Project Large-scale Data Engineering (LDE) Kick-off Meeting

Dr.-Ing. Patrick Damme
Technische Universität Berlin
Berlin Institute for the Foundations of Learning and Data
Big Data Engineering (DAMS Lab)

Last update: Oct 12, 2025

Announcements/Org

- **Hybrid Setting with Optional Attendance**
  - In-person in MAR 0.015
  - Virtual via zoom
    https://tu-berlin.zoom-x.de/j/67376691490?pwd=NmlvWTM5VUVWRjU0UGI2bXhBVkxzQT09

About Me

- **Since 10/2022: Postdoc at TU Berlin**, Germany
  - FG Big Data Engineering (DAMS Lab) headed by Prof. Matthias Böhm
  - Continuing work on integrated data analysis pipelines
  - Research interests in the fields of database and ML systems (especially compiler & runtime techniques, extensibility)

DAPHNE

- **2021-2022: Postdoc at TU Graz & Know-Center GmbH**, Austria
  - Data Management group headed by Prof. Matthias Böhm
  - Started work on integrated data analysis pipelines

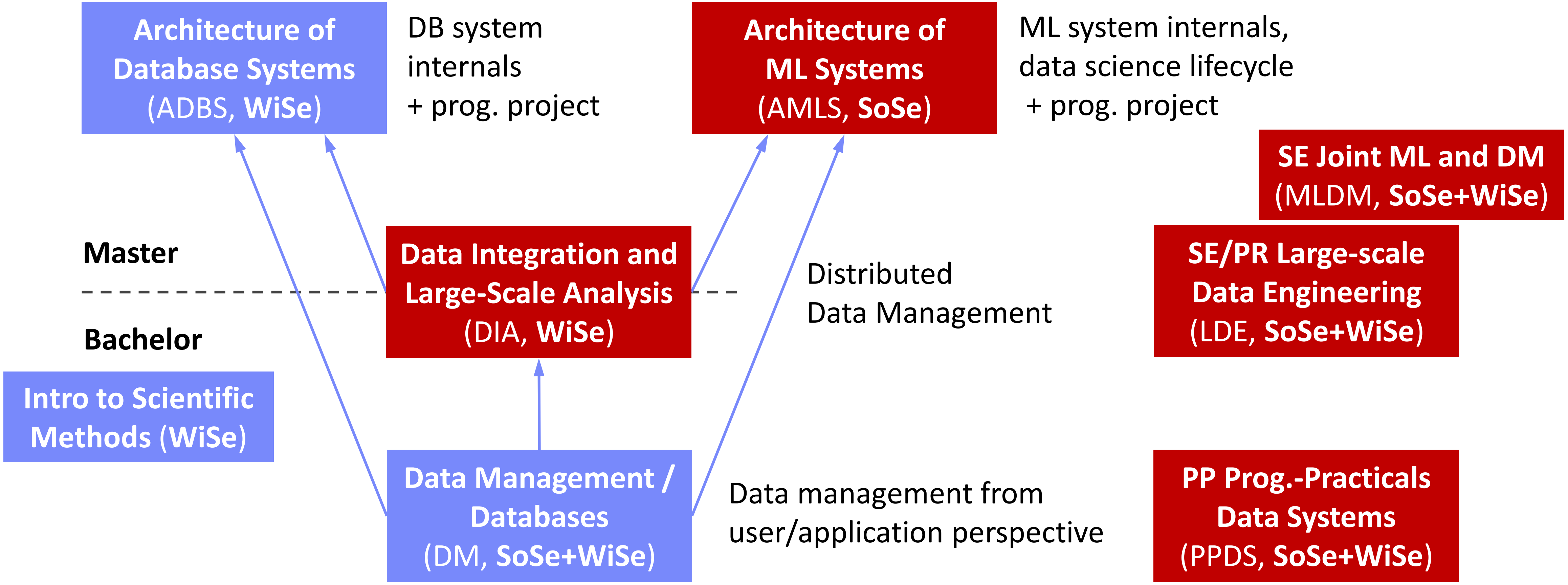
DAPHNE

- **2015-2020: PhD student at TU Dresden**, Germany
  - Dresden Database Research Group headed by Prof. Wolfgang Lehner
  - PhD thesis on making complex analytical database queries more efficient through lightweight compression of intermediate results

MorphStore & TVL

FG Big Data Engineering (DAMS Lab) – Teaching

Successfully Established TUB Teaching Portfolio (modules, slides)



Architecture of Database Systems (ADBS, **WiSe**)

DB system internals + prog. project

Architecture of ML Systems (AMLS, **SoSe**)

ML system internals, data science lifecycle + prog. project

SE Joint ML and DM (MLDM, **SoSe+WiSe**)

Master

Bachelor

Data Integration and Large-Scale Analysis (DIA, **WiSe**)

Distributed Data Management

SE/PR Large-scale Data Engineering (LDE, **SoSe+WiSe**)

Intro to Scientific Methods (WiSe)

Data Management / Databases (DM, **SoSe+WiSe**)

Data management from user/application perspective

PP Prog.-Practicals Data Systems (PPDS, **SoSe+WiSe**)

Agenda

- **Course Organization, Outline, and Deliverables**
- **Projects in DAPHNE and Apache SystemDS**
- **How to Approach the Project**
- **List of Project Topics (Proposals)**

