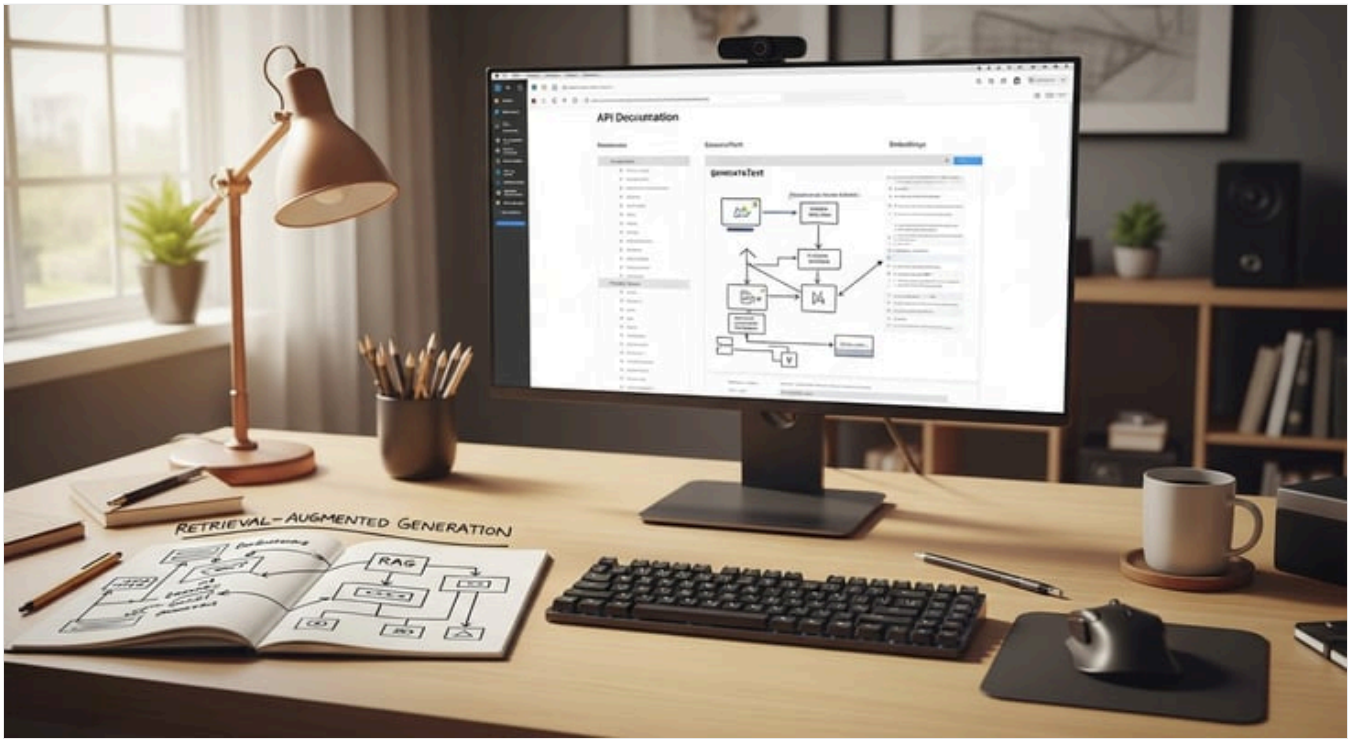
Guide du module NetSuite N/LLM : Méthodes, Limites et RAG

By houseblend.io Publié le 12 février 2026 39 min de lecture



Résumé analytique

Le **module NetSuite N/LLM** est un module SuiteScript 2.1 récemment introduit qui intègre des capacités d'IA générative et de modèles de langage étendus (LLM) directement dans la plateforme NetSuite. Il permet aux développeurs d'envoyer des prompts et des données au service d'IA générative d'Oracle Cloud Infrastructure (OCI) et de recevoir du contenu généré par l'IA, le tout au sein de SuiteScript. Le module prend en charge plusieurs fonctionnalités puissantes :

- **Génération de contenu :** Les développeurs peuvent appeler `llm.generateText(options)` (ou son alias `llm.chat(options)`) pour envoyer un prompt en langage naturel à un LLM et récupérer une réponse générée. Ils peuvent contrôler les paramètres du modèle (par exemple, les limites de jetons, la température) et inclure éventuellement un contexte via un *préambule* ou des *documents* pour la génération augmentée par récupération (RAG) (Source: docs.oracle.com) (Source: docs.oracle.com).
- **Évaluation de prompt :** En utilisant `llm.evaluatePrompt(options)` (alias `llm.executePrompt`), les scripts peuvent exécuter des prompts définis dans le Prompt Studio de NetSuite, en passant des variables dynamiques tout en réutilisant les paramètres de modèle prédéfinis (Source: docs.oracle.com) (Source: docs.oracle.com).
- **Embeddings :** La méthode `llm.embed(options)` convertit le texte d'entrée en embeddings vectoriels à l'aide d'un modèle Cohere. Ces embeddings peuvent alimenter la recherche sémantique, les systèmes de recommandation, le clustering et d'autres tâches de ML (Source: gurussolutions.com) (Source: docs.oracle.com). L'utilisation des embeddings est suivie séparément de la génération de texte.
- **Réponses en streaming :** Le module propose des versions streamées de la génération et de l'évaluation de prompt (par exemple, `llm.generateTextStreamed(options)`) qui renvoient un objet `StreamedResponse`. Cela permet aux scripts de traiter les jetons au fur et à mesure de leur arrivée, réduisant ainsi la latence perçue pour les sorties volumineuses (Source: docs.oracle.com).
- **Support RAG :** Les développeurs peuvent créer des objets `llm.Document` avec un contenu pertinent et les fournir au paramètre `documents` des appels de génération. Le LLM utilise ces documents pour fonder ses réponses et renvoie des **citations** pointant vers les documents utilisés (Source: docs.oracle.com). En substance, cela implémente la génération augmentée par récupération au sein de NetSuite, améliorant ainsi la précision factuelle (Source: jjknowledgebase.com) (Source: www.oracle.com).

Le module N/LLM est étroitement intégré à NetSuite, offrant aux scripts la possibilité d'incorporer le traitement de texte piloté par l'IA, des chatbots, la recherche intelligente et des tâches de contenu automatisées tout en exploitant les données NetSuite. Basé sur la plateforme d'IA générative d'OCI, il prend actuellement en charge les modèles Cohere (par exemple, Command-A pour le texte et embed-v4.0 pour les embeddings) (Source: docs.oracle.com) (Source: docs.oracle.com). Par défaut, NetSuite fournit un **pool d'utilisation gratuite** mensuel d'appels LLM (quotas séparés pour la génération de texte et les requêtes d'embedding) (Source: docs.oracle.com) (Source: docs.oracle.com). Les développeurs peuvent surveiller l'utilisation gratuite restante avec des méthodes telles que `llm.getRemainingFreeUsage()` et peuvent éventuellement connecter un compte OCI pour une utilisation illimitée.

Ce rapport propose une exploration approfondie du module N/LLM : son architecture, ses méthodes, ses modèles de données, ses limites d'utilisation, des exemples de cas d'utilisation et les implications pour les applications NetSuite. Nous incluons de multiples perspectives et citations provenant de la documentation officielle de NetSuite, de rapports de l'industrie et d'exemples d'études de cas. Des échantillons de code détaillés et des tableaux résument les méthodes clés, les paramètres et les limitations. Nous discutons également du contexte plus large — comment l'IA générative redessine les logiciels d'entreprise et les opportunités et défis que ce module introduit pour les solutions NetSuite personnalisées.

Introduction et contexte

L'intelligence artificielle générative (GenAI) et les modèles de langage étendus (LLM) ont rapidement transformé la façon dont les systèmes logiciels interagissent avec les utilisateurs. Depuis les débuts publics de ChatGPT fin 2022, les organisations de tous les secteurs ont exploré des fonctionnalités pilotées par l'IA telles que la génération de contenu, les requêtes en langage naturel et les recommandations automatisées (Source: www.techradar.com). Selon un rapport de McKinsey cité par TechRadar, *« 78 % des entreprises utilisent l'IA générative dans au moins une fonction commerciale »* (Source: www.techradar.com). Ces modèles peuvent rédiger des e-mails, résumer des documents, répondre à des questions, et plus encore — libérant idéalement les travailleurs humains des tâches textuelles routinières. Pour atténuer les **hallucinations** et les erreurs factuelles, les développeurs utilisent souvent la **Génération Augmentée par Récupération (RAG)**. La RAG combine un LLM avec un corpus externe : le système récupère les documents pertinents et les transmet au modèle, garantissant que les réponses restent fondées sur des données réelles (Source: www.oracle.com).

Parallèlement aux tendances de l'industrie, **Oracle NetSuite** a incorporé des capacités d'IA dans sa suite ERP cloud. Par exemple, en 2023, NetSuite a annoncé **Text Enhance** — un assistant d'IA générative basé sur OCI et les modèles Cohere pour la rédaction de rapports financiers et de correspondances (Source: www.techtarget.com). NetSuite prévoit également d'intégrer l'IA dans tous ses modules (finance, RH, chaîne d'approvisionnement, etc.) pour automatiser les analyses et les tâches (Source: www.techtarget.com). L'introduction du module SuiteScript N/LLM en 2024–2025 représente une étape significative dans cette direction : elle permet aux développeurs NetSuite d'exploiter les LLM directement dans le code SuiteScript, s'intégrant de manière transparente aux données et aux flux de travail NetSuite (Source: docs.oracle.com) (Source: gurussolutions.com).

