Université Cadi Ayyad

Faculté des Sciences

Département Informatique

Master Intelligence Artificielle

Module : MLOps

End-to-End AutoML Platform Using Kubeflow + Katib

Optimisation d'Hyperparamètres et Pipelines ML

Random Search, Bayesian Optimization & TPE

Réalisé par :

Assoumana Souley Hadiza Dite Maa

Bouamir Assia

Encadré par :

Pr. Fahd Kalloubi

Année Universitaire 2024-2025

Table des matières

1 Introduction

Ce document présente un guide complet pour mettre en place une plateforme AutoML utilisant **Katib** pour l'optimisation d'hyperparamètres et **Kubeflow Pipelines** pour orchestrer le workflow d'entraînement.

1.1 Objectifs du Projet

— Déployer une infrastructure Kubernetes locale avec Minikube
— Installer et configurer Kubeflow Pipelines et Katib
— Implémenter des pipelines d'optimisation (Random, Bayésien et TPE)
— Comparer les algorithmes et sélectionner automatiquement le meilleur modèle
— Sauvegarder les artefacts dans Kubernetes

1.2 Architecture Globale

Le système AutoML se compose de plusieurs couches :

— **Infrastructure** : Minikube (Kubernetes local)
— **Orchestration** : Kubeflow Pipelines v2.14.3
— **Optimisation** : Katib v0.17.0
— **Stockage** : MinIO pour les artefacts
— **Conteneurisation** : Docker

Information

Technologies utilisées :

— Kubernetes (Minikube)
— Kubeflow Pipelines v2.14.3
— Katib v0.17.0 (v1beta1)
— Docker Desktop
— Python 3.9
— scikit-learn, pandas, numpy

2 Installation de l'Infrastructure

2.1 Étape 1 : Installation de Minikube

2.1.1 Prérequis Système

Vérifiez que votre système dispose de :

- Minimum 6 GB de RAM (recommandé : 8 GB)
- 4 CPUs
- 20 GB d'espace disque disponible
- Windows 10/11 avec PowerShell

2.1.2 Installation sous Windows

Listing 1 – Installation Minikube - PowerShell (Administrateur)

```
# Creer le repertoire d'installation
New-Item -Path 'c:\' -Name 'minikube' -ItemType Directory -Force

# Telecharger Minikube
$ProgressPreference = 'SilentlyContinue'
Invoke-WebRequest -OutFile 'c:\minikube\minikube.exe' `
  -Uri 'https://github.com/kubernetes/minikube/releases/latest/
     download/minikube-windows-amd64.exe' `
  -UseBasicParsing
```

Listing 2 – Ajouter Minikube au PATH

```
# Ajouter au PATH systeme (PowerShell Administrateur)
$oldPath = [Environment]::GetEnvironmentVariable('Path', [
   EnvironmentVariableTarget]::Machine)
if ($oldPath.Split(';') -inotcontains 'C:\minikube'){
  [Environment]::SetEnvironmentVariable('Path', $('{0};C:\minikube'
      -f $oldPath), [EnvironmentVariableTarget]::Machine)
}
```

Attention

Important : Fermez et rouvrez votre terminal PowerShell après l'ajout au PATH !

2.1.3 Vérification

Listing 3 – Vérifier l'installation

```
# Verifier la version
minikube version
```

2.2 Étape 2 : Installation de Docker Desktop

1. Télécharger Docker Desktop : `https://www.docker.com/products/docker-desktop`
2. Exécuter l'installateur
3. Activer WSL 2 (Windows Subsystem for Linux)
4. Redémarrer l'ordinateur
5. Démarrer Docker Desktop

Listing 4 – Vérification Docker

```
# Verifier Docker
docker --version

# Tester Docker
docker run hello-world
```

Attention

Docker Desktop doit être en cours d'exécution avant de démarrer Minikube !

2.3 Étape 3 : Démarrage du Cluster Minikube

Listing 5 – Démarrer Minikube avec Docker

```
# Demarrer Minikube avec configuration optimale
minikube start --driver=docker --memory=6000 --cpus=4

# Verifier le statut
minikube status
```

Succès

Si vous voyez `minikube: Running`, votre cluster est prêt !

2.4 Étape 4 : Installation de Kubeflow Pipelines

Listing 6 – Installation Kubeflow Pipelines v2.14.3

```
# Definir la version
$env:PIPELINE_VERSION = "2.14.3"

# Appliquer les ressources cluster-scoped
kubectl apply -k "github.com/kubeflow/pipelines/manifests/kustomize
   /cluster-scoped-resources?ref=$env:PIPELINE_VERSION"

# Attendre que les CRDs soient etablies
kubectl wait --for condition=established --timeout=60s crd/
   applications.app.k8s.io

# Installer les composants Kubeflow
kubectl apply -k "github.com/kubeflow/pipelines/manifests/kustomize
   /env/platform-agnostic?ref=$env:PIPELINE_VERSION"
```

Information

Le déploiement complet prend environ 3 minutes. Soyez patient !

2.5 Étape 5 : Installation de Katib

Listing 7 – Installation Katib v0.17.0

```
# Installer Katib standalone
kubectl apply -k "github.com/kubeflow/katib.git/manifests/v1beta1/
   installs/katib-standalone?ref=v0.17.0"

# Installer le SDK Python Katib
pip install -U kubeflow-katib
```

2.6 Étape 6 : Vérification des Pods

Listing 8 – Vérifier tous les pods

```
# Verifier les pods Kubeflow et Katib
kubectl get pods -n kubeflow

# Attendre que tous soient en Running (sauf minio initialement)
kubectl get pods -n kubeflow -w
```

2.7 Étape 7 : Correction du Problème MinIO

MinIO pose un problème connu avec l'image `latest`. Voici la solution :

Listing 9 – Corriger MinIO

```
# Patcher l'image MinIO vers une version stable
kubectl patch deployment minio -n kubeflow --type='json' -p='[
  {
    "op": "replace",
    "path": "/spec/template/spec/containers/0/image",
    "value": "minio/minio:RELEASE.2021-06-17T00-10-46Z"
  }
]'

# Supprimer les pods MinIO pour forcer la recreation
kubectl delete pod -n kubeflow -l app=minio

# Supprimer les pods ml-pipeline pour qu'ils se reconnectent
kubectl delete pod -n kubeflow -l app=ml-pipeline
```

Succès

Attendez 1-2 minutes et vérifiez que tous les pods sont en `Running` :

```
kubectl get pods -n kubeflow
```

2.8 Étape 8 : Accès aux Interfaces Web

Listing 10 – Port-forward pour Kubeflow UI

```
# Exposer Kubeflow Pipelines UI (dans un terminal separe)
kubectl port-forward -n kubeflow svc/ml-pipeline-ui 8080:80
```

Ouvrir dans le navigateur : `http://localhost:8080`

2.9 Kubeflow Pipelines UI

Listing 11 – Port-forward pour Kubeflow UI

```
# Exposer l'interface Kubeflow Pipelines
kubectl port-forward -n kubeflow svc/ml-pipeline-ui 8080:80

# Ouvrir dans le navigateur : http://localhost:8080
```

