The Journal of Open Source Software

ReProspect - A framework for reproducible prospecting of CUDA applications

Romin Tomasetti [1*¶] **and Maarten Arnst** [1*]

1 University of Liège, Belgium ¶ Corresponding author * These authors contributed equally.

DOI: 10.21105/joss.10067

Software

- Review
- Repository
- Archive

Editor: Neea Rusch

Reviewers:

- @cedricchevalier19
- @mhoemmen

Submitted: 31 December 2025
Published: 15 September 2026

Summary

`ReProspect` is a Python framework for prospecting CUDA code, designed to ensure reproducibility through a fully programmatic approach. Prospecting encompasses three complementary ways of characterizing CUDA-based libraries and software components: how they interact with the CUDA runtime through API tracing, how kernels perform through kernel profiling, and how source constructs translate into machine code through binary analysis.

`ReProspect` builds on NVIDIA tools: Nsight Systems, Nsight Compute, and the CUDA binary utilities. It streamlines data collection and extraction using these tools, and it complements them with new functionalities for programmatic analysis of these data, thus making it possible to encapsulate the entire prospecting analysis in a single Python script.

Statement of need

HPC software development strives to achieve performance and sustain it over time as hardware and software continue to evolve. Yet the modern programming landscape relies on complex software stacks and compiler toolchains that make it increasingly difficult to reason about code behavior. Developers therefore need tools that support continuous assessment of their code and help them unravel the implications of their design decisions and changes.

Beyond *ad hoc* investigations, fully programmatic tools allow analyses to be integrated across the development cycle and open a range of new use cases. For instance, they support collaborative development by enabling developers to share concise, reproducible analyses that motivate design decisions and help reviewers grasp the impact of proposed changes. They also enable new types of tests in CI/CD pipelines that go beyond traditional output-correctness validation — such as confirming expected API call sequences for key library functionalities, verifying memory traffic for custom data layouts via kernel profiling, or validating the presence of specific instruction patterns in machine code. Finally, they can act as a framework for structuring research artifacts and documenting analyses, enabling others to reproduce and build upon prior work more effectively.

For the CUDA stack, NVIDIA provides a set of proprietary tools guaranteed to be up-to-date with their software and hardware. The runtime analysis tools Nsight Systems (NVIDIA Corporation, 2026d) and Nsight Compute (NVIDIA Corporation, 2026c) are designed for API tracing and kernel profiling, respectively. They both provide a GUI for exploring the results, as well as a low-level Python API for accessing the raw data. The CUDA binary utilities (NVIDIA Corporation, 2026a) provide command-line access to machine code (SASS or PTX (NVIDIA Corporation, 2026f)) and other information embedded in the binaries. However, while these tools allow raw data to be extracted, they do not themselves provide the infrastructure for effective programmatic analysis.

There is thus a need for a framework that builds on the NVIDIA tools and augments them to enable fully programmatic analysis. `RePrograspect` addresses this need and is designed to support the aforementioned new use cases. It allows developers to write well-structured, concise Python scripts focused on analysis results. It structures the collected data to be readily amenable to test assertions, and introduces new capabilities such as matchers for instruction sequence patterns in machine code extracted from binaries.

State of the field

Several well-established open-source tools are already available. Caliper (Boehme et al., 2016) can intercept CUDA API calls through the NVIDIA CUPTI library (NVIDIA Corporation, 2026b). It can interface with the Python package Hatchet (Bhatele et al., 2019) to organize results into a hierarchical data structure. Thicket (Brink et al., 2023) adds kernel profiling support through Nsight Compute, with a primary focus on exploratory data analysis of multi-run performance experiments. HPCToolkit (Zhou et al., 2021) is another comprehensive suite designed for large-scale parallel systems. It includes CUDA API tracing through CUPTI and kernel profiling through PAPI (Terpstra et al., 2010), and it has binary analysis capabilities to attribute performance data to calling contexts. It has a visual interface, and it can output raw performance data for programmatic analysis, e.g., using Hatchet. Score-P (Knüpfer et al., 2012) integrates multiple performance analysis tools in a common infrastructure. It can record CUDA API calls and GPU activities through CUPTI and provides standardized data formats. By contrast, `RePrograspect` focuses on concise programmatic analysis of individual units of functionality based on the outputs of NVIDIA tools.

