Service OCI Generative AI

Guide Complet des Capacités et Fonctionnalités

Formation approfondie sur les modèles de langage,
l'ingénierie des prompts, le fine-tuning et l'architecture sécurisée

Oracle Cloud Infrastructure

8 août 2025

Table des matières

Introduction du Module

Bienvenue dans ce module complet sur le service OCI Generative AI. Dans ce module, nous explorerons les capacités fondamentales du service en examinant d'abord les modèles foundationnels pré-entraînés pour la génération, la synthèse et les embeddings.

Nous aborderons ensuite l'ingénierie des prompts et les méthodes permettant de personnaliser les modèles de langage avec vos propres données. Nous étudierons également le tuning et l'inférence, examinerons la configuration du fine-tuning, les clusters AI dédiés, et conclurons par l'architecture sécurisée de l'IA générative.

Objectifs du module :

- Comprendre les modèles foundationnels disponibles
- Maîtriser les techniques d'ingénierie des prompts
- Apprendre à personnaliser les LLM avec vos données
- Configurer le fine-tuning et l'inférence
- Implémenter une architecture sécurisée

Chapitre 1

Modèles de Chat

1.1 Comprendre les Tokens

Avant d'approfondir les modèles de chat, examinons d'abord les tokens. Les grands modèles de langage comprennent les tokens plutôt que les caractères individuels.

Définition des Tokens

Un token peut être :

- Une partie d'un mot
- Un mot entier
- Un symbole de ponctuation

1.1.1 Exemples de Tokenisation

- Un mot courant comme `"pomme"` constitue un seul token
- Un mot comme `"amitié"` peut être décomposé en deux tokens : `"ami"` et `"tié"`
- Le nombre de tokens par mot dépend de la complexité du texte

Règles d'estimation :

- Texte simple : environ 1 token par mot en moyenne
- Texte complexe (mots rares) : 2 à 3 tokens par mot en moyenne

1.1.2 Exemple Pratique de Tokenisation

Considérons la phrase : "Beaucoup de mots correspondent à un token, mais certains non, indivisible"

Après passage dans un tokenizer, cette phrase serait décomposée en 15 tokens, alors qu'elle ne contient que 10 mots. Certains tokens correspondent à des symboles de ponctuation (virgule, apostrophe, point), et des mots moins fréquents comme "indivisible" peuvent être divisés en plusieurs tokens.

1.2 Modèles Foundationnels Disponibles

Le service OCI Generative AI propose plusieurs modèles de chat foundationnels pré-entraînés :

1.2.1 Modèle Command-R-Plus

— **Caractéristiques** : Modèle conversationnel hautement performant pour le suivi d'instructions

— **Prompt utilisateur** : Jusqu'à 128 000 tokens

— **Réponse** : Jusqu'à 4 000 tokens par exécution

— **Cas d'usage** : Q&A, recherche d'information, analyse de sentiment, chat

1.2.2 Modèle Command-R-16k

— **Caractéristiques** : Version plus petite et plus rapide du Command-R-Plus

— **Prompt utilisateur** : Jusqu'à 16 000 tokens

— **Réponse** : Jusqu'à 4 000 tokens par exécution

— **Utilisation recommandée** : Lorsque la vitesse et le coût sont prioritaires

1.2.3 Famille Llama 3.1 (Meta)

Le service propose deux tailles de modèles Llama :

— **Modèle 405 milliards de paramètres** : Le plus grand modèle publiquement disponible, adapté aux applications d'entreprise complexes

— **Modèle 70 milliards de paramètres** : Version plus compacte mais très capable

— **Capacité** : Jusqu'à 128 000 tokens pour le prompt et la réponse

1.3 Paramètres de Personnalisation

1.3.1 Tokens de Sortie Maximum

Ce paramètre définit le nombre maximum de tokens que le modèle génère par réponse. Il permet de contrôler la longueur des réponses produites.

1.3.2 Préambule Override

Exemple de Préambule

Le préambule par défaut pour les modèles Cohere Command peut être remplacé pour modifier le comportement du modèle. Par exemple, un préambule "Réponds avec un ton de pirate" changera complètement le style de réponse.

1.3.3 Température

La température contrôle le caractère aléatoire de la sortie du modèle :

— **Température = 0** : Sortie déterministe (toujours le token le plus probable)

— **Température = 1** : Distribution aplatie sur tous les mots (plus de créativité)

— **Température élevée** : Le modèle utilise des mots avec une probabilité plus faible

Exemple avec "Le ciel est..."

— Avec température = 0 : "Le ciel est **bleu**" (toujours)

— Avec température = 1 : "Le ciel est **la limite**" (plus créatif)

1.4 Paramètres Avancés

1.4.1 Top-k

Le paramètre top-k indique au modèle de choisir le prochain token parmi les k tokens les plus probables.

Exemple Top-k

Pour la phrase "Le nom de ce pays est le..." avec top-k = 3, le modèle ne considérera que les 3 options les plus probables : "France", "Canada", "Brésil".

