MPSoC'26
24TH MPSoC
Victoria, B.C., Canada, June 21st - 26th, 2026
KIOS Research and Innovation Center of Excellence

SIGHT: Saliency-Driven Selective Tile Processing for Resource-Constrained Object Detection

Theo Theocharides

Associate Professor & Department Head, Department of Electrical and Computer Engineering

Director of Research, KIOS Research and Innovation Center of Excellence

University of Cyprus

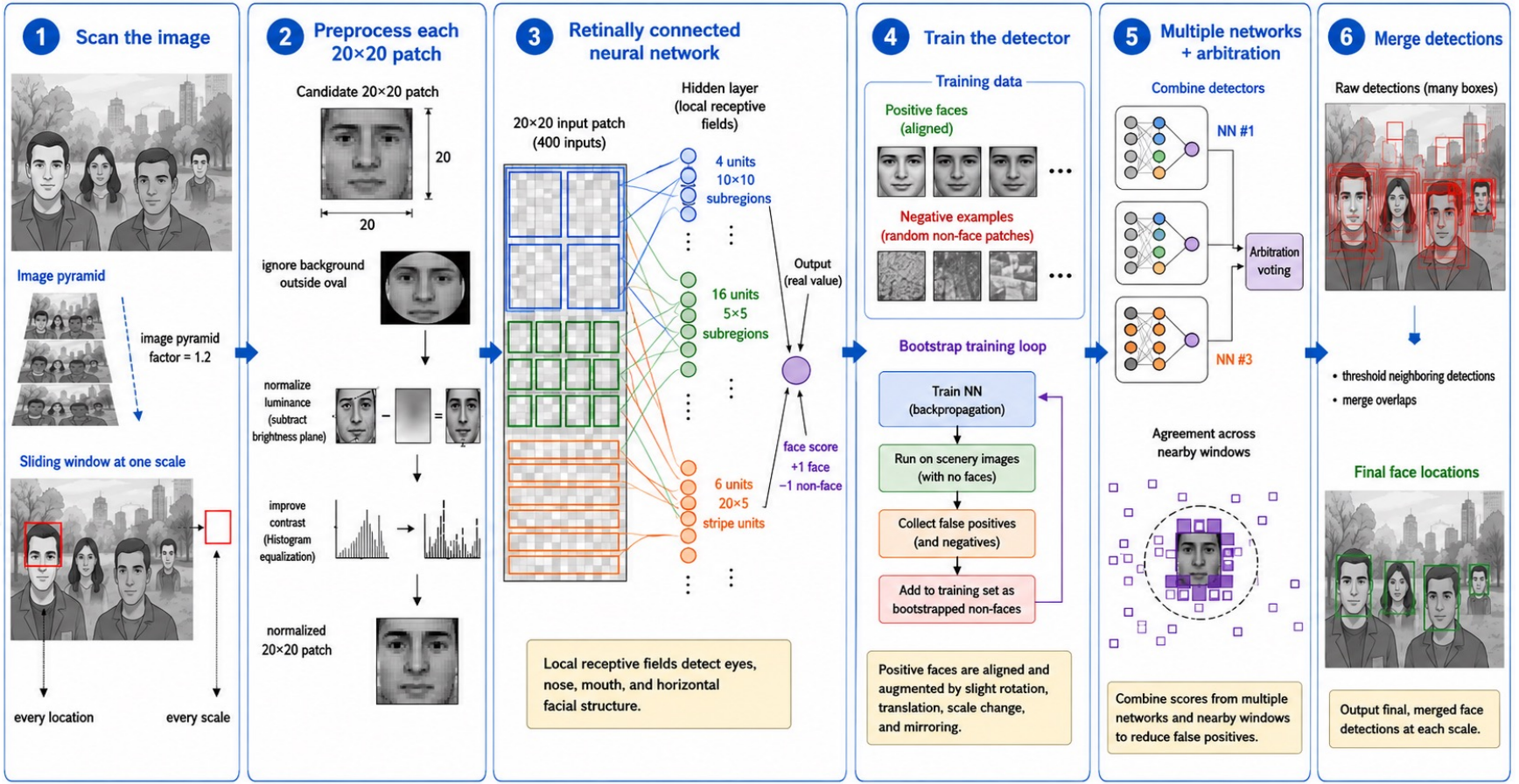
University of Cyprus
Imperial College London

Parts of this work were supported by the European Union's Horizon Europe research and innovation programme under grant agreement No 101168067 (GuardAI) and the from the European Union's Horizon 2020 research and innovation programme under grant agreement No 739551 (KIOS CoE) & from the Government of the Republic of Cyprus through the Cyprus Deputy Ministry of Research, Innovation and Digital Policy.

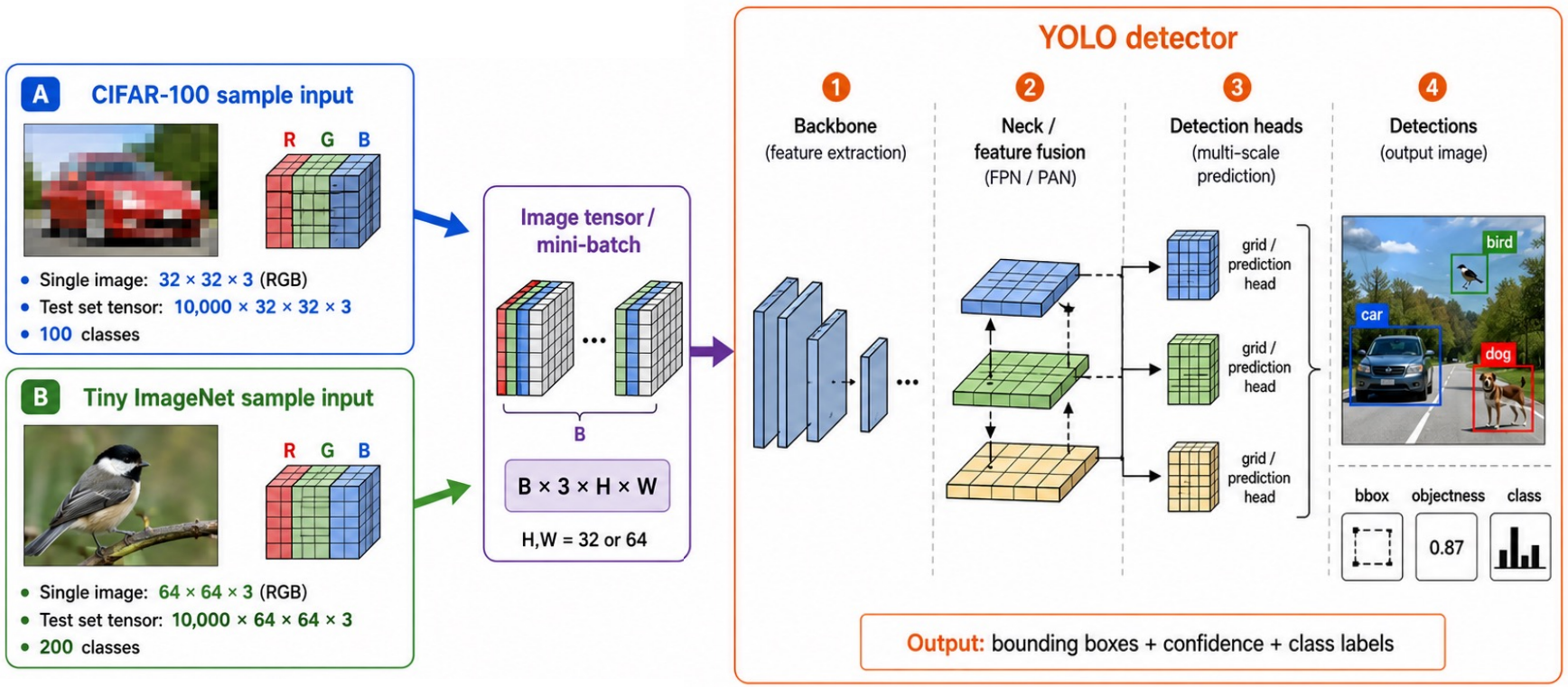
FUNDED BY:


4K TRAFFIC SCENE
3840 x 2160 | ~8.3 Million Pixels
Massive visual information
Every region may contain
objects to detect
EDGE AI
PROCESSOR
ONBOARD EDGE PLATFORM
Limited Compute
Limited Memory
Limited Energy
Thermal Constraint
EDGE UAV CONSTRAINTS
COMPUTE LOAD
Very High
MEMORY PRESSURE
High
DATA VOLUME
Very High
ENERGY CONSUMPTION
High
LATENCY REQUIREMENT
Tight
RAW DATA VOLUME
~24–30 MB per frame (4K)
At 30 FPS → ~0.7–0.9 GB/s
Exhaustive CNN processing
on edge is computationally
and energetically expensive

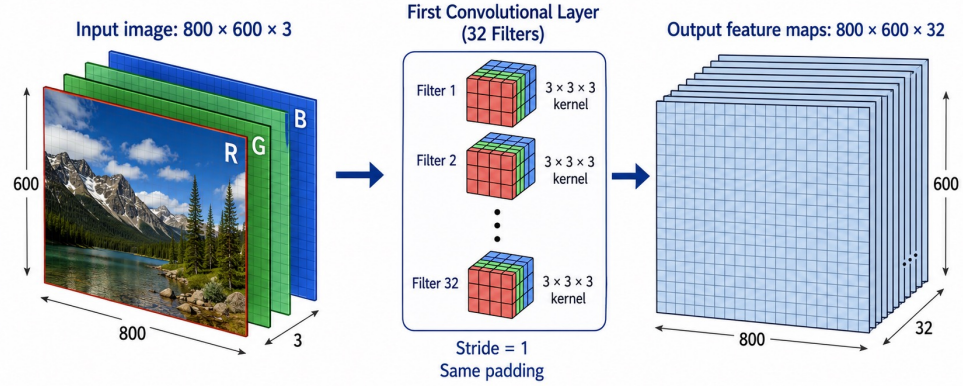
Throwback: Rowley and Kanade (PAMI, 1998)



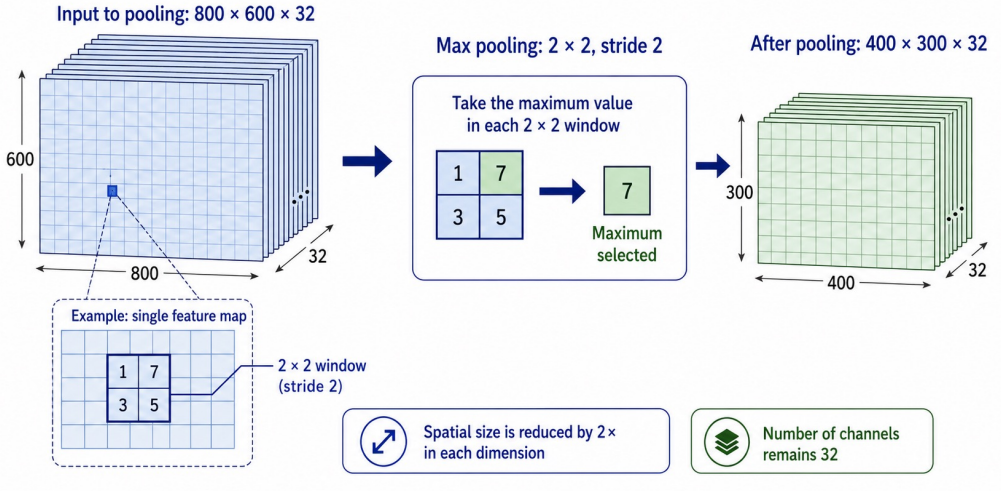
As engineers, we like to build bottom-up!



