SNT

User Best Practices and Results on MeluXina

Presented by

Mohamed Adel Mohamed ALI

UNIVERSITY OF LUXEMBOURG

Agenda

- Introduction to CVI² Research Group
- Overview of MeluXina Supercomputer
- System Structure and Access
- Working with MeluXina
- Case Studies and Results
- Tips for Maximizing Performance
- Common Pitfalls to Avoid
- Conclusion and Further Resources

Computer Vision, Imaging and Machine Intelligence

Research on computer vision, image processing, image analysis, visual data understanding, and machine learning.

22 members, **6** women
13 nationalities
> 150 peer-reviewed scientific publications

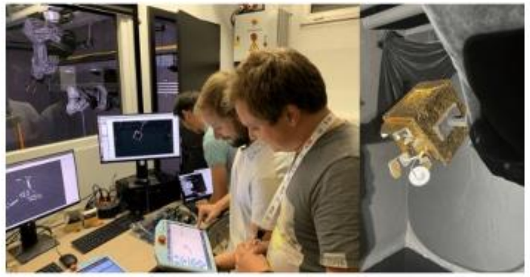
10 PhD theses and **15 MSc** theses successfully completed
8 PhD + **1 MSc** theses ongoing
8 Research Associates ongoing
2 Research Scientists
1 Professor

> 13 M€ Funding since 2009
Industrial partners: LMO, IEE, Infinite Orbits, Artec 3D, DataThings, POST Luxembourg
European Defense Fund, EU H2020 & ESA
Ministry of Economy
Fonds National de la Recherche (FNR)

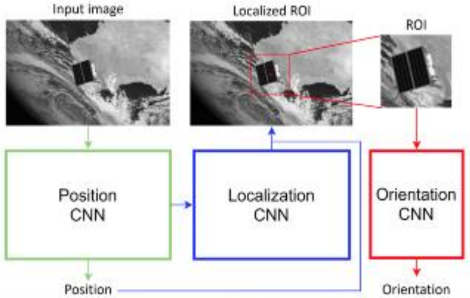
4 IEEE Best Paper Awards
4 patents

CVI²: Computer Vision, Imaging and Machine Intelligence Research Group

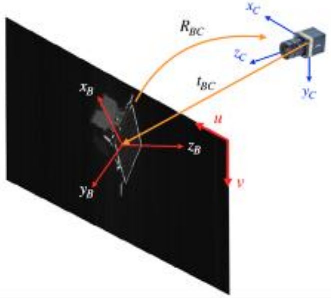
The Computer Vision, Imaging & Machine Intelligence Research Group (CVI²) at the Interdisciplinary Centre for Security, Reliability and Trust (SnT) of the University of Luxembourg (UL), headed by Prof. Dr. Djamila Aouada.



DIOSSA: Deep Learning-based In-orbit Space Situational Awareness



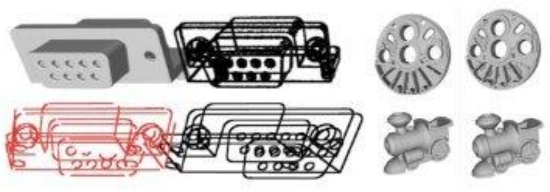
MEET-A – Multi-modal Fusion of Electro-optical Sensors for Spacecraft Pose Estimation Towards Autonomous in- Orbit Operations



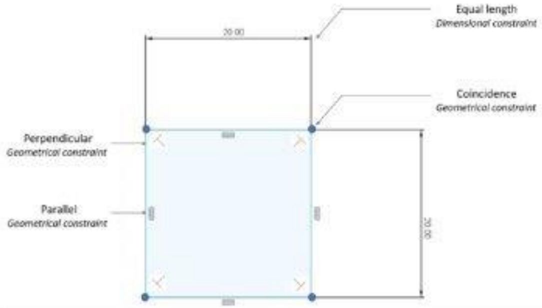
Space Situational Awareness Instrumentation

CVI²: Computer Vision, Imaging and Machine Intelligence Research Group

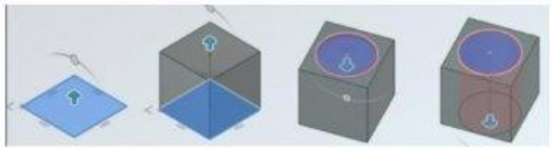
The Computer Vision, Imaging & Machine Intelligence Research Group (CVI²) at the Interdisciplinary Centre for Security, Reliability and Trust (SnT) of the University of Luxembourg (UL), headed by Prof. Dr. Djamila Aouada.



Deep Learning of 3D Scanned Data



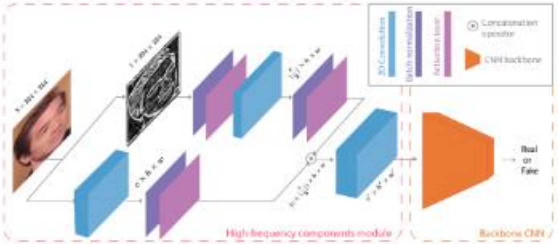
CASCADES: Constrained Sequence modelling of CAD for reverse Engineering from 3d Scans



FREE-3D: Feature-based Reverse Engineering Of 3D Scans

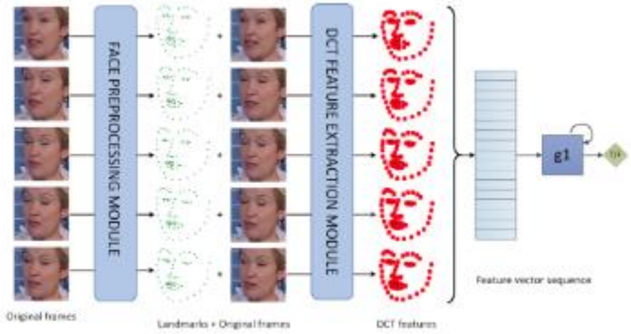
CVI²: Computer Vision, Imaging and Machine Intelligence Research Group

The Computer Vision, Imaging & Machine Intelligence Research Group (CVI²) at the Interdisciplinary Centre for Security, Reliability and Trust (SnT) of the University of Luxembourg (UL), headed by Prof. Dr. Djamila Aouada.



