DJANGO.PUB · ЛНТУ

МІТАП + ОНЛАЙН

Django Pub AI Edition

Як вбудовувати AI у Django-проєкти — від даних і класичного ML до продакшену моделей, LLM, RAG і AI-агентів. **Django лишається серцем застосунку, AI працює поруч.**

ДАТА	ПОЧАТОК	ДЕ
29.10, чт	13:20	ЛНТУ, ауд. П228

Скоро почнемо · займайте зручні місця

ТРИ БЛОКИ

01 13:20 · БЛОК 1
Data Science & Classical ML

02 14:10 · БЛОК 2
Deep Learning & MLOps

03 14:55 · БЛОК 3
LLM, RAG & AI Agents

+ живі розмови, live demo і Q&A

DJANGO.PUB · ЛНТУ

ПРОГРАМА

Чт, 29 жовтня 2026 · початок о 13:20
ЛНТУ, корпус №2, ауд. П228 · вул. Потебні, 56 · + онлайн

13:20 30–35 хв	БЛОК 1	**Data Science & Classical ML — фундамент даних** NumPy · Pandas · Polars · scikit-learn · OpenCV · Matplotlib · Plotly · Jupyter
13:55 15 хв	РОЗМОВА	**Жива розмова: дані й ML на практиці** Роботи студентів і відкрита розмова із залом
14:10 ≈ 30 хв	БЛОК 2	**Deep Learning & MLOps — продакшен та інфраструктура** PyTorch · Hugging Face · ONNX · MLflow · Docker · Kubernetes · Terraform · Celery + Redis
14:40 15 хв	РОЗМОВА	**Жива розмова: DevOps і архітектура** Автоматизація DevOps, складні архітектури, реакція на навантаження
14:55 ≈ 15 хв	БЛОК 3	**LLM, RAG & AI Agents — інтелект нового покоління** pgvector · LangChain · LangGraph · LlamaIndex · OpenAI · Claude · Ollama · Django Channels
15:10 10 хв	ІНТЕРАКТИВ	**Live demo: AI-асистент заходу** Чат, який знає матеріали всіх виступів
15:20	Q&A	**Запитання до спікерів**

DATA SCIENCE & CLASSICAL ML

ФУНДАМЕНТ ДАНИХ У DJANGO-ПРОЄКТАХ

DJANGO — ЦЕ НЕ ТІЛЬКИ CRUD

01
АНАЛІТИЧНІ ДАШБОРДИ НА МІЛЬЙОНАХ ЗАПИСІВ

02
РЕКОМЕНДАЦІЙНІ СИСТЕМИ

03
ОБРОБКА ЗОБРАЖЕНЬ: РЕСАЙЗ, МОДЕРАЦІЯ, OCR

04
СЕМАНТИЧНИЙ ПОШУК (→ БЛОК 3)

МОВА, ЯКОЮ ДУМАЮТЬ ДАНІ

NUMPY: ФУНДАМЕНТ СТЕКУ

NPARRAY

20–100× ШВИДШЕ

ОСНОВА СТЕКУ

```python
import numpy as np
values = SensorReading.objects.values_list("temperature", flat=True)
arr = np.array(list(values))
# Z-score — один рядок замість циклу
normalized = (arr - arr.mean()) / arr.std()
```

QUERYSET → DATAFRAME ЗА ОДИН КРОК

PANDAS + DJANGO ORM

.VALUES()

GROUPBY + AGG

PIVOT TABLE

```python
import pandas as pd
# .values() → словники замість моделей → менше пам'яті
qs = Order.objects.filter(status="completed").values(
    "created_at", "total", "customer_id", "category")
df = pd.DataFrame.from_records(qs)
# Групування + агрегація — те, що складно через ORM
df["month"] = df["created_at"].dt.to_period("M")
monthly = df.groupby("month").agg(revenue=("total", "sum"),
                                  avg_check=("total", "mean"))
# Pivot table — одним рядком
pivot = df.pivot_table("total", index="month", columns="category")
```

ЧОМУ PANDAS ПАДАЄ НА МІЛЬЙОНАХ РЯДКІВ?



