كلية العلوم و التقنيات
FACULTÉ DES SCIENCES ET TECHNIQUES

Recherche Scientifique

Guide pour les Doctorants A1&A2 du Dep. Informatique

FD. Python pour le Calcul Scientifique (2024-2025)

Guide pratique - Version bêta

Dr. EL BENANY Mohamed Mahmoud

23 mai 2025

Table des matières

Chapitre 1

Introduction : Python dans la Recherche Scientifique

Ce chapitre introduit Python, un langage incontournable dans la recherche scientifique, utilisé dans des institutions comme MIT, Stanford, Oxford, et ETH Zurich pour des applications en physique, biologie computationnelle, climatologie, et intelligence artificielle (IA). Destiné aux doctorants du department infrmatique de la FST de l'UNA, il explore pourquoi Python (versions 3.11 à 3.13) domine le calcul scientifique et compare son usage à d'autres langages dans un contexte académique global.

1.1 C'est quoi Python ?

Python est un langage interprété, open-source, et multi-paradigme, largement adopté dans les grandes universités et laboratoires pour sa simplicité et son écosystème. Il est utilisé dans des projets comme la simulation de particules au CERN, l'analyse génomique, ou la modélisation climatique.

- **Langage de haut niveau** : Syntaxe claire, accélérant le prototypage dans les laboratoires.
- **Polyvalence** : Applications en IA, physique computationnelle, et analyse de données.
- **Open-source** : Accessible à tous, favorisant la collaboration internationale.
- **Nouvelles fonctionnalités** : Python 3.11+ offre des performances optimisées et le pattern matching.

```python
# Exemple : Calcul de la moyenne d une s rie de donn es exp rimentales
from typing import List

def calcul_moyenne(donnees: List[float]) -> float:
    """Calcule la moyenne d une s rie de donn es."""
    return sum(donnees) / len(donnees)

mesures = [10.5, 11.2, 10.8]  # Donn es exp rimentales (ex. : temp ratures)
```

```
print(f"Moyenne : {calcul_moyenne(mesures)}")
```

1.2 Pourquoi Python pour la recherche scientifique ?

Python est le choix privilégié dans les grandes universités grâce à son écosystème riche et sa flexibilité, utilisé dans des domaines comme l'astrophysique (NASA), la bioinformatique (Broad Institute), et l'IA (Google Research).

- **Écosystème scientifique** : Bibliothèques comme NumPy, Polars, PyTorch, et Plotly, utilisées à MIT pour les simulations pour l'analyse de données.
- **Interopérabilité** : Intégration avec C++, Fortran, et outils cloud (Google Colab, AWS).
- **Outils interactifs** : Notebooks Jupyter pour les publications scientifiques (par exemple, Nature).
- **Communauté académique** : Ressources via GitHub, ArXiv, et conférences comme NeurIPS.

```
# Exemple avec Polars pour analyser des donn es g nomiques
import polars as pl

donnees = {"G ne": ["BRCA1", "TP53"], "Expression": [5000, 3000]}
df = pl.DataFrame(donnees)
print(df.group_by("G ne").agg(pl.col("Expression").sum()))
```

1.3 Comparaison des langages pour la recherche scientifique

TABLE 1.1 – Langages pour la recherche scientifique mondiale (2025)

Langage	**Classification**	**Caractéristiques clés**	**Applications académiques**
Python	Sens large	Simplicité, écosystème (NumPy, PyTorch)	IA (DeepMind), astrophysique (NASA)
Julia	Sens strict	Haute performance, calcul parallèle	Simulations physiques (MIT)
Rust	Sens large	Sécurité mémoire, systèmes embarqués	Capteurs IoT (ETH Zurich)
Go	Sens large	Simplicité, concurrence	Applications web (Google Research)
Mojo	Sens strict	Performances IA, compatible Python	Deep learning (Stanford)
R	Sens strict	Statistique, visualisation	Bioinformatique (Harvard)
MATLAB	Sens strict	Environnement intégré	Ingénierie (Caltech)

1.3.1 Remarques sur l'évolution future

Python domine grâce à son adoption dans les revues scientifiques et les collaborations internationales, mais Julia (performances) et Rust (systèmes embarqués) gagnent du terrain. Dans un contexte académique :

- **Innovation** : Adopter des outils comme Mojo pour l'IA ou Go pour les infrastructures.
- **Fondamentaux théoriques** : Maîtriser l'algèbre linéaire, les probabilités, et l'optimisation.
- **Reproductibilité** : Utiliser des outils comme Docker pour standardiser les environnements.