Course Organization, Outline, and Deliverables

Large-scale Data Engineering: Module Overview

20 seats in total

bachelor + master

#41086: LDE Seminar + Project (12 ECTS) — 11 students

9 students — #41095: Seminar LDE (3)

#41183: Project LDE (9 ECTS) — 8 students

bachelor-only

bachelor-only

Mon, 14:00-16:00
MAR 0.015 & zoom

Seminar LDE

- Reading & writing scientific papers
- Giving presentations on papers
- Summary paper
- Presentation
- Lecturer & seminar mentor

Project LDE

- Building & evaluating prototypes
- Giving presentations on prototypes
- Prototype design/impl/tests/doc/eval
- Presentation
- Project mentors

Mon, 16:00-18:00
MAR 0.015 & zoom

→ In the context of systems for data engineering, data management, machine learning

→ In combination: Ideal preparation for a bachelor/master thesis with our group

Course Organization

- **General Contact Person**
  - Dr.-Ing. Patrick Damme (patrick.damme@tu-berlin.de)
- **Course Website**
  - https://pdamme.github.io/teaching/2025-26_winter/lde/lde_winter2025-26.html
  - One site for seminar and project
  - All material, schedule, **deadlines**
- **ISIS course**
  - https://isis.tu-berlin.de/course/view.php?id=44129
  - Announcements, discussion forum, topic selection poll, submission of summary paper and presentation slides
- **Language**
  - Lectures and slides: **English**
  - Communication: **English**/**German**
  - Submitted paper and presentation: **English**
  - **Informal language** (first name is fine), immediate feedback is welcome

Semester Schedule & Deadlines

- **Kick-off Meeting Oct 13** (optional)
- **Recommended Introductory Lecture** (optional)
  - Oct 27, 14:00: Experiments, Reproducibility, and Giving Presentations
- **Self-organized Project Work**
  - Consultation hours for any questions (optional)
- **Intermediate Presentations** (prerequisite)
  - Jan 19, 16:00-18:00, MAR 0.009: All students
- **Final Presentations** (mandatory)
  - Feb 23, 14:00-18:00, MAR 0.009: All students

- **List of Project Topics**
  - Presented today, take your time to select afterwards
- **Topic Selection**
  - **Deadline: Oct 31, 23:59** (in 2½ weeks)
  - Ranked list of **5 topics** via poll on the ISIS course + pref on individual/team work [+ team members]
  - Global topic assignment based on preferences
  - **Notification of assigned topics: Nov 10** (in 4 weeks)
- **Submission of Initial Prototype** (prerequisite)
  - Implementation and tests
  - **Deadline: Jan 18, 23:59** (in 14 weeks)
  - As a pull request on GitHub (exceptionally by email)
- **Submission of Final Prototype** (mandatory)
  - Implementation, tests, docs, experiments
  - **Deadline: Feb 16, 23:59** (in 18 weeks)
  - As a pull request on GitHub (exceptionally by email)
- **Submission of Pres. Slides (Intermediate & Final Pres.)**
  - **Deadline: The day before the presentation, 23:59**
  - Upload PDF in the ISIS course

Project Deliverables: Initial Prototype & Intermediate Presentation

- **Introduced in Response to Students' Feedback (Course Evaluation)**
- **Initial Prototype**
  - 80% functionally complete prototype including good set of test cases
  - Basis for further improvements driven by experiments and feedback
- **Intermediate Presentation**
  - Slide presentation of 5-10 min per individual/team
  - Briefly **present the problem** you work on
  - Give an **overview of your initial prototype** (concepts and crucial changes to the code base)
  - Outline your **planned experiments**
  - Should be the result of **prior discussions with your project mentor**

Ungraded Prerequisites for the Portfolio Exam
to be allowed make mistakes and learn from them

- **Benefits for You**
  - **Improved time management** (retain enough time for experiments)
  - **Exchange with the other students** in the project
  - **Get feedback** by project mentors and other students for improving the **quality of your prototype**

Project Deliverables: Final Prototype & Final Presentation

- **Final Prototype**
  - 100% functionally complete prototype including good set of test cases
  - Efficiency confirmed by experiments
- **Final Presentation**
  - **Summarize the problem** and give an **overview of your final prototype**
  - Present your **experimental results**
  - **1 student: 10 min talk + 5 min discussion = 15 min**
  - **2 students: 13 min + 7 min = 20 min**
  - **3 students: 16 min + 9 min = 25 min**
  - Audience: engage in the discussion

- **Grading**
  - **#41086 (seminar + project)**
    - Graded portfolio exam
    - 25 pts: summary paper
    - 15 pts: presentation
    - 50 pts: design/impl/tests/doc
    - 10 pts: presentation
  - **#41183 (project-only)**
    - Graded portfolio exam
    - 85 pts: implementation/tests/documentation
    - 15 pts: presentation
- **Academic Honesty / No Plagiarism** implies that use of LLMs like ChatGPT is prohibited

Portfolio Exam Registration

- **Portfolio exam registration: Nov 10 – Dec 08**
  - Binding registration in Moses/MTS
  - Including selection of seminar presentation date (first-come-first-serve)
- **Portfolio exam de-registration**
  - **Until 3 days before the first graded exam part**
    - Modules "LDE"/"Seminar LDE": until **Jan 09**
    - Module "Project LDE": until **Feb 13**
    - De-register yourself in Moses/MTS
  - **With sufficient reason: Until the day of the exam**
    - In case of sickness etc.
    - Modules "LDE"/"Seminar LDE": until **Jan 12/Jan 26/Feb 02**
    - Module "Project LDE": until **Feb 16/Feb 23**

