

# Packaging ML models

DEVELOPING MACHINE LEARNING MODELS FOR PRODUCTION



**Sinan Ozdemir**

Data Scientist, Entrepreneur, and Author

# Why packaging matters

- Optimize performance
- Ensure compatibility
- Makes models easier to deploy

## Packaging Methods

- Serialization: simple, lightweight, and language-agnostic
- Environment packaging: capture entire software environments
- Containerization: a portable, reproducible, and isolated environment

# How to package ML models

- Serialization - storing and retrieving an ML model
- Environment packaging - consistent and reproducible environment for the ML Model
- Containerization - packaging the model, dependencies, and the environment in a single "container"

# Serializing scikit-learn models

Serializing an sklearn model using `pickle`:

```
import pickle

model = ... # Train the scikit-learn model

# Serialize the model to a file
with open('model.pkl', 'wb') as f:
    pickle.dump(model, f)

# Load the serialized model from the file
with open('model.pkl', 'rb') as f:
    model = pickle.load(f)
```

Serializing an sklearn model in `HDF5` format:

```
import h5py
import numpy as np
from sklearn.externals import joblib

model = ... # Train the scikit-learn model

# Serialize the model to an HDF5 file
with h5py.File('model.h5', 'w') as f:
    f.create_dataset('model_weights',
                     data=joblib.dump(model))

# Load the serialized model from the HDF5 file
with h5py.File('model.h5', 'r') as f:
    model = joblib.load(f['model_weights'][:])
```

# Serializing PyTorch and Tensorflow models

Serializing a PyTorch model:

```
import torch

# Train a PyTorch model and store it in a variable
trained_model = ...

# Serialize the trained model to a file
serialized_model_path = 'model.pt'
torch.save(trained_model.state_dict(), serialized_m

# Load the serialized model from a file
loaded_model = ... # Initialize the model
loaded_model.load_state_dict(
    torch.load(serialized_model_path))
```

Serializing a Tensorflow model:

```
import tensorflow as tf

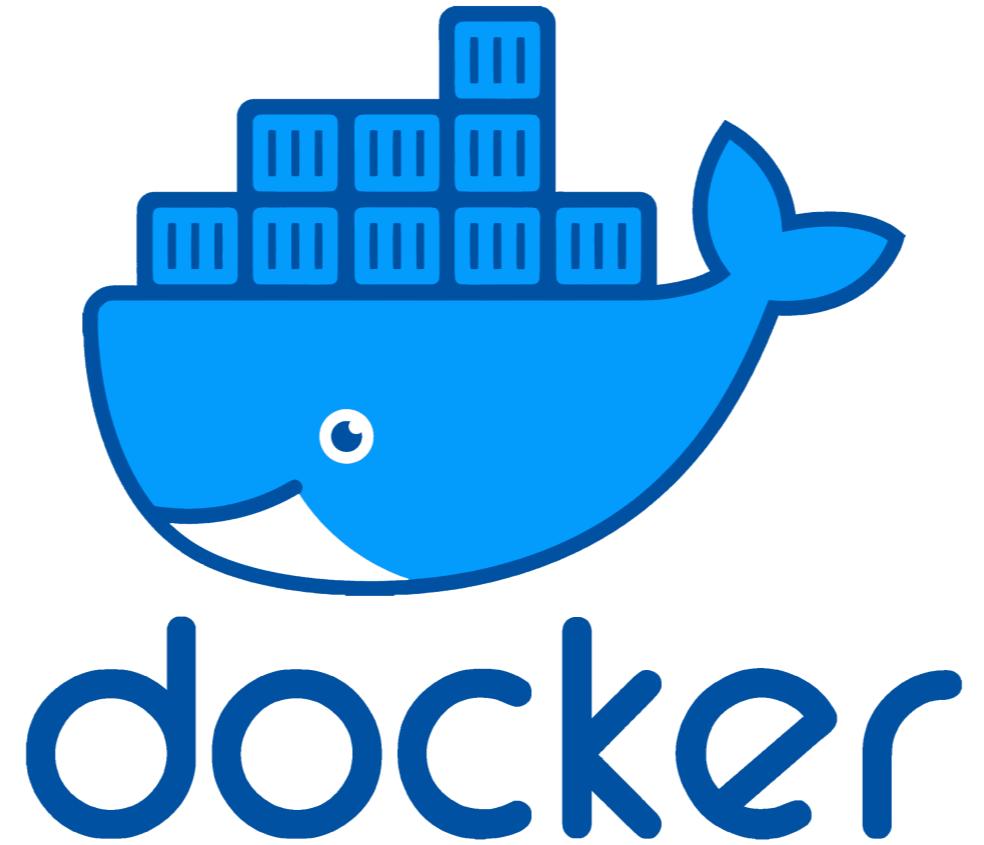
# Train a Tensorflow model
trained_model = ...

# Save the trained model to a directory
saved_model_directory = 'model/'
tf.saved_model.save
    (trained_model, saved_model_directory)

# Load the saved model from the directory
loaded_model = tf.saved_model.load(
    saved_model_directory)
```

