

Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 23.Б10-мм

Разработка экспериментального пайплайна PySATL-CPD: механизм хранения данных

Иванов Алексей Сергеевич

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
доцент кафедры системного программирования, к. ф.-м. н. Гориховский В. И.

Консультант:
инженер-исследователь лаборатории «YADRO» Кутуев В. А.

Санкт-Петербург
2025

Оглавление

Введение	3
1. Постановка задачи	5
2. Описание решения	6
2.1. Анализ архитектуры PySATL-CPD	6
2.2. Вывод требований	7
2.3. Общая архитектура экспериментального пайплайна . . .	8
2.4. Архитектура модуля хранения данных	10
2.5. Наивная реализация модуля хранения данных	11
2.6. Взаимодействие с экспериментальным пайплайном . . .	11
2.7. Особенности реализации	12
Заключение	14
Список литературы	15

Введение

В наше время данные стали невероятно важны во многих областях – от экономики и науки до обыденной жизни. Временные ряды — это последовательности данных, собранных в определённые моменты времени. Они помогают понимать, как меняются показатели со временем и делать прогнозы на будущее.

Иногда во временных рядах случаются неожиданные изменения, называемые «разладками», при которых поведение или структура данных внезапно меняется. Найти такие разладки очень важно, ведь они могут сильно повлиять на прогнозы и решения. Алгоритмы, способные обнаруживать разладки, используются повсюду: от медицины [3] до прогнозирования погоды [2] и распознавания речи [1].

Существует множество разных подходов к поиску разладок, и каждый из них хорош в своей ситуации. Нельзя обойтись одним методом, поэтому появилась необходимость в инструментах, которые могли бы объединить эти подходы. Python – один из самых популярных языков для анализа данных, и поэтому создание библиотеки для работы с алгоритмами обнаружения разладок на этом языке стало очень актуальным.

Учитывая, что каждый метод требует своих настроек, нужно программное окружение для тестирования и оценки того, насколько хорошо работают алгоритмы обнаружения разладок. Такая среда позволит исследователям и аналитикам данных легко проверять и сравнивать различные методы на разных наборах данных, чтобы понять их эффективность.

В прошлой работе главной целью было создание общей оболочки для модуля алгоритмов поиска разладки – PySATL-CPD ¹. В этом семестре задача расширилась. Возникла необходимость в разработке полноценной экспериментальной среды, которая позволит не просто использовать алгоритмы, но и проводить с ними полноценные эксперименты, начиная генераций данных и заканчивая формированием отчёта.

¹GitHub: <https://github.com/PySATL/PySATL-CPD-Module>

В рамках семестровой практики Артём Романюк отвечал за создание архитектуры экспериментального пайплайна, моя же работа посвящена выводу требований и разработке критически важной части этой системы: созданию модуля управления данными, отвечающего за их сохранение и загрузку. Именно эта система обеспечивает гибкость и повторяемость экспериментов, позволяя работать с разнообразными наборами данных, сохранять промежуточные результаты и легко их анализировать. В этом отчете будет описана архитектура и особенности реализации этого механизма, который является фундаментом для проведения эффективных исследований в области обнаружения разладок.

1. Постановка задачи

Целью работы является **разработка и реализация механизма хранения данных для экспериментального пайплайна PySATL-CPD**.

Для её выполнения были поставлены следующие **задачи**:

1. проанализировать архитектуру PySATL-CPD и вывести требования к разрабатываемому решению;
2. спроектировать архитектуру модуля хранения данных для экспериментального пайплайна;
3. реализовать модуль хранения данных;
 - реализовать архитектуру модуля хранения данных;
 - создать базовую реализацию модуля хранения данных;
 - написать unit-тесты для компонентов.

2. Описание решения

2.1. Анализ архитектуры PySATL-CPD

В рамках предыдущей работы была разработана оболочка для фреймворка PySATL-CPD ², призванная унифицировать взаимодействие пользователя с различными алгоритмами поиска разладок.