1.4.2 Top-p (Nucleus Sampling)

Similar au top-k, mais basé sur la somme cumulée des probabilités plutôt que sur un nombre fixe de tokens.

1.4.3 Pénalités de Fréquence et de Présence

Ces paramètres sont utiles pour éviter les répétitions :

— **Pénalité de fréquence** : Pénalise les tokens selon leur fréquence d'apparition

— **Pénalité de présence** : Pénalise les tokens dès leur première apparition

Chapitre 2

Modèles d'Embedding

2.1 Introduction aux Embeddings

Les embeddings sont des représentations numériques d'un morceau de texte converti en séquences de nombres. Un texte peut être un mot, une phrase, un paragraphe ou un document entier.

> **Avantage des embeddings :** Ils permettent aux ordinateurs de comprendre facilement les relations entre différents morceaux de texte.

2.2 Fonctionnement des Embeddings

2.2.1 Architecture Encodeur-Décodeur

L'utilisation pratique des embeddings se manifeste dans des tâches comme la traduction, qui est une tâche séquence-à-séquence :

1. L'**encodeur** prend le texte d'entrée et le convertit en vecteurs
2. Le **décodeur** utilise ces vecteurs pour générer des tokens dans une boucle auto-référentielle
3. Exemple : Traduction de l'anglais vers l'hébreu

2.2.2 Propriétés des Word Embeddings

Les embeddings de mots capturent les propriétés des mots. Considérons ces mots mappés selon l'âge et la taille :

— "chaton", "chat", "chiot", "chien", "lion", "éléphant"

— L'éléphant est plus grand qu'un lion, qui est plus grand qu'un chien, etc.

— Les embeddings réels représentent plus de propriétés que juste deux coordonnées

2.3 Similarité Numérique

2.3.1 Mesures de Similarité

Deux techniques principales sont utilisées :

— **Similarité cosinus**

— **Similarité par produit scalaire**

Principe important : Les embeddings numériquement similaires sont aussi sémantiquement similaires (proche en sens ou en relation).

2.3.2 Formation de Clusters

Les mots se regroupent naturellement en clusters sémantiques :

— **Cluster animals** : chiot, chien, chat, lion

— **Cluster fruits** : pomme, fraise, orange

— **Cluster villes** : Paris, Londres, New York

2.4 Embeddings de Phrases

Un embedding de phrase associe chaque phrase à un vecteur de nombres, similaire aux embeddings de mots.

Exemple d'Embeddings de Phrases

L'embedding de "les compagnons canins disent" sera plus similaire à l'embedding de "wouf" qu'à celui de "miaou". "Miaou" sera plus proche de "les amis félins disent".

2.5 Cas d'Usage : Retrieval-Augmented Generation (RAG)

2.5.1 Défi des Modèles Actuels

L'un des principaux défis des modèles génératifs actuels est leur incapacité à se connecter aux données d'entreprise. Le RAG offre une approche prometteuse pour surmonter cette limitation.

2.5.2 Architecture RAG

1. Prendre un large corpus de documents
2. Le diviser en chunks ou paragraphes
3. Générer des embeddings pour chaque paragraphe
4. Stocker tous les embeddings dans une base de données vectorielle

2.5.3 Processus RAG

1. L'utilisateur pose une question non répondue par le LLM
2. La question est encodée comme un vecteur et envoyée à la base vectorielle
3. La base effectue une recherche de correspondance la plus proche
4. Les documents/paragraphes pertinents sont identifiés
5. Ces contenus sont insérés dans le prompt envoyé au LLM

6. Le LLM utilise ce contenu plus ses connaissances générales pour fournir une réponse informée

2.6 Modèles d'Embedding OCI

2.6.1 Modèles Cohere Disponibles

— **Cohere.embed-English** : Convertit le texte anglais en embeddings vectoriels

— **english-lite** : Version plus petite et plus rapide d'embed-english

— **Cohere.embed-multilingual** : Modèle multilingue pour plus de 100 langues

2.6.2 Capacités Multilingues

Le modèle multilingue permet :

— Recherche dans la même langue (requête française sur documents français)

— Recherche inter-langues (requête française sur documents anglais)

2.6.3 Spécifications Techniques

Modèle	Dimensions	Tokens Max
English/Multilingual v3	1,024	512
English/Multilingual lite v3	384	512
Modèles génération précédente	1,024	512

TABLE 2.1 – Spécifications des modèles d'embedding

2.6.4 Améliorations du Modèle v3

Nouvelles capacités du modèle v3 :

— Évaluation de la correspondance entre requête et sujet du document

— Évaluation de la qualité globale du contenu

— Classement des documents de meilleure qualité en tête

— Amélioration significative des systèmes RAG

2.6.5 Limitations d'Usage

— Maximum 96 entrées par exécution

— Chaque entrée doit faire moins de 512 tokens

— Possibilité d'entrée par phrase, phrase ou fichier

Chapitre 3

Ingénierie des Prompts

3.1 Définitions Fondamentales

3.1.1 Qu'est-ce qu'un Prompt ?

Un prompt est l'entrée ou le texte initial fourni au modèle de langage large (LLM). Il sert de point de départ pour générer une réponse.

