Deploying Vision Foundation AI Models on the Edge. The SAM2 Experience

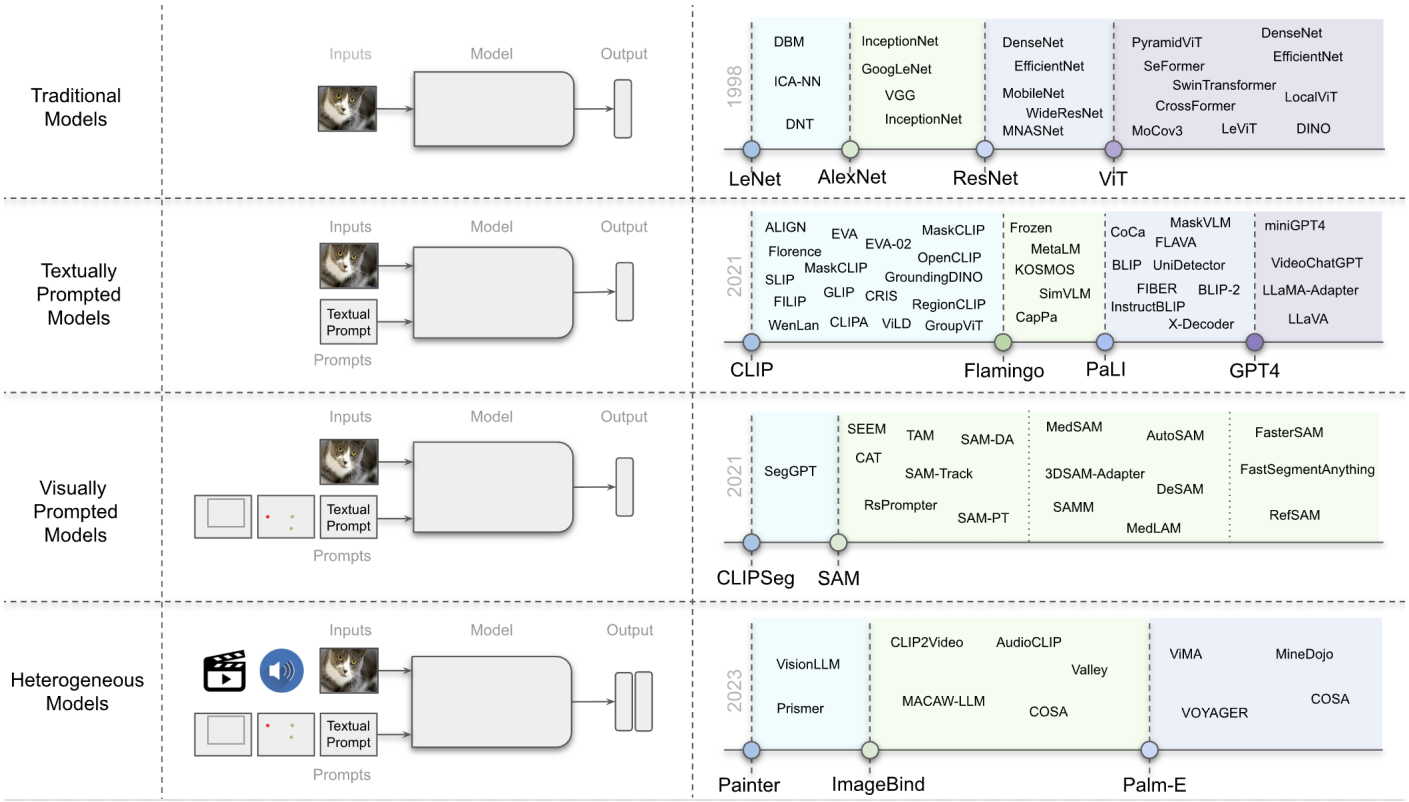
Zheshuo Lin[1], Ruben Tous[1] and Beatriz Otero[1]

