Лекція 1. Вступ до штучного інтелекту та машинного навчання в розробці ігор

Курс: Штучний інтелект в ігрових застосунках
Спеціальність: 121 — Інженерія програмного забезпечення

Національний університет "Львівська Політехніка"
Львів, 2026

Мета лекції

Після цієї лекції ви будете:

1. Розуміти три парадигми ШІ та їх застосування в іграх
2. Орієнтуватися в таксономії машинного навчання
3. Знати історію розвитку ШІ від 1950-х до сьогодення
4. Володіти базовою термінологією для лабораторної роботи №1

Теми

1. Розуміння ШІ: три парадигми
2. Історія ШІ: від Тюрінга до ChatGPT
3. Таксономія машинного навчання
4. ШІ в ігровій індустрії
5. Інструменти курсу
6. Ключові поняття для лаб. роботи №1
7. Дорожня карта курсу

Що таке штучний інтелект?

Концептуально: системи, що виконують задачі, які вимагають людського інтелекту — розпізнавання образів, прийняття рішень, розуміння мови

Технічно: алгоритми та моделі, що навчаються з даних або працюють за заданими правилами для вирішення конкретних задач

Три основні парадигми:

Парадигма	Підхід	Приклад
Символьний ШІ	Явні правила, логіка	Експертні системи, FSM
Статистичний ШІ	Навчання з даних	SVM, Random Forest
Нейронні мережі	Глибоке навчання	CNN, Transformer, LLM

Символьний (логічний) ШІ

Підхід: явне кодування знань у вигляді правил

- Експертні системи: ЯКЩО... ТО...
- Бази знань та онтології
- Дерева рішень

В іграх: скінченні автомати станів (FSM)

```
[Патрулювання] --бачить ворога--> [Переслідування] --наблизився--> [Атака]
      ^                                                                |
      +---------------------- ворог зник <-----------------------------+
```

Так працює більшість NPC у класичних іграх.

Обмеження: ручне кодування, погана масштабованість, нездатність навчатися

Статистичний ШІ та машинне навчання

Підхід: алгоритм знаходить закономірності в даних автоматично

- Лінійна та логістична регресія
- Метод опорних векторів (SVM)
- Випадковий ліс (Random Forest)
- Кластеризація (k-means)

В іграх: процедурна генерація контенту зі статистичним контролем різноманітності

Перевага: навчається з прикладів, не вимагає ручного кодування правил

Нейронні мережі та глибоке навчання

Підхід: штучні нейрони, організовані у глибокі шари, автоматично вивчають ієрархічні ознаки

- **CNN** — для зображень (комп'ютерний зір)
- **RNN / LSTM** — для послідовностей (текст, аудіо, ігрові дії)
- **Transformer** — основа сучасних LLM
- **GAN / Дифузійні моделі** — генерація контенту

В іграх: NPC з мовленнєвими моделями (Nvidia ACE, Inworld AI) — діалоги генеруються LLM в реальному часі

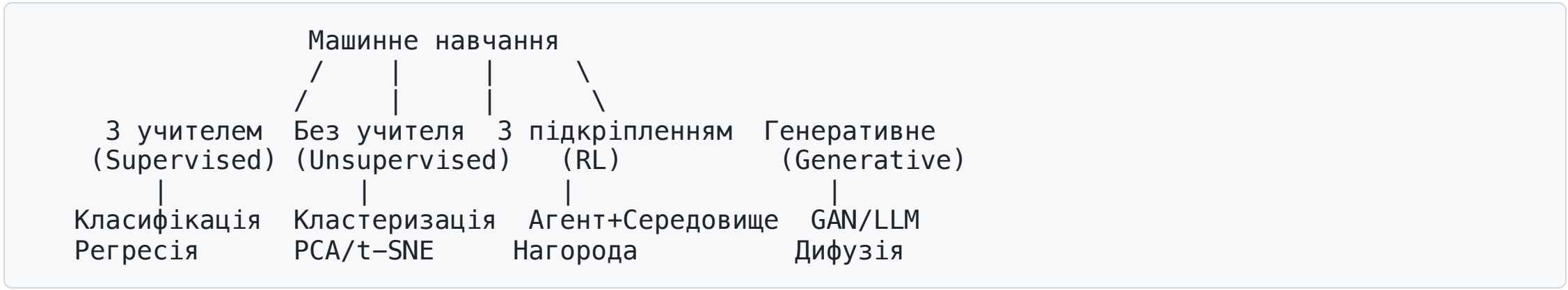
Хронологія ШІ (1950–2012)

