RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

cea cnrs Inria

FRANCE 2030 PROGRAMME DE RECHERCHE NUMÉRIQUE POUR L'EXASCALE

Infrastructure for Benchmarking

Exa-DI General Assembly
Paris, Feb 24th & 25th, 2026

Foreword

This presentation is about :

- ***INFRASTRUCTURE & TOOLS for SYSTEM-LEVEL and/or APPLICATION-LEVEL BENCHMARKS***

This presentation is not about :

- Benchmarking methodology, identifying relevant metrics & figures of merit, ...
- Micro-benchmarking, profiling, ...
- ...

Expectations for a Benchmarking Infrastructure ?

- Orchestrate the test campaign
  - select tests to run, expand parametric/combination cases, handle dependencies between tests, build the execution plan…
- Provision the application & tools
  - ensure the right version of the apps & their dependencies can be run on the computing nodes of the target machine
- Execute test campaign
  - launch test jobs on computing nodes, handle reservations with Slurm/OAR/…
- Produce test results
  - collect relevant metrics from test jobs, possibly after some postprocessing
- Gather test results in a central store
- Render benchmark reports
  - generate various plots for in-depth analysis of one benchmark or comparison between benchmarks

Our current solution : G5k-Testing

https://gitlab.inria.fr/numpex-pc5/wp2-co-design/g5k-testing

- Started in June 2025 as a proof-of-concept in the context of Proxy-GEOS-HC
  - CI pipeline was able to check the build process in multiple configurations (Kokkos / Raja, CUDA / HIP, …) but was missing the ability to actually *test* the resulting executables, since shared Gitlab runners @ Inria have no GPU
  - Idea to install project-specific runners on Grid'5000, based on some Inria tutorial.

- Main use case considered: periodic (daily ? weekly ?) test of the Proxy-GEOS-HC executables on multiple HW configurations to track possible regressions & to monitor performance trends

- Designed for simplicity and opportunistic reuse
  - Then extended incrementally for more and more use-cases

Overview of design choices for G5k-Testing

- ❑ Orchestrate the test campaign
  - ➡ 1 test job = 1 Gitlab CI job (YAML)
- ❑ Provision the applications & tools
  - ➡ Bash scripts to abstract the provisioning method (Guix, Spack, …)
    Creation of a Singularity Image including app & dependencies
- ❑ Execute test campaign
  - ➡ Gitlab runner on G5K front-end and scheduling with OAR
- ❑ Produce test results
  - ➡ Bash scripts called from Gitlab CI (application specific)
- ❑ Gather test results in a central store
  - ➡ JSON files in Git repo
- ❑ Render benchmark reports
  - ➡ Streamlit web app (Python)

Tests results & Central store

All test results stored hierarchically in "G5K-Testing" Git repo :

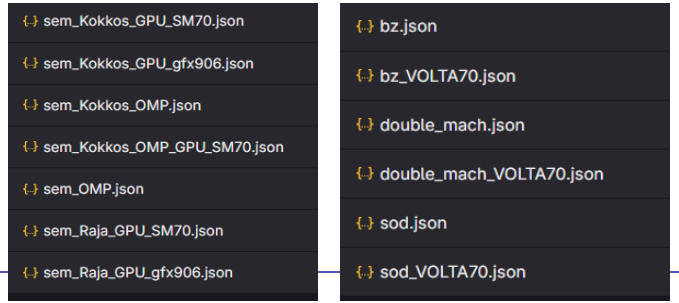
```
.
├── results
│   ├── proxy-app-1
│   │   ├── some-config.json
│   │   ├── another-config.json
│   │   └── yet-another-config.json
│   └── proxy-app-2
│       └── only-one-config-for-this-app.json
```

Each JSON file capture results from one test job, corresponding to one test configuration :