Chapitre 2

Fondamentaux de Python

Ce chapitre présente les bases de Python (3.11+), utilisé dans les grandes universités pour le calcul scientifique, l'IA, et la bioinformatique. Il couvre les variables, structures de contrôle, programmation orientée objet, et les nouvelles fonctionnalités comme le typage statique et le pattern matching.

2.1 Bases du langage

2.1.1 Déclaration de variables et typage

Le typage statique avec `typing` est standard dans les projets académiques.

```python
from typing import List, Optional

temperature: float = 298.15  # Temp rature en Kelvin
experiences: List[int] = [1, 2, 3]
resultat: Optional[float] = None
```

2.1.2 Expressions et pattern matching

Le pattern matching (Python 3.10+) est utilisé dans les algorithmes complexes.

```python
def analyser_resultat(valeur: float) -> str:
    match valeur:
        case n if n > 100:
            return "  leve  "
        case n if n > 50:
            return "Moyenne"
        case _:
            return "Faible"

print(analyser_resultat(75.0))  # Affiche : Moyenne
```

2.1.3 Instructions de contrôle

Les boucles et conditions sont essentielles pour les simulations.

```python
# Boucle for pour it rer sur des donn es exp rimentales
for i, mesure in enumerate([10.5, 11.2, 10.8], 1):
    print(f"Mesure {i}: {mesure}")

# Boucle while
iteration = 5
while iteration > 0:
    print(f"It ration {iteration}")
    iteration -= 1
```

2.2 Structures de données avancées

2.2.1 Collections

Python offre des collections puissantes pour les données scientifiques :

- **Listes** : `[1, 2, 3]`
- **Tuples** : `(1, 2, 3)`
- **Sets** : `{1, 2, 3}`
- **Dictionnaires** : `{"cle": "valeur"}`

```python
from collections import deque

# File pour donn es temporelles
donnees_temps = deque([10.5, 11.2, 10.8])
donnees_temps.appendleft(9.8)
print(donnees_temps)  # Affiche : deque([9.8, 10.5, 11.2, 10.8])
```

2.2.2 Générateurs

Les générateurs optimisent la mémoire pour les grands datasets.

```python
def lire_donnees(fichier: str):
    with open(fichier, 'r') as f:
        for ligne in f:
            yield float(ligne.strip())
```

2.3 Programmation orientée objet

2.3.1 Classes et objets

La POO est utilisée dans les simulations complexes (par exemple, dynamique moléculaire).

```python
class Experiment:
    def __init__(self, nom: str, domaine: str):
        self.nom = nom
        self.domaine = domaine

    def presenter(self) -> None:
```

```
        print(f"Exp rience {self.nom} en {self.domaine}")

exp = Experiment("Simulation QCD", "Physique")
exp.presenter()
```

2.3.2 Héritage

L'héritage structure le code dans les projets collaboratifs.

```
class Simulation(Experiment):
    def __init__(self, nom: str, domaine: str, parametre: str):
        super().__init__(nom, domaine)
        self.parametre = parametre

    def presenter(self) -> None:
        print(f"Simulation {self.nom}, {self.parametre}")

sim = Simulation("Climat", "M t orologie", "Mod le GCM")
sim.presenter()
```

2.4 Écosystème Python

2.4.1 Bibliothèques clés

- **NumPy, Polars, JAX** : Calcul numérique (MIT, CERN).
- **PyTorch, scikit-learn** : IA (Stanford, DeepMind).
- **Plotly, seaborn** : Visualisation (Nature, Science).
- **SymPy** : Calcul symbolique (Caltech).

```
import numpy as np

tableau = np.array([1.5, 2.3, 3.1])
print(tableau.mean())  # Affiche : 2.3
```

2.4.2 Environnements de développement

- **Anaconda** : Standard en recherche (Harvard).
- **Google Colab** : Cloud pour collaborations (Oxford).
- **IDE** : VSCode, PyCharm.

