MARSAIL
MARS Artificial Intelligence Laboratory

สวทช NSTDA

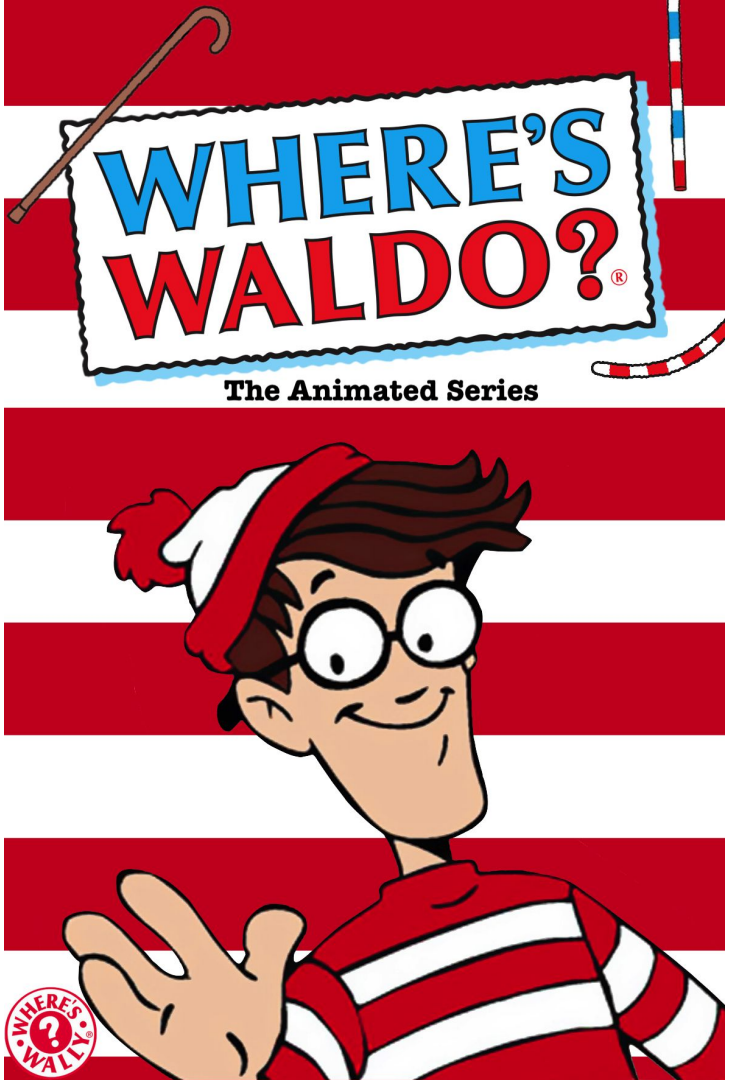
WALDO AI

Teerapong Panboonyuen, Ph.D.

https://github.com/kaopanboonyuen/where-is-waldo

Outline

- Basic AI
- Where is Waldo?
- Waldo AI Lab
- Challenge: Fine-Tuning to Find the Winner
- (Bonus) Optimizing Model Training
- (Bonus) 10 Practical Fine-Tuning Techniques



WHERE'S WALDO?®

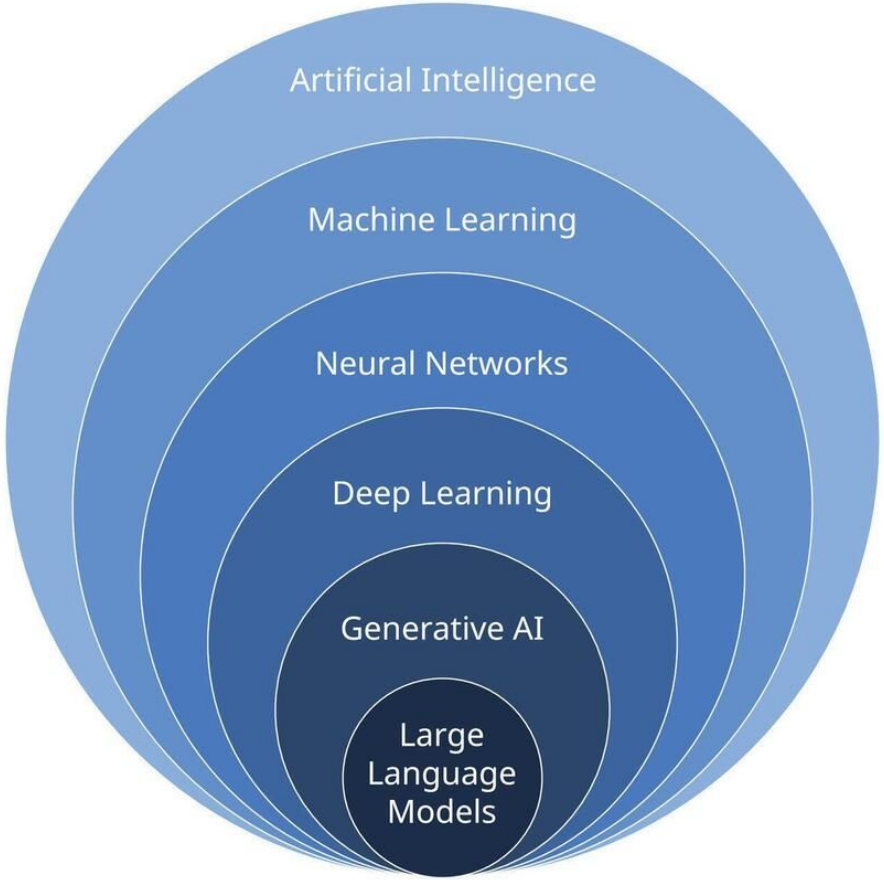
The Animated Series

WHERE'S WALDO?®

16-MONTH
2021
CALENDAR

Overview of AI in Computer Vision

- **Artificial Intelligence (AI)** enables machines to interpret and understand visual data.
- **Computer Vision (CV)** is the field that allows computers to extract, analyze, and process visual information from the world.


Artificial Intelligence
Machine Learning
Neural Networks
Deep Learning
Generative AI
Large Language Models

What is AI?

ANI vs. AGI vs. ASI

Artificial narrow intelligence (ANI)

Designed to perform specific tasks

Artificial general intelligence (AGI)

Can behave in a human-like way across all tasks

Artificial super intelligence (ASI)

Smarter than humans—the stuff of sci-fi

_zapier

TECH

‘Godfathers of AI’ honored with Turing Award, the Nobel Prize of computing

From left to right: Yann LeCun | Photo: Facebook; Geoffrey Hinton | Photo: Google; Yoshua Bengio | Photo: Botler AI

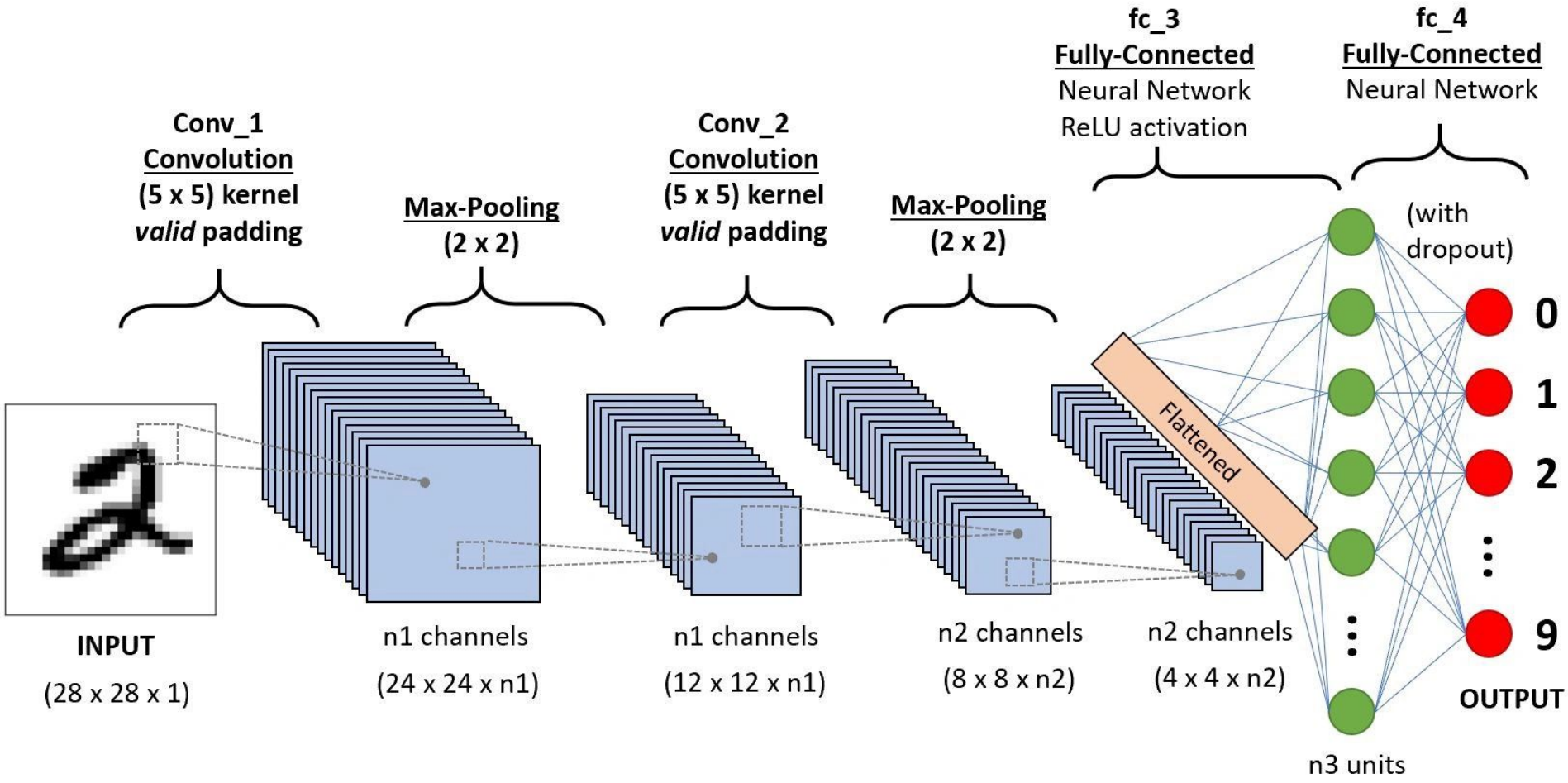
/ Yoshua Bengio, Geoffrey Hinton, and Yann LeCun laid the foundations for modern AI