3.1.2 Qu'est-ce que l'Ingénierie des Prompts ?

L'ingénierie des prompts est le processus d'affinage itératif d'un prompt dans le but de susciter un style ou un type de réponse particulier du modèle de langage large.

3.2 Fonctionnement des LLM

Les LLM sont essentiellement des prédicteurs de mots suivants. Ils tentent de produire la série de mots la plus susceptible de suivre le texte précédent.

Exemple Historique

Si vous fournissez le prompt "Il y a quatre-vingts et sept ans, nos", le LLM complétera probablement avec "pères ont fait naître sur ce continent une nouvelle nation..." (début du discours de Gettysburg de Lincoln, 1863).

3.3 Évolution vers les Modèles d'Instructions

3.3.1 Problème des Modèles de Complétion

Les LLM de complétion sont entraînés pour prédire le mot suivant sur un large ensemble de données du web plutôt que pour exécuter les tâches linguistiques souhaitées par l'utilisateur.

3.3.2 Solution : Fine-tuning avec RLHF

Le papier sur Llama 2 (2023) a démontré comment l'apprentissage par renforcement à partir de feedback humain (RLHF) permet d'affiner les LLM pour suivre une large classe d'instructions écrites.

Processus RLHF :

1. Les annotateurs humains écrivent des prompts et comparent les sorties du modèle
2. Le feedback humain est utilisé pour entraîner un modèle de récompense
3. Ce modèle apprend les patterns des préférences humaines
4. Il peut ensuite automatiser les décisions de préférence

3.3.3 Données d'Entraînement Llama 2

— **Modèle de base** : Entraîné sur 2 trillions de tokens

— **Llama 2 Chat** : Affiné sur 28 000 paires prompt-réponse

— **Alignement IA** : RLHF avec >1,4 million d'exemples Meta + 7 ensembles de données

3.4 Apprentissage en Contexte et Few-Shot Prompting

3.4.1 Apprentissage en Contexte

L'apprentissage en contexte n'est pas un apprentissage au sens traditionnel car aucun paramètre du modèle ne change. Il s'agit de conditionner un LLM avec des instructions et des démonstrations de la tâche à accomplir.

3.4.2 K-Shot Prompting

Le k-shot prompting fournit explicitement k exemples de la tâche souhaitée dans le prompt.

Exemple de 3-Shot Prompting pour la Traduction

Description de tâche : Traduire de l'anglais vers le français
Exemples :

— sea → mer

— friend → ami

— house → maison

Prompt : cheese → ?

3.4.3 Avantages du Few-Shot

Le few-shot prompting est largement considéré comme améliorant les résultats par rapport au zero-shot prompting (sans exemples).

3.5 Format des Prompts

Attention au format : Les modèles de langage comme Llama 2 sont entraînés sur un format de prompt spécifique. Un format incorrect peut produire des résultats sous-optimaux.

3.5.1 Format Llama 2

Llama 2 utilise des balises spécifiques :

```
<s>[INST] <<SYS>>
{votre_prompt_syst me}
<</SYS>>

{votre_message_utilisateur} [/INST]
```

Pour les dialogues à échanges multiples :

— `<s>` : Début de séquence

— `[INST]` : Début d'instruction

— `[/INST]` : Fin d'instruction

— «SYS» : Balises système pour le contexte personnalisé

3.6 Stratégies de Prompting Avancées

3.6.1 Chain-of-Thought Prompting

Cette stratégie fournit des exemples dans un prompt pour montrer des réponses incluant une étape de raisonnement.

Exemple Chain-of-Thought

Question : Dans un magasin, il y a 15 pommes. Si j'en achète 6 et que mon ami en achète 4, combien reste-t-il de pommes ?
Réponse avec raisonnement :

1. Pommes initiales : 15
2. J'achète : 6 pommes
3. Mon ami achète : 4 pommes
4. Total acheté : 6 + 4 = 10 pommes
5. Pommes restantes : 15 - 10 = 5 pommes

3.6.2 Zero-Shot Chain-of-Thought

Similar au chain-of-thought, mais sans exemple. On utilise simplement la phrase "Réfléchissons étape par étape" pour encourager le raisonnement structuré.

Avantages du Chain-of-Thought :

— Permet d'accomplir des tâches complexes nécessitant un raisonnement intermédiaire

— Améliore la transparence du processus de décision

— Réduit les erreurs sur les problèmes multi-étapes

Chapitre 4

Personnaliser les LLM avec vos Données

4.1 Pourquoi ne pas Entraîner de Zéro ?

Entraîner un LLM from scratch n'est pas recommandé pour trois raisons principales :

4.1.1 Coût Prohibitif

— Environ 1 million de dollars pour entraîner un modèle de 10 milliards de paramètres
— Le Llama-2 de Meta a été entraîné sur 2 trillions de tokens
— Équivalent à environ un milliard de mémoires juridiques

4.1.2 Besoin Énorme de Données Annotées

— Nécessité de données étiquetées, catégorisées et balisées
— Processus très intensif en main-d'œuvre
— Coût élevé d'acquisition des données de qualité