# ML environment packaging with Docker

- Ensure the model's environment lets it run
- virtualenv etc. create consistent and reproducible environments
- Docker containers are self-contained units that are easily deployable



# Example Dockerfile

```
# Use an existing image as the base image
FROM python:3.8-slim

# Set the working directory
WORKDIR /app

# Copy the requirements file to the image
COPY requirements.txt .

# Install the required dependencies
RUN pip install -r requirements.txt

# Copy the ML model and its dependencies to the image
COPY model/ .

# Set the entrypoint to run the model
ENTRYPOINT ["python", "run_model.py"]
```

<---- Use Python 3.8 base image

<---- Set the working directory

<---- Copy the requirements.txt file

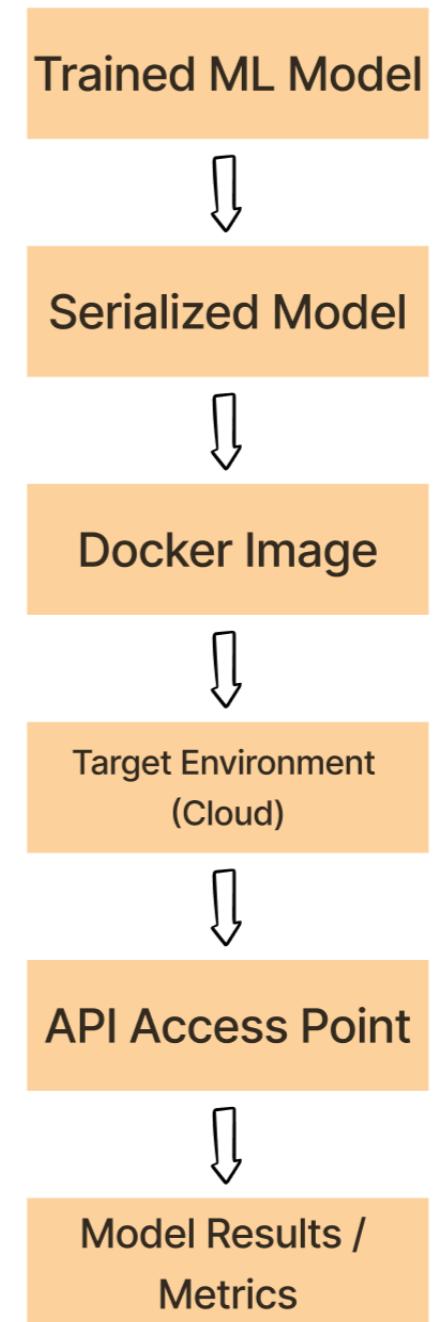
<---- Install the model's dependent packages

<---- Copy the model into the container

<---- Tell the container how to start up

# Experiment -> Docker workflow

1. **Serialize** the trained ML model using a format such as pickle, HDF5, or PyTorch.
2. **Containerize** the serialized ML model, its dependencies, and the environment
3. **Deploy** the Docker image to a target environment such as a cloud platform
4. **Run** the Docker container from the deployed Docker image and run the ML model.
5. **Use** the model within the container from an API or some other access point.