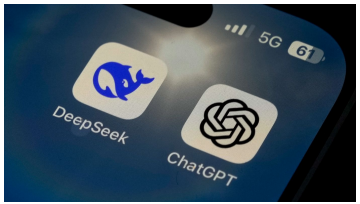
by **James Vincent**

Mar 27, 2019, 5:02 PM GMT+7

0 Comments

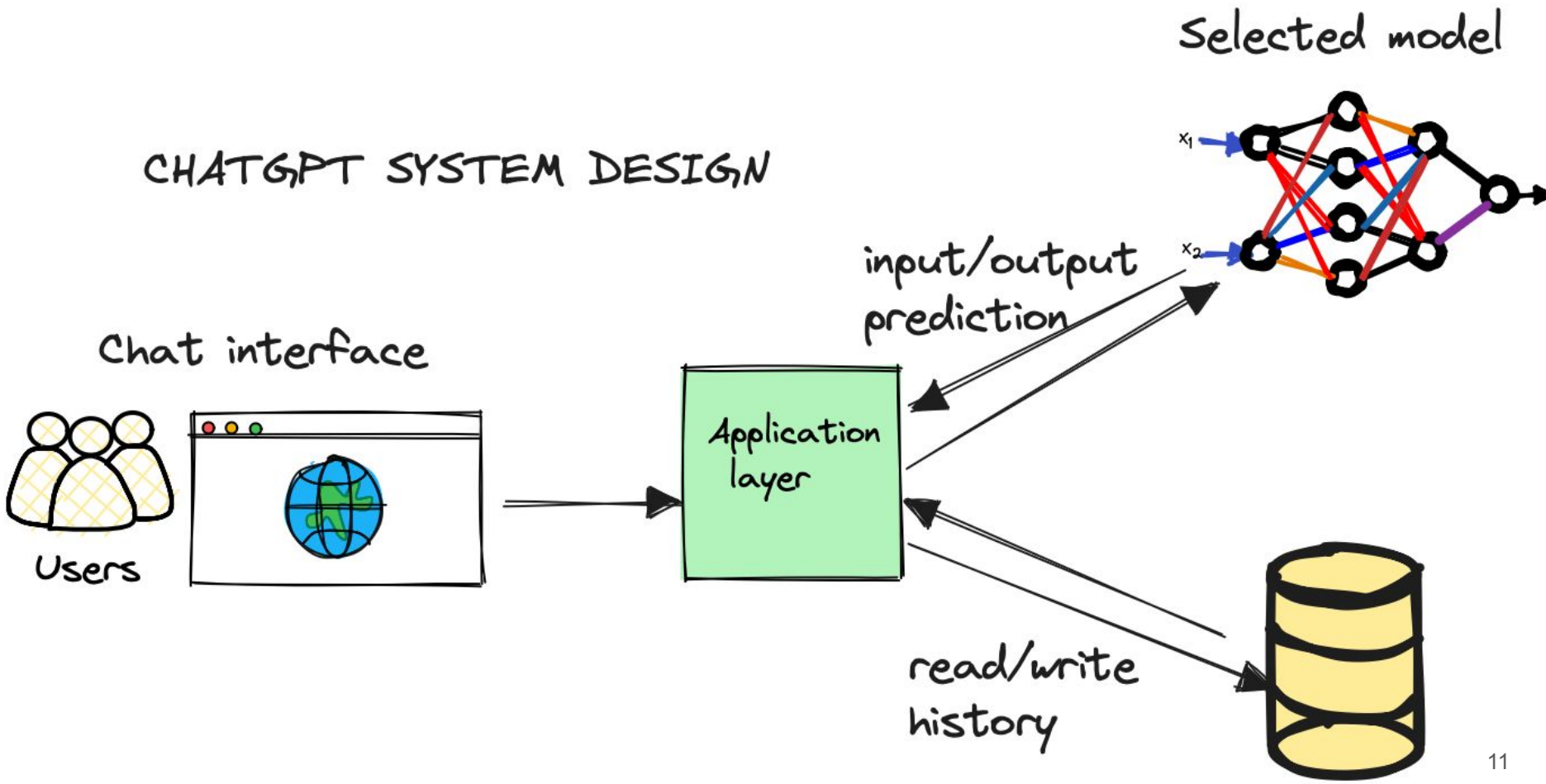

"If your are interested in AGI, don't work on LLMs"
The Next AI Revolution
Yann LeCun


Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
(with dropout)
Flattened
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units
0
1
2
9
OUTPUT

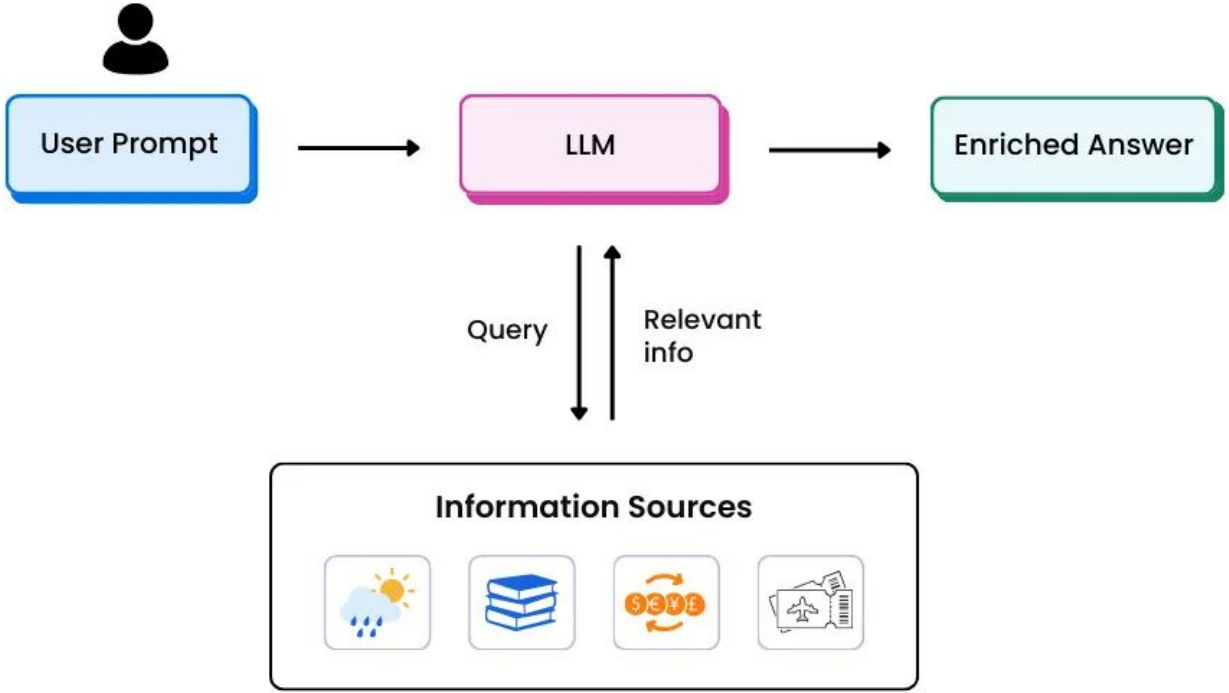


PaLM 2 / BERT	Google
DALL•E / GPT- 4	OpenAI
LLaMA	Meta
Claude	ANTHROP\C
Dolly	databricks
RedPajama	TOGETHER
MPT- 7B	mosaicML