FROM_RECORDS(QS)

- DataFrame тягне весь QuerySet у RAM.
- 10M рядків → гігабайти пам'яті.
- MemoryError або OOM kill.



ЧАНКИ АБО POLARS

- Обробка частинами по 50K рядків.
- Lazy-запити замість повного завантаження.
- Пам'ять під контролем.

КОЛИ PANDAS ЗАМАЛО

POLARS: RUST, LAZY, ARROW

RUST — УСІ ЯДРА

LAZY ЯК QUERYSET

APACHE ARROW

```python
import polars as pl
qs = Order.objects.filter(status="completed").values("total", "customer_id")
df = pl.DataFrame(list(qs))
result = (
    df.lazy()                          # план запиту, як QuerySet
      .filter(pl.col("total") > 100)
      .group_by("customer_id")
      .agg(pl.col("total").sum())
      .collect()                       # ← виконання
)
```

SCIKIT-LEARN У DJANGO

НАВЧАННЯ (ОФЛАЙН):

```python
from sklearn.ensemble import RandomForestClassifier
import joblib

X_train, y_train = build_features()   # з ORM
model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)

joblib.dump(model, "models/churn.pkl")  # кілька МБ
```

ІНФЕРЕНС У DJANGO VIEW:

```python
# views.py — модель вантажиться один раз
model = joblib.load("models/churn.pkl")

def predict_churn(request, customer_id):
    x = get_customer_features(customer_id)
    prob = model.predict_proba([x])[0][1]
    risk = ("high" if prob > .7 else
            "medium" if prob > .4 else "low")
    return JsonResponse({"churn": round(prob, 3), "risk": risk})
```

ДАНІ З ORM → **RANDOMFOREST.FIT** → **JOBLIB → VIEW**

ТИПОВІ ЗАДАЧІ

- ресайз і стиснення при завантаженні;
- детекція облич для модерації;
- OCR з документів.

Pillow — простий і вже є в ImageField. Для ресайзу цього достатньо.

OpenCV — швидший і вміє CV-задачі. Важку обробку — у Celery.

ОБРОБКА ЗОБРАЖЕНЬ: КОЛИ PILLOW ЗАМАЛО

OPENCV + DJANGO

MATPLOTLIB

Статичні PNG/SVG для звітів, email і PDF. На сервері — use("Agg") і plt.close(fig)

PLOTLY

Інтерактивні графіки: JSON з Django → Plotly.js на фронті. Дашборди і real-time

JUPYTER

Прототипи через shell_plus --notebook. Пісочниця для експериментів, а не прод

MATPLOTLIB · PLOTLY · JUPYTER

ВІЗУАЛІЗАЦІЯ І ПІСОЧНИЦЯ ДЛЯ ДАНИХ

ПОВНИЙ DATA-ПАЙПЛАЙН У ДІЇ

КАРТА ІНСТРУМЕНТІВ БЛОКУ 1 — ВІД QUERYSET ДО ГРАФІКА

NUMPY
масиви і швидкість

PANDAS · POLARS
таблична аналітика

SCIKIT-LEARN
класичний ML у view

OPENCV
комп'ютерний зір

MATPLOTLIB
і Plotly: звіти та дашборди

ЖИВА РОЗМОВА

13:55 · 15 хв

ДАНІ Й ML НА ПРАКТИЦІ

Роботи студентів і відкрита розмова із залом про інструменти в реальних проєктах.

01

Роботи студентів

Студенти показують власні проєкти з NumPy, Pandas, Polars, scikit-learn і OpenCV.

02

Інструменти в компаніях

Як команди використовують ці бібліотеки у продакшені — реальні приклади.

03

SQL чи Pandas?

Де межа між «зробити в SQL» і «витягнути в Pandas»? Коли брати Polars?

ДАЛІ 14:10 **Блок 2 · Deep Learning & MLOps**

DEEP LEARNING & MLOPS

ПРОДАКШЕН ТА ІНФРАСТРУКТУРА: ВІД НОУТБУКА ДО КЛАСТЕРА

ПРО ЩО ЦЯ ЧАСТИНА?