# **Let's practice!**

**DEVELOPING MACHINE LEARNING MODELS FOR PRODUCTION**

# Scalability

DEVELOPING MACHINE LEARNING MODELS FOR PRODUCTION



**Sinan Ozdemir**

Data Scientist, Entrepreneur, and Author

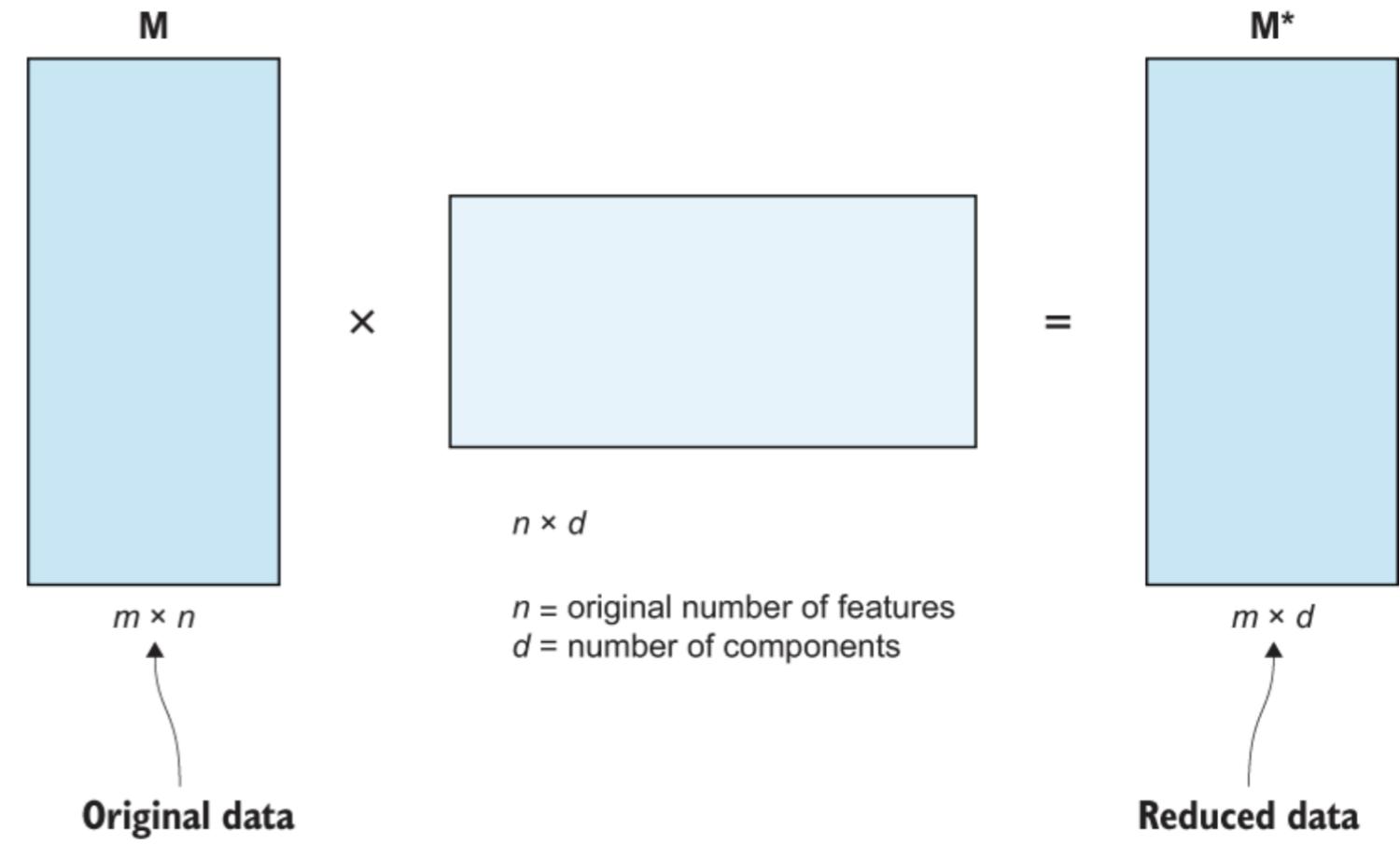
# Compute constraints in scalability

- CPU, memory, and disk requirements can impact ML model scalability
- Measure CPU, memory, and disk usage during training and inference
- Early identification leads to better model scalability



