|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА - Российский технологический университет»**  **РТУ МИРЭА** |

**Институт кибербезопасности и цифровых технологий (ИКБ)**

**Кафедра КБ-9 «Предметно-ориентированные информационные системы»**

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ**

по дисциплине «Интеллектуальные системы и технологии»

**Практическое занятие № 3**

|  |  |  |
| --- | --- | --- |
| Студенты группы | *БСМО-32-25, Гвасалия Г.В., Гирко А.Е, Кривошеев Н.А.* |  |
| Преподаватель | *Куренных А. Е.* |  |
| Отчет представлен | «\_\_\_»\_\_\_\_\_\_\_\_2025г. |  |

СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 1](#_Toc432079123)

[Цель и постановка задачи 2](#_Toc670985748)

[Данные для обучающей выборки 3](#_Toc997545208)

[Установка зависимостей и настройка конфигурации 3](#_Toc214745469)

[Предобработка данных 5](#_Toc169753852)

[Взаимодействие с моделями 5](#_Toc1753232457)

[ЗАКЛЮЧЕНИЕ 6](#_Toc1089042303)

[ЛИСТИНГ КОДА 7](#_Toc1592993209)

Цель и постановка задачи

Необходимо решить задачу определения: относится ли самолёт на фотографии к классу военных или гражданских. Для решения нужно использовать провайдеров LLM на Hugging Face, сравнить метрики двух выбранных моделей, собрать выборку (~до 30 изображений) с нетривиальными случаями. Допускается выбор иных объектов вместо самолётов (в данной работе остаёмся на самолётах).

* 1. Данные для обучающей выборки

В качестве выборки были взяты самолеты из разных источников и разной стилистик и фотографии, которые были заранее помечены в csv файле.

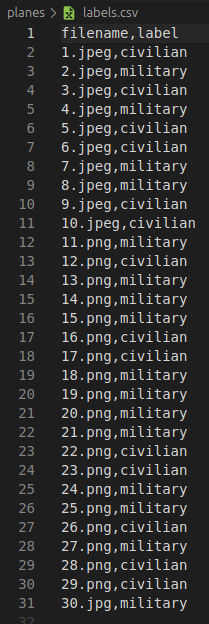
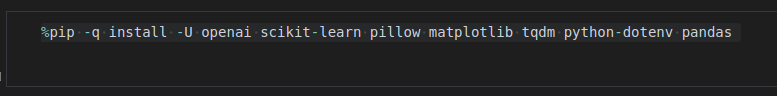


Рисунок 1 — Разметка текущей выборки

Установка зависимостей и настройка конфигурации

Для работы программы установим зависимости, которые позволят взаимодействовать с выборкой и с API Hugging Face.

Рисунок 2 — Установка зависимостей

Далее импортируем все установленные зависимости для дальнейшего использования.

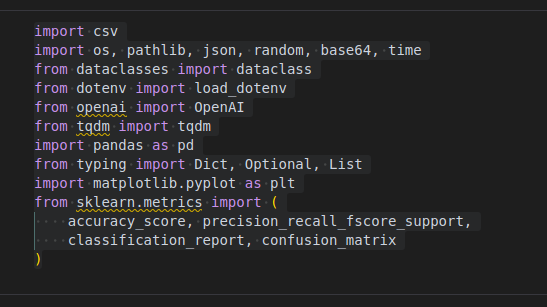
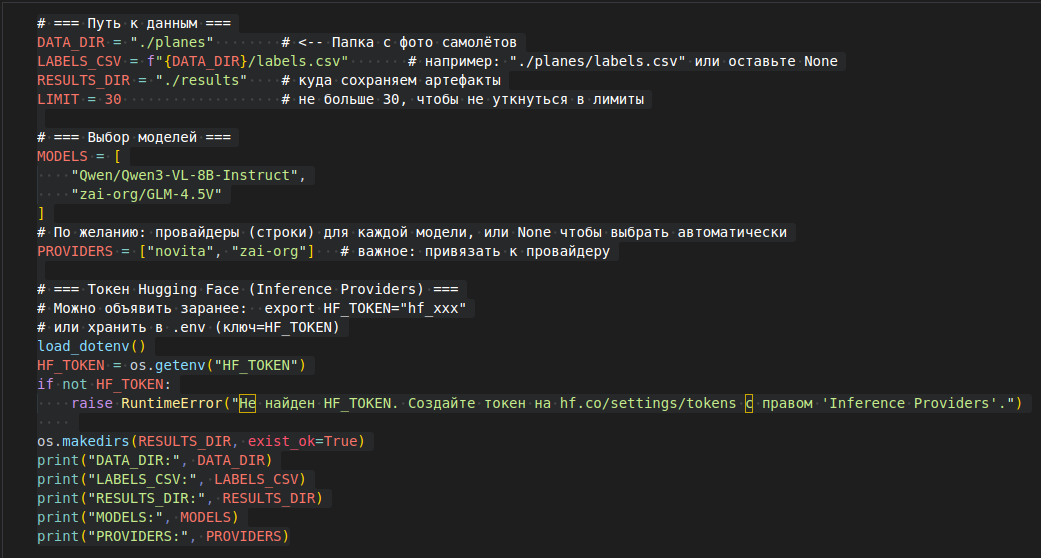