Основные компоненты включали:

- **CPDShell** для взаимодействия с пользователем посредством вызова интерфейсных методов;
- **LabeledCPData** для хранения размеченных данных;
- **CPContainer** для результатов выполнения алгоритма;
- **CPDResultsAnalyzer** для вычисления метрик, бенчмаркинга и генерации отчётов.

Несмотря на выполнение поставленных задач, предыдущая архитектура имела ряд ограничений, которые затрудняли её масштабирование и использование для проведения полноценных научных экспериментов.

1. Ограниченность управления данными.

Существующий класс **LabeledCPData** позволял генерировать и открывать размеченные наборы данных. Однако его функциональность была сосредоточена преимущественно на подготовке данных для одного запуска алгоритма. Отсутствовал унифицированный и гибкий механизм для сохранения и загрузки произвольных наборов данных. Не было функциональности для сохранения результатов экспериментов. Реализация своего генератора было проблематично. Это приводило к необходимости ручного управления файлами и сложностям при воспроизведении результатов.

2. Проблемы с воспроизводимостью.

Архитектура была ориентирована на единичный запуск алгорит-

²GitHub: <https://github.com/PySATL/PySATL-CPD-Module>

ма и получение результата. Не была предусмотрена чёткая структура для организации последовательных экспериментов, включающих различные этапы, такие как предобработка, запуск нескольких алгоритмов с разными параметрами и сравнение результатов. Отсутствовала возможность оперативного изменения одного или нескольких составляющих эксперимента, автоматизированного формирования отчётов по их сериям.

3. Недостаточная гибкость конфигурирования.

Конфигурация алгоритмов и скрабберов осуществлялась через **CPDShell**, но механизм не был достаточно универсальным для поддержки сложных сценариев тестирования и бенчмаркинга. Требовалось трудоёмкое улучшение интерфейса для динамического подбора или изменения параметров и сценариев работы.

4. Отсутствие поддержки пайплайнов.

Для проведения исследований часто требуется выстраивать цепочки обработки данных и применения алгоритмов (пайплайны). Предыдущая архитектура не предоставляла встроенных средств для создания и управления такими пайплайнами, что вынуждало исследователей реализовывать подобные цепочки вне фреймворка.

2.2. Вывод требований

Анализ предыдущей архитектуры выявил, что, несмотря на её функциональность в рамках первоначальных задач, она не в полной мере отвечала требованиям к полноценной экспериментальной среде для глубокого анализа алгоритмов обнаружения разладки. Для преодоления указанных ограничений, к новому решению и, в частности, к модулю управления данными, предъявляются следующие требования:

1. Модуль сохранения и загрузки данных.

Должен быть разработан гибкий механизм для сохранения и загрузки не только входных данных, но и промежуточных резуль-

татов, а также полных отчетов об экспериментах. Система должна поддерживать различные форматы данных и обеспечивать их целостность, быть гибкой и открытой для расширения и масштабирования пользователем.

2. Экспериментальный пайплайн.

Новая архитектура должна явно поддерживать этапы проведения эксперимента: генерацию данных, их обработку и формирование отчётов, реализуя паттерн «pipeline».

3. Конфигурация и управление параметрами.

Система должна предоставлять удобный и расширяемый интерфейс для конфигурирования всех этапов и параметров экспериментов.

4. Обеспечение воспроизводимости результатов.

Механизмы сохранения должны гарантировать полную воспроизводимость любых проведенных экспериментов.

2.3. Общая архитектура экспериментального пайплайна

Разработанная в данном семестре мной и Артёмом Романюком экспериментальная среда для фреймворка PySATL-CPD представляет собой модульную систему, вызов и оркестрирование которой осуществляется пайплайном. Основной целью новой архитектуры является обеспечение гибкости, воспроизводимости и масштабируемости экспериментального процесса.

Архитектура экспериментальной среды логически разделена на три основных функциональных блока, каждый из которых отвечает за свой этап жизненного цикла эксперимента и может включать от одного до нескольких шагов.