Several tools that operate on SASS embedded in CUDA binaries already exist, each with a distinct focus. CuAssembler (Cloudcores, 2023) and CuAsmRL (He & Yoneki, 2025) focus on modifying the SASS code. NVBit (Villa et al., 2019) is a dynamic instrumentation library for CUDA binaries. However, none of these tools provides a SASS matching framework.

Miasm (Desclaux, 2012) is a Python reverse-engineering framework for CPU assembly: it can lift instructions into an intermediate representation (IR) to support higher-level analysis and binary modification. The Miasm expressions are conceptually close to the `RePrograspect` matching framework. However, CPU assembly differs significantly from SASS, which depends on the architecture generation and embeds more information, such as scheduling control codes, thread predicates, and memory space specifiers.

Software design

`RePrograspect` is organized into three main components: API tracing, kernel profiling, and binary analysis (Figure 1).

Each component is designed to allow the entire analysis to be encapsulated into a concise Python script. This includes launching the underlying analysis tool, collecting the output into Python data structures, and performing the subsequent analysis.

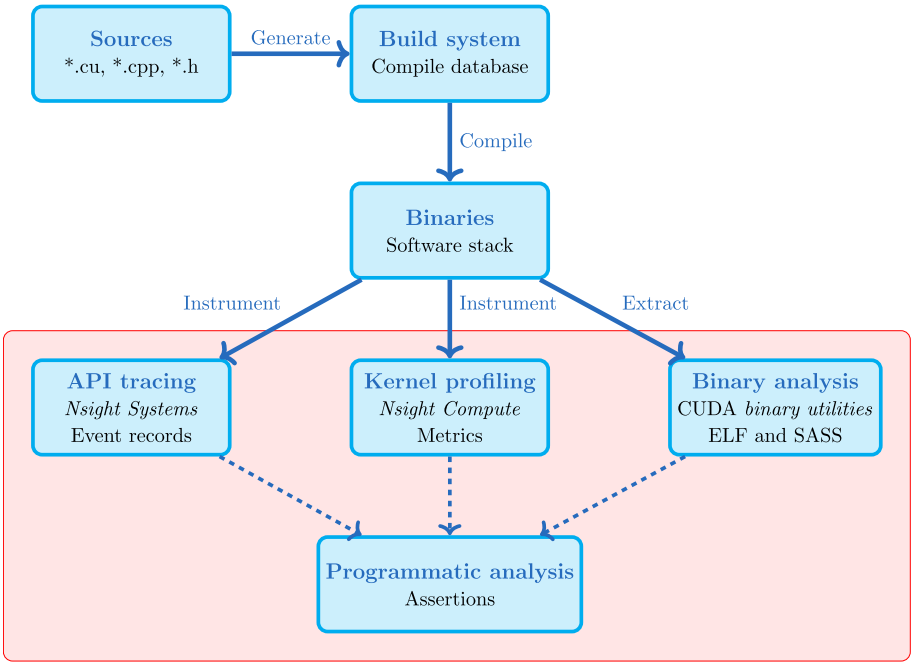

Figure 1: Overview of `ReProspect`.

API tracing and kernel profiling

The `ReProspect` `Command` and `Session` classes streamline launching Nsight Systems and Nsight Compute to collect a focused set of metrics most relevant for the analysis. The collected data are gathered in a `Report`, queryable by NVTX range annotations (NVIDIA Corporation, 2026e), readily amenable to test assertions.

To avoid unnecessary re-runs, `ReProspect` provides a `Cacher` that can serve the `Report` from a database.

The following snippet illustrates how the design allows a kernel profiling analysis to be scripted concisely, up to a test assertion—here, that the global stores of a kernel that fills a buffer with values of type `char` are fully coalesced:

```
metrics = (*L1TEXCache.GlobalStore.Instructions.create(),
           *L1TEXCache.GlobalStore.Requests.create(),
           *L1TEXCache.GlobalStore.Sectors.create())

nc = Session(command=Command(executable=executable, metrics=metrics,
                             output=workdir / executable.name,
                             nvtx_includes=('domain@start_end_range',)))
nc.run(cwd=workdir)

results = Report(command=nc.command).extract_results_in_range(metrics=metrics)
_, m = results.query_single_next_metrics(accessors=('kernel_range',))
assert m['L1/TEX cache global store requests'] \
    == m['L1/TEX cache global store sass instructions']
assert m['L1/TEX cache global store sectors'] \
    == m['L1/TEX cache global store requests']
```

The same design serves API tracing: the corresponding `Report` retrieves event records from the relational database generated by Nsight Systems, with accessors for selecting event records by NVTX range and for correlating them with data across database tables.