Рисунок 3 — Импорт зависимостей

Так же установим все конфигурации для работы с API Hugging Face и датасетом.

Рисунок 4 — Установка конфигурации

Предобработка данных

Для начала для работы с нашим датасетом, считаем его с csv файла и сделаем проверку на нескольких данных.

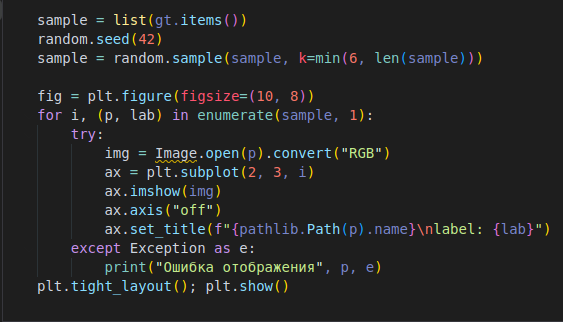
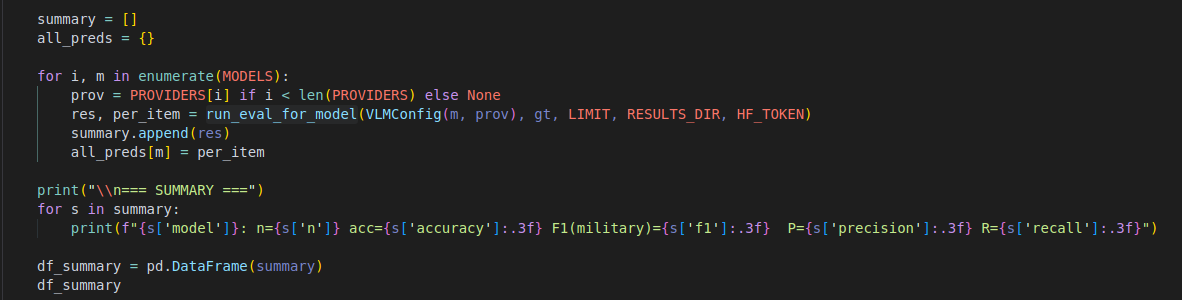


Рисунок 5 — Считывание данных и вывод несколько примеров

Для наглядной работы моделей сделаем вывод метрик и результат работы двух нейронных сетей в виде матрицы ошибок.

* 1. Взаимодействие с моделями

Для взаимодействия с моделями зарегистрируемся в Hugging face и создадим ключ, по которому будем обращаться к моделям для обработки фотографий. Для дополнительной защиты сделаем своего рода кэш данных, который не позволит привысить количество запросов на сайте, запустим модели.

Рисунок 6 — Запуск моделей и вывод результата

ЗАКЛЮЧЕНИЕ

В работе решена задача бинарной классификации самолётов по фотографии (военный/гражданский) с использованием провайдеров Hugging Face. Построен воспроизводимый конвейер: подготовка небольшой, но «неочевидной» выборки (~до 30 изображений), инференс двумя разными подходами (VLM-чат LLaVA и caption-модель BLIP), нормализация ответа модели до метки класса и расчёт метрик (accuracy, precision, recall, F1) с анализом матриц ошибок.

