MOJO

The programming language for all AI developers

Mojo est un nouveau langage de programmation qui comble le fossé entre la recherche et la production en combinant le meilleur de la syntaxe Python avec la programmation système et la métaprogrammation. Avec Mojo, vous pouvez écrire du code portable plus rapide que le C et interopérable de manière transparente avec l'écosystème Python.

Un langage pour la technologie de compilateur de nouvelle génération

pourquoi
Lorsque nous avons réalisé qu'aucun langage existant ne pouvait résoudre les défis liés au calcul de l'IA, nous nous sommes lancés dans une refonte des principes fondamentaux de la façon dont un langage de programmation devrait être conçu et mis en œuvre pour résoudre nos problèmes. Parce que nous avons besoin d'une prise en charge hautes performances pour une grande variété d'accélérateurs, les technologies de compilateur traditionnelles telles que LLVM et GCC n'étaient pas adaptées (et les langages et outils basés sur celles-ci ne suffiraient pas). Bien qu'elles prennent en charge une large gamme de processeurs et certains GPU couramment utilisés, ces technologies de compilateur ont été conçues il y a des décennies et ne sont pas en mesure de prendre pleinement en charge les architectures de puces modernes. De nos jours, la technologie standard pour les accélérateurs spécialisés d'apprentissage automatique est le MLIR.

MLIR est une infrastructure de compilateur open source relativement nouvelle lancée chez Google (dont les dirigeants sont passés à Modular) et qui a été largement adoptée par la communauté des accélérateurs d'apprentissage automatique. La force de MLIR réside dans sa capacité à créer des compilateurs *spécifiques à un domaine* , en particulier pour des domaines étranges qui ne sont pas des CPU et des GPU traditionnels, tels que les ASICS AI, les systèmes informatiques quantiques , les FPGA et le silicium personnalisé .MLIR est presenté enannexe

Caractéristiques de Mojo par rapport à d'autres langages de programmation

Mojo est devenu incroyablement populaire même s'il n'est pas encore disponible publiquement. En effet, il présente plusieurs avantages significatifs par rapport à d'autres langages de programmation pour l'apprentissage automatique et la construction de logiciels de niveau système. Dans cette section, nous allons discuter de ces avantages.

#1. Support natif pour les tâches d'IA et de Machine Learning

Mojo est conçu pour développer des applications d'intelligence artificielle. Par conséquent, il est livré avec des fonctions et des modules dans la bibliothèque standard pour construire des réseaux neuronaux, effectuer de la vision par ordinateur et préparer des données.

La plupart des langages généralistes comme Python nécessitent des bibliothèques supplémentaires pour réaliser ces tâches, mais Mojo les prend en charge dès la sortie de la boîte.

#2. Syntaxe simplifiée et abstractions de haut niveau

Pour écrire des logiciels rapides et efficaces, nous devrions normalement utiliser des langages comme C, C , et Rust. Bien que ces langages soient rapides, ils sont plus difficiles à apprendre et à utiliser. En effet, ils vous obligent à travailler à un bas niveau, ce qui vous permet d'avoir plus de contrôle.

Cependant, Mojo fournit toujours des abstractions de plus haut niveau comme Python et une syntaxe simple. Cela le rend plus facile à utiliser que d'autres langages comparables en performance.

#3. Intégration avec les frameworks et bibliothèques d'IA les plus populaires

Comme mentionné précédemment, Mojo n'est pas un langage complètement nouveau – c'est un surensemble de Python. Par conséquent, il s'intègre bien avec les bibliothèques existantes telles que NumPy et PyTorch. Cela signifie que, par défaut, Mojo a un écosystème aussi grand que celui de Python.