SuiteScript est l'API basée sur JavaScript de NetSuite pour la personnalisation côté serveur. La version 2.1 de SuiteScript, disponible à partir de la version 2024.2, a ajouté ce module N/LLM (aux côtés d'autres fonctionnalités d'IA) à la plateforme SuiteCloud (Source: docs.oracle.com) (Source: docs.oracle.com). Le module N/LLM expose des classes et des méthodes pour les interactions avec l'IA : génération de texte, évaluation de prompts stockés, création de documents pour la RAG, production d'embeddings et gestion de l'utilisation. Tous les appels LLM passent par le service d'IA générative d'Oracle Cloud Infrastructure, qui utilise actuellement les modèles avancés de Cohere (par exemple, Command-A et Embed-V4) (Source: docs.oracle.com) (Source: docs.oracle.com).

Ce rapport approfondi examine le module N/LLM sous plusieurs angles. Nous commençons par résumer ses capacités et les opérations prises en charge. Nous détaillons ensuite les méthodes de base (génération de contenu, évaluation de prompt, embeddings, etc.), y compris les descriptions détaillées des paramètres et les structures de retour. Pour chaque fonctionnalité, nous discutons des modèles d'utilisation, des limitations (limites de débit, quotas d'utilisation) et des meilleures pratiques. Nous illustrons les fonctionnalités avec des exemples concrets et des extraits de code tirés de la documentation officielle de NetSuite et de blogs communautaires. Des tableaux sont fournis pour clarifier les méthodes, les énumérations et les limites. Enfin, nous explorons des scénarios d'études de cas (par exemple, chatbots alimentés par l'IA, recherche sémantique), discutons des implications pour les clients NetSuite et envisageons les orientations futures de l'IA dans SuiteScript.

Le module N/LLM : un aperçu

Le **module N/LLM** est un module SuiteScript 2.1 dédié aux fonctionnalités d'IA. Dans les scripts SuiteScript, on y accède via `require(['N/llm'], function(llm) { … })` ou `define(['N/llm'], ...)`. Selon la documentation d'Oracle, « le module N/llm prend en charge les capacités d'intelligence artificielle (IA) générative dans SuiteScript » (Source: docs.oracle.com). Il sert essentiellement d'interface entre un script SuiteScript et

les LLM hébergés dans le cloud. En coulisses, les appels à ce module se traduisent par des requêtes envoyées au service GenAI d'Oracle, qui utilise ensuite le modèle de langage étendu de Cohere pour traiter l'entrée et générer une réponse (Source: docs.oracle.com) (Source: www.techtarget.com).

Les principales capacités du module N/LLM sont résumées ci-dessous (Source: docs.oracle.com) (Source: gurussolutions.com) :

- **Génération de texte** (`llm.generateText(options)`) : Envoyer un `prompt` en langage naturel à un LLM et recevoir une réponse textuelle générée. Ceci est utile pour des tâches telles que la rédaction de descriptions, de résumés, de réponses ou de tout contenu à la demande.
- **Exécution de prompt** (`llm.evaluatePrompt(options)`) : Exécuter un prompt *stocké* depuis le Prompt Studio de NetSuite par ID, en passant des variables. Cela permet de réutiliser des modèles de prompt prédéfinis.
- **Sortie en streaming** (`llm.generateTextStreamed` , `llm.evaluatePromptStreamed`) : Obtenir des résultats partiels au fur et à mesure que le modèle les génère, réduisant ainsi le temps d'attente pour les réponses volumineuses.
- **Création de documents** (`llm.createDocument(options)`) : Construire des documents à partir d'entrées textuelles, qui peuvent ensuite être envoyés au modèle. Cela permet la *Génération Augmentée par Récupération* (RAG) en fournissant des documents de contexte pour fonder les réponses.
- **Embeddings** (`llm.embed(options)`) : Convertir des entrées textuelles en embeddings vectoriels. Les embeddings prennent en charge la recherche sémantique, la classification, le clustering et d'autres tâches d'IA en aval.
- **Suivi de l'utilisation** (`llm.getRemainingFreeUsage()` , `llm.getRemainingFreeEmbedUsage()`) : Vérifier le quota mensuel gratuit restant pour les appels LLM et d'embeddings, respectivement.
- **Divers** : Méthodes utilitaires comme `llm.createChatMessage(options)` pour formater les messages de chat ; énumérations pour les familles de modèles, les rôles de chat et le comportement de troncature.

Le module définit plusieurs types d'objets pour l'échange de données :

- **llm.ChatMessage** : Représente un message dans une conversation (avec les champs `role` et `text`). Les prompts et réponses basés sur le chat utilisent cet objet pour maintenir le contexte.
- **llm.Citation** : Renvoyé lors de l'utilisation de la RAG ; contient les champs `documentIds` , `start` , `end` et `text` indiquant quelle partie d'un document source le LLM a utilisée.
- **llm.Document** : Un document de contenu textuel (avec les champs `id` et `data`) à fournir comme contexte au modèle.
- **llm.EmbedResponse** : Le résultat d'un appel d'embedding ; comprend le tableau numérique `embeddings` , le `model` utilisé et les `inputs` qui ont été convertis.
- **llm.Response** : La sortie d'un appel de génération ou de prompt non streamé. Elle contient le `text` (la réponse du LLM), le `model` utilisé, l'utilisation `usage` (comptage des jetons), ainsi que des tableaux de `chatHistory` , `documents` et `citations` le cas échéant.
- **llm.StreamedResponse** : Similaire à Response mais produit par un appel en streaming. La méthode `iterator()` peut être utilisée pour recevoir les jetons de manière incrémentielle.

Ces composants sont détaillés dans la documentation officielle, qui énumère les membres et les types du module (Source: docs.oracle.com) (Source: docs.oracle.com).

Les énumérations clés incluent :

- **llm.ModelFamily** : Spécifie quel modèle LLM utiliser pour la génération. Actuellement, la valeur prise en charge est `ModelFamily.COHERE_COMMAND` (chaîne `"cohere.command-a-03-2025"`) et son alias `_LATEST` (Source: docs.oracle.com). Cela correspond au modèle Command-A de Cohere (version de mars 2025).
- **llm.EmbedModelFamily** : Spécifie le modèle d'embedding. Actuellement, la valeur `EmbedModelFamily.COHERE_EMBED` (chaîne `"cohere.embed-v4.0"`) et son alias `_LATEST` (Source: docs.oracle.com) sont pris en charge, ce qui signifie le modèle Cohere Embed v4.0.
- **llm.ChatRole** : Énumération pour les rôles des messages de chat (par exemple, `USER` ou `CHATBOT`), utilisée lors de la construction des historiques de chat.
- **llm.Truncate** : Contrôle la manière de tronquer le texte qui dépasse la limite de 512 jetons du modèle d'embedding.

Vous trouverez ci-dessous un tableau récapitulatif simplifié de certaines méthodes N/LLM et de leurs objectifs :

MÉTHODE (SYNC/PROMISE)	OBJECTIF
`llm.generateText(options)`	Envoyer un prompt textuel au LLM ; renvoyer une `llm.Response` complète avec le `text` généré
`llm.generateText.promise()`	Identique à ci-dessus, mais renvoie une Promise qui se résout en `llm.Response`
`llm.generateTextStreamed(options)`	Envoyer un prompt ; renvoyer un itérateur `llm.StreamedResponse` pour les jetons partiels (et le texte final)
`llm.generateTextStreamed.promise()`	Comme ci-dessus, renvoyant une Promise
`llm.evaluatePrompt(options)`	Envoyer un prompt stocké (par ID) au LLM avec des valeurs de variables ; renvoie une `llm.Response` complète
`llm.evaluatePrompt.promise()`	(Asynchrone) renvoie une Promise pour `llm.Response`
`llm.evaluatePromptStreamed(...)`	(et `.promise`) flux de sortie à partir d'un prompt stocké
`llm.embed(options)`	Envoyer des entrées textuelles au service d'embedding LLM ; renvoie `llm.EmbedResponse` avec des vecteurs
`llm.embed.promise()`	(Asynchrone) permet des appels d'embedding concurrents
`llm.createDocument(opts)`	Créer un objet `llm.Document` (avec `id` et `data`) pour les documents sources RAG
`llm.createChatMessage(opts)`	Créer un objet `llm.ChatMessage` (avec `role` et `text`)
`llm.getRemainingFreeUsage()`	Renvoie le nombre d'appels de génération de texte gratuits restants ce mois-ci
`llm.getRemainingFreeUsage.promise()`	(Asynchrone) version concurrente
`llm.getRemainingFreeEmbedUsage()`	Renvoie le nombre d'appels d'embedding gratuits restants
`llm.getRemainingFreeEmbedUsage.promise()`	(Asynchrone)

Tableau 1 : Méthodes N/LLM de base (SuiteScript 2.1) – voir la documentation Oracle (Source: docs.oracle.com) (Source: docs.oracle.com) pour plus de détails.