Рік	Подія
1950	Тест Тюрінга
1956	Дартмутська конференція — народження ШІ
1960-70	Перші експертні системи, шахові програми
1980-ті	"Зима ШІ"; backpropagation (1986)
1997	Deep Blue перемагає Каспарова
2012	**AlexNet** — переворот у глибокому навчанні

Хронологія ШІ (2015–2026)

Рік	Подія	Зв'язок з курсом
2015	**ResNet** — skip connections	Лаб. 1: ResNet-18
2016	**AlphaGo** перемагає Лі Седоля	Лекції 10-12: RL
2017	**Transformer**, **Grad-CAM**	Лаб. 1: Grad-CAM
2020	GPT-3	Лекція 9: LLM
2022	ChatGPT, Stable Diffusion	Лекція 13: генерація
2023	GPT-4, Claude — мультимодальність	Лекція 15: агенти
2024-26	AI-агенти, ШІ в продакшені	Лекції 15-17

Таксономія машинного навчання



Навчання з учителем (Supervised Learning)

Найважливіша парадигма для лабораторної роботи №1

Вхід: набір пар (зображення, мітка): $\{(x_i, y_i)\}_{i=1}^{N}$

Мета: знайти параметри θ моделі f_θ, що мінімізують:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(x_i), y_i)$$

де ℓ — **функція втрат** (*loss function*), наприклад **крос-ентропія**

Два типи задач:

- **Класифікація:** "це кіт" або "це собака" (дискретні мітки)
- **Регресія:** рейтинг гравця = 1547.3 (неперервне значення)

Класифікація vs Регресія

Класифікація

Вхід: зображення

Вихід: клас (мітка)

```
Зображення --> [Модель] --> "кіт" (92%)
                            "собака" (8%)
```

Функція втрат: Cross-Entropy

Регресія

Вхід: ознаки гравця

Вихід: число

```
Статистика --> [Модель] --> 1547.3
```

Функція втрат: MSE (Mean Squared Error)

Лаб. робота №1 — бінарна класифікація зображень (два класи)

Навчання без учителя

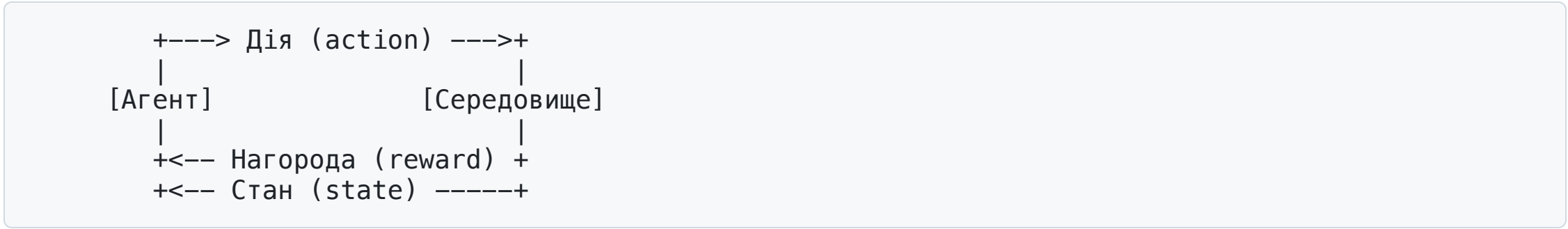
Дані без міток — модель шукає приховану структуру

- **Кластеризація** (k-means, DBSCAN) — групування схожих об'єктів
- **Зменшення розмірності** (PCA, t-SNE) — візуалізація багатовимірних даних
- **Виявлення аномалій** — знаходження нетипових поведінок

В іграх:

- Сегментація гравців за стилем гри
- Автоматичне виявлення читерів
- Аналіз патернів поведінки для балансування

Навчання з підкріпленням (RL)



Мета: навчитися стратегії, що максимізує сумарну нагороду

В іграх:

- AlphaGo (го) — надлюдський рівень
- AlphaStar (StarCraft II) — стратегія реального часу
- OpenAI Five (Dota 2) — командна гра 5 на 5
- Unity ML-Agents — тренування ігрових NPC

Генеративний ШІ

Моделі, що **створюють новий контент**:

Тип	Що генерує	Приклад
GAN	Зображення	StyleGAN (обличчя)
Дифузія	Зображення з тексту	Stable Diffusion, DALL-E
LLM	Текст, код, діалоги	GPT-4, Claude
Аудіо	Музику, голос	Bark, XTTS

В іграх:

- Процедурна генерація текстур та спрайтів
- NPC з динамічними діалогами (LLM)
- Генерація квестових описів та лору

ШІ в іграх: класична ера

Pac-Man (1980) — 4 привиди, 4 характери

Привид	Стратегія
Blinky (червоний)	Переслідує напряму
Pinky (рожевий)	Перехоплює попереду
Inky (блакитний)	Позиція відносно Blinky
Clyde (оранжевий)	Переслідує, але тікає поблизу

Інші класичні підходи:

- **FSM** — скінченні автомати (більшість NPC до 2010-х)
- **Behavior Trees** — гнучкіша альтернатива (Halo, Unreal Engine)
- **A*** — пошук шляху (NavMesh у Unity та Unreal)

ШІ в іграх: стратегії та настільні ігри

- **Deep Blue (1997)** — перемога над Каспаровим, пошук по дереву
- **Stockfish + NNUE** — нейромережева оцінка шахових позицій
- **AlphaGo (2016)** — Monte Carlo Tree Search + глибокі нейронні мережі
- **Civilization** — AI керує цілою цивілізацією
- **StarCraft: Brood War** — AI-турніри (AIIDE, SSCAIT)

Ключовий алгоритм: Monte Carlo Tree Search (MCTS)

- Поєднує випадковий пошук з побудовою дерева рішень
- Лежить в основі AlphaGo та багатьох сучасних game AI

ШІ в іграх: сучасність

ML-driven підходи:

- **Unity ML-Agents** — тренування ігрових агентів через RL
- **No Man's Sky** — процедурна генерація планет
- **NPC з LLM** — Nvidia ACE, Inworld AI
- **Автоматичне тестування** — ML-агенти шукають баги

Тенденції:

- Адаптивна складність (Dynamic Difficulty Adjustment)
- Персоналізація ігрового досвіду через ML
- AI-асистенти для розробників ігор

Комп'ютерний зір для ігор → Лаб. 1

Задачі, які ви виконаєте в лабораторній роботі №1:

Задача	Що робить	Етап лаб.
Класифікація	Одна мітка на зображення	Етап 3
Grad-CAM	"Куди дивиться" модель	Етап 4
Сегментація	Маска для кожного пікселя	Етап 5
Конвеєр	Сегментація → Класифікація	Етап 6

Ігрове застосування: класифікація ресурсів, генерація мап прохідності, автоматичне QA-тестування ігрових сцен

CNN: згорткові нейронні мережі

Архітектура, спеціально розроблена для обробки зображень:

```
Зображення --> [Conv] --> [Pool] --> [Conv] --> [Pool] --> [FC] --> Клас
   224x224     Фільтри    Зменш.    Фільтри    Зменш.    Вектор   "кіт"
               Ознаки     розміру   Складніші  розміру   ознак    95%
               (краї)               ознаки
                                    (об'єкти)
```

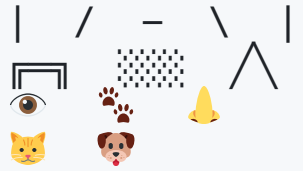
Ключові компоненти:

- **Згортковий шар (Conv)** — виявляє ознаки (краї → текстури → об'єкти)
- **Pooling** — зменшує розмір, зберігає головне
- **Fully Connected (FC)** — фінальне передбачення

CNN: від простих ознак до об'єктів

Ранні шари виявляють **прості ознаки**, глибокі — **складні**:

```
Шар 1: краї, лінії
Шар 2: кути, текстури
Шар 3: частини об'єктів
Шар 4: цілі об'єкти
```



Саме тому **передавальне навчання** працює: ранні шари (краї, текстури) однакові для будь-яких зображень!