# Model complexity and scalability

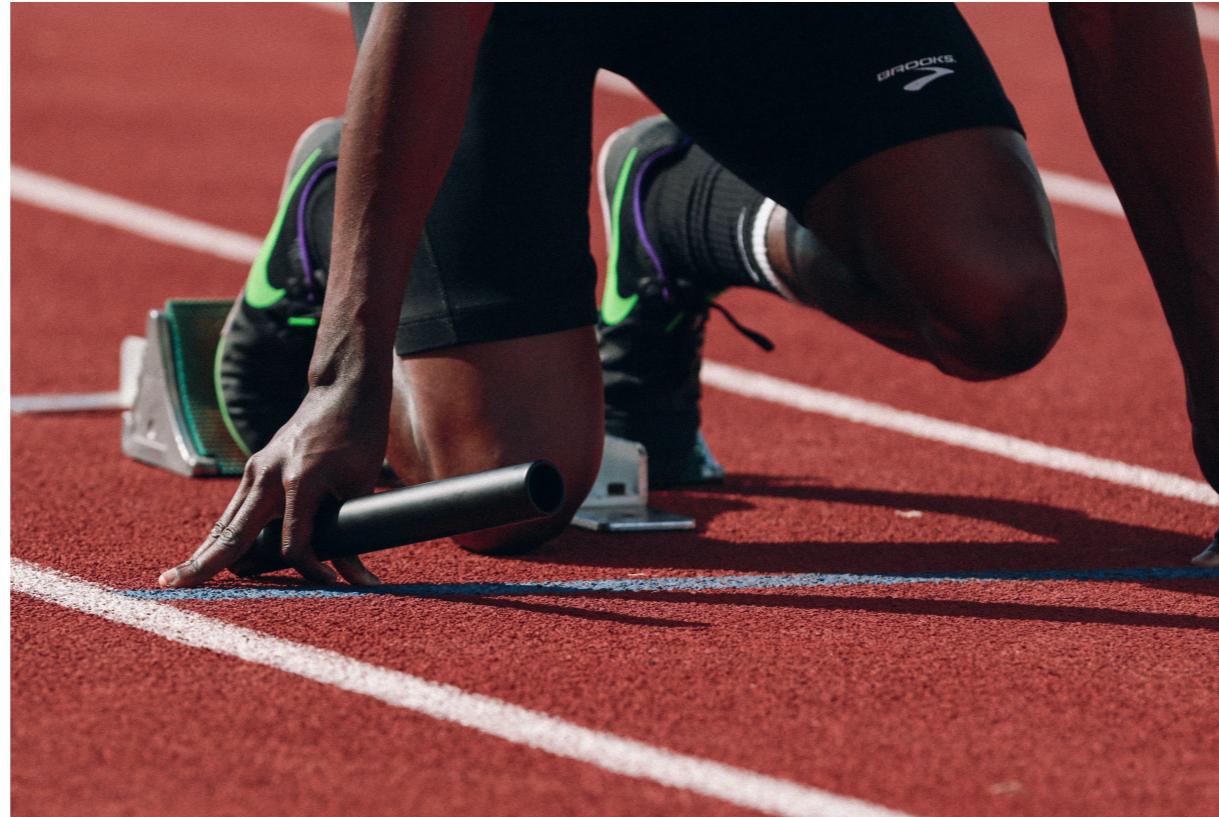
- Model complexity affects scalability
- Balancing complexity and scalability is challenging
- Strategies for balancing:
  - Feature selection techniques (e.g. Chi-squared, PCA)
  - Model compression techniques (e.g. pruning)



Principal Component Analysis (PCA)

# Velocity of deployments and scalability

- Rapid deployment and iteration is crucial for relevance and accuracy
- Strategies
  - continuous integration/deployment - covered in the next lesson
  - online learning - learning with new data in real time