FakeDeTer: DeepFake Detection using Spatio-Temporal-Spectral Representations for Effective Learning

UNFAKE: Unsupervised multi-type explainable deepFAKE detection



Proving Digital Asset Integrity Using Deepfake Detection

CVI²: Computer Vision, Imaging and Machine Intelligence Research Group

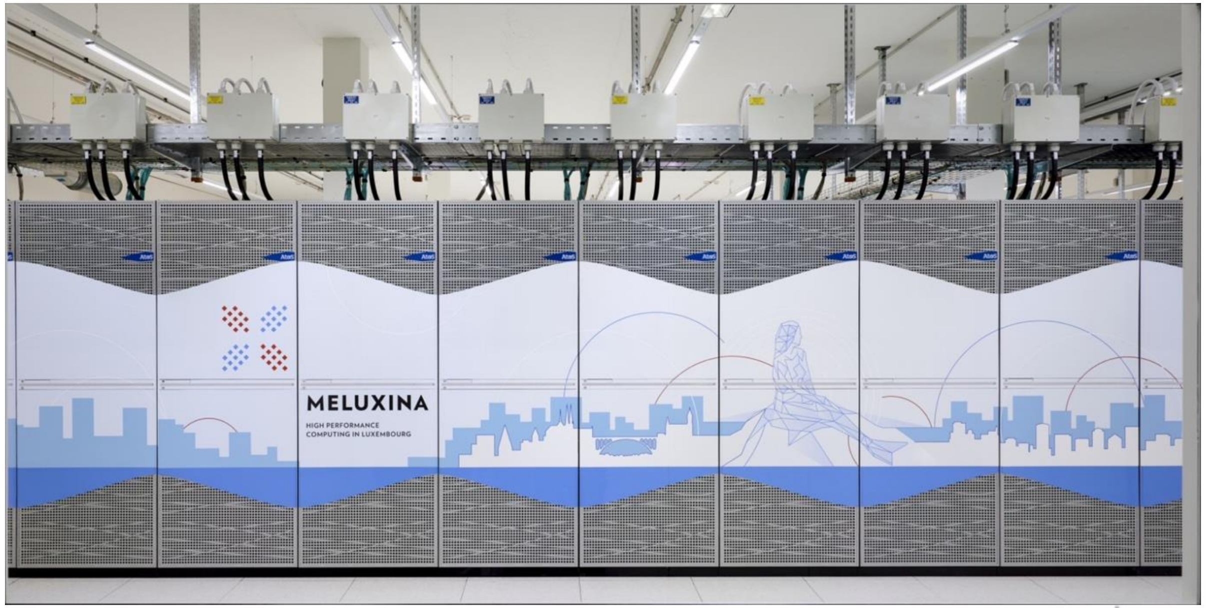
Introduction to MeluXina

LuxProvide's MELUXINA Supercomputer

- High-performance computing (HPC) cluster
    - 18 PetaFlops computing power, 20 PetaBytes storage
- Ranked 36th globally, greenest in EU (Top500)