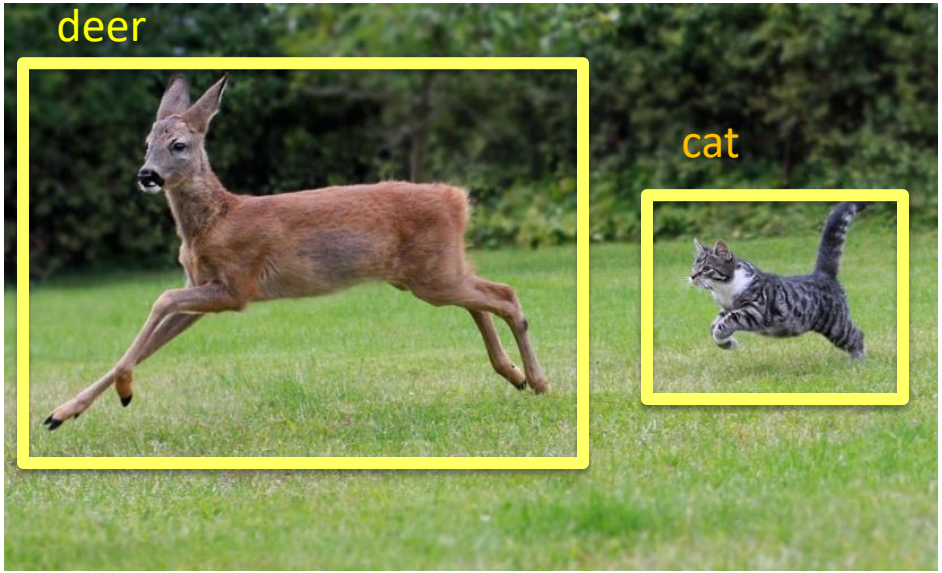
Convolutional Neural Nets - Input



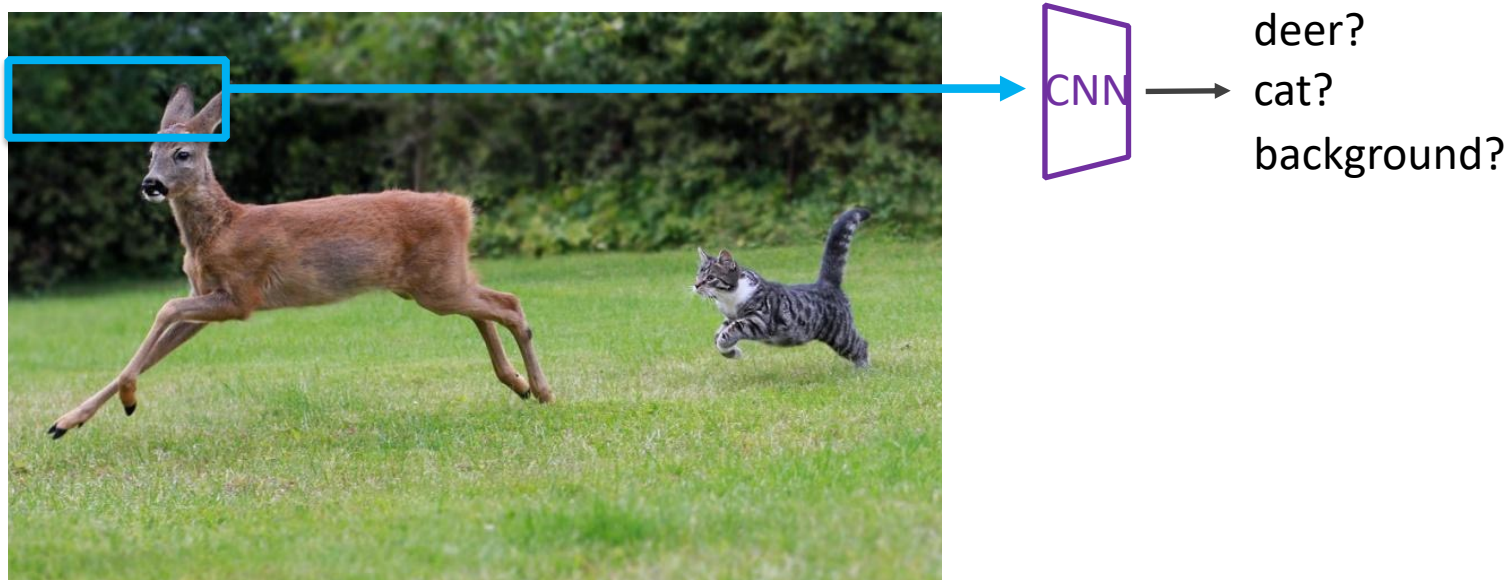
32 filters → 32 output channels



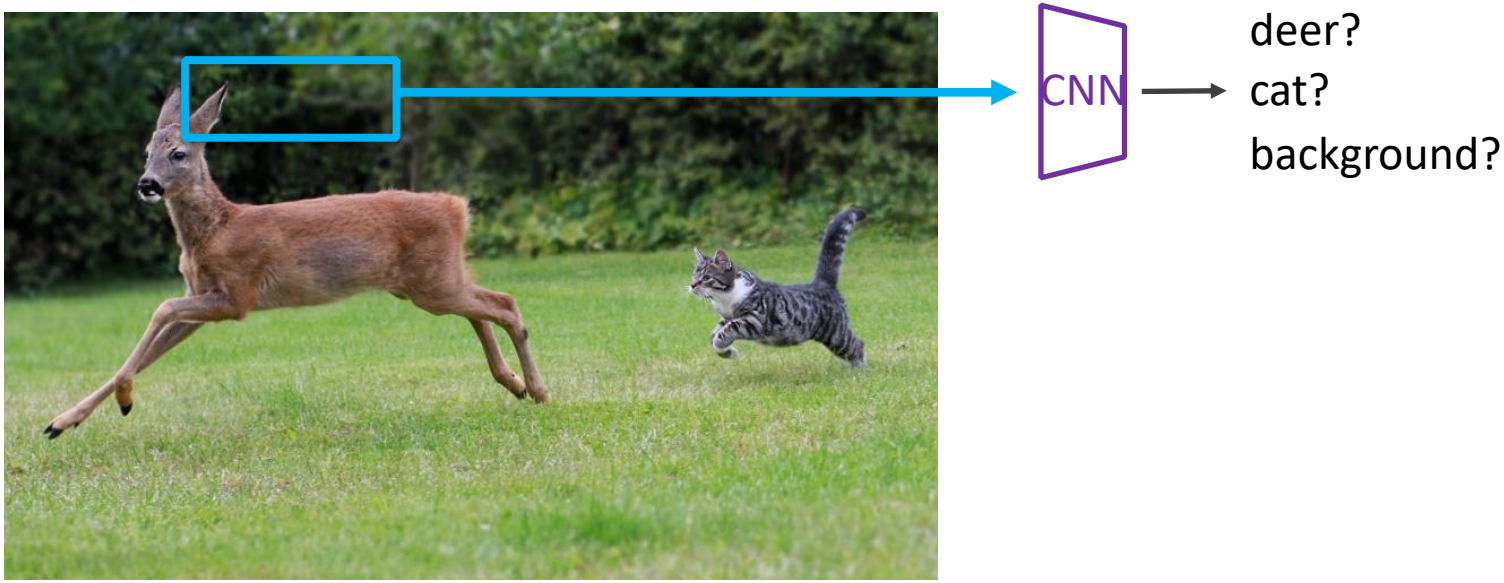
Object Detection – A Closer Look



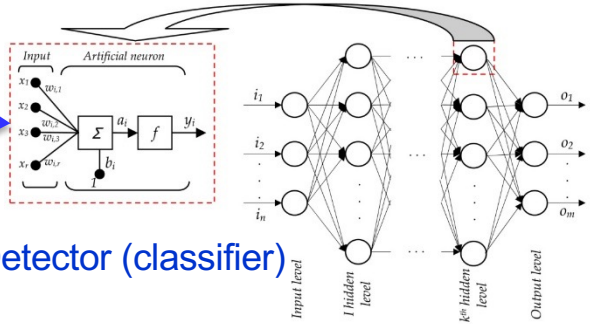
Object Detection as Classification



Object Detection as Classification with Sliding Window



Object Detection & Sliding Window Search



Detector (classifier)

Candidate Search Windows

Image Pyramid

Candidate Search Windows → Detector

Yolo 11 – Single Shot Detector

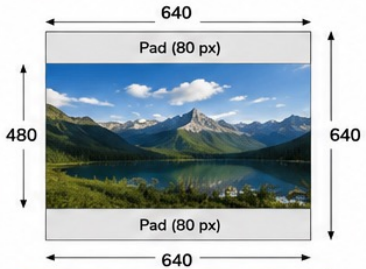
YOLO11 Input Preparation and First Convolution (from 1024×768 RGB Image)

1 Original Input Image
1024×768×3 (RGB)

Pixels: 1024 × 768 = 786,432
Channels: 3
Total values: 2,359,296

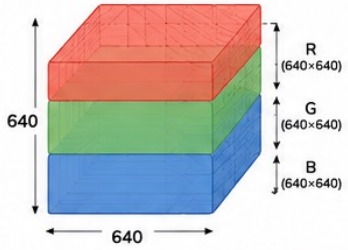
2 Resize with Aspect Ratio (Preserve) + Letterbox to 640×640

Scale factor = min(640/1024, 640/768) = 0.625
Resized image: 640×480
Pad top & bottom with 80 px each



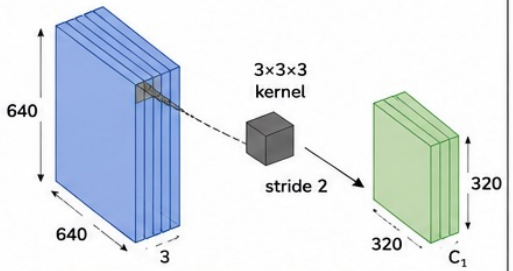
YOLO11 input size: 640 × 640 × 3
Pixels: 640 × 640 = 409,600
Total values: 1,228,800

3 YOLO11 Input Layer
640×640×3



Shape: 640 × 640 × 3
Values: 1,228,800

4 First Convolutional Layer (YOLO11)
Conv: 3×3, Stride 2, Out Channels = C_1
(C_1 depends on YOLO11 variant; base YAML uses 64)



Output Feature Map (P1/2): 320 × 320 × C_1
Spatial size reduced by 2× (H and W)
Values: 320 × 320 × C_1 = 102,400 × C_1

Summary: 1024×768 RGB image → letterbox to 640×640 RGB → first conv (3×3, stride 2) → feature map 320×320×C_1
Spatial size reduced by 2× in the first layer; channel count increases to C_1 (e.g., 64 in base model).