Binary analysis

`ReProspect` provides a set of tools for extracting and analysing the content of CUDA binaries.

The `CuObjDump` and `NVDisasm` classes drive and parse the output of the underlying CUDA binary utilities to retrieve the SASS code and resource usage of kernels (e.g., registers, constant memory).

The `ELF` class decodes ELF-formatted sections to extract complementary information, including the symbol table, toolchain metadata (from the *.note.nv.tkinfo* section), and kernel attributes such as launch bounds (from the *.nv.info.<kernel>* section) (Hayes et al., 2019).

Beyond data extraction, `ReProspect` provides an extensible framework for inspecting machine code for expected instruction patterns. This framework is structured as a hierarchy of matchers. At the lowest levels, matchers analyse instructions and their components (opcodes, modifiers, and operands). These matchers can be composed into instruction sequence matchers, enabling the identification of more intricate patterns.

One of the key challenges with SASS matching is the evolution of the CUDA instruction set and compiler toolchains. `ReProspect` addresses this challenge by abstracting away architecture- and toolchain-specific details at each level of the matcher hierarchy. For example, `AddressMatcher` matches a memory address operand, adjusting the expected address format for the target architecture. Then, at the instruction level, `LoadMatcher` matches loads from memory; it uses `AddressMatcher` for the memory address and thus needs only to adjust the opcode and modifiers for the target architecture. By extending this hierarchical design to instruction sequence and basic block matchers, `ReProspect` enables robust, composable matching across CUDA architectures and compiler toolchains.

The following snippet illustrates how matchers can be composed to assert the presence or absence of a 16-bit floating-point code path, e.g., in the SASS codes in Table 1:

```
arch = NVIDIAArch(...)
instructions = Decoder(...)
cfg = ControlFlow.analyze(instructions)

matcher_ldg = instructions_contain(instructions_are(
    LoadGlobalMatcher(arch, size=16, extend='U', readonly=False),
    LoadGlobalMatcher(arch, size=16, extend='U', readonly=False),
))
blk, matched_ldg = BasicBlockMatcher(matcher_ldg).match(cfg=cfg)

matcher_hmnmx2 = instructions_contain(instruction_is(
    Fp16MinMaxMatcher(pmax=True))
    .with_operand(index=1, operand=f'{matched_ldg[0].operands[0]}.H0_H0')
    .with_operand(index=2, operand=f'{matched_ldg[1].operands[0]}.H0_H0')
    .with_operand(index=0, operand=RegisterMatcher(special=False))
)
matched_hmnmx2 = matcher_hmnmx2.match(blk.instructions[matcher_ldg.next_index:])
```

Table 1: Comparison of the SASS code generated for the `sm_100` architecture for the 16-bit `__half` (left) *vs* 32-bit `float` (right) maximum function.

`__hmax(const __half, const __half)`	`fmax(const float, const float)`
`LDG.E.U16 R2, desc[UR6][R2.64]`	`LDG.E.U16 R2, desc[UR6][R2.64]`
`LDG.E.U16 R5, desc[UR6][R4.64]`	`LDG.E.U16 R4, desc[UR6][R4.64]`
`...`	`...`
`HMNMX2 R5, R2.H0_H0, R5.H0_H0, !PT`	`HADD2.F32 R6, -RZ, R2.H0_H0`
`...`	`HADD2.F32 R7, -RZ, R4.H0_H0`
	`FMNMX R6, R6, R7, !PT`
	`F2FP.F16.F32.PACK_AB R3, RZ, R6`
	`...`
`STG.E.U16 desc[UR6][R6.64], R5`	`STG.E.U16 desc[UR6][R6.64], R3`

The binary analysis targeting SASS rather than PTX is a trade-off. PTX is fully documented (NVIDIA Corporation, 2026f) and architecture-independent, but it is an intermediate representation. SASS is the machine code the hardware executes; assertions on SASS thus characterize the final instructions and their scheduling. Inspecting SASS is also relevant for kernel profiling: performance metrics are directly associated with executed instructions, whereas interpreting them at a higher level requires correlation, such as through compiler-emitted line information. However, SASS is only partially documented and architecture-specific; the architecture-aware matcher hierarchy addresses architecture differences.