LUX PROVIDE
NEXT-LEVEL INSIGHTS WITH SUPERCOMPUTING

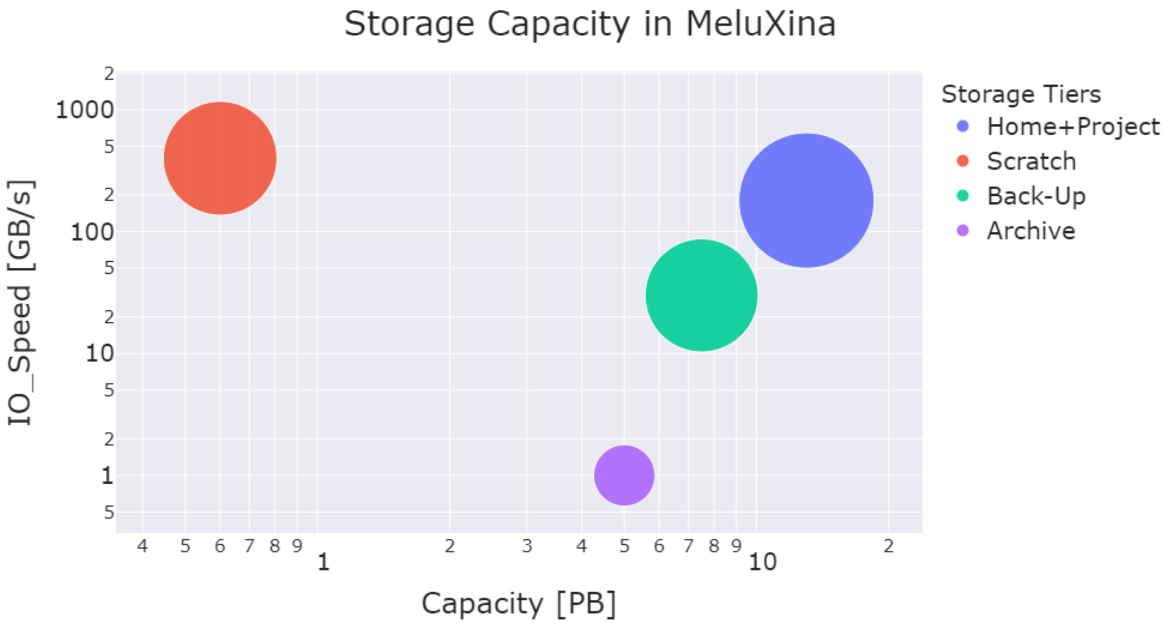
MELUXINA
HIGH PERFORMANCE COMPUTING IN LUXEMBOURG



uni.lu | SnT

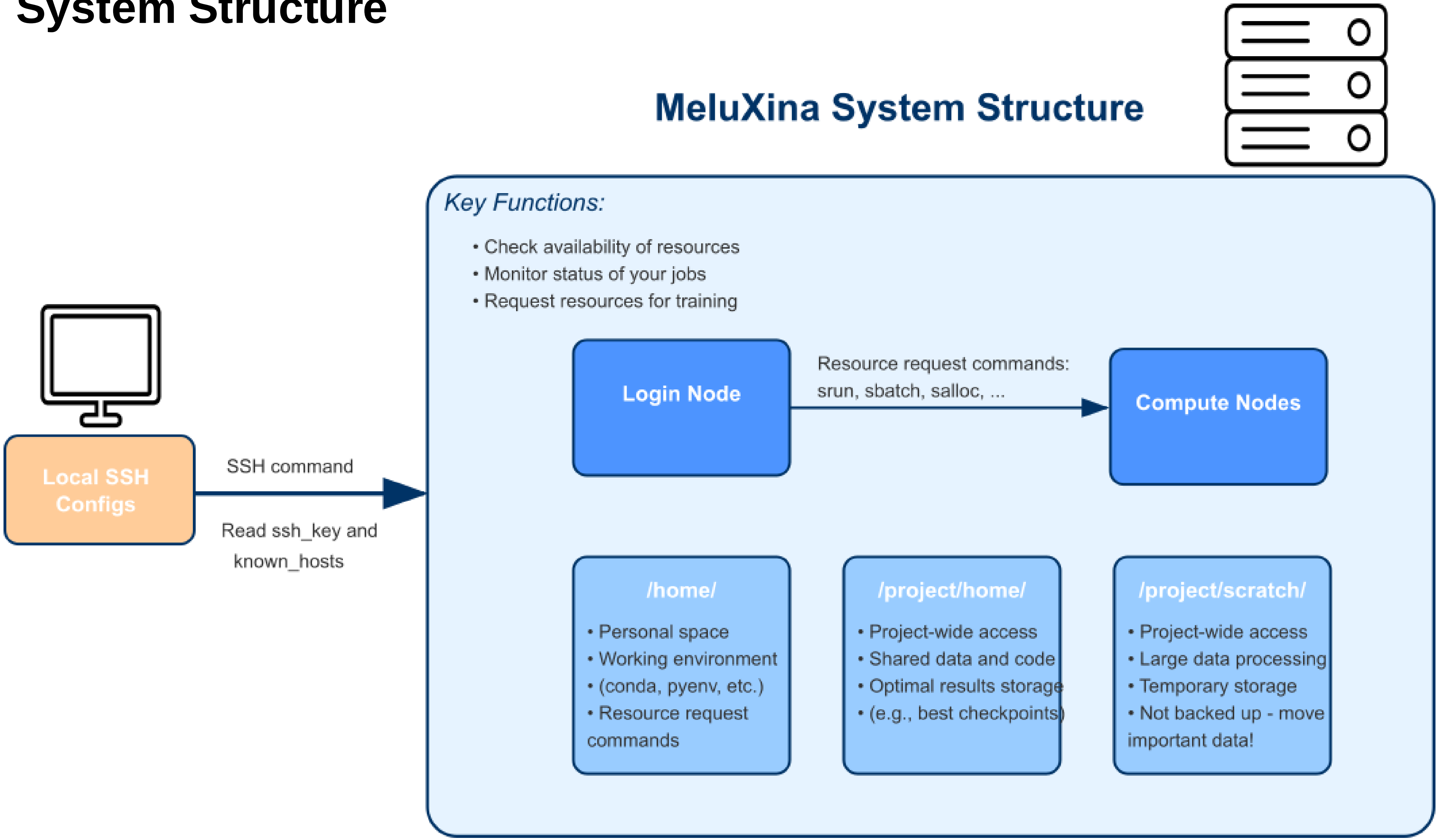
System Structure

- **Login node:**
  - Where you login after ssh command
  - Used for checking resource availability, job status, and requesting resources
- **Compute nodes:**
  - Where your model training/testing occurs
  - 200 nodes, each with:
    - 2x AMD EPYC Rome (128 cores)
    - 4x NVIDIA Ampere GPUs (40GB each)
    - 512GB RAM
- **Storage:**
  - Permanent Storage (5TiB): /project/home/p200249/
  - High speed storage (Scratch): /project/scratch/p200249/
    - Cleaned-up after some time
    - Use it to read/write during experiments, then copy back the data to permanent storage
  - /home/: Only you accessible, place your working env here (conda, pyenv, etc.)