ЛИСТИНГ КОДА

%pip -q install -U openai scikit-learn pillow matplotlib tqdm python-dotenv pandas

import csv

import os, pathlib, json, random, base64, time

from dataclasses import dataclass

from dotenv import load\_dotenv

from openai import OpenAI

from tqdm import tqdm

import pandas as pd

from typing import Dict, Optional, List

import matplotlib.pyplot as plt

from sklearn.metrics import (

accuracy\_score, precision\_recall\_fscore\_support,

classification\_report, confusion\_matrix

)

# === Путь к данным ===

DATA\_DIR = "./planes" # <-- Папка с фото самолётов

LABELS\_CSV = f"{DATA\_DIR}/labels.csv" # например: "./planes/labels.csv" или оставьте None

RESULTS\_DIR = "./results" # куда сохраняем артефакты

LIMIT = 30 # не больше 30, чтобы не уткнуться в лимиты

# === Выбор моделей ===

MODELS = [

"Qwen/Qwen3-VL-8B-Instruct",

"zai-org/GLM-4.5V"

]

# По желанию: провайдеры (строки) для каждой модели, или None чтобы выбрать автоматически

PROVIDERS = ["novita", "zai-org"] # важное: привязать к провайдеру

# === Токен Hugging Face (Inference Providers) ===

# Можно объявить заранее: export HF\_TOKEN="hf\_xxx"

# или хранить в .env (ключ=HF\_TOKEN)

load\_dotenv()

HF\_TOKEN = os.getenv("HF\_TOKEN")

if not HF\_TOKEN:

raise RuntimeError("Не найден HF\_TOKEN. Создайте токен на hf.co/settings/tokens с правом 'Inference Providers'.")

os.makedirs(RESULTS\_DIR, exist\_ok=True)

print("DATA\_DIR:", DATA\_DIR)

print("LABELS\_CSV:", LABELS\_CSV)

print("RESULTS\_DIR:", RESULTS\_DIR)

print("MODELS:", MODELS)

print("PROVIDERS:", PROVIDERS)

LABELS = ("civilian", "military")

def collect\_images(data\_dir: str) -> List[str]:

exts = {".jpg", ".jpeg", ".png", ".webp"}

files = []

for p in pathlib.Path(data\_dir).glob("\*"):

if p.is\_file() and p.suffix.lower() in exts:

files.append(str(p))

return sorted(files)

def read\_labels\_csv(csv\_path: str) -> Dict[str, str]:

df = pd.read\_csv(csv\_path)

labels = {}

base = pathlib.Path(csv\_path).parent

for \_, row in df.iterrows():

fn = str(base / str(row["filename"]))

labels[fn] = str(row["label"]).strip().lower()

return labels

def normalize\_label(text: str) -> Optional[str]:

t = str(text).strip().lower()

if "military" in t:

return "military"

if "civil" in t:

return "civilian"

if t in {"military", "civilian"}:

return t

return None

def infer\_labels\_from\_filenames(files: List[str]) -> Dict[str, str]:

labels = {}

for p in files:

name = pathlib.Path(p).name.lower()

if any(k in name for k in ["military"]):

labels[p] = "military"

elif any(k in name for k in ["civil"]):

labels[p] = "civilian"

return labels

# Собираем файлы и метки

files = collect\_images(DATA\_DIR)

if not files:

raise RuntimeError("В папке с данными не найдено изображений (.jpg/.jpeg/.png/.webp).")

if LABELS\_CSV:

labels\_dict = read\_labels\_csv(LABELS\_CSV)

else:

labels\_dict = infer\_labels\_from\_filenames(files)

# Оставляем только размеченные

gt = {}

for f in files:

if f in labels\_dict:

lab = normalize\_label(labels\_dict[f])

if lab in LABELS:

gt[f] = lab

if not gt:

raise RuntimeError("Не найдено размеченных изображений. Используйте labels.csv или именование файлов.")

print(f"Всего изображений: {len(files)} | Размеченных: {len(gt)}")