01
PYTORCH, HUGGING FACE ТА ONNX

02
MLFLOW: ЕКСПЕРИМЕНТИ І РЕЄСТР

03
DOCKER, KUBERNETES ТА TERRAFORM

04
CELERY + REDIS У DJANGO

«ПРАЦЮЄ НА МОЇЙ МАШИНІ»

Різні версії CUDA, PyTorch і бібліотек. Модель, що працює в Jupyter, падає на сервері або дає інші результати

НЕВІДТВОРЮВАНІ ЕКСПЕРИМЕНТИ

model_final_v3_REAL.pt у Downloads. Ніхто не знає, на яких даних і гіперпараметрах навчена модель, що зараз у продакшені

ІНФЕРЕНС БЛОКУЄ СЕРВЕР

Модель на 2 ГБ вантажиться в кожен воркер, а 3-секундний інференс прямо в Django view вбиває пропускну здатність

ВІД НОУТБУКА ДО ПРОДАКШЕНУ

ТРИ ТИПОВІ МІСЦЯ, ДЕ ВСЕ ЛАМАЄТЬСЯ — І ЯКІ ЗАКРИВАЄ MLOPS

FINE-TUNING ЧЕРЕЗ TRAINER:

```python
from transformers import AutoModelForSequenceClassification
from transformers import Trainer, TrainingArguments
model = AutoModelForSequenceClassification.from_pretrained(
    "xlm-roberta-base", num_labels=3)
args = TrainingArguments("out", num_train_epochs=3,
    per_device_train_batch_size=16, fp16=True)
trainer = Trainer(model=model, args=args,
    train_dataset=ds["train"], eval_dataset=ds["test"])
trainer.train()
```

ІНФЕРЕНС ЧЕРЕЗ PIPELINE:

```python
from transformers import pipeline
clf = pipeline("text-classification",
               model="./out/best", device=0)
clf("Доставка затримується вже третій день")
# [{'label': 'complaint', 'score': 0.97}]
```

ДАТАСЕТ З HUB → **FINE-TUNING НА GPU** → **ГОТОВА МОДЕЛЬ**

PYTORCH + HUGGING FACE

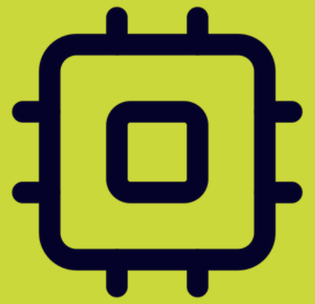
НАВІЩО ONNX, ЯКЩО Є PYTORCH?



ШВИДШИЙ ІНФЕРЕНС НА CPU



ОБРАЗ БЕЗ PYTORCH І CUDA



ОДИН ФОРМАТ — БУДЬ-ЯКИЙ RUNTIME

ЕКСПОРТ ТА ОПТИМІЗАЦІЯ МОДЕЛІ

ONNX: ВІД PYTORCH ДО RUNTIME

ЕКСПОРТ З PYTORCH **КВАНТИЗАЦІЯ INT8** **ONNX RUNTIME**

```python
# 1. Експорт з PyTorch
torch.onnx.export(model, (input_ids, attention_mask), "model.onnx",
    input_names=["input_ids", "attention_mask"], output_names=["logits"],
    dynamic_axes={"input_ids": {0: "batch", 1: "seq"}}, opset_version=17)
# 2. Динамічна квантизація: файл ~4x менший, швидше на CPU
quantize_dynamic("model.onnx", "model.int8.onnx", weight_type=QuantType.QInt8)
# 3. Інференс без PyTorch
sess = ort.InferenceSession("model.int8.onnx", providers=["CPUExecutionProvider"])
logits = sess.run(None, {"input_ids": ids, "attention_mask": mask})[0]
```