https://docs.lxp.lu/system/overview/

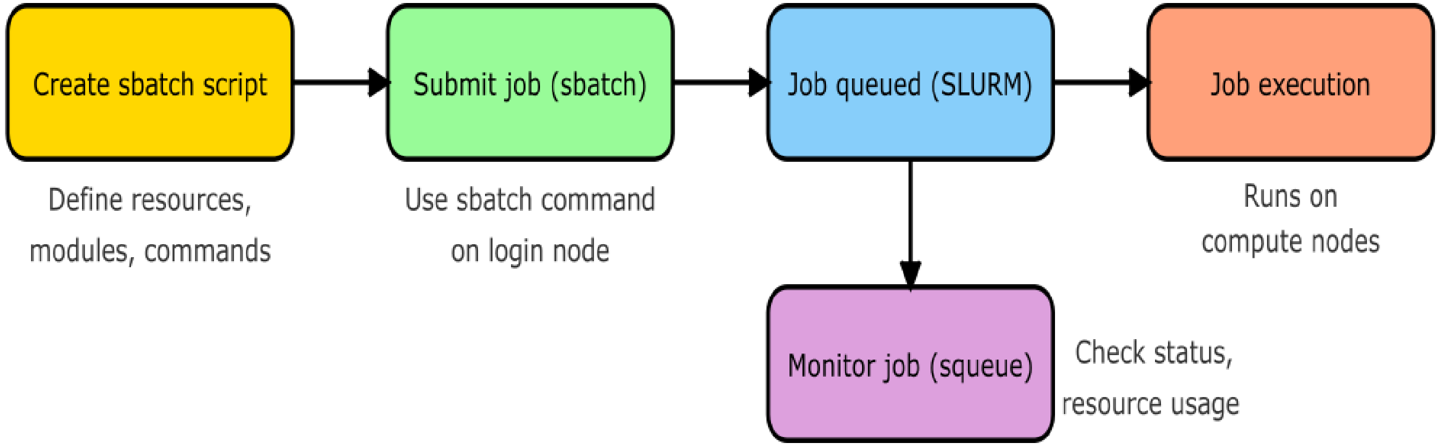
System Structure



Working with MeluXina

Resource Management with SLURM

- Key commands:
  - squeue: Check status of your jobs
  - salloc: Good for interactive jobs
    - Example: salloc -A p200111 --res {gpudev, cpudev} -q dev -N 1 -t 0-0:10:0
  - srun: For job steps
  - sbatch: For passive jobs (recommended)
- Always check GPU utilization
- Optimize data loader to maximize GPU usage



MeluXina Job Submission and Execution Workflow

Working with MeluXina

Working Environment Setup

Module system:

- module avail: List available modules
- module spider name: Search specific modules
- module list: List loaded modules
- module purge: Unload all modules
- module load: Load required modules

Python environments:

- Python virtual env: Specify Python version needed
- Conda env: Can transfer your root conda env and specific envs to MeluXina

Important: Care about CUDA version, GCC version, Pytorch/Tensorflow version

Working with MeluXina

Best Practices for Job Submission

- Use sbatch scripts for job submission
- Have a separate folder to store essential sbatch scripts for different configs/projects

```
#!/bin/bash -l
#SBATCH --account=p200111 #your project account
#SBATCH -J JobName #Your script name
#SBATCH -p gpu #Partition (GPU or CPU)
#SBATCH -N 1 #number of nodes
#SBATCH --qos default #can be default, long, urgent, ...
#SBATCH --time=1-23:50:00 #Request time
#SBATCH --gres=gpu:4
#SBATCH --constraint=a100
#SBATCH --ntasks=20
#SBATCH --cpus-per-task=4
#SBATCH --output /project/scratch/p200249/username/slurm/mel-%j.out

module purge
module load Singularity-CE
# Add other necessary module loads here

# Your job commands here
```

Example sbatch script

Working with MeluXina

Data Management and Synchronization

- Use /project/scratch/ for temporary data processing
- Move important results to /project/home/ for long-term storage
- Use rsync to synchronize code between local machines and MeluXina

```
"sync-rsync.sites": [
  {
    "name": "Sync. Scratch to Tier2",
    "localPath": "/project/scratch/p200249/username/data/results/project",
    "remotePath": "/project/home/p200249/username/data/results/project",
    "upOnly": true,
  },
  {
    "name": "Sync. Tier2 to Scratch",
    "localPath": "/project/home/p200249/username/data/",
    "remotePath": "/project/scratch/p200249/username/data/",
    "upOnly": true,
    "exclude": ["*"],
    "include": ["*/", "meta.yaml", "images/*", "datasets/*"]
  }
]
```

Example VSCode Rsync Extension configuration for local synchronization

Working with MeluXina

Data Management and Synchronization

- Use /project/scratch/ for temporary data processing
- Move important results to /project/home/ for long-term storage
- Use rsync to synchronize code between local machines and MeluXina

```
"sync-rsync.sites": [
  {
    "name": "Sync. Project",
    "localPath": "/home/users/u101185/projects",
    "remotePath": "rhermary@access-aion.uni.lu:projects",
    "shell": "ssh -p 8022",
    "downOnly": true,
    "exclude": [".vscode", "analysis", ".mypy_cache"]
  },
  {
    "name": "Sync. Results Up",
    "localPath": "/project/home/p200249/rhermary/data/results/perturbations/",
    "remotePath": "rhermary@access-iris.uni.lu:common_data/rhermary/results/",
    "shell": "ssh -p 8022",
    "upOnly": true,
    "exclude": ["mlruns_[1-9]", "slurm/test-*", ".trash"],
  },
]
```

Example VSCode Rsync Extension configuration for remote synchronization

Working with MeluXina

Optimization Techniques

Data Loading:

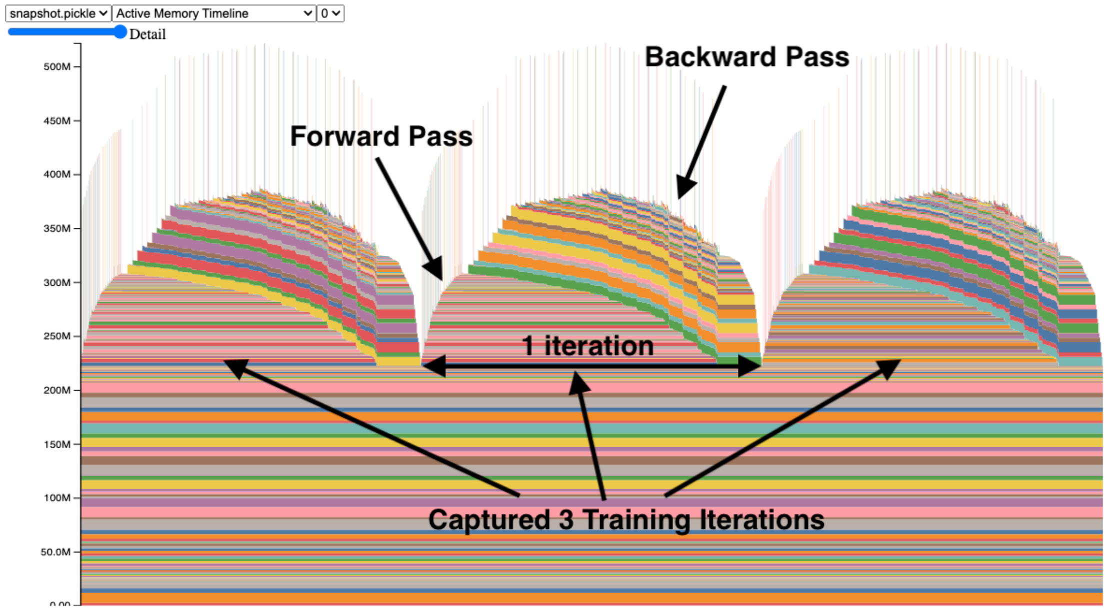
- Implement efficient data pipelines (preprocessing)
- Use multi-threading and multi-processing
- Optimize batch sizes to fit within memory constraints

GPU Memory Usage:

- Employ mixed precision training and gradient checkpointing
- Use all GPUs of a single node for efficient GPU/Node consumption

Parallel Training:

- Utilize DistributedDataParallel for multi-GPU setups
- Fine-tune batch size, learning rate, and communication overhead for distributed training

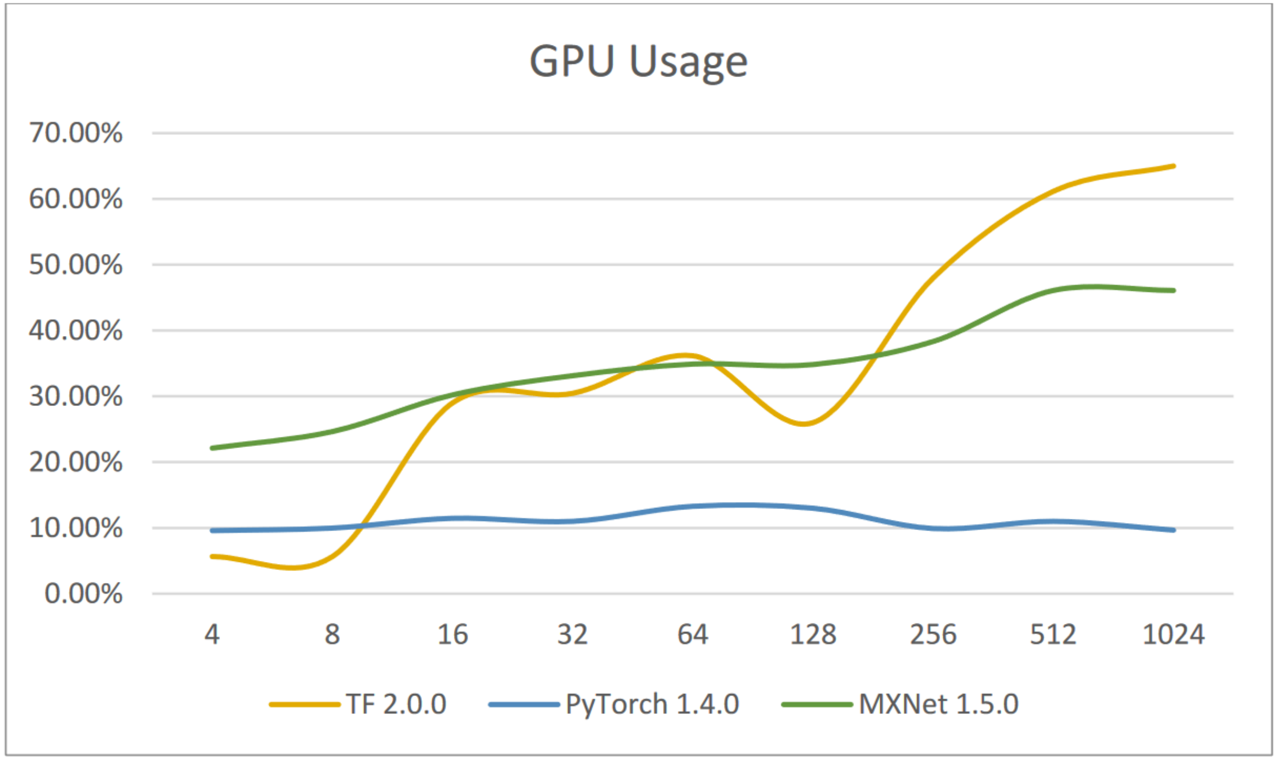


Memory Snapshot

Working with MeluXina

Monitoring and Optimization

- Use myquota to check storage quota usage
- Monitor GPU utilization in real-time
- Implement your own logging system alongside SLURM logs
- Save checkpoints wisely, not all checkpoints
- Even though SLURM provides training logs, have your own logging with your preferred log structure



Working with MeluXina

Advanced Techniques

- Build Singularity images from Docker images for stability across platforms
- Use srun for setup steps, especially for multi-node jobs
- Parallel experiment launch using srun:
- Use environment variables to identify unique experiment IDs:

```
OUTPUT="$SCRATCH/data/results/perturbations/slurm/mel-%j-%t.out"
SANDBOX_PATH="$LOCAL_TMPDIR/singularity_sandbox_$TMPDIR_NAME"
srun --ntasks 1 singularity build --sandbox $SANDBOX_PATH $IMAGE_PATH
srun --verbose --overlap --output $OUTPUT --ntasks 20 --gpu-bind=map_gpu:0,1,2,3 \
--export=ALL,SINGULARITYENV_DATA_DIR=$XP_CONTAINER_DATA_DIR \
singularity exec --nv --pwd $CONTAINER_WORKDIR \
--bind $CURRENT_ENV_DATA_DIR:$XP_CONTAINER_DATA_DIR \
$SANDBOX_PATH/ ./$SCRIPT_PATH
```

```
echo NODE_ID: $SLURM_NODEID
echo LOCAL_ID: $SLURM_LOCALID
echo NNODES: $SLURM_NNODES
TASK_ID=$((SLURM_NNODES * SLURM_LOCALID + SLURM_NODEID))
# Select a hyper-parameter: Example
NORMAL_CLASS=$(( TASK_ID % 10 ))
```