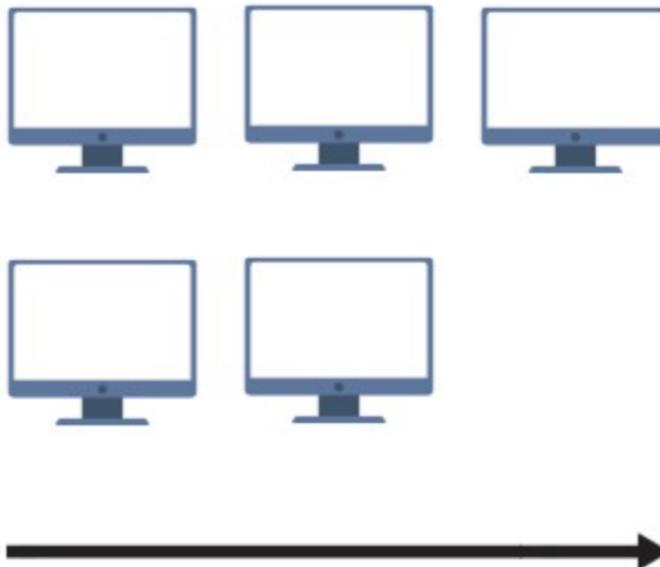
# Optimal scaling strategies

- Trade-offs include cost of serving, re-training, and velocity of deployments
- Scaling strategies: horizontal scaling, vertical scaling, auto-scaling
  - Horizontal scaling: adding more machines
  - Vertical scaling: increasing machine size
  - Auto-scaling: adjusts number of machines based on workload

## Horizontal Scaling vs. Vertical Scaling

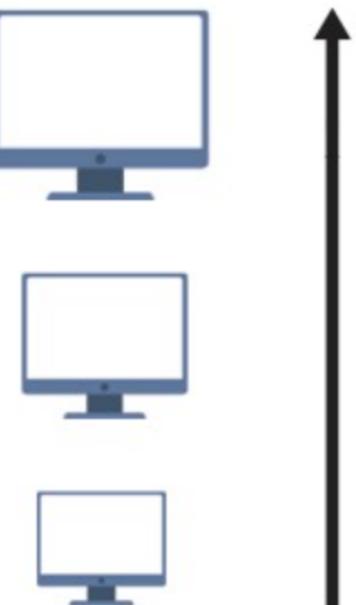
### Horizontal Scaling

Add more instances



### Vertical Scaling

Increase size of instances  
(RAM, CPU, etc.)



Horizontal vs. Vertical Scaling

<sup>1</sup> <https://www.spiceworks.com/tech/cloud/articles/horizontal-vs-vertical-cloud-scaling>

# Considering your optimal scaling strategy

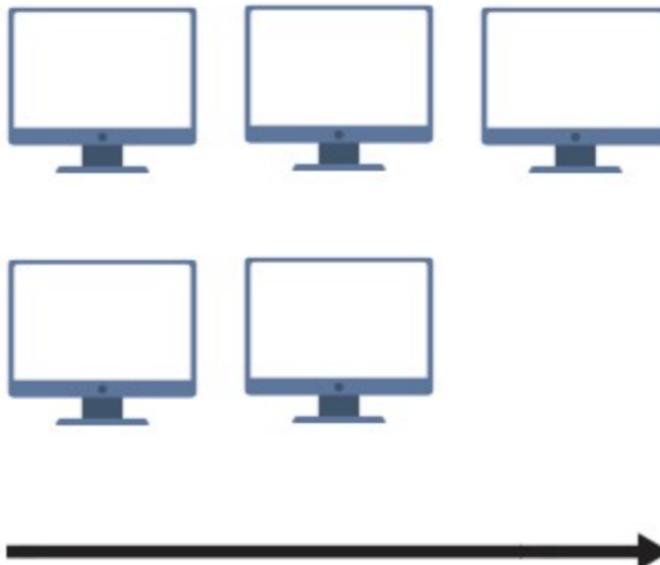
- Horizontal scaling uses load balancing, partitioning
- Horizontal scaling trade-offs: complexity, cost (\$)
- Vertical scaling increases machine size
- Vertical scaling trade-off: cost (\$)

Ideally - auto-scale horizontally and vertically.