ПАМ'ЯТЬ ВАШИХ ЕКСПЕРИМЕНТІВ

MLFLOW: ТРЕКІНГ І РЕЄСТР

ПАРАМЕТРИ ТА МЕТРИКИ

MODEL REGISTRY

ALIAS @CHAMPION

```python
mlflow.set_experiment("support-classifier")
with mlflow.start_run():
    mlflow.log_params({"lr": 2e-5, "epochs": 3, "base": "xlm-roberta"})
    trainer.train()
    mlflow.log_metric("f1", trainer.evaluate()["eval_f1"])
    mlflow.onnx.log_model(onnx_model, "model", registered_model_name="support-clf")
# Промоушен: перевішуємо alias на нову версію
MlflowClient().set_registered_model_alias("support-clf", "champion", version=7)
model = mlflow.pyfunc.load_model("models:/support-clf@champion")
```

ЖИТТЄВИЙ ЦИКЛ МОДЕЛІ

ВІД ЕКСПЕРИМЕНТУ ДО ПРОДАКШЕНУ ЧЕРЕЗ MODEL REGISTRY

ЕКСПЕРИМЕНТ

параметри, метрики, артефакти

РЕЄСТРАЦІЯ

нова версія моделі в Registry

ВАЛІДАЦІЯ

тести на відкладених даних

@CHAMPION

alias → продакшен, відкат за секунду

НАВІЩО DOCKER, ЯКЩО Є VENV?



РУЧНЕ РОЗГОРТАННЯ

- Конфлікти версій CUDA і драйверів.
- Сервер налаштовано руками — ніхто не пам'ятає як.
- Відкат = ніч без сну.



DOCKER-ОБРАЗ

- Модель, код і залежності в одному артефакті.
- Однаковий образ у dev, staging і prod.
- Відкат — це попередній тег.

DOCKERFILE ІНФЕРЕНС-СЕРВІСУ:

```dockerfile
FROM python:3.12-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY model.int8.onnx ./models/
COPY . .
CMD ["gunicorn", "core.wsgi", "-w", "2", "-b", "0.0.0.0:8000"]
```

АВТОСКЕЙЛ У KUBERNETES (HPA):

```yaml
kind: HorizontalPodAutoscaler
spec:
  scaleTargetRef:
    kind: Deployment
    name: inference
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource: {name: cpu, target: {averageUtilization: 70}}
```

DOCKER-ОБРАЗ → **DEPLOYMENT + SERVICE** → **HPA АВТОСКЕЙЛ**

DOCKER + KUBERNETES

ЩО ДАЄ KUBERNETES?

- HPA додає поди під навантаженням;
- rolling update без простою;
- GPU-ноди лише там, де потрібен інференс;

Docker-образ з ONNX-моделлю → Deployment + Service + HPA. Terraform описує сам кластер.

Readiness probe не пускає трафік на под, доки модель не завантажилась у пам'ять.

САМОВІДНОВЛЕННЯ, АВТОСКЕЙЛ І ДЕПЛОЙ БЕЗ ПРОСТОЮ

KUBERNETES: ІНФЕРЕНС, ЩО МАСШТАБУЄТЬСЯ САМ

ВІДТВОРЮВАНІСТЬ

Dev, staging і prod піднімаються з одного коду. Новий кластер — це terraform apply, а не тиждень ручної роботи

РЕВ'Ю ЧЕРЕЗ PULL REQUEST

terraform plan показує зміни ще до застосування. Інфраструктура проходить code review так само, як застосунок

КОНТРОЛЬ ВИТРАТ НА GPU

GPU-пули з автоскейлом від нуля. Дорогі ноди існують лише тоді, коли є реальне навантаження

TERRAFORM: ІНФРАСТРУКТУРА ЯК КОД

КЛАСТЕР, GPU-НОДИ ТА СХОВИЩЕ АРТЕФАКТІВ — У GIT

ЧОМУ ІНФЕРЕНС НЕ МОЖНА РОБИТИ У VIEW?



СИНХРОННИЙ VIEW

- Запит висить 5–30 секунд.
- Таймаути gunicorn і nginx.
- Один важкий запит блокує воркер.



CELERY + REDIS

- Миттєва відповідь з task_id.
- Ретраї, таймаути та пріоритети.
- Воркери масштабуються окремо від вебу.