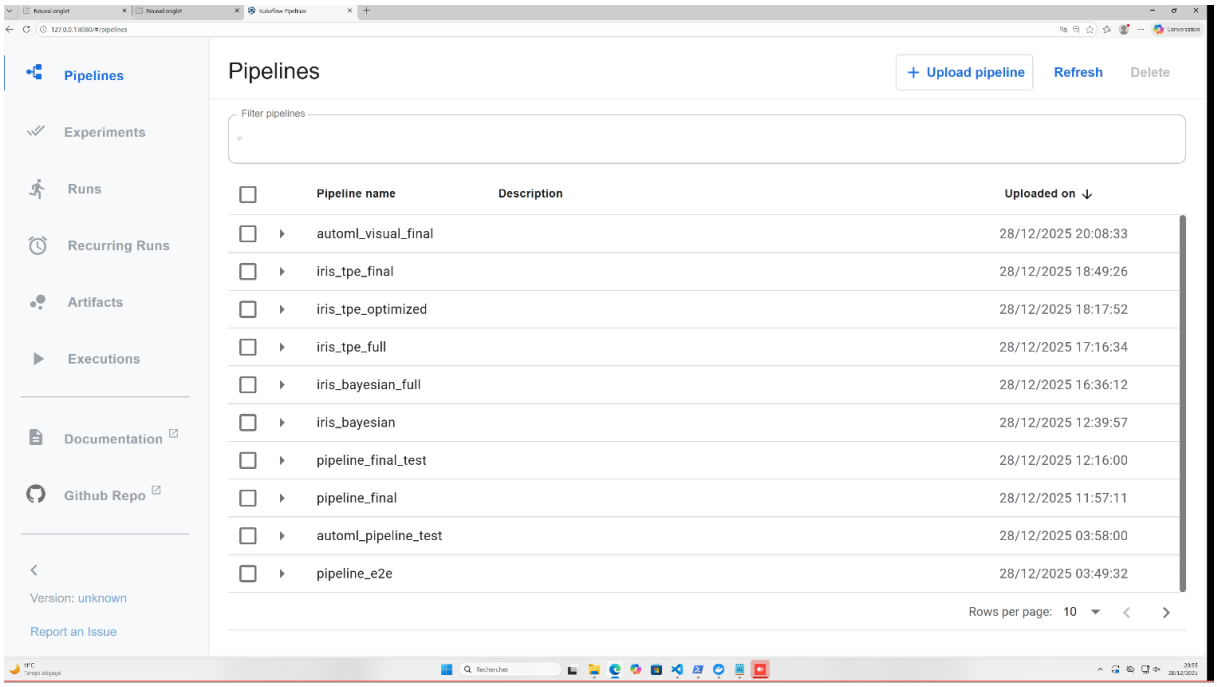

FIGURE 1 – Interface principale de Kubeflow Pipelines - Vue des pipelines

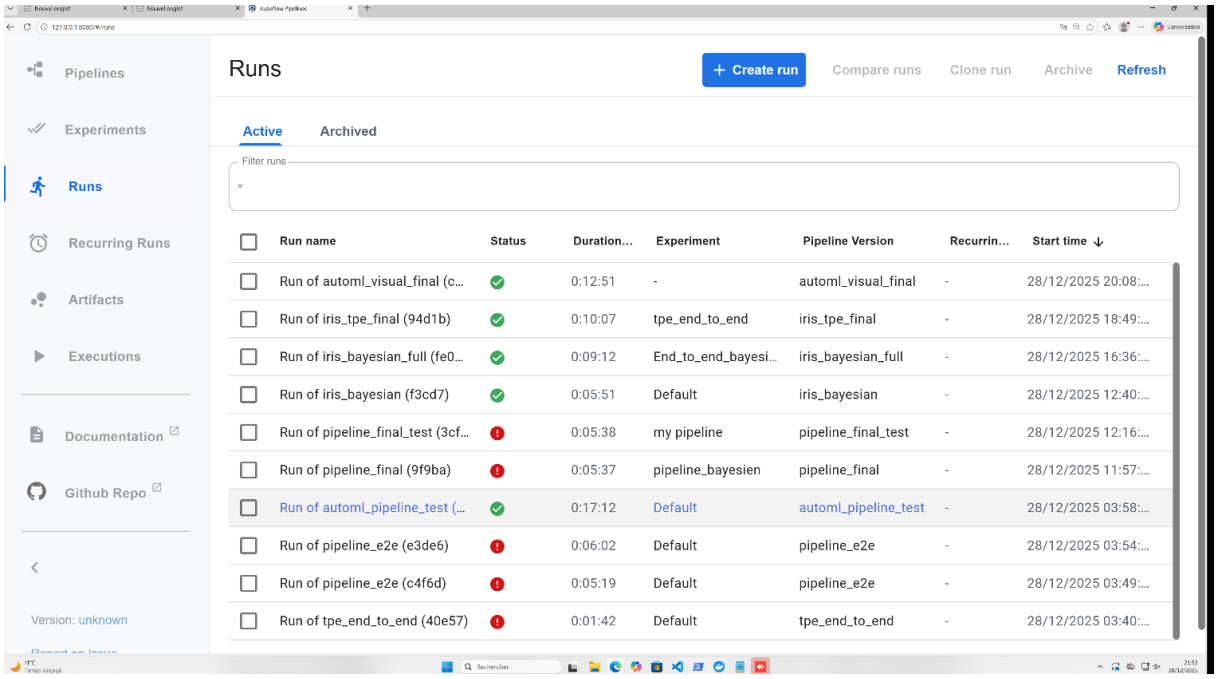

FIGURE 2 – Vue des exécutions (Runs) dans Kubeflow Pipelines

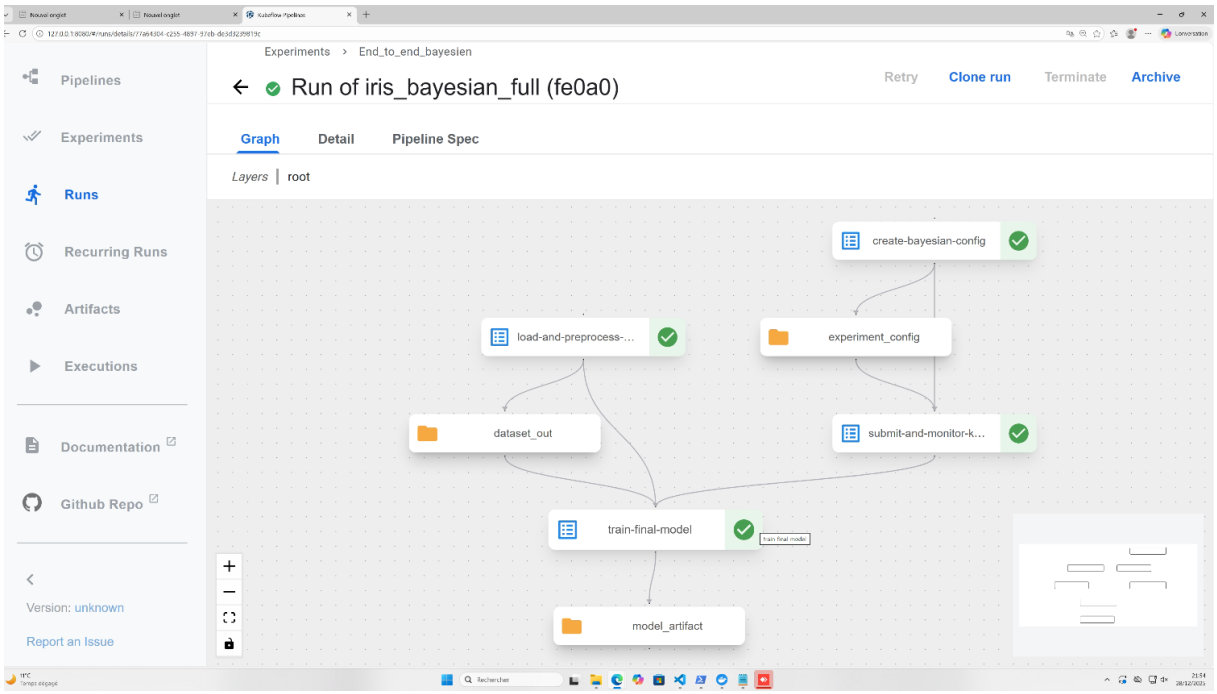

FIGURE 3 – Graphe d'exécution d'un pipeline dans Kubeflow

Listing 12 – Port-forward pour Katib UI

```
# Exposer Katib UI (dans un autre terminal)
kubectl port-forward -n kubeflow svc/katib-ui 8081:80
```

Ouvrir dans le navigateur : `http://localhost:8081/katib/`

Information

Laissez ces terminaux ouverts pendant toute la durée du travail !

3 Préparation du Code d'Entraînement

3.1 Structure du Projet

```
project/
 src/
    training/
        train.py          # Script d'entrainement
 pipelines/
    random_pipeline.py    # Pipeline Random Search
    bayesian_pipeline.py  # Pipeline Bayesian
    tpe_pipeline.py       # Pipeline TPE
    complete_pipeline.py  # Pipeline complet
 Dockerfile
 requirements.txt
```

3.2 Script d'Entraînement (train.py)

Listing 13 – src/training/train.py - Script complet

```python
import argparse
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
import json
import os

def train_model(learning_rate, hidden_units, optimizer):
    """
    Entraine un modele MLP sur le dataset Iris

    Args:
        learning_rate: Taux d'apprentissage
        hidden_units: Nombre d'unites dans la couche cachee
        optimizer: Algorithme d'optimisation (adam ou sgd)

    Returns:
        accuracy: Precision du modele
    """

    # Charger les donnees Iris
    print("Chargement du dataset Iris...")
    X, y = load_iris(return_X_y=True)

    # Split train/test
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y
    )

```

```
    # Normalisation des donnees
    mean = X_train.mean(axis=0)
    std = X_train.std(axis=0) + 1e-8
    X_train = (X_train - mean) / std
    X_test = (X_test - mean) / std

    print(f"Parametres d'entrainement:")
    print(f"  - Learning Rate: {learning_rate}")
    print(f"  - Hidden Units: {hidden_units}")
    print(f"  - Optimizer: {optimizer}")

    # Creation du modele
    model = MLPClassifier(
        hidden_layer_sizes=(hidden_units,),
        learning_rate_init=learning_rate,
        solver=optimizer,
        max_iter=1000,
        random_state=42,
        early_stopping=True,
        validation_fraction=0.1,
        n_iter_no_change=10
    )

    # Entrainement
    print("Debut de l'entrainement...")
    model.fit(X_train, y_train)

    # Evaluation
    y_pred_train = model.predict(X_train)
    y_pred_test = model.predict(X_test)

    train_accuracy = accuracy_score(y_train, y_pred_train)
    test_accuracy = accuracy_score(y_test, y_pred_test)
    loss = 1 - test_accuracy

    print(f"Resultats:")
    print(f"  - Train Accuracy: {train_accuracy:.4f}")
    print(f"  - Test Accuracy: {test_accuracy:.4f}")
    print(f"  - Loss: {loss:.4f}")

    # CRITIQUE : Ecriture des metriques pour Katib
    # Le format DOIT etre: metric_name=value
    metrics_path = "/tmp/katib-metrics.log"
    with open(metrics_path, "w") as f:
        f.write(f"accuracy={test_accuracy}\n")
        f.write(f"loss={loss}\n")

    print(f"Metriques ecrites dans {metrics_path}")

    # Sauvegarder les parametres et resultats
    results = {
```

```
        "learning_rate": learning_rate,
        "hidden_units": hidden_units,
        "optimizer": optimizer,
        "train_accuracy": float(train_accuracy),
        "test_accuracy": float(test_accuracy),
        "loss": float(loss),
        "n_iterations": model.n_iter_
    }

    results_path = "/tmp/training_results.json"
    with open(results_path, "w") as f:
        json.dump(results, f, indent=2)

    print(f"Resultats sauvegardes dans {results_path}")
    print("Entrainement termine avec succes!")

    return test_accuracy

def main():
    parser = argparse.ArgumentParser(
        description="Entrainement MLP sur Iris avec hyperparametres
            variables"
    )
    parser.add_argument(
        "--learning-rate",
        type=float,
        default=0.01,
        help="Taux d'apprentissage"
    )
    parser.add_argument(
        "--hidden-units",
        type=int,
        default=64,
        help="Nombre d'unites dans la couche cachee"
    )
    parser.add_argument(
        "--optimizer",
        type=str,
        default="adam",
        choices=["adam", "sgd"],
        help="Algorithme d'optimisation"
    )

    args = parser.parse_args()

    # Lancer l'entrainement
    accuracy = train_model(
        learning_rate=args.learning_rate,
        hidden_units=args.hidden_units,
        optimizer=args.optimizer
    )
```

```
133
134     print(f"\n=== ENTRAINEMENT TERMINE ===")
135     print(f"Accuracy finale: {accuracy:.4f}")
136
137 if __name__ == "__main__":
138     main()
```

Attention

Point critique : Le fichier `/tmp/katib-metrics.log` doit respecter strictement le format `metric_name=value` (une métrique par ligne) pour que Katib puisse parser les résultats !

3.3 Dockerfile

Listing 14 – Dockerfile pour l'image d'entrainement

```
FROM python:3.9-slim

WORKDIR /app

# Installation des dependances systeme
RUN apt-get update && apt-get install -y --no-install-recommends \
    build-essential \
    && rm -rf /var/lib/apt/lists/*

# Installation des bibliotheques Python
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copier le code d'entrainement
COPY src/ /app/src/

# Point d'entree
CMD ["python", "-u", "/app/src/training/train.py"]
```

3.4 Requirements

Listing 15 – requirements.txt

```
scikit-learn==1.3.2
numpy==1.24.3
pandas==2.0.3
```

3.5 Construction de l'Image Docker

Listing 16 – Build et chargement dans Minikube

```
# Builder l'image
docker build -t automl-training:v1.0 .

# Charger l'image dans Minikube
minikube image load automl-training:v1.0

# Verifier que l'image est presente
minikube image ls | findstr automl
```

Succès

L'image est maintenant disponible dans le cluster Minikube et peut être utilisée par les pods Katib !

4 Pipeline 1 : Random Search

4.1 Principe du Random Search

Le Random Search explore l'espace des hyperparamètres de manière aléatoire. C'est simple mais souvent efficace comme baseline.