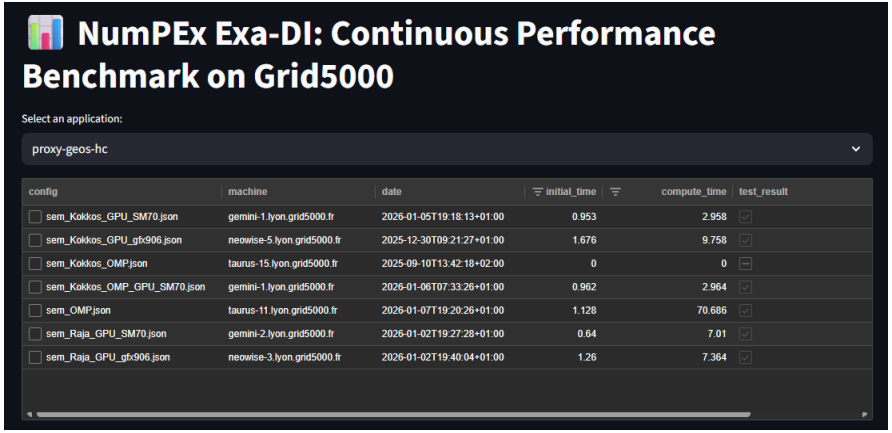
```json
{
  "machine": "neowise-5.lyon.grid5000.fr",
  "date": "2025-12-30T09:21:27+01:00",
  "initial_time": 1.676,
  "compute_time": 9.758,
  "test_result": true
}
```

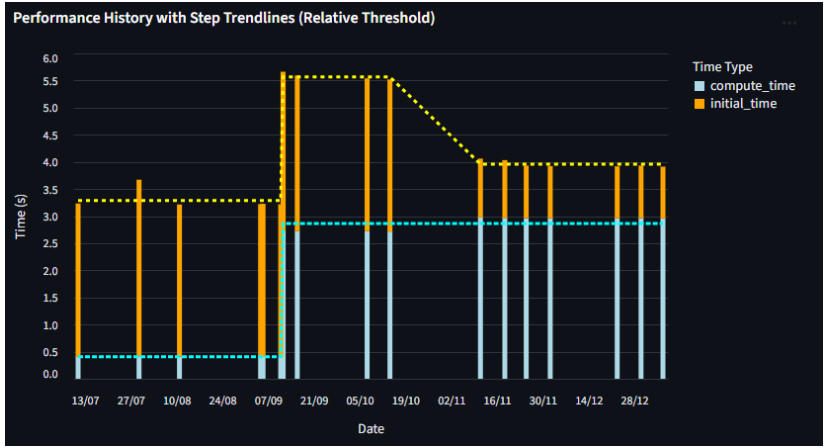
Each test job execution = 1 new Git commit !

Definition of a "test configuration" is application specific :



Rendering





- Using Streamlit, an open-source web app framework
- Hosted on Streamlit Community Cloud
- 257 loc (Python)

https://exa-di-g5k-dashboard-9fngnkdf3aenincc993rfy.streamlit.app/

Test Specification & Orchestration

Main design choice : each test job is one Gitlab CI Job

Foundation : ~350 lines of Gitlab CI YAML code defining

- The "base class" for a generic test job, implementing the complete workflow, up to the storage of test results
- Some "helper classes" for expressing trigger conditions, provisioning methods, and HW requirements

User can specify the test jobs for a given benchmark by multiple inheritance mechanism

- Mainly declarative : no code to write, only configuration variables to be set
- Only the metrics collection has to be implemented in a bash script
- Nothing available for parametric expansion : copy/paste is your friend

Test Specification: example (Proxy-geos-HC)

(1) base including all specific definitions for one application :

```yaml
# A base skeleton for all proxy-geos jobs => specify the script to be used for collecting the metrics
.proxy-geos-run:
  extends:
  - .g5k-run-single-experiment
  - .g5k-guix-provisioning
  - .ScheduledOrAPI                  # Can be called also via API
  variables:
    TARGET_BRANCH: main
    PARSE_SCRIPT_NAME: sem-proxy-geos.sh    # Relative to $PARSE_SCRIPT_DIR in the base job
    RESULT_DIR: results/proxy-geos-hc
    GUIX_TRANSFO: "--with-branch=${GUIX_PACKAGE}=${TARGET_BRANCH}"      # Default transfo => use the latest commit from the specified branch
    GUIX_VER_SUFFIX: "@0.0"                                  # We build the "version in progress", not a released version
    GUIX_PACKAGE: "${GUIX_FLAVOUR}${GUIX_VER_SUFFIX}"
    GUIX_TIME_MACHINE_URL: "https://gitlab.inria.fr/numpex-pc5/wp2-co-design/proxy-geos-hc/-/raw/${TARGET_BRANCH}/guix/config_proxyApp_time-machin
    EXEC_CMD: "${JSON_NAME}.exe --precision 2 --time-max 0.1"   # The command to launch the run
    SINGULARITY_OPTIONS: "--bind /usr/lib/x86_64-linux-gnu/:/usr/lib/x86_64-linux-gnu --bind /tmp:/tmp"   # Adding binding for Guix/GNU libs
  artifacts:
    paths:
     - $JSON_REPORT     # Store the generated JSON report as an artifact so the caller can access it latter on
    expire_in: 1 day
```