4.1.3 Expertise Technique Requise

— Compréhension approfondie des performances du modèle
— Capacité à surveiller et détecter les pannes matérielles
— Connaissance des limitations du modèle
— Mitigation des défaillances système

4.2 Trois Options de Personnalisation

4.2.1 Option 1 : Apprentissage en Contexte (Few-Shot)

— **Principe** : Fournir des démonstrations dans le prompt
— **Avantages** : Simple, sans coût d'entraînement
— **Inconvénients** : Ajoute de la latence, limité par la fenêtre de contexte
— **Usage** : Quand le LLM comprend déjà le sujet nécessaire

4.2.2 Option 2 : Fine-Tuning

— **Principe** : Optimiser un modèle sur un ensemble de données spécifique au domaine

— **Avantages** : Amélioration des performances, efficacité accrue, pas d'impact sur la latence

— **Inconvénients** : Complexe, nécessite des données étiquetées, intensif en travail

— **Usage** : Quand le modèle pré-entraîné ne performe pas bien sur votre tâche

4.2.3 Option 3 : Retrieval-Augmented Generation (RAG)

— **Principe** : Connecter le LLM à une base de connaissances d'entreprise

— **Avantages** : Accès aux données les plus récentes, réponses fondées, pas de fine-tuning requis

— **Inconvénients** : Configuration complexe, nécessite une source de données compatible

— **Usage** : Quand les données changent rapidement ou pour mitiger les hallucinations

4.3 Cadre de Décision

4.3.1 Matrice d'Optimisation

Le choix entre les techniques peut être visualisé sur deux dimensions :

— **Axe horizontal** : Optimisation du contexte (ce que le modèle doit savoir)

— **Axe vertical** : Optimisation du LLM (comment le modèle doit agir)

4.3.2 Parcours Typique

1. **Commencer par le prompt engineering** : Facile à tester, évaluation rapide
2. **Problème de contexte ?** → Utiliser RAG
3. **Problème d'optimisation LLM ?** → Utiliser le fine-tuning
4. **Souvent** : Combinaison de plusieurs approches

Parcours d'optimisation suggéré :

1. Prompt simple + framework d'évaluation
2. Ajout d'exemples few-shot
3. Implémentation d'un système RAG simple
4. Fine-tuning pour le style/format souhaité
5. Optimisation itérative du système de récupération

Cette approche itérative permet d'utiliser toutes les techniques selon les besoins spécifiques de votre cas d'usage.

Chapitre 5

Fine-Tuning et Inférence dans OCI Generative AI

5.1 Définitions Fondamentales

5.1.1 Fine-Tuning

Le fine-tuning consiste à prendre un modèle foundationnel pré-entraîné et à fournir un entraînement supplémentaire en utilisant des données personnalisées.

5.1.2 Inférence

Dans le contexte des modèles de langage, l'inférence fait référence au processus par lequel le modèle reçoit un nouveau texte en entrée et génère un texte de sortie basé sur ce qu'il a appris pendant l'entraînement et le fine-tuning.

5.1.3 Modèle Personnalisé

Un modèle personnalisé est un modèle créé en utilisant un modèle pré-entraîné comme base et en utilisant votre propre ensemble de données pour affiner ce modèle.

5.2 Workflow de Fine-Tuning

Le processus de fine-tuning dans OCI Generative AI suit ces étapes :

1. **Créer un cluster AI dédié** (type : cluster de fine-tuning)
2. **Rassembler les données d'entraînement**
3. **Lancer le processus de fine-tuning**
4. **Obtenir le modèle personnalisé** fine-tuné

> Ces étapes peuvent être interchangées - vous pouvez commencer par rassembler les données avant de créer le cluster.

5.3 Workflow d'Inférence

5.3.1 Point de Terminaison de Modèle

Un point de terminaison de modèle est un point désigné sur le cluster AI dédié où le modèle de langage peut accepter les requêtes utilisateur et renvoyer des réponses.

5.3.2 Processus d'Inférence

1. **Créer un cluster AI dédié** (type : cluster d'hébergement)
2. **Créer le point de terminaison**
3. **Servir le trafic** de production

5.4 Clusters AI Dédiés

5.4.1 Principe Fondamental

Les clusters AI dédiés offrent un déploiement GPU mono-tenant où les GPU ne hébergent que vos modèles personnalisés.