(С диаграммой компонентов можно ознакомиться на рис. 1)

- **Data generation phase.**

Отвечает за подготовку временных рядов или их генерацию с заданными параметрами и свойствами. Это позволяет исследователям моделировать различные сценарии поведения данных и вводить искусственные разлады для контроля над экспериментальными условиями. Данный этап предоставляет как готовые для работы модули, так и интерфейсы, позволяющий реализовать свои генераторы с определёнными параметрами и особенностями.

- **Test execution phase.**

Является центральным компонентом для проведения экспериментов. Этап отвечает за применение к данным необходимых действий, начиная с использования выбранных алгоритмов обнаружения разладок, подбора гиперпараметров и заканчивая подсчётом метрик, сбором промежуточных результатов и их сохранением.

- **Report generation phase.**

Предназначен для агрегации и визуализации результатов проведенных экспериментов. Он собирает выходные данные от предыдущего этапа, обрабатывает результаты и метрики и генерирует структурированный отчет. Модули этапа поддаются гибкой настройке, благодаря чему отчет может включать текстовые описания, таблицы с метриками, графики результатов работы алгоритмов и сравнения их эффективности.

Модуль-пайплайн оркестрирует модули всех этапов и позволяет удобно работать с ними, благодаря чему эксперименты можно объединять в серии, автоматизировать и удобно анализировать их результаты.

С подробностями о проектировании архитектуры экспериментальной среды и её реализации можно ознакомиться в работе Артёма Романюка.

Интеграция этих модулей осуществляется посредством модулей управления данными, которые являются ключевыми элементами для обеспечения связности и повторяемости экспериментов. **Saver** и **Loader** обеспечивают унифицированное сохранение и загрузку всех типов данных,

используемых в экспериментальном процессе: входных временных рядов (генерируемых или открываемых), промежуточных результатов работы алгоритмов и окончательных отчётов. Это позволяет гибко управлять экспериментальными данными, сохранять прогресс, возобновлять эксперименты с определенной точки и полностью воспроизводить ранее полученные результаты.

2.4. Архитектура модуля хранения данных

Основная задача модуля хранения данных — обеспечить унифицированный механизм для сохранения и загрузки данных, что критически важно для воспроизводимости и масштабируемости исследований.

Архитектура модуля базируется на концепции разделения ответственности и использовании четко определенных интерфейсов, что повышает её модульность, расширяемость и простоту. Центральными элементами являются два интерфейса: **Saver** и **Loader**.

(Диаграмму интерфейсов можно изучить на рис. 2)

- **Saver.**

Объект осуществляет сохранение данных, полученных при генерации/работе алгоритмов. Для вызова требуется передать поле, под которым сохраняются данные и сами данные.

- **Loader.**

Объект предназначен для загрузки данных, полученных при генерации/работе алгоритмов. Для вызова нужно передать множество полей, чьи данные мы хотим получить.

В зависимости от требуемого формата и типа хранения (например, файловая система, база данных, облачное хранилище) пользователь может создавать реализации, идеально подходящие под конкретные задачи. Также модульность позволяет использовать в одном эксперименте разные реализации **Saver** и **Loader**, обеспечивая максимальную гибкость.

2.5. Наивная реализация модуля хранения данных

В качестве базовой наивной реализации модуля хранения данных в рамках данной практики были реализованы **SaverCSV** и **LoaderCSV**. Данные модули работают с форматом **CSV (Comma Separated Values)**. Выбор данного формата обусловлен его простотой, читаемостью и широкой поддержкой в различных инструментах анализа данных.

По умолчанию при использовании этих модулей создастся директория **experiment_storages/csv/**, в ней создастся ещё 2 поддиректории: **generation** и **results** для двух этапов соответственно. При вызове метода сохранения каким-то шагом этапа будет создаваться файл с именем **[имя_поля]_[тип_поля].csv** в папке соответствующего этапа, в котором лежат данные. Данные разных типов будут обработаны по разному:

- Если сохраняется список, то он будет представлен в виде словаря, где ключ — это индекс в списке. Файл будет выглядеть следующим образом: **[имя_поля]_list.csv**
- Если сохраняется число или строка, то они будут представлены в виде словаря с единственным значением и ключ будет равен 0. Имя файла следующее: **[имя_поля]_literal.csv**
- Если сохраняется словарь, то приписки типа файла не будет, сохранение будет в обычном CSV формате, то есть ключ-значение. Имя файла: **[имя_поля].csv**

2.6. Взаимодействие с экспериментальным пайплайном

Разработанный модуль хранения данных интегрирован в общую архитектуру экспериментального пайплайна (представленного на рис. 3). Функционирование модуля обеспечивается посредством взаимодействия ключевых компонентов, показанных на UML-диаграмме, где **Pipeline** координирует последовательность шагов, а **Saver** и **Loader**

обеспечивают хранение данных.

Основной принцип работы заключается в том, что каждый этап эксперимента (`DataGenerationStep`, `TestExecutionStep`, `ReportGenerationStep`) использует предоставленные ему экземпляры **Saver** и **Loader** для записи и чтения данных. Это обеспечивает гибкость и модульность, позволяя легко менять способы хранения данных без изменения логики самих шагов.

Пользователю не надо самостоятельно вызывать **Saver** и **Loader**, достаточно передать нужные их реализации, а также шаги эксперимента, после чего пайплайн самостоятельно все настроит, а шаги эксперимента вызовут методы для сохранения или загрузки в нужные моменты. Таким образом, осуществляется сохранение данных между любыми шагами, предотвращая потерю результатов и обеспечивая их воспроизводимость.

2.7. Особенности реализации

По ходу работы над данной практикой возникали некоторые сложности, повлекшие за собой особенности в реализации.

- Классы **Saver** и **Loader** могут оперировать многими типами данных, а именно целыми и вещественными числами, строками, списками и словарями. При реализации стоит учитывать, что интерфейс принимает на вход все эти типы данных, и подходить к их обработке надо по разному.
- Изначально предполагалось, что данными для сохранения могут быть только словари с целочисленным ключом и вещественным значением, чтобы избежать излишней сложности с поддержкой разных систем баз данных, однако потом оказалось, что это не только не удобно, но и не выгодно. Разные алгоритмы и генераторы по умолчанию могут генерировать данные в виде множеств или возвращать единичные значения (например, ассигасы), поэтому ограничение на сохранение сугубо словарей усложнило бы взаимодействие с и так нетривиальной архитектурой. Таким образом

было принято решение предоставить возможность сохранять не только словари, но и множества со значениями, пусть это и накладывает дополнительные трудности на разработки реализаций **Saver** и **Loader**.

Заключение

Результаты работы следующие.

1. Проанализирована существующая архитектура PySATL-CPD и выделены требования к разрабатываемому решению.
(Разделы 2.1, 2.2)
2. Спроектирована архитектура модуля хранения данных для экспериментального пайплайна. (Раздел 2.4)
3. Реализован модуль хранения данных. (Разделы 2.5, 2.6)
 - Ссылка на репозиторий: <https://github.com/PySATL/pysatl-cpd/tree/wrdxwrdxwrdx/rework-architecture>
 - Логин коммитера — Sor4nex (<https://github.com/Sor4nex>)
 - Ссылки на пулл реквесты ^{3 4 5 6 7}
4. Написаны unit-тесты ко всем компонентам.

³https://github.com/wrdxwrdxwrdx/rework_cpd_architecture/pull/1

⁴https://github.com/wrdxwrdxwrdx/rework_cpd_architecture/pull/3

⁵https://github.com/wrdxwrdxwrdx/rework_cpd_architecture/pull/4

⁶<https://github.com/PySATL/pysatl-cpd/pull/51>

⁷<https://github.com/PySATL/pysatl-cpd/pull/54>

Список литературы

- [1] D. Rybach C. Gollan R. Schluter, Ney H. Audio segmentation for speech recognition using segment features // IEEE International Conference on Acoustics, Speech and Signal Processing. — 2009. — P. 4197–4200.
- [2] J. Reeves J. Chen X. L. Wang R. Lund, Lu Q. Q. A Review and Comparison of Changepoint Detection Techniques for Climate Data // J. Appl. Meteorol. Climatol., vol. 46, no. 6. — 2007. — P. 900–915.
- [3] R. Malladi G. P. Kalamangalam, Aazhang B. Online Bayesian change point detection algorithms for segmentation of epileptic activity // Asilomar Conference on Signals, Systems and Computers. — 2013. — P. 1833–1837.