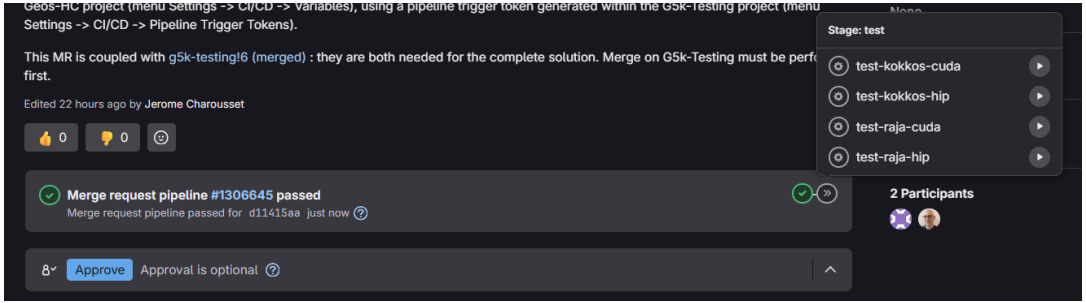
(2) many different test configurations :

```yaml
proxy-geos-kokkos-cuda:
  extends:
  - .proxy-geos-run
  - .g5k-v100-config
  variables:
    GUIX_FLAVOUR: proxy-geos-kokkos-cuda-v100
    JSON_NAME: sem_Kokkos_GPU_SM70
    SCHEDULE_PATTERN: "Weekly_A"

proxy-geos-raja-cuda:
  extends:
  - .proxy-geos-run
  - .g5k-v100-config
  variables:
    GUIX_FLAVOUR: proxy-geos-raja-cuda-v100
    JSON_NAME: sem_Raja_GPU_SM70
    SCHEDULE_PATTERN: "Weekly_B"

proxy-geos-kokkos-hip:
  extends:
  - .proxy-geos-run
  - .g5k-mi50-config
  variables:
    GUIX_FLAVOUR: proxy-geos-kokkos-hip-vega906
    JSON_NAME: sem_Kokkos_GPU_gfx906
    SCHEDULE_PATTERN: "Weekly_A"

proxy-geos-raja-hip:
  extends:
  - .proxy-geos-run
  - .g5k-mi50-config
  variables:
    GUIX_FLAVOUR: proxy-geos-raja-hip-vega906
    JSON_NAME: sem_Raja_GPU_gfx906
    SCHEDULE_PATTERN: "Weekly_B"
```

Bonus Feature : extension of Application CI Pipeline

In their daily work developpers are interested in a solution to quickly test a change on multiple HW platforms. This can be achieved very easily by "remote triggering" of G5k-Testing test jobs from the usual application CI pipeline.

For instance in Proxy-GEOS-HC, the Merge Request Pipeline offers 4 different tests for NVIDIA and AMD GPUs : they are just launching G5K-Testing jobs and wait for completion.



Needs for the Future

Benchmarking is becoming a major topic in all WGs.

We want more complex benchmarks:

- Weak-scaling, strong-scaling, ...
- Accuracy as a function of AMR min & max levels, time-to-solution for a given accuracy, …

We want benchmarks on more machines & bigger machines:

- Jean Zay, Adastra
- Alice Recoque

Main limitations of "G5k-Testing"