CHATGPT SYSTEM DESIGN

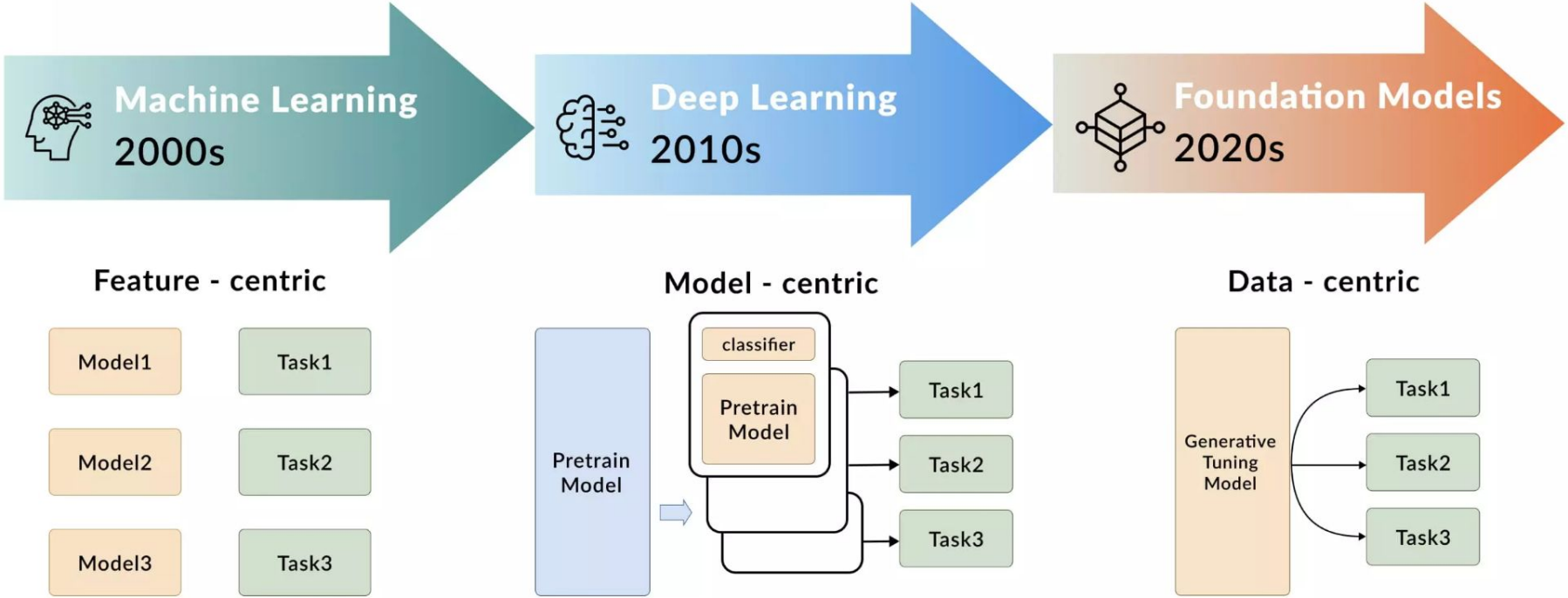


Retrieval-Augmented Generation (RAG) in LLMs



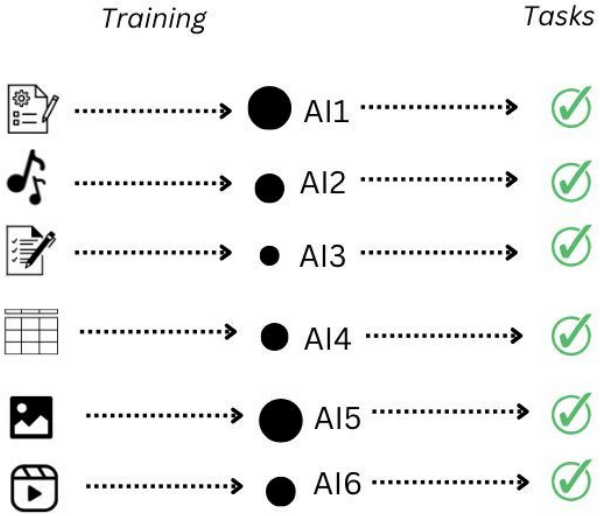
A New Era of AI: Foundation Models

Step function improvements over legacy AI technologies



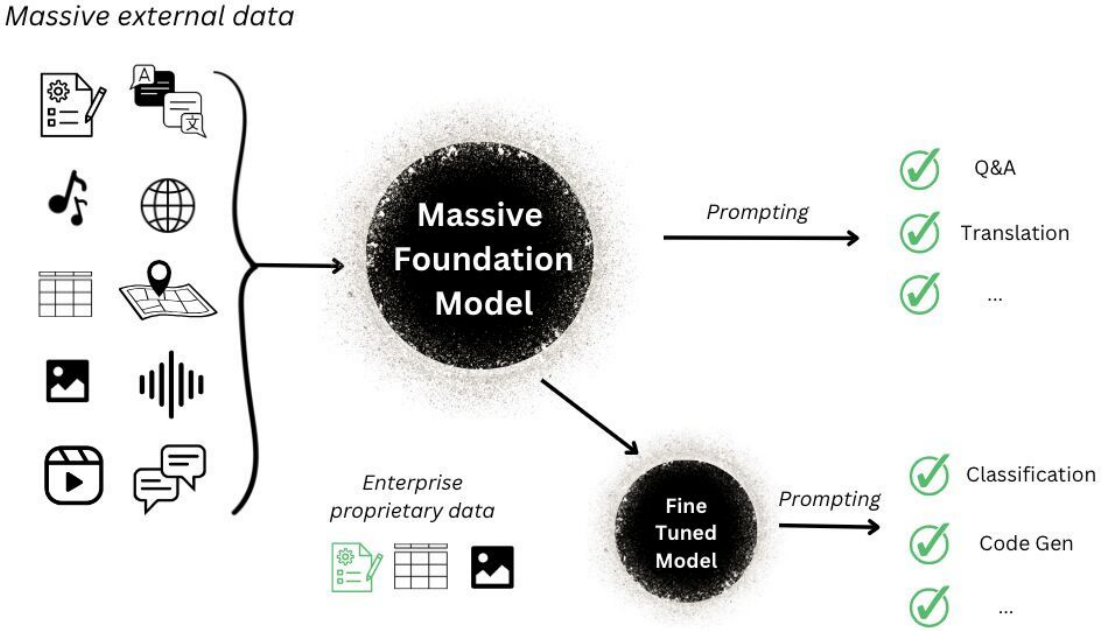
DATAFOREST

Traditional ML

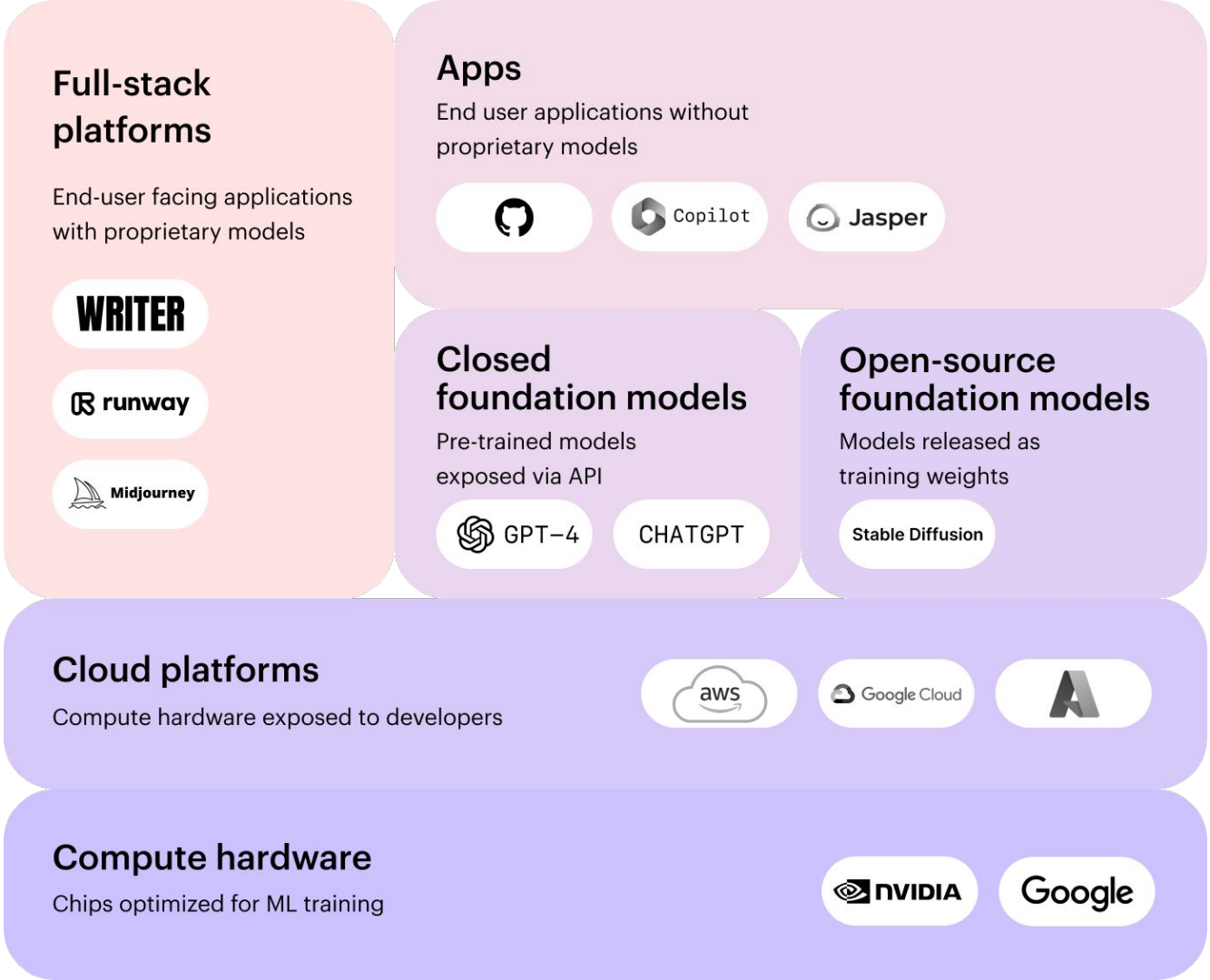


- Individual siloed models
- Require task-specific training
- Lots of human supervised training