#4. Des capacités de manipulation de données efficaces

Mojo est conçu pour manipuler efficacement plusieurs valeurs en parallèle. Ceci est particulièrement utile pour l'algèbre linéaire, sur laquelle l'apprentissage automatique s'appuie fortement. Mojo est également compilé en flux tendu, de sorte que le bytecode est optimisé pour la vitesse. Cela rend le travail avec les données et l'apprentissage automatique efficace dans Mojo.

#5. Évolutivité et support du calcul parallèle

Comme mentionné précédemment, Mojo est conçu pour supporter le paradigme de l'instruction unique et des données multiples de l'informatique parallèle. Cela est intégré dans Mojo et le rend plus rapide dès sa sortie de l'emballage. Il est également plus performant que les bibliothèques Python telles que NumPy.

1 - Principales caractéristiques de Mojo

- **Rapide :** Mojo se veut rapide et efficace. Pour améliorer les performances, il utilise diverses stratégies telles que la mise en cache, le chargement paresseux et des structures de données efficaces.
- **Léger :** Mojo est un petit framework. Son fonctionnement nécessite peu de ressources, ce qui le rend approprié pour les sites Web de petite et moyenne taille.
- **Facile à utiliser :** Mojo est simple à apprendre et à utiliser. Il propose une syntaxe facile à comprendre et une API bien définie.
- **RESTful :** Mojo simplifie la création de services Web RESTful. Il comprend un mécanisme de routage qui connecte les URL aux actions du contrôleur.
- **Système de modèles :** Mojo propose un excellent système de modèles qui facilite la création de pages Web dynamiques. Le système de modèles est construit sur Template Toolkit, un langage de modèles Perl populaire.

1 – 1 - Eléments clés de Mojo

Dans cette section, nous allons voir comment écrire des programmes dans Mojo. Mojo étant conçu comme un surensemble de Python, tout comme **TypeScript** est un surensemble de JavaScript, tout code Python valide est un code Mojo valide, mais tout code Mojo n'est pas un code Python valide.

Mojo est encore un travail en cours, et certaines caractéristiques du langage Python ne sont pas encore prises en charge, par exemple les classes. En outre, un compilateur n'est pas encore disponible, mais vous pouvez utiliser Mojo dans un cahier de jeux. Cependant, vous aurez besoin d'un compte au préalable, que vous pouvez créer sur leur site web.

Pour l'instant, il est difficile de donner un tutoriel complet sur le langage car certaines fonctionnalités doivent encore être ajoutées, et tout n'est pas encore supporté. A la place, nous allons discuter de quelques ajouts clés que Mojo ajoute à ce que Python a déjà.

1 – 2 - Syntaxe et grammaire

Mojo étant un surensemble de Python, leurs syntaxes sont identiques. Comme Python, un programme est constitué d'instructions. Ces instructions peuvent être regroupées en blocs sous forme de fonctions, de boucles ou de conditionnelles. Les instructions à l'intérieur d'un bloc sont indentées. Voici un exemple de programme écrit en Mojo :

```
defd odd_or_even() :
 for i in range(1, 101) :
 if i % 2 == 0 :
 print("Even")
 else :
 print("Odd")

odd_or_even()
```

Ce programme est parfaitement identique à un programme Python. Cependant, Mojo offre des fonctionnalités supplémentaires que vous verrez dans les sections suivantes.

1 – 3 -Déclarations de variables

Avec Mojo, vous disposez de deux façons supplémentaires de déclarer des variables. Vous pouvez utiliser le mot-clé `let` ou `var`. Le mot clé `let` déclare une variable immuable. Une fois initialisée, vous ne pouvez pas réaffecter sa valeur à autre chose. En revanche, les variables déclarées à l'aide du mot-clé `var` peuvent être réaffectées car elles sont mutables.