- **Missing deadlines/exam without de-registration**
  - Zero points in the respective exam part (!)
  - **Approach us early in case of problems**
- **If you don't want to take LDE anymore**
  - Let me know asap to give students in the queue a chance to fill in

LDE Project Goals and Mindset

- **Goals**
  - **Design/implement a prototype** in DAPHNE/SystemDS
  - **AND prove it is a valuable contribution** to the system (tests, documentation, experiments)
  - **Present and defend your work** in a presentation & discussion

High-quality and convincing contribution to an open-source system

- **Focus on Methodology**
  - LDE as a preparation for a bachelor/master thesis at the DAMS Lab
- **Mindset: Be Open Learn New Things and to Work on Any Part of the System**
  - Whatever it takes to fulfill the task
  - Self-guided acquisition of required technical skills
- **Grading Criteria**
  - Design/implementation (functionality)
  - Code quality + tests + documentation
  - Experiments

Projects in DAPHNE and Apache SystemDS

Overview: Two Systems Developed at the DAMS Lab

DAPHNE

- **An Open and Extensible System Infrastructure for Integrated Data Analysis Pipelines**
  - Intersection of data management, machine learning, high-performance computing
  - Open-source (Apache v2 license) https://github.com/daphne-eu/daphne
  - Originated from DAPHNE EU-project
  - Written mostly in C++, Python, DaphneDSL
  - Since 2020 (open-source since 2022)

[Patrick Damme et al.: DAPHNE: An Open and Extensible System Infrastructure for Integrated Data Analysis Pipelines. CIDR 2022]

THE ASF Apache SystemDS™

- **A Declarative ML System for the End-to-End Data Science Lifecycle**
  - Data integration/cleaning/prep, model selection/ training/validation/debugging/deployment/scoring
  - Open-source (Apache v2 license) https://github.com/apache/systemds
  - Originated from Apache SystemML (started at IBM)
  - Written mostly in Java, Python, and DML
  - Since 2010 (open-source since 2015) (as SystemML)

[Matthias Boehm et al.: SystemDS: A Declarative Machine Learning System for the End-to-End Data Science Lifecycle. CIDR 2020]

Simplified High-level Architecture of DAPHNE and SystemDS

clustering … neural networks

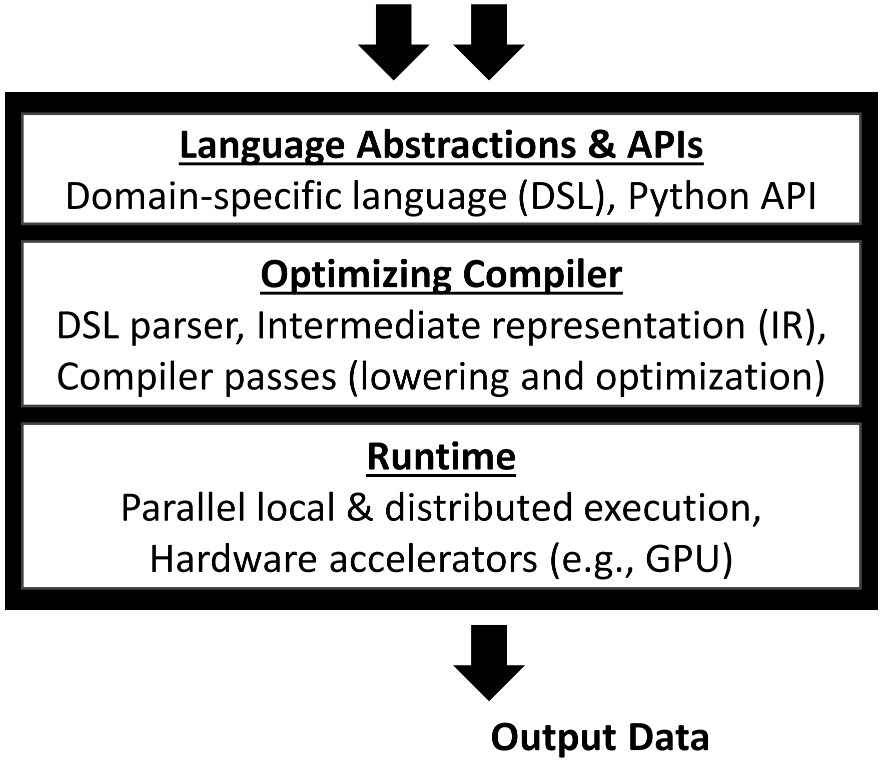
regression …

classification …

relational queries

Program
matrix/frame data types
linear/relational algebra ops
complex control flow

Input Data
open file formats
data exchange with other systems/libs



Language Abstractions & APIs
Domain-specific language (DSL), Python API

Optimizing Compiler
DSL parser, Intermediate representation (IR),
Compiler passes (lowering and optimization)

Runtime
Parallel local & distributed execution,
Hardware accelerators (e.g., GPU)

Output Data

Other such systems (examples)

