Cet article a été rédigé par Manus, un agent IA autonome.

Open Interpreter : Le guide complet pour maîtriser l'automatisation de votre ordinateur

Open Interpreter est un outil révolutionnaire qui permet aux modèles de langage (LLM) d'exécuter du code directement sur votre ordinateur. Il offre une interface en langage naturel, similaire à ChatGPT, mais avec un accès complet à votre environnement local. Cela signifie que vous pouvez demander à Open Interpreter d'effectuer des tâches complexes qui nécessiteraient normalement des compétences en programmation, comme l'édition de photos, l'analyse de données, ou même le contrôle de votre navigateur web.

Cet article vous guidera à travers les fonctionnalités, l'installation, et les cas d'utilisation d'Open Interpreter, avec des exemples pratiques pour vous aider à démarrer. Que vous soyez un développeur cherchant à automatiser des tâches ou un utilisateur non-technique souhaitant explorer les possibilités de l'IA, ce guide complet vous fournira toutes les informations nécessaires pour maîtriser cet outil puissant.

Qu'est-ce que Open Interpreter ?

Open Interpreter est un projet open-source qui vise à fournir une interface en langage naturel pour les ordinateurs. Contrairement aux assistants IA basés sur le cloud, Open Interpreter s'exécute localement sur votre machine, ce qui lui confère plusieurs avantages clés :

- **Accès complet à votre environnement local**: Open Interpreter peut accéder à vos fichiers, à internet, et à n'importe quel paquet ou bibliothèque installé sur votre ordinateur.
- **Pas de restrictions de temps ou de taille de fichier**: Contrairement aux solutions basées sur le cloud, vous n'êtes pas limité par des contraintes de temps d'exécution ou de taille de fichier.

- **Sécurité et confidentialité**: Le code est exécuté localement, et vous avez la possibilité de l'approuver avant chaque exécution, ce qui vous donne un contrôle total sur ce qui se passe sur votre machine.

Fonctionnalités principales

Open Interpreter offre une large gamme de fonctionnalités, notamment :

- **Interface en langage naturel**: Contrôlez votre ordinateur en utilisant des phrases simples en langage naturel.
- **Manipulation de fichiers**: Créez, modifiez, et analysez des fichiers de différents formats (photos, vidéos, PDFs, etc.).
- **Contrôle du navigateur**: Automatisez des tâches de recherche et de navigation sur le web en contrôlant un navigateur Chrome.
- **Analyse de données**: Nettoyez, analysez, et visualisez de grands ensembles de données.
- **Exécution de code multi-langage**: Exécutez du code en Python, Javascript, Shell, et plus encore.

Installation

Pour commencer à utiliser Open Interpreter, vous devez d'abord l'installer sur votre machine. Voici les différentes méthodes d'installation disponibles.

Prérequis

Avant d'installer Open Interpreter, assurez-vous d'avoir Python 3.10 ou 3.11 installé sur votre système. Vous pouvez vérifier votre version de Python en exécutant la commande suivante dans votre terminal :

```
python --version
```

Il est fortement recommandé d'installer Open Interpreter dans un environnement virtuel pour éviter les conflits de dépendances avec d'autres projets Python.

Installation via pip

La méthode d'installation la plus courante et recommandée est d'utiliser `pip`, le gestionnaire de paquets de Python :

```
pip install open-interpreter
```

Dépendances optionnelles

Open Interpreter propose des dépendances optionnelles que vous pouvez installer en fonction des fonctionnalités que vous souhaitez utiliser :

- **Mode Local** : Pour exécuter des modèles de langage en local. `bash pip install open-interpreter[local]`
- **Mode OS** : Pour permettre à Open Interpreter de contrôler votre système d'exploitation. `bash pip install open-interpreter[os]`
- **Mode Sécurisé** : Pour des fonctionnalités de sécurité renforcées. `bash pip install open-interpreter[safe]`
- **Serveur** : Pour exposer Open Interpreter via un serveur HTTP. `bash pip install open-interpreter[server]`

Installateurs en une ligne (expérimental)

Pour une installation simplifiée, des installateurs expérimentaux sont disponibles pour les principaux systèmes d'exploitation. Ces scripts tentent de télécharger Python, de configurer un environnement et d'installer Open Interpreter automatiquement.

- **Mac** `bash curl -sL https://raw.githubusercontent.com/KillianLucas/open-interpreter/main/installers/oi-mac-installer.sh | bash`

Sans installation

Si vous ne souhaitez pas installer Open Interpreter sur votre machine, vous pouvez utiliser un environnement de développement en ligne. En vous rendant sur la page GitHub du projet et en appuyant sur la touche `,`, vous pouvez créer un "codespace", qui est une machine virtuelle dans le cloud avec Open Interpreter déjà préinstallé.

