IMPORTANT: For all projects, any dataset you use or generate must be uploaded to a shared drive.

This includes:

- GPS datasets
- Synthetic datasets you generated
- Any additional datasets collected from the internet
- Labeled PGN chess games

You may use the university provided drive (up to 2 TB).
If you have Gemini Pro, you may also use your university Google Drive.

GPS Dataset Format

The dataset must follow this structure:

```
dataset_root/

├── images/

└── gt.csv
```

The `gt.csv` file must contain (3 columns):

1. `image_name.png` / `image_name.jpg`
2. **Latitude**
3. **Longitude**

Each row must correspond to one image in the `images/` folder.

Chess Dataset Format

PGN based data:
Save the PGN files exactly in the same format as the provided ground truth games.

Synthetic or real image based data:
Organize the data as follows (per game or per dataset):

```
dataset_root/

├── images/

└── gt.csv
```

The `gt.csv` file must contain 3 columns:

1. **`image_name.png` / `image_name.jpg`**
2. **FEN string corresponding to the image**
3. **View specification (e.g., *white pieces closer to the camera* or *black pieces closer to the camera*)**

Each row must correctly match the image and its board state.

Final Report Guidelines (All Projects)

Each project submission must include a **final written report** describing the problem, methodology, experiments, and conclusions.

The report should be written in the **style of a scientific paper**, following the structure outlined below.

Length & Format

- **Maximum length:** up to **25 pages**
- **Font size:** 12 pt (or standard LaTeX article/conference defaults)
- **Format:** PDF
- **Language:** English

Important:
You are **not required** to use all 25 pages.
Do **not** artificially inflate the report length.
Extra pages are intended for **figures, tables, experiments, and ablations**.

Recommended Report Structure

1. Abstract

- Concise summary of:
  - The problem
  - Your approach
  - Main results

---

2. Introduction

- Task description and motivation
- Challenges and goals
- Main contributions

---

3. Related Work

- Relevant papers, datasets, repositories

    - Academic papers (if you used it)
    - GitHub repositories
    - Dataset websites
- Clarify differences from prior work if you built upon that.

---

4. Method

- Model architecture
- Input/output representation
- Training procedure
- Loss functions
- Preprocessing/postprocessing
- Diagrams encouraged

---

5. Experiments

- Dataset description and splits
- Evaluation metrics
- Baselines and comparisons
- Quantitative results (tables, plots, numbers)
- Qualitative results (visualizations)

---

6. Ablation Study (Required)

If your method consists of components (e.g., **X, Y, Z**), you **must** justify them via ablation.

- Remove **Y** → what happens?
- Remove **Z** → what happens?
- Provide:
    - Comparison tables
    - Clear explanations in text

The goal is to demonstrate that each component is **beneficial and necessary**.

---

7. What Did *Not* Work (Optional)

You are encouraged to include a short section describing:

- Ideas you tried that **did not work**
- Approaches that failed or underperformed
- Design choices you abandoned and why

This section may include:

- Negative results
- Failed experiments
- Insights learned from mistakes

Important:
You are **not penalized** for things that did not work.
On the contrary, clear and honest analysis of failures is viewed positively and reflects good research practice.

---

8. Discussion / Limitations (Optional but Recommended)

- Failure cases
- Limitations of the method
- Possible future improvements (if you had more time)

---

9. References

- Papers, datasets, repositories, websites

Code Submission (Required)

A GitHub repository must be provided, containing:

- All source code
- `requirements.txt`
- Clear instructions for:
    - Environment setup (from git clone to installations)
    - Training
    - Running inference / evaluation

A `README.md` must explain how to reproduce results.

Final Project Presentation Guidelines

Each group is required to present their project in a **short oral presentation**.

Duration

- **Presentation length: 7–10 minutes**
- The **exact time per group** will be announced **next week**, after the total number of presenting groups is finalized.

Please prepare your presentation to fit **strictly within the allocated time**.

Presentation Content

The presentation should be **clear, concise, and well-structured**.
Focus on the **core ideas and insights**, not on implementation details.

