

CLAUDE FONDATIONS · COURS

Comprendre l'intelligence artificielle générative

Module 02

6 leçons

LEÇON 01

Fonctionnement d'un LLM

Un grand modèle de langage (LLM) ne « comprend » pas le monde comme nous. Il fait une chose, très bien : prédire le mot suivant. Tout le reste en découle.

Le principe : prédire la suite

Claude, comme tous les LLM, repose sur un mécanisme étonnamment simple à énoncer : à partir d'un texte, il calcule quel est le **prochain fragment de mot le plus probable**, l'ajoute, puis recommence. En répétant cette opération des milliers de fois, il produit des phrases, des paragraphes, du code. Ce qui semble être un raisonnement est en réalité une suite de prédictions extrêmement bien calibrées.

Les tokens : l'unité de base

Le modèle ne travaille pas directement avec des lettres ou des mots, mais avec des **tokens** : des morceaux de texte (un mot court, une syllabe, un signe de ponctuation). Le mot « anticonstitutionnellement » se découpe en plusieurs tokens, alors que « le » en est un seul. Cette notion reviendra souvent : c'est en tokens que se mesurent le contexte et la facturation.

Les paramètres et l'attention

Pour prédire, le modèle s'appuie sur des milliards de **paramètres** : des valeurs numériques ajustées pendant l'entraînement, qui encodent des régularités du langage. Le cœur technique s'appelle le *Transformer*, dont le mécanisme d'**attention** permet au modèle de pondérer l'importance de chaque token du texte les uns par rapport aux autres. C'est ce qui lui permet de tenir compte du contexte lointain, pas seulement du mot juste avant.

Une conséquence importante : le probabilisme

Parce que tout repose sur des probabilités, un LLM est **non déterministe** : la même question peut donner des formulations différentes. Il ne consulte pas une base de données de vérités ; il génère la réponse la plus *plausible*. C'est la clé pour comprendre à la fois sa puissance (fluidité, créativité, généralisation) et ses faiblesses (les « hallucinations », que nous verrons plus loin).

À RETENIR

Un LLM prédit le token suivant à partir du contexte, grâce à des milliards de paramètres et au mécanisme d'attention. Il produit du plausible, pas du vérifié : gardez-le en tête pour tout usage sérieux.

Les phases d'entraînement d'un LLM

Un modèle comme Claude ne naît pas compétent. Il passe par plusieurs phases d'entraînement successives, chacune avec un rôle précis.

1. Le pré-entraînement

La première phase, le **pré-entraînement**, consiste à exposer le modèle à d'immenses quantités de texte pour lui apprendre à prédire le token suivant. À l'issue de cette étape, le modèle possède une vaste culture générale et une grande maîtrise du langage, mais il ne sait pas encore se comporter en assistant : il complète du texte plutôt que de dialoguer utilement.

2. Le fine-tuning (ajustement)

Vient ensuite l'**ajustement supervisé** : on montre au modèle de nombreux exemples de bonnes réponses à des instructions, pour qu'il apprenne à *suivre des consignes* et à adopter le format d'un assistant conversationnel. C'est là qu'il apprend à répondre à une question plutôt qu'à la prolonger.

3. L'apprentissage par retour (RLHF et variantes)

La troisième phase affine le **comportement** : le modèle est entraîné à privilégier les réponses jugées les plus utiles, honnêtes et sûres. Historiquement, cela s'appuie sur le retour humain (*RLHF*) — des personnes comparent des réponses — mais Anthropic y ajoute une approche par principes (la **Constitutional AI**, détaillée dans une leçon dédiée), où le modèle s'auto-évalue au regard de règles explicites.

Une conséquence : la date de coupure des connaissances

Le savoir « intégré » d'un modèle s'arrête à la période couverte par ses données d'entraînement : c'est la **date de coupure des connaissances**. Au-delà, le modèle ne connaît pas les événements récents — d'où l'importance d'outils comme la recherche web pour les informations d'actualité.

À RETENIR

Pré-entraînement (savoir + langage), ajustement (suivre des consignes), apprentissage par retour (utile/honnête/sûr) : trois étapes qui transforment un prédicteur de texte en assistant fiable. Son savoir reste figé à une date de coupure.

LEÇON 03

Les LRM et l'effort de raisonnement

Certains modèles ne se contentent pas de répondre : ils prennent le temps de « réfléchir » avant. C'est l'idée des modèles de raisonnement et de l'effort.

Du LLM au LRM

On parle parfois de **LRM** (*Large Reasoning Model*) pour désigner les modèles capables de produire une phase de **raisonnement interne** avant de donner leur réponse finale. Plutôt que de répondre du tac au tac, le modèle déroule des étapes intermédiaires — un brouillon de réflexion — qui améliore nettement les résultats sur les tâches complexes : mathématiques, logique, planification, code difficile.

La notion d'effort

Ce raisonnement a un coût : il consomme des tokens et du temps. D'où la notion d'**effort** (ou « budget de réflexion ») : on peut, selon les cas, demander au modèle de réfléchir davantage pour une tâche ardue, ou au contraire de répondre vite pour une question simple. Plus d'effort = souvent plus de justesse, mais plus de latence et de coût. C'est un curseur à ajuster selon l'enjeu.

Quand est-ce utile ?

Le raisonnement approfondi brille sur les problèmes à *plusieurs étapes* ou qui demandent de la rigueur : vérifier un calcul, décomposer un cahier des charges, comparer des options, déboguer. Pour une reformulation, un résumé court ou une question factuelle simple, il n'apporte pas grand-chose et n'est pas nécessaire.