[1]Universitat Politecnica de Catalunya

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Índex

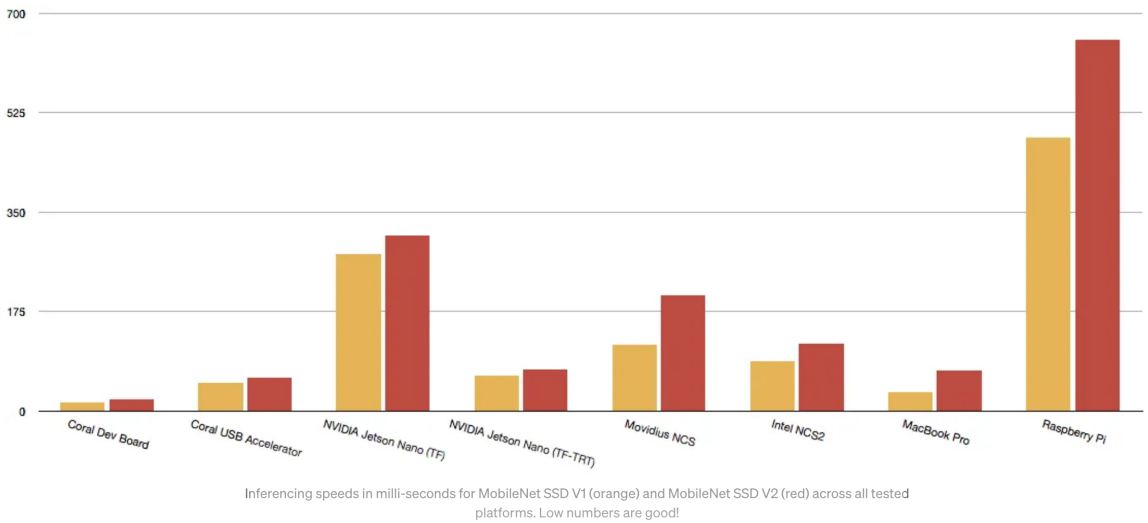
Vision Foundation Models



[Awaisand et al., 2023]

AI workloads from HPC/cloud to the edge

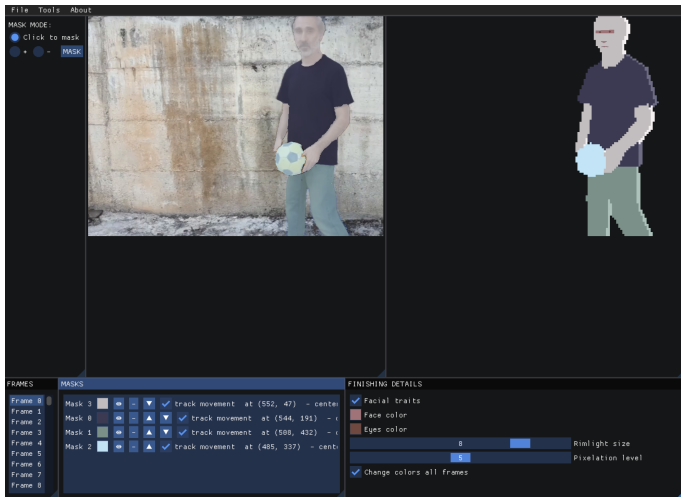
- Pros: real-time, privacy, network, cost
- Energy efficiency? latency?
- Challenges: CPU/GPU, memory, python



[Allan, 2019]

Problem Statement

- Adaptation of SAM2 for execution on edge AI environments.
- Translating the model to C++ using ONNX Runtime.
- Performance evaluation.
- Use case: Rotoscoped animation editor (mid-range workstation).