2.4.3 Gestion des paquets

- **Poetry** : Gestion moderne (`poetry add numpy`).
- **pip** : Standard pour les bibliothèques.

```
# Exemple avec Poetry
# poetry init
# poetry add pytorch
```

Chapitre 3

Python pour le Calcul Scientifique : Applications Académiques

Ce chapitre explore l'utilisation de Python dans la recherche mondiale, avec des exemples en physique, bioinformatique, et climatologie.

3.1 Calcul Numérique

3.1.1 Algèbre linéaire avec NumPy et CuPy

NumPy et CuPy (GPU) sont utilisés pour résoudre des systèmes linéaires, par exemple en dynamique des fluides (MIT).

```
import numpy as np
import cupy as cp

# Syst me lin aire (CPU)
A = np.array([[2, 1], [1, 3]])
b = np.array([8, 11])
solution = np.linalg.solve(A, b)
print(solution)  # Affiche : [3. 2.]

# Version GPU
A_gpu = cp.array(A)
b_gpu = cp.array(b)
solution_gpu = cp.linalg.solve(A_gpu, b_gpu)
print(solution_gpu)
```

3.1.2 Optimisation avec JAX

JAX est utilisé pour des calculs différentiables, par exemple en optimisation de réseaux neuronaux (Stanford).

```
import jax.numpy as jnp
from jax import grad

```

```
def energie(p: float) -> float:
    return p**2 + 100*p + 1000  #  nergie  potentielle

grad_energie = grad(energie)
print(grad_energie(0.0))  # D riv e en p=0
```

3.2 Analyse et Visualisation de Données

3.2.1 Manipulation avec Polars

Polars est plus rapide que pandas pour les grands datasets, utilisé à Harvard pour la bioinformatique.

```
import polars as pl

donnees = {
    "G ne": ["BRCA1", "TP53", "EGFR"],
    "Expression": [5000, 3000, 4000]
}
df = pl.DataFrame(donnees)
print(df.group_by("G ne").agg(pl.col("Expression").sum()))
```

3.2.2 Visualisation avec Plotly

Plotly est utilisé pour des publications interactives (Nature).

```
import plotly.express as px

donnees = {
    "Temps": [1, 2, 3],
    "Signal": [100, 150, 120]
}
df = px.data.frame(donnees)
fig = px.line(df, x="Temps", y="Signal", title="Signal
   exp rimental")
fig.show()
```

3.3 Calcul Symbolique avec SymPy

SymPy est utilisé pour des calculs analytiques, par exemple en physique théorique (Caltech).

```
from sympy import symbols, diff, solve

x = symbols('x')
equation = x**2 + 4*x + 4
derivee = diff(equation, x)
solution = solve(derivee, x)
print(solution)  # Affiche : [-2]
```

3.4 Simulation et Calcul Distribué

3.4.1 Simulation Monte Carlo

Simuler des particules au CERN.

```
import numpy as np

np.random.seed(42)
trajectoires = np.random.normal(0, 1, 10000)  # Positions
prob_collision = np.mean(np.abs(trajectoires) > 2)
print(f"Probabilit  de collision : {prob_collision:.3f}")
```

3.4.2 Dask pour le big data

Dask est utilisé pour les grands datasets, par exemple à l'Imperial College.

```
import dask.dataframe as dd

df = dd.read_csv("donnees_genomiques.csv")
moyenne = df["Expression"].mean().compute()
print(moyenne)
```

Chapitre 4

Intelligence Artificielle et Machine Learning

Ce chapitre introduit l'IA et le machine learning, essentiels dans les grandes universités (DeepMind, Stanford) pour des applications comme la prédiction climatique ou l'analyse génomique.