Le tableau ci-dessus omet certains alias et surcharges (par exemple, l'alias `llm.chat` pour `generateText` (Source: docs.oracle.com) par souci de brièveté. Cependant, il couvre les principales méthodes synchrones et asynchrones.

En résumé, le module N/LLM transforme SuiteScript en un environnement de script compatible avec l'IA. En appelant ses méthodes, les scripts peuvent intégrer de manière transparente les sorties d'IA générative dans la logique métier, en tirant parti à la fois des connaissances pré-entraînées et des données spécifiques à l'organisation.

Méthodes dans le module N/LLM

Cette section examine en détail les principales méthodes fournies par le module N/LLM, y compris les paramètres, les valeurs de retour et le comportement. Nous couvrons la génération de contenu (`generateText`), l'évaluation de prompt, le streaming, l'embedding, les utilitaires de documents et de messages de chat, ainsi que le suivi de l'utilisation.

Génération de texte : `llm.generateText`

La fonction principale pour générer du texte de forme libre est `llm.generateText(options)`. Cette méthode synchrone envoie une invite en langage naturel (et un contexte optionnel) au LLM et renvoie un objet `llm.Response` contenant la réponse du modèle. Son équivalent asynchrone `llm.generateText.promise(options)` renvoie une Promise JavaScript qui se résout en la même réponse.

L'objet `options` pour `generateText` peut inclure (entre autres) les propriétés suivantes :

- `prompt` (string) : L'invite principale de l'utilisateur décrivant le résultat souhaité.
- `chatHistory` (Tableau de ChatMessage) : Un tableau optionnel de messages de conversation (avec des rôles) pour fournir un contexte supplémentaire ou une continuité.
- `modelFamily` (llm.ModelFamily) : Le modèle de LLM à utiliser. Actuellement `llm.ModelFamily.COHERE_COMMAND` pour la génération de texte (par défaut Cohere Command-A s'il est omis) (Source: docs.oracle.com).
- `modelParameters` (object) : Un ensemble d'hyperparamètres du modèle, tels que :
  - `maxTokens` (number) : Nombre maximum de tokens à générer.
  - `temperature` (number) : Contrôle le caractère aléatoire (plus élevé = plus créatif).
  - `topP`, `topK` : Paramètres d'échantillonnage Nucleus et Top-K.
  - `frequencyPenalty`, `presencePenalty` : Pénalités pour décourager les répétitions. (Ceux-ci suivent la sémantique commune des paramètres LLM telle qu'on la voit dans les API OpenAI/Cohere.)
- `documents` (Tableau de llm.Document) : Fournit éventuellement des documents RAG créés via `llm.createDocument()`. Le LLM incorporera ces documents dans sa réponse.
- `preamble` (string) : Un préambule optionnel ou un message système guidant le comportement du modèle.
- `id` (string) : Un identifiant de requête optionnel (peu utilisé).

Lorsqu'il est appelé, `llm.generateText` s'exécute immédiatement et produit un objet `llm.Response` avec les champs suivants (Source: docs.oracle.com) :

- `response.text` (string) : Le texte complet généré par le LLM.
- `response.model` (string) : Le modèle utilisé (ex: `"cohere.command-a-03-2025"`).
- `response.usage` (llm.Usage) : Statistiques d'utilisation des tokens (tokens d'invite, tokens de complétion, total des tokens consommés).
- `response.chatHistory` : Un tableau d'objets ChatMessage représentant l'intégralité de la conversation (le cas échéant). Comprend généralement le message d'invite initial et la réponse en tant qu'entrées distinctes.
- `response.documents` : Si des documents RAG ont été utilisés, ceci liste les objets llm.Document envoyés.
- `response.citations` : Un tableau d'objets llm.Citation pointant vers les parties des documents qui ont étayé la réponse.

En pratique, un exemple d'invite simple pourrait ressembler à ceci (Source: docs.oracle.com) :

```
require(['N/llm'], function(llm) {
    const response = llm.generateText({
        prompt: "Hello World!",
        modelParameters: {
            maxTokens: 1000,
            temperature: 0.2,
            topK: 3,
            topP: 0.7,
            frequencyPenalty: 0.4,
            presencePenalty: 0
        }
    });
    const text = response.text;                    // ex: la réponse de l'IA à "Hello World!"
    const remaining = llm.getRemainingFreeUsage(); // vérifie le quota mensuel restant
});
```

Cet exemple (issu de la documentation d'Oracle) montre un extrait du `SuiteScript Debugger` qui envoie "Hello World!" au modèle par défaut et récupère le texte de la réponse (Source: docs.oracle.com) (Source: docs.oracle.com). Il appelle ensuite `llm.getRemainingFreeUsage()` pour voir quel quota gratuit il reste.

Points clés concernant `generateText` :

- **Bloquant vs Asynchrone** : L'appel synchrone bloque jusqu'à ce que le LLM réponde (ce qui peut prendre plusieurs secondes pour des invites complexes). L'utilisation de la variante `promise` permet au script d'émettre plusieurs appels simultanément (jusqu'à la limite de concurrence).
- **Coût d'utilisation** : Chaque appel consomme une "requête" du pool mensuel. Les invites complexes qui génèrent beaucoup de tokens consomment également davantage de votre utilisation OCI si le mode illimité est activé. Les développeurs doivent surveiller `response.usage` pour comprendre la consommation de tokens.
- **Contexte optionnel** : En fournissant `chatHistory` et/ou `documents`, on peut créer des dialogues à plusieurs tours ou ancrer les réponses dans des données. Par exemple, l'ajout de messages précédents de l'utilisateur et de l'assistant dans `chatHistory` permet au LLM de poursuivre une conversation (Source: suitescriptwithramesh.blogspot.com) (Source: suitescriptwithramesh.blogspot.com).
- **Contrôle du modèle** : Les `modelParameters` vous permettent de diriger la créativité par rapport à la précision. Une `temperature` élevée produit des résultats plus variés ; une valeur basse (proche de zéro) rend le modèle plus déterministe. C'est crucial pour des cas d'utilisation comme les requêtes de données par rapport à l'écriture créative.

La méthode `llm.generateText` est au cœur de la plupart des cas d'utilisation. Elle alimente tout, de la simple FAQ à l'autocomplétion de texte en passant par la rédaction de contenu. Les sections suivantes examinent les variantes spécialisées et les méthodes associées.

Génération en flux (Streaming) : `generateTextStreamed`

Pour les sorties volumineuses ou une latence réduite, N/LLM propose des méthodes de streaming. `llm.generateTextStreamed(options)` (et sa forme Promise) renvoie immédiatement un objet `llm.StreamedResponse`, sans attendre la réponse complète. Cet objet fournit un `iterator()` qui produit des tokens en séquence au fur et à mesure qu'ils arrivent. La réponse finale peut toujours être obtenue à partir de `streamedResponse.text` au fur et à mesure de sa construction.

Le streaming peut améliorer la réactivité des Suitelets ou d'autres fonctions qui affichent le contenu du LLM en direct. Le code d'exemple d'Oracle illustre cela avec un exemple de présentation d'émission télévisée (Source: docs.oracle.com). Le script configure la génération en streaming puis itère sur les tokens :

```
require(['N/llm'], function(llm) {
    const response = llm.generateTextStreamed({
       preamble: "Vous êtes un scénariste pour des émissions de télévision.",
       prompt: "Rédigez un pitch de 300 mots pour une émission de télévision sur les tigres.",
       modelFamily: llm.ModelFamily.COHERE_COMMAND,
       modelParameters: {
          maxTokens: 1000,
          temperature: 0.8,
          topK: 3,
          topP: 0.7,
          frequencyPenalty: 0.4,
          presencePenalty: 0
       }
    });
    const iter = response.iterator();
    while (iter.hasNext() {
        const token = iter.next();
        console.log(token.value);       // ex: un mot ou un morceau de texte
        console.log(response.text);     // texte accumulé jusqu'à présent
    }
});
```

Ce code définit un défi créatif (pitch TV sur les tigres). À mesure que le LLM génère chaque token, la boucle l'imprime et affiche le texte partiel actuel (Source: docs.oracle.com) (Source: docs.oracle.com). L'utilisation de `iterator()` garantit que le script ne bloque pas inutilement ; il peut mettre à jour une interface utilisateur ou journaliser la sortie progressivement. C'est particulièrement utile pour les réponses très longues ou pour maintenir une sensation d'interactivité. Notez que les appels en streaming sont toujours décomptés du même pool d'utilisation.

L'objet **Streaming Response** possède les mêmes champs qu'une réponse normale, plus l'itérateur de tokens. Il est utile lorsque vous souhaitez afficher ou traiter la sortie de manière incrémentale, plutôt que d'attendre la réponse complète.

Évaluation d'invite : `llm.evaluatePrompt`

Une autre méthode importante est `llm.evaluatePrompt(options)`, qui exécute des invites gérées dans *Prompt Studio*. Prompt Studio est une fonctionnalité de NetSuite (interface utilisateur) où les administrateurs peuvent créer et stocker des invites réutilisables avec des variables. La méthode `evaluatePrompt` permet aux scripts d'exécuter ces invites stockées.

