

The Kaspersky logo is displayed in a bold, lowercase, sans-serif font. It is positioned within a white, rounded rectangular area that is part of a larger graphic design. The background consists of a teal-to-green gradient with a white, organic, cloud-like shape in the center. The logo itself is black and stands out against the white background.

kaspersky

Kaspersky Neuromorphic Platform

© 2023 АО "Лаборатория Касперского"

Содержание

[О Kaspersky Neuromorphic Platform](#)

[Комплект поставки](#)

[Аппаратные и программные требования](#)

[Архитектура Kaspersky Neuromorphic Platform](#)

[О бекенде Kaspersky Neuromorphic Platform](#)

[Библиотека поддержки бекендов](#)

[Класс UID](#)

[Функция uid_hash](#)

[Класс continuously_uid_generator](#)

[Класс TagMap](#)

[Структура BaseData](#)

[Класс Population](#)

[Класс Projection](#)

[Класс Backend](#)

[Класс Device](#)

[Класс MessageBus](#)

[Класс MessageEndpoint](#)

[Класс Subscription](#)

[Пространство имен Messaging](#)

[Пространство имен synapse_access](#)

[Библиотека устройств](#)

[Библиотека характеристик нейронов](#)

[Библиотека характеристик синапсов](#)

[Библиотека шаблонов](#)

[Поддерживаемые бекенды](#)

[Обмен данными в Kaspersky Neuromorphic Platform](#)

[Архитектура фреймворка для C++](#)

[Объекты фреймворка для C++](#)

[Пространство имен input](#)

[Пространство имен output](#)

[Набор классов Coordinates](#)

[Класс MessageObserver](#)

[Класс Network](#)

[Класс Model](#)

[Класс ModelExecutor](#)

[Класс BackendLoader](#)

[Сторонние библиотеки и используемые форматы](#)

[Сторонние библиотеки](#)

[Форматы данных, обрабатываемые нейронной сетью](#)

[Форматы структуры нейронной сети](#)

[Установка и удаление платформы](#)

[Сценарий работы с исходным кодом платформы](#)

[Клонирование исходного кода платформы из репозитория](#)

[Сборка проекта из исходного кода платформы](#)

[Удаление платформы](#)

[Директории для хранения данных платформы](#)

[Добавление нового типа нейрона](#)

[Добавление нового типа синапса](#)

[Исполнение нейронной сети, загруженной на бекенд автоматически](#)

[Исполнение нейронной сети, созданной из проекций и популяций](#)

[Лицензирование платформы](#)

[Предоставление данных](#)

[Использование Kaspersky Neuromorphic Platform API](#)

[Глоссарий](#)

[Бекенд](#)

[Вес синапса](#)

[Вычислитель](#)

[Модификатор](#)

[Нейрон](#)

[Период ППС](#)

[Плотная последовательность спайков \(ППС\)](#)

[Популяция](#)

[Проекция](#)

[Синапс](#)

[Синаптическое воздействие](#)

[Соединение](#)

[Спайк](#)

[Фреймворк](#)

[Функция нейрона](#)

[Функция синапса](#)

[Информация о стороннем коде](#)

[Уведомления о товарных знаках](#)

О Kaspersky Neuromorphic Platform

Kaspersky Neuromorphic Platform (далее также платформа) – это программно-аппаратная платформа, предназначенная для эмуляции импульсных нейронных сетей и поддержки их выполнения на разных вычислителях.