4.1 Introduction au Machine Learning

4.1.1 Régression avec scikit-learn

Prédire des propriétés physiques.

```
from sklearn.linear_model import LinearRegression
import numpy as np

X = np.array([[1], [2], [3]])  # Param tres exp rimentaux
y = np.array([10, 20, 25])     # R sultats mesur s
model = LinearRegression().fit(X, y)
print(f"Pr diction pour x=4 : {model.predict([[4]])}")
```

4.1.2 Réseaux neuronaux avec PyTorch

Modélisation avancée (DeepMind).

```
import torch
import torch.nn as nn

class Modele(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc = nn.Linear(1, 1)
    def forward(self, x):
        return self.fc(x)

model = Modele()
```

```
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

4.2 Cas d'étude : Prédiction de données climatiques

Un exemple intégrant prétraitement, modélisation, et visualisation.

```
import polars as pl
from sklearn.ensemble import RandomForestRegressor
import plotly.express as px

df = pl.DataFrame({
    "Temps": range(1, 31),
    "Temp rature": np.random.normal(20, 5, 30),
    "CO2": np.random.normal(400, 20, 30)
})
X = df[["Temp rature"]].to_numpy()
y = df["CO2"].to_numpy()
model = RandomForestRegressor().fit(X, y)
fig = px.scatter(x=X.flatten(), y=y, labels={"x": "Temp rature (
    C )", "y": "CO2 (ppm)"})
fig.show()
```

Chapitre 5

Outils Avancés et Déploiement

Ce chapitre explore les outils avancés et le déploiement, utilisés dans les laboratoires mondiaux pour la reproductibilité et la collaboration.

5.1 Scripts vs. Notebooks

5.1.1 Scripts .py

Idéal pour les simulations reproductibles.

```python
# fichier: simulation.py
def calcul_energie(parametres: List[float]) -> float:
    return sum(p**2 for p in parametres)
```

5.1.2 Notebooks .ipynb

Parfait pour les publications interactives.

```python
import plotly.express as px

df = px.data.frame({"Temps": [1, 2, 3], "Signal": [100, 150, 120]})
fig = px.line(df, x="Temps", y="Signal", title="Signal
   exp rimental")
fig.show()
```

5.2 Débogage

Utiliser `pdb` ou des IDE comme VSCode (MIT).

```python
import pdb

def calcul_simulation(valeur: float) -> float:
    pdb.set_trace()
    return valeur * 2
```

5.3 Standards de code

5.3.1 PEP 8 et outils modernes

Ruff est utilisé pour vérifier le style dans les projets académiques.

```
# Exemple de formatage
# ruff check simulation.py
```

5.4 Déploiement

5.4.1 Applications web avec Streamlit

Partager des visualisations (Nature).

```
import streamlit as st

st.title("Analyse de Donn es")
valeur = st.slider("Param tre", 0, 100, 50)
st.write(f"Valeur s lectionn e : {valeur}")
```

5.4.2 Conteneurisation avec Docker

Standard pour la reproductibilité (CERN).

```
# Dockerfile
FROM python:3.12
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

5.5 Collaboration académique

— **GitHub** : Partage de code (Stanford).
— **ArXiv** : Publication de notebooks.
— **Google Colab** : Collaboration en temps réel.

Annexe A

Installation des Bibliothèques Scientifiques

A.1 Configuration de l'environnement Python

— Installation avec `pip` :

```
pip install numpy scipy pandas matplotlib
```

— Installation avec `conda` :

```
conda install -c conda-forge numpy scipy
```

A.2 Liste des Bibliothèques Essentielles

Table A.1 – Bibliothèques Python pour la recherche

Bibliothèque	**Utilisation**
NumPy	Calcul numérique
SciPy	Algorithmes scientifiques
Polars	Dataframes optimisés
PyTorch	Deep Learning

Annexe B

Exemples Complémentaires

B.1 Calcul Matriciel Avancé

```
import numpy as np
A = np.random.rand(100, 100)  # Matrice al atoire
valeurs_propres = np.linalg.eigvals(A)  # Calcul des valeurs
   propres
```

B.2 Optimisation avec SciPy

```
from scipy.optimize import minimize
result = minimize(lambda x: x**2 + 10, x0=1.0)
print(result.x)  # Affiche le minimum
```

Annexe C

Machine Learning avec Python

Les applications du machine learning, avec des outils comme scikit-learn, PyTorch, et JAX [?, ?].

