='tournament_deck'` filtré par `meta_format` — jusqu'à 5 decklists similaires (infra : MTGTop8 pour MTG, RiftboundStats pour Riftbound, fabtcg.com pour FAB)
- `selectRoleCandidates()` : pré-filtre chaque rôle de la spec à **cap 30 par rôle**, priorité aux cartes déjà vues dans les decklists d'inspiration
- TM / Ark Nova : pas de champs structurels → retombent silencieusement sur Phase 1 (retrieval synergy seul)
- `scrollAllPoints()` accepte `opts.filter` et `opts.withVector` (défaut true — Phase 2 passe `false` pour économiser ~120 MB sur MTG 30k cartes)

10. Validation structurelle + retry auto (Phase 3, 2026-04-24)

- `parseDeckResponse()` : extrait les cartes par section (main / sideboard / runes / battlefields) via regex stricte avec `×/x/X` obligatoire — les lignes de prose ne matchent pas
- `validateDeck()` :
  - Total exact par section

    - Max copies non-basiques (basic lands whitelist FR/EN : Plains/Island/Swamp/Mountain/Forest/Wastes + Plaine/Île/Marais/Montagne/Forêt)
    - Anchor cards présentes (≥1 copie)
    - Cartes inconnues du catalogue → warning non bloquant
- Si validation échoue : `formatRetryPrompt()` construit nouveau userPrompt listant les issues + contraintes strictes
- Opus relancé jusqu'à **2 retries** (3 tentatives max)
- Pendant retry : token séparateur visible streamé pour transparence — `done.fullAnswer` écrase le content côté frontend, l'utilisateur final voit uniquement la version corrigée
- Si après 3 tentatives la decklist reste invalide : on garde la dernière + warning log serveur (pas d'échec total)

11. Frontend

Aucun changement. Le rendu markdown gère titres + listes + `Nx [[card:X]]` (texte standard). `CardZoomModal` fonctionne automatiquement.

Hors scope Phases 1/2/3

- Pas de composant `<DeckListView>`
- Pas d'export `.dec` ni bouton Copier
- Pas de badge de validation côté frontend (verdict logué serveur, pas exposé via `done`)
- Pas de support TM / Ark Nova (pas d'équivalent format/archétype dans leurs données)

Helpers exportés

`autoWrapCardMentions`, `buildWrapAliases`, `renderFullCard`, `renderShortCard` sont exportés depuis `answer.ts` pour réutilisation par `deckbuilding/ask.ts`.

Flux d'ingestion (PDF → Qdrant)

Flux d'ingestion (PDF → Qdrant)

> *Dernière mise à jour : 2026-05-10*

Source de vérité : `ARCHITECTURE.md` § "Flux d'ingestion".