С помощью Kaspersky Neuromorphic Platform вы можете:

- Создавать и обучать нейронные сети на различных типах входных данных (например, телеметрия, события, изображения, 3D-данные и звуковые и тактильные данные).
- Оптимизировать структуру и гиперпараметры загруженных нейронных сетей.
- Проводить прикладные исследования в области классификации входных данных и других областях применения импульсных нейронных сетей.
- Разрабатывать новые топологии нейронных сетей (например, импульсные аналоги сверточных нейронных сетей, предполагающих свертку по пространству и времени).
- Разрабатывать новые модели синаптической пластичности.
- Реализовывать новые модели нейронов.
- Реализовывать прикладные решения на основе нейроморфных импульсных нейронных сетей и искусственного интеллекта в области робототехнических манипуляторов, интернета вещей, беспилотных систем, человеко-машинного взаимодействия, носимых устройств и оптимизационного планирования.
- Реализовывать прикладные решения на устройствах с пониженным энергопотреблением или с использованием нейроморфных процессоров (англ. Neural Processing Unit, NPU).

Для решения вышеперечисленных задач вы можете использовать языки C++ и Python®. Платформа поддерживает CPU. Фреймворк Kaspersky Neuromorphic Platform можно также использовать как бекенд для PyNN API.

Комплект поставки

Kaspersky Neuromorphic Platform поставляется в виде файла архива, который содержит следующие файлы:

- исходный код бекендов;
- исходный код фреймворка платформы на языке программирования C++;
- исходный код фреймворка платформы на языке программирования Python;
- исходный код библиотек платформы;
- справочное руководство по API;
- файл с информацией о платформе (Release Notes);
- файл с информацией о стороннем коде legal_notices.txt на английском языке.

Вы можете [получить исходный код платформы](#) путем клонирования его репозитория.

Аппаратные и программные требования

Для функционирования Kaspersky Neuromorphic Platform компьютер должен удовлетворять следующим минимальным требованиям.

Минимальные аппаратные требования:

- центральный процессор: совместимый процессор Intel Core i5 и выше;
- количество ядер: одно и более;
- объем оперативной памяти: 8 ГБ и более.

Поддерживаемые операционные системы:

- Debian GNU/Linux 10.x (Buster) и выше;
- Ubuntu 23.04 LTS и выше.