C.1 Régression et classification

C.1.1 Régression linéaire avec scikit-learn

Prédire des propriétés continues, comme les niveaux de CO2.

```
from sklearn.linear_model import LinearRegression
import numpy as np

X = np.array([[10], [20], [30]])  # Temp ratures
y = np.array([400, 410, 420])     # Niveaux de CO2
model = LinearRegression().fit(X, y)
print(f"Pr diction pour 25 C  : {model.predict([[25]])}")
```

C.1.2 Classification avec Random Forest

Classifier des données, par exemple pour diagnostiquer des maladies.

```
from sklearn.ensemble import RandomForestClassifier
import polars as pl

df = pl.DataFrame({
    "Feature1": [1, 2, 3, 4],
    "Feature2": [10, 20, 15, 25],
    "Label": ["Pos", "Neg", "Pos", "Neg"]
})
X = df[["Feature1", "Feature2"]].to_numpy()
y = df["Label"].to_numpy()
model = RandomForestClassifier().fit(X, y)
print(f"Pr diction : {model.predict([[2.5, 18]])}")
```

C.2 Réseaux neuronaux avec PyTorch

PyTorch est utilisé pour des modèles complexes, comme les réseaux convolutifs (CNN).

```
import torch
import torch.nn as nn

class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv = nn.Conv2d(1, 16, kernel_size=3)
        self.fc = nn.Linear(16 * 26 * 26, 10)  # Exemple pour MNIST
    def forward(self, x):
        x = torch.relu(self.conv(x))
        x = x.view(x.size(0), -1)
        return self.fc(x)

model = CNN()
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

C.3 Optimisation avec JAX

JAX permet des calculs différentiables pour optimiser les modèles.

```
import jax.numpy as jnp
from jax import grad, jit

@jit
def perte_modele(params: jnp.ndarray, x: jnp.ndarray, y: jnp.
   ndarray) -> float:
    predictions = jnp.dot(x, params)
    return jnp.mean((predictions - y) ** 2)

grad_perte = grad(perte_modele)
params = jnp.array([0.1, 0.2])
x = jnp.array([[1, 2], [3, 4]])
y = jnp.array([2, 4])
print(grad_perte(params, x, y))
```

C.4 Bibliothèques récentes à utiliser

- **scikit-learn** : Algorithmes de machine learning classiques (régression, classification) [?].
- **PyTorch** : Deep learning avec prise en charge GPU/TPU [?].
- **JAX** : Calculs différentiables et optimisations, utilisé à Stanford [?].
- **XGBoost** : Algorithmes de boosting pour des performances élevées [?].
- **LightGBM** : Gradient boosting optimisé pour les grands datasets [?].

Annexe D

Traitement du Langage Naturel (NLP)

Le NLP est un pilier de l'IA, utilisé pour analyser des textes scientifiques ou développer des assistants IA [?].

D.1 Analyse de sentiments avec Hugging Face

Hugging Face permet de déployer rapidement des modèles NLP.

```
from transformers import pipeline

classifier = pipeline("sentiment-analysis")
result = classifier("This AI model is revolutionary!")
print(result)  # Affiche : [{'label': 'POSITIVE', 'score': 0.99}]
```

D.2 Génération de texte

Un exemple de génération de texte avec GPT-2.

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer

tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
model = GPT2LMHeadModel.from_pretrained("gpt2")
inputs = tokenizer("The future of AI is", return_tensors="pt")
outputs = model.generate(**inputs, max_length=50)
print(tokenizer.decode(outputs[0], skip_special_tokens=True))
```