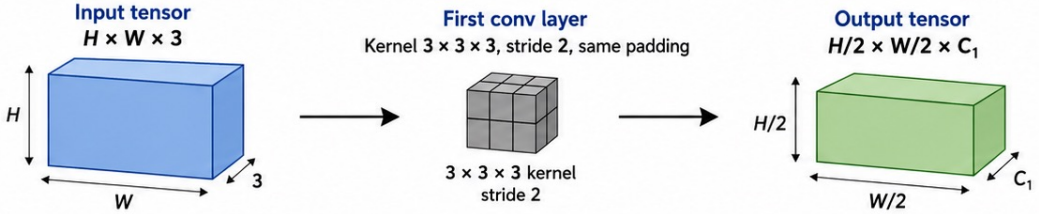
Stride-2 convolution halves the spatial dimensions:
640 × 640 → 320 × 320

C1 denotes the number of output channels in the first YOLO11 layer.

Images grow!!! (SVGA → 8K)

YOLO11 First Convolutional Layer Complexity Growth with Input Resolution

Assume the first layer is a 3×3×3 convolution, stride 2, same padding, producing an output feature map of $H/2 \times W/2 \times C_1$



ⓘ C_1 denotes the number of output channels in the first YOLO11 layer

	1024 × 768	1920 × 1080 (Full HD)	2560 × 1440 (2K / QHD)	3840 × 2160 (4K / UHD)	7680 × 4320 (8K / UHD-2)
Input size	1024 × 768 × 3	1920 × 1080 × 3	2560 × 1440 × 3	3840 × 2160 × 3	7680 × 4320 × 3
Input pixels	786,432	2,073,600	3,686,400	8,294,400	33,177,600
First conv output size	512 × 384 × C_1	960 × 540 × C_1	1280 × 720 × C_1	1920 × 1080 × C_1	3840 × 2160 × C_1
Output spatial positions / channel	196,608	518,400	921,600	2,073,600	8,294,400
First conv total MACs	196,608 × 27 × C_1	518,400 × 27 × C_1	921,600 × 27 × C_1	2,073,600 × 27 × C_1	8,294,400 × 27 × C_1
Relative first-layer complexity	1×	2.64×	4.69×	10.55×	42.19×
Relative activation memory	1×	2.64×	4.69×	10.55×	42.19×

MACs Formula

For a 3×3×3 first layer,

$$\text{MACs} = \left(\frac{H}{2} \times \frac{W}{2}\right) \times 27 \times C_1$$

Scaling Note

Because stride = 2 and C_1 is fixed, complexity scales linearly with image resolution.

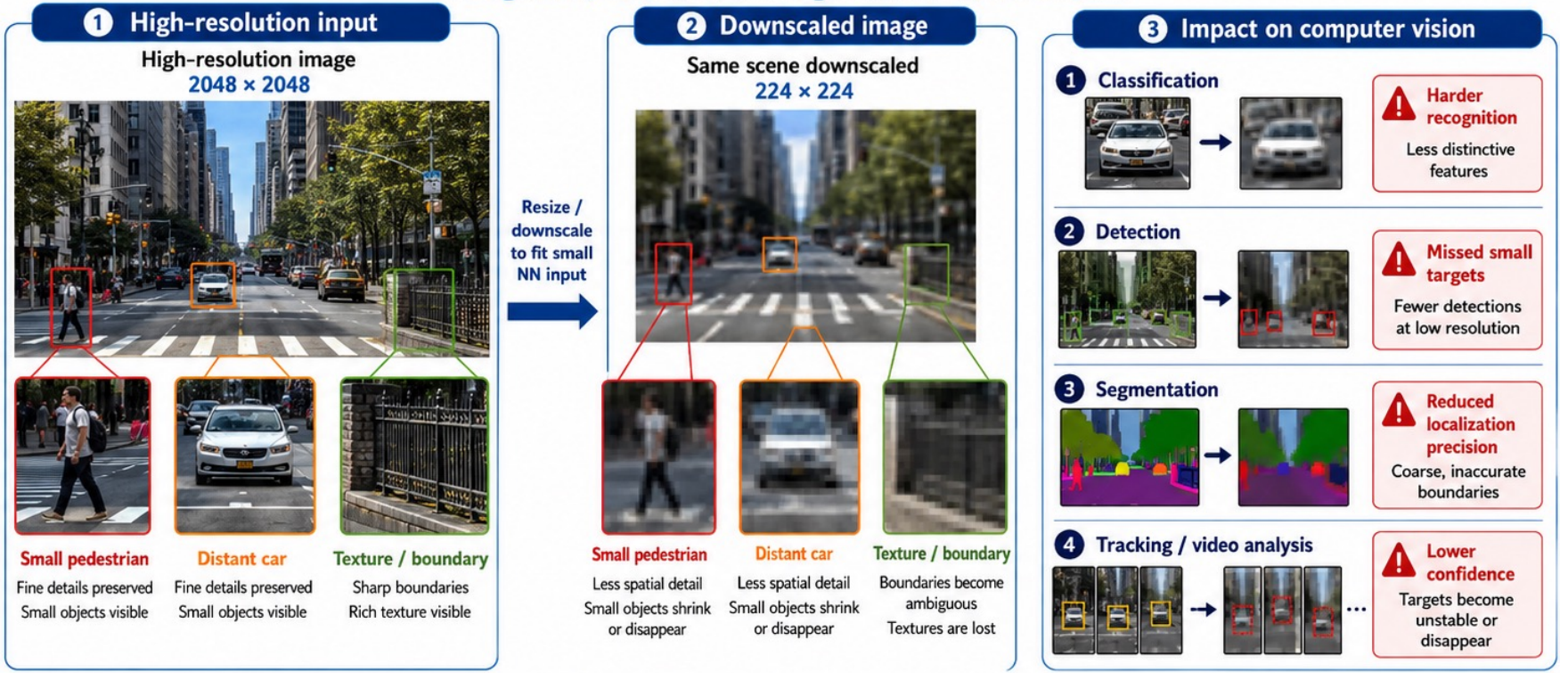
Takeaway: If YOLO11 processed the full image resolution directly, the first convolutional layer alone would become ~42.19× more expensive at 8K than at 1024×768, increasing computation, latency, power, and memory traffic.

The problem now becomes top-down!

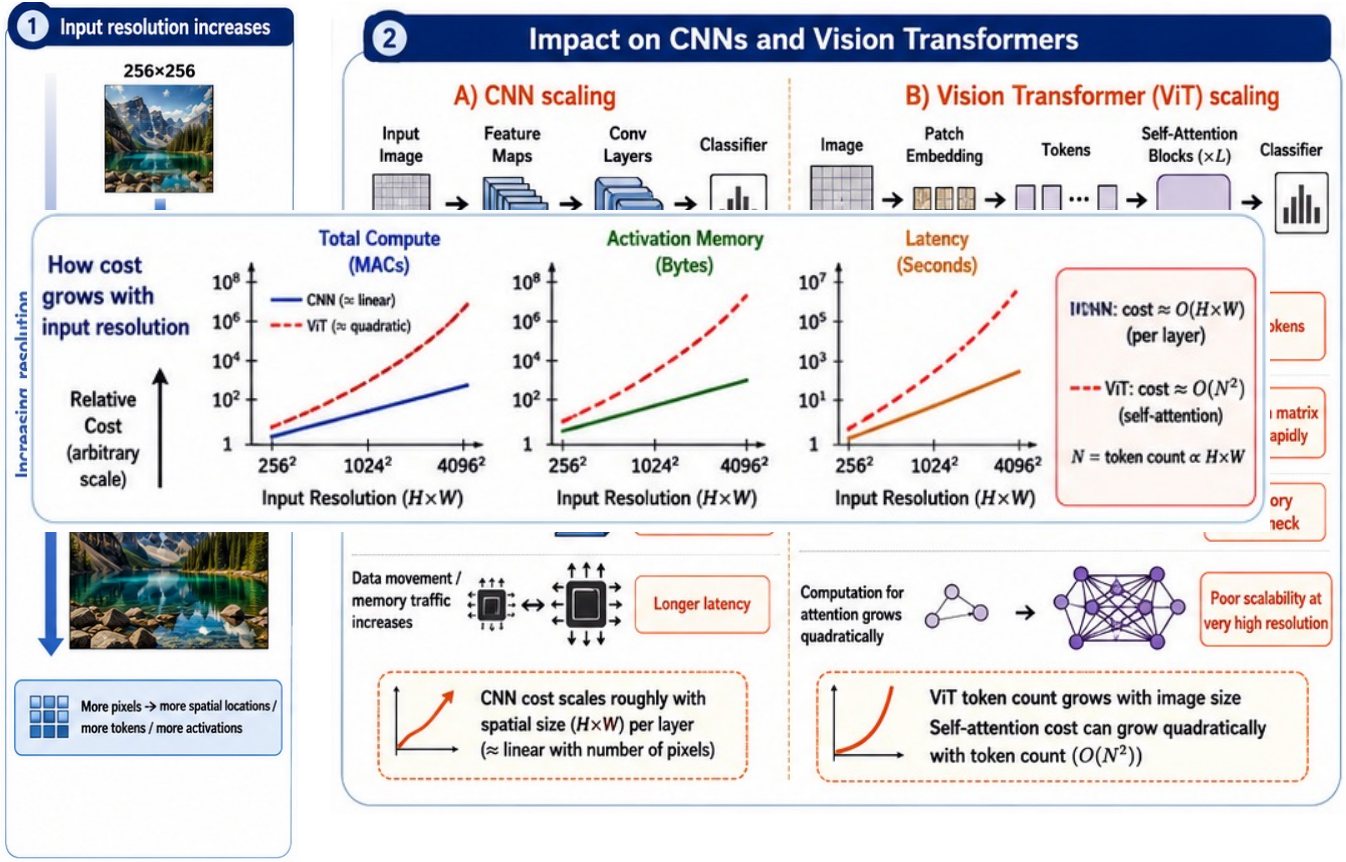
Input Resolution (H × W × 3)	1024×768 (Original)	1920×1080 (Full HD)	2560×1440 (2K / QHD)	3840×2160 (4K / UHD)	7680×4320 (8K / UHD-2)
1. YOLO11 Input (after letterbox to 640×640)	640 × 640 × 3 (1.23M values)	640 × 640 × 3 (1.23M values)	640 × 640 × 3 (1.23M values)	640 × 640 × 3 (1.23M values)	640 × 640 × 3 (1.23M values)
2. First Conv Output (3×3, stride 2)	320 × 320 × C_1 (102,400 × C_1 values)	320 × 320 × C_1 (102,400 × C_1 values)	320 × 320 × C_1 (102,400 × C_1 values)	320 × 320 × C_1 (102,400 × C_1 values)	320 × 320 × C_1 (102,400 × C_1 values)
3. Original Image Pixels (H × W)	786,432	2,073,600	3,686,400	8,294,400	33,177,600
4. Relative Pixels (vs. 1024×768)	1× (baseline)	2.64×	4.69×	10.56×	42.22×
5. Relative FLOPs* (Approx.)	1× (baseline)	2.64×	4.69×	10.56×	42.22×
6. Relative Memory BW* (Approx.)	1× (baseline)	2.64×	4.69×	10.56×	42.22×

Going from 1024x768 to 8K has a **~42x** increase in computation, memory traffic → Latency, Power, Memory Bandwidth!!!