1. Short Introduction

- Introduce the presenters:
    - Name
    - Degree program (BSc / MSc / etc, Computer Science with Physics etc.)
- Very briefly state:
    - Which project you worked on
    - **Why you chose this project**

---

2. Problem Statement

- Keep this section **short and focused (since more teams are presenting the exact same project)**

---

3. Method & Solution

- Explain your proposed approach:
    - High-level idea
    - Key components of your method
- Avoid low level code details (unless you do project 5)
- Use diagrams or illustrations if helpful

---

4. What Is Special About Your Solution

- Explain what do you think differentiates your approach from:
    - Other groups
    - Baseline methods
- This could include:
    - A design choice
    - A modeling decision
    - A clever simplification
    - A novel idea or insight

---

5. Results & Ablation

- Present:
    - Key quantitative (tables etc) results
    - Visual examples (since it's a computer vision project!)
    - Ablation results (what happens when components are removed)
    - Focus on **insights**, not on many numbers

---

6. What You Learned from the project (Optional)

- Share:
    - What you learned from the project
    - Challenges you encountered
    - Insights that surprised you
- This can be:
    - Technical
    - Conceptual
    - Practical (e.g., debugging, evaluation, dataset issues)

General Guidelines

- Slides should be **visual**, not text heavy
- Figures, diagrams, and tables are strongly encouraged
- Do **not** try to cover everything from the report, focus on the key components.
- Clarity and insight matter more than completeness

Evaluation Emphasis

Presentations will be evaluated based on:

- Clarity of explanation
- Understanding of the problem and solution
- Quality of results and insights
- Time management

Webpage (Optional but Strongly Recommended)

Students are encouraged to create a **project webpage** using **GitHub Pages (`github.io`)**. You can later show that webpage to your friends, interviewers, etc.

Example project pages:

- https://feature-3dgs.github.io/
- https://bgu-cs-vil.github.io/FastJAM/

Example source code:

- https://github.com/feature-3dgs/feature-3dgs.github.io

A webpage is optional, but highly recommended for sharing the project publicly.

Important Notes

- Do **not** use ChatGPT or similar AI tools to **artificially generate long text**.
- Quality, clarity, and honesty matter more than length.

Important Dates & Deadlines

Project Presentations

- **Dates: January 20–21, 2026**
- **Time: During regular lecture hours**
- **Attendance: Mandatory**

All groups must be present on their assigned presentation day. (a registration link will be sent).

Final Submission Deadline

- **Deadline: January 24, 2026**

By this date, each group must finalize and submit:

- **Final report (PDF)**
- **GitHub repository**
- **Optional project webpage**
- **All required evaluation functions**
- **Complete, runnable code**

The project must be fully finalized by this deadline.

Late or incomplete submissions may result in penalties.

Evaluation API Specification

Projects 1 & 2 – Chessboard State Prediction

Overview

In **Projects 1 and 2**, each submission **must implement a final evaluation function** that receives a **single RGB image of a chessboard** and returns a **structured 2D representation of the board state**.

This function will be used for **automatic evaluation**, therefore its **signature, output format, device placement, and value conventions must be followed exactly**.

Required Function

Students must implement the following function:

```
def predict_board(image: np.ndarray) -> torch.Tensor:
    """
    Predict the chessboard state from a single RGB image.
    """
```

Input Specification

- `image`
    - Type: `numpy.ndarray`
    - Shape: `(H, W, 3)`
    - Channel order: **RGB**
    - Dtype: `uint8`
    - Value range: `[0, 255]`
    - The image contains a **single chessboard**, possibly under varying illumination, viewpoint (black or white view), or partial occlusion.

Output Specification

The function must return a **2D tensor** representing the chessboard state.

Tensor Requirements

- Type: `torch.Tensor`
- Shape: `(8, 8)`
- Device: **CPU**
- Dtype: `torch.int64` (or compatible integer type)

Each element in the tensor corresponds to **one square on the chessboard**.

Board Coordinate Convention

- `output[0, 0]` corresponds to the **top-left chess board square of the image**
- `output[0, 7]` → top-right square
- `output[7, 0]` → bottom-left square
- `output[7, 7]` → bottom-right square

Important:
The mapping is purely **image-based**, not chess notation based (i.e., no assumption of White/Black orientation).