NumPy DASK julia

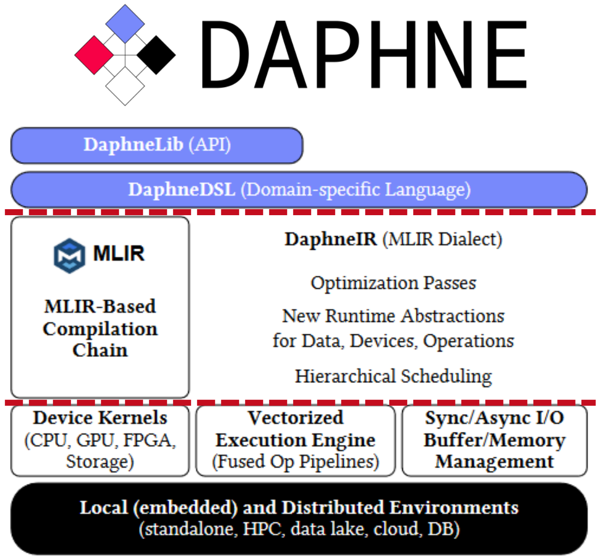
PYTORCH R TensorFlow

Architecture of ML Systems **(AMLS): Lecture + Exercise by Prof. Matthias Böhm** (every summer semester)

Simplified High-level Architecture of DAPHNE and SystemDS

Program
matrix/frame data types
linear/relational algebra ops
complex control flow

Input Data
open file formats
data exchange with other systems/libs



DAPHNE

DaphneLib (API)

DaphneDSL (Domain-specific Language)

MLIR

MLIR-Based Compilation Chain

DaphneIR (MLIR Dialect)

Optimization Passes

New Runtime Abstractions for Data, Devices, Operations

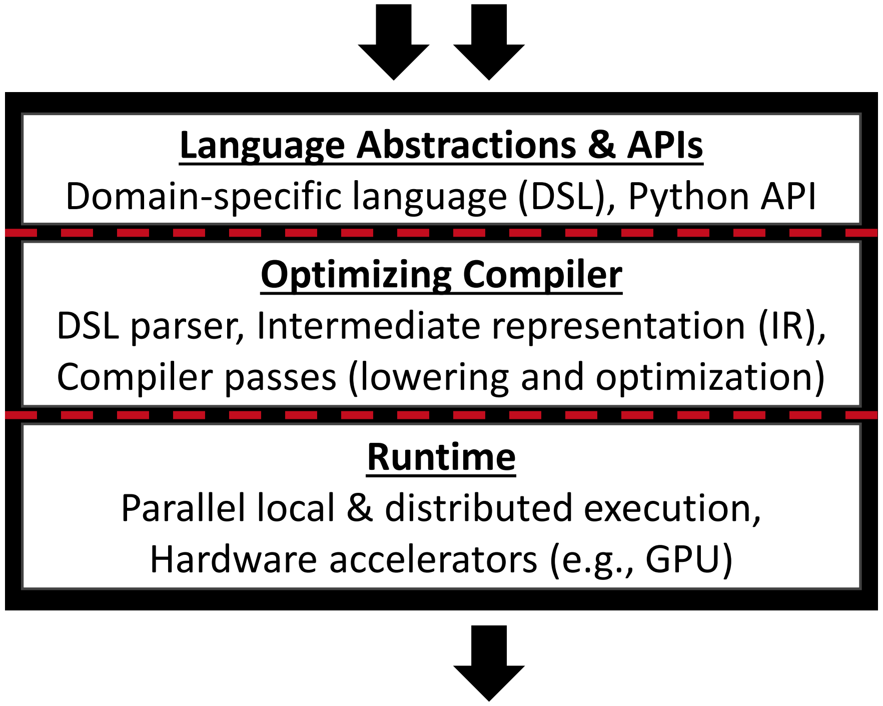
Hierarchical Scheduling

Device Kernels (CPU, GPU, FPGA, Storage)

Vectorized Execution Engine (Fused Op Pipelines)

Sync/Async I/O Buffer/Memory Management

Local (embedded) and Distributed Environments (standalone, HPC, data lake, cloud, DB)

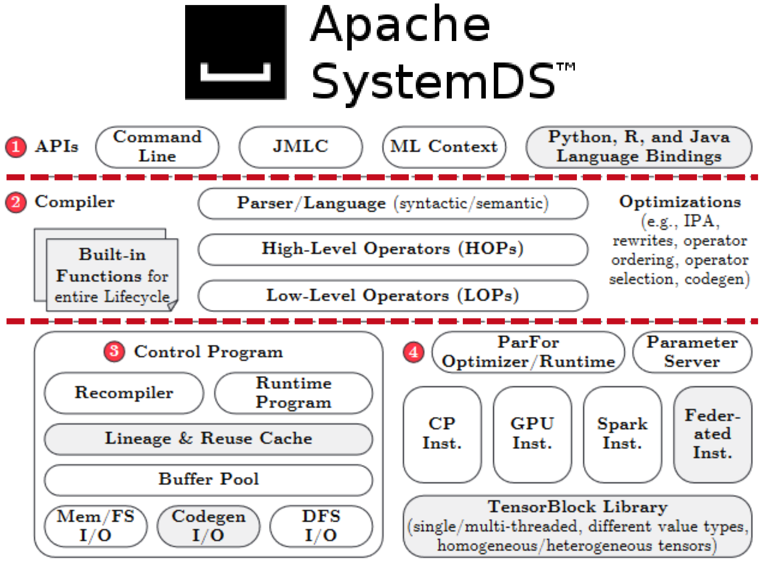


Language Abstractions & APIs
Domain-specific language (DSL), Python API

Optimizing Compiler
DSL parser, Intermediate representation (IR), Compiler passes (lowering and optimization)