Downscaling hurts!



The challenges of high-resolution



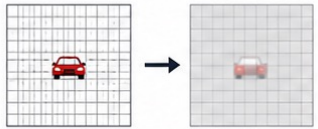
3 What this means on edge devices

- More computation (higher MAC count)
- More energy consumption
- More on-chip / off-chip memory pressure
- Higher latency (end-to-end delay)
- Bigger models or deeper hierarchies may be needed
- Naive full-image processing becomes inefficient

Motivation: process only the most informative image regions

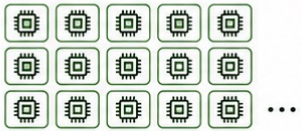
Challenges for Edge Vision & High-Res Images

1 Loss of small-object detail



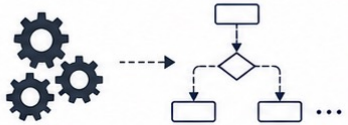
Downsampling removes fine detail

2 Redundant computation



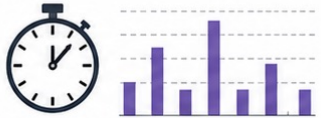
Background and overlapping regions are processed again

3 Extra control overhead



Adaptive strategies add planning and coordination cost

4 Unpredictable latency



Dynamic execution complicates real-time timing

High-Resolution Camera Input

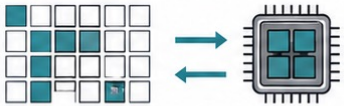
3840 × 2160 (4K)

Edge AI Device / Embedded Board

Processing Pipeline

- Preprocessing
- Inference
- Post-processing
- Results / Actions

5 Hardware inefficiency from irregularity

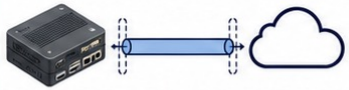


Irregular workloads reduce accelerator efficiency

6 Risk of missed objects

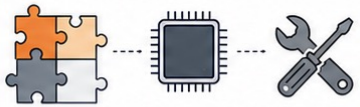
Filtering or coarse stages may skip important regions

7 Communication bottlenecks



Offloading depends on bandwidth and latency

8 Deployment complexity

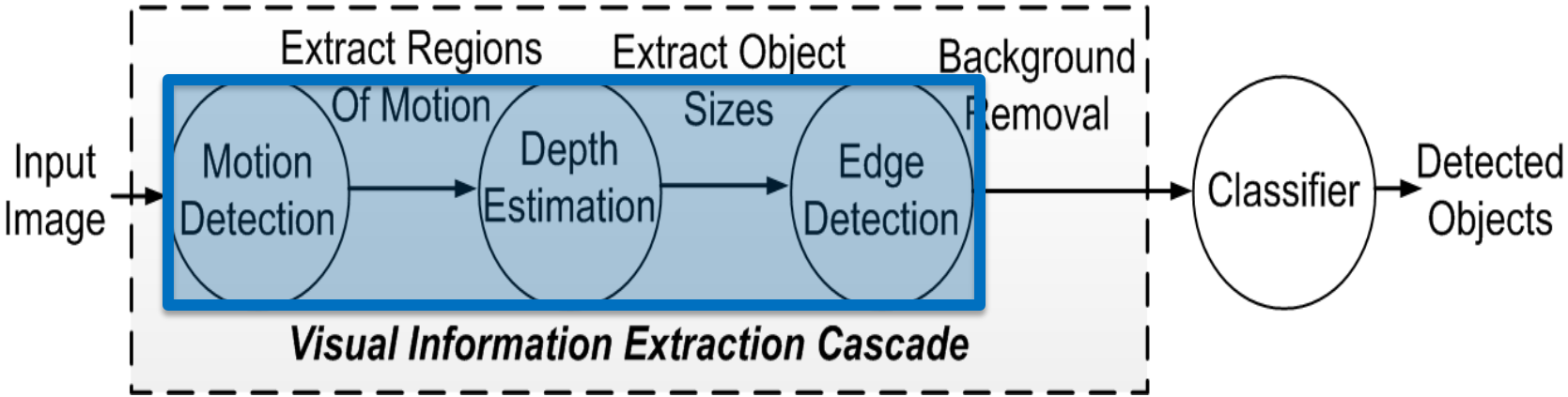


Specialized pipelines are harder to integrate and maintain

What the ANN/CNN detector sees…

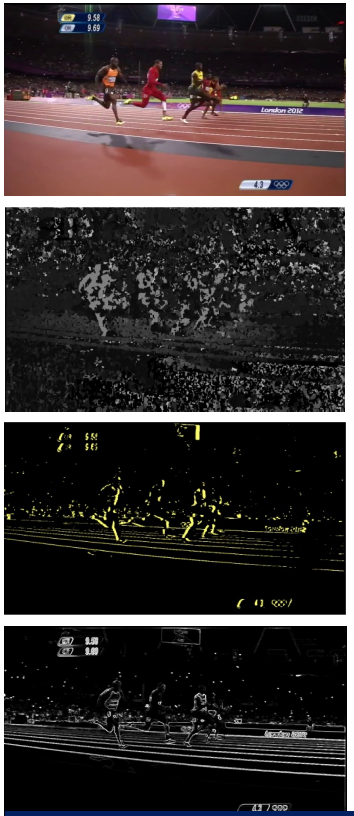
Thankfully, we don't need to search *everywhere!*

Visual Information Extraction Cascade



- Step 1: motion (focus on changes)
- Step 2: depth (focus on object size)
- Step 3: edge (focus only where's a lot of information)
- Step 4: Classify *only what's necessary!*

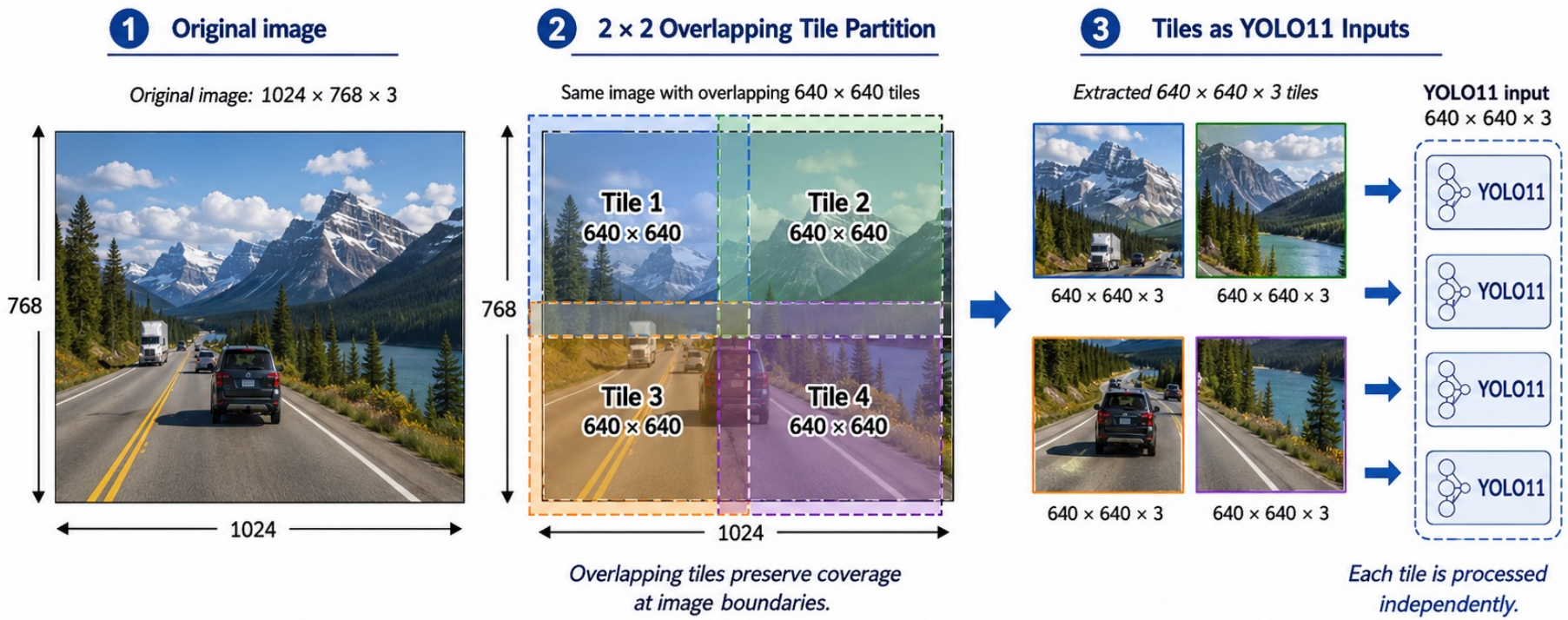
Process only what's necessary!



- A (very) high-speed 100m race!
- Cluttered background
- Various illumination conditions
- 20-76% overall reduction from motion
- 12-30% reduction from depth
- 15-20% reduction from edge
- **~1% of useful data reach the classifier**

Kyrkou, C., Theocharides, T., Bouganis, CS. et al. Embedded hardware-efficient real-time classification with cascade support vector machines. IEEE Transactions on Neural Networks and Learning Systems, Vol 27, Issue 1, pp. 99-112.

Single Shot Detectors (Yolo11) - Partitioning



A 1024 × 768 image can be partitioned into overlapping 640 × 640 tiles before being processed by YOLO11.

Tiling avoids resizing the full image and preserves local detail.

Processing high-resolution images w/ Tiles

1. High-resolution input

High-resolution image, e.g. 2048×2048

Small pedestrian
fine details preserved

Distant car
object still visible

Texture / boundary
sharp local structure

2. Tile partitioning with resolution preserved

Same scene partitioned into a grid of tiles.
Each tile keeps original pixel density and local detail.

Example tiles (at native resolution)

Small CNN (Lightweight NN)

Small CNN (Lightweight NN)

Small CNN (Lightweight NN)

Pedestrian still clear

Car details preserved

Texture remains sharp

Original local resolution preserved

Each tile fits the model input size without shrinking local content.

3. Impact on computer vision

1. Classification

Tile keeps object detail -> easier recognition of small/local content.