11 étapes

```
POST /api/games/ingest [multipart : PDF + metadata]
   |
1. Upload PDF → /app/pdfs/<slug>-<timestamp>.pdf
   |   └─ Choix type : base / extension / advanced_rules / faq
   |
2. Extraction texte (pdfjs-dist → RawPage[])
   |
3. OCR conditionnel (services/ocr/)
   |   ├─ decideOCR(rawPages) : avg chars/page < OCR_AUTO_THRESHOLD ?
   |   |   ├─ disabled (OCR_ENABLED=false)
   |   |   ├─ no_text (0 page texte → PDF entièrement scanné)
   |   |   ├─ low_text (couche texte cassée)
   |   |   └─ sufficient (skip OCR)
   |   ├─ Si shouldOCR=true :
   |   |   ├─ pdftoppm rend chaque page en PNG 300 DPI
```

```
   │   │   ├─ Pool tesseract (OCR_CONCURRENCY workers, défaut 2)
   │   │   ├─ Langues OCR_LANGUAGES (défaut fra+eng)
   │   │   ├─ Cache JSON ocr-v1-<slug>.json (sauvé tous les 3 pages)
   │   │   └─ Confiance log warn si < OCR_MIN_CONFIDENCE (40)
   │   └─ Réinjection texte OCR dans pipeline normal
   │
4. Découpage sémantique (chunking/chunker.ts)
   │   └─ Paragraphes + overlap, fusion intelligente
   │
5. Hierarchy LLM (hierarchy.ts)
   │   └─ Claude analyse le doc complet → arbre chapter > section > sub
   │   └─ Chaque chunk reçoit hierarchy_path + hierarchy_level
   │
6. Contextual Retrieval B (contextual-llm.ts)
   │   ├─ Pour chaque chunk : Claude reçoit DOC + hierarchy_path
   │   ├─ Génère 1-2 phrases de contexte
   │   ├─ 10 SSH parallèles (pool worker)
   │   ├─ Cache JSON contexts-v2-<slug>.json (reprise après crash/quota)
   │   └─ Le contexte préfixe le chunk avant embedding
   │
7. Embeddings TEI bge-m3 (batch 32)
   │   └─ Vecteur dense 1024 par chunk (préfixé par contexte LLM)
   │
8. Upsert Qdrant (collection rules_<slug>)
   │   └─ Payload : hierarchy_path, page_start/end, is_extension, contextual_text, etc.
   │
9. Progression SSE (jusqu'à 7 actes UI)
   │   ├─ Lecture
   │   ├─ [Reconnaissance optique]  ← uniquement si OCR a tourné
   │   ├─ Découpage
   │   ├─ Structuration
   │   ├─ Analyse contextuelle
   │   ├─ Compréhension (embeddings)
   │   └─ Archivage
   │
10. [Optionnel, extensions seulement] Détection conflits (conflict-detect.ts)
   │   ├─ Pour chaque chunk extension : recherche hybride dans le jeu de base
   │   ├─ Si similarité > CONFLICT_SIMILARITY_THRESHOLD (0.65) :
   │   │   └─ Claude SSH (CONFLICT_MODEL) classifie : replaces / modifies / extends
```

```
   │   ├─ 10 SSH parallèles
   │   ├─ Cache JSON conflicts-v1-<slug>.json
   │   └─ Annotations stockées dans payload Qdrant
   │       (conflict_type, conflict_base_chunk_id, conflict_summary, conflict_base_page)
   │
11. Rendu PNG des pages (non-fatal)
   │   ├─ pdftoppm 300 DPI → /app/pdfs/images/<slug>/page-XX.png
   │   ├─ Idempotent : skip si déjà rendu par étape 3 (OCR)
   │   └─ Échec → logger.warn, ingestion reste valide
```

Patterns importants

- **Coordinator + stages** : `services/ingest/coordinator.ts` orchestre, chaque stage est un fichier dans `stages/` avec signature `(game, ..., push: EventPusher) => Promise<...>`.
- **Erreurs non-fatales** : try/catch local dans le stage ou autour de l'appel dans le coordinator (cas de `renderPdfPages`).
- **Reprise après quota** : tous les stages SSH parallèles flushent le cache JSON avant de propager `ClaudeQuotaError`. Le coordinator transforme l'erreur en `ingestStatus='scheduled'` + `scheduleIngestStart(gameId, retryAt)` (pas `'error'`).
- **Auto-OCR transparent** : pas de toggle UI. Si `OCR_ENABLED=true` (défaut) et seuil dépassé, l'OCR tourne. L'acte "Reconnaissance optique" n'apparaît dans l'UI que si l'OCR a effectivement tourné.

Ingestion planifiée

`scheduled_start_at` (ISO, max +7 jours) dans le POST :

1. Ligne `games` créée avec `ingestStatus='scheduled'` + `ingestScheduledAt`
2. PDF écrit sur disque
3. `setTimeout` armé dans `cron/ingest-scheduler.ts`
4. Au boot : `restoreScheduledOnBoot()` re-programme tous les timers en attente
5. Si date passée pendant downtime → start immédiat avec grace 1s

`DELETE /api/games/:id/scheduled` annule (clear timer + supprime PDF + supprime ligne). 409 si pas en `scheduled`.