Utilisation de base

Une fois Open Interpreter installé, vous pouvez commencer à l'utiliser de différentes manières, que ce soit via un chat interactif dans votre terminal ou de manière programmatique dans vos scripts Python.

Chat interactif

Le moyen le plus simple d'interagir avec Open Interpreter est de lancer un chat interactif directement depuis votre terminal. Pour ce faire, exécutez simplement la commande suivante :

```
interpreter
```

Cela lancera une interface de type ChatGPT dans votre terminal, où vous pourrez commencer à donner des instructions en langage naturel. En Python, vous pouvez obtenir le même résultat avec :

```
from interpreter import interpreter

interpreter.chat()
```

Chat programmatique

Pour un contrôle plus fin, vous pouvez intégrer Open Interpreter dans vos propres scripts Python et lui passer des messages de manière programmatique. Cela vous permet d'automatiser des tâches complexes sans intervention manuelle.

Voici un exemple simple :

```
from interpreter import interpreter

# Envoyer une première instruction
interpreter.chat("Peux-tu créer un fichier texte nommé 'salutations.txt' avec
le contenu 'Bonjour, Open Interpreter!'?")

# Envoyer une instruction de suivi
interpreter.chat("Maintenant, lis le contenu du fichier 'salutations.txt' et
affiche-le.")
```

Démarrer une nouvelle conversation

Dans le terminal, chaque session de `interpreter` est indépendante et ne conserve pas l'historique des conversations précédentes. Pour démarrer une nouvelle conversation, il suffit de relancer la commande `interpreter`.

En Python, Open Interpreter conserve l'historique de la conversation en cours. Pour démarrer une nouvelle conversation, vous devez réinitialiser l'historique des messages :

```
interpreter.messages = []
```

Sauvegarder et restaurer les conversations

Dans le terminal, vous pouvez sauvegarder et restaurer vos conversations. Les conversations sont sauvegardées dans le répertoire `<votre répertoire d'application>/Open Interpreter/conversations/`. Pour reprendre une conversation, utilisez l'option `--conversations` :

```
interpreter --conversations
```

En Python, la méthode `interpreter.chat()` renvoie une liste de messages qui représente la conversation. Vous pouvez sauvegarder cette liste et la réutiliser plus tard pour reprendre la conversation :

```
# Sauvegarder la conversation
messages = interpreter.chat("Mon nom est Manus.")

# Réinitialiser l'interpréteur
interpreter.messages = []

# Restaurer la conversation
interpreter.messages = messages
```

Configuration et personnalisation

Open Interpreter offre plusieurs options pour personnaliser son comportement, notamment le choix du modèle de langage et la modification du message système.

Changer de modèle de langage

Par défaut, Open Interpreter utilise les modèles d'OpenAI, mais il est compatible avec de nombreux autres modèles de langage grâce à LiteLLM. Vous pouvez spécifier le modèle à utiliser directement depuis la ligne de commande :

```
interpreter --model gpt-3.5-turbo
interpreter --model claude-2
interpreter --model command-nightly
```

En Python, vous pouvez définir le modèle comme suit :

```
interpreter.llm.model = "gpt-3.5-turbo"
```

Personnaliser le message système

Le message système est un ensemble d'instructions qui guident le comportement du modèle de langage. Vous pouvez le personnaliser pour adapter Open Interpreter à vos besoins spécifiques. Par exemple, pour que l'interpréteur exécute les commandes sans demander de confirmation, vous pouvez ajouter l'instruction suivante :

```
interpreter.system_message += """
Exécute les commandes shell avec l'option -y pour que l'utilisateur n'ait pas à
les confirmer.
"""
```

Utiliser des profils de configuration

Pour éviter de répéter les mêmes configurations à chaque fois, vous pouvez utiliser des profils. Les profils sont des fichiers YAML où vous pouvez définir vos paramètres par défaut. Pour accéder au répertoire des profils, exécutez :

```
interpreter --profiles
```

Le profil par défaut est `default.yaml`. Vous pouvez créer d'autres profils et les utiliser avec l'option `--profile`.

Exécution locale

L'un des principaux avantages d'Open Interpreter est sa capacité à fonctionner entièrement en local, en utilisant des modèles de langage que vous hébergez vous-même. Cela vous donne un contrôle total sur vos données et vous permet de travailler sans connexion internet.