Le principal avantage des variables déclarées à l'aide de `let` ou de `var` est qu'elles prennent en charge les spécificateurs de type. L'exemple suivant illustre la manière dont les variables sont déclarées dans Mojo.

```
let pi : Float64 = 3.141
var greeting = "Hello, World"

# Ce serait impossible
# pi = 6.283

# Mais c'est possible
greeting = "Ola"
```

```
print(pi, greeting)
```

1- 4 - Structures

En plus d'une manière différente de déclarer les variables, Mojo supporte les structures. Une façon simple de voir les structs est qu'ils sont comme des classes, mais plus rigides. Contrairement aux classes, vous ne pouvez pas ajouter/supprimer ou modifier des méthodes en cours d'exécution, et tous les membres doivent être déclarés à l'aide des mots-clés `var` ou `let`. Cette structure plus rigide permet à Mojo de gérer la mémoire et les performances plus efficacement. Voici un exemple de structure :

```
struct Person :
 var name : StringLiteral
 var age : Int32

 fn __init__(inout self, name : StringLiteral, age : Int32) :
 self.name = name
 self.age = age


john = Person("John Doe", 32)
print(john.name, john.age)
```

1 – 5- Fonctions

Dans la structure ci-dessus, vous avez peut-être remarqué que nous avons déclaré la méthode `__init__` en utilisant le mot-clé fn au lieu de def. C'est parce que, dans Mojo, vous pouvez déclarer des fonctions en utilisant `fn` et `def`. Une fonction déclarée en utilisant `fn` est plus stricte par rapport à son homologue `def`.
En particulier, une fonction déclarée à l'aide de `fn` a ses arguments immuables par défaut. En outre, vous devez spécifier le type de données des arguments et la valeur de retour de la fonction. Toutes les variables locales doivent être déclarées avant d'être utilisées.

```
fn say_hello(name : StringLiteral) :
 print("Hello,", name)

# Ceci serait invalide
# fn say_hello(name) :
# print("Hello,", name)

say_hello("John")
```

Si la fonction lève une exception, cela doit être explicitement indiqué lors de la déclaration de la fonction à l'aide du mot-clé `raises`. De plus, Mojo n'utilise pas la classe Exception comme Python, mais la classe Error.

```
fn will_raise_error() raises :
 raise Error('Some error')

will_raise_error(
```

1 – 6 - Surcharge

Mojo supporte également la surcharge des opérateurs basés sur différents types de données. Ceci supporte le principe de polymorphisme orienté objet.

```
fn add_numbers(a : Int32, b : Int32) -> Int32 :
 return a b

fn add_numbers(a : Int32, b : Int32, c : Int32) -> Int32 :
 return a b c

let first_total = add_numbers(2, 3)
let second_total = add_numbers(1, 2, 3)

print(first_total, second_total)
```

2 - Comment Mojo est utilisé dans le développement de l'IA

Mojo est livré avec des bibliothèques permettant de construire des modèles d'apprentissage automatique. Il s'agit notamment de bibliothèques permettant de construire des réseaux neuronaux. En outre, vous pouvez également effectuer des tâches telles que le traitement du langage naturel et la vision par ordinateur.
Bien que le langage lui-même ne soit pas encore achevé et que son écosystème soit pratiquement inexistant, nous pouvons nous attendre à ce que Mojo apporte de nombreuses fonctionnalités permettant d'effectuer des tâches telles que le traitement des données, la création de modèles, l'optimisation, la gestion des modèles et la surveillance.

3 - Mojo est-il l'avenir du développement de l'IA ?

Il est difficile de prédire comment une technologie est susceptible d'évoluer et d'être adoptée. La plupart des prédictions sont fausses, mais cela ne signifie pas que nous ne pouvons pas essayer. Pour savoir si Mojo est susceptible de remplacer Python, examinons les avantages et les inconvénients/limites de Mojo :

Avantages

Selon ses créateurs, Mojo est 35000X plus rapide que Python. Un autre avantage est la prise en charge native de multiples composants hardware comme les CPU, GPU, TPU et ASICs custom.

La parallélisation automatique sur de multiples composants simplifie également l'écriture de code efficace et parallèle.

