.github/workflows	ci: run package-check ...	4 months ago
bindings/python/visi...	python: fixes for runni...	7 months ago
depend	project: update llama....	4 months ago
docs	build: clean up some ...	last year
include/visp	depth-anything: fix mi...	7 months ago
models	cli: add configurable ...	5 months ago
scripts	build: add VISP_PACKA...	4 months ago
src	build: add VISP_PACKA...	4 months ago
tests	build: add VISP_PACKA...	4 months ago
.clang-format	project: add shared lib...	last year
.gitignore	docs: update readme, ...	last year
.gitmodules	project: switch submo...	6 months ago
CMakeLists.txt	build: add VISP_PACKA...	4 months ago
CONTRIBUTING.md	docs: add contributin...	last year
LICENSE	vision.cpp: launch	last year
README.md	python: packaging an...	7 months ago
pyproject.toml	python: fixes for runni...	7 months ago

vision.cpp

Computer Vision ML inference in C++

- Self-contained C++ library
- Efficient inference on consumer CPU and GPUs (NVIDIA, AMD, Intel)
- Lightweight deployment on many platforms (Windows, Linux, MacOS)
- Growing number of supported models behind a simple API
- Modular design for full control and implementing your own models

Based on ggml similar to the llama.cpp project.

Features

Model	Task	Backends
MobileSAM	Promptable segmentation	CPU, Vulkan
BiRefNet	Dichotomous segmentation	CPU, Vulkan
Depth-Anything	Depth estimation	CPU, Vulkan
MI-GAN	Inpainting	CPU, Vulkan
ESRGAN	Super-resolution	CPU, Vulkan
*Implement a model [**Guide**]*		

Backbones: SWIN (v1), DINO (v2), TinyViT

Get Started

Get the library and executables:

- Download a release package and extract it,
- or build from source.

Example: Select an object in an image

Let's use MobileSAM to generate a segmentation mask of the plushy on the right by passing in a box describing its approximate location.

You can download the model and input image here: MobileSAM-F16.gguf | input.jpg

CLI

Find the `vision-cli` executable in the `bin` folder and run it to generate the mask:

```
vision-cli -m MobileSAM-F16.gguf -i input.jpg -p
```

Pass `--composite output.png` to composite input and mask. Use `--help` for more options.

API

```
#include <visp/vision.h>
using namespace visp;

void main() {
  backend_device cpu = backend_init(backend_type
  sam_model sam = sam_load_model("MobileSAM-F16.g

  image_data input_image = image_load("input.jpg"
  sam_encode(sam, input_image);

  image_data object_mask = sam_compute(sam, box_
  image_save(object_mask, "mask.png");
}
```

This shows the high-level API. Internally it is composed of multiple smaller functions that handle model loading, pre-processing inputs, transferring data to backend devices, post-processing output, etc. These can be used as building blocks for flexible functions which integrate with your existing data sources and infrastructure.

Models

MobileSAM

Model download | Paper (arXiv) | Repository (GitHub) | Segment-Anything-Model | License: Apache-2

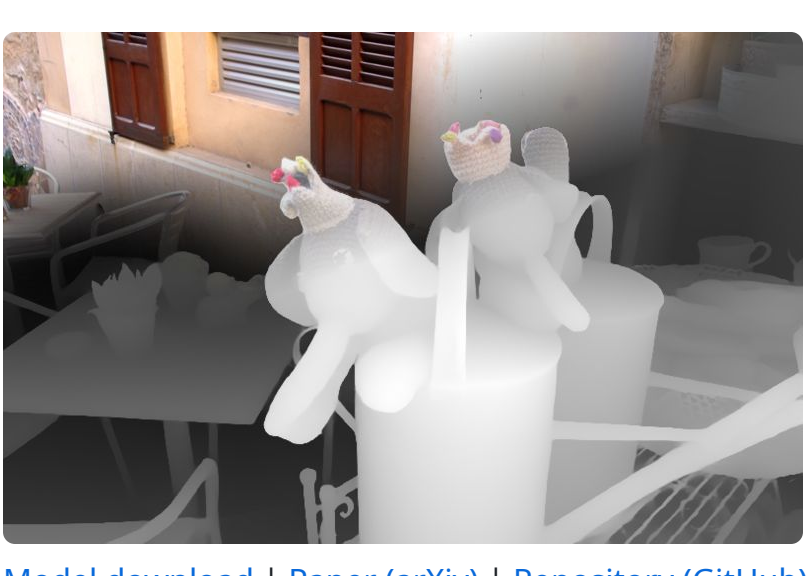
```
vision-cli sam -m MobileSAM-F16.gguf -i input.png
```

BiRefNet

Model download | Paper (arXiv) | Repository (GitHub) | License: MIT

```
vision-cli birefnet -m BiRefNet-lite-F16.gguf -i
```