5.4.2 Avantages

— **Performance cohérente** : Le point de terminaison n'est pas partagé avec d'autres clients

— **Dimensionnement prévisible** : Facilité de détermination de la taille de cluster nécessaire

— **Isolation complète** : Sécurité et confidentialité renforcées

5.4.3 Types de Clusters

— **Cluster de fine-tuning** : Utilisé pour l'entraînement d'un modèle foundationnel pré-entraîné

— **Cluster d'hébergement** : Héberge les points de terminaison de modèles personnalisés pour l'inférence

5.5 Fine-Tuning T-Few

5.5.1 Comparaison avec le Fine-Tuning Vanilla

Aspect	**Fine-Tuning Vanilla**	**Fine-Tuning T-Few**
Poids mis à jour	Toutes ou la plupart des couches	Seulement une fraction (0.01%)
Temps d'entraînement	Long	Considérablement réduit
Coût de service	Élevé	Réduit
Approche	Modification globale	Ajout de couches additionnelles

Table 5.1 – Comparaison des méthodes de fine-tuning

5.5.2 Technique T-Few

T-Few est une technique de fine-tuning efficace en paramètres qui :

— Insère des couches supplémentaires représentant 0.01% de la taille du modèle de base

— Localise les mises à jour de poids uniquement aux couches T-Few

— Réduit significativement le temps d'entraînement et les coûts

5.6 Processus T-Few en Détail

5.6.1 Composants de Base

1. **Poids initiaux** du modèle de base
2. **Ensemble de données d'entraînement annoté** (paires entrée/sortie)
3. **Génération de poids supplémentaires** (0.01% de la taille de base)
4. **Propagation ciblée** vers les couches de transformateur T-Few

5.6.2 Avantages du Processus

— Conservation des connaissances du modèle pré-entraîné

— Adaptation aux caractéristiques de la nouvelle tâche/domaine

— Efficacité computationnelle et économique

5.7 Optimisation des Coûts d'Inférence

5.7.1 Multi-Tenancy sur GPU

Un cluster d'hébergement peut héberger :

— **Un point de terminaison de modèle de base**

— **Jusqu'à n points de terminaison de modèles personnalisés**

— **Service concurrent** de toutes les requêtes

Cette approche réduit les coûts d'inférence en partageant les ressources GPU.

5.7.2 Gestion de la Mémoire GPU

Défi traditionnel : La mémoire GPU est limitée, et changer de modèle peut entraîner une surcharge significative due au rechargement complet de la mémoire GPU.

5.7.3 Solution OCI

Dans OCI Generative AI :

— Les modèles partagent la majorité des poids (via T-Few)

— Variations minimes entre les modèles personnalisés

— Déploiement efficace sur les mêmes GPU

— Partage de paramètres pour réduire l'utilisation mémoire

Architecture de Partage

- **Modèle de base** : Chargé une fois en mémoire
- **Modèles A, B, C** : Différences de poids minimales (0.01%)
- **Surcharge minimale** : Commutation rapide entre modèles
- **Partage de paramètres** : Parties communes chargées une seule fois

5.8 Gestion des Requêtes Multiples

Le système peut traiter efficacement :

- Plusieurs requêtes d'inférence simultanées
- Commutation transparente entre modèles
- Réduction significative de l'utilisation mémoire totale
- Amélioration du débit global

Chapitre 6

Dimensionnement et Tarification des Clusters AI Dédiés

6.1 Types d'Unités de Cluster

Le service OCI Generative AI propose quatre types d'unités de cluster :

6.1.1 Large Cohere Dedicated

— **Capacités** : Fine-tuning et hébergement

— **Famille** : Modèles Cohere Command R uniquement

— **SKU** : `dedicated-unit-large-cohere-count`

6.1.2 Small Cohere Dedicated

— **Capacités** : Fine-tuning et hébergement

— **Famille** : Modèles Cohere Command R

— **SKU** : `dedicated-unit-small-cohere-count`

6.1.3 Embed Cohere Dedicated

— **Capacités** : Hébergement uniquement (pas de fine-tuning des embeddings)

— **Famille** : Modèles d'embedding Cohere (English, Multilingual, versions Lite)

— **SKU** : `dedicated-unit-embed-cohere-count`

6.1.4 Large Meta Dedicated

— **Capacités** : Fine-tuning et hébergement

— **Famille** : Modèles Meta Llama (3.3, 3.1 70B, 3.2 Vision 11B/90B, 3.1 405B)

— **SKU** : `dedicated-unit-large-meta-count`

Important : Les limites de service sont mises à zéro par défaut. Vous devez demander une augmentation des limites de service pour activer ces unités dans votre compte.

6.2 Dimensionnement par Type de Modèle

6.2.1 Modèles Cohere

Modèle	Fine-Tuning	Hébergement	Type d'Unité
Command-R-Plus 08-2024	Non supporté	2 unités	Large Cohere
Command-R 08-2024	8 unités	1 unité	Small Cohere

Table 6.1 – Dimensionnement des modèles Cohere

6.2.2 Modèles Meta Llama

Modèle	Fine-Tuning	Hébergement	Type d'Unité
Llama 3.3/3.1	4 unités	1 unité	Large Meta

Table 6.2 – Dimensionnement des modèles Meta

6.2.3 Modèles d'Embedding

Modèle	Fine-Tuning	Hébergement	Type d'Unité
Cohere English/Multilingual	N/A	1 unité	Embed Cohere

Table 6.3 – Dimensionnement des modèles d'embedding

6.3 Exemple de Calcul de Coût

6.3.1 Scénario : Bob et le Fine-Tuning

Bob souhaite :