D.3 Cas d'étude : Analyse de littérature scientifique

Analyser des abstracts d'articles avec NLP.

```
import polars as pl
from transformers import pipeline

df = pl.DataFrame({
```

```
    "Abstract": ["Novel AI model improves accuracy.", "AI ethics is
        critical."]
})
classifier = pipeline("text-classification", model="distilbert-base
    -uncased")
results = classifier(df["Abstract"].to_list())
print(results)
```

D.4 Bibliothèques récentes à utiliser

- **Hugging Face Transformers** : Modèles pré-entraînés pour le NLP (BERT, GPT) [?].
- **spaCy** : Traitement linguistique avancé pour l'extraction d'entités [?].
- **NLTK** : Analyse textuelle pour la recherche académique [?].
- **SentenceTransformers** : Embeddings pour la recherche sémantique [?].
- **Flair** : Modèles NLP pour la classification et l'analyse de séquences [?].

Annexe E

Vision par Ordinateur

La vision par ordinateur est utilisée pour analyser des images médicales ou astronomiques.

E.1 Classification d'images avec PyTorch

Un exemple avec ResNet pour classifier des images.

```
import torch
import torchvision.models as models
import torchvision.transforms as transforms
from PIL import Image

model = models.resnet18(pretrained=True)
model.eval()
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.ToTensor(),
])
image = Image.open("sample_image.jpg")
image = transform(image).unsqueeze(0)
output = model(image)
print(f"Classe pr dite : {torch.argmax(output, dim=1)}")
```

E.2 Segmentation sémantique

Un exemple avec U-Net pour segmenter des images médicales.

```
import torch
import torch.nn as nn

class UNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.encoder = nn.Conv2d(1, 64, kernel_size=3)
        self.decoder = nn.ConvTranspose2d(64, 1, kernel_size=3)
    def forward(self, x):
```

```
        x = torch.relu(self.encoder(x))
        return self.decoder(x)

model = UNet()
```

E.3 Bibliothèques récentes à utiliser

- **Torchvision** : Modèles et transformations pour la vision par ordinateur [?].
- **OpenCV** : Traitement d'images et détection de caractéristiques [?].
- **Albumentations** : Augmentation de données pour l'entraînement des modèles [?].
- **Detectron2** : Plateforme pour la détection et la segmentation d'objets, développée par Meta AI [?].
- **MMEngine** : Framework pour les pipelines de vision par ordinateur [?].

Annexe F

Internet des Objets (IoT)

Python est utilisé pour collecter et analyser des données IoT, par exemple dans les villes intelligentes.

F.1 Communication MQTT

Un exemple de collecte de données de capteurs.

```
import paho.mqtt.client as mqtt

def on_message(client, userdata, message):
    print(f"Donn es re ues : {message.payload.decode()}")

client = mqtt.Client()
client.on_message = on_message
client.connect("broker.hivemq.com", 1883)
client.subscribe("capteur/temperature")
client.loop_start()
```

F.2 Traitement des données IoT

Analyser des données de capteurs avec Polars.

```
import polars as pl

df = pl.DataFrame({
    "Temps": range(1, 11),
    "Temp rature": np.random.normal(25, 2, 10)
})
print(df.group_by(pl.col("Temps") // 5).agg(pl.col("
   Temp rature").mean()))
```

F.3 Bibliothèques récentes à utiliser

- **Paho-MQTT** : Communication avec des capteurs IoT via MQTT [?].

- **Polars** : Analyse rapide de données IoT [**?**].
- **PySerial** : Communication avec des microcontrôleurs (ex. : Arduino) [**?**].
- **Adafruit-CircuitPython** : Gestion de capteurs pour microcontrôleurs [**?**].
- **InfluxDB** : Stockage et analyse de séries temporelles IoT [**?**].

Annexe G

Éthique en IA

L'éthique en IA est cruciale pour garantir des modèles équitables et transparents.