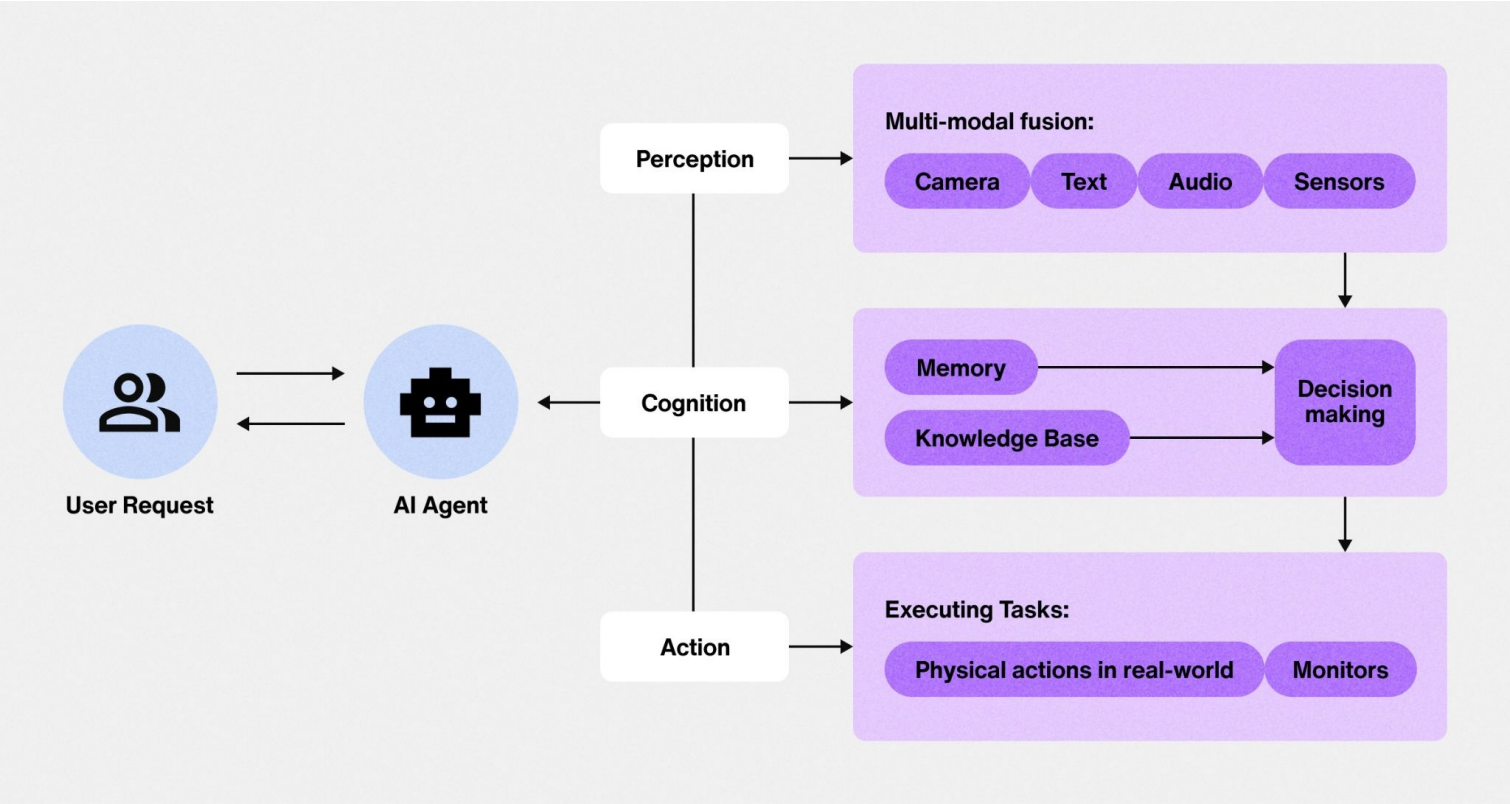
Foundation Models



- Massive multi-tasking model
- Adaptable with little or no training
- Pre-trained unsupervised learning


Full-stack platforms
End-user facing applications with proprietary models
WRITER
runway
Midjourney
Apps
End user applications without proprietary models
Copilot
Jasper
Closed foundation models
Pre-trained models exposed via API
GPT-4
CHATGPT
Open-source foundation models
Models released as training weights
Stable Diffusion
Cloud platforms
Compute hardware exposed to developers
aws
Google Cloud
Compute hardware
Chips optimized for ML training
NVIDIA
Google

Agentic AI

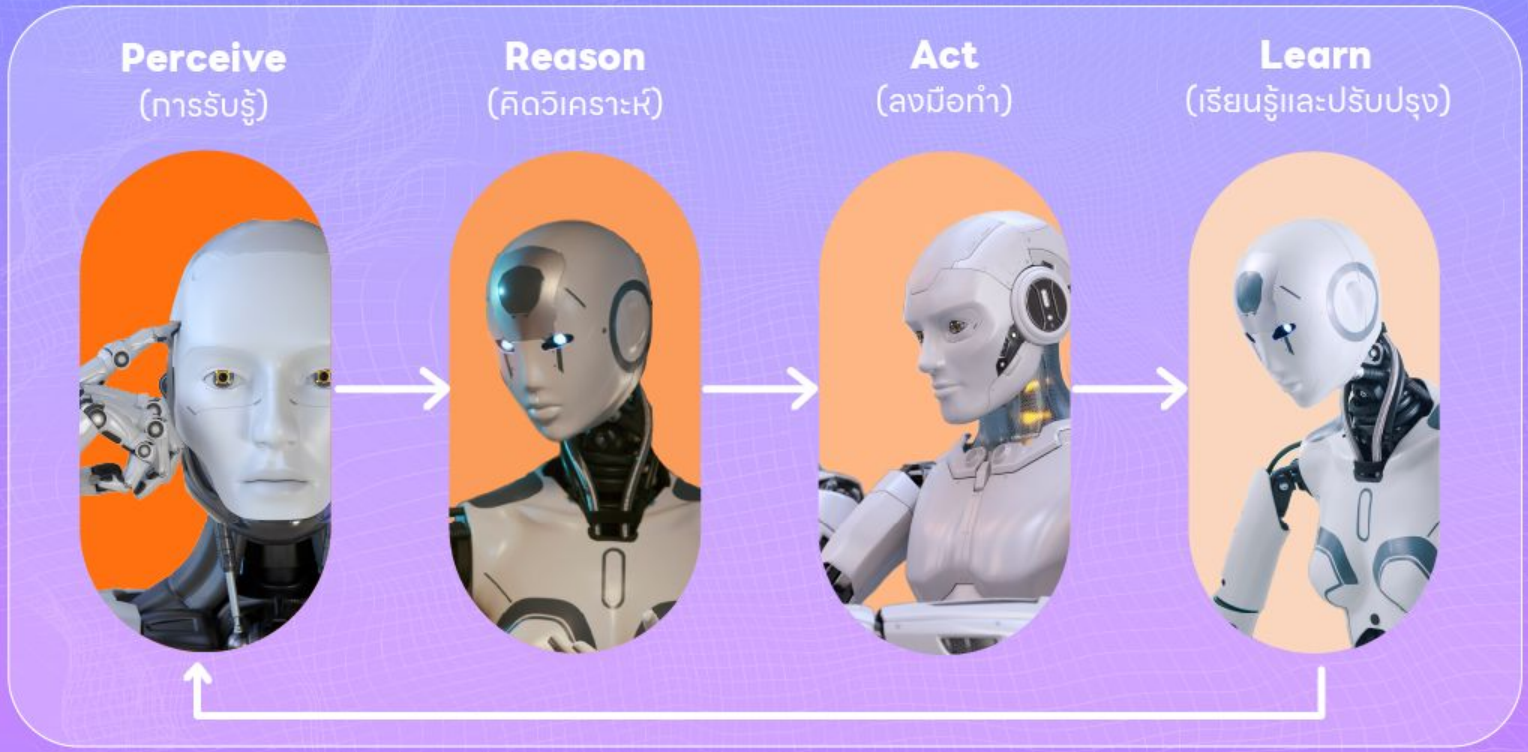

Perception
Multi-modal fusion:
Camera
Text
Audio
Sensors
User Request
AI Agent
Cognition
Memory
Knowledge Base
Decision making
Action
Executing Tasks:
Physical actions in real-world
Monitors

Generative AI

หลักการทำงานของ Agentic AI

4 ขั้นตอนการทำงานของ Agentic AI

9Expert



บทความโดย: 9Expert Training

Never Stop Learning

What is Model Distillation?

Created by genuine impact

Model distillation is the process of transferring knowledge from a large teacher model to a smaller student model to improve efficiency while maintaining performance.

- "Runs faster on smaller devices."
- "Consumes less energy."
- "Maintains good accuracy."

Big Teacher Model

A large and powerful AI model that is highly accurate but slow and resource-intensive.

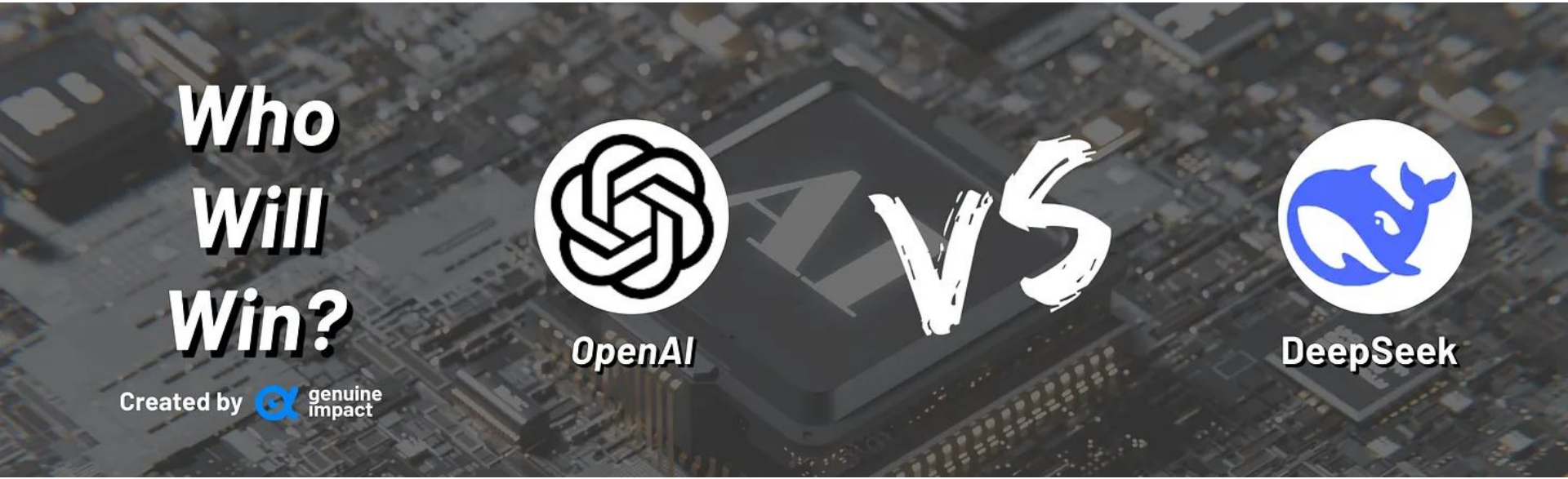
Knowledge Transfer

The teacher passes its knowledge (important patterns and decisions) to a smaller model.

Small Student Model

A lightweight and fast model that learns from the teacher's knowledge, becoming almost as good but much more efficient.

