

MACHINE LEARNING OPERATIONS



Presented by Asst. Prof. Dr. Tuchsanai Ploysuwan

 Basic_mlflow



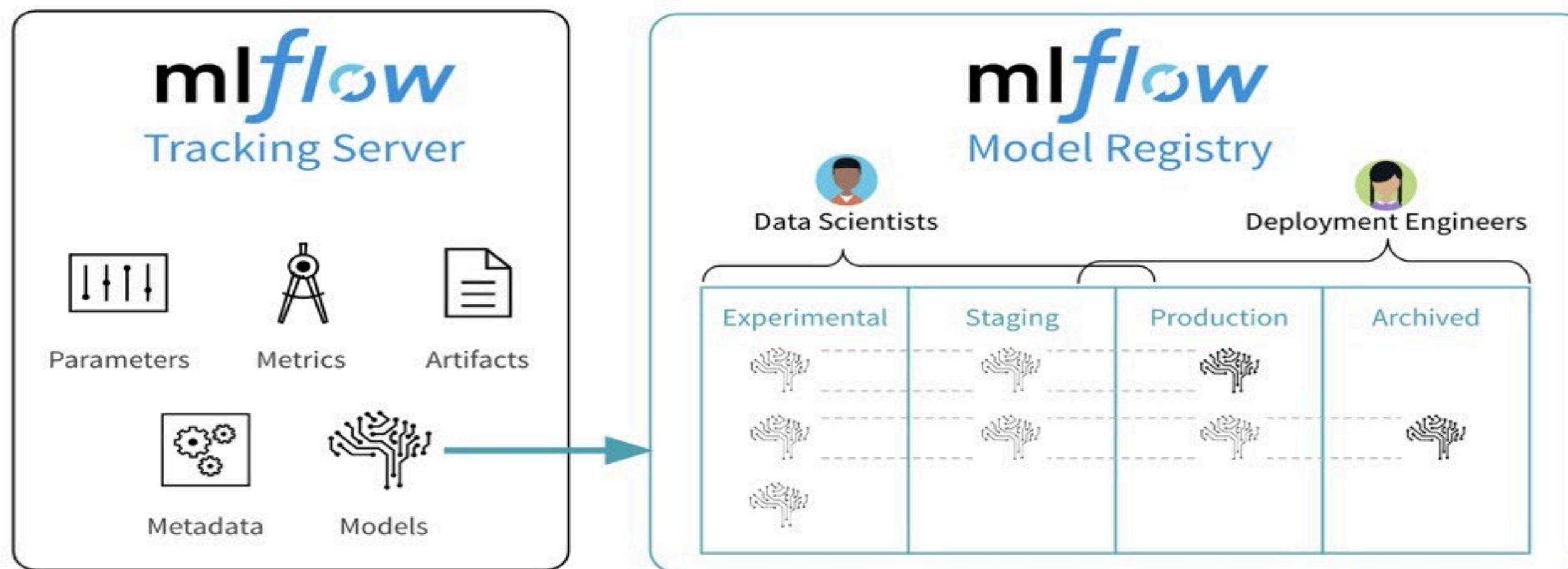
🔍 MLflow คืออะไร?

MLflow เป็น Open Source Platform สำหรับจัดการ Machine Learning Lifecycle ทั้งหมด พัฒนาโดย Databricks และเปิดตัวครั้งแรกในปี 2018 ปัจจุบันเป็นหนึ่งในเครื่องมือ MLOps ที่ได้รับความนิยมมากที่สุด โดยมีจุดเด่นคือความยืดหยุ่นสูง รองรับหลายภาษาโปรแกรม (Python, R, Java, REST API) และสามารถทำงานร่วมกับ ML Framework ยอดนิยมได้ทุกด้วย

ปัญหาที่ MLflow แก้ไข

ในการพัฒนา Machine Learning แบบดั้งเดิม Data Scientists มักประสบปัญหาหลายประการ:

1. **Reproducibility Crisis:** ไม่สามารถทำซ้ำผลการทดลองได้ เพราะลืมว่าใช้ hyperparameters อะไร ใช้ data version ไหน หรือใช้ code version ใด
2. **Experiment Chaos:** มี notebook หลายสิบไฟล์ที่ชื่อคล้ายกัน เช่น `model_v1.ipynb`, `model_v2_final.ipynb`, `model_v2_final_real.ipynb` จนไม่รู้ว่าอันไหนดีที่สุด
3. **Collaboration Difficulty:** ทีมงานไม่สามารถแชร์และเปรียบเทียบผลการทดลองได้อย่างมีประสิทธิภาพ
4. **Deployment Gap:** การนำ Model ไป Deploy ต้องเขียน code ใหม่เกือบทั้งหมด



mlflow TRACKING	mlflow PROJECTS	mlflow MODEL REGISTRY	mlflow MODELS
Record and query experiments: code, data, config, and results.	Package data science code in a format that enables reproducible runs on many platforms	Store, annotate, and manage models in a central repository	Deploy machine learning models in diverse serving environments

องค์ประกอบหลัก 4 ส่วนของ MLflow Platform

MLflow ประกอบด้วย 4 Components ที่ทำงานร่วมกันครอบคลุม ML Lifecycle ทั้งหมด:

ตารางสรุป MLflow Components

Component	Icon	Tagline	หน้าที่หลัก	ใช้เมื่อไหร?
Tracking		"บันทึกทุกการทดลอง"	บันทึก Parameters, Metrics, Artifacts	ตอน Train/Experiment
Projects		"รันที่ไหนก็ได้ผลเหมือนกัน"	Package code ให้ reproduce "ได้"	ตอนแชร์ code กับทีม
Models		"Train ครั้งเดียว Deploy ได้ทุกที่"	Package model ให้ deploy "ได้หลายที่"	ตอน Deploy model
Registry		"จัดการ Model เมื่อ像 Git"	Version control สำหรับ models	ตอนจัดการ Production

สถานการณ์	Component ที่ใช้	เหตุผล
กำลังทดลอง hyperparameters หลายค่า	 Tracking	บันทึกและเปรียบเทียบผลได้
ต้องการให้เพื่อนร่วมทีมรัน code เดียวกัน	 Projects	มี dependencies ครบ รันได้ผลเหมือนกัน
จะนำ model ไป deploy บน server	 Models	Package model ให้พร้อม serve
มี model หลาย versions ต้องจัดการ	 Registry	ติดตาม version, จัดการ Production
ทำทุกอย่างคร่าวงจร	ใช้ห้อง 4	ครอบคลุม ML Lifecycle

💡 ตัวอย่าง Use Case จริง

ขั้นตอน	สิ่งที่ทำ	Component
1	Data Scientist ทดลอง model หลายแบบ บันทึก accuracy แต่ละแบบ	 Tracking
2	เลือก model ที่ดีที่สุด แพ็คโคดให้ทีม ML Engineer	 Projects
3	ML Engineer นำ model ไป deploy เป็น REST API	 Models
4	เมื่อมี model ใหม่ที่ดีกว่า เปลี่ยน Production version	 Registry

MLflow Tracking

รายละเอียด MLflow Tracking

1. MLflow Tracking (เน้นใน Lab นี้)

"บันทึกทุกการทดลอง เปรียบเทียบได้ทุกผลลัพธ์"

MLflow Tracking คือระบบบันทึกและติดตามการทดลอง Machine Learning ช่วยให้คุณไม่ต้องจำว่าใช้ hyperparameters อะไร ได้ผลเท่าไหร่ สิ่งที่บันทึกได้:

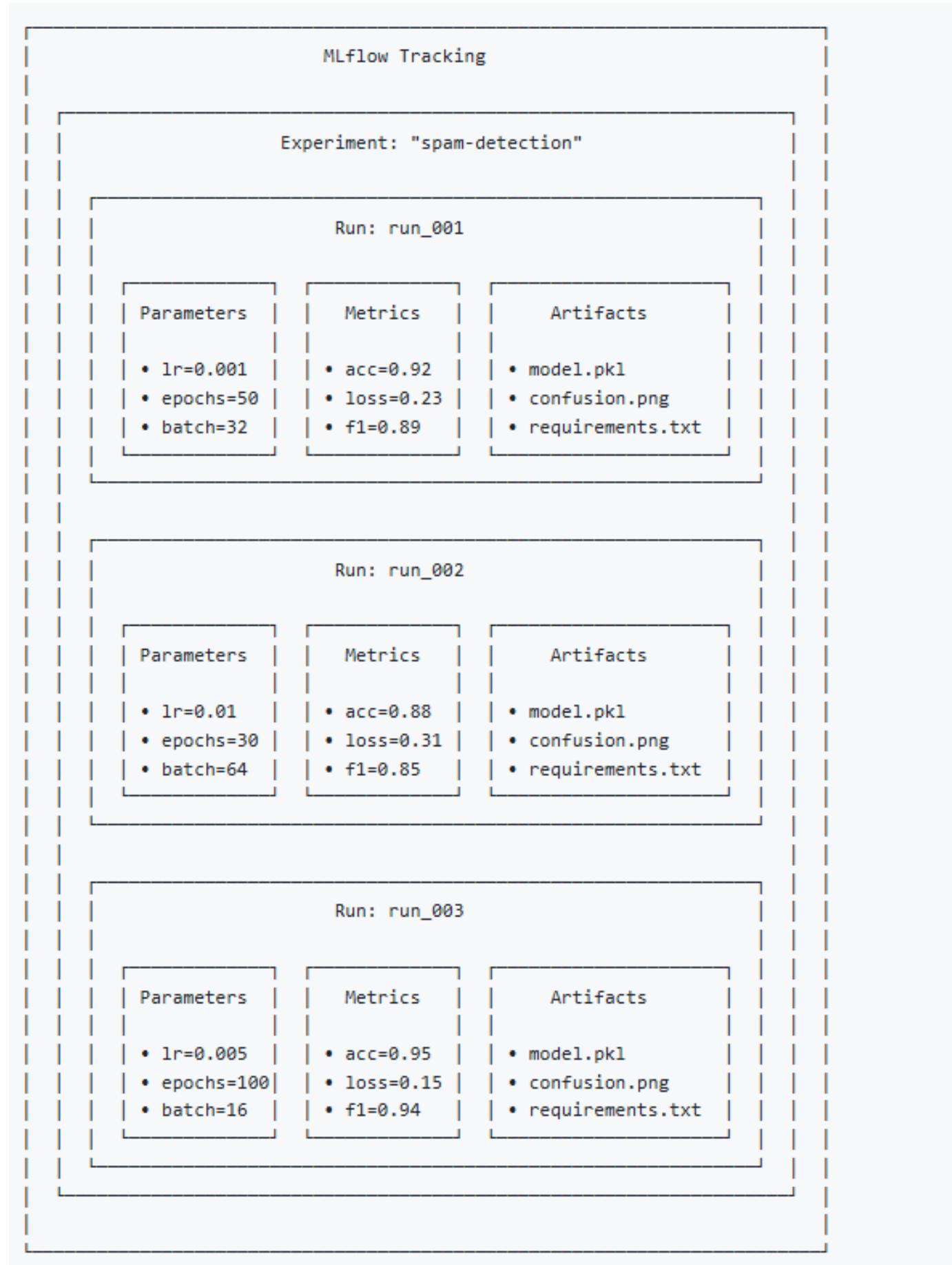
ประเภท	คำอธิบาย	ตัวอย่าง
Parameters	ค่าที่ตั้งก่อนทดลอง (Input)	learning_rate=0.001 , epochs=100
Metrics	ค่าที่วัดได้จากการทดลอง (Output)	accuracy=0.95 , loss=0.05
Artifacts	ไฟล์ที่สร้างจากการทดลอง	model.pkl , confusion_matrix.png
Tags	ข้อมูลเพิ่มเติมสำหรับจัดกลุ่ม	developer=john , env=production

ตัวอย่างการใช้งาน:

```
import mlflow

mlflow.set_experiment("my-experiment")

with mlflow.start_run():
    mlflow.log_param("learning_rate", 0.001)      # บันทึก Parameter
    mlflow.log_metric("accuracy", 0.95)            # บันทึก Metric
    mlflow.log_artifact("model.pkl")                # บันทึก Artifact
```



MLflow Tracking

MLflow Tracking		
Experiment: "spam-detection"		
Run: run_001		
Parameters	Metrics	Artifacts
<ul style="list-style-type: none">lr=0.001epochs=50batch=32	<ul style="list-style-type: none">acc=0.92loss=0.23f1=0.89	<ul style="list-style-type: none">model.pklconfusion.pngrequirements.txt
Run: run_002		
Parameters	Metrics	Artifacts
<ul style="list-style-type: none">lr=0.01epochs=30batch=64	<ul style="list-style-type: none">acc=0.88loss=0.31f1=0.85	<ul style="list-style-type: none">model.pklconfusion.pngrequirements.txt
Run: run_003		
Parameters	Metrics	Artifacts
<ul style="list-style-type: none">lr=0.005epochs=100batch=16	<ul style="list-style-type: none">acc=0.95loss=0.15f1=0.94	<ul style="list-style-type: none">model.pklconfusion.pngrequirements.txt

Experiment: "spam-detection" - Summary Table				
PARAMETERS				
Run	Learning Rate	Epochs	Batch Size	Status
run_001	0.001	50	32	Baseline
run_002	0.01	30	64	Underfitting
run_003	0.005	100	16	⭐ Best Model
METRICS				
Run	Accuracy	Loss	F1-Score	Improvement
run_001	0.92	0.23	0.89	-
run_002	0.88	0.31	0.85	-4.35%
run_003	0.95	0.15	0.94	+3.26%
ARTIFACTS				
Run	Files Logged			
run_001	model.pkl, confusion.png, requirements.txt			
run_002	model.pkl, confusion.png, requirements.txt			
run_003	model.pkl, confusion.png, requirements.txt			
KEY INSIGHTS				
Best Accuracy:	run_003 (0.95)	+	+3.26% improvement over baseline	
Lowest Loss:	run_003 (0.15)	+	34.8% reduction from baseline	
Best F1-Score:	run_003 (0.94)	+	+5.6% improvement over baseline	
Insight:	Smaller batch size (16) + more epochs (100) + moderate lr (0.005) yields the best performance for spam detection task			
Warning:	run_002 shows underfitting due to insufficient epochs (30) and high learning rate (0.01)			