Ce langage se caractérise aussi par une syntaxe de haut niveau et une sémantique comparables à Python. C'est un avantage pour les développeurs maîtrisant déjà ce langage.

On retrouve aussi un système de vérification et d'inférence permettant de détecter les erreurs de compile-time et réduisant les risques d'erreurs runtime.

Enfin, la compilation est statique permettant un temps d'exécution plus rapide et une meilleure optimisation. Le code est compilé avant l'exécution.

- Mojo est très rapide et conçu pour tirer parti du parallélisme sans en faire beaucoup, ce qui est essentiel pour l'apprentissage automatique, car l'apprentissage des modèles peut prendre beaucoup de temps.
- C'est un surensemble de Python, donc plus facile à apprendre et avec une courbe d'apprentissage plus douce. Cela réduit la friction pour l'adoption.
- Il réduit les risques d'erreurs en production, car les erreurs telles que les noms de variables mal saisis ou les incompatibilités de type sont détectées au moment de la compilation. C'est pourquoi il est préférable de l'utiliser.

Inconvénients

- Il est actuellement incomplet. Mais bien sûr, l'équipe de Modular travaille à la publication du langage et de son traducteur.
- Bien qu'il simplifie le travail des producteurs de frameworks, il n'apporte pas beaucoup d'avantages aux consommateurs de frameworks puisqu'ils utilisent déjà des frameworks d'apprentissage automatique en Python.
- Ce langage ne dispose pas encore d'un vaste écosystème d'outils et de ressources d'apprentissage. Si vous pouvez utiliser les bibliothèques de Python dans Mojo, vous pouvez toujours utiliser les bibliothèques de Python dans Python. Pour que Mojo ait un avantage sur Python, il a besoin de bibliothèques qui portent la vitesse de Mojo.

3 – 1 - Quel est le but de Mojo ?

- Le concept fondamental de Mojo est d'unifier l'infrastructure ML/AI en fournissant un langage de programmation qui fonctionne sur toute la pile. De plus, en éliminant la nécessité de créer du code MLIR, il facilite son utilisation.
- Modular affirme que Mojo fournira un modèle de programmation évolutif et unique. Les utilisateurs du domaine de l'IA auront ainsi plus de facilité à gérer les accélérateurs et les systèmes hétérogènes.
- Mojo est désormais un langage de programmation qui prend en charge la métaprogrammation au moment de la compilation. Les autres fonctionnalités prises en charge incluent la mise en cache pendant le flux de compilation, les approches de compilation adaptative, etc. Les autres langages de programmation n'ont pas ces fonctionnalités.

3 – 2 Comparaison de Mojo et Python

Python est un langage de programmation généraliste de haut niveau. Sa philosophie de conception donne la priorité à la lisibilité du code, comme en témoigne son utilisation intensive des espaces. Ses éléments de langage et son approche orientée objet sont destinés à aider les programmeurs à écrire du code clair et logique pour des projets à petite et à grande échelle.

3 – 2 – 1 - Les problèmes de Python

Nous espérons qu'en essayant de faire de Mojo un sur-ensemble de Python, nous serons en mesure de résoudre bon nombre des difficultés actuelles de Python.
Python présente divers problèmes bien connus, notamment de mauvaises performances de bas niveau et des spécificités d'implémentation de CPython telles que le verrouillage global de l'interpréteur (GIL), qui fait que Python est monothread. Bien que de nombreux efforts soient en cours pour résoudre ces difficultés, les contributions de Python vont plus loin et ont une plus grande influence dans le secteur de l'IA. Au lieu d'approfondir ces contraintes techniques, nous discuterons des implications en 2023.