More charts: genuineimpact.substack.com


Who
Will
Win?
OpenAI
VS
DeepSeek
Created by genuine impact

DeepSeek vs OpenAI

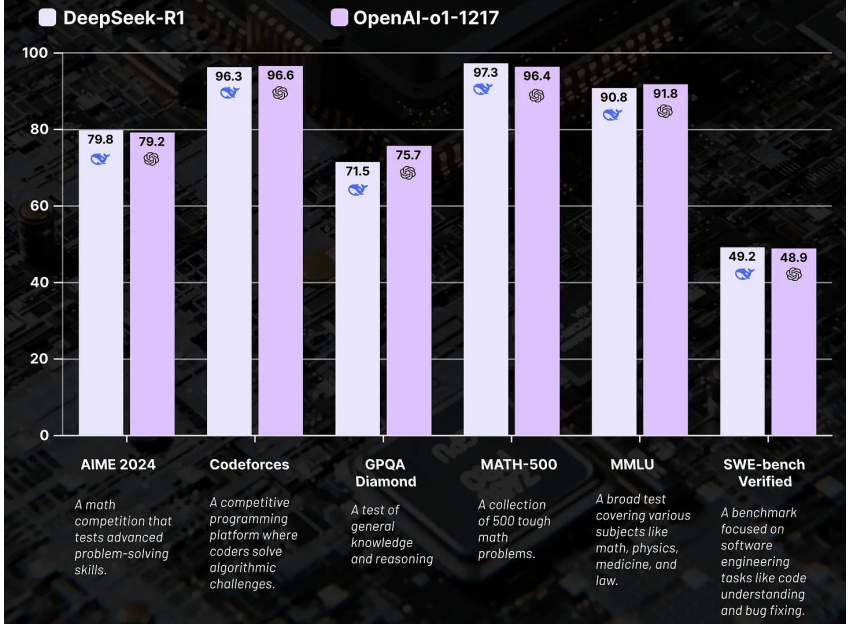
Created by genuine impact

DeepSeek-R1:

- *A reasoning model developed by the Chinese artificial intelligence company DeepSeek, designed to handle complex tasks in mathematics, programming, and natural language reasoning.*
- *Release Date: January 20, 2025.*

OpenAI-o1-1217:

- *A version of OpenAI's o1 model, focusing on enhancing AI's reasoning capabilities.*
- *Release Date: The preview version of o1 was released on September 12, 2024, with the official version launched on December 5, 2024.*



Source: DeepSeek Official Website

More charts: genuineimpact.substack.com

Model Serving: Paid Options for LLM Integration

AI Tools for the Waldo Laboratory

NumPy PyTorch scikit learn

TensorFlow pillow CO NVIDIA

What is Google Colab?

NVIDIA Tesla T4

Google Colab
https://colab.research.google.com ›

GPU Architecture

Tesla T4 is a GPU card based on the acceleration.

NVIDIA

WHERE'S WALDO?®

The Animated Series

WHERE'S WALDO?®

16-MONTH
2021
CALENDAR

Waldo, along with his trusty friend **Wenda**, travels to exciting places, always having a new adventure in every corner of the world. But wherever he goes, his sneaky rival, **Odlaw**, is never far behind, causing chaos and trying to steal the spotlight.

Luckily, Waldo can always count on the wise **Wizard Whitebeard**, who helps him out of sticky situations with his magical powers.

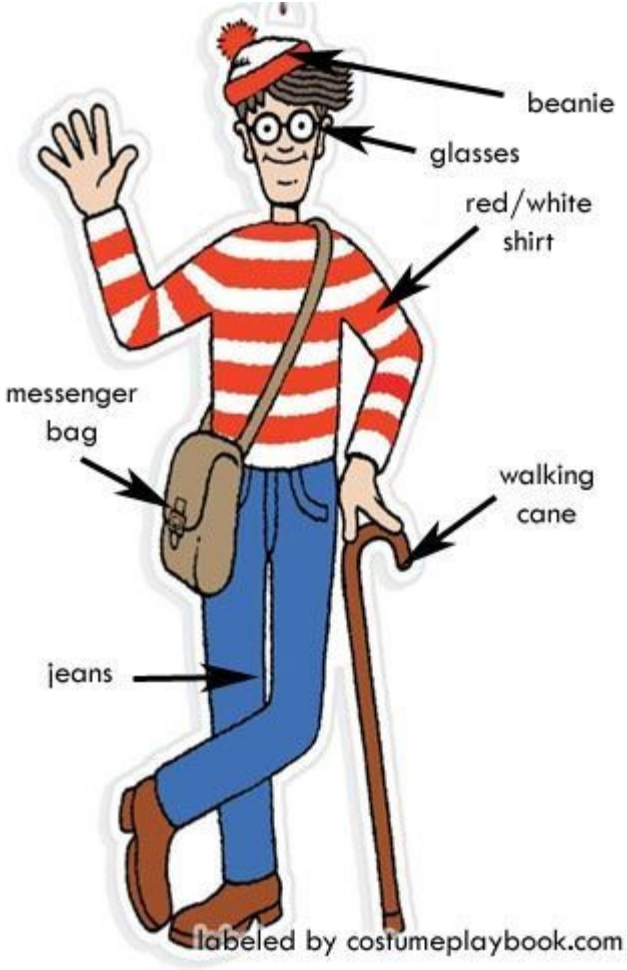
Together, they make an unbeatable team, solving mysteries, facing challenges, and—of course—having fun along the way!

Waldo (Wally)

Bio: Waldo is a brave and adventurous traveler, always sporting his red-and-white striped shirt, beanie, and round glasses. He's the kind of person who loves exploring new places—whether it's busy cities, snowy mountains, or sandy beaches—he's always on the move! But there's one thing about Waldo: he's really, really hard to find. He loves to blend into big crowds, and that's why people have been searching for him for years!

Fun Fact: Waldo never stops moving! He's always in search of his next big adventure, and you might just bump into him on your next journey... if you're lucky enough to spot him!

Helpful Tips



Wenda

Bio: Wenda is Waldo's best friend and fellow explorer! She shares Waldo's love of travel and has the same red-and-white striped shirt and glasses. But Wenda isn’t just about looking stylish—she’s clever, determined, and always ready to help out. Whether she’s solving a tricky puzzle or helping Waldo escape from tricky situations, Wenda’s the one you can count on to get things done.

Fun Fact: Wenda is always prepared for anything—she’s got a bag full of gadgets and helpful tools to help her and Waldo on their adventures!

Odlaw

Bio: Odlaw is Waldo's pesky rival. He wears a yellow-and-black striped outfit and is always trying to outdo Waldo, usually by causing some mischief. While Waldo is busy having fun and helping others, Odlaw is scheming to make things harder for him. He's got a mischievous mustache and loves playing pranks on Waldo and his friends!

Fun Fact: Odlaw's plan is always to get the spotlight away from Waldo—whether it's by hiding better than him or creating trouble in the background. But no matter what he tries, Waldo always finds a way to win!

Wizard Whitebeard

Bio: Wizard Whitebeard is the magical, wise old man who always knows just what to do when Waldo and his friends need help. With his long white beard and glowing staff, Wizard Whitebeard has lived through many adventures. He uses his magic to guide Waldo and Wenda whenever they get stuck or need a little extra wisdom.

Fun Fact: When Wizard Whitebeard waves his staff, incredible things happen! His spells can make people float, change colors, and even make objects appear out of thin air!


WHERE'S WALDO

Can you find Waldo in this 360 image - an illustration that went viral?