Forums BGG (cron quotidien)

Si `BGG_FORUM_SYNC_ENABLED=true` :

1. Cron à `BGG_FORUM_SYNC_HOUR` (défaut 3h)
2. Pour chaque jeu avec `bgg_id` en BDD :
   - `getBggForumList()` trouve le forum "Rules"
   - `getBggForumThreads()` récupère threads (max 3 pages = ~150)
   - Pour chaque thread : articles concaténés (~2000 chars max)
   - Embed (dense + sparse) + upsert dans `rules_<slug>` avec `is_forum_chunk: true`
3. Rate-limit : 5s entre appels BGG

Chunks forum identifiés par `is_forum_chunk: true` + tag `[FORUM]` dans le contexte RAG. Pas de page (`page_start: 0, page_end: 0`).

Filtrage par intent (depuis 2026-04-26) : chunks `is_forum_chunk=true` exclus quand `intent ∈ {synergy, deckbuilding, meta}`. Les forums sont conçus pour clarifier des règles, pas pour discuter deckbuilding.

Flux d'une question (RAG complet)

Flux d'une question (RAG complet)

> *Dernière mise à jour : 2026-05-10*

Source de vérité : `ARCHITECTURE.md` § "Flux principal (question)". Cette page reformule pour vue rapide.

9 étapes

```
POST /api/ask/stream { gameId, question, extensions, history, cardMentions, stickyCardMentions
}
   |
1. Validation Zod (askStreamSchema) — UUID, question ≤500 chars, history ≤5
   |
2. Résolution extensions (IDs → noms via gamesRepo.getById)
   |
3. Résolution cartes @-citées (resolveMentions) — fresh + sticky
   |
4. Classification question (classify.ts via Haiku, en parallèle du retrieval)
   ├─ Output: 'rules' | 'synergy' | 'meta' | 'deckbuilding'
   └─ Si timeout → fallback 'rules'
   |
5. RETRIEVAL (orchestrator.ts retrieveChunks)
```

```
   │
   ├─ Étape 0 — Embeddings question
   │   ├─ Question brute → TEI bge-m3 (1024 dense)
   │   └─ HyDE → Haiku passage hypothétique → TEI bge-m3
   │       (timeout 25s, fallback : query brute seule)
   │
   ├─ Étape 1 — Recherches hybrides parallèles
   │   ├─ Pour chaque collection (base + extensions cochées) :
   │   │   ├─ Dense bge-m3 (cosine)
   │   │   └─ Sparse BM25 natif Qdrant (keyword match)
   │   │
   │   ├─ Fusion RRF v2 (RAG_FUSION_V2_ENABLED)
   │   │   ├─ Score = 2/(rank_dense+2) + 1/(rank_bm25+2)  ← question×2 vs HyDE×1
   │   │   ├─ Top-rank bonus : +0.05 si rank=0, +0.02 si rank ∈ {1,2}
   │   │   └─ Dedup par pointId
   │   │
   │   ├─ [Si intent=meta] meta-retrieval
   │   │   └─ Recherche chunks 17Lands / MTGGoldfish / etc. (filter meta_format)
   │   │
   │   └─ [Si MTG/FAB/Riftbound] synergy-expansion
   │       ├─ decompose-query (Haiku) → JSON {couleurs, format, types, themes}
   │       ├─ Filters Qdrant natifs (payload matching)
   │       ├─ Multi-query TCG (synonymes/variantes)
   │       └─ set-matcher : "Strixhaven" → "STX"
   │
   └─ Étape 2 — Reranking cross-encoder (bge-reranker-v2-m3)
       ├─ Re-classe top-50 candidats RRF
       └─ Blending position-aware
           ├─ Top 1-3 : 75% RRF + 25% rerank (préserve exact-match)
           ├─ Top 4-10 : 60/40
           └─ Top 11+ : 40/60
   │
6. Enrichissement contexte (answer.ts)
   ├─ BGG metadata (mechanics, categories) si dispo
   ├─ Image PNG : 1 page max (resolvePageImageFile)
   │   └─ Granularité (livret, page) — pas seulement page
   ├─ Cartes @-mentionnées (bloc CARTES CITÉES PAR LE JOUEUR (ce tour))
   ├─ Sticky cards (bloc CARTES ÉVOQUÉES DANS LA CONVERSATION, format abrégé ~100 tokens)
   └─ Historique (≤5 turns)
```

```
   │
7. Génération réponse Claude (claude-ssh.ts promptStream)
   ├─ Construction userPrompt (chunks + cartes + image + historique)
   ├─ SSH oracle@VM : claude -p --append-system-prompt ... --output-format stream-json
   │    └─ ⚠ --append (pas --system-prompt) : préserve le contrat outils (Read, etc.)
   ├─ withTimeout HYDE_TIMEOUT_MS, fallback Haiku/Opus
   └─ Stream JSON verbose → yield tokens (event.type='assistant')
   │
8. Événements SSE streamés au client
   ├─ meta — questionId (1er event, pour fallback polling)
   ├─ phase — "L'Oracle pèse votre question", "plonge dans le grimoire", etc.
   ├─ context — chunks retrouvés + scores + mentionedCards + stickyMentionedCards
   ├─ token — streaming réponse
   ├─ heartbeat — toutes les 8s (anti idle-timeout NPM)
   ├─ diagnostics — {chunks, hyde, timings, bestScore}
   └─ done — {fullAnswer, questionId, bestScore}
   │
9. Persistance (questionsRepo.updateAnswer)
   └─ {answer, bestScore, diagnostics}
```