Fonctionnalité	Mojo	Python
Vitesse	Plus rapide	Ralentissez
Facilité d'utilisation	Courbe d'apprentissage plus abrupte	Plus facile à apprendre
Soutien communautaire	Communauté plus petite	Communauté plus grande
Fonctionnalité	Moins de bibliothèques tierces	Plus de bibliothèques tierces
Conçu pour	Apprentissage automatique et intelligence artificielle	Usage général
Meilleur pour	Petits projets qui nécessitent des performances élevées, un nouveau langage à apprendre, l'apprentissage automatique et l'intelligence artificielle	Grands projets ne nécessitant pas de hautes performances, équipe existante familiarisée avec Python, programmation généraliste

Comparaison de Mojo et Python

Enfin, le choix entre Mojo et Python sera déterminé par vos besoins individuels. Mojo peut être une alternative appropriée si vous avez besoin d'un langage de programmation rapide avec une courte courbe d'apprentissage. Python peut être une meilleure alternative si vous avez besoin d'un langage de programmation à usage général avec une communauté forte et un large éventail de bibliothèques tierces.

3 – 2 – 2 - Autres facteurs pour choisir Mojo vs Python

Taille du projet : si vous travaillez sur un petit projet, la différence de vitesse entre Mojo et Python peut ne pas être perceptible. Mojo, en revanche, peut être une meilleure alternative si vous travaillez sur un projet de grande envergure qui exige d'excellentes performances.
Expérience d'équipe : si votre équipe est déjà à l'aise avec Python, il peut être plus facile de s'en tenir à ce langage. Mojo, en revanche, peut être une excellente solution si votre personnel cherche à apprendre une nouvelle langue.

Projets pour l'avenir : Mojo serait une meilleure alternative si vous avez l'intention d'utiliser le langage pour l'apprentissage automatique ou l'intelligence artificielle. Python est également une excellente alternative pour ces applications, mais Mojo a été créé spécialement pour elles
Compte tenu de notre objectif chez Modular de créer une plate-forme d'IA de nouvelle génération, nous utilisions déjà MLIR pour certaines de nos infrastructures, mais nous ne disposions pas d'un langage de programmation capable de libérer tout le potentiel de MLIR sur notre pile. Alors que de nombreux autres projets utilisent désormais MLIR, Mojo est le premier langage majeur conçu expressément *pour MLIR* , ce qui rend Mojo particulièrement puissant lors de l'écriture de code au niveau système pour les charges de travail d'IA.

3 – 2 - 3 - un membre de la famille Python

Notre mission principale pour Mojo comprend des innovations dans les composants internes du compilateur et la prise en charge des accélérateurs actuels et émergents, mais nous ne voyons pas la nécessité d'innover dans la *syntaxe* du langage ou dans *la communauté* . Nous avons donc choisi d'adopter l'écosystème Python parce qu'il est très largement utilisé, qu'il est apprécié par l'écosystème de l'IA et parce que nous pensons que c'est un langage vraiment sympa.

Le langage Mojo a de nobles objectifs : nous voulons une compatibilité totale avec l'écosystème Python, nous voulons des performances de bas niveau prévisibles et un contrôle de bas niveau, et nous avons besoin de pouvoir déployer des sous-ensembles de code sur des accélérateurs. De plus, nous ne voulons pas créer un écosystème logiciel fragmenté : nous ne voulons pas que les utilisateurs de Python qui adoptent Mojo fassent des comparaisons avec la douloureuse migration de Python 2 vers 3. Ce ne sont pas de petits objectifs !

Pourquoi nous avons choisi Python (modular)

Python est la force dominante du ML et d'innombrables autres domaines. Il est facile à apprendre, connu par d'importantes cohortes de programmeurs, possède une communauté incroyable, propose des tonnes de packages précieux et dispose d'une grande variété de bons outils. Python prend en charge le développement d'API belles et expressives grâce à ses fonctionnalités de programmation dynamique, ce qui a conduit les frameworks d'apprentissage automatique tels que TensorFlow et PyTorch à adopter Python comme interface pour leurs environnements d'exécution hautes performances implémentés en C++.