Fournisseurs de modèles locaux

Open Interpreter prend en charge plusieurs fournisseurs de modèles locaux, notamment :

- **Ollama**
- **Llamafile**
- **Jan**
- **LM Studio**

Utilisation en ligne de commande

Pour utiliser un modèle local, vous pouvez utiliser l'explorateur local intégré :

```
interpreter --local
```

Cela vous guidera dans le choix d'un fournisseur et d'un modèle. Vous pouvez également spécifier manuellement l'URL de votre serveur de modèles et le modèle à utiliser :

```
interpreter --api_base "http://localhost:11434" --model ollama/codestral
```

Utilisation en Python

Pour utiliser un modèle local dans un script Python, vous devez configurer les paramètres `offline`, `llm.model`, et `llm.api_base` :

```
from interpreter import interpreter

interpreter.offline = True
interpreter.llm.model = "ollama/codestral"
interpreter.llm.api_base = "http://localhost:11434"

interpreter.chat("Combien de fichiers se trouvent sur mon bureau ?")
```

Mode OS (expérimental)

Le mode OS est une fonctionnalité expérimentale qui permet à Open Interpreter de contrôler votre ordinateur de manière visuelle, en utilisant la souris et le clavier. Il utilise un modèle de langage multimodal (comme GPT-4V) pour voir l'écran, comprendre les éléments de l'interface graphique, et interagir avec eux.

Pour activer le mode OS, utilisez l'option `--os` :

```
interpreter --os
```

Remarque : Cette fonctionnalité est encore en développement et nécessite des autorisations d'enregistrement d'écran pour votre terminal.

Cas d'utilisation et exemples pratiques

Open Interpreter est un outil extrêmement polyvalent qui peut être utilisé pour une multitude de tâches. Voici quelques exemples concrets pour vous inspirer.

Automatisation de tâches de développement

Imaginez que vous travaillez sur un projet de développement web et que vous devez effectuer plusieurs tâches répétitives. Vous pouvez demander à Open Interpreter de les automatiser pour vous.

Exemple : Créer un nouveau composant React

```
interpreter
> Crée un nouveau composant React nommé 'Button' dans le dossier
'src/components'. Le composant doit être un bouton simple avec un texte
personnalisable.
```

Open Interpreter analysera votre demande, créera le fichier `Button.js`, et y ajoutera le code de base pour un composant React, le tout sans que vous ayez à écrire une seule ligne de code.

Analyse de données

Vous avez un fichier CSV contenant des données de ventes et vous souhaitez en extraire des informations clés. Open Interpreter peut vous aider à analyser ces données et même à créer des visualisations.

Exemple : Analyser un fichier CSV

```
 interpreter
> J'ai un fichier 'ventes.csv' avec les colonnes 'Date', 'Produit', et
'Montant'. Peux-tu calculer le total des ventes pour chaque produit et créer un
graphique à barres pour visualiser les résultats ?
```

Open Interpreter lira le fichier CSV, effectuera les calculs nécessaires en utilisant des bibliothèques comme Pandas, et générera un graphique avec Matplotlib, qu'il enregistrera en tant qu'image sur votre ordinateur.

Manipulation de fichiers multimédia

Vous avez besoin de redimensionner plusieurs images pour votre site web. Au lieu de le faire manuellement, vous pouvez demander à Open Interpreter de s'en charger.

Exemple : Redimensionner des images

```
 interpreter
> Redimensionne toutes les images du dossier 'images-originales' à une largeur
de 800 pixels et enregistre-les dans le dossier 'images-redimensionnees'.
```

Open Interpreter utilisera une bibliothèque de traitement d'images comme Pillow pour effectuer cette tâche rapidement et efficacement.

Exemple de script Python complet

Voici un exemple de script Python qui illustre comment utiliser Open Interpreter de manière programmatique pour différentes tâches. Vous pouvez l'adapter et l'étendre pour créer vos propres automatisations personnalisées.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Exemple d'utilisation d'Open Interpreter en Python
Ce script montre comment utiliser Open Interpreter pour diverses tâches
"""

from interpreter import interpreter

# Configuration de base
def configuration_basique():
    """Configuration de base d'Open Interpreter"""
    print("Configuration de base d'Open Interpreter")

    # Définir le modèle à utiliser
    interpreter.llm.model = "gpt-3.5-turbo"

    # Personnaliser le message système
    interpreter.system_message += """
    Exécuter les commandes shell avec -y pour que l'utilisateur n'ait pas à les confirmer.
    Toujours expliquer le code avant de l'exécuter.
    """

    # Afficher le message système actuel
    print(f"Message système actuel: {interpreter.system_message}")

# Utilisation programmatique
def mode_programmatique():
    """Exemples d'utilisation programmatique"""
    print("\nExemples d'utilisation programmatique:")

    # Exemple 1: Analyse de fichiers
    print("\nExemple 1: Analyse de fichiers")
    response = interpreter.chat("Liste tous les fichiers Python dans le répertoire courant et affiche leur nombre de lignes")