4.2 Code Complet du Pipeline

Listing 17 – pipelines/random_pipeline.py

```python
import kfp
from kfp import dsl, compiler
from kfp.dsl import component, InputPath, OutputPath
from typing import NamedTuple
import json

@component(
    base_image="python:3.9",
    packages_to_install=["pandas==2.0.3", "scikit-learn==1.3.2"]
)
def load_and_preprocess_data(dataset_out: OutputPath("csv")):
    """Charge et preprocesse le dataset Iris"""
    import pandas as pd
    from sklearn.datasets import load_iris

    print("Chargement du dataset Iris...")
    iris = load_iris()
    df = pd.DataFrame(iris.data, columns=iris.feature_names)
    df['target'] = iris.target

    df.to_csv(dataset_out, index=False)
    print(f"Dataset sauvegarde: {len(df)} lignes, {len(df.columns)}
        colonnes")

@component(
    base_image="python:3.9",
    packages_to_install=["pyyaml==6.0.1"]
)
def create_random_config(
    experiment_name: str,
    namespace: str,
    training_image: str,
    experiment_config: OutputPath("yaml"),
) -> str:
    """Cree la configuration Katib pour Random Search"""
    import yaml

    config = {
        "apiVersion": "kubeflow.org/v1beta1",
        "kind": "Experiment",
        "metadata": {
```

```
            "name": experiment_name,
            "namespace": namespace
        },
        "spec": {
            "algorithm": {
                "algorithmName": "random"
            },
            "objective": {
                "type": "maximize",
                "objectiveMetricName": "accuracy",
                "additionalMetricNames": ["loss"]
            },
            "parameters": [
                {
                    "name": "learning_rate",
                    "parameterType": "double",
                    "feasibleSpace": {
                        "min": "0.001",
                        "max": "0.05"
                    }
                },
                {
                    "name": "hidden_units",
                    "parameterType": "int",
                    "feasibleSpace": {
                        "min": "32",
                        "max": "128"
                    }
                },
                {
                    "name": "optimizer",
                    "parameterType": "categorical",
                    "feasibleSpace": {
                        "list": ["adam", "sgd"]
                    }
                }
            ],
            "parallelTrialCount": 2,
            "maxTrialCount": 10,
            "metricsCollectorSpec": {
                "collector": {"kind": "File"},
                "source": {
                    "fileSystemPath": {
                        "path": "/tmp/katib-metrics.log",
                        "kind": "File"
                    }
                }
            },
            "trialTemplate": {
                "primaryContainerName": "training-container",
                "trialParameters": [
```

```
                    {"name": "lr", "reference": "learning_rate"},
                    {"name": "hu", "reference": "hidden_units"},
                    {"name": "opt", "reference": "optimizer"}
                ],
                "trialSpec": {
                    "apiVersion": "batch/v1",
                    "kind": "Job",
                    "spec": {
                        "template": {
                            "metadata": {
                                "annotations": {
                                    "sidecar.istio.io/inject": "
                                       false"
                                }
                            },
                            "spec": {
                                "containers": [{
                                    "name": "training-container",
                                    "image": training_image,
                                    "imagePullPolicy": "
                                       IfNotPresent",
                                    "command": [
                                        "python",
                                        "-u",
                                        "/app/src/training/train.py
                                           "
                                    ],
                                    "args": [
                                        "--learning-rate=${
                                           trialParameters.lr}",
                                        "--hidden-units=${
                                           trialParameters.hu}",
                                        "--optimizer=${
                                           trialParameters.opt}"
                                    ],
                                    "resources": {
                                        "limits": {
                                            "cpu": "500m",
                                            "memory": "512Mi"
                                        },
                                        "requests": {
                                            "cpu": "100m",
                                            "memory": "256Mi"
                                        }
                                    }
                                }],
                                "restartPolicy": "Never"
                            }
                        }
                    }
                }
```

```
            }
        }
    }

    with open(experiment_config, "w") as f:
        yaml.dump(config, f)

    print(f"Configuration Random Search creee pour {experiment_name}")
    return experiment_name

@component(
    base_image="python:3.9",
    packages_to_install=["kubernetes==28.1.0", "pyyaml==6.0.1"]
)
def submit_and_monitor_katib(
    experiment_name: str,
    namespace: str,
    experiment_config: InputPath("yaml"),
) -> NamedTuple("Outputs", [
    ("best_params", str),
    ("best_accuracy", float)
]):
    """Soumet l'experience Katib et monitore son execution"""
    import yaml
    import time
    import json
    from kubernetes import client, config

    # Charger la config Kubernetes
    try:
        config.load_incluster_config()
    except:
        config.load_kube_config()

    custom_api = client.CustomObjectsApi()
    group = "kubeflow.org"
    version = "v1beta1"
    plural = "experiments"

    # Charger la config de l'experience
    with open(experiment_config, "r") as f:
        exp_dict = yaml.safe_load(f)

    # Supprimer l'experience si elle existe deja
    try:
        custom_api.delete_namespaced_custom_object(
            group, version, namespace, plural, experiment_name
        )
        print(f"Experience existante {experiment_name} supprimee")
        time.sleep(5)
```

```
    except:
        pass

    # Creer l'experience
    custom_api.create_namespaced_custom_object(
        group, version, namespace, plural, body=exp_dict
    )
    print(f"Experience {experiment_name} lancee avec succes")

    # Monitoring en boucle
    max_wait_time = 3600  # 1 heure max
    start_time = time.time()

    while True:
        if time.time() - start_time > max_wait_time:
            raise Exception("Timeout: l'experience a depasse 1 heure")

        exp = custom_api.get_namespaced_custom_object(
            group, version, namespace, plural, experiment_name
        )

        status = exp.get("status", {})
        conditions = status.get("conditions", [])

        # Afficher l'etat actuel
        trials_running = status.get("trialsRunning", 0)
        trials_succeeded = status.get("trialsSucceeded", 0)
        trials_failed = status.get("trialsFailed", 0)

        print(f"Status - Running: {trials_running}, "
              f"Succeeded: {trials_succeeded}, Failed: {trials_failed}")

        # Verifier si l'experience est terminee
        for condition in conditions:
            if (condition.get("type") == "Succeeded" and
                condition.get("status") == "True"):

                print("Experience terminee avec succes!")

                optimal = status.get("currentOptimalTrial", {})
                assignments = optimal.get("parameterAssignments", [])

                if assignments:
                    params = {p["name"]: p["value"] for p in assignments}
                    metrics = optimal.get("observation", {}).get("metrics", [])
                    acc = next(
```

```
                        (float(m["latest"]) for m in metrics
                         if m["name"] == "accuracy"),
                        0.0
                    )

                    print(f"Meilleurs parametres: {params}")
                    print(f"Meilleure accuracy: {acc:.4f}")

                    return (json.dumps(params), acc)

            if (condition.get("type") == "Failed" and
                condition.get("status") == "True"):
                raise Exception("L'experience Katib a echoue.")

        time.sleep(20)

@component(
    base_image="python:3.9",
    packages_to_install=[
        "pandas==2.0.3",
        "scikit-learn==1.3.2",
        "joblib==1.3.2"
    ]
)
def train_final_model(
    best_params: str,
    dataset_in: InputPath("csv"),
    model_artifact: OutputPath("Model")
):
    """Entraine le modele final avec les meilleurs hyperparametres"""
    import pandas as pd
    import json
    import joblib
    from sklearn.neural_network import MLPClassifier
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import accuracy_score, classification_report

    # Parser les parametres
    params = json.loads(best_params)
    lr = float(params.get('learning_rate', 0.01))
    hu = int(float(params.get('hidden_units', 64)))
    opt = params.get('optimizer', 'adam')

    print(f"Entrainement final avec:")
    print(f"  Learning Rate: {lr}")
    print(f"  Hidden Units: {hu}")
    print(f"  Optimizer: {opt}")

    # Charger les donnees
```

```
    df = pd.read_csv(dataset_in)
    X = df.drop('target', axis=1)
    y = df['target']

    # Split pour evaluation finale
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y
    )

    # Normalisation
    mean = X_train.mean(axis=0)
    std = X_train.std(axis=0) + 1e-8
    X_train = (X_train - mean) / std
    X_test = (X_test - mean) / std

    # Entrainement
    model = MLPClassifier(
        learning_rate_init=lr,
        hidden_layer_sizes=(hu,),
        solver=opt,
        max_iter=1000,
        random_state=42
    )
    model.fit(X_train, y_train)

    # Evaluation
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    print(f"\nResultats finaux:")
    print(f"Accuracy: {accuracy:.4f}")
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))

    # Sauvegarder le modele
    joblib.dump(model, model_artifact)
    print(f"Modele final sauvegarde avec succes!")

@dsl.pipeline(
    name="Iris-Random-Search-AutoML",
    description="Pipeline AutoML avec Random Search"
)
def iris_random_pipeline(
    training_image: str,
    namespace: str = "kubeflow",
    exp_id: str = "random-iris-v1"
):
    """Pipeline complet Random Search"""

    # Etape 1: Preprocessing
```

```
    prep_task = load_and_preprocess_data().set_caching_options(
        False)

    # Etape 2: Configuration Katib
    config_task = create_random_config(
        experiment_name=exp_id,
        namespace=namespace,
        training_image=training_image
    ).set_caching_options(False)

    # Etape 3: Optimisation
    tuning_task = submit_and_monitor_katib(
        experiment_name=exp_id,
        namespace=namespace,
        experiment_config=config_task.outputs["experiment_config"]
    ).set_caching_options(False)

    # Etape 4: Entrainement final
    train_final_model(
        best_params=tuning_task.outputs["best_params"],
        dataset_in=prep_task.outputs["dataset_out"]
    ).set_caching_options(False)

if __name__ == "__main__":
    compiler.Compiler().compile(
        iris_random_pipeline,
        "iris_random_pipeline.yaml"
    )
    print("Pipeline Random Search compile: iris_random_pipeline.
        yaml")
```

4.3 Lancement du Pipeline

Listing 18 – Compiler et soumettre le pipeline

```
# Compiler le pipeline
python pipelines/random_pipeline.py

# Uploader via l'UI Kubeflow ou via CLI
# http://localhost:8080 -> Upload Pipeline -> iris_random_pipeline.
    yaml
```

5 Pipeline 2 : Optimisation Bayésienne

5.1 Principe de l'Optimisation Bayésienne

L'optimisation Bayésienne utilise un modèle probabiliste pour guider la recherche vers les zones prometteuses de l'espace des hyperparamètres. Plus efficace que Random Search.