Better local recognition

2. Detection

Tiles help small objects remain visible.

Improved small-target detection

3. Segmentation

Local edges and structures remain sharper.

Better boundary precision

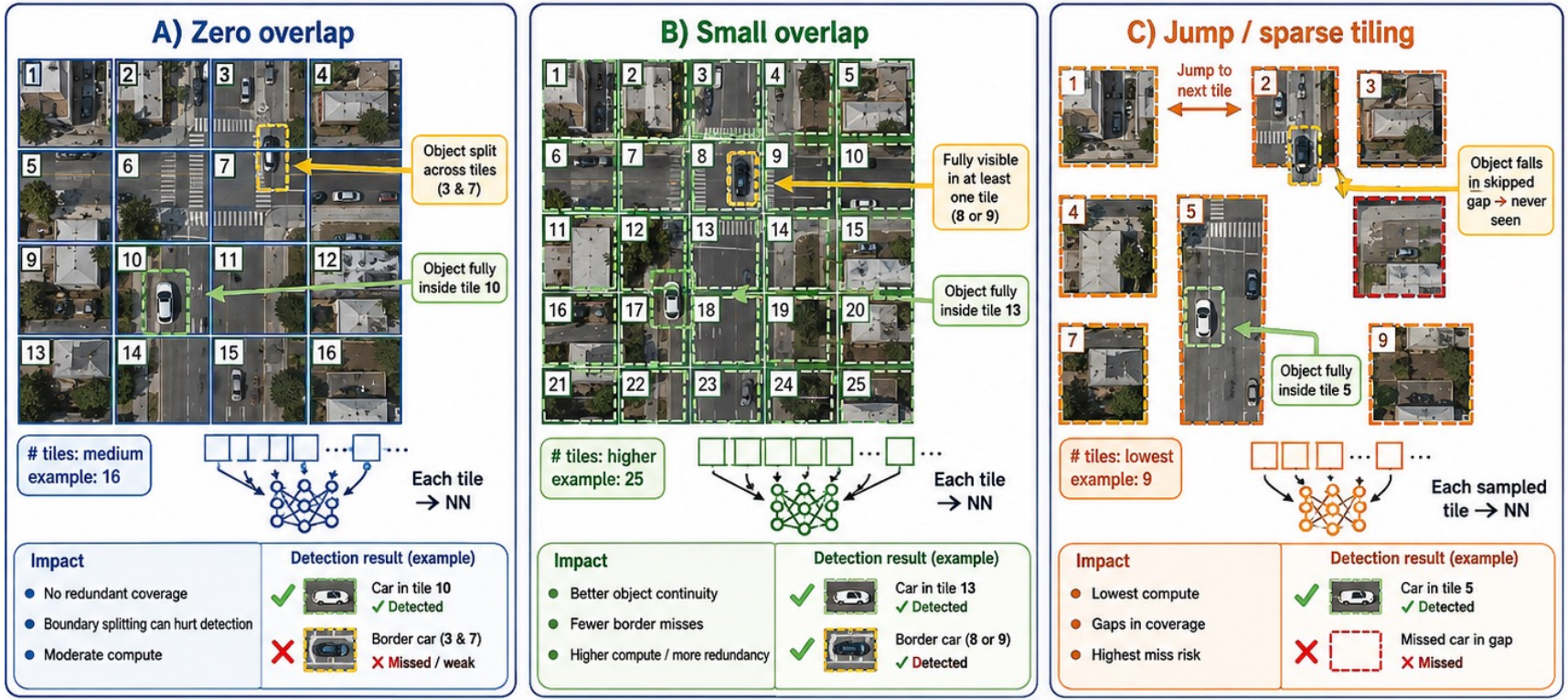
4. Tracking / video analysis

t t+1 t+2

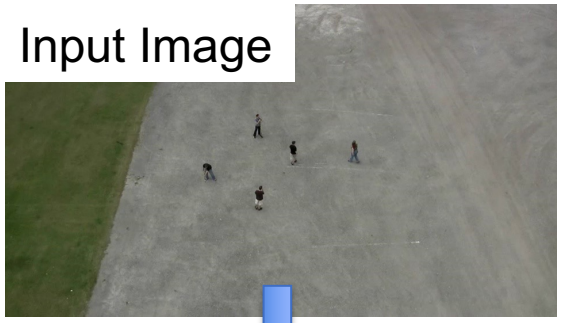
Local regions stay informative frame-to-frame.

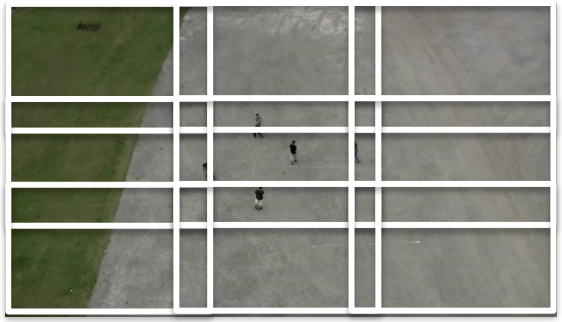
More stable local cues

Tiling Issues – Tile Size, Tile Overlap



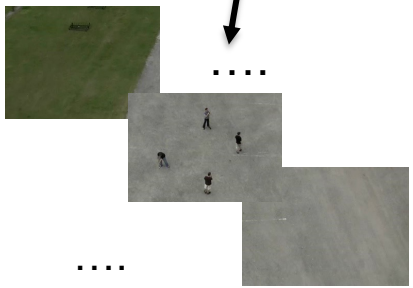
Selective Tile Processing w/ Small CNNs





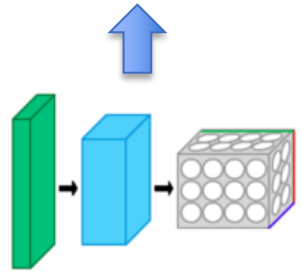
How to process a higher resolution image?

Tiling



Attention Mechanism

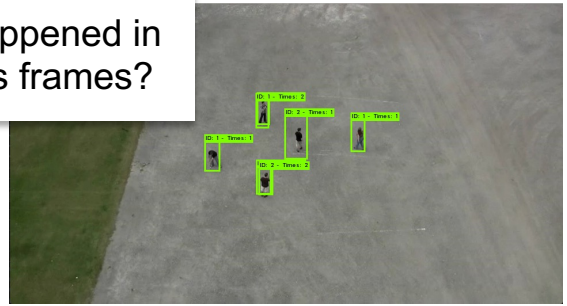
Which image parts to process?



Small Fast CNN

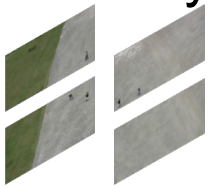
Memory Mechanism

What happened in previous frames?

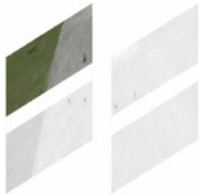


Attention Mechanism

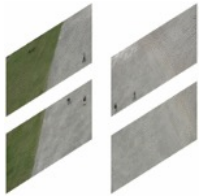
- Select which tiles to be process by the CNN
  - All-Tiles – **TA**



