دانشکار

{ یک یادگیری کارساز }

بوتکمپ

# Artificial Intelligence

# MLOps

Session 3

# Welcome & Recap

## Environment Management & Dependencies

- Quick recap: Session 2 covered ML workflows & version control
- Today's focus: Solving "It works on my machine" problems
- Poll: Who has experienced dependency issues?

# What You'll Learn Today

**Objectives:**

- Understand dependency hell in ML projects
- Master virtual environment creation and management
- Learn containerization concepts and benefits
- Achieve environment consistency across dev/staging/production
- Apply configuration management principles

# The "It Works on My Machine" Problem

## A Day at VisionaryAI

Sarah's Machine

```
├── TensorFlow 2.8
├── OpenCV 4.5
├── Python 3.8
├── CUDA 11.2
└── Ubuntu 20.04
```

Your Machine

```
├── TensorFlow 2.12
├── OpenCV 4.7
├── Python 3.10
├── CUDA 11.8
└── macOS Monterey
```

| | |
|---|---|
| **Key Points** | Same code, different environments = different results |
| | In ML: Version differences can change model behavior |
| | Traditional software: annoying<br>ML systems: catastrophic |

# What is Dependency Hell?

## Dependency Hell in ML Context

**Multiple Layers:**

- System-level: OS libraries, CUDA drivers, system Python
- Python packages: ML libraries with complex interdependencies
- Version constraints: Package compatibility matrices

**ML-Specific Challenges:**

- Model reproducibility across versions
- GPU compatibility requirements
- Pre-trained model version dependencies
- Research vs production library stability

# Real Dependency Conflict Example

## VisionaryAI NLP Project Dependency Web

**Conflict Visualization:**

- `transformers==4.20.0 → requires torch>=1.9.0`
- `torch==1.13.0 → requires numpy>=1.21.0`
- `pandas==1.4.0 → requires numpy>=1.18.5,<1.25.0`
- `System has numpy==1.25.2` ❌

**Impact**

- Import errors and crashes
- Silent behavior changes
- Development workflow disruption

# Virtual Environments Introduction

**What is a Virtual Environment?**

- Self-contained directory with:
  - Specific Python interpreter
  - Project-specific packages
  - Activation/deactivation scripts
  - Isolation from system Python

# Why Virtual Environments Matter for ML

## Benefits for ML Projects

**Key Benefits:**

- Isolation: Each project gets its own dependencies
- Reproducibility: Anyone can recreate exact environment
- Experimentation safety: Test new packages without breaking existing projects
- Version control: Track environment changes with code

**VisionaryAI Example:**

- Computer Vision: TensorFlow 2.8
- NLP: TensorFlow 2.12
- Recommender: scikit-learn 1.0
- No conflicts!

# Types of Virtual Environments

## Choosing the Right Tool

| Tool | Best For | Strengths |
|---|---|---|
| venv | Simple Python projects | Built-in, lightweight |
| conda | Data science/ML workflows | System dependencies, better resolution |
| Poetry | Modern applications | Advanced dependency management |

**Decision Matrix:**

- Simple ML learning → venv
- Complex ML projects → conda
- Production applications → Poetry

# Containerization Introduction

Beyond Virtual Environments

- **The Problem Virtual Environments Don't Solve:**
  - System libraries and dependencies
  - Operating system differences
  - GPU drivers and CUDA versions
  - Database drivers and system tools
- **Solution: Package EVERYTHING in a container**
  - Virtual environments = rooms in a house
  - Containers = entire portable houses

# Docker Basics for ML

## Docker Concepts

**Key Components**

- Dockerfile: Recipe for building container
- Image: Built container template
- Container: Running instance of image

**Analogy**

- Dockerfile = Cooking recipe
- Image = Cake mold
- Container = Actual cake

# Why Containerization Matters for ML

## Complete Environment Control

**Benefits:**

- Complete isolation: OS + system libraries + Python environment
- True reproducibility: Same container = identical behavior
- Simplified deployment: Deploy anywhere Docker runs
- Easy scaling: Multiple instances for high traffic