Runtime
Parallel local & distributed execution, Hardware accelerators (e.g., GPU)

Output Data



Apache SystemDS™

1 APIs: Command Line | JMLC | ML Context | Python, R, and Java Language Bindings

2 Compiler: Parser/Language (syntactic/semantic) | High-Level Operators (HOPs) | Low-Level Operators (LOPs) | Built-in Functions for entire Lifecycle | Optimizations (e.g., IPA, rewrites, operator ordering, operator selection, codegen)

3 Control Program: Recompiler | Runtime Program | Lineage & Reuse Cache | Buffer Pool | Mem/FS I/O | Codegen I/O | DFS I/O

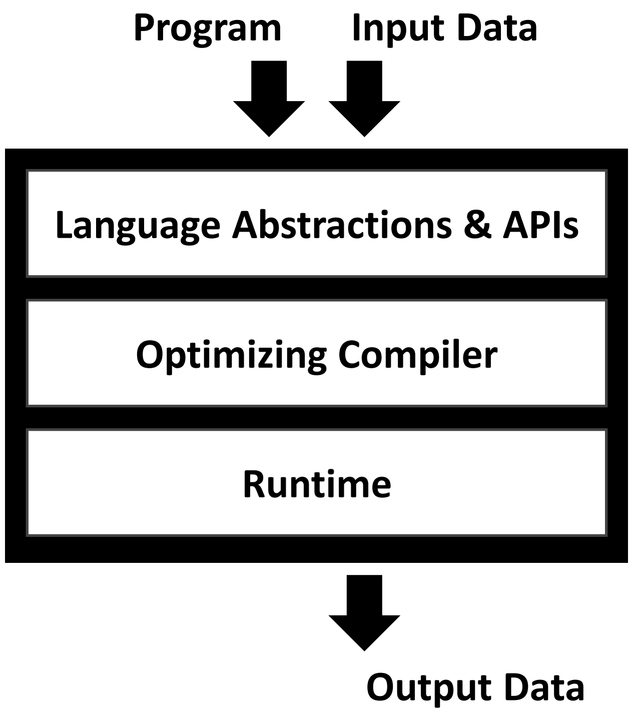
4 ParFor Optimizer/Runtime | Parameter Server | CP Inst. | GPU Inst. | Spark Inst. | Federated Inst. | TensorBlock Library (single/multi-threaded, different value types, homogeneous/heterogeneous tensors)

Kinds of LDE Project Topics

End-to-end Experiments
typically required in any project topic

External Tooling
- Helpers in the ecosystem
- Any suitable prog. lang.

Openness to All Levels of the Stack
to fulfill the task



Reusable Data Science Primitives
- Data analysis algorithms/applications
- Mostly in DaphneDSL/DML

System Internals
- Optimization and efficient execution of input programs (various aspects)
- Mostly in C++/Java, some also in Python

Choose topic based on your interests and technical skills

LDE Project Characteristics

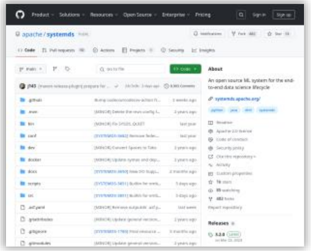
- **Individual/Team Project Work**
  - Teams of up to 3 students **strongly encouraged**
  - Unique topic for each individual/team
- **Ambitious Projects**
  - 9 ECTS (~270 h of work)
  - **≈6.75 weeks of full-time work**
- **Potential for Impact**
  - Real open issues in existing systems
  - If successful: meaningful contributions that will be used by others

- **Remarks on Topic Descriptions**
  - Many open topics in DAPHNE and SystemDS
  - Initial topic descriptions of varying level of detail
  - During topic selection: Approach project mentor directly if interested in more details
  - After topic assignment: More detailed descriptions where necessary
  - We're open to alternative topic proposals

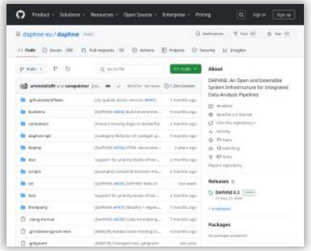
How to Approach the Project

Getting Started: Setting Up Your Development Environment

- **Goals**
  - Build the system from source
  - Successfully run the test suite
- **Navigate to the GitHub Repos**

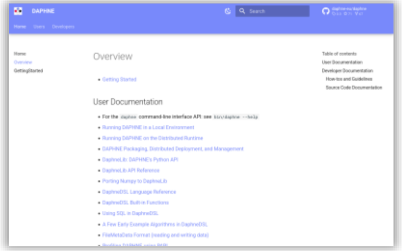


https://github.com/apache/systemds



https://github.com/daphne-eu/daphne

- **Know Where to Find the Documentation**



DAPHNE

Overview

- Getting Started

User Documentation

- For the `daphne` command-line interface API: see `bin/daphne --help`
- Running DAPHNE in a Local Environment
- Running DAPHNE on the Distributed Runtime
- DAPHNE Packaging, Distributed Deployment, and Management
- DaphneLib: DAPHNE's Python API
- DaphneLib API Reference
- Porting Numpy to DaphneLib
- DaphneDSL Language Reference
- DaphneDSL Built-in Functions
- Using SQL in DaphneDSL
- A Few Early Example Algorithms in DaphneDSL
- FileMetaData Format (reading and writing data)

Table of contents
User Documentation
Developer Documentation
How-tos and Guidelines
Source Code Documentation

https://daphne-eu.github.io/daphne/

Apache SystemDS™

SystemDS Documentation

SystemDS is a flexible, scalable machine learning system. SystemDS's distinguishing characteristics are:

1. **Algorithm customizability via R-like and Python-like languages.**
2. **Multiple execution modes**, including Spark MLContext, Spark Batch, Standalone, and JMLC.
3. **Automatic optimization** based on data and cluster characteristics to ensure both efficiency and scalability.