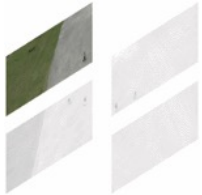
  - Single Tile (Round Robin) – **T1**



  - Tiles with Previously Detected Objects – **TO**

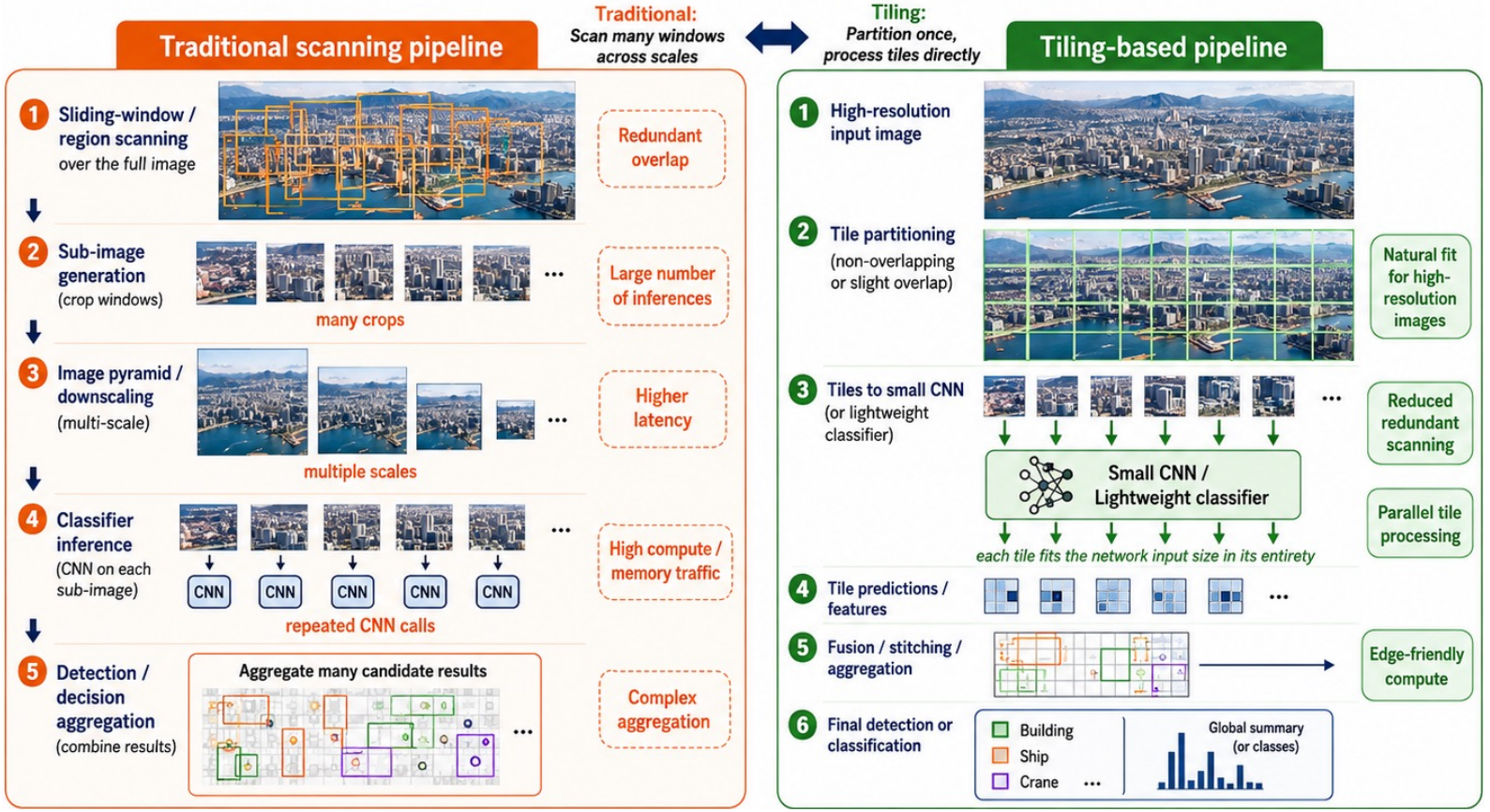


  - Tiles Selection with Memory – **TSM**

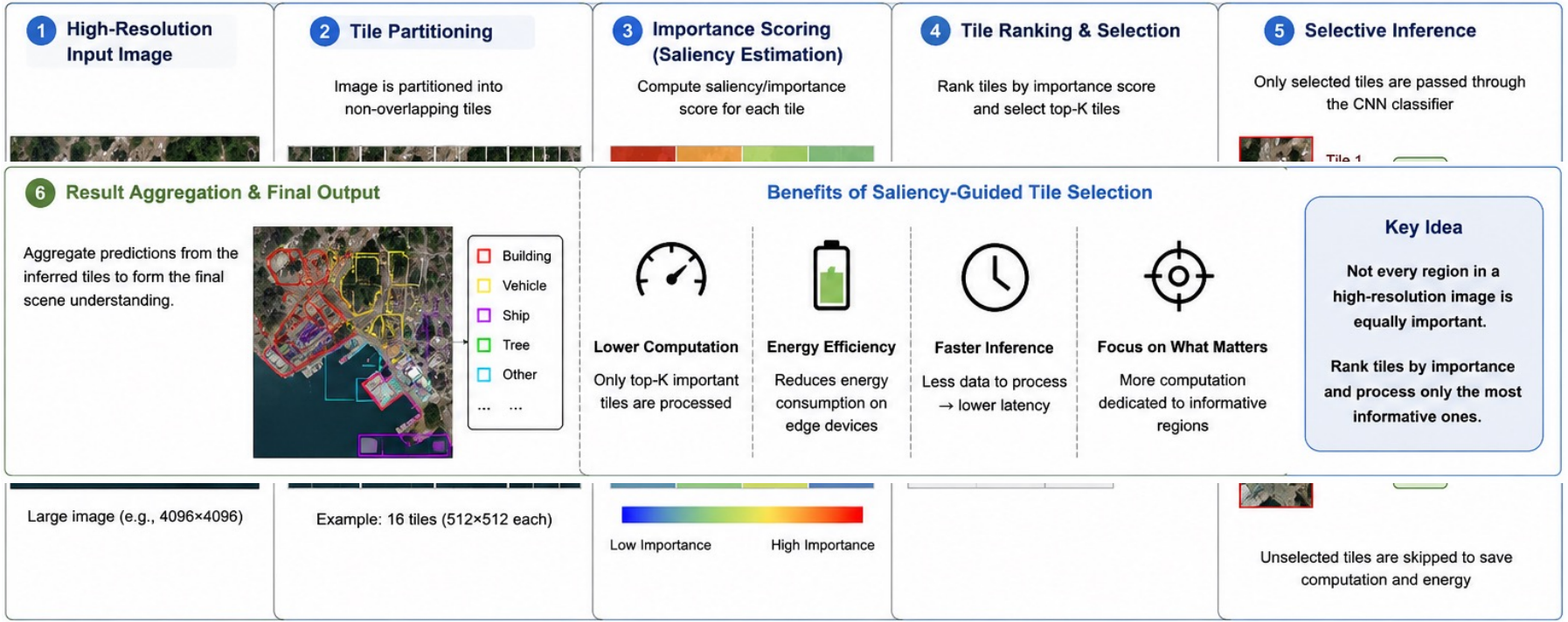


EdgeNet in Action (Car counting for traffic monitoring

Dynamic Inference with Tiling – Proven to work

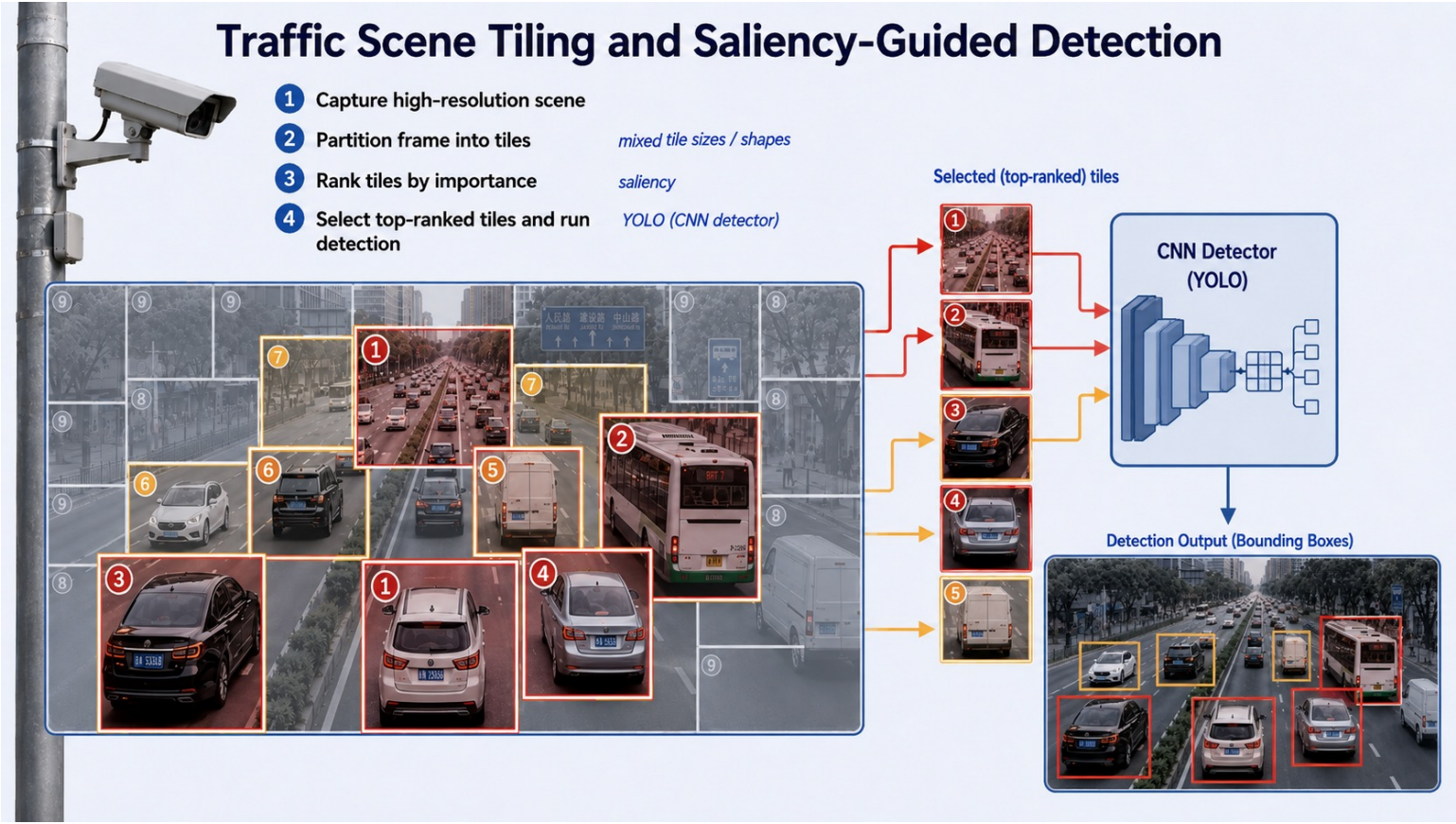


Let there be SIGHT! – Saliency-Driven Tiling

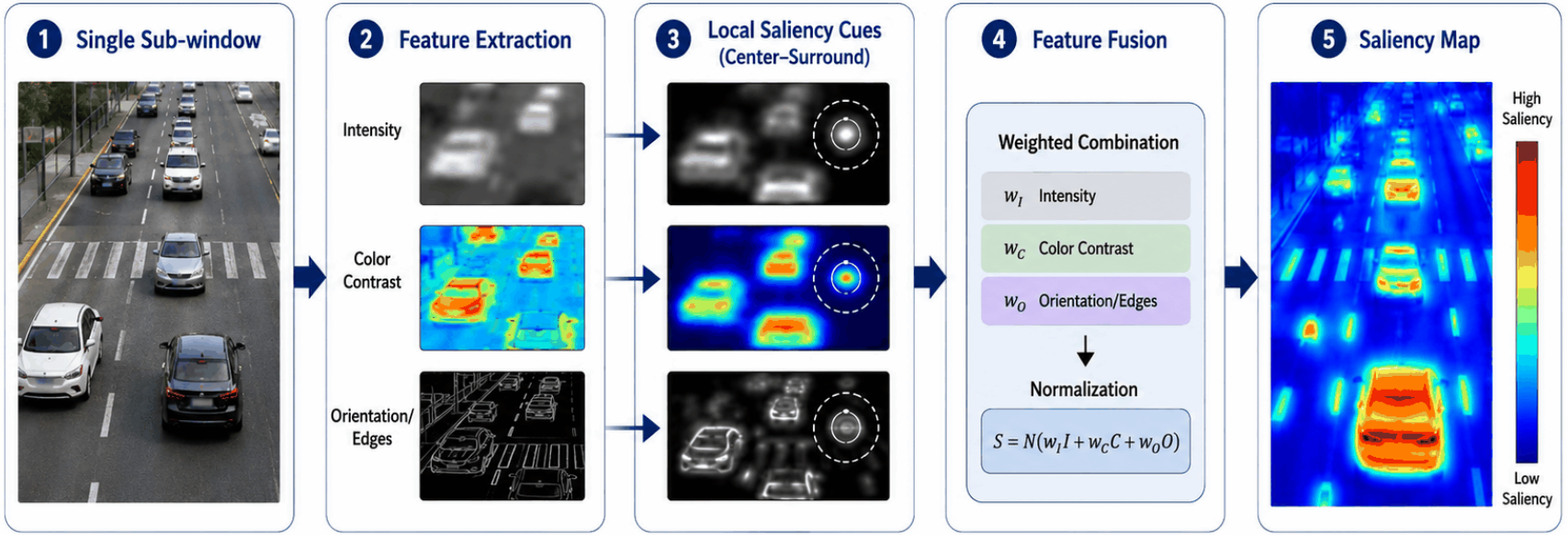


** Current work with Ph.D. Student A.Vasiliou & C. Kyrkou*

In simple terms



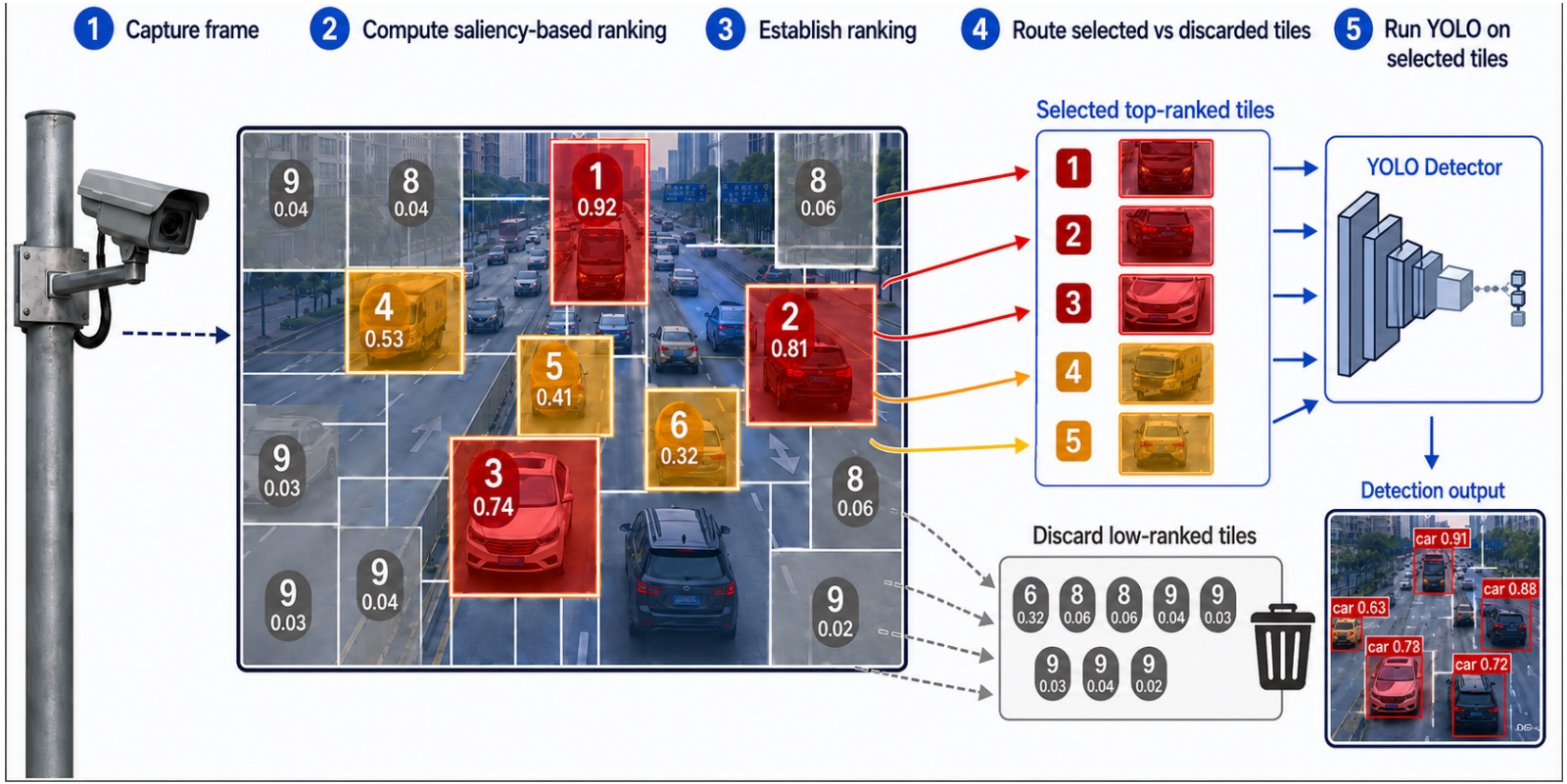
Saliency Computation – Complexity Analysis



Saliency Computation

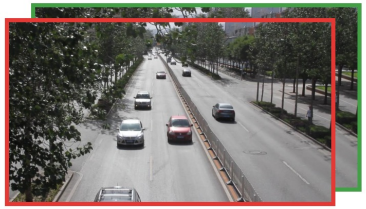
Tradeoffs among saliency methodologies				
Methodology	**Main idea**	**Computational complexity**	**Power / energy**	**Saliency quality / accuracy**