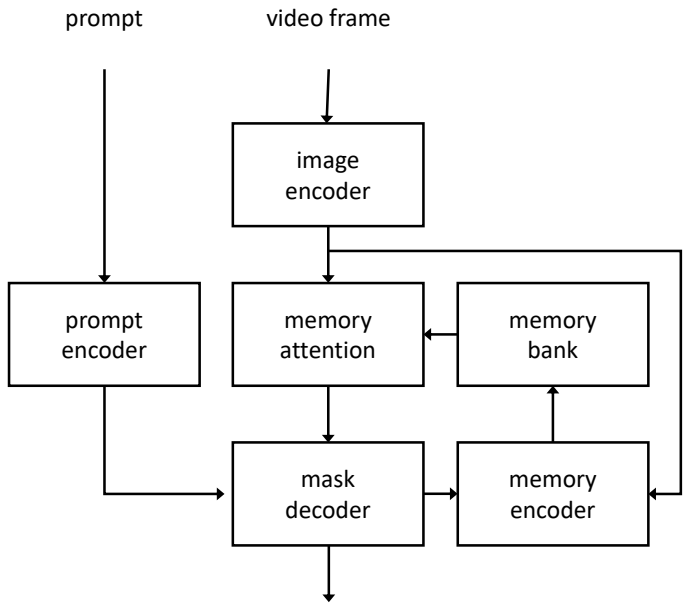
SAM2 overview

SA-V Dataset

- 642.6 K masklets
- 35.5 M masks
- 50.9 K videos
- 196.0 hours

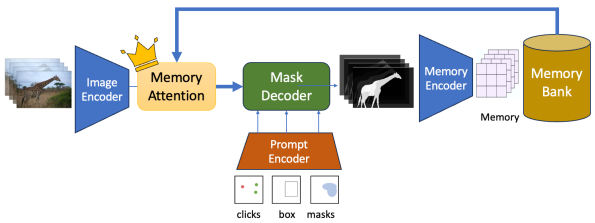
Method	$\mathcal{J}\&\mathcal{F}$					$\mathcal{G}$
	MOSE val	DAVIS 2017 val	LVOS val	SA-V val	SA-V test	YTVOS 2019 val
STCN (Cheng et al., 2021a)	52.5	85.4	-	61.0	62.5	82.7
SwinB-AOT (Yang et al., 2021b)	59.4	85.4	-	51.1	50.3	84.5
SwinB-DeAOT (Yang & Yang, 2022)	59.9	86.2	-	61.4	61.8	86.1
RDE (Li et al., 2022a)	46.8	84.2	-	51.8	53.9	81.9
XMem (Cheng & Schwing, 2022)	59.6	86.0	-	60.1	62.3	85.6
SimVOS-B (Wu et al., 2023b)	-	88.0	-	44.2	44.1	84.2
JointFormer (Zhang et al., 2023b)	-	90.1	-	-	-	87.4
ISVOS (Wang et al., 2022)	-	88.2	-	-	-	86.3
DEVA (Cheng et al., 2023b)	66.0	87.0	55.9	55.4	56.2	85.4
Cutie-base (Cheng et al., 2023a)	69.9	87.9	66.0	60.7	62.7	87.0
Cutie-base+ (Cheng et al., 2023a)	71.7	88.1	-	61.3	62.8	87.5
SAM 2 (Hiera-B+)	76.6	90.2	**78.0**	76.8	77.0	88.6
SAM 2 (Hiera-L)	**77.9**	**90.7**	**78.0**	**77.9**	**78.4**	**89.3**

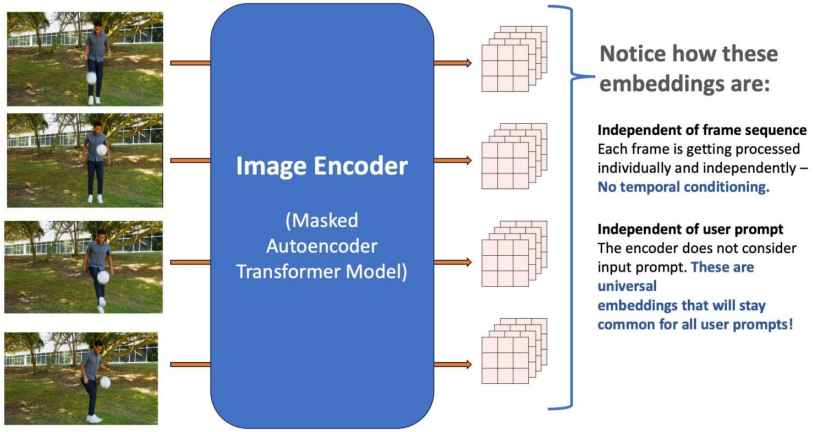
Table 6 VOS comparison to prior work. SAM 2 performs well in accuracy ($\mathcal{J}\&\mathcal{F}$, $\mathcal{G}$) for video segmentation based on first-frame ground-truth mask prompts. SAM 2 performs significantly better on SA-V val/test.

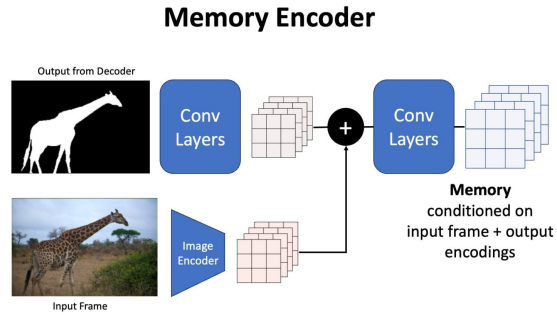


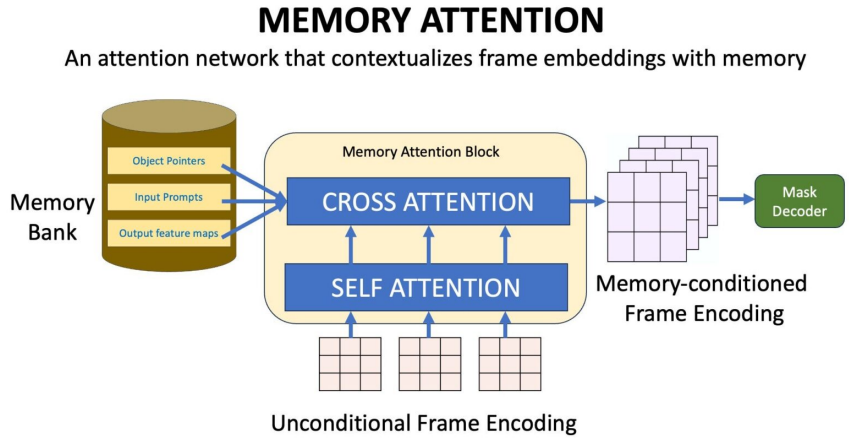
[Ravi et al., 2024]

SAM2 overview









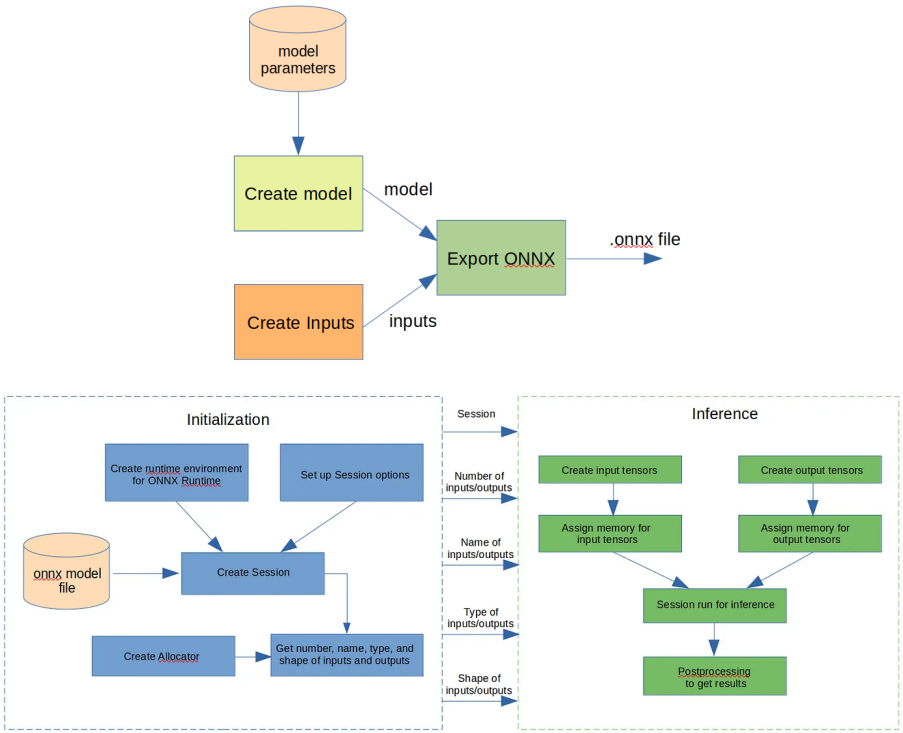
[Biswas, 2024]

SAM2 challenges

- Efficiency bottleneck in the memory module (uses past frame). Large number of tokens in cross-attention.
- EfficientTAM (non-hierarchical encoder and optimized memory module) [Xiong and et al., 2024]
- Efficient Frame Pruning (EFP) (optimizes the memory bank by retaining only the most informative frames). [Liu et al., 2024]
- Unaddressed:
  - Python's high memory use
  - Runtime overhead
  - Dependency issues
  - Release as a self-contained application

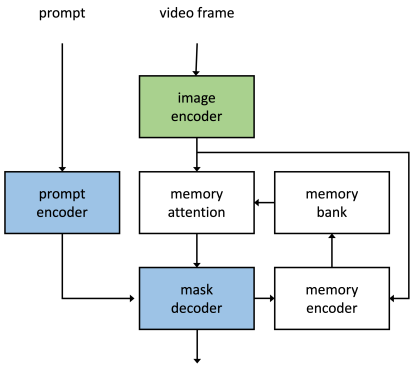
SAM2: Migration to C++

- Release for desktop, integration with edge SDKs.
- ONNX for intermediate representation.
- TODO: video inference.



Exporting the models to ONNX

- Original Python/Pytorch code analysis.
- Export process in Python, from slighly modified original SAM2 code.
- Two ONNX models: Image encoder and image decoder (prompt encoder + mask decoder)
- Some parts (e.g. image preprocessing) not exported (need to be implemented in C++)
- Problems with tuples in ONNX.



Example export code for the image decoder (Python)

```python
def export_image_decoder(model,onnx_path):  1 usage  new *
    point_coords = torch.randn(1,2,2).cpu()
    point_labels = torch.randn(1,2).cpu()
    frame_size = torch.tensor( data: [1024,1024],dtype=torch.int64)
    image_embed = torch.randn(1,256,64,64).cpu()
    high_res_feats_0 = torch.randn(1,32,256,256).cpu()
    high_res_feats_1 = torch.randn(1,64,128,128).cpu()

    out = model(
        point_coords=point_coords,
        point_labels=point_labels,
        frame_size=frame_size,
        image_embed=image_embed,
        high_res_feats_0=high_res_feats_0,
        high_res_feats_1=high_res_feats_1,
    )
    input_name = [
        "point_coords",
        "point_labels",
        "frame_size",
        "image_embed",
        "high_res_features1",
        "high_res_features2",
    ]
    output_name = ["pred_mask", "iou_predictions", "low_res_masks"]
    dynamic_axes = {
        "point_coords":{0: "X",1:"Y"},
        "point_labels": {0: "B"}
    }
```

```python
torch.onnx.export(
    model,
    args: (point_coords,point_labels,frame_size,image_embed,high_res_feats_0,high_res_feats_1),
    onnx_path+"image_decoder.onnx",
    export_params=True,
    opset_version=17,
    do_constant_folding=True,
    input_names= input_name,
    output_names=output_name,
    dynamic_axes = dynamic_axes
)
original_model = onnx.load(onnx_path+"image_decoder.onnx")
simplified_model, check = simplify(original_model)
onnx.save(simplified_model, onnx_path+"image_decoder.onnx")
onnx_model = onnx.load(onnx_path+"image_decoder.onnx")
onnx.checker.check_model(onnx_model)
print("image_decoder.onnx model is valid!")
```

Importing and using the ONNX models from C++

ONNX models (encoder and decoder) import in C++

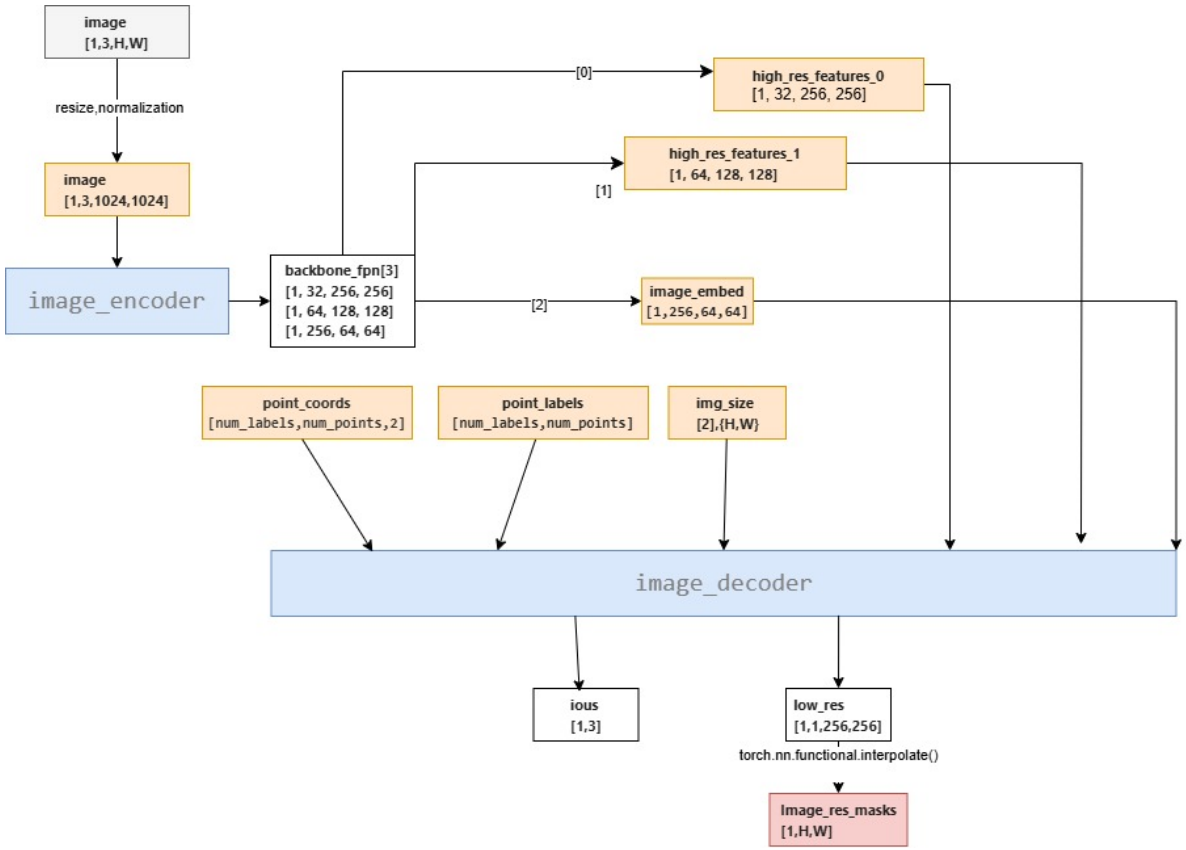
```cpp
#include <iostream>
#include <list>
#include <sstream>
#include <locale>
#include <codecvt>
#include <string>
#include <windows.h>
#include <psapi.h>
#include <chrono>

#include <opencv2/core.hpp>
#include <opencv2/opencv.hpp>
#include <onnxruntime_cxx_api.h>
#include <windows.h>
#include <fstream>
// TODO: Haga referencia aquí a los encabezados adicionales que el programa requiere.
```

```cpp
class Sam2 {


public:
    Sam2();
    ~Sam2();
    void Inicialitzacio(const std::string& encoderPath, const std::string& decoderPath);
    void SetImatge(const cv::Mat& imatge);
    cv::Mat Predict(std::vector<float>& points_coords, std::vector<float>& points_labels, cv::Size frame
    std::vector<float> PreparaPunts(const std::list<cv::Point>& points, const cv::Size imatge_size);
    cv::Mat PreparaImatge(const cv::Mat& imatge); // preparar imatge per SetImage
private:
    Ort::Env env{ ORT_LOGGING_LEVEL_WARNING, "Sam2" };
    Ort::Session* image_encoder = nullptr;
    Ort::Session* image_decoder = nullptr;
    std::vector<Ort::Value> output_image_encoder; //son image_embed, high_res_features 0 i 1
    Ort::MemoryInfo memory_info = Ort::MemoryInfo::CreateCpu(OrtArenaAllocator, OrtMemTypeDefault);
};
```

Image encoder preprocessing and invocation in C++

```cpp
cv::Mat Sam2::PreparaImatge(const cv::Mat& imatge) {
    auto input_shape = this->image_encoder->GetInputTypeInfo(0).GetTensorTypeAndShapeInfo().GetShape();
    cv::Mat imatge_copy = imatge.clone();
    cv::cvtColor(imatge_copy, imatge_copy, cv::COLOR_BGR2RGB);
    cv::resize(imatge_copy, imatge_copy, cv::Size((int)input_shape[3], (int)input_shape[2]));
    cv::Mat imatge_normalizat;
    imatge_copy.convertTo(imatge_normalizat, CV_32FC3, 1.0 / 255.0);
    cv::Scalar mean(0.485, 0.456, 0.406);
    cv::Scalar stdDev(0.229, 0.224, 0.225);
    cv::subtract(imatge_normalizat, mean, imatge_normalizat);
    cv::divide(imatge_normalizat, stdDev, imatge_normalizat);
    std::vector<cv::Mat> channels(3);
    cv::split(imatge_normalizat, channels);
    cv::vconcat(channels, imatge_normalizat);

    return imatge_normalizat;
}
void Sam2::SetImatge(const cv::Mat& imatge) {
    auto input_shape = this->image_encoder->GetInputTypeInfo(0).GetTensorTypeAndShapeInfo().GetShape();
    cv::Mat imatge_preparat = imatge.clone();
    try {
        Ort::Value input_imatge = Ort::Value::CreateTensor<float>(memory_info,
            imatge_preparat.ptr<float>(),
            imatge_preparat.total(),
            input_shape.data(),
            input_shape.size()
        );
        Ort::RunOptions runOptions;
        // obtenir noms de inputs i outputs

        std::vector<vector<string>> names = PrintInputOutputNames(*image_encoder);
        std::vector<const char*> input_names;
        for (const auto& str : names[0]) {

            input_names.push_back(str.c_str());
        }
        std::vector<const char*> output_names;
        for (const auto& str : names[1]) {
            output_names.push_back(str.c_str());
        }
        output_image_encoder = image_encoder->Run(runOptions,
            input_names.data(),
            &input_imatge,
            input_names.size(),
            output_names.data(),
            output_names.size()
        );

    }
    catch (Ort::Exception& e) {
        std::cout << e.what() << std::endl;
    }
}
```

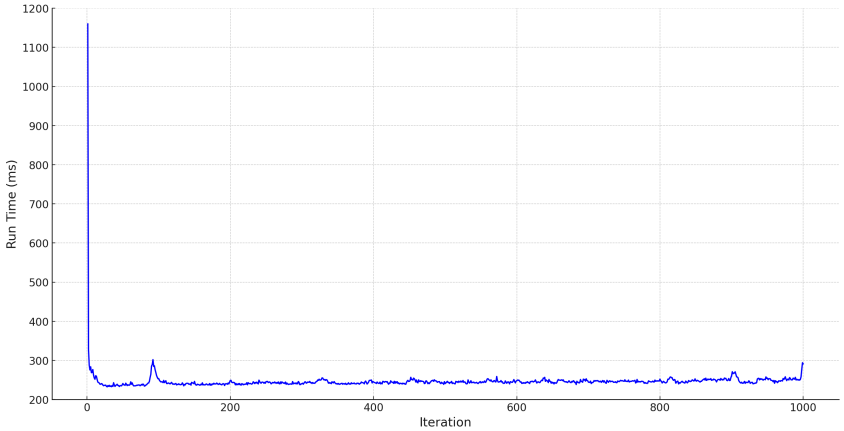
Diagram of the resulting C++ implementation