https://blog.kuula.co/where-is-waldo-360-illustration

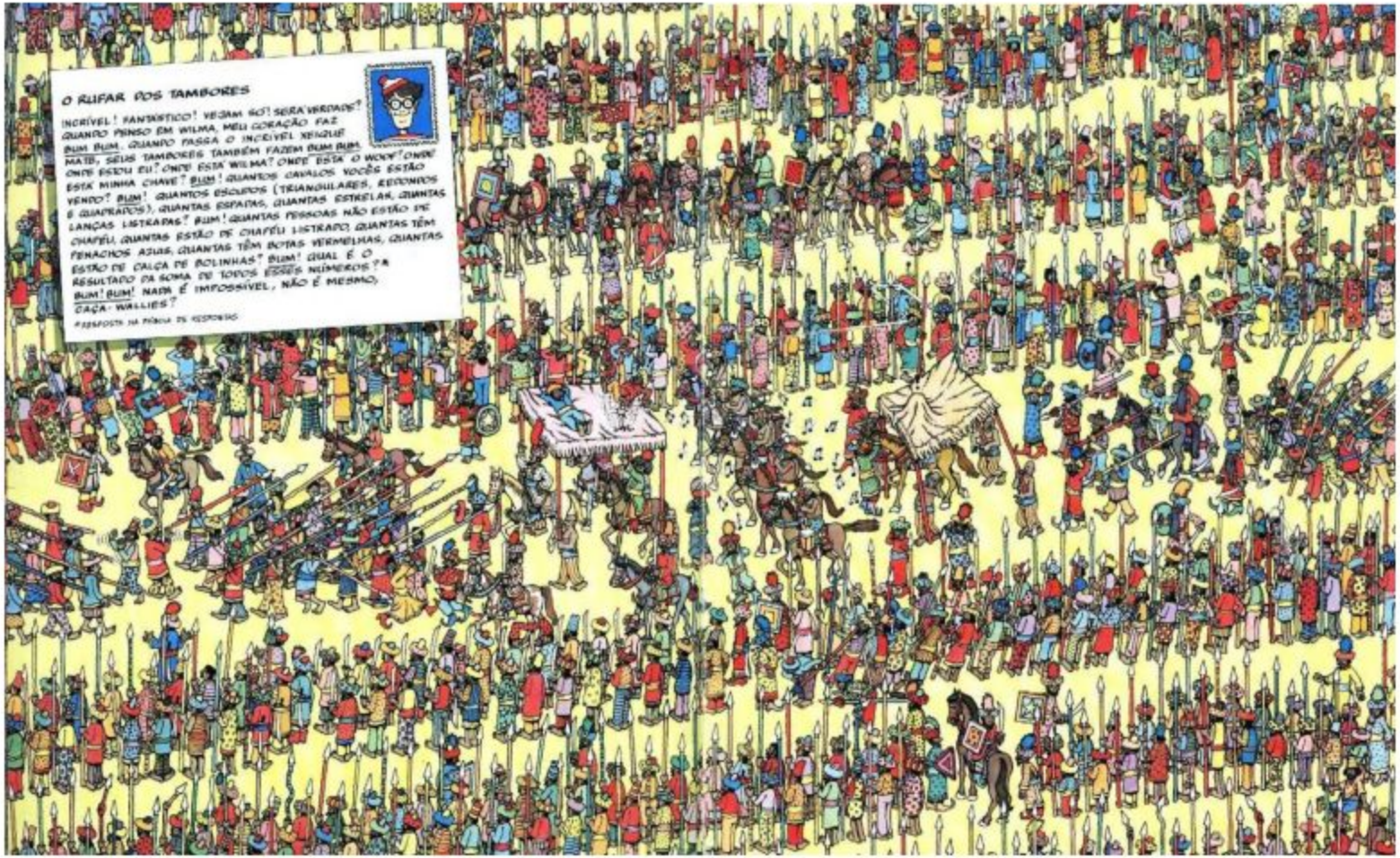

360-DRAWING.COM
LEON ROUS

Challenge: Fine-Tuning to Find the Winner

EASY_007.jpg - Found 0 Waldo Faces 😎

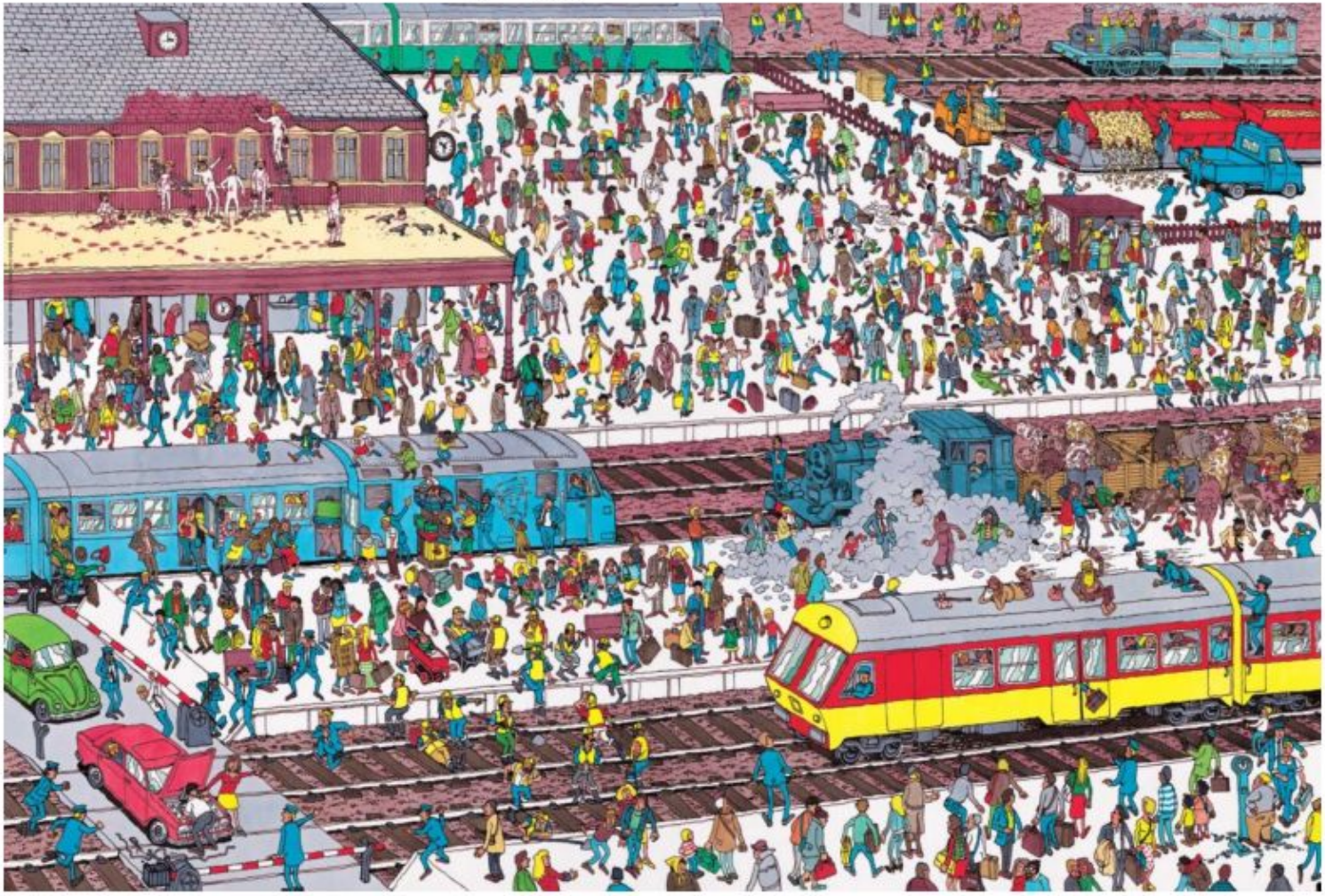
EASY_002.jpg - Found 1 Waldo Face 😎

HARD_002.jpg - Found 0 Waldo Faces 😎



HARD_001.jpg - Found 0 Waldo Faces 😎

HARD_009.jpg - Found 0 Waldo Faces 😎



EASY_001.jpg - Found 0 Waldo Faces 😎

🎉🎉🎉 Total Waldo Faces Found: 1 😎
The winner will be the student who finds the most Waldo faces! 🏆

Waldo AI Lab

https://github.com/kaopanboonyuen/where-is-waldo

Table of contents

🚀 Waldo AI: The Ultimate Hide-and-Seek Showdown
📌 Step 1: Install & Import Required Libraries
📌 Step 2: Download and Prepare the Dataset
📌 Step 3: Organize Data in YOLOv12 Format
📌 Step 4: Preview Dataset with Random Image Samples
📌 Step 5: Configure the Dataset YAML File
📌 Step 6: Load the YOLOv12 Model
📌 Step 7: Train the Model
📌 Step 8: Evaluate the Model
📌 Step 9: Inference and Error Analysis
📌 Step 10: Confusion Matrix and Precision-Recall-F1 Analysis
📌 Step 11: Summary of Waldo Faces Detected
DONE
+ Section

🚀 Waldo AI: The Ultimate Hide-and-Seek Showdown

By Kao Panboonyuen

This Colab notebook will guide you through:

- ✅ Preparing and loading the dataset
- ✅ Exploring images and labels
- ✅ Performing exploratory data analysis (EDA)
- ✅ Training AI (small model) for segmentation
- ✅ Evaluating performance with accuracy, confusion matrix, precision, recall, F1-score
- ✅ Performing inference and error analysis

📌 Step 1: Install & Import Required Libraries

In this step, we'll install the necessary libraries for training AI. You'll need to install the AI package, and PyTorch to enable GPU acceleration.

```
[ ]
!pip install torch==2.0.0+cu117 torchvision torchaudio --index-url https://download.pytorch.org/whl/cu117
!pip install ultralytics
```

```
[ ]
!pip install --upgrade torchvision
```

```
[ ]
import os
import shutil
import zipfile
import matplotlib.pyplot as plt
import cv2
from sklearn.metrics import precision_recall_fscore_support, confusion_matrix
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")

import locale
locale.getpreferredencoding = lambda: "UTF-8"
```

```
[ ]
# Check if CUDA (GPU support) is available for PyTorch and print the result (True/False).
print(torch.cuda.is_available())

# Print the number of CUDA-compatible devices (GPUs) available on the system.
print(torch.cuda.device_count())

# Print the name of the first CUDA-compatible device (GPU) available (index 0).
```

📌 Step 2: Download and Prepare the Dataset

Next, we will download the dataset from the link provided and extract it.

```
[ ]
# # # Optional Dataset (Type: A)

# # Define the global variables for the developer name and repository name
# DEV_NAME = 'kaopanboonyuen'
# REPO_NAME = 'where-is-waldo'

# # Use f-string to create the dynamic URL for the dataset
# url = f'https://github.com/{DEV_NAME}/{REPO_NAME}/raw/main/dataset/waldo-dataset-a.zip'

# # Download the zip file using wget
# # !wget https://github.com/{DEV_NAME}/{REPO_NAME}/raw/main/dataset/waldo-dataset-a.zip -O waldo-dataset-a.zip
# !wget {url} -O waldo-dataset-a.zip

# # Unzip the downloaded file
# # !unzip /content/waldo-dataset-a.zip >> logs.log
# !unzip /content/waldo-dataset-a.zip >> logs.log
```

```
[ ]
# Define the global variable for the repository name
DEV_NAME = # Write your code here
REPO_NAME = # Write your code here

# Use f-string to create the dynamic URLs
url_part1 = f'https://github.com/{DEV_NAME}/{REPO_NAME}/raw/main/dataset/waldo-dataset-b.zip.part1'
url_part2 = f'https://github.com/{DEV_NAME}/{REPO_NAME}/raw/main/dataset/waldo-dataset-b.zip.part2'
url_part3 = f'https://github.com/{DEV_NAME}/{REPO_NAME}/raw/main/dataset/waldo-dataset-b.zip.part3'
url_part4 = f'https://github.com/{DEV_NAME}/{REPO_NAME}/raw/main/dataset/waldo-dataset-b.zip.part4'
url_part5 = f'https://github.com/{DEV_NAME}/{REPO_NAME}/raw/main/dataset/waldo-dataset-b.zip.part5'

# Use wget with the generated URLs for each part
!wget {url_part1} -O waldo-dataset-b.zip.part1
!wget {url_part2} -O waldo-dataset-b.zip.part2
!wget {url_part3} -O waldo-dataset-b.zip.part3
!wget {url_part4} -O waldo-dataset-b.zip.part4
!wget {url_part5} -O waldo-dataset-b.zip.part5
```