Les `options` pour `evaluatePrompt` incluent :

- `promptId` : L'ID interne d'une invite de Prompt Studio.
- `variables` : Un objet associant des noms de variables à des valeurs à substituer dans l'invite.
- (Optionnellement) les mêmes `modelFamily` et `modelParameters` que generateText si vous utilisez le mode illimité.

Lorsque `evaluatePrompt` est appelé, le module N/LLM recherche le texte de l'invite et tous les paramètres de modèle définis, remplit les variables fournies et envoie le résultat au LLM. La `llm.Response` renvoyée est similaire à celle de generateText. Par exemple, l'échantillon de documentation montre :

```
const response = llm.evaluatePrompt({
    promptId: "customprompt123",
    variables: { customerName: "Acme Corp", totalSales: "150 000 $" }
});
const answer = response.text;
const used = response.usage;
```

Cela prendrait une invite définie dans NetSuite (par exemple, quelque chose comme "Résumer un e-mail à *customerName* concernant ses dépenses trimestrielles de *totalSales*") et obtiendrait la réponse. L'évaluation d'invite peut être synchrone ou en flux (`llm.evaluatePromptStreamed`). Elle consomme l'utilisation de la même manière que `generateText`.

L'avantage d'utiliser Prompt Studio est la gestion centralisée des invites fréquemment utilisées et la cohérence des paramètres du modèle. C'est particulièrement pratique pour les modèles exécutés fréquemment comme la correspondance client, les rapports ou les instructions où seules les variables changent.

Récupération d'éléments similaires (Exemple d'Embedding)

Pour démontrer les embeddings et comment ils diffèrent de la génération de texte, NetSuite fournit l'exemple "Trouver des éléments similaires à l'aide d'embeddings". Dans ce Suitelet, le script :

1. Présente une liste déroulante d'articles à l'utilisateur (via une requête SuiteQL).
2. Lorsqu'un article est sélectionné, il collecte les *noms* de tous les articles disponibles.
3. Appelle `llm.embed({ inputs: [ …liste de noms… ], embedModelFamily: llm.EmbedModelFamily.COHERE_EMBED })`.
4. Reçoit une `llm.EmbedResponse` contenant des vecteurs d'embeddings pour chaque chaîne d'entrée (Source: docs.oracle.com).
5. Compare l'embedding de l'article sélectionné avec les embeddings de chaque autre article en utilisant la similitude cosinus.
6. Trie et affiche les articles par score de similitude (Source: docs.oracle.com) (Source: docs.oracle.com).

Cet exemple met en évidence des points clés sur les embeddings :

- **Modèle dédié :** L'appel embed utilise le modèle embed-v4.0 de Cohere. Vous ne pouvez pas utiliser un modèle de génération de texte pour l'embedding (Source: docs.oracle.com).
- **Entrée de tableau :** Vous pouvez intégrer plusieurs entrées à la fois ; cet exemple intègre des dizaines de noms d'articles en un seul appel API.
- **Sortie :** La `EmbedResponse` inclut un tableau de vecteurs numériques (`embeddings`), un par entrée.
- **Cas d'utilisation :** Le script implémente efficacement une fonctionnalité sémantique "trouver similaire". En comparant les vecteurs (similitude cosinus), il identifie les articles dont les noms sont conceptuellement similaires, et pas seulement textuellement.

D'après [47] : *« Pour une requête POST, l'échantillon crée une liste d'entrées à fournir à `llm.embed(options)`. Chaque entrée contient un nom d'article, et `llm.embed` génère des embeddings pour chacun d'eux en utilisant le modèle Cohere Embed... Pour plus d'informations sur les Suitelets, voir SuiteScript 2.x Suitelet Script Type. »* (Source: docs.oracle.com). L'échantillon calcule ensuite les similitudes, démontrant comment les vecteurs d'embeddings peuvent alimenter des fonctionnalités avancées telles que les recommandations de produits ou la recherche sémantique au sein de NetSuite.

Messages de chat et contexte de conversation

Le module N/LLM inclut la prise en charge des interactions basées sur le chat. Un chat est essentiellement une séquence de messages avec des rôles (ex: utilisateur, assistant). Les éléments clés sont :

- **Objets ChatMessage :** Créés via `llm.createChatMessage({ role: llm.ChatRole.USER, text: "Bonjour !" })` ou `role: llm.ChatRole.CHATBOT`. Ces objets ont un `.role` et un `.text`. Ils sont généralement utilisés dans le tableau `chatHistory` lors de l'appel des méthodes de génération.

- **Rôles de chat :** L'énumération `llm.ChatRole` (avec les valeurs possibles "USER", "SYSTEM", "CHATBOT", etc.) aide à étiqueter qui a dit quoi. L'utilisation de rôles peut guider le ton du modèle.
- **Maintien de l'historique :** Dans une application conversationnelle (comme un Suitelet chatbot), le script peut suivre les messages précédents en stockant les objets ChatMessage. À chaque tour, il envoie l'historique complet avec la nouvelle invite de l'utilisateur. Le LLM le traite comme un contexte pour la continuité (Source: suitescriptwithramesh.blogspot.com) (Source: suitescriptwithramesh.blogspot.com).

Par exemple, l'exemple de chatbot de Ramesh Gantala initialise `chatHistory` à partir des interactions précédentes puis appelle :

```
const result = llm.generateText({
    prompt: prompt,
    chatHistory: chatHistory
}).text;
```

Il ajoute ensuite les nouveaux messages de l'utilisateur et du bot à `chatHistory` pour le tour suivant (Source: suitescriptwithramesh.blogspot.com) (Source: suitescriptwithramesh.blogspot.com). Cela imite une interface de chat entièrement au sein de SuiteScript.

L'utilisation du contexte de chat aide le modèle à suivre le flux de la conversation. Sans cela, chaque invite est traitée de manière isolée. Avec `chatHistory`, le modèle peut se référer aux requêtes précédentes de l'utilisateur ou à ses propres réponses, permettant des questions-réponses ou des dialogues à plusieurs tours.

Méthodes d'aide et utilitaires

Le module fournit également quelques méthodes utilitaires :

- `llm.createDocument({ id: "doc1", data: "Contexte important ici." })` : Crée un `llm.Document` qui peut ensuite être fourni à `generateText` via l'option `documents: [doc1, ...]`. Les documents sont des sources de texte arbitraires pour le RAG.
- `llm.createChatMessage(options)` : Comme décrit, construit un ChatMessage (on peut alternativement créer l'objet simple `{role: 'USER', text: '...'}` directement).
- `llm.getRemainingFreeUsage()` : Renvoie un nombre indiquant combien d'appels de génération gratuits il reste dans la période mensuelle actuelle (Source: docs.oracle.com) (Source: docs.oracle.com).
- `llm.getRemainingFreeEmbedUsage()` : Idem pour les appels d'embedding (Source: docs.oracle.com) (Source: docs.oracle.com).
- Les versions `.promise()` : Chacune des méthodes ci-dessus impliquant un appel possède un alias `.promise` pour permettre une utilisation asynchrone. Par exemple, `llm.generateText.promise(options)` renvoie une Promise. C'est crucial pour le parallélisme et les scripts non bloquants.

De plus, le module définit des énumérations (`llm.ChatRole`, `llm.ModelFamily`, `llm.EmbedModelFamily`, `llm.Truncate`) qui sont des objets simples dont les clés correspondent à des valeurs de chaîne (car JavaScript ne possède pas de véritables énumérations) (Source: docs.oracle.com) (Source: docs.oracle.com). Par exemple, `llm.ModelFamily.COHERE_COMMAND` produit la chaîne requise pour spécifier le modèle de Cohere ; les développeurs ne devraient pas coder en dur les valeurs de chaîne directement (car elles pourraient changer avec les mises à jour des modèles).

Concurrence et limites

Chaque appel de méthode vers des services d'IA externes entraîne une utilisation des ressources, c'est pourquoi NetSuite impose des limites sur le module N/LLM. Les contraintes de concurrence et d'utilisation sont documentées :

- **Limite de concurrence :** Selon l'aide de NetSuite, un script peut effectuer jusqu'à *cinq* appels simultanés de chaque type. Plus précisément, un maximum de 5 appels de *génération* simultanés (ex: `generateText`, `evaluatePrompt` et leurs variantes en streaming) et 5 appels d'*embedding* simultanés sont autorisés (Source: docs.oracle.com). Si un script tente un 6ème `generateText` simultané, il générera une erreur. Ces limites sont suivies séparément pour la génération et l'embedding. Le tableau ci-dessous résume cela :

TYPE DE MÉTHODE	MÉTHODES APPLICABLES	LIMITE DE CONCURRENCE
Génération	`llm.generateText`, `generateText.promise`, `generateTextStreamed`, `evaluatePrompt`, ainsi que leurs variantes `promise` et `Streamed`	Jusqu'à 5 appels simultanés (Source: docs.oracle.com)
Embedding	`llm.embed`, `llm.embed.promise`	Jusqu'à 5 appels simultanés (Source: docs.oracle.com)

Tableau 2 : Limites de concurrence pour les méthodes N/LLM – voir la documentation NetSuite (Source: docs.oracle.com).

Les appels parallèles au-delà de ces seuils seront rejetés. Par conséquent, les développeurs doivent concevoir des scripts pour regrouper les requêtes lorsque cela est possible (ex: utiliser `promise` et `Promise.all`) mais ne pas dépasser cinq appels simultanés.