This version of SystemDS supports: Java 11, Python 3.5+, Hadoop 3.3.x, and Spark 3.5.x, Nvidia CUDA 10.2 (CuDNN 7.x) Intel MKL (<=2019.x).

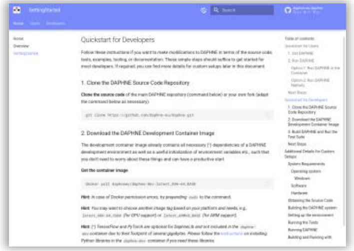
Links

Various forms of documentation for SystemDS are available.

- a DML Language Reference for an list of operations possible inside SystemDS.
- Builtin Functions contains a collection of builtin functions providing an high level abstraction on complex machine learning algorithms.
- Algorithm Reference contains specifics on algorithms supported in systemds.
- Entity Resolution provides a collection of customizable entity resolution primitives and pipelines.
- Run SystemDS contains an Helloworld example along with an environment setup guide.
- Instructions on python can be found at Python Documentation
- The JavaDOC contains internal documentation of the system source code.
- Install from Source guides through setup from git download to running system.
- If you want to contribute take a look at Contributing
- R to DML walks through the basics of converting a script from R to dml.

https://apache.github.io/systemds/

- **Clone, Build, Test According to the Documentation**



GettingStarted

Quickstart for Developers

Follow these instructions if you want to make modifications to DAPHNE in terms of the source code, tests, examples, tooling, or documentation. These simple steps should suffice to get started for most developers. If required, you can find more details for custom setups later in this document.

1. Clone the DAPHNE Source Code Repository

Clone the source code of the main DAPHNE repository (command below) or your own fork (adapt the command below as necessary).

```
git clone https://github.com/daphne-eu/daphne.git
```

2. Download the DAPHNE Development Container Image

The development container image already contains all necessary (*) dependencies of a DAPHNE development environment as well as a useful initialization of environment variables etc., such that you don't need to worry about these things and can have a productive start.

Get the container image

```
docker pull daphneeu/daphne-dev:latest_X86-64_BASE
```

Hint: In case of Docker permission errors, try prepending `sudo` to the command.

https://daphne-eu.github.io/daphne/GettingStarted/#quickstart-for-developers

Apache SystemDS™

SystemDS Install from source

This guide helps in the install and setup of SystemDS from source code.

- Install
  - Windows
  - Ubuntu 22.04
    - Testing
  - MAC
- 2. Build the project
- 3. Run A Component Test

Once the individual versions is set up skip to the common part of building the system.

Install

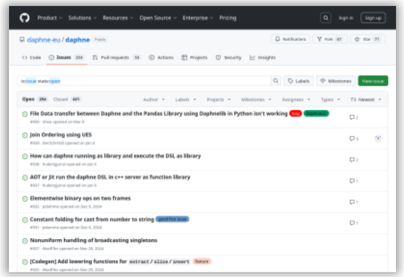
Windows

First setup java and maven to compile the system note the java version is 11, we suggest using Java OpenJDK 11.

https://apache.github.io/systemds/site/install.html

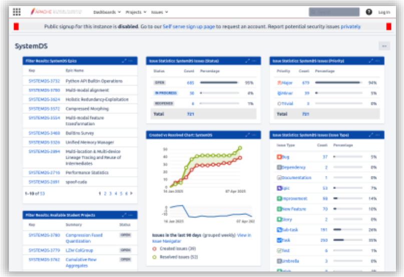
Getting Started: Preparing Your First Contribution

- **Goals**
  - Ability to modify the source code and run/test it
  - Initial overview of relevant part of the code base
- **Set Up Your Editor/IDE etc.**
- **Issue Tracking**



- File Data transfer between Daphne and the Pandas Library using Daphnelib in Python isn't working `bug` `DaphneLib`
- Join Ordering using UES
- How can daphne running as library and execute the DSL as library
- AOT or Jit run the daphne DSL in c++ server as function library
- Elementwise binary ops on two frames
- Constant folding for cast from number to string `good first issue`
- Nonuniform handling of broadcasting singletons
- [Codegen] Add lowering functions for `extract` / `slice` / `insert` `feature`

https://github.com/daphne-eu/daphne/issues



Public signup for this instance is **disabled**. Go to our Self serve sign up page to request an account. Report potential security issues privately

SystemDS

Filter Results: SystemDS Epics

Key	Epic Name
SYSTEMDS-3732	Python API Builtin Operations
SYSTEMDS-3700	Multi-modal alignment
SYSTEMDS-3624	Holistic Redundancy-Exploitation
SYSTEMDS-3572	Compressed Morphing
SYSTEMDS-3554	Multi-modal feature transformation
SYSTEMDS-3460	Builtins Survey
SYSTEMDS-3326	Unified Memory Manager
SYSTEMDS-2894	Multi-location & Multi-device Lineage Tracing and Reuse of Intermediates
SYSTEMDS-2716	Performance Statistics
SYSTEMDS-2691	spoof-cuda