— Fine-tuner un modèle Cohere Command R 08-2024

— Héberger le modèle personnalisé après fine-tuning

— Effectuer le fine-tuning une fois par semaine

— Héberger le modèle 24/7 pendant tout le mois

6.3.2 Exigences en Ressources

— **Fine-tuning** : 8 unités Small Cohere Dedicated

— **Durée du fine-tuning** : 5 heures par session

— **Fréquence** : 4 fois par mois (1 fois/semaine)

— **Hébergement** : 1 unité Small Cohere Dedicated minimum

— **Durée d'hébergement** : 744 heures/mois (24/7)

6.3.3 Engagements de Service Minimum

Service	Engagement Minimum	Facturation
Hébergement	Mois complet (744 heures)	Pas de facturation partielle
Fine-tuning	1 heure minimum	Par heure après la première

Table 6.4 – Engagements de service

6.3.4 Calcul Détaillé

Coût du Fine-Tuning :

— Unités par session : 8

— Durée par session : 5 heures

— Unités-heures par session : $8 \times 5 = 40$

— Sessions par mois : 4

— Total unités-heures/mois : $40 \times 4 = 160$

Coût d'Hébergement :

— Unités : 1

— Durée : 744 heures/mois

— Total unités-heures/mois : 744

Coût Total :

$$\text{Coût mensuel} = (160 + 744) \times \text{Prix unitaire} \tag{6.1}$$

$$= 904 \times 6,50\$ \tag{6.2}$$

$$5\,900\$ \tag{6.3}$$

Note sur les prix : Les tarifs mentionnés sont à titre indicatif. Consultez toujours la documentation officielle Oracle pour les prix actuels, car ils peuvent évoluer.

Chapitre 7

Configuration du Fine-Tuning

7.1 Méthodes d'Entraînement Disponibles

OCI Generative AI propose deux méthodes de fine-tuning efficaces en paramètres (PEFT) :

7.1.1 T-Few (Task-Few)

Nous avons déjà exploré T-Few en détail. Pour rappel, cette méthode ajoute de petites couches d'assistance au modèle pour ajuster quelques éléments sans tout changer.

7.1.2 LoRA (Low Rank Adaptation)

LoRA fonctionne comme l'ajout d'engrenages spéciaux à une machine pour l'ajuster tout en laissant les parties principales inchangées.

Analogie Machine Complexe

Imaginez un modèle d'apprentissage profond comme une machine très complexe avec de nombreuses pièces. Pour faire exécuter une nouvelle tâche à la machine :

- **T-Few** : Ajouter de petits assistants (nouvelles couches)
- **LoRA** : Ajouter des engrenages spéciaux
- **Résultat** : Performance de nouvelle tâche sans refonte complète

7.2 Hyperparamètres de Fine-Tuning

7.2.1 Époques d'Entraînement Totales (Total Training Epochs)

- **Définition** : Nombre de fois que la machine étudie les données de tâche
- **Impact** : Plus d'époques = plus de temps d'étude
- **Considération** : Équilibrer entre apprentissage et sur-apprentissage

7.2.2 Taille de Lot d'Entraînement (Training Batch Size)

- **Définition** : Nombre d'éléments de données étudiés simultanément
- **Lots plus grands** : Accélération de l'apprentissage
- **Lots plus petits** : Insights plus détaillés, meilleure généralisation

7.2.3 Taux d'Apprentissage (Learning Rate)

— **Définition** : Vitesse d'ajustement des paramètres du modèle

— **Taux élevé** : Ajustements rapides (risque d'instabilité)

— **Taux faible** : Changements prudents (convergence plus lente)

7.2.4 Seuil d'Arrêt Anticipé (Early Stopping Threshold)

— **Définition** : Standard pour arrêter si l'amélioration est insuffisante

— **Objectif** : Éviter le sur-apprentissage

— **Critère** : Basé sur la métrique de validation

7.2.5 Patience d'Arrêt Anticipé (Early Stopping Patience)

— **Définition** : Durée d'attente avant arrêt définitif

— **Fonction** : Nombre d'époques sans amélioration tolérées

— **Équilibrage** : Patience vs efficacité computationnelle

7.2.6 Intervalle de Journalisation (Log Model Metrics Interval)

— **Définition** : Fréquence de vérification et enregistrement des progrès

— **Utilité** : Surveillance de l'entraînement en temps réel

— **Unité** : Exprimé en nombre d'étapes

7.3 Évaluation des Résultats de Fine-Tuning

7.3.1 Métrique 1 : Précision (Accuracy)

Définition

La précision mesure si les tokens générés par le modèle correspondent aux tokens annotés (tokens de référence corrects).

Calcul

$$\text{Précision} = \frac{\text{Nombre de tokens corrects}}{\text{Nombre total de tokens}} \times 100 \tag{7.1}$$

Exemple Pratique

Calcul de Précision

Vérité terrain : "Le chat était assis sur le tapis"
Prédiction modèle : "Le chat dormait sur le tapis"
Analyse :

- Tokens corrects : "Le", "chat", "sur", "le" (4/6)
- Tokens incorrects : "dormait" (au lieu d'/"était"), "tapis" (identique mais contexte différent)
- Précision : $\frac{4}{6} \times 100 = 67\%$

Limitations de la Précision

Limitation importante : Même si la sortie du modèle transmet le même sens mais utilise des mots différents, la précision peut rester faible. C'est pourquoi la précision n'est pas toujours la meilleure métrique pour le fine-tuning de modèles.