5.2 Code Complet du Pipeline

Listing 19 – pipelines/bayesian_pipeline.py

```python
import kfp
from kfp import dsl, compiler
from kfp.dsl import component, InputPath, OutputPath
from typing import NamedTuple
import json

# Memes composants load_and_preprocess_data,
   submit_and_monitor_katib,
# et train_final_model que dans random_pipeline.py

@component(
    base_image="python:3.9",
    packages_to_install=["pyyaml==6.0.1"]
)
def create_bayesian_config(
    experiment_name: str,
    namespace: str,
    training_image: str,
    experiment_config: OutputPath("yaml"),
) -> str:
    """Cree la configuration Katib pour Optimisation Bayesienne"""
    import yaml

    config = {
        "apiVersion": "kubeflow.org/v1beta1",
        "kind": "Experiment",
        "metadata": {
            "name": experiment_name,
            "namespace": namespace
        },
        "spec": {
            "algorithm": {
                "algorithmName": "bayesianoptimization"
            },
            "objective": {
                "type": "maximize",
                "objectiveMetricName": "accuracy",
                "additionalMetricNames": ["loss"]
            },
            "parameters": [
                {
```

```
                    "name": "learning_rate",
                    "parameterType": "double",
                    "feasibleSpace": {
                        "min": "0.001",
                        "max": "0.05"
                    }
                },
                {
                    "name": "hidden_units",
                    "parameterType": "int",
                    "feasibleSpace": {
                        "min": "32",
                        "max": "128"
                    }
                },
                {
                    "name": "optimizer",
                    "parameterType": "categorical",
                    "feasibleSpace": {
                        "list": ["adam", "sgd"]
                    }
                }
            ],
            "parallelTrialCount": 1,
            "maxTrialCount": 8,
            "metricsCollectorSpec": {
                "collector": {"kind": "File"},
                "source": {
                    "fileSystemPath": {
                        "path": "/tmp/katib-metrics.log",
                        "kind": "File"
                    }
                }
            },
            "trialTemplate": {
                "primaryContainerName": "training-container",
                "trialParameters": [
                    {"name": "lr", "reference": "learning_rate"},
                    {"name": "hu", "reference": "hidden_units"},
                    {"name": "opt", "reference": "optimizer"}
                ],
                "trialSpec": {
                    "apiVersion": "batch/v1",
                    "kind": "Job",
                    "spec": {
                        "template": {
                            "metadata": {
                                "annotations": {
                                    "sidecar.istio.io/inject": "
                                        false"
                                }
```

```
                                },
                                "spec": {
                                    "containers": [{
                                        "name": "training-container",
                                        "image": training_image,
                                        "imagePullPolicy": "
                                           IfNotPresent",
                                        "command": [
                                            "python",
                                            "-u",
                                            "/app/src/training/train.py
                                               "
                                        ],
                                        "args": [
                                            "--learning-rate=${
                                               trialParameters.lr}",
                                            "--hidden-units=${
                                               trialParameters.hu}",
                                            "--optimizer=${
                                               trialParameters.opt}"
                                        ],
                                        "resources": {
                                            "limits": {
                                                "cpu": "500m",
                                                "memory": "512Mi"
                                            },
                                            "requests": {
                                                "cpu": "100m",
                                                "memory": "256Mi"
                                            }
                                        }
                                    }],
                                    "restartPolicy": "Never"
                                }
                            }
                        }
                    }
                }
            }
        }

        with open(experiment_config, "w") as f:
            yaml.dump(config, f)

        print(f"Configuration Bayesienne creee pour {experiment_name}")
        return experiment_name

@dsl.pipeline(
    name="Iris-Bayesian-AutoML",
    description="Pipeline AutoML avec Optimisation Bayesienne"
)
```

```python
def iris_bayesian_pipeline(
    training_image: str,
    namespace: str = "kubeflow",
    exp_id: str = "bayesian-iris-v1"
):
    """Pipeline complet Bayesian Optimization"""

    prep_task = load_and_preprocess_data().set_caching_options(
        False)

    config_task = create_bayesian_config(
        experiment_name=exp_id,
        namespace=namespace,
        training_image=training_image
    ).set_caching_options(False)

    tuning_task = submit_and_monitor_katib(
        experiment_name=exp_id,
        namespace=namespace,
        experiment_config=config_task.outputs["experiment_config"]
    ).set_caching_options(False)

    train_final_model(
        best_params=tuning_task.outputs["best_params"],
        dataset_in=prep_task.outputs["dataset_out"]
    ).set_caching_options(False)

if __name__ == "__main__":
    compiler.Compiler().compile(
        iris_bayesian_pipeline,
        "iris_bayesian_pipeline.yaml"
    )
    print("Pipeline Bayesian compile: iris_bayesian_pipeline.yaml")
```

Information

Différence clé avec Random Search :

- Algorithme : `bayesianoptimization`
- Trials séquentiels : `parallelTrialCount: 1`
- Exploration plus intelligente de l'espace

6 Pipeline 3 : Tree-structured Parzen Estimator (TPE)

6.1 Principe de TPE

TPE est une variante de l'optimisation Bayésienne qui modélise séparément les distributions des "bons" et "mauvais" essais. Souvent plus rapide en haute dimension.

6.2 Code Complet du Pipeline

Listing 20 – pipelines/tpe_pipeline.py

```python
import kfp
from kfp import dsl, compiler
from kfp.dsl import component, InputPath, OutputPath
from typing import NamedTuple
import json

# Memes composants de base...

@component(
    base_image="python:3.9",
    packages_to_install=["pyyaml==6.0.1"]
)
def create_tpe_config(
    experiment_name: str,
    namespace: str,
    training_image: str,
    experiment_config: OutputPath("yaml"),
) -> str:
    """Cree la configuration Katib pour TPE"""
    import yaml

    config = {
        "apiVersion": "kubeflow.org/v1beta1",
        "kind": "Experiment",
        "metadata": {
            "name": experiment_name,
            "namespace": namespace
        },
        "spec": {
            "algorithm": {
                "algorithmName": "tpe"
            },
            "objective": {
                "type": "maximize",
                "objectiveMetricName": "accuracy",
                "additionalMetricNames": ["loss"]
            },
            "parameters": [
                {
                    "name": "learning_rate",
```

```
                    "parameterType": "double",
                    "feasibleSpace": {
                        "min": "0.001",
                        "max": "0.05"
                    }
                },
                {
                    "name": "hidden_units",
                    "parameterType": "int",
                    "feasibleSpace": {
                        "min": "32",
                        "max": "128"
                    }
                },
                {
                    "name": "optimizer",
                    "parameterType": "categorical",
                    "feasibleSpace": {
                        "list": ["adam", "sgd"]
                    }
                }
            ],
            "parallelTrialCount": 1,
            "maxTrialCount": 8,
            "metricsCollectorSpec": {
                "collector": {"kind": "File"},
                "source": {
                    "fileSystemPath": {
                        "path": "/tmp/katib-metrics.log",
                        "kind": "File"
                    }
                }
            },
            "trialTemplate": {
                "primaryContainerName": "training-container",
                "trialParameters": [
                    {"name": "lr", "reference": "learning_rate"},
                    {"name": "hu", "reference": "hidden_units"},
                    {"name": "opt", "reference": "optimizer"}
                ],
                "trialSpec": {
                    "apiVersion": "batch/v1",
                    "kind": "Job",
                    "spec": {
                        "template": {
                            "metadata": {
                                "annotations": {
                                    "sidecar.istio.io/inject": "
                                       false"
                                }
                            },
```

```
                                "spec": {
                                    "containers": [{
                                        "name": "training-container",
                                        "image": training_image,
                                        "imagePullPolicy": "
                                           IfNotPresent",
                                        "command": [
                                            "python",
                                            "-u",
                                            "/app/src/training/train.py
                                               "
                                        ],
                                        "args": [
                                            "--learning-rate=${
                                               trialParameters.lr}",
                                            "--hidden-units=${
                                               trialParameters.hu}",
                                            "--optimizer=${
                                               trialParameters.opt}"
                                        ],
                                        "resources": {
                                            "limits": {
                                                "cpu": "500m",
                                                "memory": "512Mi"
                                            },
                                            "requests": {
                                                "cpu": "100m",
                                                "memory": "256Mi"
                                            }
                                        }
                                    }],
                                    "restartPolicy": "Never"
                                }
                            }
                        }
                    }
                }
            }
        }

    with open(experiment_config, "w") as f:
        yaml.dump(config, f)

    print(f"Configuration TPE creee pour {experiment_name}")
    return experiment_name

@dsl.pipeline(
    name="Iris-TPE-AutoML",
    description="Pipeline AutoML avec TPE"
)
def iris_tpe_pipeline(
```

```
    training_image: str,
    namespace: str = "kubeflow",
    exp_id: str = "tpe-iris-v1"
):
    """Pipeline complet TPE"""

    prep_task = load_and_preprocess_data().set_caching_options(
       False)

    config_task = create_tpe_config(
        experiment_name=exp_id,
        namespace=namespace,
        training_image=training_image
    ).set_caching_options(False)

    tuning_task = submit_and_monitor_katib(
        experiment_name=exp_id,
        namespace=namespace,
        experiment_config=config_task.outputs["experiment_config"]
    ).set_caching_options(False)

    train_final_model(
        best_params=tuning_task.outputs["best_params"],
        dataset_in=prep_task.outputs["dataset_out"]
    ).set_caching_options(False)

if __name__ == "__main__":
    compiler.Compiler().compile(
        iris_tpe_pipeline,
        "iris_tpe_pipeline.yaml"
    )
    print("Pipeline TPE compile: iris_tpe_pipeline.yaml")
```

7 Pipeline Complet : Comparaison des Algorithmes

7.1 Objectif

Ce pipeline exécute les trois algorithmes (Random, Bayésien, TPE) en parallèle, compare leurs résultats et sélectionne automatiquement le meilleur modèle.