Research impact statement

Three getting-started examples in the documentation demonstrate the components end to end. The API tracing example programmatically verifies the sequence of runtime calls that realize a diamond-shaped dependency among four asynchronous kernel launches. The kernel profiling example analyzes the performance effect of a restructuring of a fill kernel and encodes the findings in test assertions. The binary analysis example asserts the SASS instructions to which an atomic function call compiles, with a single matcher specification that adapts across CUDA architectures. Run in CI/CD pipelines, such assertions can detect when an analysis ceases to hold as toolchains and libraries evolve.

`ReProspect` has been successfully used as a support for contributions to the open-source Kokkos library (Trott et al., 2022) and the development of an in-house finite element code built on top of the open-source Trilinos library (Mayr et al., 2026) (Arnst & Tomasetti, 2024) (Tomasetti & Arnst, 2024).

The `examples` directory contains several case studies inspired by these research efforts.

The documentation provides detailed instructions for building and running the tests and examples.

`Kokkos::View` allocation

CUDA API tracing provides insight into microbenchmarking results assessing the behavior of `Kokkos::View` allocation.

See online example and Kokkos issue.

Impact of `Kokkos::complex` alignment

Our in-house finite element code uses complex arithmetic for frequency-domain electromagnetism simulations. This example combines kernel profiling and SASS

analysis to assess how aligning `Kokkos::complex<double>` to 8 or 16 bytes impacts memory instructions and traffic.

See online example.

Dynamic dispatch for virtual functions on device

This example uses `ReProspect` to create a research artifact that reproduces the dynamic dispatch instruction pattern identified by (Zhang et al., 2021).

See online example.

Atomics with `desul`

Kokkos provides extended atomic support through the `desul` library (Trott et al., 2022), which maps atomic operations to one of several methods with varying performance. The choice of the method depends on intricate logic, and must be tested. A micro-benchmarking approach is feasible, but requires the physical device and suffers from runtime variability. Yet, the machine code already contains information about the selected code paths. This case study demonstrates how to verify which method is chosen by matching an instruction sequence pattern.

See online example.

AI usage disclosure

No AI was used for the design of the code. Alongside traditional tools such as `pylint` and `mypy`, `Claude` and `CoPilot` were used as assistants to improve implementation details of individual functions. AI helped improve the clarity of the manuscript and the documentation.

Acknowledgements

This work was supported by the Fonds de la Recherche Scientifique (F.R.S.-FNRS, Belgium) through a Research Fellowship.

References

Arnst, M., & Tomasetti, R. (2024). *Design patterns and performance analysis of polymorphism in multiphysics FE assembly on GPU*. https://hdl.handle.net/2268/314521.

Bhatele, A., Brink, S., & Gamblin, T. (2019). Hatchet: Pruning the overgrowth in parallel profiles. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. https://doi.org/10.1145/3295500.3356219

Boehme, D., Gamblin, T., Beckingsale, D., Bremer, P.-T., Gimenez, A., LeGendre, M., Pearce, O., & Schulz, M. (2016). Caliper: Performance introspection for HPC software stacks. *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 550–560. https://doi.org/10.1109/SC.2016.46

Brink, S., McKinsey, M., Boehme, D., Scully-Allison, C., Lumsden, I., Hawkins, D., Burgess, T., Lama, V., Lüttgau, J., Isaacs, K. E., Taufer, M., & Pearce, O. (2023). Thicket: Seeing the performance experiment forest for the individual run trees. *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing*, 281–293. https://doi.org/10.1145/3588195.3592989

Cloudcores. (2023). *CuAssembler: An unofficial CUDA assembler*. https://github.com/cloudcores/CuAssembler.