# Вспомогательная таблица

df = pd.DataFrame([{"filename": pathlib.Path(k).name, "label": v} for k, v in gt.items()])

df.head()

sample = list(gt.items())

random.seed(42)

sample = random.sample(sample, k=min(6, len(sample)))

fig = plt.figure(figsize=(10, 8))

for i, (p, lab) in enumerate(sample, 1):

try:

img = Image.open(p).convert("RGB")

ax = plt.subplot(2, 3, i)

ax.imshow(img)

ax.axis("off")

ax.set\_title(f"{pathlib.Path(p).name}\nlabel: {lab}")

except Exception as e:

print("Ошибка отображения", p, e)

plt.tight\_layout(); plt.show()

def encode\_image\_to\_data\_url(path: str) -> str:

ext = pathlib.Path(path).suffix.lower().lstrip(".")

if ext not in {"jpg", "jpeg", "png", "webp"}:

ext = "jpeg"

with open(path, "rb") as f:

b64 = base64.b64encode(f.read()).decode("utf-8")

return f"data:image/{ext};base64,{b64}"

@dataclass

class VLMConfig:

name: str

provider: Optional[str] = None # например: "together", "sambanova", ...

class HfVlmClassifier:

def \_\_init\_\_(self, api\_key: str, model\_id: str, provider: Optional[str] = None, temperature: float = 0.0):

self.client = OpenAI(base\_url="https://router.huggingface.co/v1", api\_key=api\_key)

self.base\_model = model\_id # <- сохраняем базовый id

self.provider = provider # <- сохраняем провайдера (вот этой строки не хватало)

self.model = f"{model\_id}:{provider}" if provider else model\_id

self.temperature = temperature

self.response\_format = {

"type": "json\_schema",

"json\_schema": {

"name": "plane\_type",

"strict": True,

"schema": {

"type": "object",

"properties": {

"label": {"type": "string", "enum": ["civilian", "military"]},

"confidence": {"type": "number", "minimum": 0, "maximum": 1},

"rationale": {"type": "string"}

},

"required": ["label", "confidence"]

}

}

}

def classify(self, image\_path: str, retry: int = 2) -> Dict:

img\_data = encode\_image\_to\_data\_url(image\_path)

prompt = (

"Classify the airplane in this photo strictly as 'civilian' or 'military'. "

"Consider cues like livery, windows, weapon pylons, refueling booms, serial numbers, etc. "

"Output JSON with fields: label, confidence (0..1), rationale (short)."

)

last\_err = None

for \_ in range(retry + 1):

try:

completion = self.client.chat.completions.create(

model=f"{self.model}:{self.provider}" if self.provider else self.model,

messages=[{

"role": "user",

"content": [

{"type": "text", "text": prompt}, # твой текст-инструкцию

{"type": "image\_url", "image\_url": {"url": img\_data}}, # data:image/jpeg;base64,...

],

}],

response\_format={"type": "json\_object"},

max\_tokens=200,

temperature=0,

)

data = json.loads(completion.choices[0].message.content) # дальше как у тебя

msg = completion.choices[0].message

data = json.loads(msg.content) if isinstance(msg.content, str) else msg.content

lab = normalize\_label(data.get("label", ""))

if not lab:

raise ValueError(f"Bad label in response: {data}")

data["label"] = lab

data["raw"] = msg.content

return data

except Exception as e:

last\_err = e

time.sleep(1.2)

raise RuntimeError(f"Inference failed for {image\_path}: {last\_err}")

print("Классификатор готов.")

def evaluate(y\_true: List[str], y\_pred: List[str]):

acc = accuracy\_score(y\_true, y\_pred)

p, r, f1, \_ = precision\_recall\_fscore\_support(

y\_true, y\_pred, average="binary", pos\_label="military", zero\_division=0

)

return acc, p, r, f1

def plot\_confusion(y\_true, y\_pred, title, out\_png=None):

cm = confusion\_matrix(y\_true, y\_pred, labels=list(LABELS))

fig = plt.figure(figsize=(4,4))

ax = plt.gca()

im = ax.imshow(cm, interpolation='nearest')

ax.set\_xticks(range(len(LABELS))); ax.set\_yticks(range(len(LABELS)))

ax.set\_xticklabels(LABELS); ax.set\_yticklabels(LABELS)

ax.set\_xlabel("Predicted"); ax.set\_ylabel("True"); ax.set\_title(title)

for i in range(cm.shape[0]):