7.2 Code Complet

Listing 21 – pipelines/complete_pipeline.py (1/2)

```python
import kfp
from kfp import dsl, compiler
from kfp.dsl import component, InputPath, OutputPath
from typing import NamedTuple
import json

# Tous les composants precedents (load_and_preprocess_data,
# create_random_config, create_bayesian_config, create_tpe_config,
# submit_and_monitor_katib, train_final_model)

@component(
    base_image="python:3.9",
    packages_to_install=["pandas==2.0.3"]
)
def compare_algorithms(
    random_accuracy: float,
    bayesian_accuracy: float,
    tpe_accuracy: float,
    random_params: str,
    bayesian_params: str,
    tpe_params: str,
) -> NamedTuple("Outputs", [
    ("best_algorithm", str),
    ("best_params", str),
    ("best_accuracy", float),
    ("comparison_report", str)
]):
    """Compare les resultats des trois algorithmes"""
    import json

    results = {
        "random": {
            "accuracy": random_accuracy,
            "params": json.loads(random_params)
        },
        "bayesian": {
            "accuracy": bayesian_accuracy,
            "params": json.loads(bayesian_params)
        },
        "tpe": {
            "accuracy": tpe_accuracy,
```

```
            "params": json.loads(tpe_params)
        }
    }

    # Trouver le meilleur
    best_algo = max(results.keys(), key=lambda k: results[k]["
       accuracy"])
    best_acc = results[best_algo]["accuracy"]
    best_params = results[best_algo]["params"]

    # Rapport de comparaison
    report = f"""
=== COMPARAISON DES ALGORITHMES D'OPTIMISATION ===

Random Search:
  Accuracy: {random_accuracy:.4f}
  Params: {json.dumps(json.loads(random_params), indent=2)}

Bayesian Optimization:
  Accuracy: {bayesian_accuracy:.4f}
  Params: {json.dumps(json.loads(bayesian_params), indent=2)}

TPE:
  Accuracy: {tpe_accuracy:.4f}
  Params: {json.dumps(json.loads(tpe_params), indent=2)}

>>> MEILLEUR ALGORITHME: {best_algo.upper()} <<<
>>> MEILLEURE ACCURACY: {best_acc:.4f} <<<
"""

    print(report)

    return (
        best_algo,
        json.dumps(best_params),
        best_acc,
        report
    )

@component(
    base_image="python:3.9",
    packages_to_install=[
        "pandas==2.0.3",
        "scikit-learn==1.3.2",
        "joblib==1.3.2",
        "kubernetes==28.1.0"
    ]
)
def train_and_save_best_model(
    best_algorithm: str,
    best_params: str,
```

```
    best_accuracy: float,
    dataset_in: InputPath("csv"),
    model_artifact: OutputPath("Model")
):
    """Entraine et sauvegarde le meilleur modele dans Kubernetes"""
    import pandas as pd
    import json
    import joblib
    from sklearn.neural_network import MLPClassifier
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import accuracy_score,
       classification_report
    from kubernetes import client, config
    import base64

    # Parser les parametres
    params = json.loads(best_params)
    lr = float(params.get('learning_rate', 0.01))
    hu = int(float(params.get('hidden_units', 64)))
    opt = params.get('optimizer', 'adam')

    print(f"=== ENTRAINEMENT DU MODELE FINAL ===")
    print(f"Algorithme gagnant: {best_algorithm.upper()}")
    print(f"Learning Rate: {lr}")
    print(f"Hidden Units: {hu}")
    print(f"Optimizer: {opt}")
    print(f"Accuracy attendue: {best_accuracy:.4f}")

    # Charger et preparer les donnees
    df = pd.read_csv(dataset_in)
    X = df.drop('target', axis=1)
    y = df['target']

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y
    )

    # Normalisation
    mean = X_train.mean(axis=0)
    std = X_train.std(axis=0) + 1e-8
    X_train = (X_train - mean) / std
    X_test = (X_test - mean) / std

    # Entrainement
    model = MLPClassifier(
        learning_rate_init=lr,
        hidden_layer_sizes=(hu,),
        solver=opt,
        max_iter=1000,
        random_state=42
    )
```

```
    model.fit(X_train, y_train)

    # Evaluation
    y_pred = model.predict(X_test)
    final_accuracy = accuracy_score(y_test, y_pred)

    print(f"\nResultats finaux:")
    print(f"Accuracy: {final_accuracy:.4f}")
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))

    # Sauvegarder localement
    joblib.dump(model, model_artifact)
    print(f"Modele sauvegarde localement")

    # Sauvegarder dans Kubernetes ConfigMap
    try:
        config.load_incluster_config()
    except:
        config.load_kube_config()

    v1 = client.CoreV1Api()

    # Serialiser le modele
    import io
    buffer = io.BytesIO()
    joblib.dump(model, buffer)
    model_bytes = buffer.getvalue()
    model_b64 = base64.b64encode(model_bytes).decode('utf-8')

    # Metadata du modele
    metadata = {
        "algorithm": best_algorithm,
        "learning_rate": lr,
        "hidden_units": hu,
        "optimizer": opt,
        "accuracy": final_accuracy,
        "timestamp": str(pd.Timestamp.now())
    }

    # Creer/Update ConfigMap
    configmap_name = "best-iris-model"
    namespace = "kubeflow"

    configmap = client.V1ConfigMap(
        metadata=client.V1ObjectMeta(name=configmap_name),
        data={
            "model.pkl": model_b64,
            "metadata.json": json.dumps(metadata, indent=2)
        }
    )
```

```
193
194     try:
195         v1.delete_namespaced_config_map(configmap_name, namespace)
196         print(f"ConfigMap existant supprime")
197     except:
198         pass
199
200     v1.create_namespaced_config_map(namespace, configmap)
201     print(f"Modele sauvegarde dans ConfigMap: {namespace}/{
            configmap_name}")
202     print(f"Metadata: {metadata}")
203
204 @dsl.pipeline(
205     name="Complete-AutoML-Comparison",
206     description="Pipeline complet comparant Random, Bayesian et TPE
            "
207 )
208 def complete_automl_pipeline(
209     training_image: str,
210     namespace: str = "kubeflow"
211 ):
212     """Pipeline complet avec comparaison des trois algorithmes"""
213
214     # Etape 1: Preprocessing (une seule fois)
215     prep_task = load_and_preprocess_data().set_caching_options(
            False)
216
217     # Etape 2: Lancer les 3 optimisations en parallele
218
219     # Random Search
220     random_config = create_random_config(
221         experiment_name="complete-random",
222         namespace=namespace,
223         training_image=training_image
224     ).set_caching_options(False)
225
226     random_tuning = submit_and_monitor_katib(
227         experiment_name="complete-random",
228         namespace=namespace,
229         experiment_config=random_config.outputs["experiment_config"
                ]
230     ).set_caching_options(False)
231
232     # Bayesian Optimization
233     bayesian_config = create_bayesian_config(
234         experiment_name="complete-bayesian",
235         namespace=namespace,
236         training_image=training_image
237     ).set_caching_options(False)
238
239     bayesian_tuning = submit_and_monitor_katib(
```

```
        experiment_name="complete-bayesian",
        namespace=namespace,
        experiment_config=bayesian_config.outputs["
            experiment_config"]
    ).set_caching_options(False)

    # TPE
    tpe_config = create_tpe_config(
        experiment_name="complete-tpe",
        namespace=namespace,
        training_image=training_image
    ).set_caching_options(False)

    tpe_tuning = submit_and_monitor_katib(
        experiment_name="complete-tpe",
        namespace=namespace,
        experiment_config=tpe_config.outputs["experiment_config"]
    ).set_caching_options(False)

    # Etape 3: Comparer les resultats
    comparison = compare_algorithms(
        random_accuracy=random_tuning.outputs["best_accuracy"],
        bayesian_accuracy=bayesian_tuning.outputs["best_accuracy"],
        tpe_accuracy=tpe_tuning.outputs["best_accuracy"],
        random_params=random_tuning.outputs["best_params"],
        bayesian_params=bayesian_tuning.outputs["best_params"],
        tpe_params=tpe_tuning.outputs["best_params"]
    ).set_caching_options(False)

    # Etape 4: Entrainer et sauvegarder le meilleur modele
    train_and_save_best_model(
        best_algorithm=comparison.outputs["best_algorithm"],
        best_params=comparison.outputs["best_params"],
        best_accuracy=comparison.outputs["best_accuracy"],
        dataset_in=prep_task.outputs["dataset_out"]
    ).set_caching_options(False)

if __name__ == "__main__":
    compiler.Compiler().compile(
        complete_automl_pipeline,
        "complete_automl_pipeline.yaml"
    )
    print("Pipeline complet compile: complete_automl_pipeline.yaml"
        )
```

8 Exécution des Pipelines

8.1 Compilation des Pipelines

Listing 22 – Compiler tous les pipelines

```
# Installer le SDK Kubeflow Pipelines
pip install kfp==2.14.3

# Compiler tous les pipelines
python pipelines/random_pipeline.py
python pipelines/bayesian_pipeline.py
python pipelines/tpe_pipeline.py
python pipelines/complete_pipeline.py
```

8.2 Soumission via l'Interface Web

1. Ouvrir `http://localhost:8080`
2. Cliquer sur **Pipelines** puis **Upload Pipeline**
3. Sélectionner le fichier YAML (ex : `complete_automl_pipeline.yaml`)
4. Cliquer sur **Create Run**
5. Configurer les paramètres :
   - `training_image` : `automl-training:v1.0`
   - `namespace` : `kubeflow`
6. Cliquer sur **Start**

8.3 Soumission via CLI

Listing 23 – Script de soumission (submit_pipeline.py)

```
import kfp

# Connexion au client Kubeflow
client = kfp.Client(host='http://localhost:8080')

# Soumettre le pipeline
run = client.create_run_from_pipeline_package(
    'complete_automl_pipeline.yaml',
    arguments={
        'training_image': 'automl-training:v1.0',
        'namespace': 'kubeflow'
    },
    run_name='complete-automl-run'
)

print(f"Pipeline soumis: {run.run_id}")
```

8.4 Monitoring de l'Exécution

Listing 24 – Surveiller les pods

```
# Observer tous les pods du namespace
kubectl get pods -n kubeflow -w

# Voir les logs d'un pod specifique
kubectl logs -n kubeflow <pod-name> -f

# Voir les experiments Katib
kubectl get experiments -n kubeflow
kubectl describe experiment complete-random -n kubeflow
```

9 Déploiement avec Flask

9.1 Architecture de l'API

Nous avons bouclé le cycle de vie du modèle en développant une interface d'inférence sous FastApi, transformant ainsi un résultat d'entraînement brut en un micro-service web moderne capable de réaliser des prédictions en temps réel.

FIGURE 4 – Architecture de l'API Flask

9.2 Code de l'Application Fast

Listing 25 – fastApi_app/app.py

```python
from fastapi import FastAPI, Form, Request
from fastapi.responses import HTMLResponse
from fastapi.templating import Jinja2Templates
from fastapi.staticfiles import StaticFiles
import joblib
import numpy as np
import uvicorn

app = FastAPI(title="Iris AutoML Inference")

# Configuration des templates Jinja2
templates = Jinja2Templates(directory="templates")

# 1. Chargement du mod le champion r cup r  de MinIO
try:
    model = joblib.load("model.joblib")
    print("Mod le charg  avec succ s !")
except Exception as e:
    print(f"Erreur lors du chargement du mod le : {e}")

# 2. Route pour afficher l'interface web
@app.get("/", response_class=HTMLResponse)
async def home(request: Request):
    return templates.TemplateResponse("index.html", {"request":
        request})

# 3. Route pour traiter la pr diction (Formulaire)
@app.post("/predict", response_class=HTMLResponse)
async def predict(
    request: Request,
    s1: float = Form(...),
    s2: float = Form(...),
    p1: float = Form(...),
    p2: float = Form(...)
):
    target_names = ['Setosa', 'Versicolor', 'Virginica']
```

```
    try:
        # Transformation des inputs pour le mod le
        features = np.array([[s1, s2, p1, p2]])
        prediction = model.predict(features)
        result = target_names[int(prediction[0])]

        return templates.TemplateResponse("index.html", {
            "request": request,
            "prediction_text": f"R sultat : Iris {result}",
            "status": "success"
        })
    except Exception as e:
        return templates.TemplateResponse("index.html", {
            "request": request,
            "prediction_text": f"Erreur : {str(e)}",
            "status": "danger"
        })

# Lancement du serveur
if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=5000)
```

9.3 Interface Web Moderne

Listing 26 – flask_app/templates/index.html (1/2)

