## Segment Anything (SAM) Project Documentation

Generated: 2025-09-24

Audience: Project Manager, Software/ML Engineer

### 1. Executive Summary

The project integrates Meta AI's Segment Anything Model (SAM) to enable preemptable image segmentation. It includes:

* Python package code for building and running SAM models, utilities, and example notebooks
* A simple React + TypeScript web demo that runs the exported ONNX mask decoder in the browser
* CLI scripts and utilities for automatic mask generation and ONNX export
* Example output artifacts under `output\_dir/`

Key outcomes:

* + Generate accurate object masks from input prompts (points/boxes) or automatically for all objects
  + Export and run the lightweight mask decoder in ONNX, including in-browser inference
  + Provide examples, demo UI, and scripts for quick adoption

### 2. Scope & Objectives

* + - Integrate core SAM components for promptable segmentation
    - Provide automatic mask generation capability
    - Support ONNX export for portable inference; demonstrate a browser-based demo
    - Supply runnable notebooks for exploratory use and reproducibility

Out of scope:

* + - * Large-scale training or dataset tooling beyond examples
      * Production deployment infrastructure

### 3. Repository Structure (high level)

* + - * + `segment\_anything/` Python library (modeling, predictor API, utilities)
        + `scripts/` CLI utilities (automatic mask generation, ONNX export)
        + `notebooks/` Example Jupyter notebooks (predictor, automatic mask generator, ONNX)
        + `segment-anything/demo/` React + TypeScript web demo for ONNX in-browser inference
        + `segment-anything/assets/` Images and diagrams
        + `output\_dir/` Example generated outputs (masks and overlays)

Relevant top-level docs:

`segment-anything/README.md` Installation, usage, ONNX export, checkpoints

`segment-anything/demo/README.md` Running the demo and structure

### 4. Architecture Overview

Conceptual flow:

1) Input image and optional prompts (points, boxes, masks)

2) Image encoded by vision transformer backbone

3) Prompts encoded by prompt encoder

4) Mask decoder outputs segmentation masks

5) Post-processing to produce binary masks and quality metrics

Components:

Modeling (`segment\_anything/modeling/`):

`sam.py` Model assembly (image encoder, prompt encoder, mask decoder)

`image\_encoder.py`, `prompt\_encoder.py`, `mask\_decoder.py`, `transformer.py`, `common.py`

Predictor API (`segment\_anything/predictor.py`):

`SamPredictor`: high-level interface to set images and generate masks from prompts

Automatic Mask Generator (`segment\_anything/automatic\_mask\_generator.py`):

`SamAutomaticMaskGenerator`: generates multiple masks per image automatically

Utilities (`segment\_anything/utils/`): transforms, ONNX helpers, AMG utilities

CLI scripts (`scripts/`):

`amg.py` command-line automatic mask generation

`export\_onnx\_model.py` export mask decoder to ONNX

Web demo (`segment-anything/demo/`):

Loads an image embedding and runs ONNX mask decoder in browser (WASM, workers, SIMD)

### 5. Installation & Setup

Prerequisites:

Python >= 3.8, PyTorch >= 1.7, TorchVision >= 0.8 (CUDA recommended)

Node.js >= 16 for running the demo

Install the Python package (editable):

cd segment-anything

pip install -e .

Optional dependencies for notebooks and ONNX:

pip install opencv-python pycocotools matplotlib onnxruntime onnx jupyter

### 6. Getting Started (Python)

Load a checkpoint and run with prompts:

from segment\_anything import SamPredictor, sam\_model\_registry

sam = sam\_model\_registry["<model\_type>"](checkpoint="<path/to/checkpoint>")

predictor = SamPredictor(sam)

predictor.set\_image(<your\_image>)

masks, scores, logits = predictor.predict(<input\_prompts>)

Automatic mask generation:

from segment\_anything import SamAutomaticMaskGenerator, sam\_model\_registry

sam = sam\_model\_registry["<model\_type>"](checkpoint="<path/to/checkpoint>")

mask\_generator = SamAutomaticMaskGenerator(sam)

masks = mask\_generator.generate(<your\_image>)

Command-line automatic mask generation:

python scripts/amg.py --checkpoint <path/to/checkpoint> \

--model-type <model\_type> --input <image\_or\_folder> --output <path/to/output>

Model types and checkpoints (from README):

`default` or `vit\_h`: ViT-H

`vit\_l`: ViT-L

`vit\_b`: ViT-B

### 7. ONNX Export and Web Demo

Export ONNX mask decoder:

python scripts/export\_onnx\_model.py --checkpoint <path/to/checkpoint> \

--model-type <model\_type> --output <path/to/output>

Run the browser demo:

cd segment-anything/demo

npm install --g yarn

yarn && yarn start

Then open http://localhost:8081/.

Demo inputs to update in src/App.tsx:

* `IMAGE\_PATH`, `IMAGE\_EMBEDDING`, `MODEL\_DIR`

Notes on multithreading:

* Dev server headers enable `SharedArrayBuffer` via cross-origin isolation in `configs/webpack/dev.js`.

### 8. Data Flow Details

* Preprocessing: images are resized/normalized; longest side typically 1024 for the demo
* Prompt encoding: points, boxes, or masks converted into embeddings
* Image encoding: ViT backbone extracts image features
* Mask decoding: transformer-based head predicts masks conditioned on prompts
* Post-processing: thresholding, stability scoring, and quality metrics

### 9. Outputs and Artifacts

* `output\_dir/mask/\*.png` – Per-instance or per-region binary masks
* `output\_dir/mask\_overlay.png` – Composite overlay for quick visualization

### 10. Quality, Performance, and Risks

* Performance depends on backbone size and hardware (CPU vs GPU)
* Large images increase memory/time; tiling or downscaling may be necessary
* ONNX CPU inference is slower than GPU; in-browser uses WASM + SIMD where available
* Deterministic outputs for fixed prompts and seeds; beware of preprocessing differences

### 11. Roadmap Suggestions

* Provide Docker images for repeatable demos and notebooks
* Add CLI for interactive promptable segmentation (points/boxes) beyond AMG
* Integrate CI with linting, unit tests, and sample inference tests
* Add quantitative evaluation scripts (e.g., IoU, stability) and benchmarks

### 12. References

* Main README: `segment-anything/README.md`
* Demo README: `segment-anything/demo/README.md`
* Notebooks: `notebooks/\*`
* Paper, demo, dataset links provided in `segment-anything/README.md`
  + - Mask decoding: ransformer-based head predicts masks conditioned on prompts
    - Post-processing: thresholding, stability scoring, and quality metrics

### 9. Outputs and Artifacts

* + - * `output\_dir/mask/\*.png` Per-instance or per-region binary masks
      * `output\_dir/mask\_overlay.png` Composite overlay for quick visualization

### 10. Quality, Performance, and Risks

* + - * + Performance depends on backbone size and hardware (CPU vs GPU)
        + Large images increase memory/time; tiling or downscaling may be necessary
        + ONNX CPU inference is slower than GPU; in-browser uses WASM + SIMD where available
        + Deterministic outputs for fixed prompts and seeds; beware of preprocessing differences

### 11. Roadmap Suggestions

Provide Docker images for repeatable demos and notebooks

Add CLI for interactive promptable segmentation (points/boxes) beyond AMG

Integrate CI with linting, unit tests, and sample inference tests

Add quantitative evaluation scripts (e.g., IoU, stability) and benchmarks

### 12. References

Main README: `segment-anything/README.md`

Demo README: `segment-anything/demo/README.md`

Notebooks: `notebooks/\*`

Paper, demo, dataset links provided in `segment-anything/README.md`