SVO2-SAM3 Analyzer

Software Solution Plan

Multi-Camera SVO2 Processing with SAM 3 Object Detection

Stereo RGB + Depth + IMU | KITTI-Style Annotations

Version:	**1.0**
Date:	**January 2026**
Author:	**Mauricio Mesones**
Classification:	**Internal**

Executive Summary

This document outlines the software architecture for a web-based application that processes Stereolabs ZED 2i camera recordings (SVO2 files) using Meta's Segment Anything Model 3 (SAM 3) for comprehensive object detection and analysis. The solution extracts and correlates stereo RGB images (left/right), depth maps, and IMU sensor data, maintaining full traceability from raw SVO2 frames through to KITTI-style annotation outputs.

100% LOCAL DEPLOYMENT

All models, inference, and data processing run entirely on the local workstation.
No cloud services, external APIs, or internet connection required for operation.

Target Hardware:

- Intel Core Ultra 9 processor
- 128GB DDR5 RAM
- NVIDIA GeForce RTX 5090 (32GB VRAM)
- Ubuntu 22.04.5 LTS (64-bit, GNOME 42.9)

SVO2 File Contents

Each SVO2 file contains synchronized multi-modal sensor data that must be extracted and correlated:

Data Stream	Description
Left RGB	Primary camera image (used for SAM 3 inference)
Right RGB	Secondary stereo camera image (for disparity/3D reconstruction)
Depth Map	32-bit float depth per pixel in millimeters (Neural mode)
Point Cloud	XYZRGBA per pixel for 3D spatial data
IMU - Accelerometer	Linear acceleration (m/s^2) in X, Y, Z axes
IMU - Gyroscope	Angular velocity (rad/s) in X, Y, Z axes
IMU - Orientation	Quaternion (w, x, y, z) from sensor fusion
Timestamps	Nanosecond-precision timestamps for frame/IMU sync

System Overview

The SVO-SAM3 Analyzer is a full-stack web application consisting of three primary layers: a React-based frontend for user interaction, a FastAPI backend for orchestration and processing, and a processing engine that handles SVO extraction and SAM 3 inference.

High-Level Architecture (Local)

PRESENTATION LAYER (localhost:3000)
React + TypeScript + TailwindCSS File Browser | Object Selector | Parameter Config | Results Dashboard
API LAYER (localhost:8000)
FastAPI + WebSockets + Celery (Local Redis) REST Endpoints | Real-time Progress | Async Processing
PROCESSING LAYER (Local GPU: RTX 5090)
ZED SDK 5.1 + SAM 3 (Local Weights) + Open3D + ByteTrack SVO2 Extraction | Local Inference | 3D Reconstruction | Tracking
DATA LAYER (Local Storage)
SQLite + Redis (Local) + Local Filesystem Job State | Detection Results | SVO2 Files | Extracted Frames | KITTI Output

Web UI Components

1. Directory Explorer & File Selector

The file browser provides a familiar tree-view interface for navigating the server's file system and selecting SVO files for processing.

- Tree-view directory navigation with expandable folders and breadcrumb path display
- Multi-select capability with checkboxes for batch processing multiple SVO files
- File filtering (.svo2 extension) with search functionality
- File metadata preview (size, duration, resolution, FPS, IMU rate, recording date)
- Drag-and-drop support for reordering selected files
- Configurable root directories (restricted to authorized paths)

2. Object Class Selector

SAM 3 supports text-prompt segmentation for any describable concept. The selector provides both preset categories and custom prompt entry.

Category	Available Objects
Vehicles	Haul truck, Excavator, Loader, Light vehicle, Water truck, Dozer
Personnel	Person, Worker with PPE, Worker without PPE
Equipment	GET (Ground Engaging Tool), Bucket, Tire, Cable, Hose
Infrastructure	Berm, Road edge, Stockpile, Building, Power line
Safety Items	Cone, Barrier, Sign, Light tower
Custom	User-defined text prompts (free-form entry)

- Toggle switches for each object class with select all/none options
- Custom prompt text field for detecting any describable object
- Save/load preset configurations for common detection scenarios
- Confidence threshold slider per class (0.0 - 1.0)

3. SAM 3 Configuration Panel

Parameter	Default	Description
Detection Threshold	**0.5**	Minimum confidence score for valid detections
Mask Threshold	**0.5**	Binary mask threshold for segmentation output
Frame Sampling	**1**	Process every Nth frame (1 = all frames)
Max Detections	**100**	Maximum objects per frame
NMS IoU Threshold	**0.7**	Non-maximum suppression overlap threshold

Model Variant	**sam3-base**	Model size: tiny, small, base, large
Precision Mode	**FP16**	Inference precision: FP32, FP16, BF16
Batch Size	**8**	Frames processed per GPU batch
Enable Tracking	**Yes**	Enable ByteTrack for object persistence
Export 3D Data	**Yes**	Include 3D bounding boxes in output

4. Processing Controls

- Start Processing button (enabled when files selected and at least one object class enabled)
- Real-time progress bar with estimated time remaining
- Per-file progress indicators for batch processing
- Live detection count and GPU utilization display
- Pause/Resume and Cancel controls
- Processing log viewer with error highlighting

5. Results Dashboard

Upon completion, the dashboard presents comprehensive analysis results with interactive visualizations.

- Summary statistics: total objects, unique tracks, processing time, frames analyzed
- Object count breakdown by class (bar chart visualization)
- Timeline view showing object presence over video duration
- Interactive data table with sorting, filtering, and search
- Frame viewer with overlay visualization of detections
- 3D point cloud viewer for spatial analysis (using Three.js)
- Export options: KITTI dataset (.zip), JSON, CSV, COCO format, annotated video

Processing Pipeline

Stage 1: SVO2 Extraction & Frame Registry

The ZED SDK extracts all sensor streams from SVO2 files and registers each frame with a unique identifier for full traceability through the annotation pipeline.

- Open SVO2 file with Neural depth mode for AI-enhanced depth estimation
- Extract stereo RGB pair (LEFT and RIGHT views) as PNG or numpy arrays
- Extract depth map (32-bit float, millimeters) aligned to left camera
- Extract point cloud (XYZRGBA) for 3D reconstruction
- Query IMU data: accelerometer (m/s^2), gyroscope (rad/s), orientation quaternion
- Record nanosecond timestamps for frame-IMU synchronization
- Generate unique frame_id (format: {svo2_hash}_{frame_number}) for correlation

Frame Registry Schema

Each extracted frame is registered with complete metadata for annotation traceability:

Field	Type	Description
frame_id	string	Unique identifier: {svo2_hash}_{frame_num}

svo2_file	string	Source SVO2 filename
svo2_frame_idx	integer	Frame index within SVO2 file
timestamp_ns	integer	Nanosecond timestamp from camera
timestamp_iso	string	ISO 8601 formatted timestamp
left_rgb_path	string	Path to left camera RGB image
right_rgb_path	string	Path to right camera RGB image
depth_path	string	Path to depth map file (.npy or .png)
pointcloud_path	string	Path to point cloud file (.ply)
imu_accel	float[3]	Accelerometer [x, y, z] in m/s^2
imu_gyro	float[3]	Gyroscope [x, y, z] in rad/s
imu_orientation	float[4]	Quaternion [w, x, y, z]
camera_intrinsics	object	{fx, fy, cx, cy, k1, k2, p1, p2}
camera_extrinsics	object	Stereo baseline and rotation

Stage 2: SAM 3 Local Inference

SAM 3 runs entirely on the local RTX 5090 GPU. Model weights are loaded from disk once at startup and cached in VRAM for the duration of processing.

- Load SAM 3 weights from ~/.cache/jebi/models/sam3/ (no network calls)
- Apply torch.compile optimization for RTX 5090 Blackwell architecture
- Batch frames according to configured batch size (8-16 for 32GB VRAM)
- Execute local GPU inference with text prompts for all enabled object classes
- Post-process on GPU: apply confidence threshold, NMS, generate instance masks
- Output: per-frame list of detections with masks, boxes, scores, and class labels

Stage 3: 3D Reconstruction

- Project 2D segmentation masks onto depth data using camera intrinsics
- Generate per-object point clouds using Open3D
- Compute oriented bounding boxes (center, dimensions, rotation)
- Calculate physical measurements: volume, surface area, distance from camera

Stage 4: Object Tracking

- ByteTrack associates detections across frames using IoU and appearance features
- Assign persistent track IDs to maintain object identity
- Handle occlusion and re-identification with configurable track buffer
- Generate trajectory data for each tracked object

Technology Stack (Local Deployment)

All components run locally on the workstation. No external services or cloud dependencies are required after initial setup.

Layer	Technology	Purpose
Frontend	**React 18 + TypeScript**	Component framework (local dev server)
Frontend	**TailwindCSS**	Styling with Jebi brand colors
Frontend	**React Query**	Server state management
Frontend	**Three.js**	3D point cloud visualization
Backend	**FastAPI**	REST API (localhost:8000)
Backend	**WebSockets**	Real-time progress updates
Backend	**Celery + Redis**	Local async task queue
Backend	**SQLAlchemy**	ORM for local database
AI/ML	**PyTorch 2.7+ (local)**	Deep learning framework
AI/ML	**SAM 3 weights (local)**	Downloaded model checkpoint (~2.4GB)
AI/ML	**CUDA 12.6 + cuDNN**	Local GPU acceleration
Processing	**ZED SDK 5.1 (pyzed)**	SVO2 file processing
Processing	**Open3D**	3D geometry processing
Processing	**ByteTrack**	Multi-object tracking
Data	**SQLite / PostgreSQL**	Local persistent storage
Data	**Redis (local)**	Caching and task broker
Data	**Local filesystem**	SVO2 files and outputs

Local Model & Environment Setup

One-time setup downloads all required model weights and dependencies. After initial setup, the system operates fully offline.

SAM 3 Model Weights

Model Variant	Size	VRAM Required	Recommended For
sam3-tiny	~400 MB	4 GB	Quick testing
sam3-small	~900 MB	8 GB	Balanced performance
sam3-base	~1.8 GB	12 GB	Production (default)
sam3-large	~2.4 GB	16 GB	Maximum accuracy

Local Storage Paths

- Model weights: ~/.cache/jebi/models/sam3/ (downloaded once, ~2.4GB)
- ZED SDK: /usr/local/zed/ (system installation)
- Processing cache: ~/.cache/jebi/svo2_cache/ (extracted frames)
- Output directory: ~/jebi_outputs/ (KITTI datasets)
- Database: ~/.local/share/jebi/db.sqlite (job history, settings)
- Redis data: /var/lib/redis/ (task queue state)

GPU Configuration

RTX 5090 optimization settings for maximum local inference throughput:

Setting	Configuration
CUDA Version	12.6+ (local installation)
cuDNN Version	9.x (local installation)
PyTorch Build	torch==2.7+ with CUDA 12.6 support
Precision Mode	BF16 (bfloat16) for RTX 5090 Blackwell
torch.compile	mode='reduce-overhead' for inference
Memory Format	channels_last for CNN optimization
Batch Size	8-16 frames (adjustable based on VRAM)
TF32 Math	Enabled for matrix operations

Offline Operation Guarantee

- No HuggingFace Hub calls after initial model download
- No telemetry or analytics transmitted
- All inference runs on local GPU (RTX 5090)
- Web UI served from localhost (127.0.0.1:3000)
- API backend on localhost (127.0.0.1:8000)

- Can operate in air-gapped environments after setup

API Endpoints

File Management

Method	Endpoint	Description
GET	/api/files/browse	List directory contents
GET	/api/files/metadata/{path}	Get SVO file metadata
POST	/api/files/validate	Validate SVO file integrity

Processing Jobs

Method	Endpoint	Description
POST	/api/jobs/create	Create new processing job
GET	/api/jobs/{id}	Get job status and progress
POST	/api/jobs/{id}/start	Start processing
POST	/api/jobs/{id}/pause	Pause processing
POST	/api/jobs/{id}/cancel	Cancel processing
GET	/api/jobs/{id}/results	Get processing results
WS	/ws/jobs/{id}/progress	Real-time progress stream

Configuration

Method	Endpoint	Description
GET	/api/config/object-classes	List available object classes
GET	/api/config/presets	List saved presets
POST	/api/config/presets	Save new preset
GET	/api/config/model-info	Get SAM 3 model details

Export

Method	Endpoint	Description
GET	/api/export/{id}/kitti	Export KITTI-format dataset (.zip)
GET	/api/export/{id}/json	Export full results as JSON
GET	/api/export/{id}/csv	Export summary as CSV
GET	/api/export/{id}/coco	Export COCO-format annotations
POST	/api/export/{id}/video	Generate annotated video overlay

Output: KITTI-Style Annotations

The primary output follows the KITTI 3D Object Detection benchmark format, extended with SVO2 correlation metadata. This enables direct use with autonomous driving toolchains while maintaining full traceability to source data.

KITTI Label File Format

Each frame generates a .txt label file with one line per detected object:

Column	Index	Description
type	0	Object class (e.g., HaulTruck, Person, Excavator)
truncated	1	Truncation level: 0.0 (fully visible) to 1.0 (truncated)
occluded	2	Occlusion state: 0=visible, 1=partial, 2=mostly, 3=unknown
alpha	3	Observation angle relative to camera center [-π, π]
bbox	4-7	2D bounding box: left, top, right, bottom (pixels)
dimensions	8-10	3D dimensions: height, width, length (meters)
location	11-13	3D center location: x, y, z in camera coords (meters)
rotation_y	14	Rotation around Y-axis in camera coordinates [-π, π]
score	15	Detection confidence score (0.0 to 1.0)

Extended Metadata File

Each frame also generates a .json metadata file linking annotations to SVO2 source data:

Field	Type	Description
frame_id	string	Unique frame identifier for SVO2 correlation
svo2_source	object	{file, frame_idx, timestamp_ns}
stereo_images	object	{left_path, right_path}
depth_source	object	{path, format, unit: 'mm'}
imu_data	object	{accel, gyro, orientation, temp}
camera_calibration	object	Full intrinsics and extrinsics
detections	array	Extended detection data per object
detections[].track_id	integer	Persistent tracking ID across frames
detections[].mask_rle	string	Run-length encoded segmentation mask
detections[].pointcloud	object	{count, centroid, obb_rotation}

Output Directory Structure

The KITTI-style output follows a standardized directory layout:

- image_2/ — Left camera RGB images (PNG, numbered 000000.png, 000001.png, ...)
- image_3/ — Right camera RGB images (PNG, same numbering)
- depth/ — Depth maps (16-bit PNG or .npy float32)
- velodyne/ — Point clouds in .bin format (KITTI compatible)
- label_2/ — KITTI format label files (.txt)
- metadata/ — Extended JSON metadata with SVO2 correlation
- calib/ — Camera calibration files (P0, P1, P2, P3, R0_rect, Tr_velo_cam)
- oxts/ — IMU/GPS data in KITTI oxts format
- frame_registry.json — Master index mapping frame numbers to SVO2 source

Implementation Phases

Phase	Deliverables
1	Project setup, FastAPI skeleton, React scaffold, database schema
2	File browser UI, directory API, SVO2 metadata extraction
3	SAM 3 local model integration, inference pipeline, GPU optimization
4	Object selector UI, configuration API, preset management
5	3D reconstruction, ByteTrack integration, tracking pipeline
6	Results dashboard, visualizations, KITTI export functionality
7	WebSocket progress, error handling, performance tuning
8	Testing, documentation, local deployment packaging

Next Steps

1. Review and approve this software solution plan
2. Set up RTX 5090 workstation with Ubuntu 22.04.5 LTS (64-bit, GNOME 42.9)
3. Install NVIDIA drivers, CUDA 12.6, and cuDNN 9.x locally
4. Install ZED SDK 5.1 (local license) and verify camera connectivity
5. Download SAM 3 model weights (one-time, ~2.4GB) for local storage
6. Define initial set of object classes for mining operations
7. Provide sample SVO2 files for development and testing
8. Verify air-gapped operation capability (disconnect network, run inference)