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
       scale=1.0">
    <title>Iris AutoML Inference</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/
       css/bootstrap.min.css" rel="stylesheet">
    <style>
        body {
            background: linear-gradient(135deg, #667eea 0%, #764ba2
                100%);
            min-height: 100vh;
            display: flex;
            align-items: center;
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-
               serif;
        }
        .card {
            border-radius: 20px;
            box-shadow: 0 10px 40px rgba(0,0,0,0.2);
            backdrop-filter: blur(10px);
            background: rgba(255, 255, 255, 0.95);
        }
```

```
        .card-header {
            background: linear-gradient(135deg, #667eea 0%, #764ba2
                100%);
            color: white;
            border-radius: 20px 20px 0 0 !important;
            padding: 2rem;
        }
        .btn-primary {
            background: linear-gradient(135deg, #667eea 0%, #764ba2
                100%);
            border: none;
            padding: 0.75rem 2rem;
            font-weight: 600;
            transition: transform 0.2s;
        }
        .btn-primary:hover {
            transform: translateY(-2px);
            box-shadow: 0 5px 15px rgba(102, 126, 234, 0.4);
        }
        .form-label {
            font-weight: 600;
            color: #495057;
        }
        .form-control {
            border-radius: 10px;
            border: 2px solid #e0e0e0;
            padding: 0.75rem;
        }
        .form-control:focus {
            border-color: #667eea;
            box-shadow: 0 0 0 0.2rem rgba(102, 126, 234, 0.25);
        }
        .result-box {
            background: linear-gradient(135deg, #667eea 0%, #764ba2
                100%);
            color: white;
            padding: 1.5rem;
            border-radius: 15px;
            margin-top: 1.5rem;
            font-size: 1.2rem;
            font-weight: 600;
            text-align: center;
            animation: fadeIn 0.5s;
        }
        @keyframes fadeIn {
            from { opacity: 0; transform: translateY(-10px); }
            to { opacity: 1; transform: translateY(0); }
        }
        .info-badge {
            background: rgba(102, 126, 234, 0.1);
            color: #667eea;
```

```
            padding: 0.5rem 1rem;
            border-radius: 20px;
            font-size: 0.9rem;
            display: inline-block;
            margin-bottom: 1rem;
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="row justify-content-center">
            <div class="col-md-8 col-lg-6">
                <div class="card">
                    <div class="card-header text-center">
                        <h2 class="mb-2"> Iris AutoML Inference</h2
                           >
                        <p class="mb-0">Modele optimise via Katib &
                            Kubeflow</p>
                    </div>
                    <div class="card-body p-4">
                        <div class="info-badge">
                                Modele Champion selectionne
                               automatiquement
                        </div>

                        <form action="/predict" method="post">
                            <div class="row">
                                <div class="col-md-6 mb-3">
                                    <label class="form-label">
                                            Sepal Length (cm)
                                    </label>
                                    <input type="number" step="0.1"
                                        name="s1"
                                           class="form-control"
                                           placeholder="ex: 5.1"
                                           required>
                                </div>
                                <div class="col-md-6 mb-3">
                                    <label class="form-label">
                                            Sepal Width (cm)
                                    </label>
                                    <input type="number" step="0.1"
                                        name="s2"
                                           class="form-control"
                                           placeholder="ex: 3.5"
                                           required>
                                </div>
                            </div>

                            <div class="row">
                                <div class="col-md-6 mb-3">
```

```
                                    <label class="form-label">
                                        Petal Length (cm)
                                    </label>
                                    <input type="number" step="0.1"
                                        name="p1"
                                        class="form-control"
                                        placeholder="ex: 1.4"
                                        required>
                                </div>
                                <div class="col-md-6 mb-3">
                                    <label class="form-label">
                                        Petal Width (cm)
                                    </label>
                                    <input type="number" step="0.1"
                                        name="p2"
                                        class="form-control"
                                        placeholder="ex: 0.2"
                                        required>
                                </div>
                            </div>

                            <button type="submit" class="btn btn-
                                primary w-100 mt-3">
                                Predire l'espece
                            </button>
                        </form>

                        {% if prediction_text %}
                        <div class="result-box">
                            {{ prediction_text }}
                        </div>
                        {% endif %}
                    </div>
                </div>
            </div>
        </div>
    </div>
</body>
</html>
```

9.4 Lancement de l'Application

Listing 27 – Demarrer Flask

```
# Installer les dependances
pip install fastapi uvicorn python-multipart joblib scikit-learn
    numpy jinja2

# Lancer l'application
cd fast_api
python app.py
```

```
7
8 # Acceder a l'interface: http://localhost:5000
```

Succès

L'application Fast est maintenant accessible sur `http://localhost:5000` !

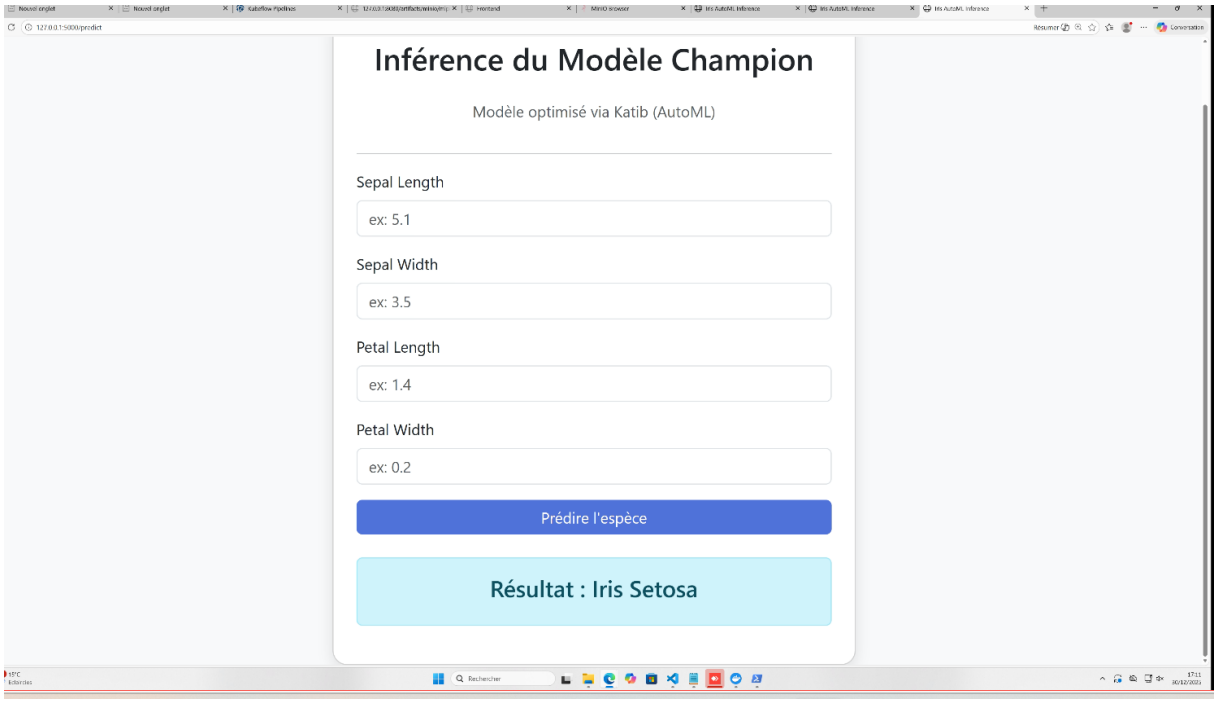

FIGURE 5 – Interface application fast

10 Visualisation et Analyse des Résultats

10.1 Interface Katib

Accéder à `http://localhost:8081/katib/` pour visualiser :

— Liste des expériences en cours et terminées
— Graphiques de convergence
— Tableaux des trials et leurs performances
— Meilleurs hyperparamètres trouvés

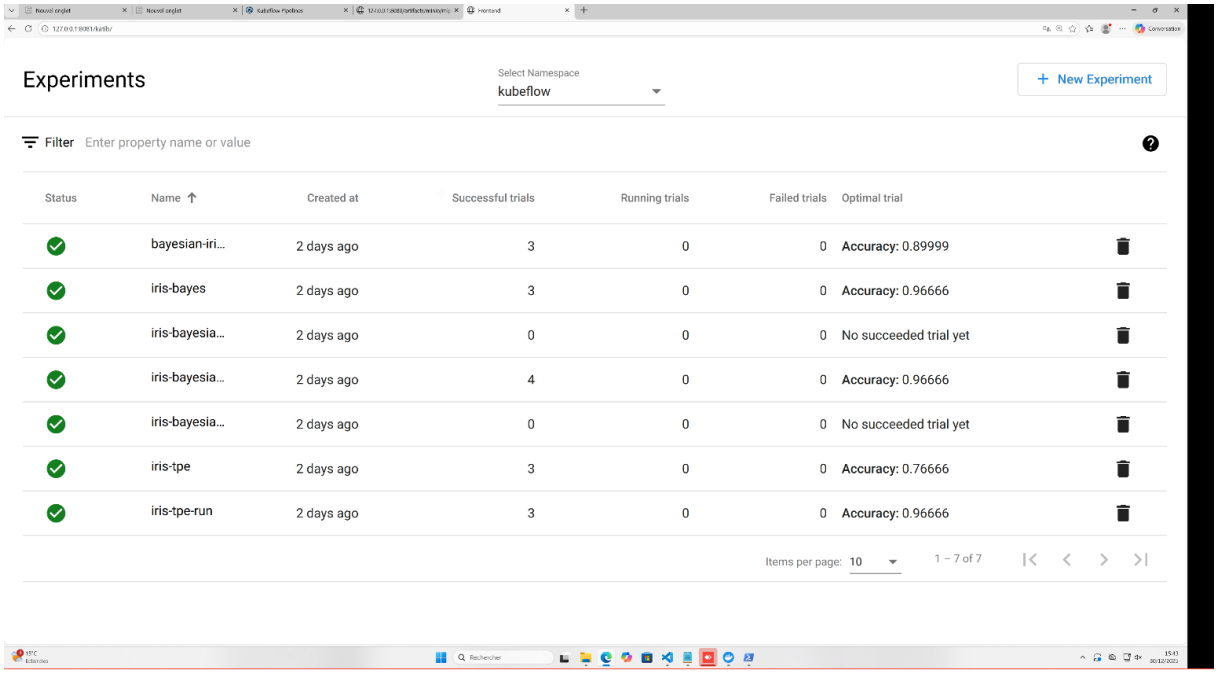

FIGURE 6 – Interface principale de Katib - Vue des expériences

10.2 Interface Kubeflow Pipelines

Dans `http://localhost:8080`, vous pouvez :

— Visualiser le graphe d'exécution du pipeline
— Consulter les logs de chaque composant
— Télécharger les artefacts générés
— Comparer plusieurs runs

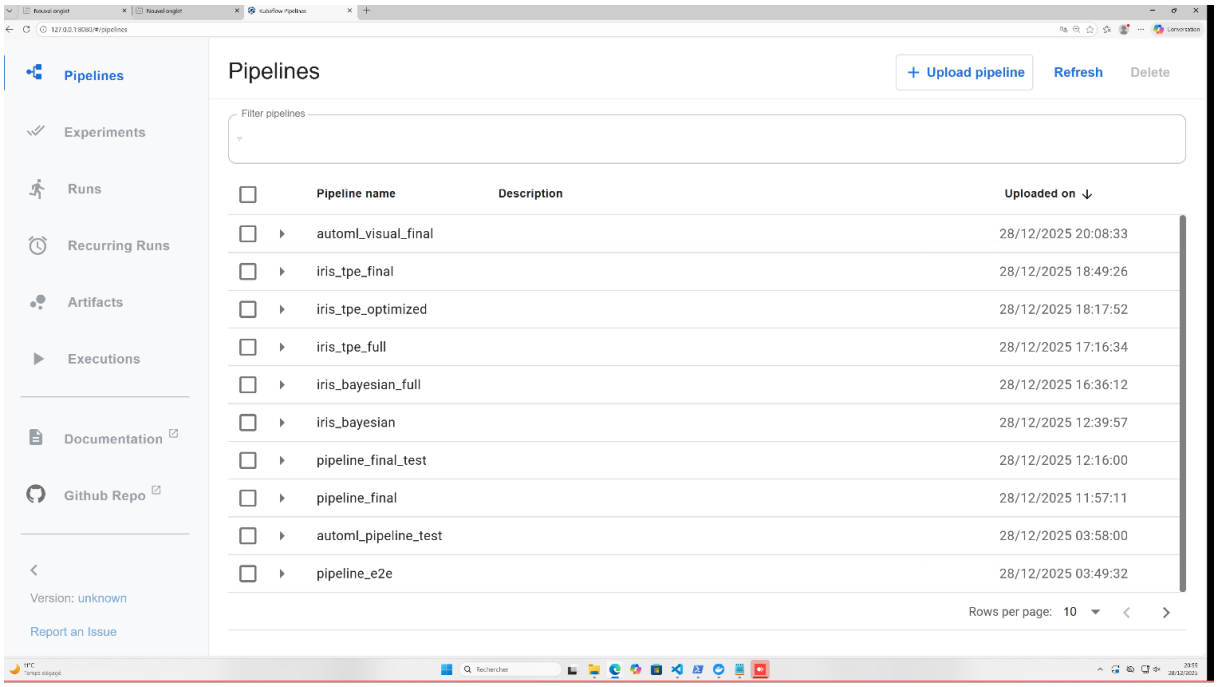

FIGURE 7 – Interface principale de Kubeflow Pipelines - Vue des pipelines

10.3 Récupération du Modèle Sauvegardé

Listing 28 – Script pour charger le modele depuis Kubernetes