À RETENIR

Les modèles de raisonnement « pensent » avant de répondre, ce qui améliore les tâches complexes. L'effort est un curseur : plus de réflexion pour les problèmes durs, moins pour les demandes simples.

Composition et gestion du contexte

Le contexte, c'est la mémoire de travail du modèle le temps d'une conversation. Bien le gérer change tout dans la qualité des réponses.

Qu'est-ce que le contexte ?

À chaque réponse, Claude « voit » un ensemble de texte : votre message, l'historique de la conversation, les instructions, les documents joints, les résultats d'outils. C'est la **fenêtre de contexte**, mesurée en tokens. Tout ce qui doit influencer la réponse doit s'y trouver : le modèle n'a aucune mémoire en dehors de ce qui lui est présenté à cet instant.

Une fenêtre large, mais pas infinie

Les modèles récents disposent de fenêtres très grandes (permettant d'ingérer de longs documents), mais elle reste **limitée**. Au-delà, les éléments les plus anciens sortent du champ. De plus, une fenêtre pleine coûte plus cher et peut « diluer » l'attention : noyer une consigne importante au milieu de milliers de tokens la rend moins efficace.

Bien composer son contexte

La bonne pratique est de fournir **ce qui est utile, et rien de superflu** : les documents pertinents, une consigne claire, des exemples ciblés. Pour les longues sessions, il est souvent préférable de résumer, de repartir sur une nouvelle conversation, ou d'utiliser les *projets* (vus plus loin) pour ancrer un contexte durable sans tout recoller à chaque fois.

Contexte ≠ mémoire permanente

Par défaut, une nouvelle conversation repart d'une page blanche. Les mécanismes de mémoire, de projets ou d'instructions servent justement à réintroduire du contexte de façon structurée d'une fois sur l'autre.

À RETENIR

Le contexte est la mémoire de travail momentanée du modèle, mesurée en tokens et limitée. Fournissez l'essentiel, évitez le bruit, et structurez le contexte durable via les instructions et les projets.

La spécificité Anthropic : la Constitution IA

C'est la signature d'Anthropic : entraîner le modèle à respecter des principes explicites, plutôt que de dépendre uniquement de jugements humains au cas par cas.

Le problème que ça résout

Aligner un modèle uniquement avec du retour humain pose des limites : c'est coûteux, peu transparent, et difficile à rendre cohérent à grande échelle. Comment garantir que le modèle refuse le nuisible tout en restant utile et honnête, de façon stable et explicable ?

Le principe de la Constitution IA

La **Constitutional AI** repose sur une idée forte : donner au modèle une « **constitution** » — un ensemble de principes écrits (inspirés de droits fondamentaux, de règles d'éthique et de sécurité). Pendant l'entraînement, le modèle apprend à **critiquer et corriger ses propres réponses** à la lumière de ces principes. Une partie du travail d'alignement se fait ainsi via l'IA elle-même, guidée par des règles explicites, plutôt que par des annotateurs humains sur chaque exemple.

Ce que ça change concrètement

Cette approche vise un équilibre entre trois qualités souvent en tension : être **utile**, **honnête** et **inoffensif**. Elle explique certains comportements de Claude que vous observerez : refus argumentés de demandes dangereuses, prudence sur les sujets sensibles, transparence sur ses incertitudes. Les principes étant explicites, le comportement est plus *cohérent* et plus *auditable*.

À RETENIR

La Constitution IA entraîne Claude à s'aligner sur des principes écrits qu'il applique à ses propres réponses. Résultat : un modèle qui cherche à être utile, honnête et sûr de manière plus cohérente et transparente.

LEÇON 06

L'économie des tokens et la facturation

Comprendre les tokens, c'est comprendre à la fois les limites techniques et le coût réel de l'IA générative.

Le token, unité de mesure et de coût

Tout ce que traite un modèle se compte en **tokens** : le texte que vous envoyez (tokens d'*entrée*) et le texte qu'il génère (tokens de *sortie*). En règle générale, un token correspond à peu près à trois quarts d'un mot en français ; une page de texte représente donc quelques centaines de tokens. C'est cette unité qui sert à mesurer le contexte et à facturer l'usage via l'API.

Entrée, sortie et raisonnement

La facturation distingue généralement les tokens d'entrée et de sortie (souvent plus chers), et, pour les modèles de raisonnement, les tokens de « réflexion » consommés avant la réponse. Plus la conversation est longue, plus l'entrée grossit à chaque tour, puisque l'historique est renvoyé au modèle : c'est pourquoi une très longue session finit par coûter et ralentir davantage.

Deux logiques de facturation

Côté **application** (Claude.ai, Cework), on paie le plus souvent un **abonnement** avec des quotas d'usage — pas de calcul de tokens à faire soi-même. Côté **API**, on paie à **l'usage**, au prorata des tokens consommés, avec des tarifs qui varient selon la gamme de modèle (Haiku < Sonnet < Opus). Choisir le bon modèle pour la bonne tâche est donc aussi une décision économique.

Optimiser sa consommation

Quelques réflexes utiles : fournir un contexte concis, éviter de recoller inutilement de longs documents, utiliser un modèle plus léger quand la tâche est simple, et réserver l'effort de raisonnement aux problèmes qui le méritent. Des mécanismes comme la mise en cache du contexte (côté API) permettent aussi de réduire les coûts sur des contenus répétés.

À RETENIR

Le token est l'unité qui mesure le contexte et le coût. On paie l'entrée, la sortie et (le cas échéant) le raisonnement. Abonnement à quotas côté application, paiement à l'usage côté API : dans les deux cas, le bon modèle et un contexte sobre font la différence.