1-10 of 53 1 2 3 4 5 6

Filter Results: Available Student Projects

Key	Summary	Status
SYSTEMDS-3780	Compression Fused Quantization	OPEN
SYSTEMDS-3779	LZW ColGroup	OPEN
SYSTEMDS-3762	Cumulative Row Aggregates	OPEN

Issue Statistics: SystemDS Issues (Status)

Status	Count	Percentage
OPEN	685	95%
IN PROGRESS	30	4%
REOPENED	6	1%
Total	721	

Issue Statistics: SystemDS Issues (Priority)

Priority	Count	Percentage
Major	679	94%
Minor	39	5%
Trivial	3	0%
Total	721	

Created vs Resolved Chart: SystemDS

Issues in the last 90 days (grouped weekly) View in Issue Navigator

Created Issues (39)

Resolved Issues (52)

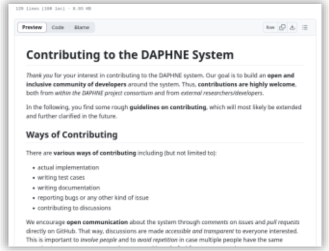
Issue Statistics: SystemDS Issues (Issue Type)

Issue Type	Count	Percentage
Bug	37	5%
Dependency	2	0%
Documentation	1	0%
Epic	53	7%
Improvement	98	14%
New Feature	70	10%
Story	2	0%
Sub-task	191	26%
Task	250	35%
Test	6	1%
Umbrella	3	0%

https://issues.apache.org/jira/secure/Dashboard.jspa?selectPageId=12335852

- **Make Your First Modifications to the Code**
  - DAPHNE: “good first issues”
  - SystemDS: Write initial test cases for your task

- **Read the Contribution Guidelines**



Contributing to the DAPHNE System

Thank you for your interest in contributing to the DAPHNE system. Our goal is to build an **open and inclusive community of developers** around the system. Thus, **contributions are highly welcome**, both from *within the DAPHNE project consortium* and from *external researchers/developers*.

In the following, you find some rough **guidelines on contributing**, which will most likely be extended and further clarified in the future.

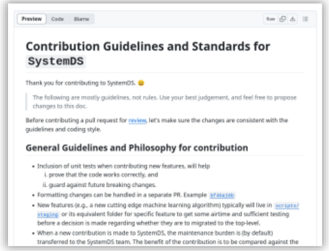
Ways of Contributing

There are **various ways of contributing** including (but not limited to):

- actual implementation
- writing test cases
- writing documentation
- reporting bugs or any other kind of issue
- contributing to discussions

We encourage **open communication** about the system through *comments on issues and pull requests* directly on GitHub. That way, discussions are made *accessible and transparent* to everyone interested. This is important to *involve people* and to *avoid repetition* in case multiple people have the same

https://github.com/daphne-eu/daphne/blob/main/CONTRIBUTING.md



Contribution Guidelines and Standards for `SystemDS`

Thank you for contributing to SystemDS. 😄

> The following are mostly guidelines, not rules. Use your best judgement, and feel free to propose changes to this doc.

Before contributing a pull request for review, let's make sure the changes are consistent with the guidelines and coding style.

General Guidelines and Philosophy for contribution

- Inclusion of unit tests when contributing new features, will help
  1. prove that the code works correctly, and
  2. guard against future breaking changes.
- Formatting changes can be handled in a separate PR. Example `bf4ba16b`
- New features (e.g., a new cutting edge machine learning algorithm) typically will live in `scripts/staging` or its equivalent folder for specific feature to get some airtime and sufficient testing before a decision is made regarding whether they are to migrated to the top-level.
- When a new contribution is made to SystemDS, the maintenance burden is (by default) transferred to the SystemDS team. The benefit of the contribution is to be compared against the

https://github.com/apache/systemds/blob/main/CONTRIBUTING.md

Initial Prototype: Design/Implementation/Tests (≈2/3 of the semester)

- **Goals**
  - 80% functionally complete prototype including good set of test cases
  - Basis for further improvements driven by experiments and feedback

- **Mindset**
  - **Understand the topic** (task description, mentor, additional material)
  - **Understand the code base** (overview and relevant parts)
  - **Understand the employed libraries/frameworks**
  - **Design and implement** step-by-step
  - Not always “the” right solution: **explore alternatives**

- **Recommendations**
  - **Start as soon as possible**
    → Don’t underestimate the ramp-up effort
  - **Actively approach your project mentor**
    → Your mentor can give you valuable guidance

Final Prototype: Creating a Convincing Contribution (≈1/3 of the semester)

- **Goals**
  - High-quality code contribution whose value can easily be appreciated and understood
- **Recommendations**
  - **Submit tidy, well documented, extensively tested code**
  - **View experiments as equally important as features**,

    → Start initial experiments as soon as possible

    → Focus primarily on investigating and improving your initial prototype after the intermediate presentation