MLflow Project

รายละเอียด MLflow Projects

2. MLflow Projects

"เขียนครั้งเดียว รันที่ไหนก็ได้ผลเหมือนกัน"

MLflow Projects คือ standard format สำหรับ packaging ML code ให้สามารถ reproduce ได้

องค์ประกอบหลัก:

```
my_project/
├── MLproject          # ไฟล์กำหนดค่า entry points
├── conda.yaml         # Dependencies (conda)
├── requirements.txt    # Dependencies (pip)
└── train.py           # โค้ดหลัก
```

ตัวอย่าง MLproject file:

```
name: my_ml_project

conda_env: conda.yaml

entry_points:
  main:
    parameters:
      learning_rate: {type: float, default: 0.01}
      epochs: {type: int, default: 100}
    command: "python train.py --lr {learning_rate} --epochs {epochs}"
```

การรัน Project:

```
# รันจาก local
mlflow run ./my_project -P learning_rate=0.001

# รันจาก GitHub
mlflow run git@github.com:user/repo.git -P epochs=50
```

MLflow Models

🤖 3. MLflow Models

"Train ครึ่งเดียว Deploy ได้ทุกที่"

MLflow Models คือ standard format สำหรับ packaging trained model ให้ deploy ได้หลาย platforms

Model Flavors ที่รองรับ:

Category	Flavors
Traditional ML	sklearn, xgboost, lightgbm, catboost
Deep Learning	pytorch, tensorflow, keras
Big Data	spark, h2o
Other	onnx, prophet, statsmodels, custom

Deployment Targets:

Target	คำอธิบาย
Local	รัน prediction บนเครื่อง
REST API	Deploy เป็น REST endpoint
Docker	Package เป็น Docker container
AWS SageMaker	Deploy บน Amazon SageMaker
Azure ML	Deploy บน Azure Machine Learning
Spark	รันบน Apache Spark cluster

ตัวอย่างการบันทึกและ Deploy Model:

```
# บันทึก Model
import mlflow.sklearn
mlflow.sklearn.log_model(model, "my_model")

# Deploy เป็น REST API
# mlflow models serve -m runs:/<run_id>/my_model -p 5000
```

MLflow Registry

รายละเอียด MLflow Model Registry

"จัดการ Model เมื่อjoinจัดการ Code ด้วย Git"

Model Registry คือ centralized store สำหรับจัดการ model versioning และ lifecycle เปรียบเทียบได้กับ Git ที่ใช้จัดการ source code

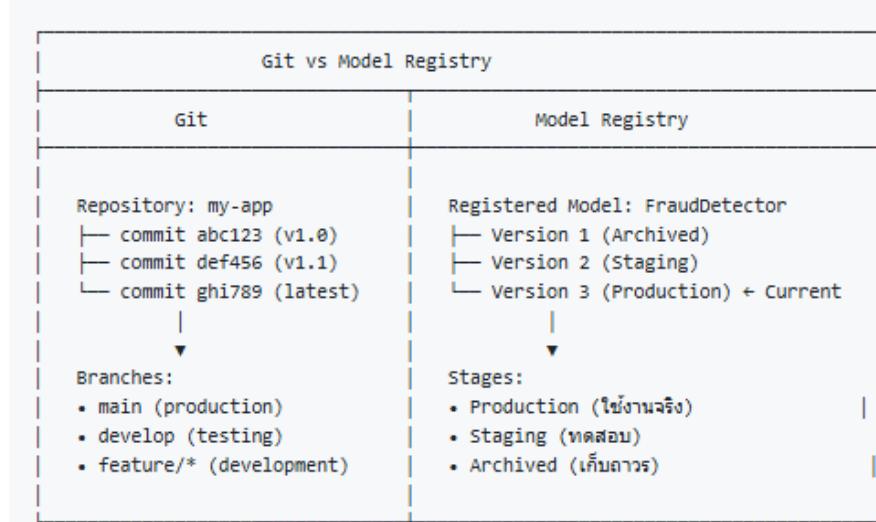
ทำไมต้องมี Model Registry?

ปรับเปลี่ยนเที่ยบ: ไม่มี vs มี Model Registry

คำถามที่พบบ่อย	ไม่มี Model Registry	มี Model Registry
"Model ที่รันบน Production คือไฟล์ไหน?"	ไม่รู้... น่าจะเป็น model_final_v2_new.pkl ?	FraudDetector version 3, stage: Production ✓
"Model นี้ train ด้วย hyperparameters อะไร?"	จำไม่ได้... ลองหาใน notebook ดู	คลิกดู lineage → Run abc123, lr=0.001, epochs=100 ✓
"Model เมื่อวานดีกว่า rollback ได้ไหม?"	ไม่ได้... ลบไฟล์เก่าไปแล้ว 🚫	ได้เลย! เปลี่ยน version 2 เป็น Production ✓
"ใครเป็นคน deploy model นี้? เมื่อไหร่?"	ไม่รู้... ไม่มี log	john@company.com, 2024-01-15 14:30:00 ✓
"Model version ไหนดีที่สุด?"	ต้องเปิด notebook เพียงอย่างเดียว	ดูใน Registry UI ปรับเปลี่ยน metrics ได้เลย ✓
"ทีม QA จะทดสอบ model ในเมื่อไง?"	ส่งไฟล์ให้ทาง Slack/Email	เปลี่ยน stage เป็น Staging แล้วแจ้งทีม ✓

ปรับเปลี่ยน Model Registry กับ Git

แนวคิด	Git (Code)	Model Registry (Model)
Repository	Git Repository	Registered Model
Version	Commit	Model Version
Branch/Tag	main, develop, v1.0	Production, Staging, Archived
History	Commit History	Version History
Rollback	git checkout <commit>	Transition to previous version
Collaboration	Pull Request	Stage Transition + Approval
Metadata	Commit Message	Description, Tags, Lineage



Model Stages คืออะไรและเบื้องต้น

Model Registry มี 4 Stages สำหรับจัดการ lifecycle ของ model:

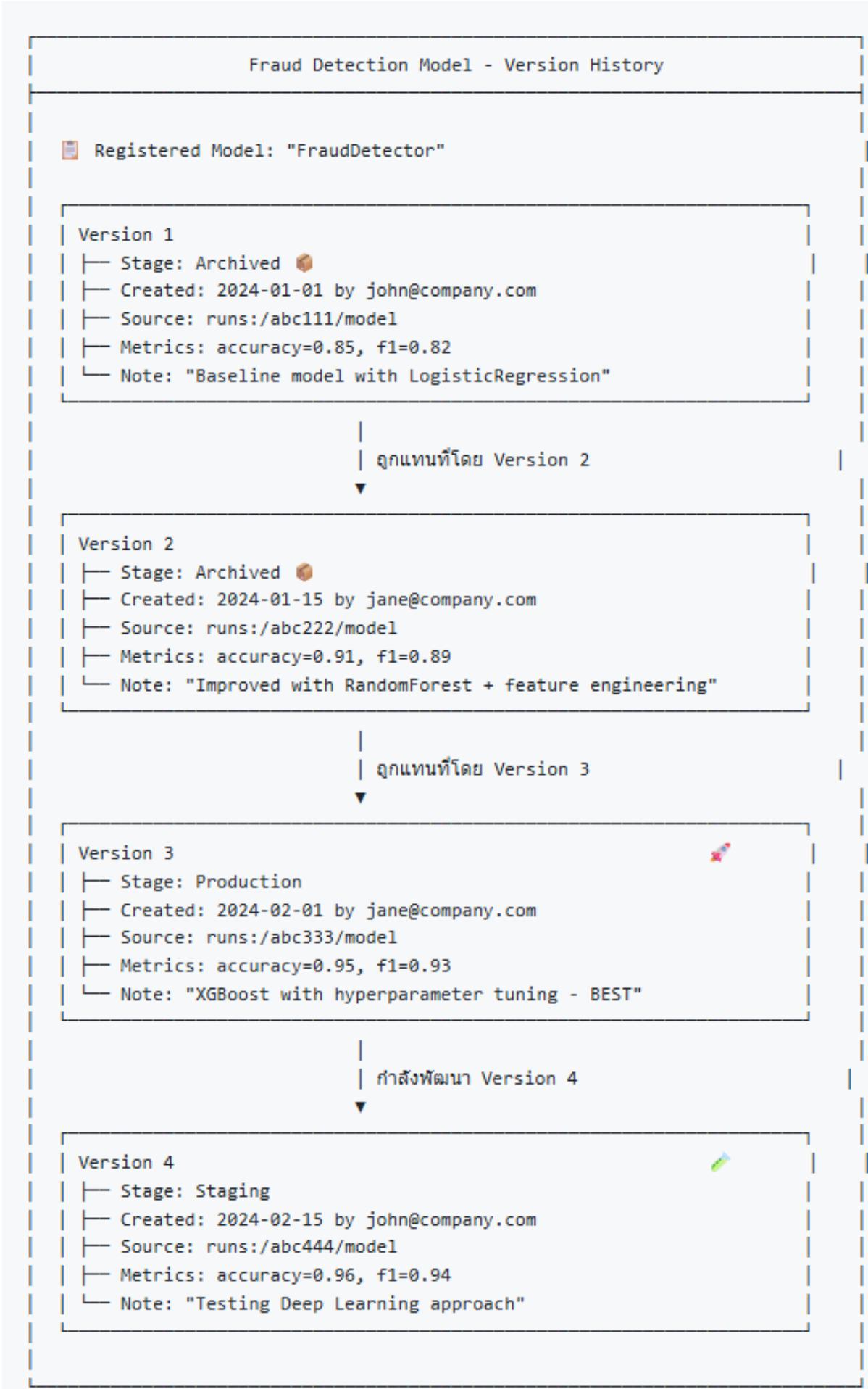


สรุป Stage แต่ละประเภท:

Stage	สัญลักษณ์	สำหรับ	ใครใช้?
None	NONE	เพิ่งลงทะเบียน ยังไม่พร้อม	Data Scientist (กำลังพัฒนา)
Staging	STAGING	กำลังทดสอบ อาจมีหลาย versions	QA Team, ML Engineer
Production	PRODUCTION	ใช้งานจริง ควรมี 1 version	End Users, Applications
Archived	ARCHIVED	เลิกใช้แล้ว เก็บไว้อ้างอิง	ไม่มีใครใช้ (backup)

ตัวอย่าง Workflow จริง

สมมติทีมพัฒนา Fraud Detection Model:



Workflow อธิบาย:

วันที่ 1: John สร้าง Version 1 (Baseline)
→ ลงทะเบียนเป็น FraudDetector Version 1
→ ทดสอบผ่าน → เป็น Production

วันที่ 15: Jane สร้าง Version 2 (ดีกว่า)
→ ลงทะเบียนเป็น FraudDetector Version 2
→ ทดสอบใน Staging
→ ผ่านการทดสอบ → เป็น Production
→ Version 1 ถูกเปลี่ยนเป็น Archived

วันที่ 30: Jane สร้าง Version 3 (ดีที่สุด)
→ ลงทะเบียน → ทดสอบ → Production
→ Version 2 → Archived

วันที่ 45: John ก้าวสั้งทดสอบ Version 4
→ อยู่ใน Staging ทดสอบ
→ ถ้าผ่าน จะเข้า Production แทน Version 3