Class Encoding

Each square in the output tensor must contain an integer value in the range [0, 14], according to the following encoding:

Value	Class
0	White Pawn
1	White Rook
2	White Knight
3	White Bishop

4	White Queen
5	White King
6	Black Pawn
7	Black Rook
8	Black Knight
9	Black Bishop
10	Black Queen
11	Black King
12	Empty Square
13	Out-of-Distribution (OOD) / Unknown / Invalid (project 1 only)

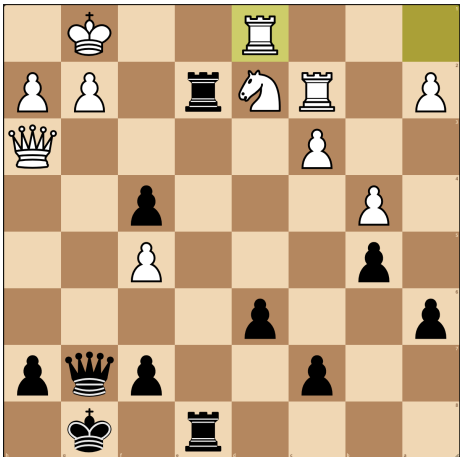
Out-of-Distribution (OOD) Handling (Project 1 only)

- If a square:
    - Does not contain a valid chess piece
    - Is occluded or ambiguous
    - Contains an object not belonging to the 12 known classes or it's not a clear empty square (occluded etc)

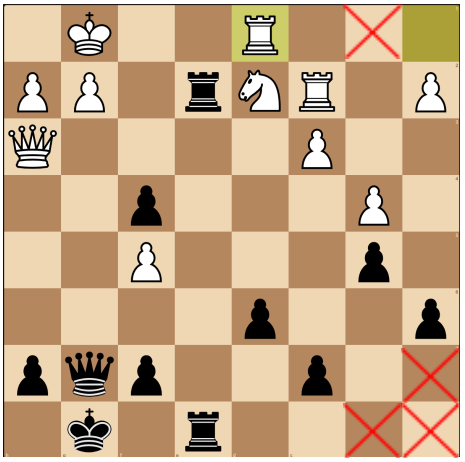
→ **Return 13 for that square**.

Using 13 is **mandatory** for all uncertain predictions.

Output image (save in `./results/`)



Project 1: for OOD squares draw red **X**



Example Output

```
tensor([
    [7, 8, 9, 10, 11, 9, 8, 7],
    [6, 6, 6, 6, 6, 6, 6, 6],
    [12,12,12,12,12,12,12,12],
    [12,12,12,12,12,12,12,12],
    [12,12,12,12,12,12,12,12],
    [12,12,12,12,12,12,12,12],
    [0, 0, 0, 0, 0, 0, 0, 0],
    [1, 2, 3, 4, 5, 3, 2, 1]
])
```

Additional Constraints

- ❌ Do **not** return probabilities or logits
- ❌ Do **not** return NumPy arrays
- ❌ Do **not** return tensors on GPU
- ✅ The function must run **independently** without interactive input
- ✅ The output must be **deterministic** given the same image

---

Evaluation Notes

- The evaluation system will:
    - Call `predict_board(...)`
    - Verify tensor **shape, dtype, device**
    - Compare predicted values against ground truth
- Any deviation from the specification may result in **automatic failure**.

Project 3 – Chessboard Image Generation from FEN

Overview

In **Project 3**, each submission must implement a final evaluation function that receives a **ground-truth chess position (FEN)** and a **viewpoint specification**, and generates:

1. A **synthetic chessboard image**
2. A **realistic chessboard image**
3. A **side-by-side comparison image**

The outputs must be saved to disk in a **fixed directory structure**.
These results will be used for **manual evaluation**, therefore **file paths, formats, and naming conventions must be followed exactly**.

Required Function

Students must implement the following function:

```
def generate_chessboard_image(fen: str, viewpoint: str) -> None:
    """
    Generate synthetic and realistic chessboard images from a given FEN.
    """
```

Input Specification

`fen`

- Type: `str`
- Format: **FEN string**
- Must follow **exactly the same format and conventions** as the ground-truth FENs provided in the CSV files for Projects 1 & 2.

`viewpoint`

- Type: `str`
- Allowed values:
    - `"white"` – white-side view
    - `"black"` – black-side view

The viewpoint defines the **camera orientation/board orientation** in the generated images, it just means if black pieces or white pieces are closer to the camera.