A) Gradient / edge-based saliency	Uses edges, intensity gradients, contrast boundaries	Low	Very low	Low–Medium
B) Optical flow saliency	Uses motion vectors and motion boundaries	Medium–High	Medium–High	High
C) Frame-differencing saliency	Uses absolute change between consecutive frames	Very low or Low	Very low	Low–Medium
D) Temporal phase saliency	Uses temporal frequency/ phase changes and periodic motion patterns	Medium	Medium	Medium–High
E) Learned lightweight saliency model	Uses a small learned network to estimate importance	Medium–High	Medium–High	High

Experimental Approach



Saliency across MCU and Embedded GPU

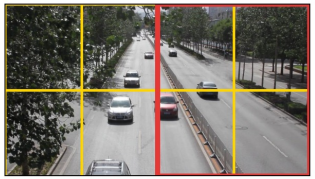
Input frames (**t+1**, **t**)



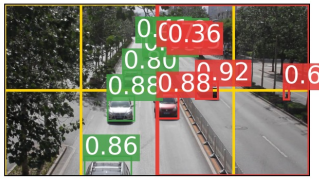
Saliency mask

Tile scores (2×4)

Supertile (2×2)



AG detections (**cached** + **supertile**)



MCU (STM)

1. Frame Differencing

40.0 ms (adopted)