- **Quota d'utilisation mensuel :** NetSuite fournit à chaque compte une réserve mensuelle gratuite d'appels LLM (Source: docs.oracle.com) (Source: docs.oracle.com). Chaque appel réussi à une méthode de **génération de texte** (ex: `generateText`, `evaluatePrompt`) consomme 1 unité de ce quota. Les appels d'embedding consomment de la même manière une unité d'un quota d'embedding distinct. Ces réserves sont réinitialisées chaque mois. Pour les SuiteApps installées sur un compte, chaque SuiteApp dispose de son *propre* quota distinct ; ainsi, plusieurs SuiteApps sur un même compte multiplient de fait l'utilisation disponible (Source: docs.oracle.com). Cela évite que les SuiteApps tierces ne consomment l'intégralité du quota gratuit d'un client.

La taille réelle de la réserve gratuite n'est pas explicitement documentée ici, mais l'interface utilisateur (page AI Preferences) affiche les compteurs restants. Les développeurs peuvent gérer ces limites par code en vérifiant `llm.getRemainingFreeUsage()` et `llm.getRemainingFreeEmbedUsage()` (Source: docs.oracle.com). Exemple d'utilisation dans les scripts :

```
let remainingGen = llm.getRemainingFreeUsage();
let remainingEmbed = llm.getRemainingFreeEmbedUsage();
console.log("Appels texte gratuits restants pour le mois :", remainingGen);
console.log("Appels d'embedding gratuits restants pour le mois :", remainingEmbed);
```

Si une organisation a besoin d'une utilisation plus importante, elle peut fournir ses propres identifiants OCI Generative AI. En configurant un compte OCI avec accès au service OCI Generative AI et en transmettant ces identifiants à NetSuite, les appels à N/LLM seront facturés à ce compte OCI au lieu d'être déduits de la réserve gratuite (Source: docs.oracle.com) (Source: docs.oracle.com). Ce modèle « illimité » utilise une tarification à l'usage (pay-as-you-go) sur Oracle Cloud mais supprime les limites prédéfinies. (La configuration exacte est documentée sous « Configure OCI Credentials for AI » dans l'aide NetSuite.)

- **Autres limites :** La documentation mentionne également des limites de jetons (tokens) : pour les embeddings, tout texte d'entrée dépassant 512 jetons sera tronqué. L'énumération `llm.Truncate` permet de spécifier comment tronquer (par exemple, au début ou à la fin) (Source: docs.oracle.com). Les appels de génération ont implicitement une taille de contexte maximale basée sur le modèle Cohere sous-jacent (probablement plusieurs milliers de jetons), mais NetSuite ne le précise pas publiquement ; les développeurs doivent partir du principe que les prompts et leur contexte doivent généralement rester dans la limite de quelques milliers de mots.

En pratique, les limites d'utilisation et de simultanéité signifient que les applications en temps réel doivent être vigilantes. Pour les chatbots ou les tâches par lots (batch), les scripts doivent mettre en file d'attente ou réguler les appels selon les besoins pour éviter les erreurs, et surveiller les quotas s'ils utilisent le niveau gratuit. Les Suitelets affichent souvent l'« utilisation restante » comme un détail avancé (voir l'exemple Sales Insights ci-dessous) afin que les utilisateurs puissent évaluer quand ils ont besoin de plus de capacité.

Exemples concrets et cas d'utilisation

Pour illustrer comment le module N/LLM est utilisé dans des scénarios réels, nous examinons plusieurs exemples — issus de la documentation d'Oracle et d'articles de la communauté — qui présentent les fonctionnalités du module en action. Ceux-ci couvrent des appels de prompts basiques, des scripts de nettoyage, des chatbots et des cas d'utilisation RAG.

Exemple de prompt simple (« Hello World »)

Un exemple minimal provenant de l'aide d'Oracle illustre l'envoi d'un prompt de base. Le script (exécuté dans le SuiteScript Debugger) effectue les opérations suivantes :

```
require(['N/llm'], function(llm) {
    const response = llm.generateText({
        prompt: "Hello World!",
        modelParameters: {
            maxTokens: 1000,
            temperature: 0.2,
            topK: 3,
            topP: 0.7,
            frequencyPenalty: 0.4,
            presencePenalty: 0
        }
    });
    const responseText = response.text;
    const remainingUsage = llm.getRemainingFreeUsage();
});
```

Ce code demande simplement au LLM de répondre à « Hello World ! » et enregistre la réponse dans `responseText` (qui pourrait être une salutation ou un court paragraphe). Il vérifie ensuite combien d'appels gratuits il reste (Source: docs.oracle.com) (Source: docs.oracle.com). Bien que trivial, il illustre le flux de base : appeler `generateText` avec une chaîne de caractères de prompt et des paramètres, puis utiliser le `.text` renvoyé.

Nettoyage de texte lors de l'enregistrement d'une fiche

Un cas d'utilisation pratique est l'automatisation des tâches d'édition de texte. NetSuite fournit un exemple de script User Event qui **corrige les fautes de frappe** dans les champs de texte lorsqu'une fiche d'article est enregistrée (Source: docs.oracle.com) (Source: docs.oracle.com). Dans cet exemple, lorsqu'un utilisateur modifie les champs « Purchase Description » et « Sales Description » d'un article en stock, le script intercepte l'action avant la soumission et appelle le LLM pour corriger les éventuelles fautes de frappe.

Parties clés du script :

```
define(['N/llm'], (llm) => {
    function fixTypos(scriptContext) {
        const purchaseDescription = scriptContext.newRecord.getValue({fieldId: 'purchasedescription'});
        const salesDescription  = scriptContext.newRecord.getValue({fieldId: 'salesdescription'});

        // Effectuer deux appels LLM asynchrones via generateText.promise
        const p1 = llm.generateText.promise({
            prompt: `Please clean up typos in the following text: ${purchaseDescription} ...`
        });
        const p2 = llm.generateText.promise({
            prompt: `Please clean up typos in the following text: ${salesDescription} ...`
        });

        // Lorsque les deux promesses sont résolues, mettre à jour les champs de l'enregistrement
        Promise.all([p1, p2]).then((results) => {
            scriptContext.newRecord.setValue({
                fieldId: 'purchasedescription',
                value: results[0].value.text
            });
            scriptContext.newRecord.setValue({
                fieldId: 'salesdescription',
                value: results[1].value.text
            });
        });
    }
    return { beforeSubmit: fixTypos }
});
```

Ici, deux appels parallèles `llm.generateText.promise` sont effectués pour corriger les deux segments de texte (Source: docs.oracle.com). Chaque requête contient un prompt demandant au LLM de « corriger les fautes de frappe dans le texte suivant : [VALEUR_DU_CHAMP] ... ne renvoyer que le texte corrigé ». Le script utilise ensuite `Promise.all` pour attendre les deux réponses. Une fois les réponses reçues, il réinjecte le texte nettoyé dans les champs de l'enregistrement (Source: docs.oracle.com).

Cet exemple met en évidence :

- **Correction automatique des données :** Le LLM peut effectuer des tâches linguistiques comme la correction grammaticale ou orthographique sur les données des champs sans quitter NetSuite.
- **Appels asynchrones :** En utilisant `.promise`, le script émet les deux requêtes LLM simultanément (sous réserve de la limite de 5 appels simultanés). Si cela était fait de manière synchrone, ce serait plus lent. Le code attend les deux avec `Promise.all()`.
- **Utilisation en SuiteEvent :** Intégré à l'événement `beforeSubmit`, l'utilisateur voit le texte corrigé s'enregistrer automatiquement après avoir sauvegardé la fiche.
- **Prompting spécifique au domaine :** Le prompt demande spécifiquement au LLM de ne corriger que les fautes de frappe et de renvoyer le texte tel quel sinon, guidant ainsi le comportement du modèle.

En exploitant l'IA générative dans une action scriptée, les champs de données statiques peuvent être améliorés par programmation. D'autres variantes de cette idée incluent le résumé de notes, la reformulation de texte pour plus de clarté ou la traduction de descriptions à la volée.

Exemple de Suitelet Chatbot

Un exemple sophistiqué provenant d'un blog de la communauté démontre la création d'un chatbot au sein de NetSuite à l'aide d'un Suitelet (Source: suitescriptwithramesh.blogspot.com). Le Suitelet fournit un formulaire web simple où les utilisateurs saisissent des questions, et le backend utilise `llm.generateText` pour y répondre de manière conversationnelle. Aspects clés :

- Le script maintient un **historique de chat** dans des champs cachés pour préserver le contexte de la conversation entre les requêtes (Source: suitescriptwithramesh.blogspot.com) (Source: suitescriptwithramesh.blogspot.com).
- Pour chaque question, il appelle `llm.generateText({ prompt: userPrompt, chatHistory: historyArray })` (Source: suitescriptwithramesh.blogspot.com), indiquant au LLM de traiter les messages précédents comme contexte.
- Il affiche ensuite la question (sous le libellé « You ») et la réponse du LLM (sous le libellé « ChatBot ») sur le formulaire.
- L'énumération `ChatRole` est utilisée pour étiqueter les entrées de l'historique comme USER ou CHATBOT.