ResNet та skip connections

Проблема: дуже глибокі мережі (50+ шарів) не тренуються — градієнти зникають

Рішення (He et al., 2015): залишкові з'єднання

```
Вхід x ----+
   |       |
[Conv]     |  (skip connection)
[Conv]     |
   |       |
   + <-----+
   |
Вихід: H(x) = F(x) + x
```

Мережа навчає лише **залишок** $F(x) = H(x) - x$, а не всю функцію

ResNet-18 (~11.7M параметрів) — використовується в лабораторній роботі №1

Передавальне навчання (Transfer Learning)

Замість тренування з нуля — використовуємо модель, навчену на ImageNet (1.2M зображень):

```
1. Завантажити ResNet-18       [=== ImageNet ваги ===][FC: 1000 класів]
                                                      ↓ видалити
2. Заморозити backbone         [=== ЗАМОРОЖЕНО ====][новий FC: 2 класи]
                                                      ↓ тренувати
3. Тренувати нову голову       [=== ЗАМОРОЖЕНО ====][FC: кіт / собака]
```

Чому це працює: ранні шари (краї, текстури) — універсальні!

Коли потрібно: мало даних (сотні зображень замість мільйонів)

Аугментація даних (Data Augmentation)

Збільшення ефективного обсягу набору шляхом випадкових трансформацій:

```
Оригінал  →  Обрізання  →  Відзеркалення  →  Поворот  →  Зміна кольору
  [🐱]        [🐱 crop]     [🐱 flip]         [🐱 rot]     [🐱 color]
```

Трансформація	Ефект
RandomResizedCrop	Випадково вирізає та масштабує
RandomHorizontalFlip	Дзеркально відображає (50%)
ColorJitter	Змінює яскравість/контраст
RandomRotation	Повертає на випадковий кут
Normalize	ImageNet mean/std

Мета: зменшити перенавчання (overfitting)

Grad-CAM: візуалізація рішень моделі

Gradient-weighted Class Activation Mapping (Selvaraju et al., 2017)

"На що дивиться модель, коли каже, що це кіт?"

```
Зображення → [CNN forward] → Скор класу "кіт"
                                    ↓ backward
                             Градієнти останнього Conv шару
                                    ↓ усереднення
                             Ваги важливості каналів
                                    ↓ зважена сума + ReLU
                             Теплова карта (7×7 → 224×224)
```

Яскравіші ділянки = більший вплив на рішення

Міст до сегментації: Grad-CAM дає розмиту область → сегментація дає точну маску

Семантична сегментація

Класифікація кожного пікселя зображення:

```
Вхід (224×224×3)  →  [Модель]  →  Маска (224×224)
                                   Кожен піксель = клас
```

Класифікація	Сегментація
Одна мітка на зображення	Мітка на кожен піксель
"Це кіт"	Які саме пікселі — кіт
Вихід: вектор (C)	Вихід: маска (H×W)

DeepLabV3 — архітектура для сегментації (натренована на COCO, 21 клас)

DeepLabV3: архітектура

```
Зображення → [Backbone: ResNet-50] → [ASPP] → [Декодер] → Маска сегментації
```

Ключові компоненти:

- **Backbone** — ResNet-50/101 витягує ієрархічні ознаки
- **Atrous (Dilated) Convolution** — згортки з "розширеними" ядрами для збільшення рецептивного поля
- **ASPP** — паралельні atrous convolution з різними коефіцієнтами → контекст на різних масштабах
- **Декодер** — відновлює просторову роздільність

COCO класи: фон, кіт, собака, людина, автомобіль, ... (21 клас)