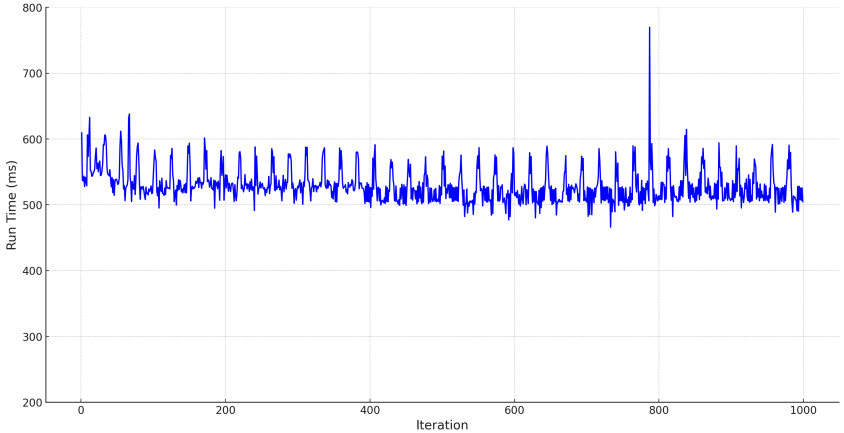
Experimental Setup

- Laptop
  - Alder Lake i7-12650H integrated SoC
  - 16GB of DDR4 RAM (8GB*2 at 3200MHz), a 1TB NVMe PCIe Gen4x4 SSD without DRAM,
  - NVIDIA RTX 4050 graphics card with 6GB of GDDR6 memory, 120 tensor cores, and a 96-bit memory interface.
- Cloud
  - Tesla T4 GPU with 16GB of GDDR6 memory, 320 tensor cores, and a 256-bit memory interface.
- 1000 consecutive executions (varying images and prompts)
- each experiment repeated 5 times (mean reported)

Execution time



Execution Time on a Local Machine



Execution Time in a Cloud Environment

Execution time

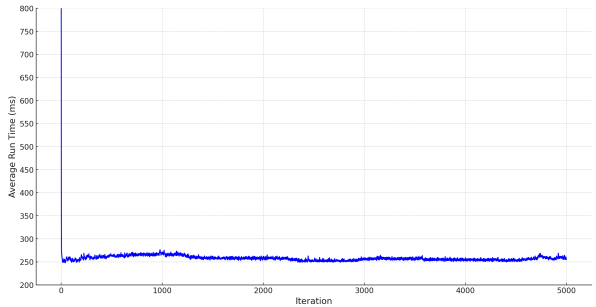


Image Encoder (Local Machine)

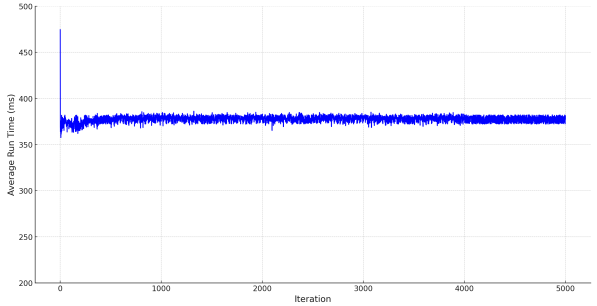


Image Encoder (Cloud Environment)

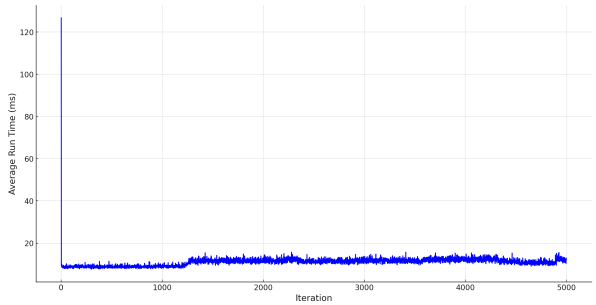


Image Decoder (Local Machine)