7.3.2 Métrique 2 : Perte (Loss)

Définition

La perte mesure à quel point les sorties générées du modèle sont erronées. Elle est calculée par la différence de distribution de probabilité entre la prédiction du modèle et la sortie réelle.

Caractéristiques

- **Perte = 0** : Toutes les sorties étaient parfaites
- **Perte élevée** : Sorties très aléatoires
- **Tendance** : La perte diminue à mesure que le modèle s'améliore

Exemple Comparatif

Comparaison de Perte

Vérité terrain : "Le chat était assis sur le tapis"
Prédiction 1 : "Le chat dormait sur le tapis"

- 4 tokens corrects
- "dormait" similaire à "était" (verbes de repos)
- "tapis" identique
- **Perte relativement faible** : erreurs mineures, contexte préservé

Prédiction 2 : "L'avion volait à minuit"

- Aucun mot ne correspond
- Phrase complètement non pertinente
- **Perte très élevée** : sortie complètement différente

7.3.3 Comparaison des Métriques

Métrique	Avantages	Inconvénients
Précision	— Simple à comprendre — Comparaison directe	— Ne capture pas le contexte — Pénalise les synonymes
Perte	— Considère le contexte — Mesure la qualité globale — Préférée pour l'IA générative	— Plus complexe à interpréter — Nécessite compréhension statistique

TABLE 7.1 – Comparaison des métriques d'évaluation

Recommandation : La perte est la métrique préférée pour l'évaluation du fine-tuning car l'IA générative n'a pas toujours une seule réponse correcte. Le contexte et le sens global sont plus importants que la correspondance exacte des mots.

Chapitre 8

Sécurité d'OCI Generative AI

8.1 Principes de Conception Sécurisée

La sécurité et la confidentialité des charges de travail client constituent un principe de conception essentiel pour OCI Generative AI. Cette approche sécurisée par conception garantit que vos données et modèles restent complètement isolés et protégés.

8.2 Isolation des Ressources

8.2.1 Isolation GPU Dédiée

- **GPU dédiés** : Les GPU alloués pour les tâches génératives d'un client sont isolés des autres GPU
- **Réseau RDMA dédié** : Chaque cluster fonctionne dans un réseau RDMA dédié
- **Allocation exclusive** : Les GPU ne sont alloués qu'à un seul client
- **Pas de partage inter-client** : Aucun partage de ressources entre différents clients

8.2.2 Isolation des Modèles et Données

Votre cluster GPU dédié ne traite que vos modèles :

- Modèles de base de votre tenancy
- Modèles fine-tunés personnalisés de votre tenancy
- Isolation complète des modèles entre clients
- Isolation complète des données entre clients

Exemple d'Isolation

Client 1 :

- Cluster AI dédié A
- Modèles personnalisés X et Y
- Application A accède uniquement aux modèles X et Y

Client 2 :

- Cluster AI dédié B
- Modèle personnalisé Z
- Application B accède uniquement au modèle Z

Résultat : L'application B ne peut jamais accéder aux modèles X ou Y du Client 1.

8.3 Restriction d'Accès aux Données

8.3.1 Accès Limité à la Tenancy

- L'accès aux données client est restreint dans la tenancy du client
- Les données d'un client ne peuvent pas être vues par un autre client
- Seules les applications d'un client peuvent accéder aux modèles personnalisés créés et hébergés dans sa tenancy

8.4 Intégration avec les Services de Sécurité OCI

Le service Generative AI s'appuie sur les services de sécurité robustes d'OCI :

8.4.1 OCI Identity and Access Management (IAM)

- **Authentification** : Vérification de l'identité des utilisateurs
- **Autorisation** : Contrôle granulaire des permissions d'accès
- **Contrôle d'accès basé sur les rôles** : Définition précise de qui peut accéder à quels modèles

Exemple de Contrôle d'Accès

- Application X → Modèle personnalisé X (autorisé)
- Application Y → Modèle de base (autorisé)
- Application X → Modèle personnalisé Y (non autorisé)

8.4.2 OCI Key Management

- **Stockage sécurisé** : Toutes les clés des modèles de base sont stockées de manière sécurisée
- **Gestion centralisée** : Gestion centralisée des clés de chiffrement
- **Rotation automatique** : Rotation périodique des clés pour une sécurité renforcée

8.4.3 OCI Object Storage

— **Stockage des poids** : Les poids des modèles fine-tunés sont stockés dans Object Storage

— **Chiffrement par défaut** : Tous les objets sont chiffrés par défaut

— **Intégration Key Management** : Les clés de chiffrement sont gérées par le service Key Management

8.5 Architecture Sécurisée Globale

8.5.1 Diagramme d'Architecture de Sécurité

L'architecture de sécurité suit un modèle en couches :