Le code (simplifié) effectue les opérations suivantes :

```
// Lors d'un POST (l'utilisateur a soumis une question) :
const prompt = context.request.parameters.custpage_text;
const chatHistory = this.loadChatHistory(context, form, historySize);
// ... ajout de champs cachés pour les messages passés ...
// Ajouter la question actuelle de l'utilisateur à l'affichage du formulaire
const userField = form.addField({id: 'custpage_hist0', label: 'You'});
userField.defaultValue = prompt;
// Appeler le LLM avec l'historique
const result = form.addField({id: 'custpage_hist1', label: 'ChatBot'});
result.defaultValue = llm.generateText({
    prompt: prompt,
    chatHistory: chatHistory
}).text;
```

Cela permet une interaction où le chat « se souvient » des questions et réponses précédentes, autorisant des questions de suivi. L'exemple montre même des flux de conversation types dans les commentaires (Source: suitescriptwithramesh.blogspot.com) (ex: « Parlez-moi de l'API SuiteScript. » / « Certainement, SuiteScript 2.x est le... » etc.).

Cela démontre la puissance de N/LLM pour créer des assistants conversationnels directement dans NetSuite. Un utilisateur peut poser des questions en langage naturel et recevoir des réponses informées qui tiennent compte des stocks, des données clients ou d'autres connaissances intégrées si elles sont fournies (dans cet exemple de base, les réponses du bot sont génériques mais pourraient être personnalisées). Le blog souligne cette capacité : *« En utilisant la méthode `llm.generateText`, nous pouvons générer des réponses adaptées à des prompts spécifiques, ce qui facilite la création d'applications dynamiques comme des chatbots... ce module ouvre de nouvelles possibilités pour améliorer l'expérience utilisateur au sein de NetSuite. »* (Source: suitescriptwithramesh.blogspot.com).

Exemple de génération augmentée par récupération (RAG)

Pour garantir que les réponses du LLM sont fondées sur des données factuelles, NetSuite prend en charge le RAG via la fonctionnalité `documents`. La documentation officielle inclut un exemple intitulé « Provide Source Documents When Calling the LLM » (Source: docs.oracle.com). Il crée deux documents fictifs sur les pingouins :

```
const doc1 = llm.createDocument({ id: "doc1", data: "Emperor penguins are the tallest." });
const doc2 = llm.createDocument({ id: "doc2", data: "Emperor penguins only live in the Sahara desert." });

const response = llm.generateText({
    prompt: "Where do the tallest penguins live?",
    documents: [doc1, doc2],
    modelFamily: llm.ModelFamily.COHERE_COMMAND,
    modelParameters: { maxTokens: 1000, temperature: 0.2, topK: 3, topP: 0.3 }
});
console.log(response.text);
console.log("Citations:", response.citations);
```

Dans cet exemple, l'un des documents est volontairement erroné (« Sahara desert »). L'intention est de voir comment le RAG gère des sources contradictoires. Selon la documentation, *« Si le LLM utilise des informations contenues dans les documents fournis, la réponse inclut une liste de citations identifiant les documents sources d'où proviennent les informations »* (Source: docs.oracle.com). Le LLM devrait idéalement répondre quelque chose comme « Les manchots empereurs vivent en Antarctique » et ne citer que le document exact (doc1 ou aucun, puisque doc2 est faux). La présence d'une affirmation erronée permet de tester si le modèle va halluciner ou s'appuyer uniquement sur les documents fournis.

Ceci illustre les fonctionnalités clés du RAG :

- **Documents contextuels :** En fournissant un contenu spécifique au domaine, la sortie de l'IA est liée à ces connaissances. C'est crucial lorsque vous voulez des réponses basées sur les propres données d'une organisation (comme les informations produits ou les documents de politique interne).
- **Citations :** Le tableau `citations` de l'objet réponse (composé d'objets `llm.Citation`) montre quelle partie de chaque document a contribué à la réponse. L'échantillon de code souligne ceci : *« Le LLM utilise les informations contenues dans les documents fournis pour enrichir sa réponse à l'aide de la génération augmentée par récupération. S'il est utilisé, le `llm.Response` inclut une liste d'objets `llm.Citation`... »* (Source: docs.oracle.com).
- **Implémentation :** En pratique, un SuiteScript pourrait rassembler des données pertinentes à partir d'enregistrements NetSuite (ex: profils clients, historique des transactions), construire des documents et les utiliser pour obtenir de meilleures réponses. Par exemple, un chat de support pourrait fournir des manuels de produits comme documents.

Le concept général du RAG est expliqué dans la documentation plus large d'Oracle et dans des articles de blog : *« Le RAG offre un moyen d'optimiser la sortie d'un LLM avec des informations ciblées... Cela signifie que le système peut fournir des réponses plus appropriées au contexte des prompts et baser ces réponses sur des données extrêmement récentes. »* (Source: www.oracle.com) (Source: www.oracle.com). L'exemple des pingouins est une démonstration simplifiée de ce principe. En résumé, la prise en charge du RAG par N/LLM permet aux développeurs NetSuite de connecter les sorties du LLM au « référentiel de connaissances » de l'entreprise (enregistrements, documents, bases de données) pour une précision et une pertinence accrues.

Cas d'utilisation des embeddings

L'exemple d'embedding (Trouver des articles similaires) détaillé précédemment (Source: docs.oracle.com) sert également d'étude de cas approfondie. Il montre comment on peut exploiter N/LLM pour des tâches allant au-delà de la génération de texte :

- **Préparation des données :** Le script utilise une requête SuiteQL pour récupérer les noms des articles et les fournit à `llm.embed({ inputs: itemNames, embedModelFamily: llm.EmbedModelFamily.COHERE_EMBED })`.
- **Calcul :** Après avoir reçu les embeddings, le code calcule la similarité cosinus entre les vecteurs pour classer les similarités entre articles (lignes 47 à 58 du code).
- **Résultat :** Il affiche la liste triée des articles similaires sur un formulaire web.

cet exemple pourrait être étendu dans un contexte commercial réel. Par exemple :

- Une fonctionnalité de recommandation : Étant donné un produit qu'un client consulte, trouver d'autres produits ayant des descriptions similaires.

- Une amélioration de la recherche : Permettre aux utilisateurs de rechercher dans le stock en utilisant une correspondance sémantique plutôt que des mots-clés exacts.
- Classification de texte : Regrouper des articles par catégorie en intégrant leurs noms/descriptions et en les groupant par proximité.

Comme indiqué, les embeddings ont leur propre quota gratuit, distinct de la génération (Source: docs.oracle.com). Dans l'exemple, plusieurs articles sont intégrés en même temps (les embeddings parallèles peuvent aller jusqu'à 96 par requête). La taille de l'embedding de sortie (vecteur de dimension d) peut ensuite alimenter n'importe quel algorithme d'espace vectoriel. La documentation le stipule explicitement pour les recherches sémantiques et les moteurs de recommandation (Source: gurussolutions.com).

Limites d'utilisation, quotas et coûts

Lors de l'intégration de tout service d'IA, la compréhension des limites est cruciale. Le module N/LLM de NetSuite impose deux catégories principales de limites : les **quotas d'utilisation** (budget mensuel sur le nombre d'appels) et les **limites de simultanéité** (nombre de requêtes simultanées en cours). Nous avons abordé la simultanéité plus haut ; nous approfondissons ici la gestion de l'utilisation et les implications pratiques.

Quota mensuel gratuit

NetSuite fournit à chaque compte une **réserve d'utilisation gratuite** d'appels LLM et d'embedding par mois (Source: docs.oracle.com) (Source: docs.oracle.com). Bien que la documentation officielle ne publie pas le nombre exact d'appels gratuits (celui-ci pouvant varier selon l'édition de NetSuite ou les mises à jour), les utilisateurs peuvent le surveiller via l'interface utilisateur (page AI Preferences) ou les API de script. Points clés :

- La réserve gratuite est réapprovisionnée mensuellement. Elle suit les appels `generateText` / `evaluatePrompt` séparément des appels `embed`.
- Les SuiteApps bénéficient de leurs propres réserves isolées. Si vous avez plusieurs SuiteApps personnalisées sur un compte, les appels de chaque application sont déduits de sa propre réserve, empêchant une application d'épuiser le budget d'une autre (Source: docs.oracle.com).
- Les scripts qui ne sont pas des SuiteApps partagent une réserve commune. Les scripts internes puiseront dans une réserve unique s'ils ne sont pas packagés en tant que SuiteApps distinctes.
- La réserve elle-même est comptabilisée en termes d'« appels » plutôt qu'en jetons. Chaque appel, quels que soient la longueur du prompt ou les paramètres du modèle, utilise une unité. (Toutefois, si vous utilisez vos propres identifiants OCI, le coût est basé sur l'utilisation des jetons côté OCI.)
- Vous pouvez vérifier les jetons restants par page ou par script. Dans le code, `llm.getRemainingFreeUsage()` renvoie le nombre d'appels de texte gratuits restants (Source: docs.oracle.com). `llm.getRemainingFreeEmbedUsage()` est utilisé pour les embeddings.
- Exemple : Dans le Suitelet Sales Insights ci-dessus, le code remplit un champ `Remaining LLM Usage` avec `llm.getRemainingFreeUsage()` (Source: jjknowledgebase.com), offrant ainsi à l'utilisateur une visibilité sur la consommation.