Pour Modular aujourd'hui, Python est une partie non négociable de notre pile de surface API : elle est dictée par nos clients. Étant donné que tout le reste de notre pile est négociable, il va de soi que nous devrions partir d’une approche « Python d’abord ».

Compatibilité avec Python

Nous prévoyons une compatibilité totale avec l'écosystème Python, mais il existe en réalité deux types de compatibilité, voici donc où nous en sommes actuellement :

- En termes de capacité à importer des modules Python existants et à les utiliser dans un programme Mojo, Mojo est 100 % compatible car nous utilisons CPython pour l'interopérabilité.
- En ce qui concerne votre capacité à migrer n'importe quel code Python vers Mojo, il n'est pas encore entièrement compatible. Mojo prend déjà en charge de nombreuses fonctionnalités principales de Python, notamment l'async/wait, la gestion des erreurs, la variadique, etc. Cependant, Mojo est encore jeune et manque de nombreuses autres fonctionnalités de Python. Mojo ne prend même pas encore en charge les cours !
- Le parcours vers le compilateur Clang (un compilateur pour C, C++, Objective-C, CUDA, OpenCL et autres), qui est un « remplacement compatible » pour GCC, MSVC et autres compilateurs existants. Il est difficile de faire une comparaison directe, mais la complexité du problème Clang semble être d'un ordre de grandeur supérieur à celui de la mise en œuvre d'un remplacement compatible pour Python.
- Le voyage vers le langage de programmation Swift , qui a embrassé l'écosystème d'exécution et de langage Objective-C, et a progressivement migré des millions de programmeurs (et d'énormes quantités de code). Avec Swift, nous avons appris comment être « compatible avec le runtime » et coopérer avec un runtime existan

4 – Disponibilité

Le SDK Mojo est encore en début de développement et n’est actuellement disponible que pour les systèmes Linux Ubuntu. Voici quelques-uns des problèmes notables que nous prévoyons de résoudre :

- Prise en charge native manquante pour Windows, macOS et autres distributions Linux. Actuellement, nous prenons en charge les systèmes Ubuntu avec des processeurs x86-64 uniquement. La prise en charge d’un plus grand nombre de distributions Linux (y compris Debian et RHEL), Windows et Mac est en cours.
- L’interopérabilité Python peut échouer lors de l’exécution d’un programme Mojo compilé, avec le message . En effet, nous n’intégrons actuellement pas la version Python dans le binaire Mojo. Pour plus de détails et la solution de contournement, consultez le numéro #551.Unable to locate a suitable libpython, please set MOJO_PYTHON_LIBRARY
- L’installation de l’interface de ligne de commande modulaire peut échouer et nécessiter avant la réinstallation.modular clean

  S’il vous demande d’effectuer une authentification, exécutez et utilisez la valeur affichée pour la commande sur la page de téléchargement.modular auth <MODULAR_AUTH>MODULAR_AUTHcurl

- modular install mojo est lent et peut sembler ne pas répondre (car le programme d'installation télécharge des packages en arrière-plan). Nous ajouterons une barre de progression dans une prochaine version.
- Si vous tentez de désinstaller Mojo avec , votre tentative ultérieure d'installation de Mojo peut échouer avec un code d'erreur HTTP 500. Si c'est le cas, exécutez et réessayez.modular uninstallmodular clean
- Mojo REPL peut se bloquer (cesser de répondre pendant plus de 10 secondes) lors de l'interprétation d'une expression si votre système dispose de 4 Gio ou moins de RAM. Si vous rencontrez ce problème, veuillez le signaler avec les spécifications de votre système.