    → Experiments show the value of your contribution

    → You need time to incorporate your insights

    Actively approach your project mentor

    → Your mentor can give you valuable guidance

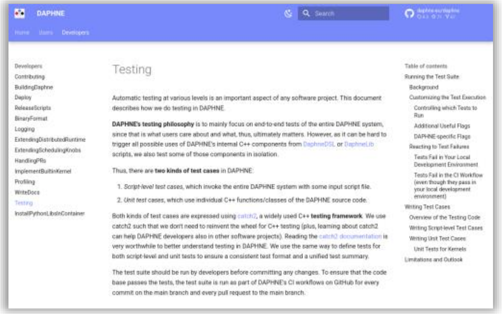
Final Prototype: Code Quality

- **Goals**
  - Make your code easy to read/understand
  - Others will have to maintain it after your contribution is merged
- **General Guidelines**
  - Clearly structure your code into **meaningful units** (classes, functions, etc.)
  - Use **clear yet concise identifiers** (variable/function/class names)
  - Stay **consistent with the existing code base** (e.g., use the same patterns) **OR refactor if necessary**
  - Adhere to the code base's **coding style/formatting**
- **Keep Your Pull Request Tidy**
  - Stay focused: **Avoid changes unrelated to your task** (can be contributed as individual small pull requests)
  - **Don't submit anything that's useless for others** (e.g., build artifacts, generated files (e.g., logs), IDE projects)
  - **Exclude specifics of your local setup**: Avoid local paths, usernames, passwords, IP addresses etc.
  - **Read your own pull request on GitHub** (changed files)

Final Prototype: Tests

- **Goals**
  - Show functionally correct behavior
  - Experiments don’t make sense
    if prototype doesn’t do what it should
- **General Hints**
  - **Unit tests** and **script-level tests**
  - **Test cases that should work**
  - **Test cases that should not work**
    (e.g., invalid DSL scripts, invalid input data, …)
  - Construct **simple and complex scenarios**
  - Think of **corner cases**
  - Small input data is often fine
    (but some bugs only triggered by large inputs)

- **Integrate with the Existing Test Suite**
  - **DAPHNE**
    → directory: `test/`
    → see the documentation on writing test cases



Testing

Automatic testing at various levels is an important aspect of any software project. This document describes how we do testing in DAPHNE.

DAPHNE's testing philosophy is to mainly focus on end-to-end tests of the entire DAPHNE system, since that is what users care about and what, thus, ultimately matters. However, as it can be hard to trigger all possible uses of DAPHNE's internal C++ components from DaphneDSL or DaphneLib scripts, we also test some of those components in isolation.

Thus, there are **two kinds of test cases** in DAPHNE:

1. *Script-level* test cases, which invoke the entire DAPHNE system with some input script file.
2. *Unit test* cases, which use individual C++ functions/classes of the DAPHNE source code.

Both kinds of test cases are expressed using catch2, a widely used C++ **testing framework**. We use catch2 such that we don't need to reinvent the wheel for C++ testing (plus, learning about catch2 can help DAPHNE developers also in other software projects). Reading the catch2 documentation is very worthwhile to better understand testing in DAPHNE. We use the same way to define tests for both script-level and unit tests to ensure a consistent test format and a unified test summary.

The test suite should be run by developers before committing any changes. To ensure that the code base passes the tests, the test suite is run as part of DAPHNE's CI workflows on GitHub for every commit on the main branch and every pull request to the main branch.

https://daphne-eu.github.io/daphne/development/Testing/

  - **SystemDS**
    → directory: `src/test/`

Final Prototype: Documentation

- **Goals**
  - Make your contribution understandable for **users** and **developers**
- **User Documentation**
  - **What’s not documented doesn’t exist**
  - Add **high-level explanation** of features and concepts behind them
  - Update documentation of **language abstractions** (e.g., new DSL built-in functions, new types, …)
  - Update documentation of **user APIs** (e.g., new command-line arguments)
- **Developer Documentation**
  - **Negative example: “The code is the documentation”**
  - **High-level explanation** of your contribution; justify design decisions
  - **API documentation** of classes, functions, members, etc.
    (integrate with the existing source code documentation of the system, e.g., doxygen or javadoc style)
  - **Comments within function bodies** (e.g., high-level steps of an algorithm)

Final Prototype: Experiments

- **Goals**
  - Understand your prototype to discover potential for improvement (e.g., performance bottlenecks)
  - Demonstrate functional improvements (new features)
  - Showcase non-functional improvements (performance compared to status quo and state-of-the-art baselines)
- **Design Experiments**
  - **Don't just conduct any random experiments**
  - **Think about which questions you want/need to answer, design experiments accordingly**
  - **Types of experiments**: Exploratory, micro benchmarks, benchmarks, end-to-end applications
  - **Aspects of an experiment**: Data, workload, baselines, hardware & software stack, metrics
- **Conduct Experiments**
  - Automate as much as possible for repeatability (shell scripts etc.)
- **Visualize and Interpret the Results**
  - Automate the visualization based on raw experimental data
  - Draw conclusions and react

See 3rd seminar intro lecture on "Experiments, Reproducibility"

List of Project Topics (Proposals)

See list at https://pdamme.github.io/teaching/2025-26_winter/lde/ProjectTopics.pdf

Summary and Q&A

- **Course Organization, Outline, and Deliverables**
- **Projects in DAPHNE and Apache SystemDS**
- **How to Approach the Project**
- **List of Project Topics (Proposals)**

- **Remaining Questions?**

- **Reminder: Seminar Introductory Lecture Recommended for the Project**
  - 03 **Experiments, Reproducibility, and Giving Presentations** [Oct 27, 14:00]

- **See you during the consultation hours and intermediate presentations** ☺