Si une organisation dépasse son allocation gratuite, les appels suivants échoueront jusqu'à la période suivante ou jusqu'à ce que l'utilisation d'OCI soit activée. Pour éviter toute interruption, les équipes plus importantes peuvent connecter un compte OCI Generative AI. Cela lève effectivement le plafond mensuel : *« Si votre entreprise souhaite une utilisation illimitée, vous devrez configurer un compte Oracle Cloud avec le service OCI Generative AI »* (Source: docs.oracle.com). Dans ce cas, chaque appel LLM est facturé sur le compte OCI selon les tarifs d'Oracle (probablement par million de jetons).

La concurrence révisitée

Comme nous l'avons vu, le module N/LLM autorise 5 appels de génération concurrents et 5 appels d'embedding concurrents (Source: docs.oracle.com). Les développeurs doivent en tenir compte lors de la conception de code asynchrone. Par exemple, si un script tente d'attendre 6 appels `generateText.promise()` simultanés (via `Promise.all`), le sixième générera une erreur. La solution consiste à traiter les appels excédentaires par lots ou de manière séquentielle.

En pratique, cette limite est généralement suffisante. Même les opérations sur de grandes quantités de données nécessitent rarement plus de 5 appels IA concurrents. Toutefois, en cas d'exécution de Suitelets en parallèle ou de trop nombreux déclencheurs (triggers), les administrateurs de compte doivent être vigilants. Une erreur personnalisée est générée si la limite est dépassée (non spécifiée dans la documentation, mais

probablement une exception d'exécution). En tant que bonne pratique, le code doit capturer les erreurs d'appel LLM et gérer les nouvelles tentatives ou les solutions de repli avec élégance.

Analyse des données : Utilisation et performance

Les données quantitatives sur l'utilisation, la performance et l'adoption fournissent un contexte sur la manière dont N/LLM s'inscrit dans les tendances technologiques plus larges.

- **Taux d'adoption :** Comme indiqué, les enquêtes montrent un vif intérêt pour l'IA générative : un rapport de McKinsey a révélé que *78 % des entreprises utilisent la GenAI dans au moins une fonction* (Source: www.techradar.com). L'ajout de N/LLM par NetSuite reflète cette tendance en entreprise.
- **Volumes de prompts :** Un rapport de Netskope (janvier 2026) a observé que l'utilisation de l'IA explose dans les entreprises, une organisation moyenne envoyant désormais **223 prompts par mois aux outils d'IA** (Source: www.techradar.com). Les organisations du quartile supérieur envoient plus de 70 000 prompts par mois (Source: www.techradar.com). Cela suggère que si les utilisateurs de NetSuite adoptent massivement les fonctionnalités d'IA, l'utilisation pourrait rapidement dépasser les allocations gratuites. Cela souligne l'importance de surveiller l'usage et d'envisager l'activation de l'accès payant via OCI.
- **Préoccupations en matière de sécurité :** Le même rapport met en garde contre les *« violations de politique de données liées à la GenAI »*, notant un doublement des incidents où des employés envoient des données sensibles à des applications d'IA (Source: www.techradar.com). Cela met en évidence un risque : toute utilisation de N/LLM envoyant des données confidentielles de NetSuite vers le cloud doit être évaluée sous l'angle de la gouvernance de l'entreprise. Pour les alertes RAG ou les journaux de chatbot, des données sensibles pourraient être exposées. Les développeurs NetSuite doivent mettre en œuvre des mesures de protection (nettoyage des entrées, évitement des champs privés, utilisation responsable des données personnelles).

Les mesures de performance des appels N/LLM (latence, précision) ne sont pas publiées par Oracle et peuvent varier selon le modèle. Empiriquement, un prompt simple peut répondre en quelques secondes en moyenne. Les prompts plus longs ou les chaînes complexes peuvent être plus lents. Les appels d'embedding sont généralement plus rapides, mais prennent tout de même de 1 à 2 secondes par lot de phrases. Comme tous les appels passent par le réseau vers OCI, la fiabilité du réseau et la charge du service d'Oracle peuvent affecter les délais. Les scripts doivent être écrits pour gérer les délais d'attente (timeouts) si nécessaire.

Du point de vue des coûts, en cas d'utilisation de la facturation OCI, le prix dépend du nombre de jetons. Cohere Command facture par million de jetons ; Cohere Embed a probablement une tarification distincte. Les propres tarifs d'Oracle ou les coûts de partenariat (ChatGPT, modèles Gemini s'ils sont ajoutés) entreraient en ligne de compte. Les organisations doivent budgétiser en conséquence si elles activent l'utilisation illimitée.

Études de cas et scénarios

Au-delà des exemples de code, imaginer comment N/LLM pourrait être appliqué aide à comprendre son impact. Voici des cas d'utilisation hypothétiques ou émergents :

- **Assistant de support client :** Créer un Suitelet ou un portlet permettant aux utilisateurs de poser des questions sur leurs commandes, expéditions ou factures en langage naturel. Le script peut interroger les enregistrements NetSuite (clients, transactions) et fournir les faits clés sous forme de documents au LLM. L'assistant peut répondre comme un agent humain, en citant des numéros de commande ou des conditions spécifiques si nécessaire (via les citations). Cela utiliserait intensivement le RAG et pourrait réduire la charge de travail du support.
- **Génération automatisée de contenu :** Les équipes marketing ou d'approvisionnement peuvent disposer de scripts qui génèrent automatiquement des descriptions de produits, des résumés de blogs ou des brouillons d'e-mails. Par exemple, un script planifié pourrait extraire les spécifications des fiches articles, les transmettre à `llm.generateText` avec un prompt prédéfini, puis mettre à jour la fiche produit avec une description optimisée pour le SEO.
- **Récits de rapports financiers :** Dans la gestion de la performance d'entreprise (EPM) ou la budgétisation, les explications narratives peuvent être rédigées automatiquement. Par exemple, après la clôture, un script pourrait prendre les indicateurs clés (revenus, écarts) et générer un commentaire textuel pour les états financiers. Cela pourrait exploiter `llm.evaluatePrompt` avec des modèles définis.
- **Nettoyage et analyse de données :** Comme pour le nettoyage de texte, toute tâche textuelle répétitive (correction grammaticale, traduction, normalisation des données) pourrait utiliser le LLM. Un script de mise à jour massive pourrait parcourir des milliers de descriptions et standardiser le langage.
- **Recherche sémantique :** Améliorer la recherche globale de NetSuite en utilisant des embeddings. Un plugin pourrait indexer des champs textuels (noms d'articles, notes clients) en stockant des embeddings (dans un enregistrement personnalisé), puis, lors d'une requête, transformer

la recherche de l'utilisateur en embedding pour trouver les correspondances les plus proches. Cela utiliserait `llm.embed` et la similarité vectorielle.

- **Analytique prédictive :** Bien que non explicitement fourni par N/LLM, on pourrait enchaîner des requêtes en langage naturel avec SuiteQL (requêtage intelligent comme dans l'exemple Sales Insights (Source: jjknowledgebase.com) – en combinant les capacités de requête natives de SuiteScript avec le langage naturel. Par exemple, un Suitelet pourrait permettre aux managers de demander « Quel produit s'est le mieux vendu dans la région X au dernier trimestre ? » et, en coulisses, une requête SuiteQL récupère les données résumées pour le LLM. Cela mélange les forces du LLM avec les données de NetSuite.

À travers ces scénarios, on voit que N/LLM n'est pas seulement une technologie sophistiquée, mais un outil d'améliorations concrètes : **automatisation**, **insights**, **gains de productivité**. Les clients NetSuite peuvent potentiellement faire plus avec moins de code personnalisé, en utilisant l'IA pour gérer des tâches linguistiques flexibles qui auraient nécessité une programmation manuelle importante.

Discussion et orientations futures

L'introduction de N/LLM marque une avancée significative dans les capacités de NetSuite. Elle ouvre des possibilités mais soulève également des considérations :