Метрики: класифікація

Метрика	Формула	Що вимірює
Accuracy	(TP+TN) / All	Загальна точність
Precision	TP / (TP+FP)	Точність позитивних передбачень
Recall	TP / (TP+FN)	Повнота знаходження позитивних
F1-score	2·P·R / (P+R)	Баланс Precision та Recall

```
                Передбачено
                  +      −
Справжнє    +   TP     FN
            −   FP     TN
```

Метрики: сегментація

IoU (Intersection over Union)

Mean IoU — середнє IoU по всіх класах — основна метрика сегментації

Ідеальна сегментація: IoU = 1.0

Python та PyTorch

Чому PyTorch:

- Тензорні обчислення з GPU (CUDA)
- Автоматичне диференціювання (autograd)
- Гнучкий, Pythonic інтерфейс
- Стандарт у дослідженнях та індустрії

Приклад — створення моделі за 3 рядки:

```
import torch
from torchvision import models
from torchvision.models import ResNet18_Weights

model = models.resnet18(weights=ResNet18_Weights.IMAGENET1K_V1)
```

torchvision — моделі (ResNet, DeepLabV3), трансформації, набори даних

Google Colab + VS Code

Google Colab — безкоштовний GPU (NVIDIA T4, 16 ГБ VRAM)

Рекомендований спосіб роботи:

1. Встановити VS Code
2. Встановити розширення **Jupyter** (Microsoft)
3. Встановити розширення **Google Colab** (Google)
4. Відкрити `.ipynb` файл
5. Select Kernel → **Colab**
6. Авторизуватися через Google
7. Обрати GPU T4

Поєднує зручність VS Code з хмарним GPU!

Набори даних курсу

Набір	Розмір	Використання
ImageNet	1.2M зображень, 1000 класів	Pre-training моделей
COCO	21 клас об'єктів	Сегментація (DeepLabV3)
Oxford-IIIT Pet	37 порід, ~7400 зобр.	Лаб. 1 (основний)
Caltech-101	101 категорія, ~9000 зобр.	Лаб. 1 (альтернативний)

Oxford-IIIT Pet рекомендовано для лаб. 1, бо коти та собаки є класами COCO → DeepLabV3 їх коректно сегментує

Дорожня карта курсу: 18 лекцій

1. **Вступ до ШІ** (ця лекція)
2. Навчання з учителем
3. Нейронні мережі
4. Комп'ютерний зір
5. CNN та Transfer Learning
6. Detection та Segmentation
7. RNN, LSTM, Attention
8. Transformer
9. Великі мовні моделі
10. Основи RL
11. Deep RL
12. Game AI та інструменти
13. Генеративні моделі
14. Fine-tuning та RAG
15. AI-агенти
16. MLOps
17. ШІ в розробці ігор
18. Тенденції та етика

Підготовка до лабораторної роботи №1

Що потрібно мати:

- [ ] VS Code (версія 1.98+)
- [ ] Розширення Jupyter та Google Colab
- [ ] Google-акаунт
- [ ] Базові знання Python

Що ви будете робити:

1. Завантажити набір даних (Oxford-IIIT Pet або Caltech-101)
2. Натренувати класифікатор зображень (ResNet-18 + Transfer Learning)
3. Візуалізувати рішення через Grad-CAM
4. Застосувати семантичну сегментацію (DeepLabV3)
5. Побудувати конвеєр: сегментація → класифікація

Підсумок

1. **Три парадигми ШІ:** символьний → статистичний → нейронні мережі
2. **Чотири типи ML:** з учителем, без учителя, з підкріпленням, генеративний
3. **ШІ в іграх:** від FSM у Pac-Man до LLM-діалогів
4. **Інструменти:** Python + PyTorch + Google Colab
5. **Лаб. 1:** класифікація → Grad-CAM → сегментація → конвеєр

Наступна лекція:

Навчання з учителем: регресія, класифікація, градієнтний спуск, оптимізація

Запитання?

Контакти

Лектор: Бауск О.Є., к.т.н., асистент кафедри ПЗ

Корисні посилання:

- PyTorch: https://pytorch.org/docs/
- torchvision: https://pytorch.org/vision/
- Google Colab + VS Code: https://developers.googleblog.com/google-colab-is-coming-to-vs-code/