📌 Step 3: Organize Data in YOLOv12 Format

YOLOv12 requires the data to be organized in a specific format. We'll make sure that the labels and images are correctly set up.

```
[ ]
os.makedirs('/content/dataset', exist_ok=True)
os.makedirs('/content/dataset/train', exist_ok=True)
os.makedirs('/content/dataset/valid', exist_ok=True)

shutil.move('/content/waldo-dataset-b/train/images', '/content/dataset/train/images')
shutil.move('/content/waldo-dataset-b/train/labels', '/content/dataset/train/labels')
shutil.move('/content/waldo-dataset-b/valid/images', '/content/dataset/valid/images')
shutil.move('/content/waldo-dataset-b/valid/labels', '/content/dataset/valid/labels')
```

📌 Step 4: Preview Dataset with Random Image Samples

```
[ ]
import random
import os
import cv2
import numpy as np
from matplotlib import pyplot as plt

def show_random_label_overlay(train_images_dir, train_labels_dir, num_images=4):
    """
    Function to show random images with their labels overlaid in a grid (1 row, 4 columns).

    :param train_images_dir: Path to the directory containing the images
    :param train_labels_dir: Path to the directory containing the labels
    :param num_images: Number of random images to show in the grid (default is 4)
    """
    fig, axs = plt.subplots(1, num_images, figsize=(15, 5))

    for i in range(num_images):
        # Randomly select an image and its corresponding label
        image_file = random.choice(os.listdir(train_images_dir))
        label_file = image_file.replace('.jpg', '.txt')  # Assuming the labels are .txt files with the same name as the image

```

📌 Step 5: Configure the Dataset YAML File

For YOLOv12 to know how to process the dataset, we need to create a `data.yaml` configuration file. This file specifies where the dataset is located and defines the class names.

```
[ ]
# ## FOR WALDO-A DATASET

# data_yaml = """
# train: /content/dataset/train/images
# val: /content/dataset/valid/images

# nc: 4
# names: ['odlaw', 'wally', 'wenda', 'wizard_whitebeard']
# """
# with open("/content/dataset/data.yaml", "w") as f:
#     f.write(data_yaml)
```

```
[ ]
## FOR WALDO-B DATASET

data_yaml = """
train: # Write your path here
val: # Write your path here

nc: # Write your code here
names: # Write your code here
"""
with open("/content/dataset/data.yaml", "w") as f:
    f.write(data_yaml)
```

📌 Step 6: Load the YOLOv12 Model

Now, load the YOLOv12 small model (YOLOv12n) from the ultralytics package.

```
[ ]  1 # Download the pretrained YOLOv12 model (if available)
     2 from ultralytics import YOLO
```

```
[ ]  1 # Load the pretrained model (adjust the URL or model name if necessary)
     2 model = YOLO("Write your model here")  # Replace this with the correct path or model
```

📌 Step 7: Train the Model

Now that we have everything ready, we can start training the model. We will fine-tune the pre-trained YOLOv12 small model on the pothole dataset.

```
[ ]  1 # Write your trainer code here
```

📌 Step 8: Evaluate the Model

Once training is complete, evaluate the model's performance on the validation dataset.

```
[ ]  1 results = # Write your code here  # Evaluate on validation set
```

📌 Step 9: Inference and Error Analysis

After evaluation, it's time to run inference on some images and analyze the errors. Let's run inference on a few images and visualize the predictions.

```
[ ]  1 # Perform inference on validation images
     2 results = # Write your code here
```

```
[ ]  1 # Load the trained model
     2 model = YOLO('# Write your best of AI model path here')  # Replace with the correct path to your trained model
     3
     4 # Perform inference on the image
     5 img_path = 'Write your inference path here'
     6 results = model(img_path)
     7
     8 # Access the first result from the list (since results is a list)
     9 result = results[0]
    10
    11 # Render the results (bounding boxes, labels, and confidence scores)
    12 # Write your code here  # This will display the image with bounding boxes and labels overlaid
```

📌 Step 10: Confusion Matrix and Precision-Recall-F1 Analysis

Evaluate the predictions using metrics like Precision, Recall, and F1-Score:

```
[ ]
from IPython.display import Image, display

# Path to the confusion matrix image
confusion_matrix_path = 'Write your CF matrix image here'

# Display the confusion matrix image
display(Image(filename=confusion_matrix_path))
```

📌 Step 11: Summary of Waldo Faces Detected

🎉 **Yeah, it's the final step!** We've been on an exciting journey, and now it's time to find out who found the most Waldo faces! 🕵️

```
[ ]
DEV_NAME = # Write your code here
REPO_NAME = # Write your code here

# Use f-string to create the dynamic URL
url = f'https://github.com/{DEV_NAME}/{REPO_NAME}/raw/main/dataset/waldo-dataset-test.zip'

# Now use wget with the generated URL
!wget {url} -O waldo-dataset-test.zip
```

```
6
7 # Load YOLO model with the best weights
8 model = YOLO('Write your best of AI model here')
9
10 # Path to the test dataset
11 test_images_dir = Path('Write your path here')
12
```

```
56 # Summary after processing all images
57 print(f"\n🎉🎉🎉 Total Waldo Faces Found: {total_waldo_faces} 😎")
58 print("The winner will be the student who finds the most Waldo faces! 🏆")
```

Optimizing Model Training

Learning Rate: The Step Size in Training

Objective: Learn how the learning rate affects model training and how to adjust it for optimal performance.

- **What is it?**
  - The learning rate determines how much the model's weights are updated with each step during training.
  - **High learning rate**: Faster but risks overshooting optimal solutions.
  - **Low learning rate**: Slower, more precise updates but may get stuck in local minima.
- **How to Adjust**:
  - Start with a small learning rate (e.g., 0.001) and experiment with scheduling to reduce it over time.
  - Use **learning rate schedulers** like **StepLR** or **ReduceLROnPlateau** for more efficient training.
- **Practical Example**:

python Copy

```
model.train(data='/content/dataset/data.yaml',
            epochs=20,
            imgsz=640,
            batch=16,
            device=0,
            lr0=0.001)  # Set initial learning rate
```

Momentum: Smoothing Out Training

Objective: Understand how momentum accelerates training by smoothing out updates.

- **What is it?**
  - Momentum helps the optimizer use past gradients to accelerate updates, especially in the case of noisy or fluctuating gradients.
  - **High momentum** (e.g., 0.9) makes training faster, but too high can lead to instability.
- **How to Adjust**:
  - Start with a default momentum of 0.9 and adjust if the model seems to oscillate or converge too slowly.
- **Practical Example**:

```python
model.train(data='/content/dataset/data.yaml',
            epochs=20,
            imgsz=640,
            batch=16,
            device=0,
            momentum=0.9)  # Set momentum
```

Focal Loss: Addressing Class Imbalance

Objective: Use focal loss to make the model focus on harder-to-classify examples.

- **What is it?**
  - Focal loss modifies the standard cross-entropy loss by down-weighting easy examples and focusing more on hard ones.
  - Especially useful for tasks with **class imbalance**, like detecting rare objects (e.g., Waldo).
- **How to Adjust**:
  - Enable focal loss by setting `focal_loss=True` in the training parameters.
  - **Alpha and gamma parameters** can be adjusted to control the focus on hard examples.
- **Practical Example**:

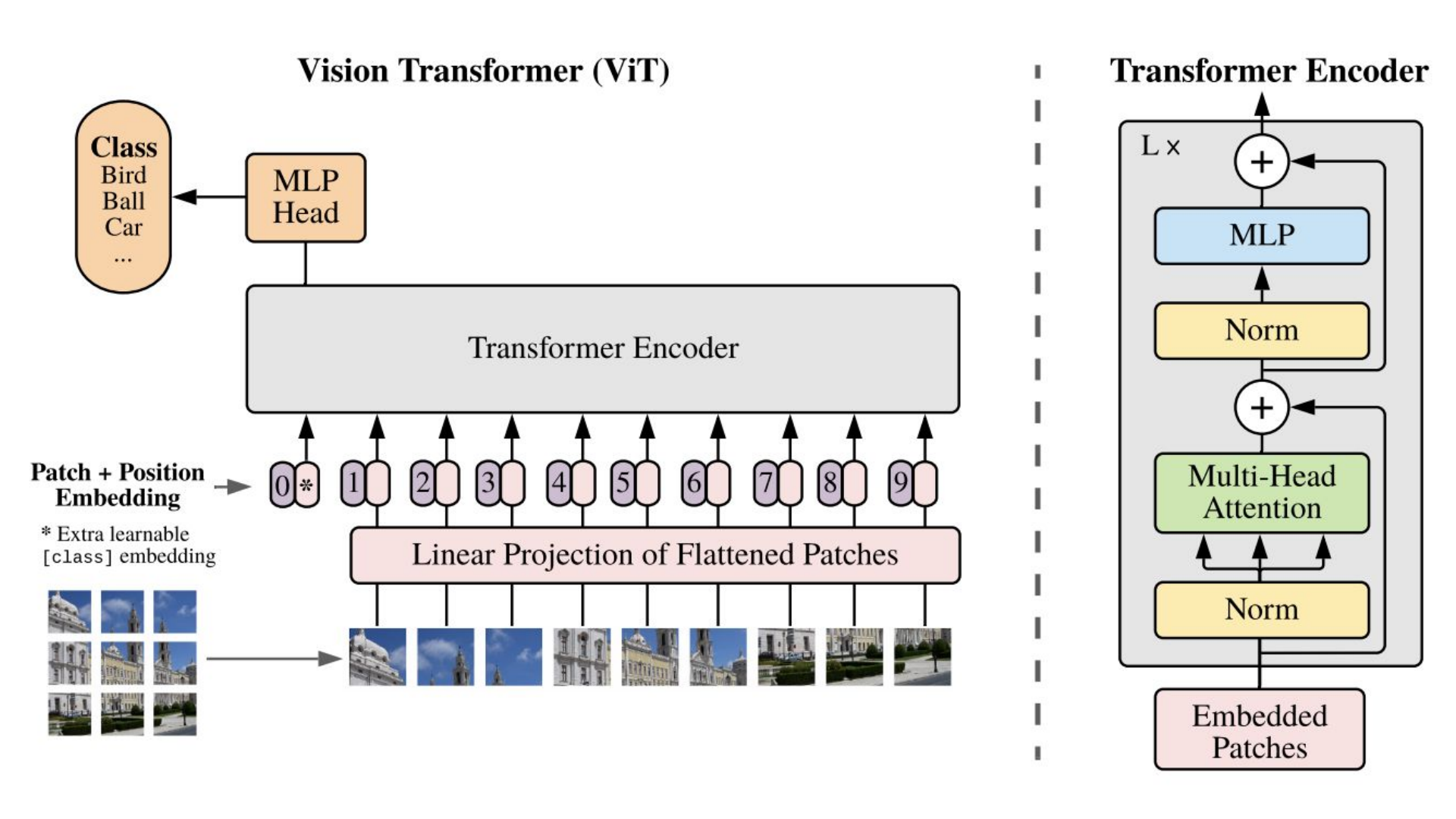
```python
model.train(data='/content/dataset/data.yaml',
            epochs=20,
            imgsz=640,
            batch=16,
            device=0,
            focal_loss=True)  # Enable focal loss for imbalance handling
```

Pretrained Models: Leverage Transfer Learning (ResNet, Vision Transformer)

Objective: Improve model performance and reduce training time by using pretrained backbones like **ResNet** and **Vision Transformers** in YOLO.