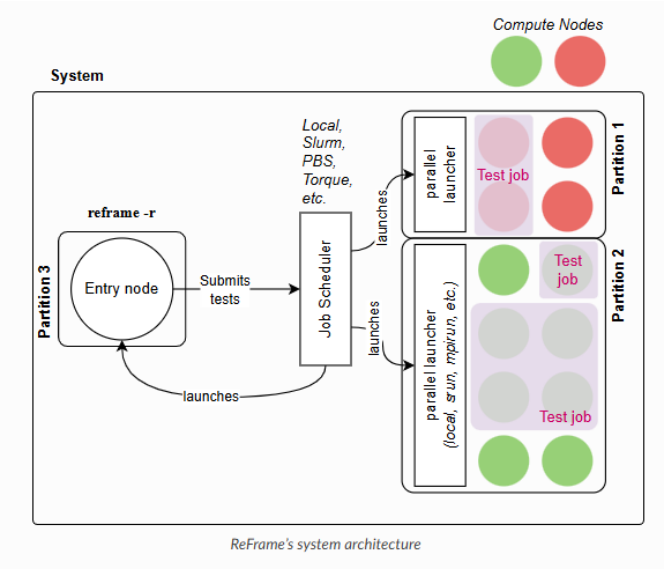
- ❑ Assumes a Gitlab runner on the HPC front-end (ok for Jean Zay, but for others ?)
- ❑ Assumes Singularity on the HPC front-end & computing nodes (ok for Jean Zay, but for others ?)
- ❑ Only 3 provisioning methods supported ; each new method is about 1-2 weeks of dev
- ❑ Only supports OAR today ; adding support of Slurm is easy though (1-2 weeks of dev)
- ❑ No support for parametric expansion of test cases (would require a generator ? no easy solution)
- ❑ Rendering capabilities quite basic ; significant dev efforts expected for new benchmarks
- ❑ Some refactoring needed
    - define core services and call them from YAML instead of embedding long bash sequences in YAML (debugging nightmare)
    - add support for "hooks" at various stages of the workflow to improve flexibility

Any alternative or helper ?

- ReFrame-HPC (Inbetween alternative and helper)
- Feel++ benchmark (Alternative)
- IOPS (Helper)
- Benchopt (?)

ReFrame-HPC

- Python framework by CSCS (Swiss National Supercomputing Center)
- Supports definition of Benchmarks as Python code
- Supports provisioning of applications by many different methods (but not Guix)
- Supports execution with many different schedulers, including OAR & Slurm
- Supports orchestration depending on the capabilities of the target machine
- Supports test parametrization programmatically
- Supports storage in local file system or SQlite database
- Trivial support for rendering - mostly text outputs !



ReFrame's system architecture

name	sysenv	job_nodelist	pvar	punit	pval	result
stream_test	generic:default+builtin	myhost	copy_bw	MB/s	17154.1	pass
stream_test	generic:default+builtin	myhost	triad_bw	MB/s	13664.8	pass

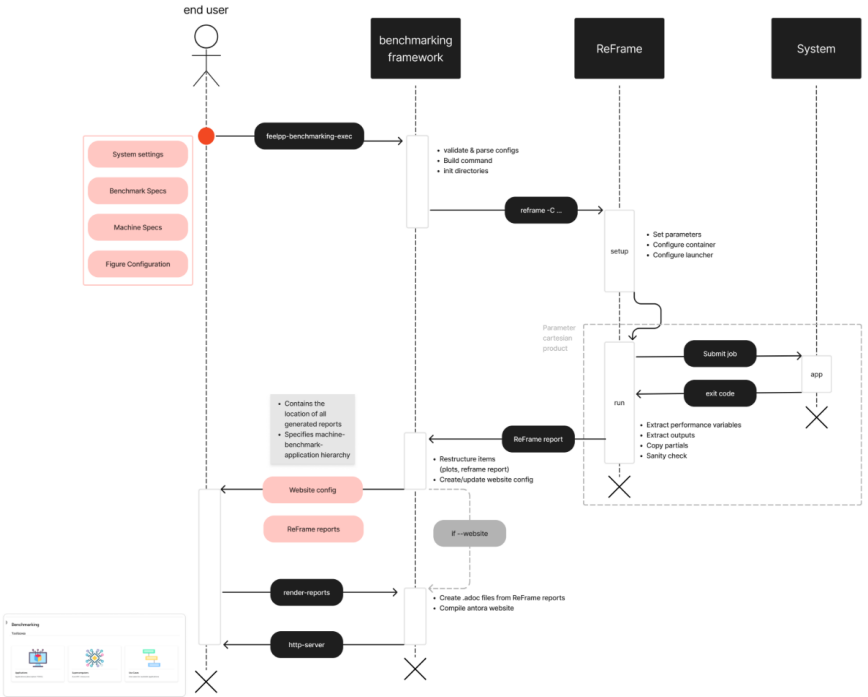
Example of Reframe Test job

```python
@rfm.simple_test
class stream_build_test(rfm.RegressionTest):
    valid_systems = ['*']
    valid_prog_environs = ['+openmp']
    build_system = 'SingleSource'
    sourcepath = 'stream.c'
    executable = './stream.x'

    @run_before('compile')
    def prepare_build(self):
        omp_flag = self.current_environ.extras.get('omp_flag')
        self.build_system.cflags = ['-O3', omp_flag]

    @sanity_function
    def validate(self):
        return sn.assert_found(r'Solution Validates', self.stdout)

    @performance_function('MB/s')
    def copy_bw(self):
        return sn.extractsingle(r'Copy:\s+(\S+)', self.stdout, 1, float)

    @performance_function('MB/s')
    def triad_bw(self):
        return sn.extractsingle(r'Triad:\s+(\S+)', self.stdout, 1, float)
```