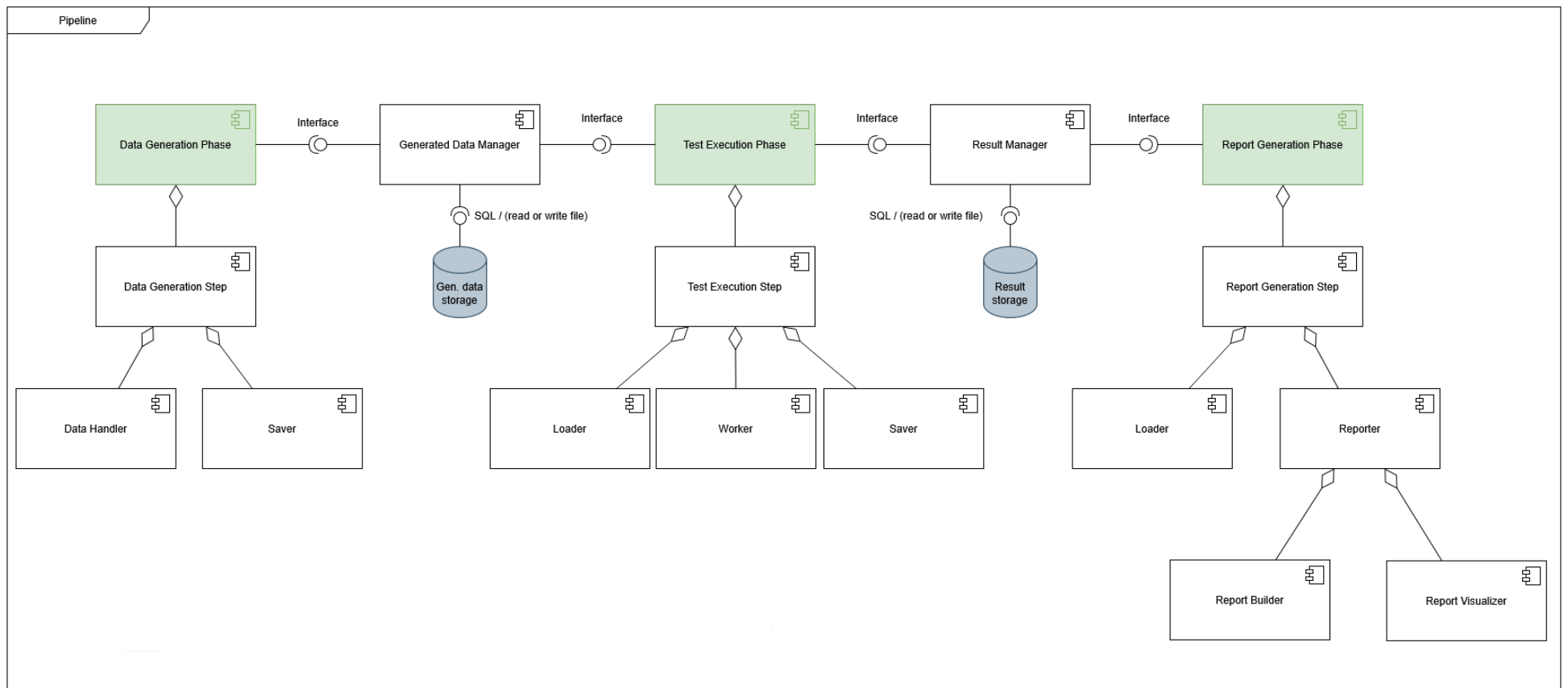


Рис. 1: Архитектура экспериментального пайплайна.