Output Specification

The function **must save images to disk** under the following directory:

`./results/`

All images must be saved **independently**, not returned from the function.

Required Output Files

For each `(fen, viewpoint)` pair, the following files must be generated:

1. Synthetic Image
Path:

`./results/synthetic.png`

-
- Description:
    - A clean, synthetic rendering of the chessboard as you used in your data.
    - No photorealism required here (it's the synthetic image)
    - Must accurately reflect the input FEN and viewpoint

---

2. Realistic Image
Path:

`./results/realistic.png`

- Description:
    - A realistic image derived from the synthetic image
    - Must preserve as best as you can the **board state and viewpoint in the fen and synthetic image you used.**

---

3. Side-by-Side Comparison Image
Path:

`./results/side_by_side.png`

-
- Description:
    - A single image showing:
        - **Left:** synthetic image
        - **Right:** realistic image
    - Images must be aligned and clearly visible

Image Format Requirements

- File format: `PNG`
- Color format: RGB

Resolution is **not fixed**, but all three images must have **compatible dimensions** such that side-by-side visualization is clear.

Viewpoint Consistency

- The generated images must reflect the given `viewpoint`:
    - `"white"` → white pieces closest to the camera
    - `"black"` → black pieces closest to the camera
- No assumptions about chess notation orientation are allowed beyond the viewpoint input.

Additional Constraints

- ❌ Do not return images from the function
- ❌ Do not require interactive input
- ❌ Do not overwrite files outside `./results/`
- ✅ The function must be deterministic for the same `(fen, viewpoint)`
- ✅ The function must create the `./results/` directory if it does not exist

---

Project 4 – Image-to-GPS Regression

Overview

In **Project 4**, each submission must implement a final evaluation function that receives a **single RGB image** and returns a **GPS coordinate prediction**: **latitude** and **longitude**.

This function will be used for **automatic evaluation**, therefore its **signature, input format, output format, and conventions must be followed exactly**.

Required Function

Students must implement the following function:

```
def predict_gps(image: np.ndarray) -> np.ndarray:
    """
    Predict GPS latitude and longitude from a single RGB image.
    """
```

Input Specification

- **`image`**
    - Type: `numpy.ndarray`
    - Shape: `(H, W, 3)`
    - Channel order: **RGB** (NOT BGR)
    - Dtype: `uint8`
    - Value range: `[0, 255]`

The image corresponds to a real location from the dataset. Images may include viewpoint changes, illumination differences, occlusions, and motion blur.

Output Specification

The function must return a **NumPy array** containing the predicted GPS coordinate.

Array Requirements

- Type: `numpy.ndarray`
- Shape: `(2,)`
- Dtype: floating type (`float32`)
- Format:
    - `output[0]` = **latitude**
    - `output[1]` = **longitude**

Important Notes

- The returned values must be the **absolute GPS coordinates** (e.g, not relative meters).
- Predictions must be **as accurate as possible** because evaluation is done against the **exact ground-truth GPS** for each images. (which means “5 meter” error is considered **perfect (and probably not achievable)** results, as GPS give accurate location result up to 5 meters. I will take some margin (some more mistake than 5 meters will still be considered as perfect result, but I will not provide you with that number!). I will do that since different phones have different focal lengths and field of views.

Example Output

```
np.array([31.262345, 34.803210], dtype=np.float32)
```

Additional Constraints

- ❌ Do not return a list/tuple (must be `np.ndarray`)
- ❌ Do not return strings
- ❌ Do not return meters or any relative coordinate system
- ✅ No interactive input
- ✅ Must run end-to-end from a single function call

Evaluation Notes

The evaluation system will:

- Call `predict_gps(image)`
- Verify output:
    - Type is `np.ndarray`
    - Shape is `(2,)`
    - Values are finite floats
    - Latitude/longitude are within valid ranges
- Compare against the ground truth GPS for each test image (exact latitude/longitude format from the dataset).

Any deviation from the specification may result in **automatic failure**.

Project 5 – WTConv:

You should provide a comparison code between WTConv and your WTConv implementation that compares the times.

It should also compare basic network training using your WTConv against the Real WTConv comparing the total time (of all the training) and accuracy (using the same seed,weights etc). You can choose any standard architecture that we learned like ResNet, MobileNet, ConvNext etc (need to choose 1).