ФОНОВИЙ ІНФЕРЕНС У DJANGO

CELERY + REDIS: ЧЕРГА ЗАДАЧ

МОДЕЛЬ У ПАМ'ЯТІ ВОРКЕРА

РЕТРАЇ ТА ТАЙМАУТИ

REDIS: БРОКЕР І РЕЗУЛЬТАТ

```python
_session = None
def get_session():  # модель вантажиться один раз на воркер
    global _session
    if _session is None:
        _session = ort.InferenceSession("models/model.int8.onnx")
    return _session

@shared_task(autoretry_for=(ConnectionError,), max_retries=3, soft_time_limit=30)
def classify_ticket(ticket_id):
    ticket = Ticket.objects.get(pk=ticket_id)
    ticket.label = predict(get_session(), ticket.text)
    ticket.save(update_fields=["label"])

# у view: classify_ticket.delay(ticket.id) → 202 Accepted
```

ПОВНИЙ MLOPS-ПАЙПЛАЙН У ДІЇ

ЯК УСІ ІНСТРУМЕНТИ ЗБИРАЮТЬСЯ В ОДИН ПРОДАКШЕН-ПРОДУКТ

PYTORCH + HF
навчання та fine-tuning

MLFLOW
трекінг і реєстр моделей

ONNX
оптимізація інференсу

DOCKER · K8S · TF
деплой та інфраструктура

CELERY + REDIS
фоновий інференс у Django

1

ЛОГУЙТЕ В MLFLOW

Додайте mlflow.autolog() у тренувальний скрипт — і кожен запуск уже має історію

2

ЕКСПОРТУЙТЕ В ONNX

Заміряйте латентність PyTorch vs ONNX Runtime на своїй моделі та даних

3

ІНФЕРЕНС — У CELERY

Жодної моделі у view: лише .delay(), task_id і статус задачі



З ЧОГО ПОЧАТИ:

ТРИ КРОКИ НА ЗАВТРА

ПІДСУМОК ТА Q&A

DEEP LEARNING & MLOPS: ПРОДАКШЕН ТА ІНФРАСТРУКТУРА

ЖИВА РОЗМОВА

14:40 · 15 хв

DEVOPS І АРХІТЕКТУРА

Роботи студентів і відкрита розмова із залом про інфраструктуру й навантаження.

01

Роботи студентів

Студентські проєкти: деплой моделей, контейнери, черги задач і моніторинг.

02

Автоматизація DevOps

Що вже автоматизується і яке майбутнє чекає на саму професію.

03

Складні архітектури

Оптимізація архітектури, ухвалення рішень і реакція на пікові навантаження.

ДАЛІ 14:55 **Блок 3 · LLM, RAG & AI Agents**

LLM, RAG & AI AGENTS У DJANGO

ІНТЕЛЕКТ НОВОГО ПОКОЛІННЯ: ВІД ЗАПИТІВ ДО АВТОНОМНИХ СИСТЕМ

ПРО ЩО ЦЯ ЧАСТИНА?

01
RAG З ВИКОРИСТАННЯМ PGVECTOR

02
LANGCHAIN & LANGGRAPH

03
STREAMING В DJANGO ЗАСТОСУНКАХ

НАВІЩО PGVECTOR, ЯКЩО Є POSTGRESQL?



ВСЕ В ОДНІЙ БАЗІ



ВИКОРИСТАННЯ ЗВИЧАЙНОГО DJANGO ORM



ВІДСУТНІСТЬ РОЗСИНХРОНУ В БД

ПОШУК ЧЕРЕЗ PGVECTOR:

```python
from pgvector.django import VectorField, CosineDistance
class KnowledgeChunk(models.Model):
    text = models.TextField()
    embedding = VectorField(dimensions=1536)
chunks = KnowledgeChunk.objects.order_by(
    CosineDistance("embedding", q_vec)
)[:3]
```

СТРІМІНГ ЧЕРЕЗ CHANNELS:

```python
async def receive(self, text_data):
    # Генерація та стрімінг у сокет
    async for chunk in agent.astream(prompt):
        await self.send(text_data=chunk)
```

ЗАПИТ КОРИСТУВАЧА → ПОШУК В ORM + LLM → СТРІМІНГ ТОКЕНІВ У ЧАТ

ЯК ЦЕ ПРАЦЮЄ?