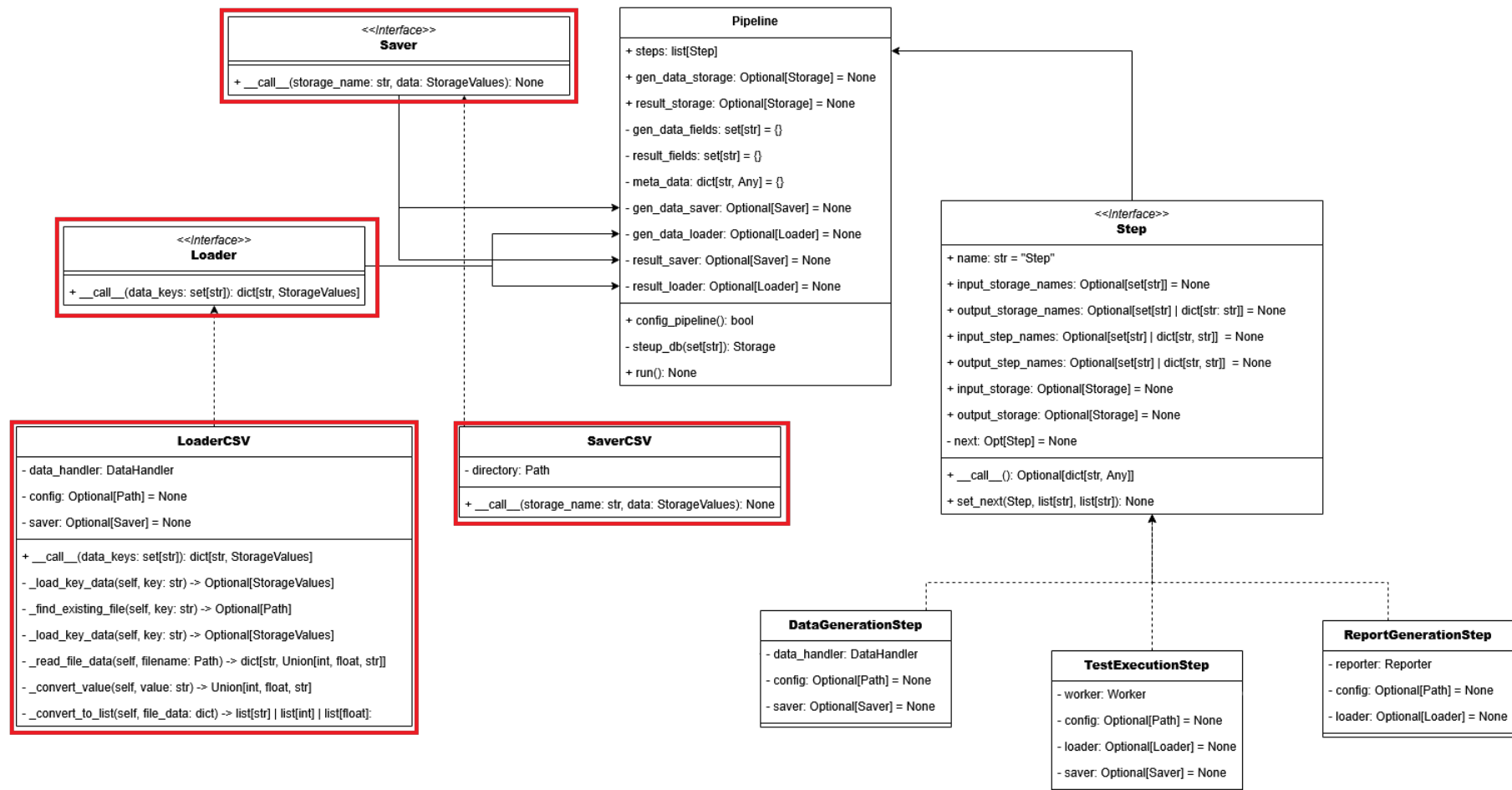


Рис. 2: Архитектура механизма хранения данных. Красным выделены классы, разработанные в рамках данной практики.