Desclaux, F. (2012). Miasm: Framework de reverse engineering. *Actes de La Conférence Francophone Sur Le Thème de La Sécurité de l'information*. https://www.sstic.org/media/SSTIC2012/SSTIC-actes/miasm_framework_de_reverse_engineering/SSTIC2012-Article-miasm_framework_de_reverse_engineering-desclaux_1.pdf

Hayes, A. B., Hua, F., Huang, J., Chen, Y., & Zhang, E. Z. (2019). *Decoding CUDA binary - file format*. Zenodo. https://doi.org/10.5281/zenodo.2339027

He, G., & Yoneki, E. (2025). CuAsmRL: Optimizing GPU SASS schedules via deep reinforcement learning. *Proceedings of the 23rd ACM/IEEE International Symposium on Code Generation and Optimization*, 493–506. https://doi.org/10.1145/3696443.3708943

Knüpfer, A., Rössel, C., Mey, D. an, Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A., Nagel, W. E., Oleynik, Y., Philippen, P., Saviankou, P., Schmidl, D., Shende, S., Tschüter, R., Wagner, M., Wesarg, B., & Wolf, F. (2012). Score-P: A joint performance measurement run-time infrastructure for Periscope, Scalasca, TAU, and Vampir. In H. Brunst, M. S. Müller, W. E. Nagel, & M. M. Resch (Eds.), *Tools for High Performance Computing 2011* (pp. 79–91). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-31476-6_7

Mayr, M., Heinlein, A., Glusa, C., Rajamanickam, S., Arnst, M., Bartlett, R., Berger-Vergiat, L., Boman, E., Devine, K., Harper, G., Heroux, M., Hoemmen, M., Hu, J., Kelley, B., Kim, K., Kouri, D. P., Kuberry, P., Liegeois, K., Ober, C. C., … Yamazaki, I. (2026). Trilinos: Enabling scientific computing across diverse hardware architectures at scale. *ACM Transactions on Mathematical Software*. https://doi.org/10.1145/3802822

NVIDIA Corporation. (2026a). *CUDA Binary Utilities Version 13.3*. https://docs.nvidia.com/cuda/archive/13.3.0/cuda-binary-utilities/index.html.

NVIDIA Corporation. (2026b). *NVIDIA CUDA Profiling Tools Interface (CUPTI) - CUDA Toolkit Version 13.3*. https://developer.nvidia.com/cupti-ctk13_3.

NVIDIA Corporation. (2026c). *NVIDIA Nsight Compute Version v2026.2.1*. https://archive.docs.nvidia.com/nsight-compute/2026.2/index.html.

NVIDIA Corporation. (2026d). *NVIDIA Nsight Systems Version v2026.1*. https://archive.docs.nvidia.com/nsight-systems/2026.1/index.html.

NVIDIA Corporation. (2026e). *NVTX: NVIDIA Tools Extension Library Version v3.5.0*. https://github.com/NVIDIA/NVTX.

NVIDIA Corporation. (2026f). *Parallel Thread Execution ISA Version 9.3*. https://docs.nvidia.com/cuda/archive/13.3.0/parallel-thread-execution/index.html.

Terpstra, D., Jagode, H., You, H., & Dongarra, J. (2010). Collecting performance data with PAPI-C. In M. S. Müller, M. M. Resch, A. Schulz, & W. E. Nagel (Eds.), *Tools for High Performance Computing 2009* (pp. 157–173). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-11261-4_11

Tomasetti, R., & Arnst, M. (2024). *Efficiently implementing FE boundary conditions using stream-orchestrated execution on GPU*. https://hdl.handle.net/2268/314520.

Trott, C. R., Lebrun-Grandié, D., Arndt, D., Ciesko, J., Dang, V., Ellingwood, N., Gayatri, R., Harvey, E., Hollman, D. S., Ibanez, D., Liber, N., Madsen, J., Miles, J., Poliakoff, D., Powell, A., Rajamanickam, S., Simberg, M., Sunderland, D., Turcksin, B., & Wilke, J. (2022). Kokkos 3: Programming model extensions for the exascale era. *IEEE Transactions on Parallel and Distributed Systems*, *33*(4), 805–817. https://doi.org/10.1109/TPDS.2021.3097283

Villa, O., Stephenson, M., Nellans, D., & Keckler, S. W. (2019). NVBit: A dynamic binary instrumentation framework for NVIDIA GPUs. *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 372–383. https://doi.org/10.1145/3352460.

3358307

Zhang, M., Alawneh, A., & Rogers, T. G. (2021). Judging a type by its pointer: Optimizing GPU virtual functions. *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. https://doi.org/10.1145/3445814.3446734

Zhou, K., Adhianto, L., Anderson, J., Cherian, A., Grubisic, D., Krentel, M., Liu, Y., Meng, X., & Mellor-Crummey, J. (2021). Measurement and analysis of GPU-accelerated applications with HPCToolkit. *Parallel Computing*, *108*, 102837. https://doi.org/10.1016/j.parco.2021.102837