ЛІНІЙНІ ЛАНЦЮЖКИ

Односпрямованість без зворотного зв'язку. Будь-яка помилка моделі у форматі JSON чи збій API обриває весь процес

НЕСКІНЧЕННІ ЦИКЛИ

При помилці інструменту або відсутності даних модель зациклюється, повторюючи виклики та спалюючи гроші на API

НЕСТРУКТУРОВАНИЙ СТАН

Перекидання сирих нетипізованих словників між кроками. Відсутність детермінованого State Machine унеможливлює контроль контексту у складній бізнес-логіці

ЕВОЛЮЦІЯ ОРКЕСТРАЦІЇ: LANGCHAIN → LANGGRAPH

ПЕРЕХІД ВІД СТАТИЧНИХ ПАЙПЛАЙНІВ ДО ДЕТЕРМІНОВАНИХ ГРАФІВ

КОМПОНЕНТНИЙ ПІДХІД ДО АГЕНТІВ

LANGGRAPH: СТАН, ВУЗЛИ ТА ДІЇ

ПАМ'ЯТЬ ДІАЛОГУ

ВИКЛИК ІНСТРУМЕНТІВ

БЕЗПЕЧНА ВАЛІДАЦІЯ

```python
@tool
def get_order_status(order_id: int, user: User) -> str:
    """Отримує статус замовлення з обов'язковою перевіркою прав"""
    order = Order.objects.filter(id=order_id, user=user).first()
    return order.status if order else "Замовлення не знайдено"
# Збірка графа агента в LangGraph
builder.add_node("agent", llm_with_tools)
builder.add_node("tools", ToolNode([get_order_status]))
builder.add_edge("agent", "tools")
```

ЧОМУ СТРІМІНГ, А НЕ ЗВИЧАЙНИЙ HTTP?



ОДИН ВЕЛИКИЙ ЗАПИТ (HTTP)

- Порожній екран зі спінером на 15 секунд.
- Відчуття, що сайт або чат завис.
- Кілька користувачів здатні покласти весь сервер.



DJANGO CHANNELS

- Текст друкується миттєво в реальному часі.
- Звичний і приємний UX рівня ChatGPT.
- Сервер спокійно витримує пікові навантаження залу.

ПОВНИЙ AI-ПАЙПЛАЙН У ДІЇ

ЯК ВСЕ ЗБИРАЄТЬСЯ В ОДИН ПРОДАКШЕН-ПРОДУКТ

CHANNELS (ВХІД) ● **LANGGRAPH + RAG** ● **СТРІМІНГ (ВИХІД)**

ЩО ВМІЄ НАВЧАЛЬНИЙ БОТ?

- точні цитати з презентацій спікерів;
- жодних вигаданих фактів або галюцинацій;
- пошук за технічним стеком заходу;

Система розгорнута на зв'язці Django + pgvector + LangGraph.

У базу завантажено всі матеріали виступів. Ви можете запитати усе про що ми сьогодні говорили

RAG-БОТ З УСІМА ЗНАННЯМИ СЬОГОДНІШНЬОГО ЗАХОДУ

ЖИВИЙ ІНТЕРАКТИВ ДЛЯ.. ЗАЛУ: AI-АСИСТЕНТ ПОДІЇ

ЯК ПРОТЕСТУВАТИ:

ТРИ ПРОСТІ КРОКИ

1

КЛОНУЙТЕ РЕПО

Завантажте навчальний репозиторій з GitHub за посиланням або QR (Буде в кінці виступу)

2

ВСТАВТЕ КЛЮЧ

Вставте тимчасовий API-ключ у .env (даємо доступ на 3 дні)

3

ЗАПУСКАЙТЕ ПРОЕКТ

Запустіть локально та поспілкуйтесь з ботом, змініть контекст, розберіться з реалізацією



ПІДСУМОК ТА Q&A

LLM, RAG & AI AGENTS У DJANGO



ДОСТУП ДО РЕПОЗИТОРІЮ ТА КЛЮЧА ДОСТУПУ