Annexe 1 : bibliographie

- https://www.modular.com/mojo
- https://docs.modular.com/mojo/programming-manual.html
- https://docs.modular.com/mojo/manual/get-started/
- https://github.com/modularml/mojo
- https://docs.modular.com/mojo/programming-manual.html
- Docs modulaires - Feuille de route Mojo🔥 et arêtes vives (modular.com)

Annexe 2 : Projet MLIR

Le projet MLIR (Multi-Level Intermediate Representation) est un sous-projet de TensorFlow, un framework d'apprentissage machine de Google. Il s'agit d'un langage intermédiaire commun pour les frameworks d'apprentissage machine qui permet une compilation plus rapide et une optimisation matérielle plus facile des modèles d'apprentissage machine haute performance. Le langage MLIR ne ressemble pas au C++ ou à Python : il introduit une étape de compilation intermédiaire entre ces langages de niveau supérieur et le code machine. MLIR permettra de compiler les projets utilisant TensorFlow et d'autres bibliothèques d'apprentissage machine dans un code plus efficace capable de tirer le meilleur parti du matériel sous-jacent. De plus, à terme, MLIR pourrait être utilisé par tous les compilateurs et faire profiter d'autres projets de ses avantages d'optimisation, en dehors des seuls projets

Avantage

Le projet MLIR d'apprentissage machine ,est un langage intermédiaire commun pour les Framework d'apprentissage machine qui permet une compilation plus rapide et une optimisation matérielle plus facile des modèles d'apprentissage machine haute performance 1. Les avantages de MLIR sont les suivants:

- **Optimisation matérielle plus facile**: MLIR permet une optimisation matérielle plus facile des modèles d'apprentissage machine haute performance 1.
- **Compilation plus rapide**: MLIR permet une compilation plus rapide des projets utilisant TensorFlow et d'autres bibliothèques d'apprentissage machine dans un code plus efficace capable de tirer le meilleur parti du matériel sous-jacent 1.
- **Langage intermédiaire commun**: MLIR est un langage intermédiaire commun pour les frameworks d'apprentissage machine, ce qui permet une collaboration plus facile entre les différents projets

Pour utiliser MLIR dans vos projets d'apprentissage machine, vous pouvez suivre les instructions de démarrage rapide fournies par le site officiel de MLIR [1](mlir.livm.org Les instructions vous guideront à travers la compilation et les tests de MLIR. Vous pouvez également consulter le tutoriel sur la création d'un compilateur pour un langage jouet (mlir[2]. Ce tutoriel vous aidera à comprendre les concepts de MLIR et comment les dialectes peuvent aider à prendre en charge facilement des constructions et des transformations spécifiques au langage tout en offrant un chemin facile pour abaisser LLVM ou une autre infrastructure de codegen

Le projet MLIR prend en charge plusieurs dialectes, qui sont des extensions de la syntaxe de base de MLIR.

Ces dialectes permettent de prendre en charge facilement des constructions et des transformations spécifiques au langage tout en offrant un chemin facile pour abaisser LLVM ou une autre infrastructure de codegen .

Voici quelques-uns des dialectes les plus couramment utilisés:

- **Dialecte affine**: Il fournit une représentation pour les expressions affines et les boucles imbriquées [1].

- **Dialecte linalg**: Il fournit une représentation pour les opérations linéaires algébriques [1].

- **Dialecte LLVM**: Il fournit une représentation pour les opérations LLVM IR
- Dialecte GPU
- Dialecte vectoriel

Les dialectes sont des extensions de la syntaxe de base de MLIR qui permettent de prendre en charge facilement des constructions et des transformations spécifiques au langage tout en offrant un chemin facile pour abaisser LLVM ou une autre infrastructure de codegen [1]. Les avantages de l'utilisation des dialectes sont les suivants:

1. **Facilité d'abstraction**: Les dialectes permettent une abstraction plus facile des constructions et des transformations spécifiques au langage [1].
2. **Facilité d'optimisation**: Les dialectes permettent une optimisation plus facile des modèles d'apprentissage machine haute performance [1].
3. **Facilité de collaboration**: Les dialectes permettent une collaboration plus facile entre les différents projets [1].