1. **Couche Infrastructure** : Clusters GPU dédiés avec réseaux RDMA isolés
2. **Couche Modèles** : Isolation complète des modèles par tenancy
3. **Couche Accès** : IAM pour l'authentification et l'autorisation
4. **Couche Données** : Object Storage chiffré avec Key Management
5. **Couche Applications** : Contrôle d'accès granulaire aux endpoints

8.5.2 Flux de Sécurité

1. **Requête client** : Authentification via IAM
2. **Vérification des permissions** : Autorisation d'accès au modèle spécifique
3. **Routage sécurisé** : Redirection vers le cluster dédié approprié
4. **Traitement isolé** : Exécution dans l'environnement GPU dédié
5. **Réponse chiffrée** : Retour sécurisé des résultats

8.6 Conformité et Gouvernance

8.6.1 Standards de Sécurité

OCI Generative AI respecte les standards de sécurité industriels :

— **Chiffrement en transit** : Toutes les communications sont chiffrées

— **Chiffrement au repos** : Données et modèles chiffrés dans le stockage

— **Audit complet** : Journalisation de tous les accès et opérations

— **Conformité réglementaire** : Respect des réglementations de protection des données

8.6.2 Monitoring et Alertes

— **Surveillance continue** : Monitoring 24/7 des clusters et accès

— **Détection d'anomalies** : Identification automatique des comportements suspects

— **Alertes en temps réel** : Notification immédiate des incidents de sécurité

— **Réponse automatique** : Mesures de protection automatiques en cas de menace

Responsabilité partagée : Bien qu'OCI fournisse une infrastructure sécurisée, les clients sont responsables de la sécurisation de leurs applications, de la gestion appropriée des identités et des accès, et du respect des bonnes pratiques de sécurité.

Conclusion

Récapitulatif des Concepts Clés

Ce cours complet sur OCI Generative AI a couvert les aspects fondamentaux du service, depuis les concepts de base jusqu'aux considérations avancées de sécurité. Nous avons exploré :

Modèles Foundationnels

— **Modèles de chat** : Command-R-Plus, Command-R-16k, et la famille Llama 3.1

— **Modèles d'embedding** : Cohere English, Multilingual, et leurs versions v3 optimisées

— **Tokenisation** : Compréhension des tokens et leur impact sur les performances

Techniques de Personnalisation

— **Ingénierie des prompts** : Few-shot, chain-of-thought, et bonnes pratiques

— **Fine-tuning** : T-Few et LoRA pour l'adaptation aux domaines spécifiques

— **RAG** : Intégration avec les bases de connaissances d'entreprise

Infrastructure Dédiée

— **Clusters AI dédiés** : Types, dimensionnement, et optimisation des coûts

— **Architecture multi-tenant** : Partage efficace des ressources GPU

— **Gestion des endpoints** : Déploiement et monitoring des modèles

Bonnes Pratiques

Approche Itérative

Adoptez une approche progressive pour optimiser vos modèles :

1. Commencez par des prompts simples avec évaluation baseline
2. Expérimentez avec le few-shot prompting
3. Implémentez RAG pour les données dynamiques
4. Utilisez le fine-tuning pour des adaptations spécialisées
5. Optimisez itérativement selon les métriques de performance

Considérations Économiques

— **Évaluez le coût total** : Incluez le développement, l'entraînement, et l'inférence
— **Optimisez l'utilisation** : Utilisez T-Few pour réduire les coûts de fine-tuning
— **Planifiez la charge** : Dimensionnez les clusters selon vos besoins réels
— **Surveillez les métriques** : Accuracy et loss pour mesurer l'efficacité

Sécurité et Gouvernance

— **Implémentez IAM correctement** : Contrôle d'accès granulaire
— **Utilisez les clusters dédiés** : Isolation complète des données
— **Auditez régulièrement** : Surveillance des accès et utilisation
— **Respectez les réglementations** : Conformité avec les standards industriels

Perspectives d'Évolution

Le domaine de l'IA générative évolue rapidement. Restez informés des :

— **Nouveaux modèles** : Versions améliorées des modèles foundationnels
— **Techniques d'optimisation** : Méthodes de fine-tuning plus efficaces
— **Intégrations étendues** : Nouveaux connecteurs et APIs
— **Capacités multimodales** : Support étendu pour images, audio, vidéo

Ressources Complémentaires

Pour approfondir vos connaissances :

— **Documentation Oracle** : Guides officiels et références API
— **Communauté OCI** : Forums et groupes d'utilisateurs
— **Formation continue** : Certifications et cours avancés
— **Expérimentation** : Projets pilotes et proof-of-concepts

Message final : L'IA générative représente une transformation majeure dans le traitement du langage naturel. Avec OCI Generative AI, vous disposez d'une plateforme robuste, sécurisée et économique pour déployer ces technologies à l'échelle entreprise. L'approche progressive et la compréhension des concepts fondamentaux abordés dans ce cours vous permettront de réussir vos projets d'IA générative.

Fin du cours OCI Generative AI
Bonne continuation dans vos projets d'IA générative !