Feel++ Benchmarking Project

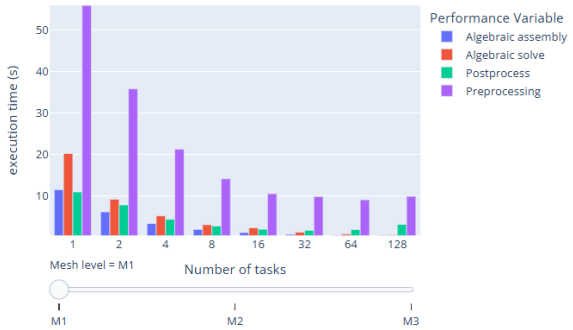
- Developped by CEMOSIS
- Declarative approach: everything in JSON files
- Internally calling ReFrame
- Adding a powerful report generator
- Coupled with Antora to generate a benchmark visualisation website
- Coupled with Girder for data storage
- Runs on Discoverer, Karolina, Lumi, MeluXina

- Slightly monolithic
- Steep learning curve, support needed

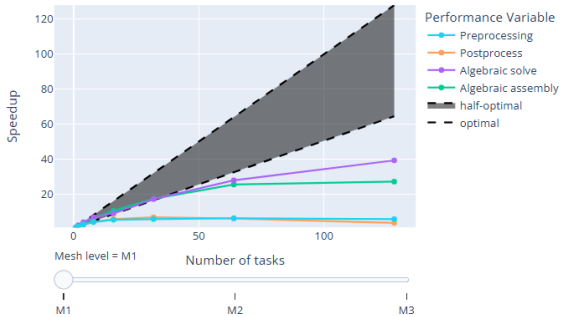


Example of Feel++ Benchmark reports

Performance (P2)



Speedup (P2)



Number of iterations of GMRES

IOPS

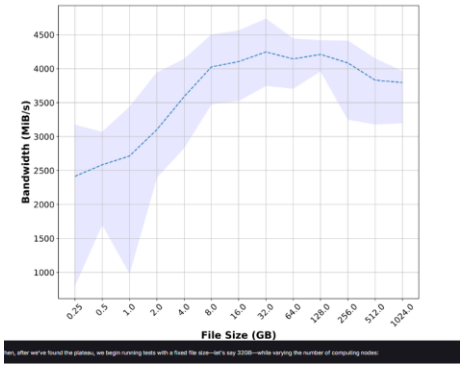
- Developped by Luan Teylo (Inria Bordeaux/ Exa-DoST)
- Originally designed for I/O performance studies
- Evolved into a generic framework for parametric benchmarks
- Declarative approach in YAML
  + Python code snippets

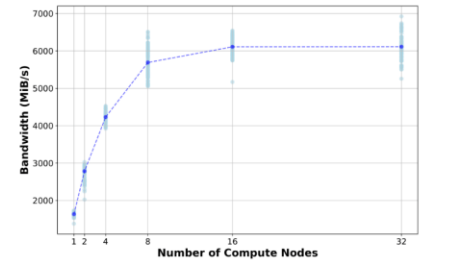
- Bonus feature: explore the parameter space to optimize a metric

Key Features

- **Parameter Sweeping**: Automatically generate and execute tests for all parameter combinations
- **Multiple Search Strategies**: Exhaustive, Bayesian optimization or random sampling
- **Execution Backends**: Run locally or submit to SLURM clusters
- **Execution Exploration**: Find and filter execution folders by parameter values
- **Smart Caching**: Skip redundant tests with parameter-aware result caching
- **Budget Control**: Set core-hour limits to avoid exceeding compute allocations
- **Automatic Reports**: Generate interactive HTML reports with plots and statistical analysis
- **Flexible Output**: Export results to CSV, Parquet, or SQLite