G.1 Évaluation des biais

Utiliser Fairlearn pour analyser les biais dans un modèle.

```
from fairlearn.metrics import MetricFrame
from sklearn.metrics import accuracy_score
import numpy as np

y_true = np.array([0, 1, 1, 0])
y_pred = np.array([0, 1, 0, 0])
groupes = np.array(["A", "A", "B", "B"])
mf = MetricFrame(metrics=accuracy_score, y_true=y_true,
    y_pred=y_pred, sensitive_features=groupes)
print(mf.by_group)
```

G.2 Transparence et documentation

Documenter les modèles avec des métadonnées.

```
from sklearn.ensemble import RandomForestClassifier
import json

model = RandomForestClassifier()
metadata = {
    "model_type": "RandomForest",
    "training_date": "2025-05-23",
    "data_source": "Dataset X"
}
with open("model_metadata.json", "w") as f:
    json.dump(metadata, f)
```

G.3 Bibliothèques récentes à utiliser

- **Fairlearn** : Évaluation et mitigation des biais dans les modèles d'IA [**?**].
- **AIF360** : Analyse de l'équité des algorithmes, développé par IBM [**?**].
- **EthicalML** : Outils pour auditer les modèles d'IA [**?**].
- **Responsible-AI-Toolbox** : Analyse de l'équité et de l'interprétabilité [**?**].
- **Shap** : Explicabilité des modèles via les valeurs de Shapley [**?**].

Annexe H

Calcul Quantique

Le calcul quantique, intégré à l'IA, est exploré avec Qiskit

H.1 Circuits quantiques avec Qiskit

Un exemple de circuit quantique.

```
from qiskit import QuantumCircuit

qc = QuantumCircuit(2, 2)
qc.h(0)  # Porte Hadamard
qc.cx(0, 1)  # Porte CNOT
qc.measure([0, 1], [0, 1])
print(qc)
```

H.2 Machine learning quantique

Un exemple de QSVM (Quantum Support Vector Machine).

```
from qiskit import QuantumCircuit
from qiskit_machine_learning.algorithms import QSVC

qc = QuantumCircuit(2)
qc.h([0, 1])
qsvc = QSVC(quantum_kernel=qc)
# Entra ner le mod le avec des donn es
```

H.3 Bibliothèques récentes à utiliser

- **Qiskit** : Simulation de circuits quantiques et algorithmes d'IA quantique

Annexe I

Outils Avancés et Déploiement

Ce chapitre explore les outils pour la reproductibilité et le déploiement.

I.1 Streamlit pour les applications web

Créer une interface pour visualiser les résultats d'IA.

```
import streamlit as st
import plotly.express as px

st.title("Visualisation de Mod le IA")
valeur = st.slider("Seuil de confiance", 0.0, 1.0,
    0.5)
df = px.data.iris()
fig = px.scatter(df, x="sepal_width", y="
    sepal_length", color="species")
st.plotly_chart(fig)
```

I.2 Docker pour la reproductibilité

Standardiser les environnements.

```
# Dockerfile
FROM python:3.12
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
CMD ["streamlit", "run", "app.py"]
```

I.3 Gestion des dépendances avec Poetry

Installer des bibliothèques comme PyTorch.

```
# Commandes Poetry
# poetry init
```

```
# poetry add torch transformers polars
```

I.4 Bibliothèques récentes à utiliser

- **Streamlit** : Création d'applications web interactives pour visualiser les modèles [**?**].
- **Poetry** : Gestion des dépendances pour des projets reproductibles

Annexe J

Tendances Émergentes

J.1 IA Générative

Les modèles comme LLaMA et Stable Diffusion dominent [?].

```
from diffusers import StableDiffusionPipeline

pipe = StableDiffusionPipeline.from_pretrained("
    runwayml/stable-diffusion-v1-5")
image = pipe("A futuristic city").images[0]
image.save("futuristic_city.png")
```

J.2 Calcul quantique et IA

Explorer les algorithmes quantiques pour l'IA.

```
from qiskit import QuantumCircuit

qc = QuantumCircuit(3)
qc.h([0, 1, 2])  # Superposition
print(qc)
```

J.3 Bibliothèques récentes à utiliser

- **Diffusers** : Génération d'images avec des modèles comme Stable Diffusion [?].
- **LangChain** : Développement d'applications avec des modèles de langage [?].
- **Qiskit** : Calcul quantique pour des algorithmes d'IA innovants