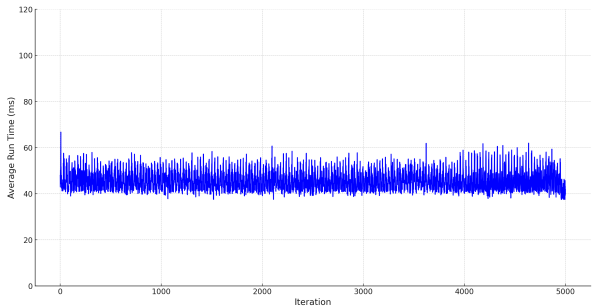
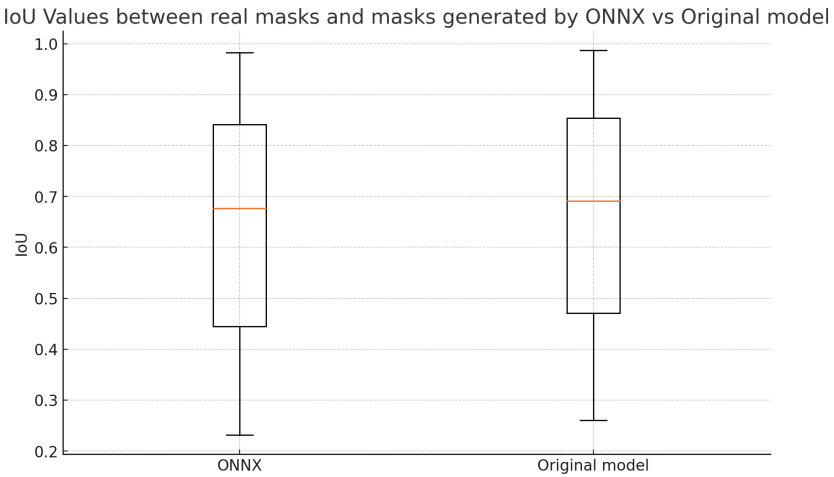


Image Decoder (Cloud Environment)

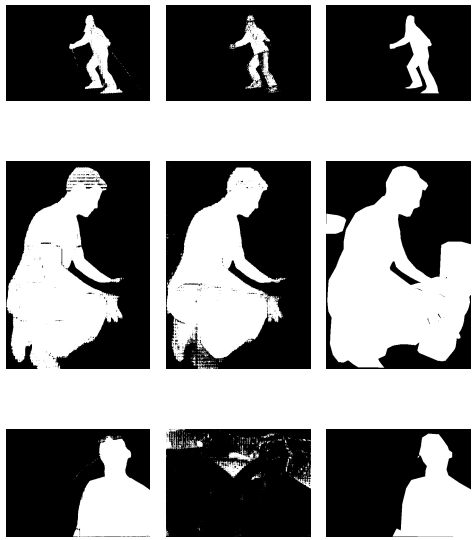
Accuracy Evaluation

- To verify that the resulting ONNX/C++ version behaves consistently with the original model
- segmentation accuracy using the COCO dataset (segmentation masks)
- Points randomly sampled from groundtruth segmentation masks.

Accuracy Evaluation



Jaccard index between the actual mask and the predicted mask.



(a) **(b)** **(c)**

(**a**) Our C++/ONNX implementation; (**b**) original SAM2 implementation; (**c**) ground truth.

Conclusions

- Translated SAM2 (image inference) to C++
- Not easy but feasible.
- ONNX is key (and enables hardware abstraction), but the devil is in the details.
- Reduced latency: 50% improvement in latency with respect to cloud.
- Other advantages:
  - No inference costs, no scalability problems
  - User experience no dependent on network state, servers load, etc.
  - Release as a self-contained application (no python, deps, etc.).
  - Data privacy

Future Work

- Video inference (hard)
- Optimizations
- Alternative (for desktop): Node.js
- Test on other hardware setups
- Problem: model upgrades, new models, etc.

Bibliography


Allan, A. (2019).
Benchmarking edge computing.
https://aallan.medium.com/benchmarking-edge-computing-ce3f13942245.


Awaisand, M. et al. (2023).
Foundational models defining a new era in vision: A survey and outlook.
arXiv preprint arXiv:2307.13721.


Biswas, A. (2024).
Segment anything 2: What is the secret sauce?
https://towardsdatascience.com/segment-anything-2-what-is-the-secret-sauce-a-deep-learners-guide-1c43dd07a6f8/.


Liu, H., Zhang, E., Wu, J., Hong, M., and Jin, Y. (2024).
Surgical SAM 2: Real-time segment anything in surgical video by efficient frame pruning.
CoRR, abs/2408.07931.


Ravi, N., Gabeur, V., Hu, Y.-T., Hu, R., Ryali, C., Ma, T., Khedr, H., Rädle, R., Rolland, C., Gustafson, L., et al. (2024).
Sam 2: Segment anything in images and videos.
arXiv preprint arXiv:2408.00714.

Questions?