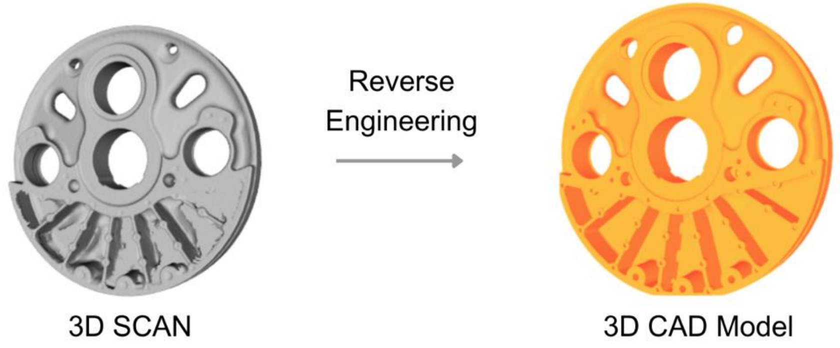
Case Studies and Results

DAVINCI Project

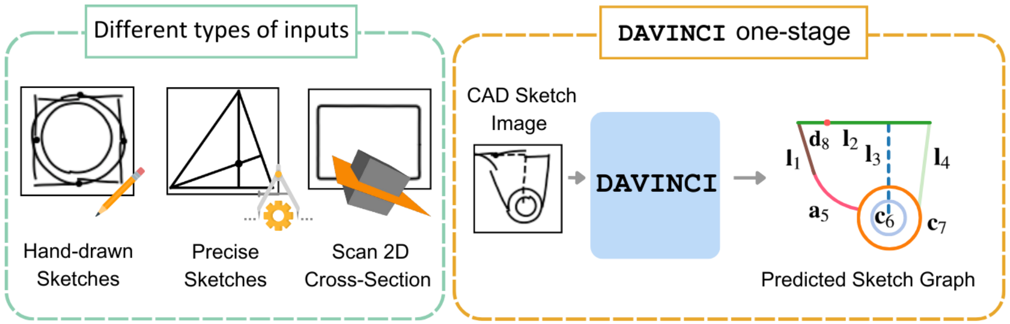
Example 1: DAVINCI Project

- Paper: "DAVINCI: A Single-Stage Architecture for Constrained CAD Sketch Inference", BMVC 2024
- Setup:
  - Used 4 GPUs (48GB each) with DistributedDataParallel
  - Increased batch size from 64 to 512 per GPU
  - Increased learning rate from 1e-4 to 3.5e-4
- Results:
  - Reduced training time from 2-2.5 hours to 30 minutes
  - Additional optimizations made to better utilize GPUs

Scan2CAD Project



Oveview of Davinci

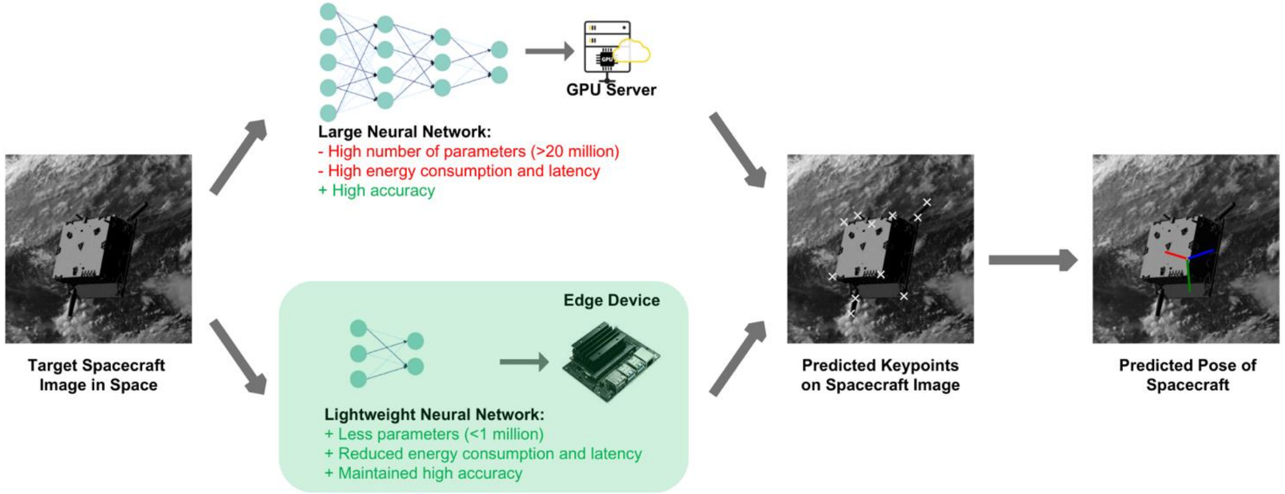


Case Studies and Results

Knowledge Distillation

Example 2: Efficient Pose estimation using Knowledge Distillation

- Setup:
  - Used up to 50GB of GPU memory per node
  - Implemented parallel training (multi-GPU setup)
  - Used job dependencies to ensure order and avoid conflicts between jobs
- Results:
  - Higher throughput, completing multiple epochs per hou consistently
  - Better ability to handle larger workloads efficiently
  - Consistently high resource utilization
  - Optimal performance during training

