01 MODÈLE MENTAL

savoir le faire et savoir le faire bien sont 2 choses di"érente

Tout ce que fait Claude repose sur quatre propriétés. Chacune est un continuum, pas un interrupteur. Et la majorité des échecs réels surviennent quand deux propriétés se percutent.

PROPRIÉTÉ

	ZONE DE CAPACITÉ	ZONE DE LIMITATION
Next Token Prediction	résumer, reformater, expliquer des concepts courants	Territoire inconnu, schémas rares, distinguer « vrai » de « sonne vrai »
connaissances	Sujets fréquents et récents dans l'entraînement, sources convergentes	Sujets rares, postérieurs à la coupure, de niche, locaux ou controversés
mémoire	document fournit, exemple cité, session fraîche, contexte fourni	Documents très longs, continuité entre sessions, infos enterrées au milieu
Steerability	Instructions courtes, concrètes, vérifiables	Raisonnements longs, demandes abstraites, précision numérique native

Le diagnostic par collision

Quand un output te déçoit, nomme lesdeux propriétés en jeu. Le duo te donne directement le correctif :

02 PROMPTING

La première ligne fait tout

Restructurer l'ouverture d'unpromptfaitpasserlescore de 2.32 à 3.92 — sans toucher au reste.

Sur un pipeline d'évaluation mesuré, le seul fait de reformuler la première ligne (d'une question passive à une instruction directe) produit un gain de +69 % sur le score de qualité.

Les deux principes

Clarté — langage direct, sans ambiguïté. Commence par ce que Claude doit faire, pas par ce que tu te demandes.

Directivité — formule des instructions, pas des questions. Des verbes d'action : « Écris », « Analyse », « Génère », « Compare ».

Impact mesuré

03 PROMPTING

La spéci#cité double la qualité

Ajouter desdirectives de sortie et des étapes deraisonnement fait passer le scorede3.92 à 7.86— un ×2 surla qualitébrute.

Deux leviers complémentaires transforment un prompt « correct » en prompt « excellent ».

Levier A — Directives de sortie

Contrôle les aspects tangibles : longueur, structure, éléments à inclure, ton. À utiliser dans chaque prompt comme garde-fou de cohérence.

Levier B — Étapes de raisonnement

Fournis les étapes de réflexion quand la tâche demande un raisonnement multi-angles. Le modèle suit tes étapes au lieu d'improviser les siennes.

La combinaison des deux

Les prompts professionnels combinent toujours les deux : directives sur le format ET étapes sur le raisonnement. L'un sans l'autre laisse un trou.

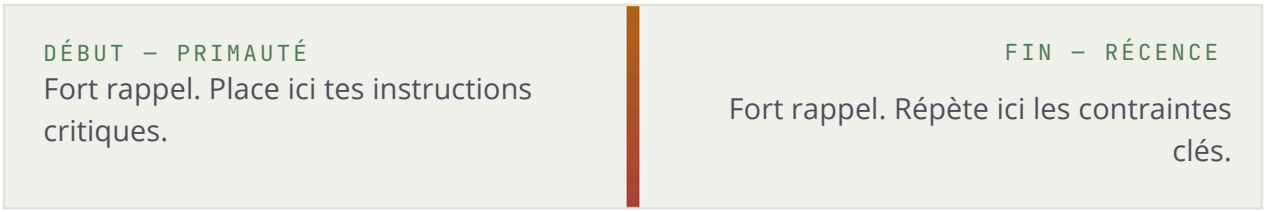
04 ARCHITECTURE

Le context engineering, pas « donne plus de contexte »

Le modèle a uneattention"nie.«Plusde contexte= meilleure réponse » est faux. La vraie compétence : que garder, où le placer, que laisser dehors.

Le concept le plus contre-intuitif de l'Academy. Mesure de Stanford (2023) : quand un fait clé est enterré au milieu d'un long contexte, la précision chute de plus de 30 %. Le début et la fin captent l'attention — le milieu est une zone morte attentionnelle.

L'effet de position sérielle



Trois règles pratiques

- Mets l'essentiel au début ET à la fin. Une contrainte à suivre : dis-la dans le system prompt et redis-la à la fin. Ne compte pas sur le modèle pour peser tout le milieu également.
- Chaque élément ajouté pousse les autres vers le milieu — la zone morte. Ajouter du contexte a un coût indirect : ça dégrade l'attention sur le reste.
- Sélectionne sans pitié. Le context engineering, c'est l'art de savoir ce qu'il faut exclure, pas seulement ce qu'il faut inclure.

Working Memory : la falaise, pas le gradient

Contrairement aux autres propriétés qui se dégradent progressivement, la Working Memory a un seuil brutal. Ça marche jusqu'à ce que ça ne marche plus — troncation silencieuse, sans avertissement. Le modèle ne te dira pas qu'il a oublié quelque chose.

> Outils pratiques : dans Claude Code, `/compact` avec pilotage (`/compact Focus sur l'implémentation du flag #$version`) contrôle ce que le résumé conserve. `/context` affiche taille et répartition du contexte. Les subagents ont leur propre fenêtre — délègue-leur l'exploration sans polluer ton contexte principal.

Les balises XML : structurer pour la machine

Quand un prompt mélange instructions et données, Claude peine à les distinguer. Les balises XML créent une structure explicite :

```
<donnees_clients>
  Fichier CSV des 47 factures de juillet…
#%donnees_clients>

<instructions>
  Analyse les données ci-dessus. Focus : prix et tva.
#%instructions>
```

Utilise des noms descriptifs : `<donnees_clients>` bat un générique `<data>`. Le nom aide Claude à saisir l'intention de chaque section.

```
STRUCTURE D'UN EXEMPLE OPTIMAL
<exemple>
 <input>Analyse du dossier #23 (import CH→FR)#%input>

  <output_ideal>
    Régime : mise en libre pratique + dédouanement à domicile
    Classement : SH 8471.30 (ordinateurs portables), droit 0 %
    TVA import : 20 % acquittée, exonération origine UE justifiée (EUR.1)
    Erreur : valeur en douane sous-déclarée, frais de transport oubliés
  #%output_ideal>
  <pourquoi_cest_bon>
    Nomme le régime douanier exact, cite le code SH et le taux,
    isole l'erreur actionnable — pas un vague « dossier à revoir ».
  #%pourquoi_cest_bon>
#%exemple>
```

05 STEERABILITY

Dis l'objectif, pas seulement le format

Quandune instruction estsuivie àlalettre mais rate l'intention, la répéter plus fort ne comblera jamais l'écart. Reformuler l'objectif, oui.

C'est l'écart lettre vs esprit. Claude suit tes instructions par le même mécanisme que tout le reste : la prédiction du prochain token. Remarquablement pilotable — et il reste toujours un écart entre ce que tu as dit et ce que tu voulais dire.

FORMAT SEUL

« Raccourcis ce texte. »

→ Claude coupe au hasard. Plus court, techniquement. Inutile.

OBJECTIF + FORMAT

« Raccourcis ce texte. Objectif : garder l'attention du lecteur jusqu'à la conclusion en page 2. Coupe tout ce qui ne sert pas cet objectif. »

→ Claude comprend le “pourquoi” et coupe intelligemment.

Les deux échecs caractéristiques

- Dérive de raisonnement — les petites erreurs se composent sur plusieurs étapes. Fix : insérer des points de contrôle (« Montre-moi l'étape 2 avant de continuer »).
- Lettre sans l'esprit — l'instruction est honorée, pas l'intention. Fix : reformuler avec l'objectif, pas répéter l'instruction plus fort.

Exemples avec le « pourquoi »

Le few-shot est l'une des techniques les plus efficaces. Mais le gain maximal vient quand tu expliques pourquoi chaque sortie d'exemple est bonne — pas seulement en montrant entrée et sortie.

06 MÉTHODE

La boucle d'évaluation en 5 étapes

Unscore n'estni bon ni mauvais en soi—ce qui compte,c'estde pouvoir l'améliorer en a#inant ton prompt. Les scores sont des benchmarks de progression, pas des jugements absolus.

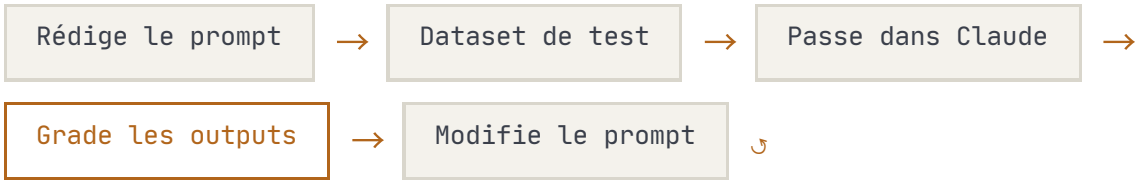
Une méthode systématique que 95 % des utilisateurs n'appliquent pas. Trois approches — seule la troisième fonctionne :

→Test unique — tu testes une fois, ça marche, tu livres. Fort risque de casse en production quand les entrées varient.

→Tests itératifs limités — mieux, mais les utilisateurs fournissent souvent des entrées non anticipées.

→Pipeline d'évaluation — plus de travail en amont, mais la seule approche qui donne une confiance réelle.

Le pipeline



Les deux types de notation

Par le modèle : un autre appel Claude note sur 10, avec forces, faiblesses et raisonnement. Critique : sans demander le raisonnement, le modèle donne des notes médiocres (~6) par défaut.

Par le code : validation programmatique — `json.loads()` pour le JSON, `ast.parse()` pour le Python, des regex pour les formats. Score binaire : 10 si ça passe, 0 sinon.

> Astuce de l'Academy : génère ton dataset de test avec un modèle rapide et pas cher (Haiku) — réserve le modèle puissant aux exécutions réelles. Commence avec 2–3 cas en développement, monte pour la validation finale. Tes meilleures sorties (10/10) deviennent tes exemples pour le prompt suivant.

La boucle délégation-véri!cation
Pour décider *quoi* déléguer : prends des données déjà analysées manuellement (dont tu connais les bons résultats), donne-les à Claude, compare. S'il reproduit tes résultats avec le bon prompt, tu peux déléguer ce type de tâche. Sinon, non.

07 ARCHITECTURE

L'architecture de contexte persistant

PlusClaudeen sait sur toi, tontravail et les conventions de ton équipe, meilleur est chaque livrable — et plus vite il arrive. Le contexte fourni une fois continue de payer.

Une hiérarchie de quatre couches de contexte persistant. Chacune se cumule avec les autres :

COUCHE	PORTÉE	CAS D'USAGE
Global Instructions	Toutes les sessions	Ton rôle, tes acronymes, ton format préféré, ton ton
Projets	Un flux de travail	Un client, un livrable récurrent, un lancement — avec mémoire automatique
Skills	Un type de tâche	Un guide réutilisable chargé automatiquement quand la tâche correspond
CLAUDE.md	Un codebase	Architecture, commandes de dev, conventions de code

Le work"ow Explore → Plan → Code → Commit

La majorité des gens sautent les premières étapes et demandent directement du code, ce qui génère des corrections excessives. Le workflow optimal :

Les deux premières étapes en plan mode (read-only) — le point optimal pour corriger la trajectoire, avant qu'une ligne de code soit écrite.

Trois optimisations en phase Code

- Définis les critères de succès — dis explicitement ce que « fini » veut dire, pour que Claude valide son résultat.
- Ajoute des outils — navigateur (Claude in Chrome), tests, vérification de code. Claude les utilise pour contrôler son travail.
- Inclus une suite de tests — Claude peut les écrire. Avant de lui déléguer la vérification, assure-toi qu'ils sont fiables.

Ré"exion étendue : dernier recours, pas premier ré"exe

Teste tes prompts sans la réflexion étendue d'abord. Active-la seulement « si la précision ne répond pas à tes exigences après avoir déjà optimisé ton prompt ». C'est un dernier recours après l'ingénierie de prompt, pas un réglage par défaut.

Hooks > CLAUDE.md pour les contraintes dures

Point subtil mais critique : CLAUDE.md est de l'*orientation*, pas de l'*application forcée*. Les contraintes dures (« ne jamais pousser sur main », « ne jamais commiter de secrets ») appartiennent aux hooks — du code déterministe exécuté à un point fixe de la boucle agentique.

> CLAUDE.MD — ORIENTATION
> « Ne pousse jamais sur main. »
> → *Demande polie. Peut s'oublier en fin de longue session.*

> HOOK PRETOOLUSE — FORCÉ
> Code qui intercepte git push, vérifie la branche, bloque si c'est main.
> → *Impossible à contourner.*

Le CLAUDE.md qui marche

- Plus le fichier est concis, plus Claude suit chaque ligne. Chaque règle rivalise pour l'attention.
- Spécifique et vérifiable. « Suis les bonnes pratiques » échoue (immesurable). « Mets les routes API dans src/api/handlers, un fichier par route » fonctionne.
- Nomme le remplacement, pas juste l'interdit. « Utilise des exports nommés, pas par défaut » bat « N'utilise pas d'exports par défaut ».

L'emphase est une ressource finie. Si tout est IMPORTANT, rien ne l'est. Réserve les majuscules aux 2–3 règles vraiment critiques.

Légende des termes

NTP	Next Token Prediction — le mécanisme par lequel Claude génère du texte mot par mot, en prédisant ce qui suit le plus probablement.
Context window	La mémoire de travail de Claude — tout ce qu'il « voit » simultanément (prompt, fichiers, historique). Taille fixe, puis troncation silencieuse.
Hallucination	Quand Claude génère une information plausible mais fausse, surtout sur des faits précis (dates, noms, chiffres).
Few-shot	Technique où tu fournis des exemples entrée/sortie dans ton prompt pour montrer le format et le raisonnement attendus.
System prompt	Instructions « système » chargées avant la conversation, qui cadrent le comportement de Claude pour toute la session.
RAG	Retrieval-Augmented Generation — enrichir le prompt avec des documents récupérés dynamiquement pour pallier les limites de connaissance.
Hook	Code déterministe exécuté automatiquement à un point fixe de la boucle agentique (avant/après une action), garantissant un comportement.
CLAUDE.md	Fichier Markdown à la racine d'un projet, lu automatiquement par Claude Code au début de chaque session — mémoire persistante du projet.
Compaction	Résumé automatique du contexte quand la fenêtre approche sa limite, pour libérer de l'espace. Risque de perte d'information.

Subagent	Agent secondaire avec sa propre fenêtre de contexte, utilisé pour déléguer une sous-tâche sans polluer le contexte principal.
Eval	Évaluation systématique — chaîne de test qui mesure la qualité des sorties pour guider l'optimisation du prompt.
Extended Thinking	Réflexion étendue — « brouillon » interne de Claude, raisonnement étape par étape avant la réponse. Coûte plus en tokens et en latence.
Sycophancy	Complaisance — tendance du modèle à valider les affirmations de l'utilisateur plutôt qu'à donner un retour critique honnête.
Prefilling	Préremplissage — technique API où tu préremplis le début de la réponse de Claude pour forcer un format ou une direction.

CLAUDE ACADEMY — SYNTHÈSE INTÉGRALE

Sept. 2026