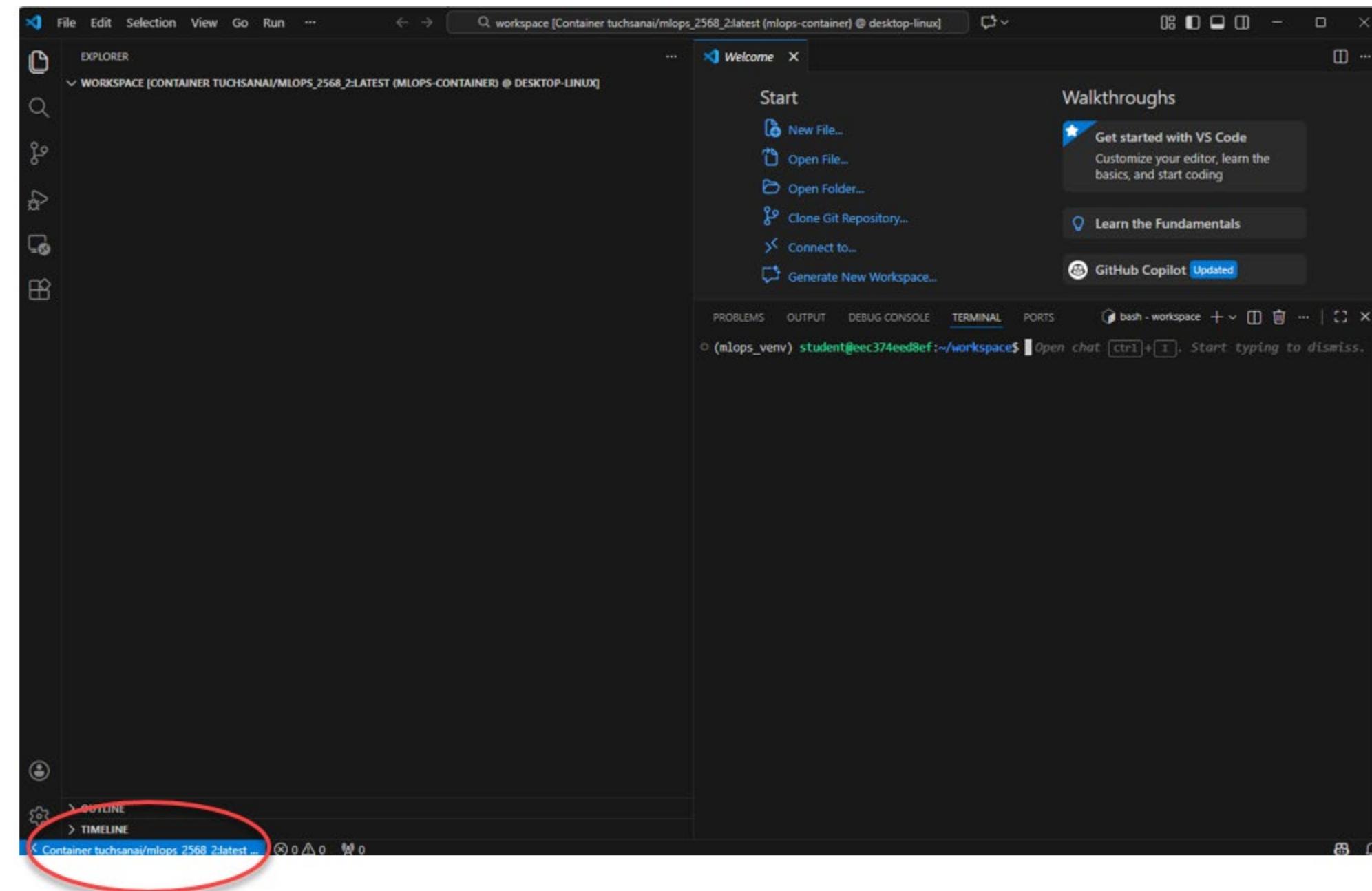
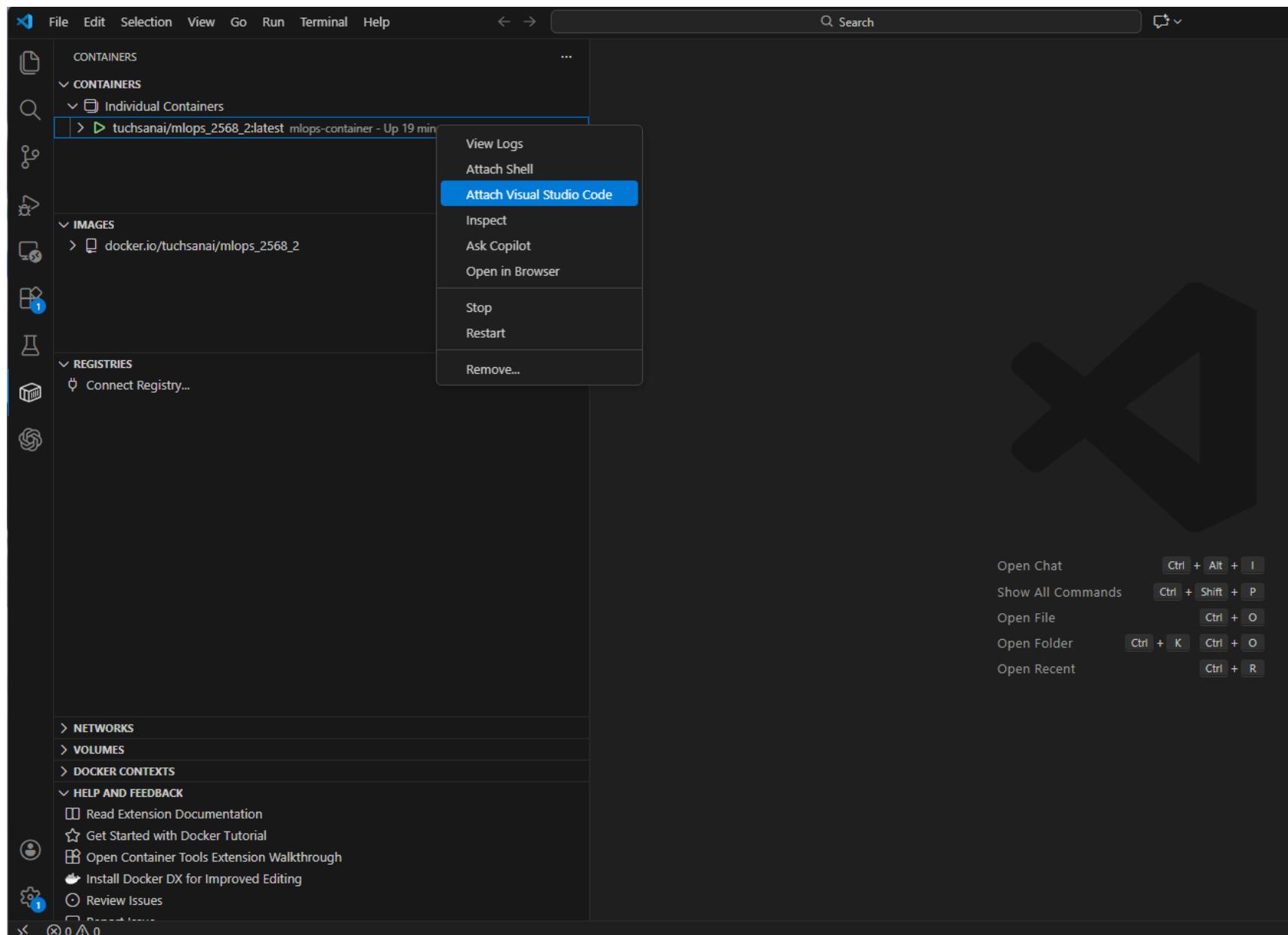
How to install and set up MLflow

All LAB Run in Dev Container

การใช้งาน LAB ผ่าน Docker

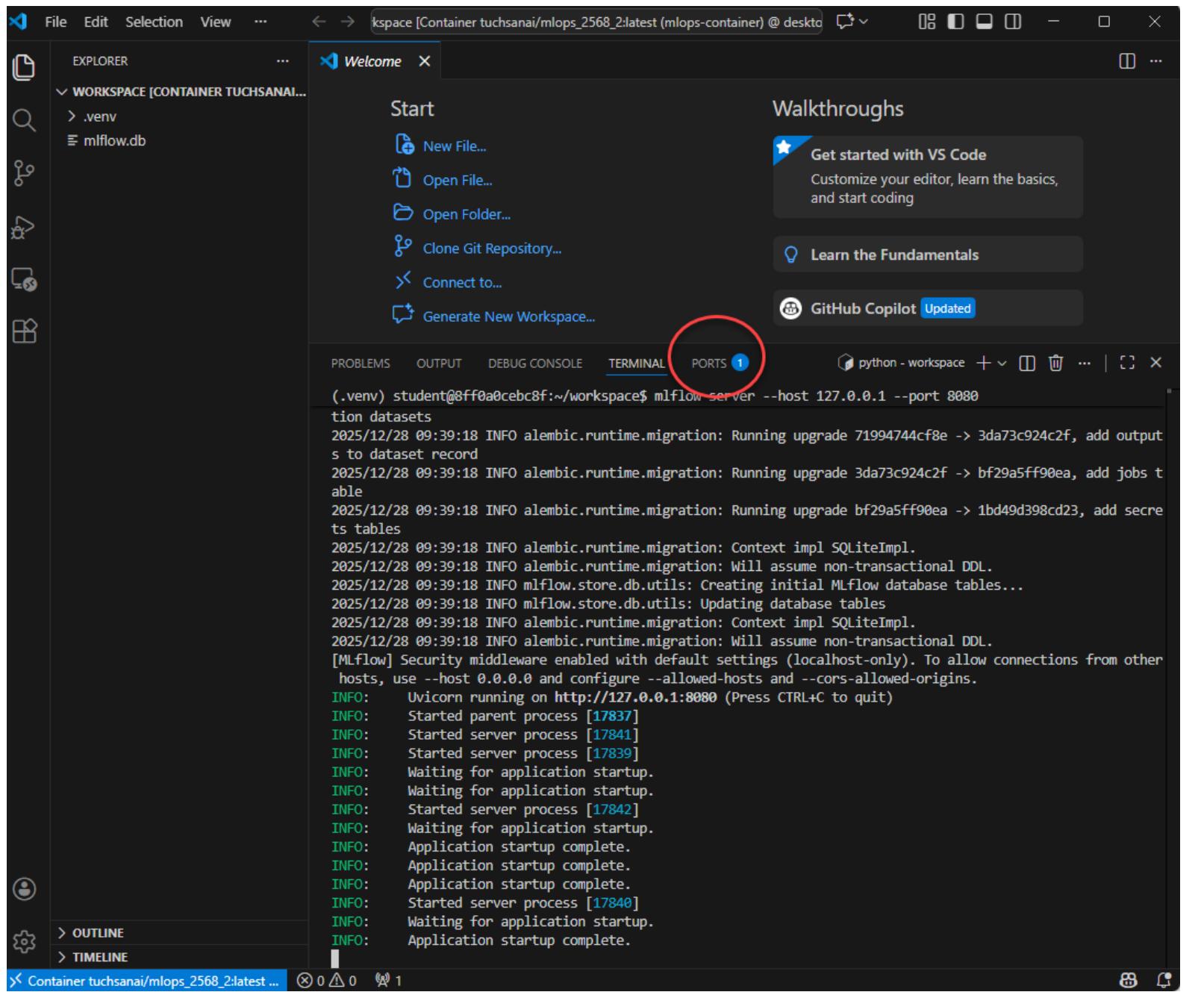
สำหรับผู้ที่ต้องการใช้งานผ่าน Docker Container ที่ติดตั้ง Library พื้นฐานไว้ครบถ้วนแล้ว สามารถรันคำสั่งดังนี้:

```
docker run -d -p 8080:8080 -p 8888:8888 --name mlops-container tuchsanai/mlops_2568_2:latest
```