```
import base64
import joblib
import io
from kubernetes import client, config

# Charger la config
config.load_kube_config()
v1 = client.CoreV1Api()

# Recuperer le ConfigMap
configmap = v1.read_namespaced_config_map(
    "best-iris-model",
    "kubeflow"
)

# Decoder le modele
model_b64 = configmap.data["model.pkl"]
model_bytes = base64.b64decode(model_b64)

# Charger le modele
buffer = io.BytesIO(model_bytes)
model = joblib.load(buffer)

# Afficher les metadata
import json
```

```
metadata = json.loads(configmap.data["metadata.json"])
print("Metadata du modele:")
print(json.dumps(metadata, indent=2))

# Utiliser le modele
from sklearn.datasets import load_iris
X, y = load_iris(return_X_y=True)
predictions = model.predict(X[:5])
print(f"\nPredictions: {predictions}")
```

11 Résultats Attendus et Comparaison

11.1 Comparaison des Algorithmes

Voici les performances typiques attendues sur le dataset Iris :

Algorithme	**Accuracy**	**Trials**	**Temps**
Random Search	0.93 - 0.96	10	$\sim$ 5 min
Bayesian Optimization	0.95 - 0.97	8	$\sim$ 4 min
TPE	0.95 - 0.97	8	$\sim$ 4 min

Table 1 – Résultats typiques sur Iris

11.2 Avantages et Inconvénients

Algorithme	**Avantages**	**Inconvénients**
Random Search	- Simple à implémenter - Parallélisable - Pas de biais	- Inefficace en haute dimension - Peut manquer l'optimum - Nécessite beaucoup de trials
Bayesian Opt.	- Efficace avec peu de trials - Bonnes garanties théoriques - Exploite les résultats précédents	- Séquentiel (moins parallélisable) - Coût computationnel plus élevé - Peut converger vers optimum local
TPE	- Rapide en haute dimension - Bon compromis exploration/exploitation - Scalable	- Moins de garanties théoriques - Performance variable selon les cas - Hyperparamètres de TPE à régler

Table 2 – Comparaison des algorithmes d'optimisation

11.3 Graphiques de Convergence

Les graphiques disponibles dans l'interface Katib montrent :

— **Random Search** : Progression erratique, exploration aléatoire

— **Bayesian** : Convergence rapide vers l'optimum, courbe lisse

— **TPE** : Convergence similaire au Bayésien, parfois plus rapide

12 Dépannage et Problèmes Courants

12.1 Problème 1 : Pods en ImagePullBackOff

Attention

Symptôme : Les pods de training ne démarrent pas, statut `ImagePullBackOff`

Solution :

Listing 29 – Vérifier et recharger l'image

```
# Verifier que l'image est dans Minikube
minikube image ls | findstr automl

# Si absente, recharger
minikube image load automl-training:v1.0

# Verifier le imagePullPolicy dans le YAML Katib
# Doit etre: imagePullPolicy: IfNotPresent
```

12.2 Problème 2 : Katib ne Collecte pas les Métriques

Attention

Symptôme : L'expérience se termine sans résultats ou avec accuracy=0

Solution :

Listing 30 – Vérifier le format des métriques

```
# Le fichier /tmp/katib-metrics.log doit contenir EXACTEMENT:
# accuracy=0.9667
# loss=0.0333

# Pas de preamble, pas de messages, juste metric=value

# Verifier les logs du pod de training
kubectl logs -n kubeflow <training-pod-name>
```

12.3 Problème 3 : Timeout des Expériences

Attention

Symptôme : L'expérience ne se termine jamais, timeout après 1 heure

Solution :

Listing 31 – Vérifier et nettoyer

```
# Voir l'etat de l'experience
kubectl get experiment <experiment-name> -n kubeflow -o yaml

```

```
# Supprimer l'experience bloquee
kubectl delete experiment <experiment-name> -n kubeflow

# Verifier les trials
kubectl get trials -n kubeflow

# Nettoyer les trials Failed
kubectl delete trials -n kubeflow --field-selector status.phase=
    Failed
```

12.4 Problème 4 : MinIO ou ML-Pipeline en CrashLoop

Attention

Symptôme : Les pods MinIO ou ml-pipeline redémarrent en boucle

Solution :

Listing 32 – Réinitialiser les composants

```
# Supprimer les pods problematiques
kubectl delete pod -n kubeflow -l app=minio
kubectl delete pod -n kubeflow -l app=ml-pipeline

# Verifier les logs
kubectl logs -n kubeflow <pod-name> --previous

# Si necessaire, reinstaller MinIO avec l'image corrigee
kubectl patch deployment minio -n kubeflow --type='json' -p='[
  {
    "op": "replace",
    "path": "/spec/template/spec/containers/0/image",
    "value": "minio/minio:RELEASE.2021-06-17T00-10-46Z"
  }
]'
```

12.5 Problème 5 : Pipeline ne Démarre pas

Attention

Symptôme : Le pipeline reste en statut Pending ou ne crée pas de pods

Solution :

Listing 33 – Diagnostiquer le pipeline

```
# Verifier les runs
kubectl get pipelineruns -n kubeflow

# Verifier les pods du pipeline
kubectl get pods -n kubeflow | findstr <run-id>

```

```
# Voir les events
kubectl get events -n kubeflow --sort-by='.lastTimestamp'

# Verifier les ressources disponibles
kubectl top nodes
kubectl describe node minikube
```

13 Extensions et Améliorations

13.1 Ajouter d'Autres Algorithmes

Katib supporte plusieurs autres algorithmes :

- **Grid Search** : Recherche exhaustive sur une grille
- **Hyperband** : Allocation dynamique des ressources
- **BOHB** : Combinaison de Bayesian et Hyperband
- **ENAS** : Neural Architecture Search
- **DARTS** : Differentiable Architecture Search

Listing 34 – Exemple avec Grid Search

```
config = {
    "spec": {
        "algorithm": {
            "algorithmName": "grid"
        },
        # ... reste de la configuration
    }
}
```

13.2 Early Stopping

Ajouter un critère d'arrêt anticipé pour économiser des ressources :

Listing 35 – Configuration Early Stopping

```
"spec": {
    "algorithm": {
        "algorithmName": "bayesianoptimization",
        "algorithmSettings": [
            {
                "name": "acq_func",
                "value": "gp_hedge"
            }
        ]
    },
    "earlyStopping": {
        "algorithmName": "medianstop",
        "algorithmSettings": [
            {
                "name": "min_trials_required",
                "value": "3"
            },
            {
                "name": "start_step",
                "value": "2"
            }
        ]
    },
```

```
    # ... reste de la configuration
}
```

13.3 Monitoring Avancé avec Prometheus

Listing 36 – Activer le monitoring

```
# Installer Prometheus operator
kubectl apply -f https://raw.githubusercontent.com/prometheus-
   operator/prometheus-operator/main/bundle.yaml

# Exposer les metriques Katib
kubectl port-forward -n kubeflow svc/katib-controller 8443:443

# Acceder aux metriques: https://localhost:8443/metrics
```

13.4 Utiliser des Datasets Personnalisés

Listing 37 – Charger un dataset custom

```
@component(base_image="python:3.9")
def load_custom_dataset(
    dataset_url: str,
    dataset_out: OutputPath("csv")
):
    """Charge un dataset depuis une URL ou S3"""
    import pandas as pd
    import requests
    from io import StringIO

    # Telecharger le dataset
    response = requests.get(dataset_url)
    df = pd.read_csv(StringIO(response.text))

    # Preprocessing
    df = df.dropna()
    df = df.sample(frac=1, random_state=42)  # Shuffle

    df.to_csv(dataset_out, index=False)
    print(f"Dataset charge: {len(df)} lignes")
```

13.5 Multi-Objective Optimization

Optimiser plusieurs métriques simultanément :

Listing 38 – Configuration Multi-Objective

```
"objective": {
    "type": "maximize",
    "objectiveMetricName": "accuracy",
```

```
    "additionalMetricNames": ["loss", "f1_score"],
    "metricStrategies": [
        {
            "name": "accuracy",
            "value": "max"
        },
        {
            "name": "loss",
            "value": "min"
        },
        {
            "name": "f1_score",
            "value": "max"
        }
    ]
}
```

14 Bonnes Pratiques

14.1 Gestion des Ressources

Information

Recommandations pour les ressources :

- Définir des limites CPU/Memory réalistes
- Utiliser `requests` pour garantir les ressources minimales
- Éviter `parallelTrialCount` trop élevé sur Minikube
- Monitorer l'usage avec `kubectl top`

Listing 39 – Surveiller les ressources