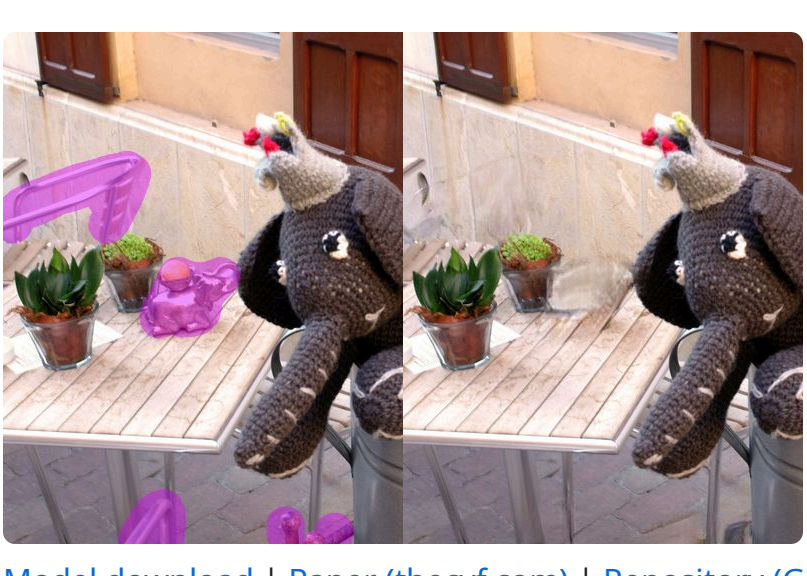
Depth-Anything V2



Model download | Paper (arXiv) | Repository (GitHub) | License: Apache-2 / CC-BY-NC-4

```
vision-cli depth-anything -m Depth-Anything-V2-Sm
```

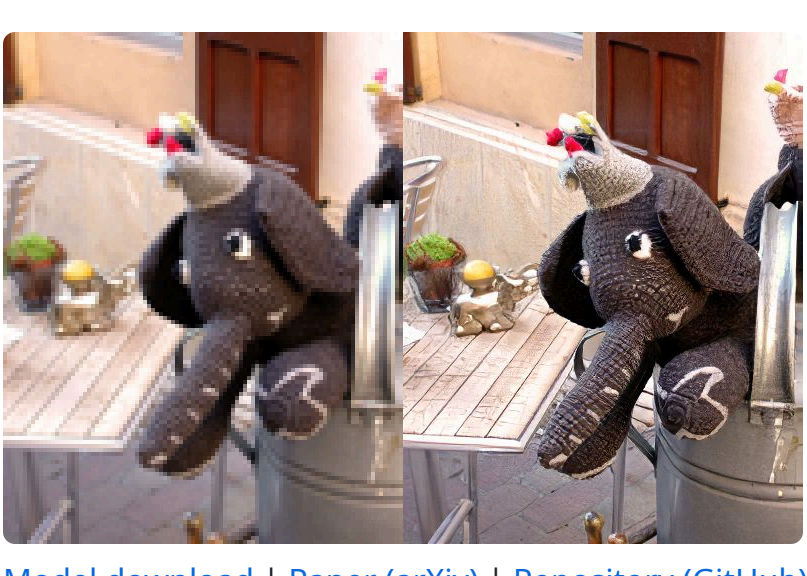
MI-GAN



Model download | Paper (thecvf.com) | Repository (GitHub) | License: MIT

```
vision-cli migan -m MIGAN-512-places2-F16.gguf -i
```

Real-ESRGAN



Model download | Paper (arXiv) | Repository (GitHub) | License: BSD-3-Clause

```
vision-cli esrgan -m ESRGAN-4x-foolhardy_Remacri
```

Converting models

Models need to be converted to GGUF before they can be used. This will also rearrange or precompute tensors for more optimal inference.

To convert a model, install uv and run:

```
uv run scripts/convert.py <arch> MyModel.pth
```

where `<arch>` is one of `sam, birefnet, esrgan, ...` .

This will create `models/MyModel.gguf` . See `convert.py --help` for more options.

Building

Building requires CMake and a compiler with C++20 support.

Get the sources

```
git clone https://github.com/Acly/vision.cpp.git
cd vision.cpp
```

Configure and build

```
cmake . -B build
cmake --build build --config Release
```

Vulkan *(Optional)*

Building with Vulkan GPU support requires the Vulkan SDK to be installed.

```
cmake . -B build -D VISP_VULKAN=ON
```

Tests *(Optional)*

Build with `-DVISP_TESTS=ON` . Run all C++ tests with the following command:

```
cd build
ctest -C Release
```

Some tests require a Python environment. It can be set up with uv:

```
# Setup venv and install dependencies (once only
uv sync --dev

# Run python tests
uv run pytest
```

Performance

Performance optimization is an ongoing process. The aim is to be in the same ballpark as other frameworks for inference speed, but with:

- much faster initialization and model loading time (<100 ms)
- lower memory overhead
- tiny deployment size (<5 MB for CPU, +30 MB for GPU)

Inference speed

- CPU: AMD Ryzen 5 5600X (6 cores)
- GPU: NVIDIA GeForce RTX 4070

MobileSAM, 1024x1024

		vision.cpp	PyTorch	ONNX Runtime
cpu	f32	669 ms	601 ms	805 ms
gpu	f16	19 ms	16 ms	

BiRefNet, 1024x1024

Model			*vision.cpp*	PyTorch	ONNX Runtime
Full	cpu	f32	16333 ms	18290 ms	
Full	gpu	f16	208 ms	190 ms	
Lite	cpu	f32	4505 ms	10900 ms	6978 ms
Lite	gpu	f16	85 ms	84 ms	

Depth-Anything, 518x714

Model			vision.cpp	PyTorch
Small	gpu	f16	11 ms	10 ms
Base	gpu	f16	24 ms	22 ms

MI-GAN, 512x512

Model			vision.cpp	PyTorch
512-places2	cpu	f32	523 ms	637 ms
512-places2	gpu	f16	21 ms	17 ms

Setup

- vision.cpp: using vision-bench, GPU via Vulkan, eg. `vision-bench -m sam`
- PyTorch: v2.7.1+cu128, eager eval, GPU via CUDA, average *n* iterations after warm-up

Dependencies (integrated)

- ggml - ML tensor library | MIT
- stb_image - Image load/save/resize | Public Domain
- fmt - String formatting *(only if compiler doesn't support <format>)* | MIT