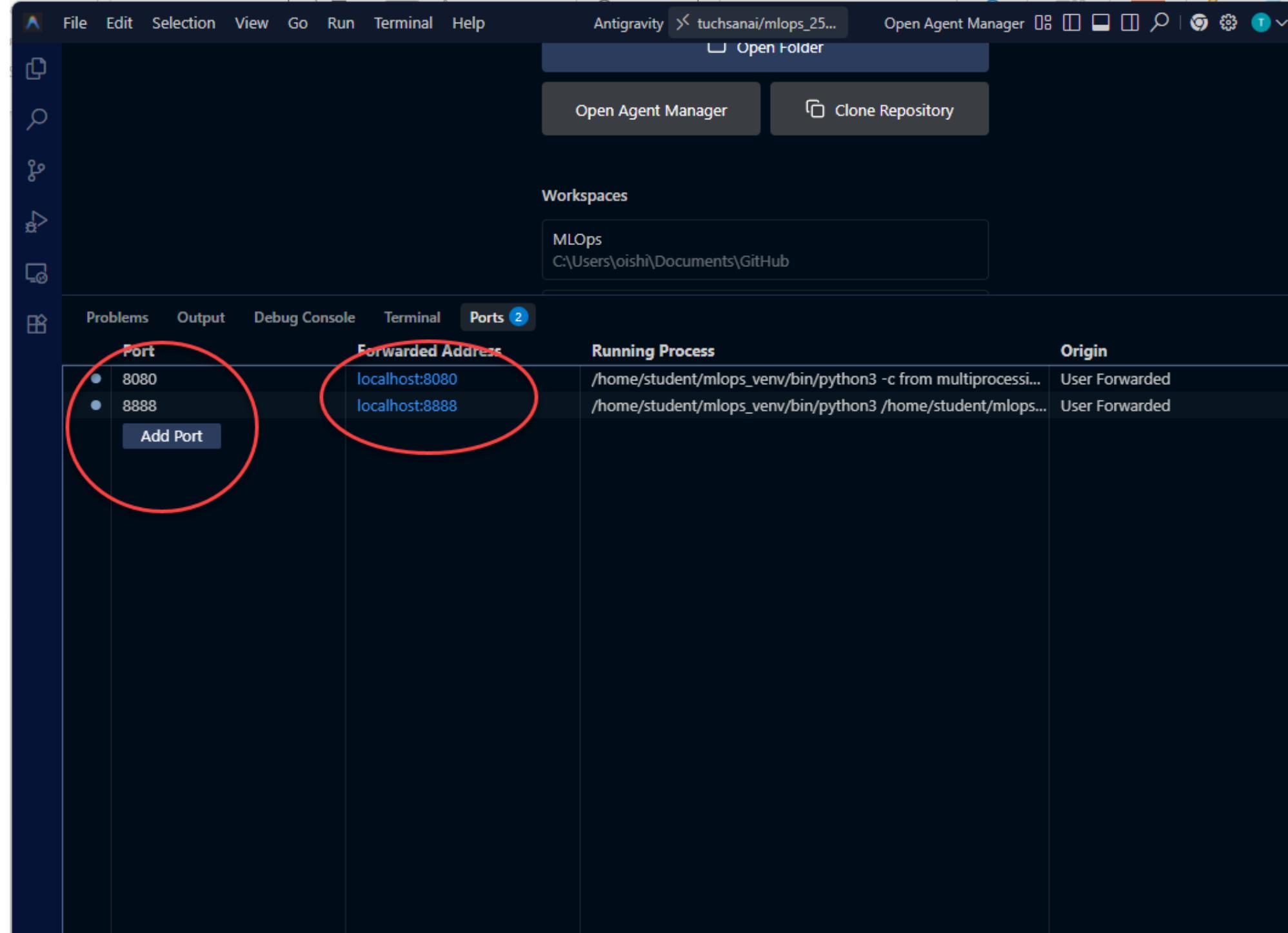
Add port 8080 สำหรับ mlflow

Add port 8888 สำหรับ jupyter



The screenshot shows the VS Code interface with the terminal tab active. The terminal window displays the command `mlflow server --host 127.0.0.1 --port 8080` being run in a container. The output shows the server starting up, including migration logs and application startup messages.

```
(.venv) student@8ff0a0cebc8f:~/workspace$ mlflow server --host 127.0.0.1 --port 8080
2025/12/28 09:39:18 INFO alembic.runtime.migration: Running upgrade 71994744cf8e -> 3da73c924c2f, add outputs to dataset record
2025/12/28 09:39:18 INFO alembic.runtime.migration: Running upgrade 3da73c924c2f -> bf29a5ff90ea, add jobs table
2025/12/28 09:39:18 INFO alembic.runtime.migration: Running upgrade bf29a5ff90ea -> 1bd49d398cd23, add secrets tables
2025/12/28 09:39:18 INFO alembic.runtime.migration: Context impl SQLiteImpl.
2025/12/28 09:39:18 INFO alembic.runtime.migration: Will assume non-transactional DDL.
2025/12/28 09:39:18 INFO mlflow.store.db.utils: Creating initial MLflow database tables...
2025/12/28 09:39:18 INFO mlflow.store.db.utils: Updating database tables
2025/12/28 09:39:18 INFO alembic.runtime.migration: Context impl SQLiteImpl.
2025/12/28 09:39:18 INFO alembic.runtime.migration: Will assume non-transactional DDL.
[MLflow] Security middleware enabled with default settings (localhost-only). To allow connections from other hosts, use --host 0.0.0.0 and configure --allowed-hosts and --cors-allowed-origins.
INFO:    Uvicorn running on http://127.0.0.1:8080 (Press CTRL+C to quit)
INFO:    Started parent process [17837]
INFO:    Started server process [17841]
INFO:    Started server process [17839]
INFO:    Waiting for application startup.
INFO:    Waiting for application startup.
INFO:    Started server process [17842]
INFO:    Waiting for application startup.
INFO:    Application startup complete.
INFO:    Application startup complete.
INFO:    Application startup complete.
INFO:    Started server process [17840]
INFO:    Waiting for application startup.
INFO:    Application startup complete.
```



The screenshot shows the Antigravity interface with the 'Ports' tab selected. It lists two forwarded ports: 8080 (localhost:8080) and 8888 (localhost:8888). A red circle highlights the 'Add Port' button at the bottom of the list.

Port	Forwarded Address	Running Process	Origin
8080	localhost:8080	/home/student/mlops_venv/bin/python3 -c from multiprocessing...	User Forwarded
8888	localhost:8888	/home/student/mlops_venv/bin/python3 /home/student/mlops...	User Forwarded

localhost:8888/login?next=%2Flab%3F

jupyter

Password : mlops

Password or token: Log in

Token authentication is enabled

If no password has been configured, you need to open the server with its login token in the URL, or paste it above. This requirement will be lifted if you [enable a password](#).

The command:

```
jupyter server list
```

will show you the URLs of running servers with their tokens, which you can copy and paste into your browser. For example:

```
Currently running servers:  
http://localhost:8888/?token=c8de56fa... :: /Users/you/notebooks
```

or you can paste just the token value into the password field on this page.

See [the documentation on how to enable a password](#) in place of token authentication, if you would like to avoid dealing with random tokens.

Cookies are required for authenticated access to the Jupyter server.

Setup a Password

You can also setup a password by entering your token and a new password on the fields below:

Token

...

New Password

localhost:8888/lab

JupyterLab

File Edit View Run Kernel Tabs Settings Help

Launcher

Notebook

Python 3 (ipykernel) Python (MLOps Venv)

Console

Python 3 (ipykernel) Python (MLOps Venv)

Other

Terminal Text File Markdown File Python File Show Contextual Help

(225) YouTube Push running container to Dock MLOps/02_MLFLOW/01_basic/ JupyterLab

ขั้นตอนการทำ Lab

ขั้นตอนที่ 1: สร้างโฟลเดอร์สำหรับ Lab

```
# สร้างโฟลเดอร์สำหรับเก็บไฟล์ Lab
mkdir -p mlflowserver-lab

# เข้าไปในโฟลเดอร์
cd mlflowserver-lab
```

ขั้นตอนที่ 1: สร้าง Virtual Environment

ทำไมต้องใช้ Virtual Environment?

Virtual Environment ช่วยแยก Package ของแต่ละโปรเจกต์ออกจากกัน ป้องกันปัญหา Package Version ขัดแย้งกัน

```
# 1.1 สร้าง Virtual Environment ชื่อ .venv
python -m venv .venv

# 1.2 เปิดใช้งาน Virtual Environment
source .venv/bin/activate

# 1.3 อัพเดท pip ให้เป็นเวอร์ชันล่าสุด
python -m pip install --upgrade pip
```

ตรวจสอบ: เมื่อเปิดใช้งานสำเร็จ จะเห็น (.venv) นำหน้า prompt

ขั้นตอนที่ 2: ติดตั้ง Package ที่จำเป็น

```
pip install mlflow[extras] scikit-learn pandas numpy
```

รายละเอียด Package:

Package	หน้าที่
mlflow[extras]	MLflow เวอร์ชันสมบูรณ์ รวม Model Serving, SQL Storage และ Dependencies ทั้งหมด
scikit-learn	Library สำหรับ Machine Learning
pandas	จัดการข้อมูลแบบ DataFrame
numpy	คำนวณทางคณิตศาสตร์

แบบที่ 1: In-Memory (สำหรับทดลองใช้งานเร็ว)

```
nohup mlflow server --host 0.0.0.0 --port 8080 > mlflow.log 2>&1 &
```

ข้อดี: เริ่มต้นได้เร็ว ไม่ต้องตั้งค่าอะไรเพิ่ม

ข้อเสีย: ข้อมูลหายเมื่อปิด Server

ผลลัพธ์ที่คาดหวัง:

```
INFO:     Started server process [28550]
INFO:     Application startup complete.
Uvicorn running on http://127.0.0.1:8080 (Press CTRL+C to quit)
```

แบบที่ 2: Persistent Storage (แนะนำสำหรับใช้งานจริง)

```
# 3.1 สร้างโฟลเดอร์เก็บข้อมูล
mkdir -p mlruns_db mlartifacts

# 3.2 เปิด Server ให้เข้าถึงได้จากทุก IP
nohup mlflow server \
--host 0.0.0.0 --port 8080 \
--backend-store-uri sqlite:///mlruns_db/mlflow.db \
--artifacts-destination ./mlartifacts \
--serve-artifacts > mlflow.log 2>&1 &
```

อธิบายโครงสร้างไฟล์:

```
โปรเจกต์ของคุณ/
├── mlruns_db/
│   └── mlflow.db      # ฐานข้อมูล SQLite เก็บ Experiment และ Run Metadata
├── mlartifacts/
└── .venv/            # Virtual Environment
```

ส่วนประกอบ	หน้าที่
Backend Store	เก็บข้อมูล Experiment, Run, Parameters, Metrics
Artifact Store	เก็บไฟล์โมเดล, กราฟ, ไฟล์ผลลัพธ์อื่นๆ

+ New

Home

Experiments

Models

Prompts

Welcome to MLflow

Information about UI telemetry

MLflow collects usage data to improve the product. To confirm your preferences, please visit the settings page in the navigation sidebar. To learn more about what data is collected, please visit the [documentation](#).

Get started

Log traces

Trace LLM applications for debugging and monitoring.

Run evaluation

Iterate on quality with offline evaluations and comparisons.

Train models

Track experiments, parameters, and metrics throughout training.

Register prompts

Manage prompt updates and collaborate across teams.

Experiments

[View all](#)

<input type="checkbox"/>	Name	Time created	Last modified	Description	Tags
<input type="checkbox"/>	Default	12/28/2025, 04:39:18 PM	12/28/2025, 04:39:18 PM	-	

Discover new features

[View all](#)

MLflow MCP server

Connect your coding assistants and AI applications to MLflow and automatically analyze your experiments and traces.



Optimize prompts

Access the state-of-the-art prompt optimization algorithms such as MIPROv2, GEPA, through MLflow Prompt Registry.



Agents-as-a-judge

Leverage agents as a judge to perform deep trace analysis and improve your evaluation accuracy.



Dataset tracking

Track dataset lineage and versions and effectively drive the quality improvement loop.



MLflow Tracking

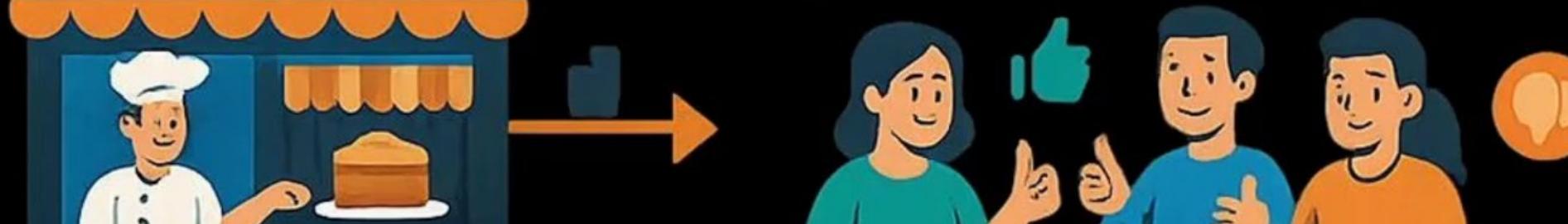
EXPERIMENTS (TRYING DIFFERENT CAKES)



METRICS (TASTE TEST RESULTS)



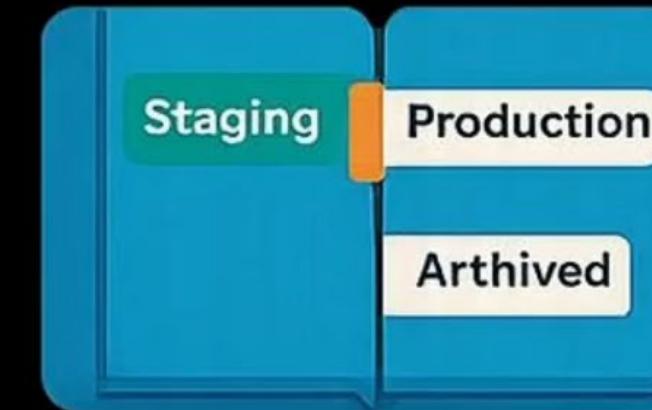
DEPLOYMENT & MONITORING (CAKE SHOP & CUSTOMERS FEEDBACK)



PARAMETERS (RECIPE SETTINGS)

Sugar: 100g
Oven Temp:
190°C
Time: 30 min

MODEL REGISTRY (RECIPE BOOK)



องค์ประกอบหลัก 4 ส่วนของ MLflow Platform

MLflow ประกอบด้วย 4 Components ที่ทำงานร่วมกันครอบคลุม ML Lifecycle ทั้งหมด:

ตารางสรุป MLflow Components

Component	Icon	Tagline	หน้าที่หลัก	ใช้เมื่อไหร?
Tracking		"บันทึกทุกการทดลอง"	บันทึก Parameters, Metrics, Artifacts	ตอน Train/Experiment
Projects		"รันที่ไหนก็ได้ผลเหมือนกัน"	Package code ให้ reproduce ได้	ตอนแชร์ code กับทีม
Models		"Train ครั้งเดียว Deploy ได้ทุกที่"	Package model ให้ deploy ได้หลายที่	ตอน Deploy model
Registry		"จัดการ Model เมื่อไหร"	Version control สำหรับ models	ตอนจัดการ Production

MLflow Tracking คือระบบบันทึกและติดตามการทดลอง Machine Learning ช่วยให้คุณไม่ต้องจำว่าใช้ hyperparameters อะไร ได้ผลเท่าไหร่ สิ่งที่บันทึกได้:

ประเภท	คำอธิบาย	ตัวอย่าง
Parameters	ค่าที่ตั้งก่อนทดลอง (Input)	<code>learning_rate=0.001, epochs=100</code>
Metrics	ค่าที่วัดได้จากการทดลอง (Output)	<code>accuracy=0.95, loss=0.05</code>
Artifacts	ไฟล์ที่สร้างจากการทดลอง	<code>model.pkl, confusion_matrix.png</code>
Tags	ข้อมูลเพิ่มเติมสำหรับจัดกลุ่ม	<code>developer=john, env=production</code>

ตัวอย่างการใช้งาน:

```
import mlflow

mlflow.set_experiment("my-experiment")

with mlflow.start_run():
    mlflow.log_param("learning_rate", 0.001) # บันทึก Parameter
    mlflow.log_metric("accuracy", 0.95) # บันทึก Metric
    mlflow.log_artifact("model.pkl") # บันทึก Artifact
```

II องค์ประกอบหลักของ MLflow Tracking

MLflow Tracking มีองค์ประกอบสำคัญ 5 ส่วนที่ทำงานร่วมกันเป็นระบบ:

1 Experiment (การทดลอง)

Experiment คือกลุ่มของ Runs ที่เกี่ยวข้องกัน เปรียบเสมือน "โปรเจกต์" หรือ "หัวข้อการทดลอง" ทุก Run ต้องอยู่ภายใต้ Experiment ไดเสมอ

หลักการตั้งชื่อ Experiment:

- ใช้ชื่อที่สื่อความหมายชัดเจน เช่น `fraud-detection-lstm`, `customer-churn-prediction`
- หลีกเลี่ยงชื่อที่ว่าไป เช่น `test`, `experiment1`
- ใช้ kebab-case หรือ snake_case เพื่อความสม่ำเสมอ

```
Experiment: "Image Classification"
└── Run 1: ทดลองด้วย ResNet50
└── Run 2: ทดลองด้วย VGG16
└── Run 3: ทดลองด้วย EfficientNet
└── Run 4: ทดลองด้วย MobileNet
```

ตัวอย่างการใช้งาน:

```
import mlflow

# สร้างหรือเลือก Experiment
# ถ้า Experiment นี้มีอยู่แล้ว จะเลือกใช้อันเดิม
# ถ้ายังไม่มี จะสร้างใหม่อัตโนมัติ
mlflow.set_experiment("image-classification")

# ดู Experiment ID ที่กำลังใช้อยู่
experiment = mlflow.get_experiment_by_name("image-classification")
print(f"Experiment ID: {experiment.experiment_id}")
print(f"Artifact Location: {experiment.artifact_location}")
```

2 Run (การรันทดลอง)

Run คือการทดลองแต่ละครั้ง เป็นหน่วยพื้นฐานที่สุดของ MLflow Tracking ทุกครั้งที่ train model ด้วย hyperparameters ชุดใหม่ ควรสร้าง Run ใหม่

Run ประกอบด้วย:

- Parameters: ค่าที่ตั้งก่อนเริ่ม (input)
- Metrics: ค่าที่ได้ได้ (output)
- Artifacts: ไฟล์ที่สร้างขึ้น
- Metadata: เวลาเริ่ม, เวลาจบ, สถานะ, Git commit hash ฯลฯ
- Tags: ข้อมูลเพิ่มเติมสำหรับการค้นหาและจัดกลุ่ม

รูปแบบการสร้าง Run:

```
# แบบที่ 1: ใช้ Context Manager (แนะนำ)
# Run จะปิดตัวโน้มตีเมื่อออกจาก with block
with mlflow.start_run(run_name="resnet50-experiment"):
    # ติดการทดลองอยู่ตรงนี้
    mlflow.log_param("learning_rate", 0.001)
    mlflow.log_metric("accuracy", 0.95)

# แบบที่ 2: Nested Runs (สำหรับ Hyperparameter Tuning)
with mlflow.start_run(run_name="parent-run"):
    for lr in [0.001, 0.01, 0.1]:
        with mlflow.start_run(run_name=f"child-lr-{lr}", nested=True):
            mlflow.log_param("learning_rate", lr)
```

Experiment Properties:

Property	คำอธิบาย
<code>experiment_id</code>	Unique identifier (auto-generated)
<code>name</code>	ชื่อ Experiment ที่ตั้ง
<code>artifact_location</code>	ที่เก็บ Artifacts
<code>lifecycle_stage</code>	<code>active</code> หรือ <code>deleted</code>
<code>tags</code>	Metadata เพิ่มเติม

4 Metrics (เมตทริก)

Metrics คือค่าที่วัดได้จากการทดลอง (Output/Results) สามารถบันทึกได้หลายครั้งพร้อม step number เพื่อติดตามการเปลี่ยนแปลงตามเวลา

ประเภทของ Metrics:

ประเภท	ตัวอย่าง	คำอธิบาย
Model Hyperparameters	learning_rate, batch_size, epochs, dropout	ค่าที่ควบคุมพฤติกรรมการเรียนรู้ของ Model
Data Parameters	train_split, validation_split, image_size, augmentation	ค่าที่เกี่ยวกับการเตรียมข้อมูล
Architecture	model_type, num_layers, hidden_units, activation	โครงสร้างของ Model
Training Config	optimizer, loss_function, early_stopping_patience	การตั้งค่าการ Train
Environment	random_seed, gpu_id, num_workers	การตั้งค่าสภาพแวดล้อม

วิธีการบันทึก Metrics:

```
# วิธีที่ 1: บันทึก Metric ค่าเดียว (Final metric)
mlflow.log_metric("accuracy", 0.95)
mlflow.log_metric("f1_score", 0.92)

# วิธีที่ 2: บันทึก Metric ที่เปลี่ยนแปลงตามเวลา
# step parameter ใช้ระบุลำดับ (เช่น epoch number)
for epoch in range(100):
    train_loss, val_loss = train_one_epoch()
    mlflow.log_metric("train_loss", train_loss, step=epoch)
    mlflow.log_metric("val_loss", val_loss, step=epoch)

# วิธีที่ 3: บันทึกหลาย Metrics พร้อมกัน
mlflow.log_metrics({
    "precision": 0.89,
    "recall": 0.94,
    "f1": 0.91
}, step=epoch)

# วิธีที่ 4: บันทึก Metric พร้อม timestamp
mlflow.log_metric("accuracy", 0.95, timestamp=1234567890)
```

ความแตกต่างระหว่าง Parameters และ Metrics:

คุณสมบัติ	Parameters	Metrics
เวลาบันทึก	ก่อนเริ่มทดลอง	ระหว่าง/หลังทดลอง
จำนวนค่า	ค่าเดียวต่อชื่อ	หลายค่าได้ (ต่าง step)
การเปลี่ยนแปลง	คงที่	เปลี่ยนได้ตามเวลา
วัตถุประสงค์	บอกว่าทำอะไร (Input)	บอกว่าได้ผลอย่างไร (Output)
ตัวอย่าง	learning_rate=0.001	accuracy=0.95

3 Parameters (พารามิเตอร์)

Parameters คือค่าที่เราตั้งก่อนเริ่มการทดลอง (Input Configuration) เป็นค่าคงที่ตลอดการทดลอง ไม่เปลี่ยนแปลง

ประเภทของ Parameters:

ประเภท	ตัวอย่าง	คำอธิบาย
Model Hyperparameters	learning_rate, batch_size, epochs, dropout	ค่าที่ควบคุมพฤติกรรมการเรียนรู้ของ Model
Data Parameters	train_split, validation_split, image_size, augmentation	ค่าที่เกี่ยวกับการเตรียมข้อมูล
Architecture	model_type, num_layers, hidden_units, activation	โครงสร้างของ Model
Training Config	optimizer, loss_function, early_stopping_patience	การตั้งค่าการ Train
Environment	random_seed, gpu_id, num_workers	การตั้งค่าสภาพแวดล้อม

วิธีการบันทึก Parameters:

```
# วิธีที่ 1: บันทึกที่ลະค่า
mlflow.log_param("learning_rate", 0.001)
mlflow.log_param("batch_size", 32)
mlflow.log_param("epochs", 100)

# วิธีที่ 2: บันทึกหลายค่าพร้อมกัน (แนะนำ)
mlflow.log_params({
    "optimizer": "adam",
    "dropout": 0.5,
    "hidden_units": 256,
    "activation": "relu"
})

# วิธีที่ 3: บันทึกจาก argparse หรือ config dict
import argparse
args = argparse.Namespace(lr=0.001, batch=32)
mlflow.log_params(vars(args))
```

ข้อควรระวัง:

- Parameter value ต้องเป็น string หรือ number เท่านั้น
- ไม่สามารถแก้ไข Parameter หลังจากบันทึกแล้ว
- ชื่อ Parameter ควรสื่อความหมายและใช้รูปแบบเดียวกันตลอด

5 Artifacts (สิ่งประดิษฐ์)

Artifacts คือไฟล์ที่สร้างจากการทดลอง สามารถเป็นไฟล์ประเภทใดก็ได้ MLflow จะเก็บไว้ใน Artifact Store

ประเภทของ Artifacts:

ประเภท	ตัวอย่าง	คำอธิบาย
Models	model.pkl, model.h5, model.pt, model.onnx	ไฟล์ Model ที่ Train เสร็จ
Visualizations	confusion_matrix.png, loss_curve.png, roc_curve.png	กราฟและภาพ
Data	predictions.csv, feature_importance.json, test_results.parquet	ข้อมูลผลลัพธ์
Code	train.py, config.yaml, requirements.txt	Source code และ config
Reports	report.html, summary.pdf	รายงานสรุป

วิธีการบันทึก Artifacts:

```

import mlflow
import matplotlib.pyplot as plt
import os

# สร้างโฟลเดอร์สำหรับเก็บไฟล์ชั่วคราว
os.makedirs("outputs/models", exist_ok=True)
os.makedirs("outputs/plots", exist_ok=True)
os.makedirs("outputs/data", exist_ok=True)

with mlflow.start_run():

    # ----- วิธีที่ 1: บันทึกไฟล์เดียว -----
    # ไฟล์จะถูกเก็บที่ root ของ artifacts/
    mlflow.log_artifact("model.pkl")           # → artifacts/model.pkl
    mlflow.log_artifact("confusion_matrix.png") # → artifacts/confusion_matrix.png

    # ----- วิธีที่ 2: บันทึกไฟล์ใน subdirectory -----
    # ใช้ artifact_path กำหนดโฟลเดอร์ปลายทาง
    mlflow.log_artifact("model.pkl", artifact_path="models") # → artifacts/models/model.pkl
    mlflow.log_artifact("predictions.csv", artifact_path="data") # → artifacts/data/predictions.csv

    # ----- วิธีที่ 3: บันทึกทั้งโฟลเดอร์ -----
    # ไฟล์ทั้งหมดในโฟลเดอร์จะถูกบันทึก
    mlflow.log_artifacts("./outputs/plots")          # → artifacts/[ไฟล์ทั้งหมดใน plots]
    mlflow.log_artifacts("./outputs/plots", artifact_path="visualizations") # → artifacts/visualizations/[ไฟล์]

    # ----- วิธีที่ 4: บันทึก dict เป็น JSON (MLflow 2.0+) -----
    config = {"learning_rate": 0.001, "batch_size": 32}
    mlflow.log_dict(config, artifact_file="config/hyperparameters.json") # → artifacts/config/hyperparameters.json

    # ----- วิธีที่ 5: บันทึก text (MLflow 2.0+) -----
    mlflow.log_text("RandomForest v1.0", artifact_file="models/model_metadata.txt") # → artifacts/models/model_metadata.txt

    # ----- วิธีที่ 6: บันทึก figure จาก matplotlib (MLflow 1.13+) -----
    fig, ax = plt.subplots()
    ax.plot([1, 2, 3], [1, 4, 9])
    ax.set_title("Loss Curve")
    mlflow.log_figure(fig, artifact_file="plots/loss_curve.png") # → artifacts/plots/loss_curve.png
    plt.close(fig) # ปิด figure หลังบันทึกเพื่อป้องกัน memory leak

```

Artifact Organization (ผลลัพธ์ที่ได้จาก Code ด้านบน):

```

artifacts/
|
├── model.pkl
|   + วิธีที่ 1: log_artifact("model.pkl")
|   + วิธีที่ 1: log_artifact("confusion_matrix.png")
|
├── models/
|   ├── model.pkl
|   |   + วิธีที่ 2: log_artifact("model.pkl", artifact_path="models")
|   |   + วิธีที่ 5: log_text(..., artifact_file="models/model_metadata.txt")
|
|   └── model_metadata.txt
|
├── data/
|   └── predictions.csv
|       + วิธีที่ 2: log_artifact("predictions.csv", artifact_path="data")
|
├── plots/
|   └── loss_curve.png
|       + วิธีที่ 6: log_figure(fig, artifact_file="plots/loss_curve.png")
|
└── visualizations/
    └── [ไฟล์จาก ./outputs/plots] + วิธีที่ 3: log_artifacts("./outputs/plots", artifact_path="visualizations")
|
└── config/
    └── hyperparameters.json + วิธีที่ 4: log_dict(config, artifact_file="config/hyperparameters.json")

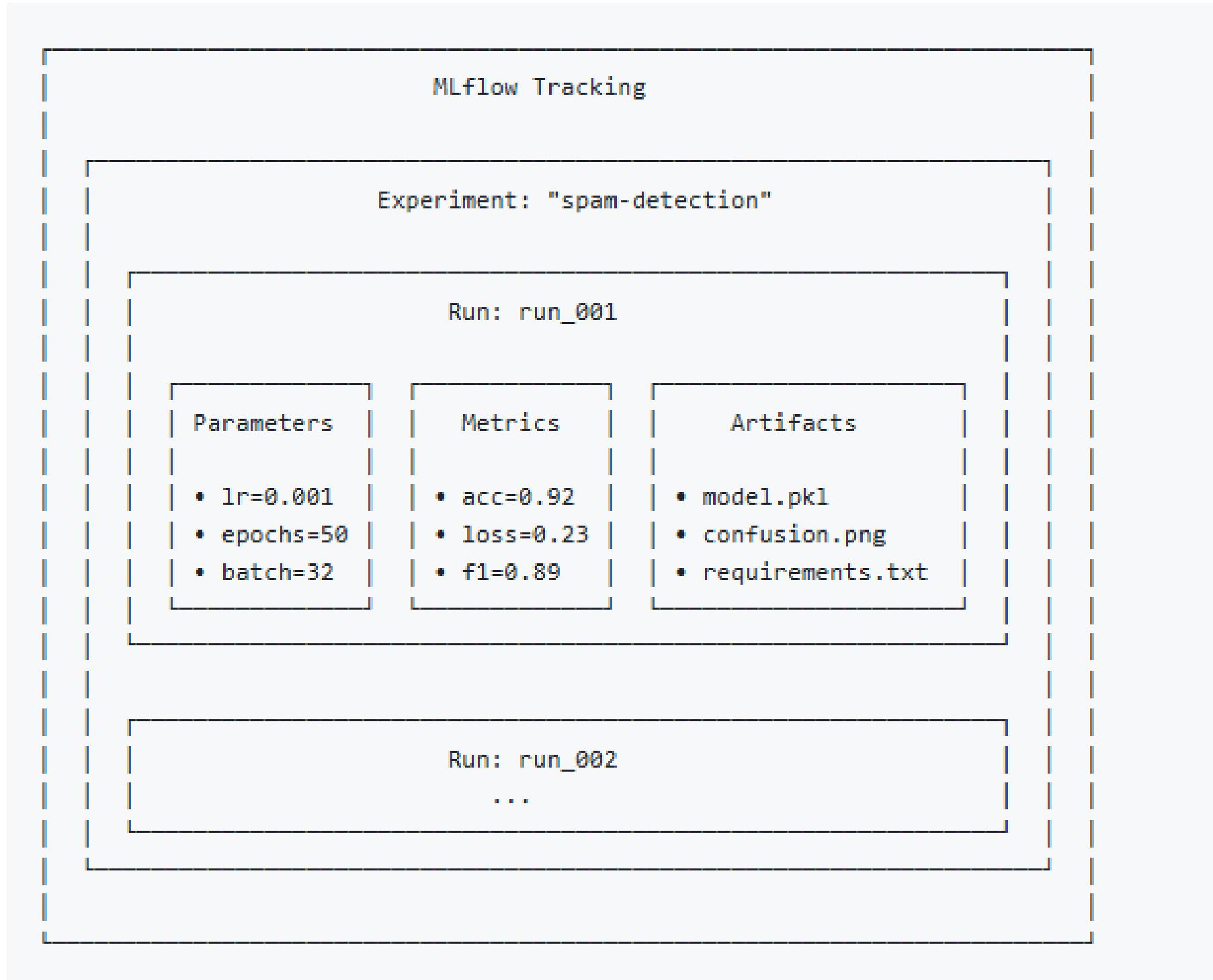
```