for j in range(cm.shape[1]):

ax.text(j, i, str(cm[i, j]), ha='center', va='center')

fig.colorbar(im)

plt.tight\_layout()

if out\_png:

plt.savefig(out\_png, dpi=160)

plt.show()

def run\_eval\_for\_model(model: VLMConfig, gt: Dict[str, str], limit: int, out\_dir: str, token: str):

os.makedirs(out\_dir, exist\_ok=True)

cache\_path = os.path.join(out\_dir, f"cache\_{model.name.replace('/','\_').replace(':','-')}.json")

cache = {}

if os.path.exists(cache\_path):

cache = json.load(open(cache\_path, "r", encoding="utf-8"))

items = list(gt.items())

if limit and len(items) > limit:

random.seed(42)

items = random.sample(items, k=limit)

clf = HfVlmClassifier(api\_key=token, model\_id=model.name, provider=model.provider)

y\_true, y\_pred = [], []

per\_item = {}

for img\_path, true\_lbl in tqdm(items, desc=f"Infer {model.name}"):

if img\_path in cache:

pred = cache[img\_path]

else:

pred = clf.classify(img\_path)

cache[img\_path] = pred

json.dump(cache, open(cache\_path, "w", encoding="utf-8"), ensure\_ascii=False, indent=2)

y\_true.append(true\_lbl)

y\_pred.append(pred.get("label", ""))

per\_item[img\_path] = {

"true": true\_lbl,

"pred": pred.get("label", ""),

"confidence": pred.get("confidence", None),

"rationale": pred.get("rationale", ""),

}

acc, p, r, f1 = evaluate(y\_true, y\_pred)

# Сохраняем csv с предсказаниями

pred\_csv = os.path.join(out\_dir, f"pred\_{model.name.replace('/','\_').replace(':','-')}.csv")

with open(pred\_csv, "w", encoding="utf-8", newline="") as f:

w = csv.writer(f)

w.writerow(["filename", "true", "pred", "confidence", "rationale"])

for k, v in per\_item.items():

w.writerow([pathlib.Path(k).name, v["true"], v["pred"], v["confidence"], (v["rationale"] or "").replace(',', ' ')])

# Матрица ошибок

cm\_png = os.path.join(out\_dir, f"cm\_{model.name.replace('/','\_').replace(':','-')}.png")

plot\_confusion(y\_true, y\_pred, f"{model.name}", out\_png=cm\_png)

# Текстовый отчёт

report\_txt = os.path.join(out\_dir, f"report\_{model.name.replace('/','\_').replace(':','-')}.txt")

with open(report\_txt, "w", encoding="utf-8") as f:

f.write(classification\_report(y\_true, y\_pred, labels=list(LABELS)))

f.write("\\n")

f.write(json.dumps({

"model": model.name,

"n": len(y\_true),

"accuracy": float(acc),

"precision\_military": float(p),

"recall\_military": float(r),

"f1\_military": float(f1),

}, ensure\_ascii=False, indent=2))

return {"model": model.name, "n": len(y\_true), "accuracy": acc, "precision": p, "recall": r, "f1": f1}, per\_item

client = OpenAI(base\_url="https://router.huggingface.co/v1", api\_key=HF\_TOKEN)

for m in ["Qwen/Qwen3-VL-8B-Instruct:novita", "zai-org/GLM-4.5V:zai-org"]:

r = client.chat.completions.create(

model=m,

messages=[{"role": "user", "content": [{"type":"text","text":"Say ok."}]}],

max\_tokens=5,

)

print(m, "->", r.choices[0].message.content)

summary = []

all\_preds = {}

for i, m in enumerate(MODELS):

prov = PROVIDERS[i] if i < len(PROVIDERS) else None

res, per\_item = run\_eval\_for\_model(VLMConfig(m, prov), gt, LIMIT, RESULTS\_DIR, HF\_TOKEN)

summary.append(res)

all\_preds[m] = per\_item

print("\\n=== SUMMARY ===")

for s in summary:

print(f"{s['model']}: n={s['n']} acc={s['accuracy']:.3f} F1(military)={s['f1']:.3f} P={s['precision']:.3f} R={s['recall']:.3f}")

df\_summary = pd.DataFrame(summary)

df\_summary