Optimisations notables

- **Parallélisation Qdrant** (commit `95229af`) : dense + sparse en parallèle, pas séquentiel
- `withTimeout` **partout** (8 sites) : évite blocages de boot et de RAG
- **Fusion RRF v2** : pondération question brute ×2, blending position-aware (cf. ADR-005)
- **Sticky mentions** : maîtrise tokens sur multi-tours (cap FIFO 20 / 80 si deck)
- **Vision 1 image max** : ~30s par image lue par Claude — passer à 2-3 doublait la latence pour gain marginal

Fallback connexion SSE

Si la connexion SSE casse mais qu'on a déjà reçu `meta.questionId` :

1. Le frontend bascule sur polling `GET /api/ask/:questionId` (3s × 15 = 45s max)
2. La réponse est récupérée depuis la BDD telle qu'elle a été persistée
3. L'UI affiche "L'Oracle finalise hors-ligne..." pendant la récupération
4. Si rien après 45s, erreur user-visible

Pattern réutilisable pour tout endpoint streamSSE : exposer `GET /api/<resource>/:id` retournant 200 (prêt) / 202 (pending) / 404.

Pause / reprise après quota Claude

Pause / reprise après quota Claude

> *Dernière mise à jour : 2026-05-10*

Le compte Claude utilisé par la VM oracle a un quota Pro/Team (fenêtre 5h glissante). Quand il sature, on veut **mettre en pause les jobs en cours** et **reprendre auto** à l'heure de reset, sans perdre le travail déjà fait.

Détection (`services/claude-quota.ts`)

Le service parse la sortie du CLI Claude pour détecter le message de quota.

Marqueurs supportés

```
QUOTA_MARKERS = [
  'usage limit reached',
  "you've hit",
  'hit your usage limit',
  'hit your session',
  'rate limit reached',
  // …
```

```
]
```

⚠ Le scan se fait sur **toute** la sortie du CLI, **même quand Claude a émis du texte assistant** — certaines versions du CLI renvoient le message de quota comme une réponse assistant normale au lieu d'exiter en erreur.

Patterns d'extraction de l'heure de reset

Par ordre de priorité :

1. `|<unix_seconds>` — timestamp Unix direct (le plus fiable)
2. `reset at HH:MM AM/PM` (12h)
3. `resets at HH:MM AM/PM`
4. `try again at HH:MM AM/PM`
5. `available again at HH:MM AM/PM`
6. `reset at HH:MM` (24h, sans AM/PM)
7. **Fallback** : `+1h` si rien ne match

→ `ClaudeQuotaError` avec champ `resetAt: Date`.

Pattern à respecter partout où Claude est appelé pendant un job long

Émetteurs (`claude-ssh.ts`, `claude-local.ts`)

En fin de stream, si rien n'a été généré, appeler `throwIfQuota(stderr + streamOutput)` **avant** de yield l'erreur générique. Sinon le quota se transforme en "answer empty" silencieux.

Workers parallèles (pool SSH)

Pour `contextual-llm.ts`, `conflict-detect.ts`, etc. :

```
let quotaError: ClaudeQuotaError | null = null;

while (todoIndex < todo.length && !quotaError) {
  const task = todo[todoIndex++];
  try {
    await processTask(task);
  } catch (err) {
    if (err instanceof ClaudeQuotaError) {
      quotaError = err;
      return;  // sans compter l'échec comme un fallback
                // (le chunk n'est pas en cache, sera retenté à la reprise)
    }
    throw err;  // autres erreurs : propagation normale
  }
}

await Promise.all(workers);
saveCache(...);  // ⚠ TOUJOURS flusher AVANT de throw
if (quotaError) throw quotaError;
```

L'ordre `flush THEN throw` est critique : si on throw avant le flush, la reprise repart des derniers ~20 chunks au lieu de pile où on s'est arrêté.

Coordinator (`ingest/coordinator.ts`)

Dans le `catch` global :

```
catch (err) {
  if (err instanceof ClaudeQuotaError) {
    const resetAt = err.resetAt;
    const retryAt = new Date(resetAt.getTime() + 2 * 60 * 1000);  // grace 2 min

    await gamesRepo.update(gameId, {
      ingestStatus: 'scheduled',
      ingestScheduledAt: retryAt.toISOString(),
    });

    scheduleIngestStart(gameId, retryAt);
```

```
    push('quota_pause', { resetAt, retryAt });

    // ⚠️ NE PAS passer le jeu en 'error' — ce n'est pas un échec, juste une pause
    return;
  }
  // ...erreur générique
}
```

Frontend

L'event SSE `quota_pause` bascule le wizard d'ingestion sur le panneau `step === 'scheduled'` avec `reason: 'quota'` (au lieu de `'user'` pour la planification volontaire). La copie est dédiée :

> "L'oracle se repose. Reprise prévue à HH:MM."

L'utilisateur n'a rien à faire — le scheduler relance automatiquement.

Ajouter un nouveau marker

Si Claude change le wording du message de quota et que la détection rate :

1. Vérifier les logs serveur (`/app/data/logs/server.log`) pour voir le wording exact
2. Étendre `QUOTA_MARKERS` dans `claude-quota.ts`
3. Si le format de l'heure change, étendre aussi `detectQuotaResetTime` (les patterns regex)
4. Ajouter un test unitaire dans `claude-quota.test.ts` (ce service a déjà des tests pour les patterns existants)

Pas dans `'error'`, jamais

Le quota n'est pas un échec applicatif : c'est une pause normale prévue. Toute logique downstream (UI, monitoring, alerting) doit traiter `'scheduled' + reason='quota'` ≠ `'error'`.

Vision inline (Claude lit les PNG au moment Q)

Vision inline (Claude lit les PNG au moment Q)

> *Dernière mise à jour : 2026-05-10*

Depuis 2026-04-11, la vision n'est **plus** un pipeline d'ingestion. Claude reçoit les images PNG **directement au moment de la question**, uniquement pour la page la plus pertinente du retrieval, via son outil `Read` côté VM SSH.

Zéro appel LLM vision à l'ingestion. Zéro description intermédiaire textuelle. Zéro cache d'image légendée.

Flux côté `answer.ts`

1. **Group by** `(livret, page)` : après `retrieveChunks()`, group les chunks non-forum par clé composite `(gameName, pageStart)`. Garde le meilleur score par clé.
   ⚠ **La granularité doit être** `(livret, page)` **et pas seulement** `page` : le retrieval mélange des chunks venant de plusieurs livrets (base + extension + ADV + FAQ), chacun dans son propre dossier PNG (`/app/pdfs/images/<slug>/`). La page 4 de HEAT et la page 4 de Heavy Rain sont deux images distinctes — grouper par page seule renvoie la mauvaise image.
2. **Sélection** : la meilleure paire `(livret, page)` par score est retenue (`MAX_INJECTED_IMAGES = 1`).
   Pourquoi 1 et pas 3 : chaque image coûte ~20-30s de lecture côté Claude (Read tool sur PNG 300 DPI). Passer à 2-3 doublait/triplait la latence pour un gain marginal.
3. **Résolution du fichier** (`resolvePageImageFile()` dans `pdf-images.ts`) :

- `pdftoppm` produit un padding variable (`page-1.png` / `page-01.png` / `page-001.png` selon le total de pages)
- Le helper lit le dossier une fois et cache `{pageNumber → filename}`
- Cache invalidé dans `renderPdfPages` quand un nouveau rendu est lancé

4. **Injection dans le userPrompt** : sous un bloc "IMAGES DES RÈGLES", préfixé par le nom du livret pour que Claude sache quel livret il ouvre :

```
IMAGES DES RÈGLES :
- Heavy Rain, page 4 : /projet/boardgame-pdfs/heavy-rain/page-04.png
```

5. **Instruction SYSTEM_PROMPT** (règle 9) : Claude ouvre l'image avec son outil `Read` **uniquement** quand la question a une dimension visuelle (icônes, couleurs, symboles, schémas, plateau, tuiles...). Sinon il ignore l'image.

⚠ `--append-system-prompt` (PAS `--system-prompt`)

`claude-ssh.ts` lance Claude Code CLI avec `--append-system-prompt` et **pas** `--system-prompt`. C'est critique :

- `--system-prompt` **remplace** le system prompt par défaut de Claude Code, qui contient le contrat d'utilisation des outils (Read, Bash, etc.). Sans ce contrat, Claude garde ses outils techniquement disponibles mais n'en invoque plus aucun proactivement, et finit par halluciner une description à partir du texte seul.
- `--append-system-prompt` ajoute notre `SYSTEM_PROMPT` Oracle **par-dessus** le contrat par défaut. Donc Claude voit les chemins PNG et les ouvre réellement.

C'est un piège facile à reproduire — toujours `--append`.

Volume PNG

- Côté container backend : `/app/pdfs/images/<slug>/page-XX.png`
- Côté VM oracle : monté en read-only sous `/projet/boardgame-pdfs/<slug>/page-XX.png` (chemin contrôlé par `VISION_SSH_IMAGES_PATH`, défaut `/projet/boardgame-pdfs`)
- Le settings.json oracle (`/home/oracle/.claude/settings.json`) doit avoir ce chemin dans `permissions.additionalDirectories` ET dans `permissions.allow` pour que Claude puisse Read.

Frontend : pas de viewer PNG côté user

Le clic sur une citation `[HEAT, p.4 : "..."]` ouvre toujours **le PDF** (pas le PNG) à la bonne page via `PdfCitationViewer`. Le PNG existe uniquement pour Claude côté backend.