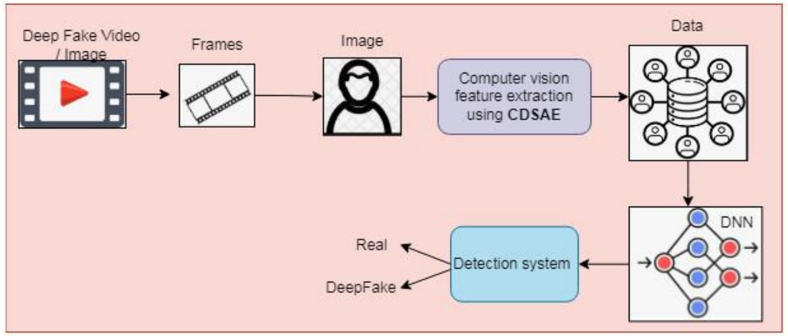


Case Studies and Results

Deepfake Detection

Example 3: Multi-task TimeSFormer-based Learning Framework for Deepfake Detection

- Setup:
  - Used up to 40GB of GPU memory per node
  - Implemented parallel training (multi-GPU setup)
- Results:
  - Higher throughput, completing multiple epochs per hour consistently
  - Better ability to handle larger workloads efficiently
  - Consistently high resource utilization
  - Optimal performance during training
  - Speed up compared to other alternatives: …



Deepfake Detection

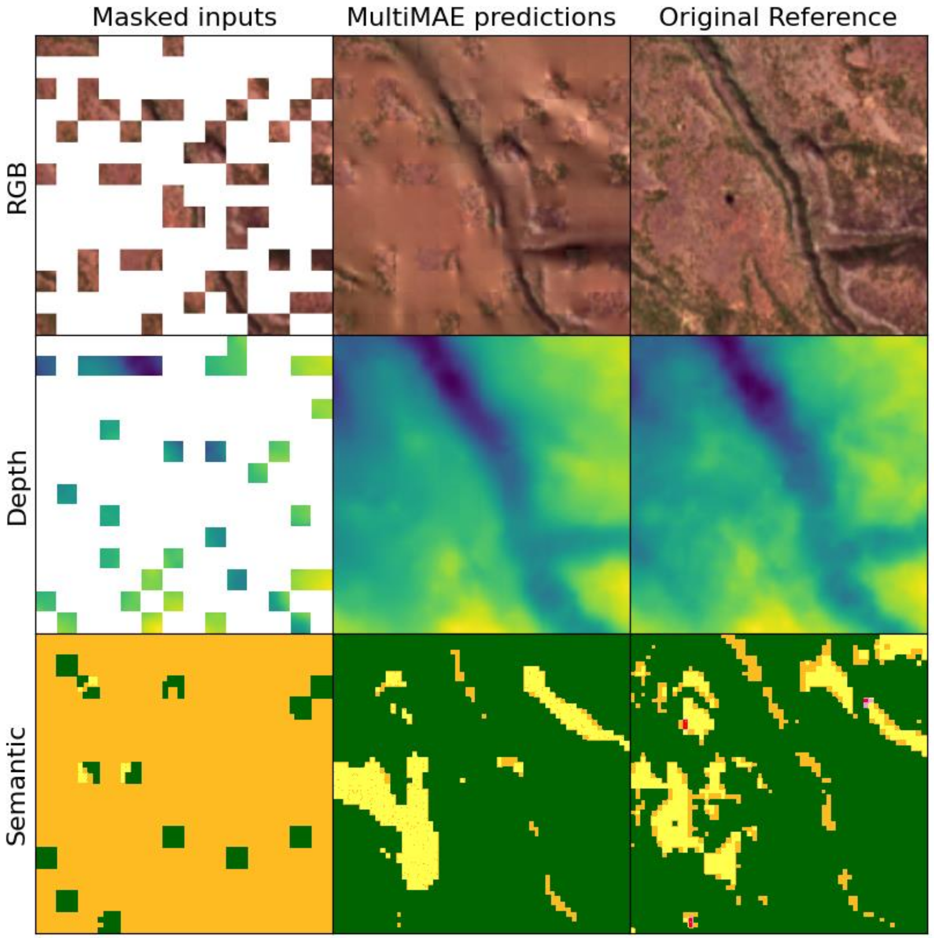
Visualization results

Case Studies and Results

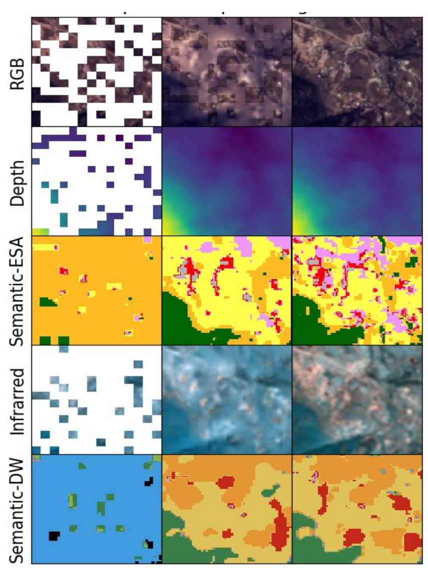
AI4CC Earth Observation

Example 4: Pretraining Large Masked Autoencoders for Earth Observation Downstream tasks

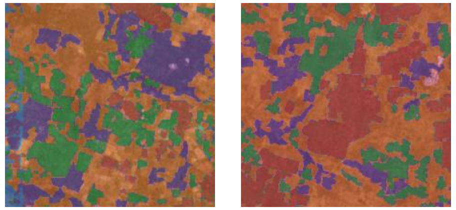
- **Setup:**
  - Implementing parallel training:
    - Pretraining with up to 4 GPUs
  - Efficient storage and optimal access of large-scale datasets (HDF5 files).
  - Handling large inputs, e.g. multimodal and multispectral data.
  - Integration of monitoring/visualising tools external, e.g. weight and biases.
- **Results:**
  - Faster training time, even for heavy models, such as ViTs.
  - Optimal utilisation of training resources.
  - Efficient handling of large non-standard inputs, e.g. multispectral and multimodal data.