IOPS Example






```yaml
benchmark:
  name: "My Benchmark Study"
  workdir: "./workdir"
  executor: "local"
  repetitions: 3

vars:
  threads:
    type: int
    sweep:
      mode: list
      values: [1, 2, 4, 8]

  buffer_size:
    type: int
    sweep:
      mode: list
      values: [64, 256, 1024]

command:
  template: "my_benchmark --threads {{ threads }} --buffer {{ buffer_size }}"

scripts:
  - name: "benchmark"
    submit: "bash"
    script_template: |
      #!/bin/bash
      # Built-in variables: execution_id, execution_dir, repetition
      echo "Running execution {{ execution_id }}, repetition {{ repetition }}"
      {{ command.template }} > output.txt

    parser:
      # execution_dir is automatically set to each execution's folder
      file: "{{ execution_dir }}/output.txt"
      metrics:
        - name: throughput
      parser_script: |
        import re

        def parse(file_path: str):
            with open(file_path) as f:
                content = f.read()
            match = re.search(r"throughput:\s*([\d.]+)", content)
            return {"throughput": float(match.group(1)) if match else 0}

output:
  sink:
    type: csv
    path: "{{ workdir }}/results.csv"
```

Direction of Work:

From "G5k-Testing" to "BENDI" (BENchmark for exa-DI)

- ❑ Refactoring ✅
- ❑ Integration of IOPS ✅
- ❑ Pilot with GT3 (AMR) Benchmarks ➡ SOON
- ❑ Add support for Jean-Zay
- ❑ Enable caching mechanism for long experiments

Work in progress

CI Job:

```yaml
.samurai-run:
  extends:
  - .bendi-run-iops-experiment
  - .bendi-singularity-provisioning
  - .g5k-config
  variables:
    BENDI_APP_CONTROL_DIR: ${BENDI_CONTROL_DIR}/apps/samurai
    RESULT_DIR: results/samurai
    SCHEME: hllc
    #Following line not neeeded with IOPS runs
    #EXEC_CMD: "euler_2d --min-level $LEVEL_MIN --test-case $TEST_CASE --timers"
    SINGULARITY_DEF_FILE: ${BENDI_APP_CONTROL_DIR}/samurai-euler.def
  rules:
  - if: $CI_PIPELINE_SOURCE == "schedule" && $CI_PIPELINE_SCHEDULE_DESCRIPTION =~ /Samurai/

double-mach-reflection-samurai:
  variables:
    TEST_CASE: "double_mach_reflection"
    OAR_WALLTIME: "0:10:00"
    REPORT_NAME: "${TEST_CASE}"
    RESULT_DIR: results/samurai/${TEST_CASE}
    BENDI_IOPS_SPEC: ${BENDI_ROOT_DIR}/iops-specs/samurai_l1_vs_lmin.yml
  extends: .samurai-run
```

IOPS spec:

```yaml
benchmark:
  name: "Samurai - accuracy vs Lmin"
  repetitions: 1
  search_method: "exhaustive"
  # cache_file: "./iops_cache.db"

output:
  sink:
    type: csv

vars:
  # Level min
  levelmin:
    type: int
    sweep:
      mode: list
      values: [4, 5, 6, 7, 8]

  # Level max
  levelmax:
    type: int
    sweep:
      mode: list
      values: [8]

  # Output file
  timers_file:
    type: str
    expr: "{{ execution_dir }}/stdout"

command:
  template: >
    euler_2d --min-level {{ levelmin }} --test-case {{ os_env.TEST_CASE }}  --timers

scripts:
  - name: "main"
    script_template: |
      {{ command.template }}
    parser:
      file: "{{ timers_file }}"
      metrics:
        - name: total_time
        - name: l1_norm

      parser_script: |
        import os, subprocess, sys
        def parse(file_path: str):
            """Parse output and extract duration metric."""
            with open(file_path, "r") as f:
                total_time = float(next(line.split()[4] for line in f if "total runtime" in line))

                # Now calculate the L1 norm
                control_dir = os.environ["BENDI_APP_CONTROL_DIR"]  # raises KeyError if missing
                script_path = os.path.join(control_dir, "collect_L1_norm.sh")
```

Work in Progress (2/2)