2. Hybrid

147.4 ms

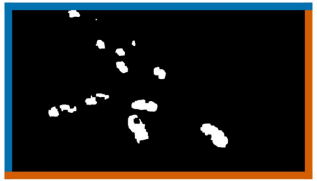
3. FD + k-means

245.1 ms

4. Temporal Phase

1.1 s

5. Optical Flow



1.9 s

eGPU - Jetson

1. DEVA

31.5 ms

2. U^2-Net

116.9 ms

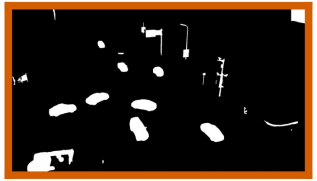
3. PoolNet

1.1 s

4. YOLO-Seg + Flow

1.3 s

5. InSPyReNet

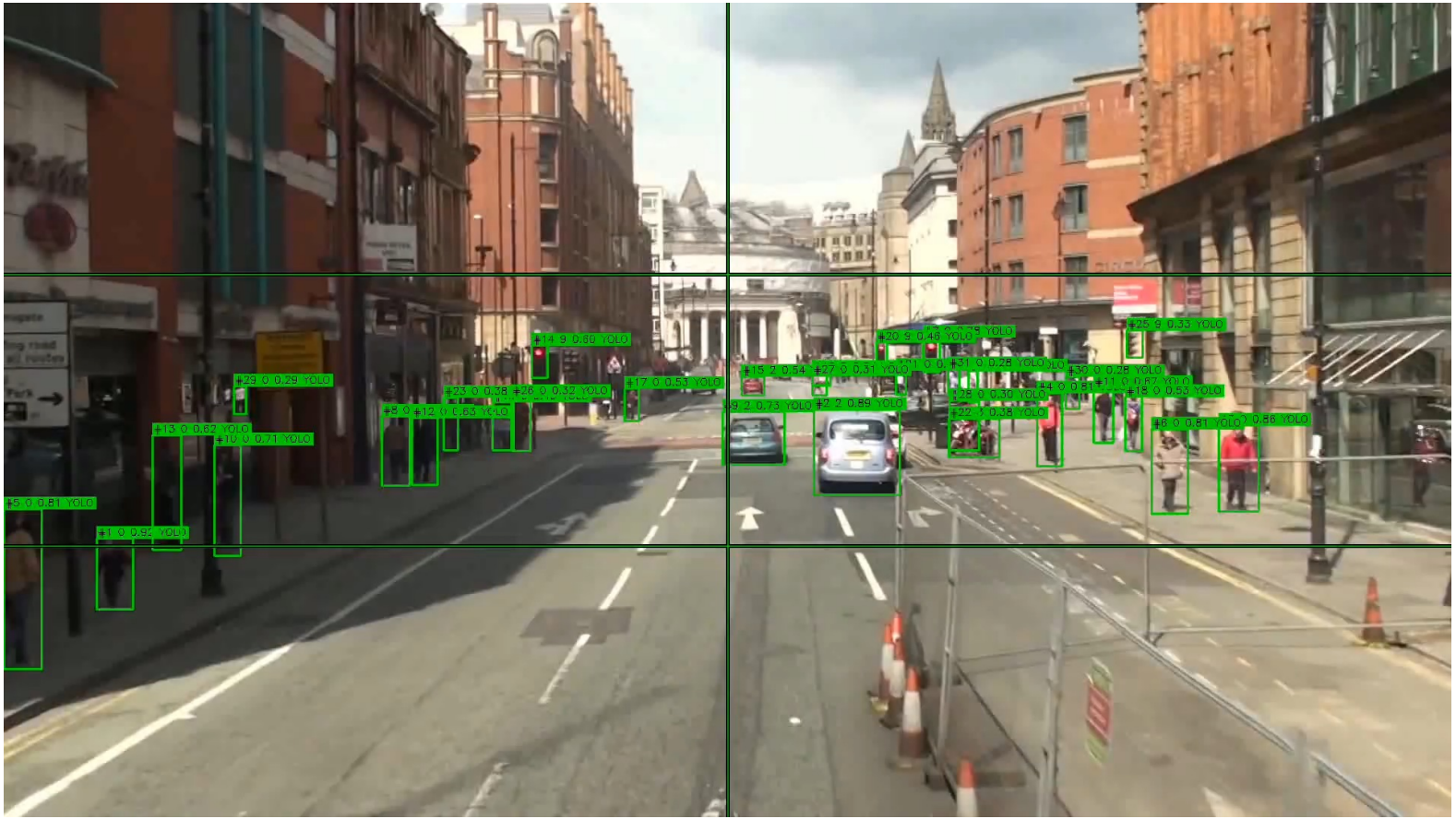


16.0 s

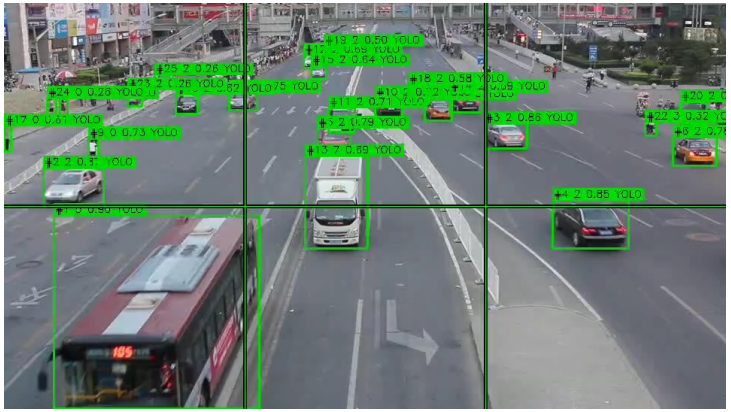
Results w/ Jetson Nano

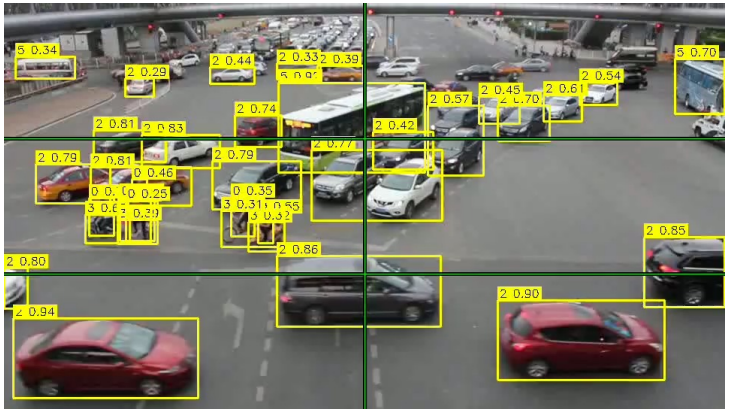
Benchmark	Camera	Model	FPS	mAP	Power (W)	Utilization %	Unique Objects Detected
MOT17-02	Static	Baseline	4.2	0.613	15.41	49.7	52/56
		SIGHT	**8.4**	**0.566**	**9.92**	**33.3**	**50/56**
MOT17-04	Static	Baseline	2.4	0.748	11.34	63.6	81/81
		SIGHT	**6.4**	**0.720**	**9.92**	**33.3**	**81/81**
MOT17-05	Moving	Baseline	16.5	0.850	11.12	51.2	120/124
		SIGHT	**18.6**	**0.661**	**8.03**	**35.9**	**114/124**
MOT17-09	Static	Baseline	2.4	0.925	11.40	69.3	26/26
		SIGHT	**6.6**	**0.862**	**10.15**	**36.3**	**26/26**
MOT17-10	Moving	Baseline	2.2	0.709	10.98	60.7	57/57
		SIGHT	**6.9**	**0.658**	**9.31**	**26.1**	**57/57**
MOT17-11	Moving	Baseline	2.1	0.741	10.60	52.1	58/73
		SIGHT	**4.6**	**0.654**	**11.55**	**46.0**	**59/73**
MOT17-13	Moving	Baseline	2.2	0.598	10.62	43.4	104/108
		SIGHT	**6.7**	**0.540**	**9.31**	**26.1**	**103/108**

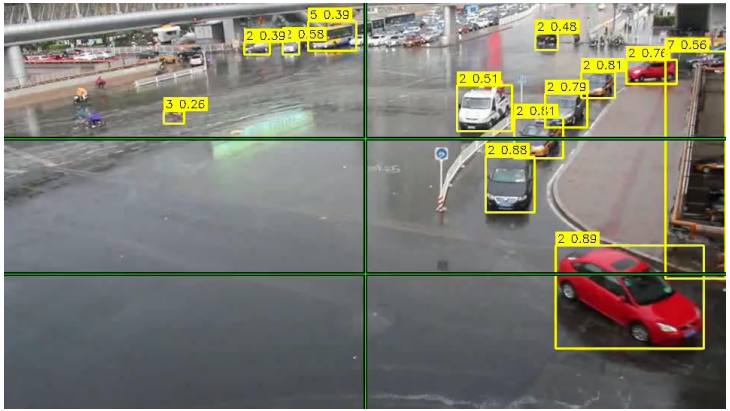
Demo (running on Jetson Nano)



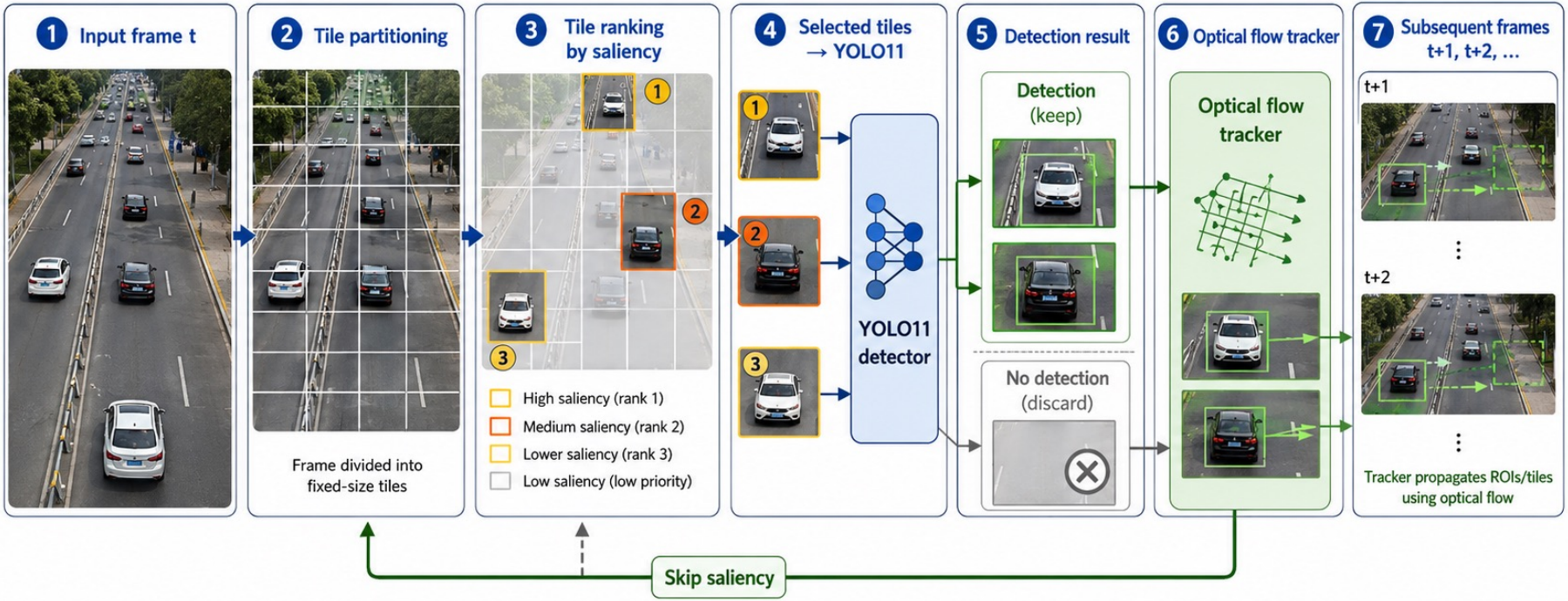
Encouraging Progress!



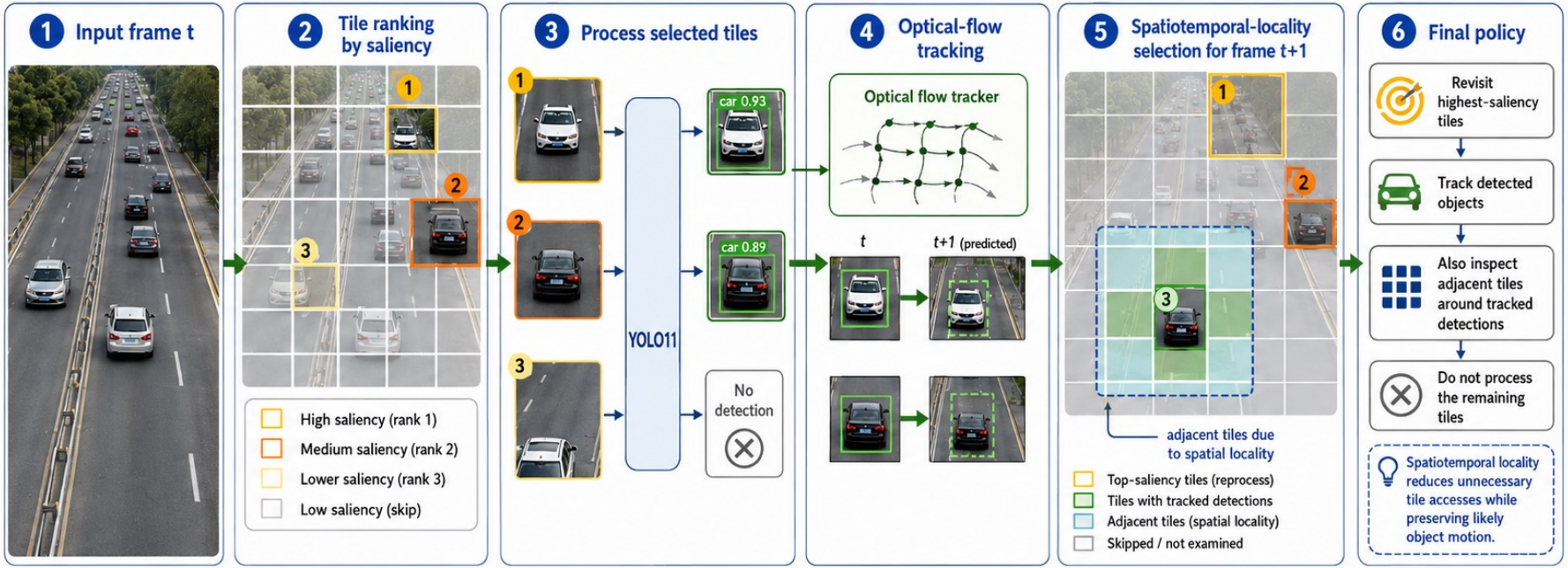




Tracking (human-like approach!)



Spatiotemporal Locality (Cache-Like)



Active processing / tracking

Skipped / not processed

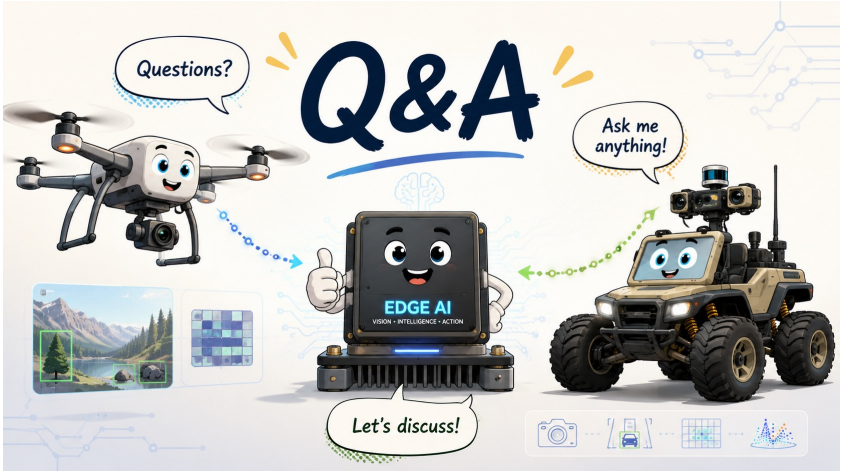
Top-saliency tiles (process again)

Tracked detection tile (t+1)

Adjacent tiles selected due to spatial locality

Skipped / not examined

Thank you!



Acknowledgement

Parts of this work were supported by the European Union's Horizon Europe research and innovation programme under grant agreement No 101168067 (GuardAI) and the from the European Union's Horizon 2020 research and innovation programme under grant agreement No 739551 (KIOS CoE) & from the Government of the Republic of Cyprus through the Cyprus Deputy Ministry of Research, Innovation and Digital Policy.

Funded by: