EVENT CLAUDE & IA

Claude sous le capot

Comprendre la mécanique pour mieux l'exploiter

Fabien • Koality.fr

ACTE 1

Démystifier le vocabulaire

Avant d'aller plus loin, il faut démêler ce qu'on entend par "Claude".

L'interface web, l'app desktop, Claude Code, les modèles Haiku, Sonnet, Opus... Ce n'est pas la même chose.

AU PROGRAMME

- Modèles vs Outils
- Desktop vs Code
- Le point clé

Modèles vs Outils

MODÈLES

Le cerveau — hébergé chez Anthropic

- **Haiku** — rapide, économique
- **Sonnet** — équilibré
- **Opus** — le plus puissant

OUTILS

Le véhicule — interfaces d'accès

- **Claude.ai** — interface web
- **Desktop** — app macOS/Win
- **Claude Code** — CLI dev
- **API** — intégration

Même cerveau, carrosseries différentes

Desktop vs Code : 3 différences

1. Prompt système

Le briefing invisible au démarrage

DESKTOP
"Assistant polyvalent, prudent"

CODE
"Ingénieur senior, autonome"

2. Outils (tools)

Les capacités exposées

DESKTOP
Web search, artifacts, uploads

CODE
Bash, filesystem, écriture

3. Environnement

Où ça s'exécute

DESKTOP
Sandboxé, serveurs Anthropic

CODE
Local, accès à ta machine

POINT CLÉ

Le modèle seul ne fait rien d'utile

Un LLM, c'est une fonction : texte → texte. C'est tout.

La "magie" — lire des fichiers, exécuter du code, chercher sur le web — vient de **l'orchestration** autour du modèle.

ACTE 2

La mécanique agentique

Comment Claude "réfléchit", agit, et boucle sur ses résultats.

Spoiler : c'est beaucoup plus simple qu'on ne croit.

AU PROGRAMME

- Le LLM est stateless
- Client vs Serveur
- Appels d'outils
- Le "thinking"

Le LLM est bête et amnésique

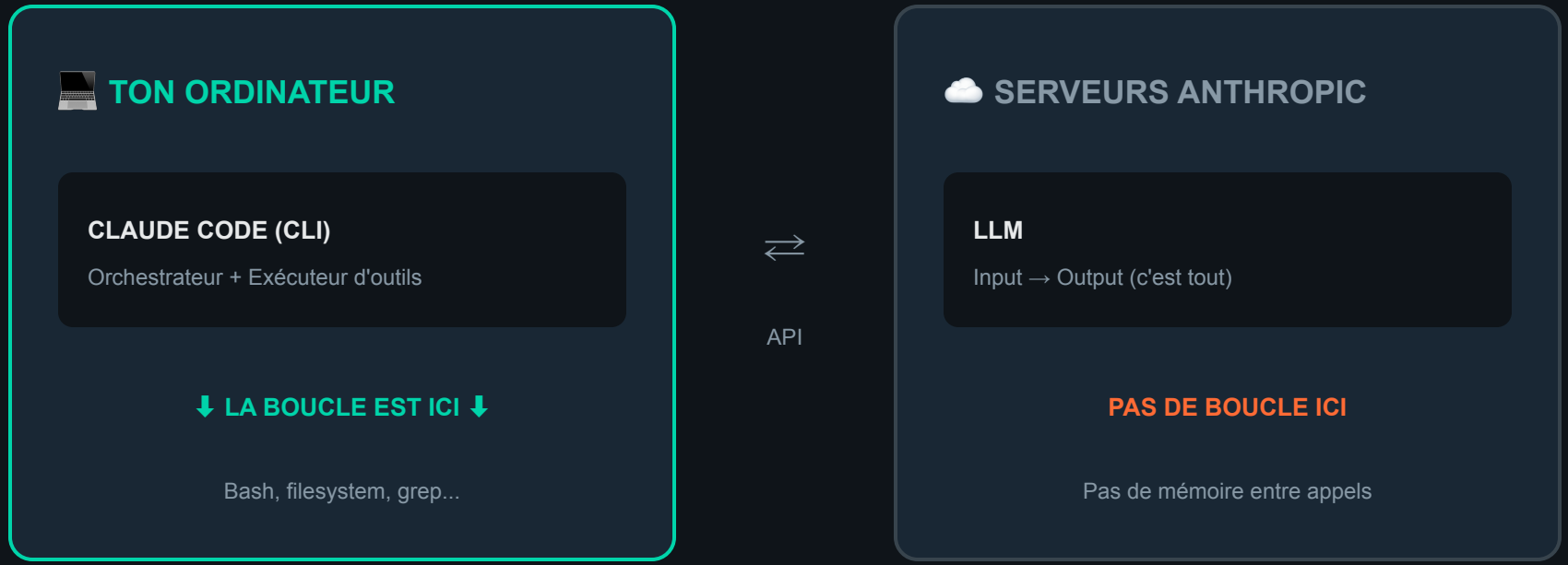
À chaque appel API :

1. Reçoit un blob de texte
2. Génère une réponse
3. **Oublie tout**

CONSÉQUENCES

- **Stateless** — pas de mémoire entre appels
- **Passif** — répond, n'initie rien
- **Inconscient** — ne "sait" pas qu'il est dans une conversation

Où se passe la boucle ?



La boucle en action

"Corrige le bug dans utils.py"

1 TOUR 1

→ Envoi : "Corrige le bug"

← LLM : "Je vais lire..."

tool: read_file("utils.py")

2 TOUR 2

→ Envoi : contexte + fichier

← LLM : "Bug ligne 42..."

tool: write_file("utils.py")

3 TOUR 3

→ Envoi : "fichier modifié"

← LLM : "J'ai corrigé..."

✓ Réponse finale

3 appels API, 3 "réveils" amnésiques. Le contexte renvoyé crée l'illusion de continuité.

Comment le modèle "décide" ?

Il ne "décide" pas vraiment. Il génère du texte, et parfois ce texte a un format spécial.

ENTRÉE

```
"Lis config.py" + tools: [read_file, bash, grep]
```

SORTIE

```
"Je vais lire le fichier" + tool_call: read_file("config.py")
```

CE QUI SE PASSE

Le modèle génère token par token. Les probabilités favorisent le format tool_call car :

- L'entraînement l'a associé à ce contexte
- Le prompt système l'encourage
- "Je ne peux pas" a une proba plus faible

Les 3 types de "réflexion"

1. Implicite

Toujours là, invisible

Dans les poids du modèle, pendant la génération. C'est la boîte noire.

OÙ : Serveurs (forward pass)

2. Extended thinking

Optionnel, bloc visible

UN SEUL appel API. Le modèle verbalise ses étapes avant de répondre.

OÙ : Serveurs (génération)

3. Boucle agentique

Plusieurs appels API

L'orchestrateur (Claude Code) gère la boucle. Le LLM ne sait pas qu'il boucle.

OÙ : Ton ordinateur

ACTE 3

Les Skills

Personnaliser Claude avec des compétences sur-mesure.

Sans fine-tuning, sans complexité. Juste du texte structuré.

AU PROGRAMME

- C'est quoi une skill ?
- Chargement progressif
- Saturation et collisions
- Skills vs GPT Custom

Anatomie d'une Skill

STRUCTURE

```
ma-skill/
├── SKILL.md
├── scripts/
├── references/
└── assets/
```

SKILL.md

```
---
name: email-writer
description: "Rédaction emails. Utiliser quand... NE PAS utiliser pour..."
---
# Email Writer
## Workflow
```

Le trigger, c'est la description. Mal écrite = skill jamais déclenchée.

Chargement progressif

NIVEAU 1
~100 tokens/skill

Métadonnées
Noms et descriptions de toutes les skills — toujours en contexte

TOUJOURS

NIVEAU 2
~2k tokens

SKILL.md complet
Instructions, workflows, exemples — chargé si la demande match la description

SI TRIGGER

NIVEAU 3
Variable

Ressources
Scripts, documentation détaillée, templates — chargé quand Claude en a besoin

À LA DEMANDE

50+ skills : le vrai problème

✓ PAS UN PROBLÈME DE PLACE

70 skills = ~4000 tokens

Sur 200k disponibles = 2%

✗ COLLISION DE TRIGGERS

Descriptions trop proches → hésitation

Domaines qui se chevauchent → erreur

Instructions contradictoires → incohérence

EXEMPLE DE COLLISION

A: "Aide à rédiger des emails"

B: "Communications écrites"

C: "Rédaction business"

User: "Aide-moi à écrire un email"

→ Les 3 matchent. Laquelle ?

Skills vs GPT Custom

GPT CUSTOM

Un assistant = une spécialité

- Changer de domaine = changer de GPT
- Contexte perdu au switch
- Tâches transverses difficiles
- Interface graphique de création

SKILLS

Un assistant = plusieurs compétences

- Changer de domaine = transparent
- Contexte préservé
- Transversalité native
- Fichiers Markdown

Analogie : 10 consultants spécialisés vs 1 consultant senior polyvalent

Ce qu'il faut retenir

MODÈLES VS OUTILS

Même cerveau (Sonnet, Opus), carrosseries différentes (Desktop, Code)

BOUCLE AGENTIQUE

Côté client, pas serveur. Le LLM est stateless et amnésique.

SKILLS

Du texte injecté dans le prompt. Le trigger, c'est la description.

Pas de magie. De l'ingénierie logicielle autour d'un modèle très capable.

DÉMO LIVE

Créer une Skill

On va décortiquer le Skill Creator d'Anthropic, puis créer une skill ensemble.

Les meilleures pratiques, directement de la source.

AU PROGRAMME

- Anatomie du Skill Creator
- Principes clés
- Création en live

Skill Creator : les 4 principes

Extrait du guide officiel d'Anthropic

1. Concis avant tout

"The context window is a public good"

Chaque token compte. Les skills partagent l'espace avec tout le reste.

2. Claude est déjà smart

"Only add context Claude doesn't have"

Pas besoin d'expliquer ce qu'il sait déjà. Focus sur le spécifique.

3. Degrés de liberté

"Match specificity to fragility"

Tâche fragile = instructions précises. Tâche flexible = guidelines.

4. Disclosure progressive

"Three-level loading system"

Métadonnées → SKILL.md → Ressources. Charger le minimum.

La description : make or break

✗ MAUVAIS

```
description: "Aide à créer des documents"
```

Trop vague. Conflit avec d'autres skills. Ne trigger jamais correctement.

✓ BON

```
description: "Création de fichiers .docx Word. Utiliser pour: nouveaux documents, modification de .docx. NE PAS utiliser pour: PDF, markdown."
```

Précis. Cas d'usage explicites. Exclusions claires.

Règle d'or : La description dit CE QUE fait la skill, QUAND l'utiliser, et QUAND NE PAS l'utiliser.

Ressources bundlées : quand les utiliser ?

scripts/

Code exécutable

Quand le même code est réécrit à chaque fois. Déterministe, testé, réutilisable.

rotate_pdf.py, export_csv.py

references/

Documentation

Infos que Claude doit consulter pendant le travail. Chargé à la demande.

schema.md, api_docs.md

assets/

Fichiers pour l'output

Templates, images, fonts. Utilisés dans le résultat final, pas lus en contexte.

template.pptx, logo.png

CRÉATION EN LIVE

Quelle skill on crée ensemble ?