- **What are Pretrained Models?**
  - Pretrained models are models that have been trained on large datasets (e.g., **ImageNet**) and can be used as feature extractors for new tasks.
  - These models have already learned general features like edges, textures, and patterns, which can significantly improve performance when fine-tuning on smaller datasets.
- **Benefits of Using Pretrained Models:**
  - **Faster convergence**: Pretrained models already have useful feature representations.
  - **Improved accuracy**: Leveraging knowledge from large datasets like ImageNet boosts performance, especially on limited data.
  - **Reduced training time**: Less data and epochs are needed to achieve good results.

- **Pretrained Backbones in YOLO**:
  - **ResNet**: Known for its deep architecture and residual learning, ideal for feature extraction.
  - **Vision Transformer (ViT)**: A newer architecture that leverages self-attention mechanisms to learn long-range dependencies in images.
- **How to Implement**:
  - You can use pretrained versions of ResNet or Vision Transformer in YOLO by specifying the backbone parameter.


Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP
Head
Transformer Encoder
Patch + Position
Embedding
* Extra learnable
[class] embedding
0
*
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches
Transformer Encoder
L x
MLP
Norm
Multi-Head
Attention
Norm
Embedded
Patches

- **Practical Example**:

```python
model.train(data='/content/dataset/data.yaml',
            epochs=20,
            imgsz=640,
            batch=16,
            device=0,
            backbone='resnet50')  # Using a pretrained ResNet50 backbone
```

Or, if using a Vision Transformer:

```python
model.train(data='/content/dataset/data.yaml',
            epochs=20,
            imgsz=640,
            batch=16,
            device=0,
            backbone='vit')  # Using a pretrained Vision Transformer backbone
```

- **Why Use These Models?**
  - **ResNet**: Excellent for tasks requiring deep feature extraction and robust performance.
  - **Vision Transformer**: Effective for capturing global context and spatial relationships in images.

10 Practical Fine-Tuning Techniques for Improved Accuracy

1. Freeze Initial Layers (Transfer Learning)

Objective: Freeze the first few layers of a pretrained model to retain general features learned from a larger dataset and prevent overfitting.

python Copy

```python
# Freeze the first 3 layers and fine-tune the remaining layers
model.train(data='/content/dataset/data.yaml',
            epochs=20,
            imgsz=640,
            batch=16,
            device=0,
            freeze=[0, 1, 2])  # Freeze layers 0, 1, and 2
```

- **Explanation**: Freezing early layers allows the model to retain general features like edges and textures while focusing on learning task-specific features in the later layers. This helps prevent overfitting, especially with limited data.

2. Learning Rate Scheduling

Objective: Use a learning rate scheduler to reduce the learning rate during training for smoother convergence.

python Copy

```python
# Use a learning rate scheduler (StepLR) to reduce the learning rate every 10 epochs
model.train(data='/content/dataset/data.yaml',
            epochs=20,
            imgsz=640,
            batch=16,
            device=0,
            lr_scheduler='step',  # Use StepLR
            lr0=0.001,            # Initial learning rate
            lrf=0.01,             # Learning rate final factor
            lrs=10)               # Reduce learning rate every 10 epochs
```

- **Explanation**: Learning rate scheduling helps the model converge more smoothly by decreasing the learning rate as it approaches optimal performance. This prevents overshooting the minimum of the loss function.

3. Data Augmentation

Objective: Augment the training data to introduce more variability and make the model more robust.

python Copy

```python
# Apply random flips, rotations, and color jitter for data augmentation
model.train(data='/content/dataset/data.yaml',
            epochs=20,
            imgsz=640,
            batch=16,
            device=0,
            augment=True,        # Enable augmentation
            flipud=0.5,         # Flip images vertically with 50% probability
            fliplr=0.5,         # Flip images horizontally with 50% probability
            hsv_h=0.1,          # Randomly adjust hue by up to 0.1
            hsv_s=0.1,          # Randomly adjust saturation by up to 0.1
            hsv_v=0.1)          # Randomly adjust brightness by up to 0.1
```

- **Explanation**: Data augmentation artificially increases the size of your training dataset by applying random transformations. This helps the model generalize better and reduces the likelihood of overfitting.

4. Using Focal Loss for Class Imbalance

Objective: Apply focal loss to address class imbalance by focusing on hard-to-classify examples.

python Copy

```python
# Enable focal loss to focus more on the "Waldo" class (if rare)
model.train(data='/content/dataset/data.yaml',
            epochs=20,
            imgsz=640,
            batch=16,
            device=0,
            focal_loss=True)  # Enable focal loss
```

- **Explanation**: Focal loss is particularly effective for tasks with class imbalance, like "Where's Waldo?". It reduces the loss for well-classified examples and focuses more on those that are difficult to classify.

5. Adjusting Batch Size

Objective: Experiment with smaller batch sizes for better model generalization.

python Copy

```python
# Use a smaller batch size (8 instead of 16) to improve model generalization
model.train(data='/content/dataset/data.yaml',
            epochs=20,
            imgsz=640,
            batch=8,            # Use a smaller batch size
            device=0)
```

- **Explanation**: A smaller batch size often leads to better generalization and allows the model to adjust more frequently to the data. It also helps with memory limitations on GPUs.

6. Weight Decay (L2 Regularization)

Objective: Use weight decay to regularize the model and reduce overfitting.

python Copy

```python
# Apply weight decay (L2 regularization) to reduce overfitting
model.train(data='/content/dataset/data.yaml',
            epochs=20,
            imgsz=640,
            batch=16,
            device=0,
            weight_decay=0.0005)  # Apply regularization
```

- **Explanation**: Weight decay penalizes large weights, encouraging the model to learn smaller, more generalized weights. This helps prevent overfitting, especially with smaller datasets.

7. Increase Epochs for More Training

Objective: Train for more epochs to allow the model to better learn from the data.

python Copy

```python
# Increase training epochs to 40 for more training time
model.train(data='/content/dataset/data.yaml',
            epochs=40,            # Increased epochs
            imgsz=640,
            batch=16,
            device=0)
```

- **Explanation**: Training for more epochs gives the model more time to learn and fine-tune its parameters, especially if the data is complex or noisy.

8. Use a Pretrained Backbone

Objective: Start with a pretrained model backbone like ResNet or YOLO for feature extraction.

python Copy

```python
# Use a pretrained ResNet backbone for feature extraction
model.train(data='/content/dataset/data.yaml',
            epochs=20,
            imgsz=640,
            batch=16,
            device=0,
            backbone='resnet50')  # Use pretrained ResNet50 backbone
```

- **Explanation**: Using a pretrained backbone leverages knowledge from a large dataset, such as ImageNet, and can significantly improve model performance, especially when you have limited labeled data.

9. Early Stopping Based on Validation Loss

Objective: Stop training early if the validation loss stops improving to avoid overfitting.

python Copy

```python
# Use early stopping to halt training when validation loss stops improving
model.train(data='/content/dataset/data.yaml',
            epochs=20,
            imgsz=640,
            batch=16,
            device=0,
            early_stop=True)  # Enable early stopping
```

- **Explanation**: Early stopping monitors the validation loss during training. If it doesn't improve for a set number of epochs, the training stops, helping to prevent overfitting and saving time.

10. Use Gradient Clipping

Objective: Apply gradient clipping to avoid exploding gradients, especially when training deep networks.

python Copy

```python
# Enable gradient clipping to prevent exploding gradients
model.train(data='/content/dataset/data.yaml',
            epochs=20,
            imgsz=640,
            batch=16,
            device=0,
            clip_grad=5.0)  # Clip gradients to 5.0
```

- **Explanation**: Gradient clipping limits the magnitude of gradients during backpropagation. This can prevent the model from making unstable updates and is particularly useful when using large batch sizes or deep architectures.

Summary

- **Fine-tuning** a model requires experimentation with various techniques like freezing layers, adjusting learning rates, augmenting data, and addressing class imbalances.
- Use **practical techniques** like focal loss, weight decay, and early stopping to get better accuracy on complex tasks such as "Where’s Waldo?".
- Regularly **monitor your model** during training and adjust parameters for optimal performance.

Find Waldo, Find Your AI Path!

Congrats, AI Explorer! 🎉

- You've learned how to build an AI to find **Waldo**—now it's your turn to apply these skills in the real world! 🕵️

Hope You Had Fun! 🎨

- AI is all about **experimenting** and **learning**—and finding Waldo is just the beginning! 🌟

What's Next?

- Keep exploring, keep creating, and maybe one day you'll be the next **AI researcher**! 🚀

Good Luck on Your AI Adventure! ✨