Références

1. T. Darcet et al.
   Vision Transformers Need Registers. arXiv preprint arXiv :2310.02557, 2024.
   https://arxiv.org/abs/2310.02557
2. Z. Shi et al.
   Why Larger Language Models Do In-context Learning Differently ? arXiv preprint arXiv :2405.19592, 2024.
   https://arxiv.org/abs/2405.19592
3. A. Grattafiori et al.
   The Llama 3 Herd of Models. arXiv preprint arXiv :2407.21783, 2024.
   https://arxiv.org/abs/2407.21783
4. T. Mesnard et al.
   Gemma : Open Models Based on Gemini Research and Technology. arXiv preprint arXiv :2403.08295, 2024.
   https://arxiv.org/abs/2403.08295
5. K. Tian et al.
   Visual Autoregressive Modeling : Scalable Image Generation via Next-Scale Prediction. arXiv preprint arXiv :2404.02905, 2024.
   https://arxiv.org/abs/2404.02905
6. N. Ravi et al.
   SAM 2 : Segment Anything in Images and Videos. OpenReview, 2025.
   https://openreview.net/forum?id=Ha6RTeWMd0
7. Y. Ren and D. Sutherland.
   Learning Dynamics of LLM Finetuning. OpenReview, 2025.
   https://openreview.net/forum?id=tPNHOoZFl9
8. J. T. Wang et al.
   Data Shapley in One Training Run. OpenReview, 2025.
   https://openreview.net/forum?id=HD6bWcj87Y
9. H. Narasimhan et al.
   Faster Cascades via Speculative Decoding. OpenReview, 2025.
   https://openreview.net/forum?id=vo9t20wsmd
10. B. Vasudeva et al.
    Transformers Learn Low Sensitivity Functions. OpenReview, 2025.
    https://openreview.net/forum?id=4ikjWBs3tE
11. A. Javadi-Abhari et al.

Quantum Computing with Qiskit. arXiv preprint arXiv :2405.08810, 2024.
https://arxiv.org/abs/2405.08810

12. Qiskit Contributors.
Qiskit : An Open-source Framework for Quantum Computing. Zenodo, 2023.
https://doi.org/10.5281/zenodo.2573505
13. SciPy Contributors.
SciPy 1.15.0 Release Notes. SciPy Documentation, 2025.
https://docs.scipy.org/doc/scipy/release.html
14. SciPy Contributors.
SciPy 2024 Proceedings. Proceedings of the 23rd Python in Science Conference, 2024.
https://proceedings.scipy.org/2024
15. P. Virtanen et al.
SciPy 1.0 : Fundamental Algorithms for Scientific Computing in Python. Nature Methods, vol. 17, no. 3, 2020, pp. 261–272.
https://doi.org/10.1038/s41592-019-0686-2
16. H. P. Langtangen.
A Primer on Scientific Programming with Python. 5e éd., Springer, 2016.
https://link.springer.com/book/10.1007/978-3-662-49887-3
17. A. Géron.
Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. 2e éd., O'Reilly Media, 2019.
18. A. Paszke et al.
PyTorch : An Imperative Style, High-Performance Deep Learning Library. NeurIPS, 2019.
https://arxiv.org/abs/1912.01703
19. J. VanderPlas.
Python Data Science Handbook. O'Reilly Media, 2016.
https://jakevdp.github.io/PythonDataScienceHandbook/
20. T. E. Oliphant.
The NumPy Array : A Structure for Efficient Numerical Computation. Computing in Science & Engineering, vol. 13, no. 2, 2011, pp. 22–30.
https://doi.org/10.1109/MCSE.2011.37
21. T. Wolf et al.
HuggingFace's Transformers : State-of-the-Art Natural Language Processing. arXiv, 2020.
https://arxiv.org/abs/1910.01108
22. Python Software Foundation.
PEP 2026 : Calendar Versioning for Python. Python Enhancement Proposals, 2024.
https://peps.python.org/pep-2026/
23. Python Software Foundation.
PEP 750 : Template Strings. Python Enhancement Proposals, 2024.

https://peps.python.org/pep-0750/