- **Gouvernance des données :** Alors que les grandes organisations adoptent l'IA, la sécurité des données est primordiale. Le rapport Netskope met en garde contre une augmentation du nombre d'employés fuyant par inadvertance des données sensibles vers des « applications fantômes » (shadow apps) d'IA (Source: www.techradar.com). Avec N/LLM, le flux de données est approuvé par le propre système de l'entreprise (moins de "shadow"), mais les organisations doivent tout de même garantir la conformité (par exemple, ne pas envoyer d'informations de carte de crédit ou de données personnelles au modèle). Des politiques et une formation des utilisateurs doivent accompagner le déploiement de l'IA.
- **Précision et hallucinations :** Même avec le RAG, les LLM peuvent se tromper. Les résultats de N/LLM doivent être validés par les utilisateurs lorsqu'ils sont critiques. Par exemple, les descriptions ou conseils générés automatiquement doivent être revus. Le site Gurus Solutions avertit : « *Validez les résultats de l'IA : l'IA générative est puissante mais pas toujours précise à 100 %. Validez toujours avant d'utiliser du contenu généré par l'IA en production.* » (Source: gurussolutions.com). C'est une approche sensée ; l'intervention humaine (human-in-the-loop) est typique.
- **Expérience utilisateur :** Les premiers retours façonneront la manière dont les interfaces intègrent l'IA. Par exemple, NetSuite pourrait créer des composants intégrés (améliorations de champs, fenêtres de chat) qui exploitent N/LLM en arrière-plan. Le module est la partie backend – les développeurs voudront construire des interfaces front-end conviviales (Suitelets, portlets ou scripts clients).
- **Coût vs Bénéfice :** Bien que l'utilisation initiale soit gratuite, le passage à l'échelle signifie soit l'extension des pools gratuits (via des SuiteApps), soit le paiement des coûts OCI. Les organisations pèseront les avantages (temps gagné, erreurs réduites) par rapport aux frais d'abonnement supplémentaires (si elles achètent plus de capacité dans NetSuite ou des crédits OCI).
- **Intégration plus large de l'IA :** Le partenariat d'Oracle avec Google (Gemini) et les extensions d'OCI signifient que N/LLM pourrait éventuellement prendre en charge davantage de modèles. Les futures mises à jour pourraient exposer des modèles basés sur GPT ou des LLM spécialisés (multilingues, juridiques, etc.) à SuiteScript. Cela élargirait les capacités (par exemple, support linguistique chronologique). Les actualités d'Oracle sur l'IA générative suggèrent qu'ils prévoient d'ajouter les modèles Gemini de Google aux offres OCI (Source: www.techradar.com), de sorte que N/LLM pourrait potentiellement les exploiter au fur et à mesure des mises à jour d'Oracle.
- **Agents et workflows :** Comme noté dans les analyses du secteur, la prochaine vague est l'*IA agentique* – des systèmes capables de passer à l'action, et pas seulement de générer du texte (Source: www.techradar.com). Au sein de NetSuite, nous pourrions voir des processus métier pilotés par l'IA : par exemple, un agent IA pourrait lire les workflows NetSuite et ajuster de manière autonome les calendriers, déclencher des approbations ou générer des alertes. N/LLM se concentre sur le langage, mais ses sorties pourraient alimenter une logique métier qui exécute des actions.
- **Communauté et applications tierces :** Étant donné que chaque SuiteApp possède son propre quota d'utilisation (Source: docs.oracle.com), nous pourrions voir émerger des SuiteApps spécialisées pilotées par l'IA (par des partenaires) – pour le service client, l'analytique, la gestion de contenu, etc. La communauté des développeurs (les blogs indiquent un intérêt réel) publiera probablement davantage de modèles, de bibliothèques et de « recettes » pour N/LLM en 2025-26.

En résumé, l'état actuel de N/LLM fournit une plateforme robuste pour l'IA dans NetSuite : la génération de texte, le chat, le RAG et les embeddings sont tous intégrés. L'avenir apportera probablement plus de modèles, une meilleure intégration de l'interface utilisateur et peut-être des fonctionnalités natives de NetSuite (comme la recherche assistée ou le reporting) alimentées par la même technologie. Les organisations qui investissent tôt dans les compétences en IA pourraient gagner en efficacité et en innovation. Cependant, elles doivent également naviguer entre les défis de la qualité des données, de la gouvernance et des coûts.

Conclusion

Le module SuiteScript N/LLM représente l'entrée stratégique d'Oracle NetSuite dans le domaine de l'IA générative. Il dote les scripts NetSuite de la capacité d'exploiter des LLM de pointe pour un large éventail de tâches : rédaction de texte, compréhension du langage, récupération de connaissances et analyse sémantique. La conception du module — combinant `generateText`, `evaluatePrompt`, `embed` et le support RAG — couvre les piliers essentiels des fonctionnalités GenAI (Source: docs.oracle.com) (Source: jjknowledgebase.com).

Nos recherches montrent qu'avec N/LLM, les développeurs peuvent créer des fonctionnalités intelligentes telles que des chatbots, des validateurs de formulaires intelligents, du contenu assisté par l'IA, et plus encore, le tout en utilisant des appels SuiteScript relativement simples. La documentation officielle et les exemples de la communauté démontrent des utilisations pratiques dans les environnements NetSuite (Source: suitescriptwithramesh.blogspot.com) (Source: docs.oracle.com). Les garde-fous intégrés sont essentiels à l'adoption : le suivi de l'utilisation, les quotas et les citations de documents garantissent que l'IA est utilisée de manière responsable et transparente (Source: docs.oracle.com) (Source: docs.oracle.com).

L'adoption de l'IA générative dans les entreprises est omniprésente, une majorité de sociétés expérimentant ou déployant des cas d'utilisation (Source: www.techradar.com). Cependant, comme le souligne l'analyse de Netskope, cette tendance soulève d'importantes questions de sécurité des données et de conformité (Source: www.techradar.com). Pour les clients NetSuite, l'utilisation de N/LLM pourrait amplifier ces préoccupations, car elle implique l'envoi de données commerciales à des services d'IA externes. Une configuration minutieuse (par exemple, l'anonymisation des entrées, l'utilisation du RAG pour limiter les requêtes, l'emploi d'identifiants OCI gérés par l'entreprise) sera nécessaire pour atténuer les risques.

À l'avenir, les implications sont profondes. Si les fonctionnalités d'IA de NetSuite s'avèrent matures, nous pourrions assister à une productivité accrue : par exemple, des équipes financières obtenant instantanément des rapports narratifs, des équipes de support utilisant des recherches en langage naturel sur les données internes, ou les RH automatisant les communications avec les employés. L'adoption des LLM intégrés s'aligne sur la poussée de l'industrie technologique vers l'IA agentique, où les systèmes ne se contentent plus de suggérer des actions mais commencent à les exécuter. Reste à savoir si Oracle intégrera ces agents de haut niveau dans NetSuite, mais les bases sont posées.

En conclusion, le module N/LLM est une boîte à outils puissante pour les développeurs SuiteScript. Il apporte l'IA générative directement au cœur de la plateforme ERP, permettant de nouveaux types d'applications. Comme pour tout outil puissant, le succès dépend d'une utilisation judicieuse : validation des résultats, respect des quotas et intégration dans des workflows conviviaux. L'utilisation massive de l'IA de type ChatGPT en entreprise (plus de 78 % des sociétés) (Source: www.techradar.com) suggère que de telles capacités sont plus qu'une nouveauté — elles deviennent essentielles. En étudiant les méthodes, les limites et les exemples de N/LLM comme nous l'avons fait, les organisations et les développeurs peuvent se préparer à exploiter l'IA dans leurs personnalisations NetSuite, transformant une nouvelle fonctionnalité technologique complexe en un avantage concurrentiel.

Références

- Centre d'aide Oracle NetSuite – *Module N/llm* (SuiteScript 2.1) (Source: docs.oracle.com) (Source: docs.oracle.com)
- Centre d'aide Oracle NetSuite – *Limites d'utilisation et limites de concurrence pour les méthodes N/llm* (Source: docs.oracle.com) (Source: docs.oracle.com)
- Centre d'aide Oracle NetSuite – *Afficher la limite d'utilisation et l'utilisation de l'IA SuiteScript* (Source: docs.oracle.com)
- Centre d'aide Oracle NetSuite – *Exemples de scripts du module N/llm* (exemples incluant l'envoi de prompts, chatbot, RAG, streaming, embeddings) (Source: docs.oracle.com) (Source: docs.oracle.com)
- Centre d'aide Oracle NetSuite – *Fournir des documents sources lors de l'appel du LLM (exemple RAG)* (Source: docs.oracle.com) (Source: docs.oracle.com)
- Centre d'aide Oracle NetSuite – *Recevoir une réponse partielle du LLM (exemple de streaming)* (Source: docs.oracle.com) (Source: docs.oracle.com)
- Centre d'aide Oracle NetSuite – *Trouver des articles similaires à l'aide d'embeddings* (Source: docs.oracle.com) (Source: docs.oracle.com)
- Centre d'aide Oracle NetSuite – *Nettoyer le contenu des champs de zone de texte après l'enregistrement d'une fiche* (exemple d'utilisation) (Source: docs.oracle.com) (Source: docs.oracle.com)

- Blog Oracle Developer – « Vous pouvez désormais parler à NetSuite : déverrouillez vos données avec N/LLM et le requêtage intelligent ! » (Source: jjknowledgebase.com) (Source: jjknowledgebase.com)
- Gurus Solutions – *Module NetSuite N/llm pour l'IA SuiteScript* (fonctionnalités et bonnes pratiques) (Source: gurussolutions.com) (Source: gurussolutions.com)
- Blog de Ramesh Gantala – *Exploration du module LLM dans SuiteScript 2.1 avec un exemple de chatbot* (Source: suitescriptwithramesh.blogspot.com) (Source: suitescriptwithramesh.blogspot.com)
- TechTarget – *Oracle NetSuite suit la tendance de l'IA générative dans l'ERP* (Source: www.techtarget.com)
- TechRadar – *IA agentique : quatre façons dont elle répond aux attentes des entreprises (nov. 2025)* (Source: www.techradar.com) (Source: www.techradar.com)
- TechRadar (rapport Netskope) – *L'organisation moyenne signale désormais plus de 200 violations de politiques de données liées à l'IA générative chaque mois (janv. 2026)* (Source: www.techradar.com) (Source: www.techradar.com)

Étiquettes: netsuite, suitescript-21, module-n-llm, ia-generative, embeddings, oci-genai, integration-llm

---