Pretraining stage: Visualisation of results from reconstruction of multiple modalities.



Pretraining stage: Scaling up number of modalities



Finetuning stage: Results on semantic segmentation

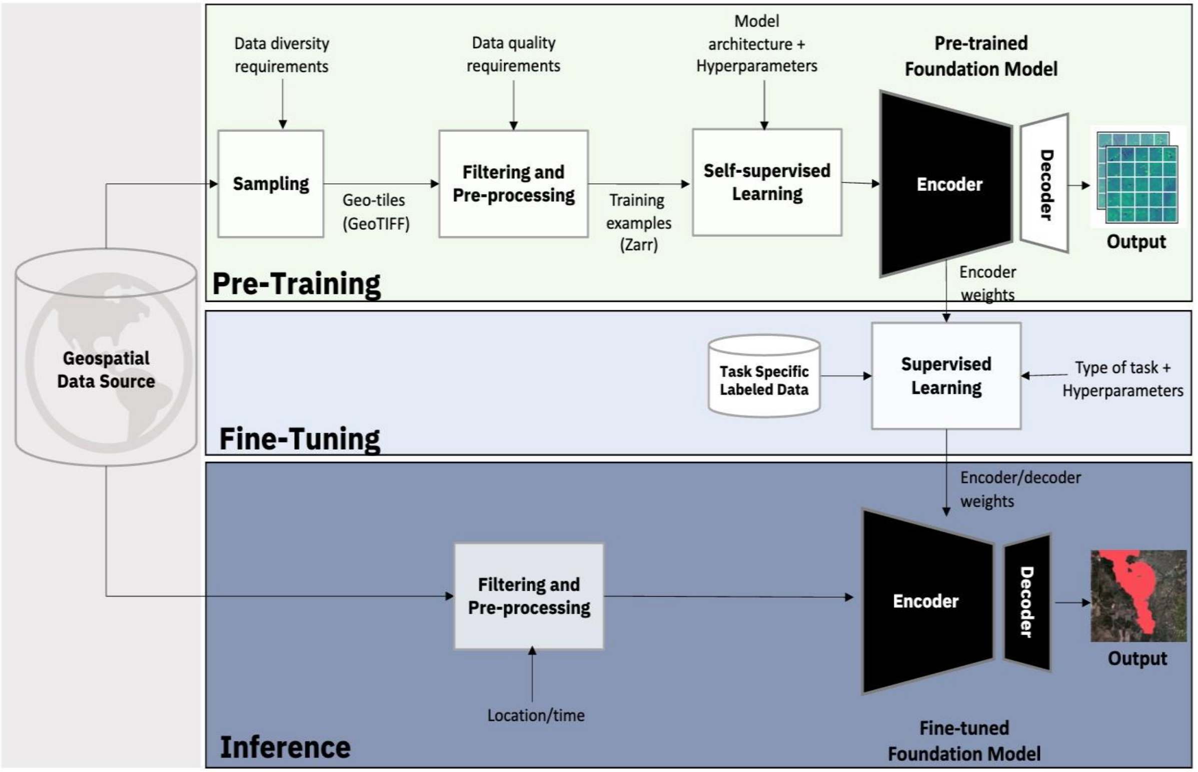
Case Studies and Results

AI4CC Earth Observation

SELF-SUPERVISED TRAINING OF LARGE MODELS FOR EARTH OBSERVATION TASKS.

- Involves the use of models with huge number of parameters, e.g. models based on Vision Transformers (ViT).
  - Some versions of ViT-based models could have up to **632 millions of parameters**.
  - Pretraining those models might be **computationally expensive**.
- Models should handle **not standard inputs,** which is commonly **memory intensive**.

Example: **Training one ViT-B based MAE (~95m parameters) for 400 epochs could take up to 5 days using 4 GPUs on Meluxina.**



Example of different stages for training large models (foundation models) for Earth Observation (EO) tasks.

Tips for Maximizing Performance

- Always use all GPUs on a single node for efficient GPU/Node consumption
- Adjust learning rates when scaling batch sizes
- Optimize code and training process for better GPU utilization
- Use Singularity containers for consistent environments across platforms
- Implement efficient data pipelines and preprocessing
- Use multi-threading and multi-processing for data loading
- Fine-tune batch size, learning rate, and communication overhead for distributed training

Common Pitfalls to Avoid

- Not checking GPU utilization regularly
- Saving all checkpoints instead of only essential ones
- Neglecting to optimize data loaders
- Ignoring the importance of proper logging
- Underutilizing available GPUs on a node

Conclusion

- MeluXina offers significant performance gains for large-scale machine learning tasks
- Proper resource allocation and optimization techniques are crucial for maximizing efficiency
- Continuous monitoring and adjustment of parameters lead to optimal performance
- Utilize advanced features like Singularity containers and parallel job launching for best results
- Always strive for efficient GPU utilization and optimized data processing

Further Resources

- MeluXina documentation: https://docs.lxp.lu/
- SLURM documentation
- Singularity and Docker documentation
- University tutorials and presentations

Thanks For the Team :)

Van Dat Nguyen
Doctoral Researcher

Ahmet Serdar Karadeniz
Doctoral Researcher

Nassim Mohamed ALIOUSALAH
Doctoral Researcher

Romain Hermary
Doctoral Researcher

Dr. Jose SOSA
Research Associate

Thank You

UNIVERSITY OF LUXEMBOURG