## Horizontal Scaling vs. Vertical Scaling

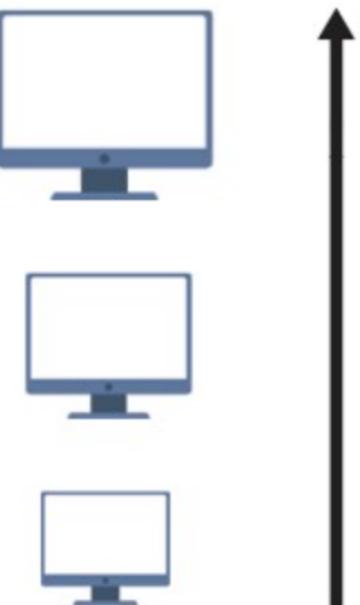
### Horizontal Scaling

Add more instances



### Vertical Scaling

Increase size of instances  
(RAM, CPU, etc.)



Horizontal vs. Vertical Scaling

<sup>1</sup> <https://www.spiceworks.com/tech/cloud/articles/horizontal-vs-vertical-cloud-scaling>

# **Let's practice!**

**DEVELOPING MACHINE LEARNING MODELS FOR PRODUCTION**

# Automation

DEVELOPING MACHINE LEARNING MODELS FOR PRODUCTION



**Sinan Ozdemir**

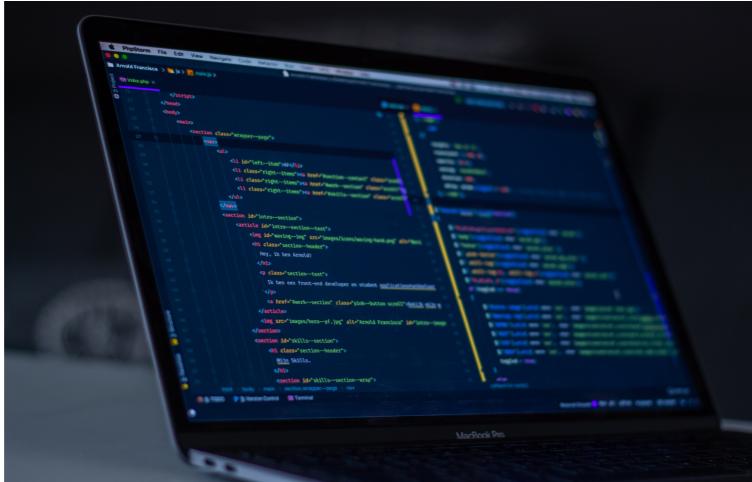
Data Scientist, Entrepreneur, and Author

# Introduction to ML automation

- Automation ensures reliability & efficiency of ML pipelines
- Reduces risk of human error
- Streamlines development & deployment process
- The four main MLOps principles are:
  - Continuous Integration (CI)
  - Continuous Deployment (CD)
  - Continuous Training (CT)
  - Continuous Monitoring (CM)

# Four principles of automation

**Continuous Integration (CI)** Integrating code changes into a shared repository regularly



**Continuous Training (CT)** Continuously training & updating the model with new data



**Continuous Delivery (CD)** Automatically building, testing, and deploying code changes



**Continuous Monitoring (CM)** Monitoring model performance and accuracy on an ongoing basis



# Continuous integration and delivery

## Continuous Integration (CI)

- Ensures code always in working state
- Reduces risk of human error
- Catches issues early

## Continuous Delivery (CD)

- Quick & consistent model deployment
- Reduces time to bring models to production
- Reduces risk of human error

CI/CD tools: Git, AWS CodePipeline, Jenkins, Travis CI



# Jenkins

# Continuous training and monitoring

## Continuous Training (CT)

- Ensures model accuracy & up-to-date
- Reduces risk of model decay
- Reduces time to re-train

## Continuous Monitoring (CM)

- Reduces risk of model decay
- Improves overall accuracy
- Access to consistent & reliable ML metrics
- Identifies issues early



# Example of ML automation at scale

1. CI: The code for the model is committed to Git.
2. CD: The committed code is built and tested using a CI/CD tool like Jenkins. If they pass, we deploy.
  - The model is serialized.
  - Dependencies are set using Docker.
  - The Docker image is deployed.
3. CM: Model performance is continuously monitored.
  - Monitoring informs decisions about the model.
  - CM tools include Prometheus & Grafana
4. CT: The model is trained on new data.
5. New code is written / models are updated ... back to step 1

# **Let's practice!**

**DEVELOPING MACHINE LEARNING MODELS FOR PRODUCTION**