สรุปความแตกต่างของแต่ละวิธี:

วิธี	Function	Input	Output Location
1	log_artifact(local_path)	ไฟล์ในเครื่อง	artifacts/ (root)
2	log_artifact(local_path, artifact_path)	ไฟล์ในเครื่อง	artifacts/{artifact_path}/
3	log_artifacts(local_dir)	โฟลเดอร์ในเครื่อง	artifacts/ หรือ artifacts/{artifact_path}/
4	log_dict(dict, artifact_file)	Python dict	artifacts/{artifact_file}
5	log_text(text, artifact_file)	Python string	artifacts/{artifact_file}
6	log_figure(fig, artifact_file)	Matplotlib figure	artifacts/{artifact_file}

หมายเหตุ: วิธีที่ 1-3 ใช้ artifact_path (ระบุโฟลเดอร์) ส่วนวิธีที่ 4-6 ใช้ artifact_file (ระบุ path เต็มรวมชื่อไฟล์)

ภาพรวมความซัมพันธ์ขององค์ประกอบ



Backend Store vs Artifact Store

MLflow แยกการเก็บข้อมูลออกเป็น 2 ส่วนที่ชัดเจน เพื่อประสิทธิภาพและความยืดหยุ่นในการ Scale:

MLflow Storage Architecture	
Backend Store (เก็บ Metadata)	Artifact Store (เก็บไฟล์)
เก็บอะไร?	เก็บอะไร?
<ul style="list-style-type: none">Experiment infoRun metadataParametersMetricsTags	<ul style="list-style-type: none">Model filesImages/PlotsData filesAny binary filesLarge files
ลักษณะข้อมูล:	ลักษณะข้อมูล:
<ul style="list-style-type: none">Structured (key-value)Small sizeQuery-able	<ul style="list-style-type: none">Unstructured (files)Variable sizeDownload/Upload
ตัวเลือก:	ตัวเลือก:
<ul style="list-style-type: none">File system (mlruns/)SQLiteMySQL/PostgreSQLCloud databases	<ul style="list-style-type: none">Local filesystemAmazon S3Azure Blob StorageGoogle Cloud StorageSFTP

ทำไมต้องแยก?

1. Performance: Metadata ถูก query บ่อย ต้องเร็ว ส่วนไฟล์ใหญ่เข้าถึงไม่บ่อย
2. Scalability: สามารถ scale แต่ละส่วนแยกกันตามความต้องการ
3. Cost Optimization: ใช้ storage ที่เหมาะสมกับประเภทข้อมูล
4. Flexibility: เลือก backend ได้ตามโครงสร้างพื้นฐานที่มี

ใน Lab นี้เราใช้:

- Backend Store: SQLite (ไฟล์ `mlflow.db`) - เมมาระสำหรับการเรียนรู้และทีมขนาดเล็ก
- Artifact Store: Local filesystem (โฟลเดอร์ `mlartifacts/`) - เก็บไฟล์ในเครื่อง

Pre-requisite: เตรียมความพร้อมก่อนเริ่ม Lab

สิ่งที่ต้องมี

ก่อนเริ่ม Lab นี้ ต้องมี services ต่อไปนี้รันอยู่และเข้าถึงได้:

Service	URL	หน้าที่
MLflow Server	http://127.0.0.1:8080	ติดตามและจัดการ experiments
Jupyter Notebook	http://127.0.0.1:8888	เขียนและรัน code

ตรวจสอบ MLflow Server

เปิด Browser และไปที่: <http://127.0.0.1:8080>

The screenshot shows the MLflow UI at <http://127.0.0.1:8080>. The main navigation bar includes Home, Experiments, Models, and Prompts. The 'Experiments' section displays a table with one row for 'Default'. The 'Discover new features' section highlights MLflow MCP server, Optimize prompts, and Agents-as-a-judge.

Name	Time created	Last modified	Description	Tags
Default	01/01/2026, 09:21:54 AM	01/01/2026, 09:21:54 AM	-	

ตรวจสอบ Jupyter Server

เปิด Browser และไปที่: <http://127.0.0.1:8888>

The screenshot shows the Jupyter Notebook interface at <http://127.0.0.1:8888>. It displays a file tree with 'mlflowserver-lab' and two notebooks: 'Python 3 (ipykernel)' and 'Python (MLOps Venv)'. Other options include 'Console', 'Terminal', 'Text File', 'Markdown File', and 'Python File'.

Clone Lab Repository

เปิด Terminal ใน Jupyter และรันคำสั่ง:

```
git clone -b dev https://github.com/Tuchsanai/MLOps.git
```

หมายเหตุ: ใช้ branch `dev` สำหรับ Lab นี้

The screenshot shows a Jupyter Notebook interface with a terminal window and a file browser.

Terminal Window: A red oval highlights the terminal output. It shows the command \$ git clone -b dev https://github.com/Tuchsanai/MLOps.git followed by the progress of the cloning process: Cloning into 'MLOps'..., remote: Enumerating objects: 3967, done., remote: Counting objects: 100% (69/69), done., remote: Compressing objects: 100% (57/57), done., remote: Total 3967 (delta 13), reused 56 (delta 8), pack-reused 3898 (from 2). Receiving objects: 100% (3967/3967), 213.61 MiB | 15.10 MiB/s, done. Resolving deltas: 100% (1528/1528), done. \$

File Browser: On the left, there is a sidebar with icons for file operations. The main area shows a list of directories. Two items are circled with red ovals: 'mlflowserver-lab' and 'MLOps'. The 'MLOps' folder is highlighted with a larger red oval.

3. Clone Lab Repository

เปิด Terminal ใน Jupyter และรันคำสั่ง:

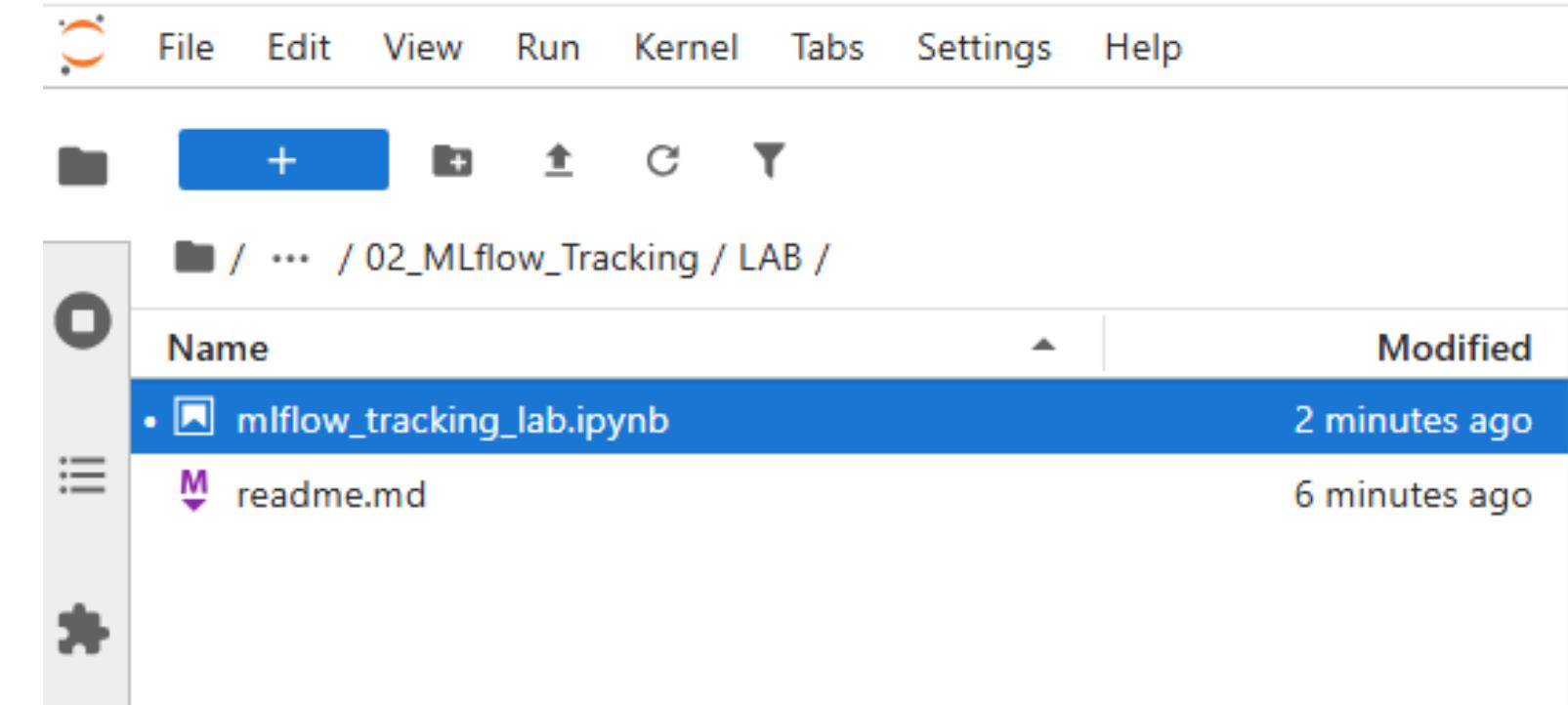
```
git clone -b dev https://github.com/Tuchsanai/MLOps.git
```

หมายเหตุ: ใช้ branch `dev` สำหรับ Lab นี้

4. เปิดไฟล์ Lab

เปิดไฟล์ `mlflow_tracking_lab.ipynb` ที่อยู่ใน path:

```
MLOps/02_MLFLOW/01_basic/02_MLflow_Tracking/LAB/mlflow_tracking_lab.ipynb
```



The screenshot shows a Jupyter Notebook cell with the following content:

Lab: MLflow Tracking พื้นฐาน

วัตถุประสงค์การเรียนรู้:

- เข้าใจองค์ประกอบหลัก 5 ส่วนของ MLflow Tracking
- สามารถสร้าง Experiment และ Run ได้
- บันทึก Parameters, Metrics และ Artifacts ได้อย่างถูกต้อง
- บันทึกและโหลด Model ได้
- เมริตรันที่บันทึกของ MLflow UI

องค์ประกอบหลักของ MLflow Tracking:

องค์ประกอบ	คำอธิบาย	ตัวอย่าง
Experiment	คลุนช่องการทดลองที่เก็บข้อมูล	fraud-detection, customer-churn
Run	การทำทดลองและครึ้ง	ผู้ทดสอบที่ train model
Parameters	ค่าที่ส่งกลับเมื่อทดลอง (Input)	learning_rate=0.001, epochs=100
Metrics	ค่าที่รับได้จากการทดลอง (Output)	accuracy=0.92, loss=0.15
Artifacts	ไฟล์ที่สร้างจากการทดลอง	model.pkl, confusion_matrix.png

Pre-requisite: เตรียมความพร้อมก่อนเริ่ม Lab

สิ่งที่ต้องมี
ก่อนเริ่ม Lab นี้ ต้องมี MLflow Server รันอยู่และเข้าถึงได้ที่ <http://127.0.0.1:8080>

ตรวจสอบ MLflow Server
เปิด Browser และไปที่ <http://127.0.0.1:8080>
MLflow UI

ส่วนที่ 1: การเชื่อมต่อ MLflow Server

แนวคิด
ก่อนใช้งาน MLflow Tracking ต้องกำหนด Tracking URI เพื่อบอกว่าจะเก็บข้อมูลการทดลองไว้ที่ไหน

พึงกันเสี่ยง
ทางเลือกอื่นสำหรับ Tracking URI

方	ตัวอย่าง URI	คำอธิบาย
mlflow.set_tracking_uri(uri)	กำหนด URL ของ MLflow Server	
mlflow.get_tracking_uri()	ตรวจสอบ URL ที่กำหนดไว้	

ประเภท	ตัวอย่าง URI	คำอธิบาย
Local File	file:///path/to/mlruns	เก็บในไฟล์เดียว local
Remote Server	http://server:5000	เก็บบน MLflow Server
Databricks	databricks	เก็บบน Databricks
SQLite	sqlite:///mlflow.db	เก็บใน SQLite database