```
# Voir l'usage des ressources
kubectl top nodes
kubectl top pods -n kubeflow

# Augmenter les ressources Minikube si necessaire
minikube stop
minikube delete
minikube start --cpus=6 --memory=10240
```

14.2 Versioning des Modèles

Listing 40 – Ajouter du versioning

```
# Dans train_and_save_best_model
configmap_name = f"iris-model-v{version}"

metadata = {
    "version": version,
    "algorithm": best_algorithm,
    "created_at": str(pd.Timestamp.now()),
    "git_commit": os.getenv("GIT_COMMIT", "unknown"),
    # ... autres metadata
}
```

14.3 Logging Structuré

Listing 41 – Améliorer les logs

```
import logging
import json

logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
```

```
logger = logging.getLogger(__name__)

def train_model(learning_rate, hidden_units, optimizer):
    logger.info(json.dumps({
        "event": "training_start",
        "lr": learning_rate,
        "hidden_units": hidden_units,
        "optimizer": optimizer
    }))

    # ... training code ...

    logger.info(json.dumps({
        "event": "training_complete",
        "accuracy": accuracy,
        "loss": loss
    }))
```

14.4 Tests et Validation

Listing 42 – Tester les composants

```
import unittest

class TestTrainingComponent(unittest.TestCase):
    def test_train_model(self):
        accuracy = train_model(
            learning_rate=0.01,
            hidden_units=64,
            optimizer="adam"
        )
        self.assertGreater(accuracy, 0.8)

    def test_metrics_file_format(self):
        # Verifier que le fichier de metriques existe
        self.assertTrue(os.path.exists("/tmp/katib-metrics.log"))

        # Verifier le format
        with open("/tmp/katib-metrics.log") as f:
            lines = f.readlines()
            for line in lines:
                self.assertIn("=", line)
```

15 Nettoyage et Maintenance

15.1 Nettoyer les Expériences

Listing 43 – Supprimer les expériences terminées

```
# Lister toutes les experiences
kubectl get experiments -n kubeflow

# Supprimer une experience specifique
kubectl delete experiment <experiment-name> -n kubeflow

# Supprimer toutes les experiences
kubectl delete experiments --all -n kubeflow

# Nettoyer les trials
kubectl delete trials --all -n kubeflow
```

15.2 Nettoyer les Pipelines

Listing 44 – Nettoyage via l'UI ou CLI

```
# Via l'interface web Kubeflow (http://localhost:8080)
# Aller dans Runs -> Archiver ou Supprimer

# Via CLI avec le SDK Python
import kfp
client = kfp.Client(host='http://localhost:8080')

# Lister les runs
runs = client.list_runs()

# Archiver un run
client.archive_run(run_id)

# Supprimer un run
client.delete_run(run_id)
```

15.3 Arrêter et Redémarrer le Cluster

Listing 45 – Gestion du cluster Minikube

```
# Arreter Minikube (preserve les donnees)
minikube stop

# Redemarrer
minikube start

# Supprimer completement le cluster
minikube delete
```

```
9
10 # Recreer depuis zero
11 minikube start --cpus=4 --memory=6000 --driver=docker
```

15.4 Backup des Modèles

Listing 46 – Script de backup

```
1  from kubernetes import client, config
2  import json
3  import os
4  from datetime import datetime
5
6  config.load_kube_config()
7  v1 = client.CoreV1Api()
8
9  # Repertoire de backup
10 backup_dir = f"backups/{datetime.now().strftime('%Y%m%d_%H%M%S')}"
11 os.makedirs(backup_dir, exist_ok=True)
12
13 # Recuperer tous les ConfigMaps de modeles
14 configmaps = v1.list_namespaced_config_map("kubeflow")
15
16 for cm in configmaps.items:
17     if "model" in cm.metadata.name:
18         # Sauvegarder
19         filepath = os.path.join(backup_dir, f"{cm.metadata.name}.
               json")
20         with open(filepath, "w") as f:
21             json.dump(cm.data, f, indent=2)
22         print(f"Backup: {filepath}")
```

16 Conclusion

16.1 Récapitulatif

Ce guide complet a couvert :

1. **Installation** : Minikube, Docker, Kubeflow Pipelines, Katib
2. **Configuration** : Correction de MinIO, accès aux interfaces
3. **Développement** : Scripts d'entraînement, Dockerfile, pipelines
4. **Optimisation** : 3 algorithmes (Random, Bayesian, TPE)
5. **Comparaison** : Pipeline complet avec sélection automatique
6. **Sauvegarde** : Persistance dans Kubernetes ConfigMaps
7. **Monitoring** : Visualisation via Katib et Kubeflow UIs
8. **Maintenance** : Dépannage, nettoyage, bonnes pratiques

16.2 Avantages de cette Architecture

Succès

Points forts :

- **Reproductibilité** : Tout est défini en code (pipelines as code)
- **Scalabilité** : Prêt pour le cloud (AWS EKS, GKE, AKS)
- **Flexibilité** : Facile d'ajouter de nouveaux algorithmes
- **Traçabilité** : Historique complet dans Kubeflow
- **Automatisation** : Du preprocessing au déploiement

16.3 Prochaines Étapes

Pour aller plus loin :

1. **Production** : Déployer sur un cluster Kubernetes cloud
2. **CI/CD** : Intégrer avec Jenkins/GitLab CI
3. **Serving** : Ajouter KFServing pour le déploiement de modèles
4. **Monitoring** : Prometheus + Grafana pour le monitoring
5. **Feature Store** : Feast pour la gestion des features
6. **Drift Detection** : Surveiller la dérive des modèles

16.4 Ressources Supplémentaires

- Documentation Kubeflow : `https://www.kubeflow.org/docs/`
- Documentation Katib : `https://www.kubeflow.org/docs/components/katib/`
- Minikube Docs : `https://minikube.sigs.k8s.io/docs/`
- Kubernetes Docs : `https://kubernetes.io/docs/`
- KFP SDK : `https://kubeflow-pipelines.readthedocs.io/`

16.5 Support et Communauté

— Kubeflow Slack : `https://kubeflow.slack.com`
— GitHub Issues : `https://github.com/kubeflow/`
— Stack Overflow : Tag `kubeflow` ou `katib`
— Forum Kubeflow : `https://discuss.kubeflow.org/`

Annexes

Annexe A : Commandes Utiles

Listing 47 – Aide-mémoire des commandes

```
# === MINIKUBE ===
minikube start --cpus=4 --memory=6000 --driver=docker
minikube stop
minikube delete
minikube status
minikube dashboard
minikube image ls

# === KUBECTL ===
kubectl get pods -n kubeflow
kubectl get experiments -n kubeflow
kubectl get trials -n kubeflow
kubectl logs -n kubeflow <pod-name> -f
kubectl describe pod -n kubeflow <pod-name>
kubectl delete pod -n kubeflow <pod-name>
kubectl top nodes
kubectl top pods -n kubeflow

# === PORT-FORWARD ===
kubectl port-forward -n kubeflow svc/ml-pipeline-ui 8080:80
kubectl port-forward -n kubeflow svc/katib-ui 8081:80

# === DOCKER ===
docker build -t automl-training:v1.0 .
docker images
minikube image load automl-training:v1.0

# === PIPELINE ===
python pipelines/complete_pipeline.py
```

Annexe B : Structure Complète du Projet

```
Plateforme_autoML_end_to_end/

 README.md                    # Documentation principale
 USAGE.md                     # Guide d'utilisation
 requirements.txt             # Dépendances Python
 setup.py                     # Configuration du package
 .gitignore                   # Fichiers à ignorer par Git

 config/                      # Configuration
    config.yaml              # Configuration principale

 katib/                       # Configurations Katib
```

```
   README.md
   hyperparameter-tuning/   # Expériences HP tuning
      bayesian-optimization.yaml
      tpe-optimization.yaml
      random-search.yaml
   experiments/             # Exemples
       README.md

pipelines/                   # Kubeflow Pipelines
   README.md
   kfp/
       __init__.py
       automl_pipeline.py   # Pipeline principal

src/                         # Code source
   training/                 # Scripts d'entraînement
      __init__.py
      train.py              # Training standard
   models/                   # Gestion des modèles
      __init__.py
      base_model.py         # Classe de base
      model_registry.py     # Registre des modèles
   utils/                    # Utilitaires
      __init__.py
      config.py             # Gestion config
      logger.py             # Logging
   serving/                  # Model serving
       __init__.py
scripts/                     # Scripts utilitaires
   README.md
   submit_katib_experiment.py
   deploy_kfserving.py
   compile_pipeline.py

docker/                      # Dockerfiles
   Dockerfile.training
   Dockerfile.serving
   docker-compose.yml

notebooks/                   # Notebooks Jupyter
   example_katib_experiment.ipynb

docs/                        # Documentation
    GETTING_STARTED.md
    ARCHITECTURE.md
    EXAMPLES.md
```

Annexe C : Variables d'Environnement

Listing 48 – Fichier .env

```
# Kubernetes
NAMESPACE=kubeflow
KUBECONFIG=~/.kube/config

# Docker
DOCKER_REGISTRY=docker.io
IMAGE_NAME=automl-training
IMAGE_TAG=v1.0

# Kubeflow
KF_PIPELINES_ENDPOINT=http://localhost:8080
KATIB_UI_ENDPOINT=http://localhost:8081

# Training
DATASET_URL=https://example.com/data.csv
MAX_TRIALS=10
PARALLEL_TRIALS=2

# Monitoring
LOG_LEVEL=INFO
METRICS_PATH=/tmp/katib-metrics.log
```

Annexe D : Exemple de Metadata

Listing 49 – Format des métadonnées du modèle

```
{
  "model_id": "iris-mlp-2024-01-15-1234",
  "version": "1.0.0",
  "algorithm": "bayesian",
  "framework": "scikit-learn",
  "model_type": "MLPClassifier",
  "dataset": {
    "name": "iris",
    "samples": 150,
    "features": 4,
    "classes": 3
  },
  "hyperparameters": {
    "learning_rate": 0.0234,
    "hidden_units": 96,
    "optimizer": "adam",
    "max_iter": 1000
  },
  "metrics": {
    "train_accuracy": 0.9833,
    "test_accuracy": 0.9667,
```

```
    "loss": 0.0333,
    "f1_score": 0.9662
  },
  "training": {
    "trials_run": 8,
    "best_trial": 6,
    "duration_seconds": 240,
    "timestamp": "2024-01-15T14:30:45Z"
  },
  "environment": {
    "python_version": "3.9.18",
    "sklearn_version": "1.3.2",
    "kubernetes": "v1.28.0",
    "cluster": "minikube"
  }
}
```