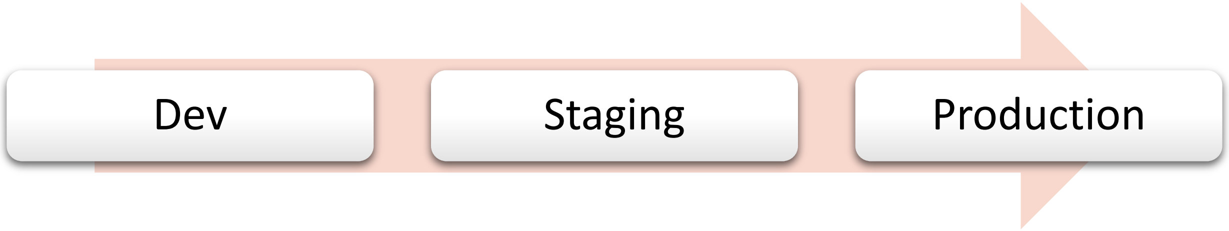
**VisionaryAI Use Case: NLP Helpdesk**

**Agent container includes:**

- Ubuntu 20.04 + Python 3.9 + transformers + torch + custom libraries
- Runs identically on: Mac laptops, Linux staging, Kubernetes production

# Environment Consistency Challenge

- **The Three Environments**
  - Development: Fast iteration, debugging tools, local machine
  - Staging: Production-like testing, integration validation
  - Production: Real users, high availability, security focus
- **ML-Specific Risks**
  - Different model predictions for same input
  - Silent failures with wrong results
  - Performance degradation hard to trace



# VisionaryAI Consistency Example

## Computer Vision Defect Detection Inconsistency

**Problem Scenario:**

- `Dev Environment:     OpenCV 4.8 (latest features)`
- `Staging Environment: OpenCV 4.5 (stability testing)`
- `Production:          OpenCV 4.6 (performance optimized)`

**Result:**

- Same factory image → Different defect detection results

**Solution:**

- Containerized environments with identical base images

# Achieving Environment Consistency

## Best Practices for Consistency

**Strategies**

- Infrastructure as Code: Define environments in version-controlled files
- Containerization: Same image across all environments
- Environment parity: Dev mirrors production closely
- Automated deployment: Eliminate manual configuration errors

**Key Principle**

- Only configuration should change between environments, not the underlying software stack

# Configuration Management

## Separating Config from Code

- **What is Configuration in ML?**
  - Model parameters and hyperparameters
  - Data connection strings and file paths
  - Environment settings (APIs, resources, logging)
  - Infrastructure settings (servers, authentication)
- **12-Factor App Principles:**
  - Store config in environment variables
  - Strict separation of config from code
  - Config varies substantially across environments

# VisionaryAI Configuration Example

## NLP Helpdesk Agent Configurations

**Development:**

```
DATABASE_URL=sqlite:///local.db
HUGGINGFACE_API_KEY=dev_key_123
LOG_LEVEL=DEBUG
MAX_WORKERS=2
```

**Production:**

```
DATABASE_URL=postgresql://prod.cluster.com/db
HUGGINGFACE_API_KEY=prod_key_456
LOG_LEVEL=INFO
MAX_WORKERS=20
```

**Same code,**
**different behavior through configuration**

# Configuration Management Tools

## Tools and Best Practices

### Configuration Options

- Environment variables: Simple, widely supported
- Configuration files: YAML/JSON for complex configs
- Management tools:Consul, AWS Parameter Store, K8s ConfigMaps

### Security Best Practices

- Never hardcode sensitive information
- Use secret management systems
- Rotate secrets regularly
- Validate configurations at startup

# Demo Transition

Time for Hands-On Practice!

**What We'll Build:**

- Create virtual environments for VisionaryAI projects
- Compare requirements.txt vs conda environments
- Build a basic Dockerfile for ML application
- See configuration management in action

**Progressive approach:** Start simple → Build complexity

# Key Takeaways

**Remember These Points**

- Dependency hell is expensive in ML (behavior changes)
- Virtual environments provide Python package isolation
- Containers ensure complete environment consistency
- Environment consistency across dev/staging/production is critical
- Configuration management enables flexibility and security

**ROI:**
Time spent on environment management saves exponentially more time later

# Next Session Preview

## Coming Up: Data Management & Experiment Tracking

### Session 4 Topics

- Data versioning and lineage
- Experiment management and reproducibility
- Tracking parameters, metrics, and artifacts
- Building on today's environment foundation

### Preparation

- Think about current project environment challenges

### Questions?

- Bring your environment management challenges to discuss!


دانشگاه
THANK YOU
FOR YOUR ATTENTION!