ส่วนที่ 1: การเชื่อมต่อ MLflow Server

แนวคิด

ก่อนใช้งาน MLflow Tracking ต้องกำหนด **Tracking URI** เพื่อบอกว่าจะเก็บข้อมูลการทดลองไว้ที่ไหน

ฟังก์ชันสำคัญ

ฟังก์ชัน	คำอธิบาย
<code>mlflow.set_tracking_uri(uri)</code>	กำหนด URL ของ MLflow Server
<code>mlflow.get_tracking_uri()</code>	ตรวจสอบ URL ที่กำหนดไว้

ทางเลือกอื่นสำหรับ Tracking URI

ประเภท	ตัวอย่าง URI	คำอธิบาย
Local File	<code>file:///path/to/mlruns</code>	เก็บในโฟลเดอร์ local
Remote Server	<code>http://server:5000</code>	เก็บบน MLflow Server
Databricks	<code>databricks</code>	เก็บบน Databricks
SQLite	<code>sqlite:///mlflow.db</code>	เก็บใน SQLite database

```
[1]: import mlflow
import os

# กำหนด Tracking URI ไปยัง MLflow Server
mlflow.set_tracking_uri("http://127.0.0.1:8080")

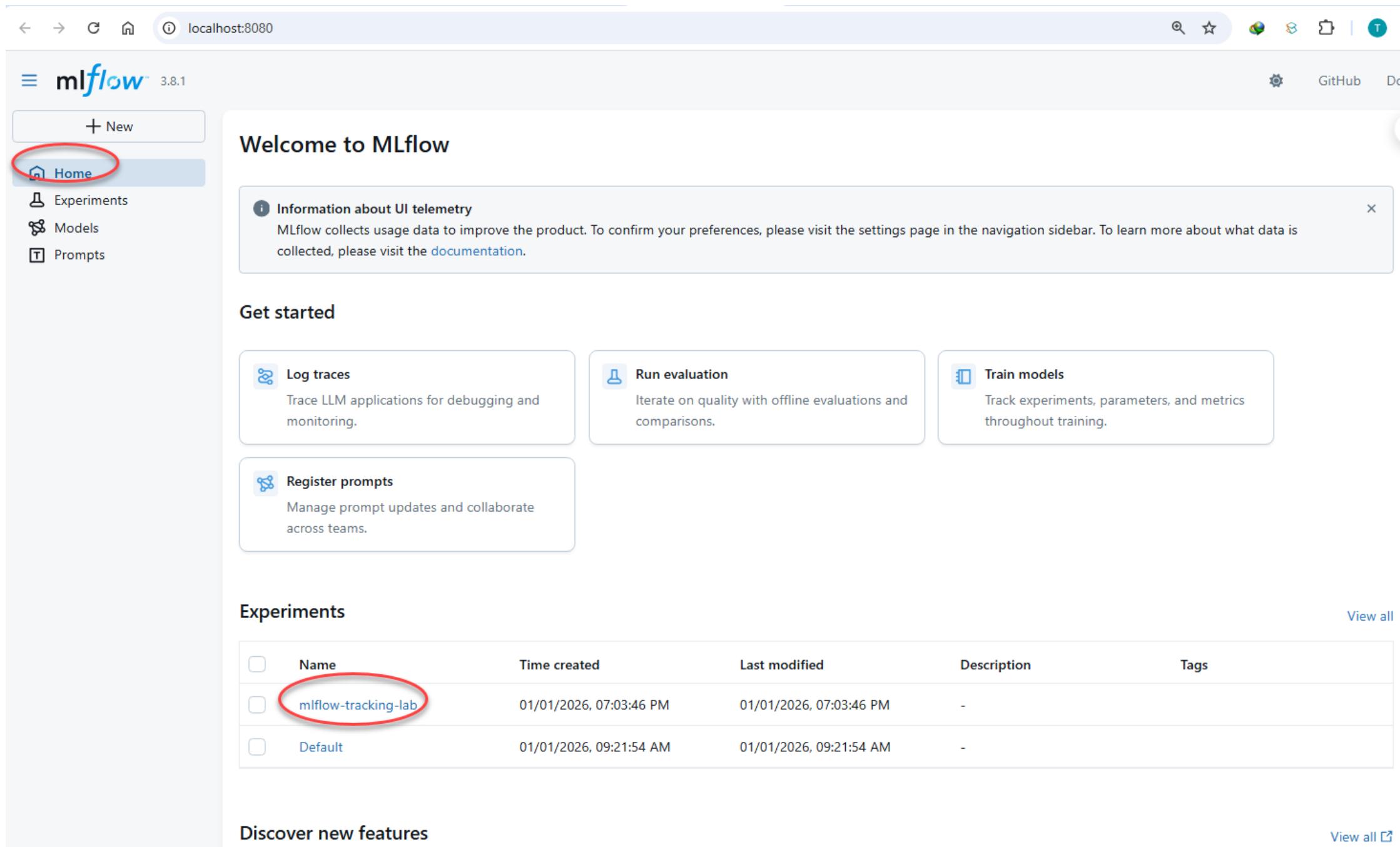
# ตรวจสอบว่าเชื่อมต่อถูกต้อง
print(f"✓ MLflow Tracking URI: {mlflow.get_tracking_uri()}")
```

✓ MLflow Tracking URI: http://127.0.0.1:8080

```
[2]: # สร้างหรือเลือก Experiment
experiment_name = "mlflow-tracking-lab"
mlflow.set_experiment(experiment_name)

# ดูข้อมูล Experiment
experiment = mlflow.get_experiment_by_name(experiment_name)
print(f"Experiment Name: {experiment.name}")
print(f"Experiment ID: {experiment.experiment_id}")
print(f"Artifact Location: {experiment.artifact_location}")
print(f"Lifecycle Stage: {experiment.lifecycle_stage}")

2026/01/01 12:03:46 INFO mlflow.tracking.fluent: Experiment with name 'mlflow-tracking-lab' does not exist. Creating a new experiment.
Experiment Name: mlflow-tracking-lab
Experiment ID: 1
Artifact Location: mlflow-artifacts:/1
Lifecycle Stage: active
```



The screenshot shows the MLflow UI interface. The top part is a terminal window displaying Python code and its execution output. The output shows that an experiment named 'mlflow-tracking-lab' was created because it did not exist. The bottom part is a web browser window showing the 'Welcome to MLflow' page. The sidebar on the left has a 'Home' button highlighted with a red circle. The main content area includes sections for 'Get started' (Log traces, Run evaluation, Train models, Register prompts) and 'Experiments' (a table listing experiments). The 'mlflow-tracking-lab' experiment is listed in the experiments table, also circled in red.

localhost:8080

Welcome to MLflow

Information about UI telemetry

MLflow collects usage data to improve the product. To confirm your preferences, please visit the settings page in the navigation sidebar. To learn more about what data is collected, please visit the [documentation](#).

Get started

- Log traces
- Run evaluation
- Train models
- Register prompts

Experiments

Name	Time created	Last modified	Description	Tags
mlflow-tracking-lab	01/01/2026, 07:03:46 PM	01/01/2026, 07:03:46 PM	-	
Default	01/01/2026, 09:21:54 AM	01/01/2026, 09:21:54 AM	-	

Discover new features

localhost:8080/#/experiments

mlflow 3.8.1

+ New

Experiments

Home Experiments Models Prompts

Filter experiments by name Tag filter

Name	Time created	Last modified	Description	Tags
mlflow-tracking-lab	01/01/2026, 07:03:46 PM	01/01/2026, 07:03:46 PM	-	
Default	01/01/2026, 09:21:54 AM	01/01/2026, 09:21:54 AM	-	

Create Compare Delete

localhost:8080/#/experiments/1/runs?searchFilter=&orderByKey=attributes.start_time&orderByAsc=false&startTime=ALL&lifecycleFilter=Active&modelVersionFilt...

mlflow 3.8.1

mlflow-tracking-lab Machine learning

Runs Models Traces

Share View docs

Run Name Created Dataset Duration Source Models

ส่วนที่ 3: การสร้าง Run และบันทึก Parameters

แนวคิด

Run คือการทดลองแต่ละครั้ง เป็นหน่วยพื้นฐานที่สุดของ MLflow Tracking ทุกครั้งที่ train model ด้วย hyperparameters ชุดใหม่ ควรสร้าง Run ใหม่

Parameters คือค่าที่เราตั้งก่อนเริ่มการทดลอง (Input Configuration) เป็นค่าคงที่ตลอดการทดลอง ไม่เปลี่ยนแปลง

ประเภทของ Parameters

ประเภท	ตัวอย่าง
Model Hyperparameters	learning_rate, batch_size, epochs
Data Parameters	train_split, image_size
Architecture	model_type, num_layers, hidden_units
Training Config	optimizer, loss_function

ฟังก์ชันสำคัญ

ฟังก์ชัน	คำอธิบาย
mlflow.start_run(run_name)	เริ่มต้น Run ใหม่
mlflow.log_param(key, value)	บันทึก Parameter ที่ล็อกค่า
mlflow.log_params(dict)	บันทึกหลาย Parameters พร้อมกัน
mlflow.active_run()	ดึงข้อมูล Run ที่กำลังทำงานอยู่

Context Manager (with statement)

การใช้ `with mlflow.start_run()` มีข้อดีคือ:

- Run จะถูกปิดอัตโนมัติเมื่อออกจาก block
- ป้องกันการลืมปิด Run
- จัดการ error ได้ดีกว่า

```
[3]: # กลับมาใช้ Experiment หลัก
mlflow.set_experiment("mlflow-tracking-lab")

# สร้าง Run และบันทึก Parameters
with mlflow.start_run(run_name="demo-parameters"):

    # วิธีที่ 1: บันทึกทีละค่า
    mlflow.log_param("learning_rate", 0.001)
    mlflow.log_param("batch_size", 32)
    mlflow.log_param("epochs", 100)

    # วิธีที่ 2: บันทึกหลายค่าพร้อมกัน (แนะนำ)
    mlflow.log_params({
        "optimizer": "adam",
        "dropout": 0.5,
        "hidden_units": 256,
        "activation": "relu",
        "model_type": "MLP"
    })

# ดู Run ID
run_id = mlflow.active_run().info.run_id
print(f"✅ สร้าง Run สำเร็จ!")
print(f">ID: Run ID: {run_id}")
print(f"📝 บันทึก Parameters: learning_rate, batch_size, epochs, optimizer, dropout, hidden_units, activation, model_type")

✅ สร้าง Run สำเร็จ!
ID Run ID: 2a3088fef75e47258e0ee66314eb46c3
📝 บันทึก Parameters: learning_rate, batch_size, epochs, optimizer, dropout, hidden_units, activation, model_type
🔗 View run demo-parameters at: http://127.0.0.1:8080/#/experiments/1/runs/2a3088fef75e47258e0ee66314eb46c3
🔗 View experiment at: http://127.0.0.1:8080/#/experiments/1
```

The screenshot shows the MLflow tracking interface. At the top, there's a browser header with the URL `localhost:8080/#/experiments/1/runs?searchFilter=&orderByKey=attributes.start_time&orderByAsc=false&s...`. Below it, the MLflow logo and version 3.8.1 are displayed. The main area shows a table of runs with columns: Run Name, Created, Dataset, Duration, Source, and Models. One run is listed: 'demo-parameters' was created 2 minutes ago. The interface also features various filters and sorting options at the top of the table.

localhost:8080#/experiments/1/runs/2a3088fef75e47258e0ee66314eb46c3

mlflow 3.8.1

mlflow-tracking-lab > Runs >

demo-parameters

Overview Model metrics System metrics Traces Artifacts

About this run

Created at	01/01/2026, 07:15:23 PM
Created by	student
Experiment ID	1
Status	Finished
Run ID	2a3088fef75e47258e0ee66314eb46c3
Duration	245ms
Source	ipykernel_launcher.py
Logged models	—
Registered prompts	—

Datasets

None

Tags

Add tags

Registered models

None

mlflow 3.8.1

mlflow-tracking-lab > Runs >

demo-parameters

Overview Model metrics System metrics Traces Artifacts

Parameters (8)

Search parameters

Parameter	Value
learning_rate	0.001
batch_size	32
epochs	100
optimizer	adam
dropout	0.5
hidden_units	256
activation	relu
model_type	MLP

การอ่านข้อมูลจาก MLflow วิธีที่ 1 (อ่านจาก Run ID โดยตรง)

```
from mlflow.tracking import MlflowClient
import mlflow

# ตั้งค่า tracking URI (ถ้าจำเป็น)
mlflow.set_tracking_uri("http://127.0.0.1:8080")

client = MlflowClient()

# ดึงข้อมูล Run
run = client.get_run(run_id)

# แสดงข้อมูลทั้งหมด
print("Run Info:")
print(f" - Run Name: {run.info.run_name}")
print(f" - Status: {run.info.status}")
print(f" - Start Time: {run.info.start_time}")

print("\nParameters:")
for key, value in run.data.params.items():
    print(f" - {key}: {value}")

print("\nMetrics:")
for key, value in run.data.metrics.items():
    print(f" - {key}: {value}")
```

Run Info:
- Run Name: demo-parameters
- Status: FINISHED
- Start Time: 1767269723348

Parameters:
- learning_rate: 0.001
- batch_size: 32
- epochs: 100
- optimizer: adam
- dropout: 0.5
- hidden_units: 256
- activation: relu
- model_type: MLP

Metrics:

การอ่านข้อมูลจาก MLflow วิธีที่ 2 (ค้นหา Runs จาก Experiment)

```
import mlflow

# วิธีที่ 1: ใช้ search_runs (ได้ DataFrame)
df = mlflow.search_runs(experiment_names=["mlflow-tracking-lab"])

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1 entries, 0 to 0
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   run_id          1 non-null      object  
 1   experiment_id  1 non-null      object  
 2   status          1 non-null      object  
 3   artifact_uri   1 non-null      object  
 4   start_time     1 non-null      datetime64[ns, UTC]
 5   end_time       1 non-null      datetime64[ns, UTC]
 6   params.learning_rate  1 non-null  object  
 7   params.epochs   1 non-null      object  
 8   params.activation  1 non-null  object  
 9   params.dropout  1 non-null      object  
 10  params.hidden_units 1 non-null  object  
 11  params.batch_size 1 non-null      object  
 12  params.model_type 1 non-null      object  
 13  params.optimizer 1 non-null      object  
 14  tags.mlflow.user 1 non-null      object  
 15  tags.mlflow.runName 1 non-null      object  
 16  tags.mlflow.source.type 1 non-null  object  
 17  tags.mlflow.source.name 1 non-null  object  
dtypes: datetime64[ns, UTC](2), object(16)
memory usage: 276.0+ bytes
```

```

import random

with mlflow.start_run(run_name="demo-metrics"):

    # บันทึก Parameters ก่อน
    mlflow.log_params({
        "learning_rate": 0.01,
        "epochs": 10,
        "batch_size": 64
    })

    # จำลองการ Training และบันทึก Metrics ทุก epoch
    print("🌟 เริ่มการจำลอง Training...")

    for epoch in range(10):
        # จำลองค่า Loss และ accuracy (ในการใช้งานจริงจะได้จากการ train)
        train_loss = 1.0 - (epoch * 0.08) + random.uniform(-0.05, 0.05)
        val_loss = 1.0 - (epoch * 0.07) + random.uniform(-0.05, 0.05)
        train_acc = 0.5 + (epoch * 0.04) + random.uniform(-0.02, 0.02)
        val_acc = 0.5 + (epoch * 0.035) + random.uniform(-0.02, 0.02)

        # บันทึก Metrics พร้อม step
        mlflow.log_metrics({
            "train_loss": train_loss,
            "val_loss": val_loss,
            "train_accuracy": train_acc,
            "val_accuracy": val_acc
        }, step=epoch)

        print(f" Epoch {epoch+1}/10 - train_loss: {train_loss:.4f}, val_loss: {val_loss:.4f}, train_acc: {train_acc:.4f}, val_acc: {val_acc:.4f}")

    # บันทึก Final Metrics
    mlflow.log_metrics({
        "final_accuracy": val_acc,
        "final_loss": val_loss
    })

    print("\n✅ Training เสร็จสิ้น!")
    print("📊 Final Accuracy: {val_acc:.4f}")

```

🌟 เริ่มการจำลอง Training...

Epoch 1/10 - train_loss: 0.9657, val_loss: 0.9951, train_acc: 0.5037, val_acc: 0.5006
 Epoch 2/10 - train_loss: 0.8975, val_loss: 0.9628, train_acc: 0.5503, val_acc: 0.5311
 Epoch 3/10 - train_loss: 0.8601, val_loss: 0.8126, train_acc: 0.5815, val_acc: 0.5516
 Epoch 4/10 - train_loss: 0.7395, val_loss: 0.7995, train_acc: 0.6359, val_acc: 0.6211
 Epoch 5/10 - train_loss: 0.7140, val_loss: 0.7004, train_acc: 0.6677, val_acc: 0.6473
 Epoch 6/10 - train_loss: 0.6404, val_loss: 0.6123, train_acc: 0.6941, val_acc: 0.6758
 Epoch 7/10 - train_loss: 0.5017, val_loss: 0.5989, train_acc: 0.7547, val_acc: 0.6939
 Epoch 8/10 - train_loss: 0.4280, val_loss: 0.5267, train_acc: 0.7809, val_acc: 0.7646
 Epoch 9/10 - train_loss: 0.3767, val_loss: 0.4464, train_acc: 0.8319, val_acc: 0.7661
 Epoch 10/10 - train_loss: 0.2600, val_loss: 0.3514, train_acc: 0.8759, val_acc: 0.8084

✅ Training เสร็จสิ้น!
 📊 Final Accuracy: 0.8084
 🌐 View run demo-metrics at: <http://127.0.0.1:8080/#/experiments/1/runs/1355e2db8aee48869256a03376faf4c1>
 🌐 View experiment at: <http://127.0.0.1:8080/#/experiments/1>

ส่วนที่ 4: การบันทึก Metrics

แนวคิด

Metrics คือค่าที่รับได้จากการทดลอง (Output/Results) สามารถบันทึกได้หลายครั้งพร้อม step number เพื่อติดตามการเปลี่ยนแปลงตามเวลา

ความแตกต่างระหว่าง Parameters และ Metrics

คุณสมบัติ	Parameters	Metrics
เวลาบันทึก	ก่อนเริ่มทดลอง	ระหว่าง/หลังทดลอง
จำนวนค่า	ค่าเดียวต่อชื่อ	หลายค่าได้ (ต่าง step)
การเปลี่ยนแปลง	คงที่	เปลี่ยนได้ตามเวลา
วัตถุประสงค์	บอกว่าท่าอะไร (Input)	บอกว่าได้ผลอย่างไร (Output)

ฟังก์ชันสำคัญ

ฟังก์ชัน	คำอธิบาย
<code>mlflow.log_metric(key, value, step)</code>	บันทึก Metric ค่าเดียว
<code>mlflow.log_metrics(dict, step)</code>	บันทึกหลาย Metrics พร้อมกัน

การใช้ Step Parameter

Step ทำให้ MLflow สามารถสร้างกราฟแสดงการเปลี่ยนแปลงของ metric ตามเวลาได้

mlflow-tracking-lab Machine learning ⓘ

State: Active Datasets Sort: Created Columns

	Run Name	Created	Dataset
<input type="checkbox"/>	demo-metrics	6 minutes ago	-
<input type="checkbox"/>	demo-parameters	46 minutes ago	-

mlflow™ 3.8.1

mlflow-tracking-lab > Runs > demo-metrics

Overview Model metrics System metrics Traces Artifacts

Description edit

No description

Metrics (6)

Metric	Value
train_loss	0.2599854330182305
val_loss	0.35141339328568155
train_accuracy	0.875930284106983
val_accuracy	0.8083779754296648
final_accuracy	0.8083779754296648
final_loss	0.35141339328568155

Parameters (3)

Parameter	Value
learning_rate	0.01
epochs	10
batch_size	64

About this run

Created at	01/01/2026, 07:55:34 PM
Created by	student
Experiment ID	1 copy
Status	Finished
Run ID	1355e2db8aee48869256a03376faf4c1 copy
Duration	0.7s
Source	ipykernel_launcher.py
Logged models	—
Registered prompts	—

Datasets

None

Tags

Add tags

Registered models

None

การอ่านข้อมูลจาก MLflow ใช้ Run ID โดยตรง

```
: import mlflow

mlflow.set_tracking_uri("http://127.0.0.1:8080")
# ตรวจสอบว่าเชื่อมต่อได้
print("Tracking URI:", mlflow.get_tracking_uri())

# ดึงข้อมูลจาก run_id ที่ได้
run_id = "1355e2db8aee48869256a03376faf4c1" # หรือ run_id ใดก็ได้
run = mlflow.get_run(run_id)

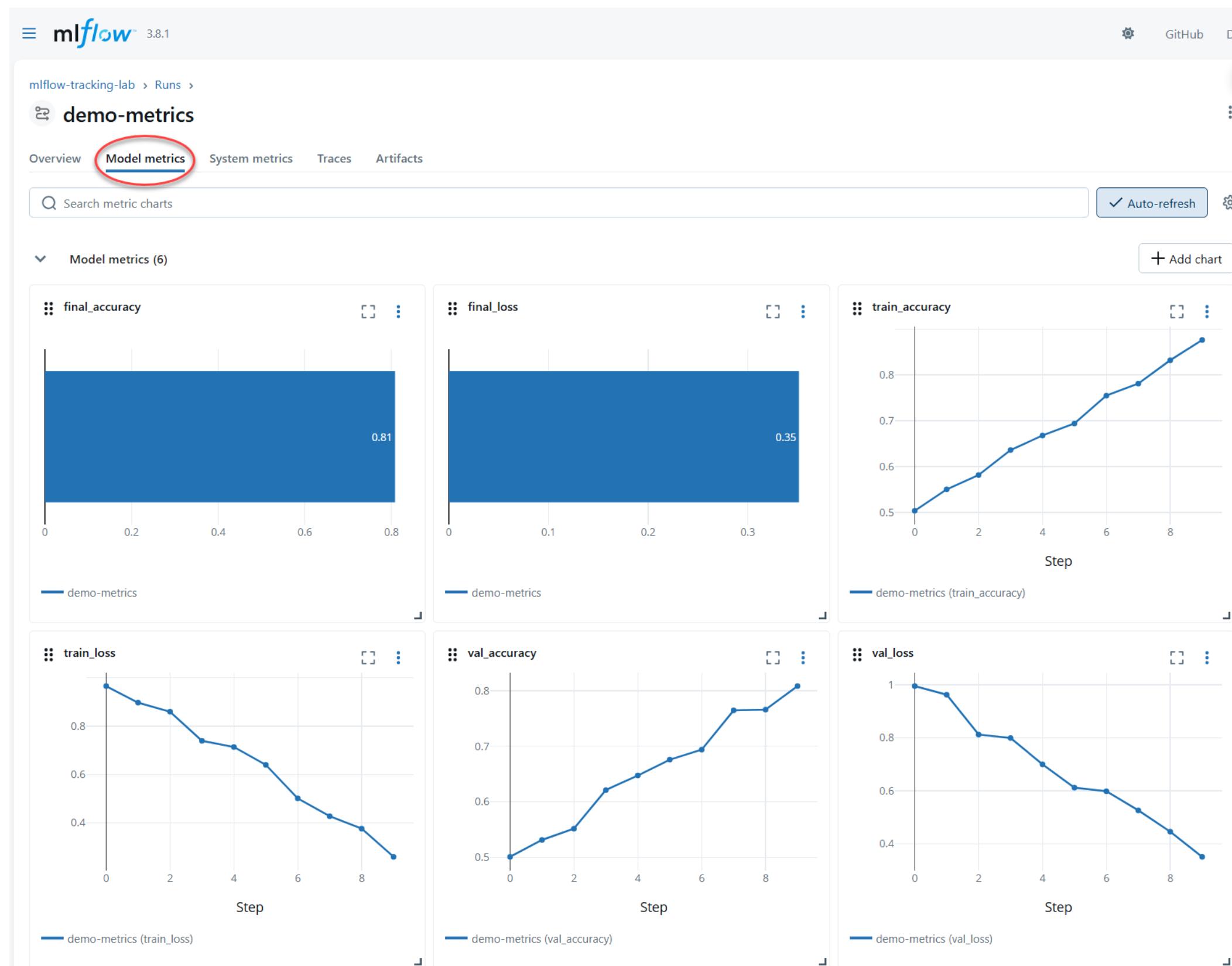
# อ่าน Parameters
print("Parameters:", run.data.params)

# อ่าน Metrics (เฉพาะค่าสุดท้าย)
print("Metrics:", run.data.metrics)

# ข้อมูล run อื่นๆ
print("Run Name:", run.info.run_name)
print("Status:", run.info.status)
print("Start Time:", run.info.start_time)

Parameters: {'learning_rate': '0.01', 'epochs': '10', 'batch_size': '64'}
Metrics: {'train_loss': 0.2599854330182305, 'val_loss': 0.35141339328568155, 'train_accuracy': 0.875930284106983, 'val_accuracy': 0.8083779754296648, 'final_accuracy': 0.8083779754296648, 'final_loss': 0.35141339328568155}
Run Name: demo-metrics
Status: FINISHED
Start Time: 1767272134227
```

การอ่านข้อมูลจาก MLflow ดึง Metric History (ทุก step)



```
: import mlflow
from mlflow.tracking import MlflowClient

# ต้องระบุ tracking URI ก่อนสร้าง client
mlflow.set_tracking_uri("http://127.0.0.1:8080")

client = MlflowClient()
run_id = "1355e2db8aee48869256a03376faf4c1"

history = client.get_metric_history(run_id, "train_loss")
for m in history:
    print(f"Step {m.step}: {m.value:.4f}")
```

Step 0: 0.9657
Step 1: 0.8975
Step 2: 0.8601
Step 3: 0.7395
Step 4: 0.7140
Step 5: 0.6404
Step 6: 0.5017
Step 7: 0.4280
Step 8: 0.3767
Step 9: 0.2600

```
: from mlflow.tracking import MlflowClient
import pandas as pd

client = MlflowClient()
run_id = "1355e2db8aee48869256a03376faf4c1" # หรือ run_id เดิม
```

```
# สร้าง DataFrame จาก metric history
metrics_data = {}
for metric_name in ["train_loss", "val_loss", "train_accuracy", "val_accuracy"]:
    history = client.get_metric_history(run_id, metric_name)
    metrics_data[metric_name] = [m.value for m in history]
```

```
df = pd.DataFrame(metrics_data)
```

```
df.index.name = "epoch"
```

```
print(df)
```

epoch	train_loss	val_loss	train_accuracy	val_accuracy
0	0.965741	0.995146	0.503669	0.500636
1	0.897483	0.962763	0.550318	0.531058
2	0.860108	0.812604	0.581517	0.551557
3	0.739505	0.799473	0.635928	0.621111
4	0.713990	0.700421	0.667724	0.647310
5	0.640437	0.612332	0.694094	0.675755
6	0.501693	0.598935	0.754742	0.693862
7	0.428044	0.526652	0.780856	0.764595
8	0.376678	0.446439	0.831931	0.766116
9	0.259985	0.351413	0.875930	0.808378