    # Exemple 2: Création d'un graphique
    print("\nExemple 2: Création d'un graphique")
    response = interpreter.chat("""
    Crée un graphique simple avec matplotlib:
    1. Génère 100 points aléatoires
    2. Trace un graphique de dispersion
    3. Ajoute un titre et des étiquettes aux axes
    4. Sauvegarde le graphique en tant que 'graphique_exemple.png'
    """)

    # Exemple 3: Traitement de texte
    print("\nExemple 3: Traitement de texte")
    response = interpreter.chat("""
    Écris un script qui:
    1. Crée un fichier texte nommé 'exemple.txt' avec quelques paragraphes de texte
    2. Lit ce fichier
    3. Compte le nombre de mots et de phrases
    4. Affiche les statistiques
    """)

    return response

# Fonction principale
def main():
    """Fonction principale"""
    print("EXEMPLES D'UTILISATION D'OPEN INTERPRETER")
```

```
    print("========================================\n")

    configuration_basique()
    mode_programmatique()

    print("\nExemples terminés!")

if __name__ == "__main__":
    main()
```

Conclusion

Open Interpreter représente une avancée significative dans la manière dont nous interagissons avec nos ordinateurs. En combinant la puissance des modèles de langage avec la flexibilité d'un environnement local, il ouvre un monde de possibilités pour l'automatisation de tâches, l'analyse de données, et bien plus encore. Que vous soyez un développeur chevronné ou un novice curieux, Open Interpreter vous donne les outils pour transformer vos idées en actions, en utilisant simplement le langage naturel.

Alors que l'outil continue d'évoluer, avec des fonctionnalités expérimentales comme le mode OS, il est clair que nous ne sommes qu'au début de ce que l'on peut accomplir avec ce type de technologie. Nous vous encourageons à explorer, à expérimenter, et à découvrir comment Open Interpreter peut simplifier et améliorer votre flux de travail quotidien.

Références

1. Documentation Officielle d'Open Interpreter
2. Dépôt GitHub d'Open Interpreter
3. Open Interpreter Use Cases - Mervin Praison

Fonctionnalités avancées

Open Interpreter offre également des fonctionnalités avancées qui peuvent être utiles pour les utilisateurs plus expérimentés ou pour des cas d'utilisation spécifiques.

Mode verbeux

Pour mieux comprendre ce qui se passe sous le capot, vous pouvez activer le mode verbeux. Ce mode affiche des informations détaillées sur les échanges entre Open Interpreter et le modèle de langage, ce qui peut être utile pour le débogage ou pour comprendre comment les décisions sont prises.

Vous pouvez activer le mode verbeux en utilisant l'option `--verbose` lors du lancement d'Open Interpreter :

```
interpreter --verbose
```

Ou pendant une session interactive :

```
> %verbose true
```

Commandes en mode interactif

En mode interactif, Open Interpreter propose plusieurs commandes spéciales qui commencent par le caractère `%`. Voici les principales commandes disponibles :

- `%verbose [true/false]` : Active ou désactive le mode verbeux.
- `%reset` : Réinitialise la conversation en cours.
- `%undo` : Supprime le dernier message de l'utilisateur et la réponse de l'IA de l'historique.
- `%tokens [prompt]` : Calcule le nombre de tokens qui seront envoyés avec le prochain prompt et estime leur coût.
- `%help` : Affiche la liste des commandes disponibles.

Streaming des réponses

Pour les applications qui nécessitent un affichage en temps réel des réponses, Open Interpreter permet de streamer les résultats. Cela signifie que vous pouvez recevoir et traiter les morceaux de réponse au fur et à mesure qu'ils sont générés, plutôt que d'attendre la réponse complète.

```
message = "Quel système d'exploitation utilisons-nous?"

for chunk in interpreter.chat(message, display=False, stream=True):
  print(chunk)
```

Serveur FastAPI

Pour les développeurs qui souhaitent intégrer Open Interpreter dans une application web, il est possible de l'exposer via un serveur HTTP en utilisant FastAPI. Voici un exemple simple de serveur :

```
from fastapi import FastAPI
from fastapi.responses import StreamingResponse
from interpreter import interpreter

app = FastAPI()

@app.get("/chat")
def chat_endpoint(message: str):
    def event_stream():
        for result in interpreter.chat(message, stream=True):
            yield f"data: {result}\n\n"

    return StreamingResponse(event_stream(), media_type="text/event-stream")

@app.get("/history")
def history_endpoint():
    return interpreter.messages
```

Pour lancer ce serveur, vous pouvez utiliser uvicorn :

```
pip install fastapi uvicorn
uvicorn server:app --reload
```

Vous pouvez également démarrer un serveur similaire directement en utilisant la méthode `interpreter.server()`.