Générateur de devis

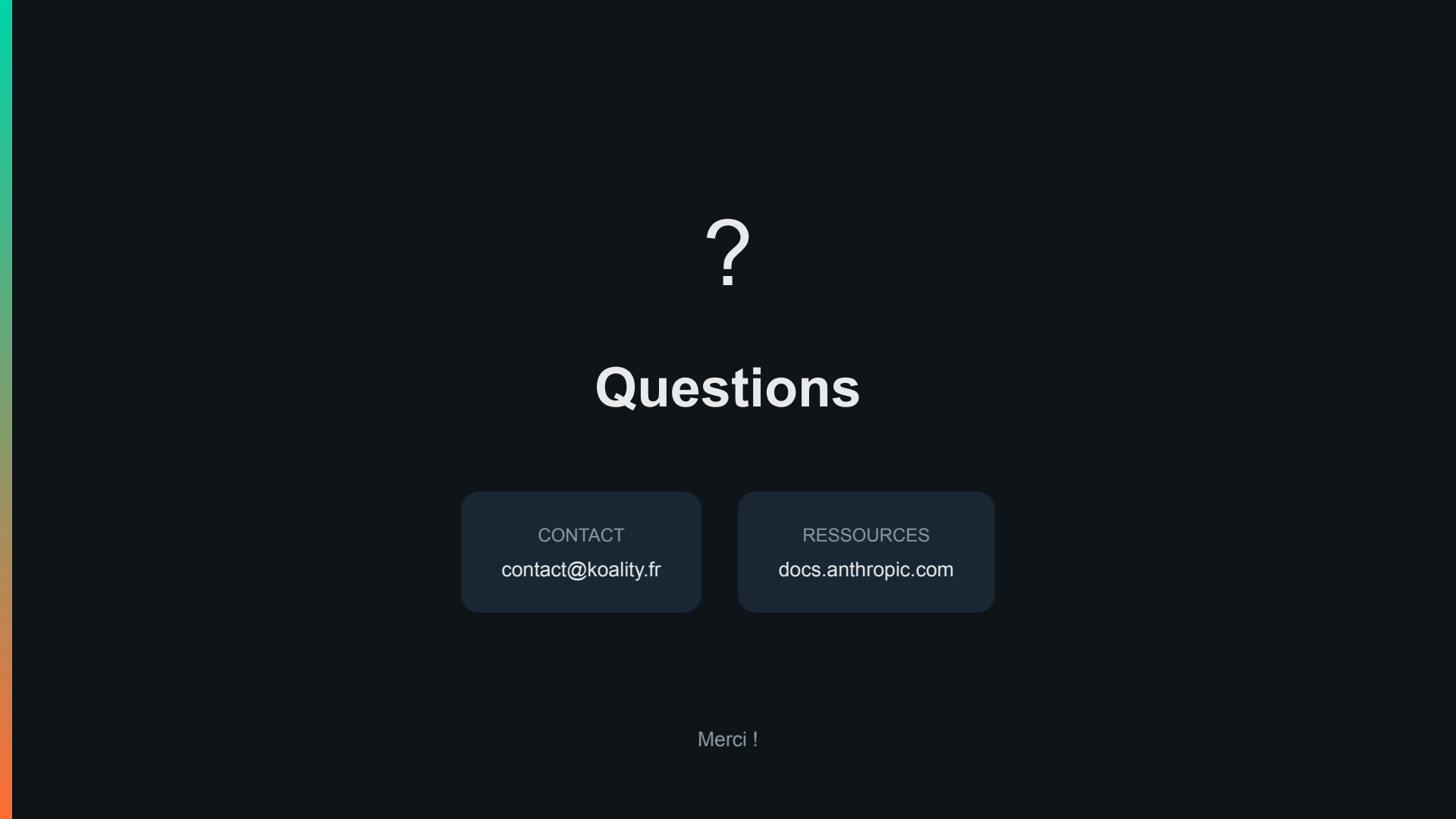
Freelance-friendly

Assistant email

Pro et concis

Autre idée ?

Proposez !


?
Questions
CONTACT
contact@koality.fr
RESSOURCES
docs.anthropic.com
Merci !