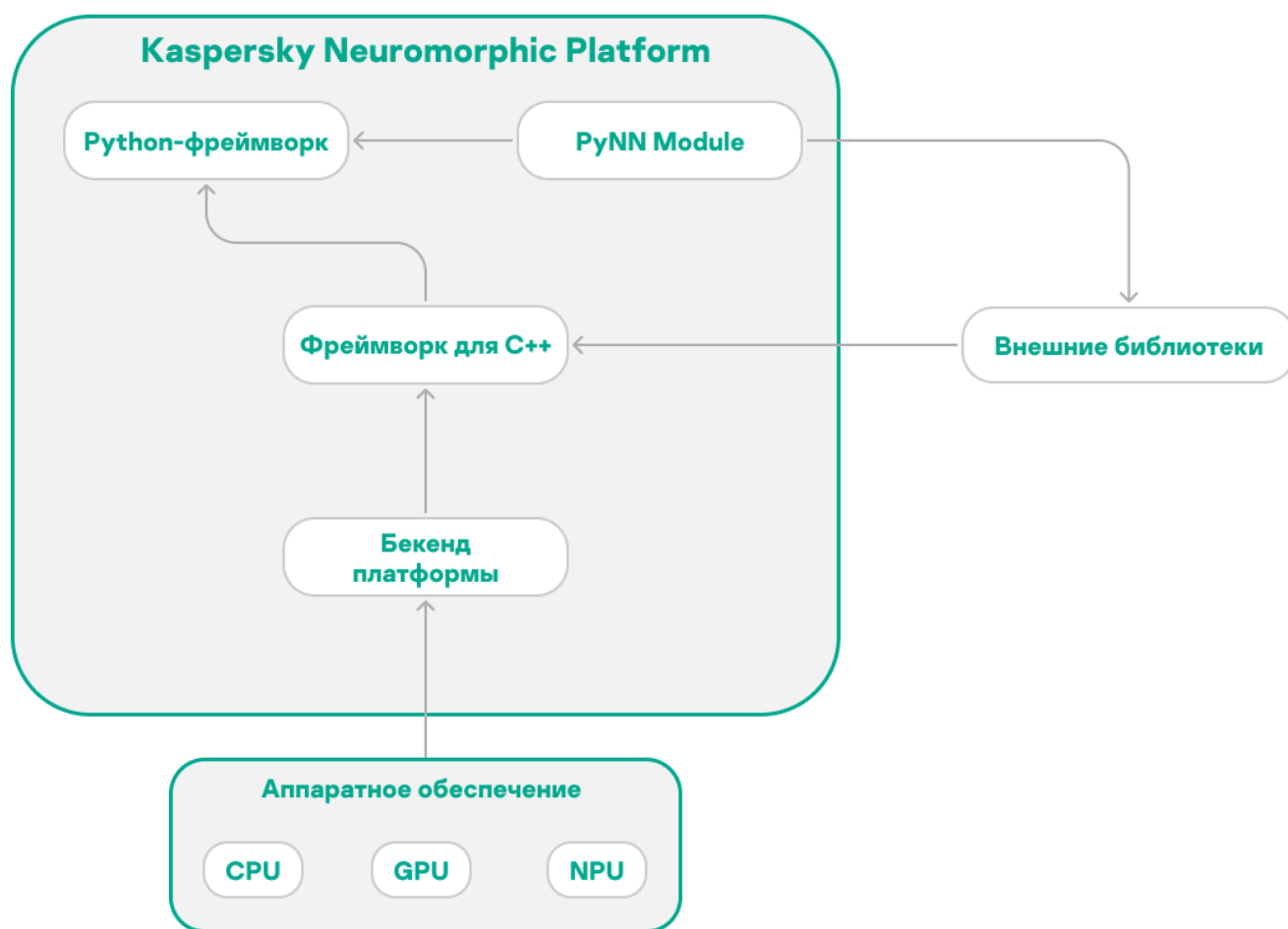
Вы можете использовать устройство под управлением любой другой операционной системы из семейства Linux, если дистрибутив операционной системы содержит библиотеку Boost версии 1.80 и выше.

Архитектура Kaspersky Neuromorphic Platform

Kaspersky Neuromorphic Platform включает в себя следующие компоненты:

- Бекенд. Запускает эмуляцию работы импульсной нейронной сети на различных вычислительных устройствах (например, центральных процессорах).
- Фреймворк для C++. Обеспечивает управление бекендами.
- *Python-фреймворк*. Обеспечивает управление бекендами.
- *PyNN Module*. Обеспечивает интерфейс к Python-фреймворку.

На рисунке ниже представлена схема взаимодействия компонентов Kaspersky Neuromorphic Platform.



Архитектура Kaspersky Neuromorphic Platform

О бекенде Kaspersky Neuromorphic Platform

Бекенд Kaspersky Neuromorphic Platform предоставляет стандартизированный интерфейс для работы с [вычислителем](#) ². Бекенд платформы позволяет запускать эмуляцию работы импульсной нейронной сети на различных вычислительных устройствах.

Физически бекенд платформы разделяется на библиотеки и компилируется в несколько независимых друг от друга бекендов. Каждый такой бекенд позволяет выполнять эмуляцию на разных типах устройств.

Бекенд Kaspersky Neuromorphic Platform выполняет следующие функции:

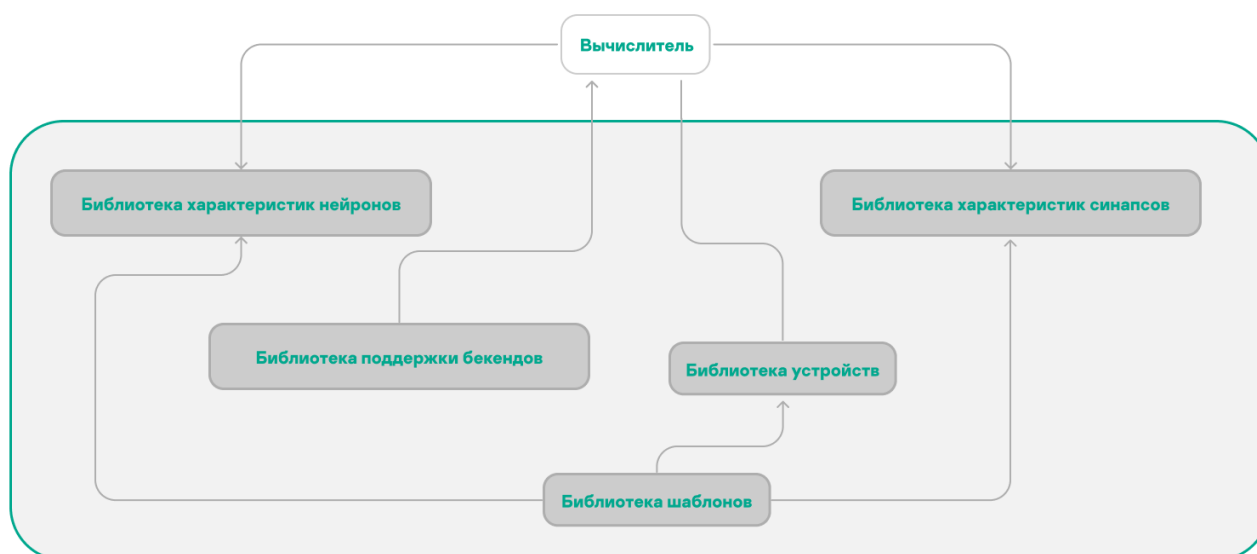
- Запускает импульсные нейронные сети на различных вычислителях.
- Выполняет статические проверки корректности построения импульсной нейронной сети.
- Предоставляет общую кодовую базу для реализации вычислителей.

Для реализации бекендов вы можете использовать следующие библиотеки платформы:

- [Библиотека поддержки бекендов](#). Реализует основные объекты и шаблоны вычислителя.
- [Библиотека устройств](#). Предоставляет набор интерфейсов для взаимодействия с физическими устройствами.
- [Библиотека характеристик нейронов](#). Содержит наборы свойств и типизацию популяций моделей нейронов [?](#).
- [Библиотека характеристик синапсов](#). Содержит набор классов, определяющих поведение [синапса](#). [?](#)
- [Библиотека шаблонов](#). Предоставляет шаблоны общего назначения.

На рисунке ниже представлена схема взаимодействия библиотек платформы.

В Kaspersky Neuromorphic Platform вычислитель хранит в себе интерфейс устройства, а также экземпляры объектов библиотеки поддержки бекендов (например, [проекции](#) [?](#) и [популяции](#) [?](#)), необходимые для выполнения прикладных задач на вычислителе. Шаблоны популяций и проекций нейронной сети должны быть специализированы типами нейронов и синапсов, описанными в библиотеках характеристик нейронов и синапсов.



Взаимодействие библиотек Kaspersky Neuromorphic Platform

Библиотека поддержки бекендов

В библиотеке поддержки бекендов реализуются интерфейсы к объектам, необходимым для работы бекендов и выполнения прикладных задач. Библиотека содержит пространство имен `core`.

Таблица ниже содержит описание объектов библиотеки.

Объект	Описание
Backend	Базовый класс для конкретных реализаций бекендов.
BaseData	Структура, определяющая набор базовых данных.
continuously_uid_generator	Класс, реализующий генератор уникальных идентификаторов.
Device	Базовый класс устройств.
MessageBus	Класс, реализующий интерфейс к шине сообщений.
MessageEndpoint	Класс, реализующий интерфейс к точке подключения.
Messaging	Пространство имен, определяющее интерфейсы к сообщениям.
Population	Шаблонный класс, реализующий контейнер нейронов ? одной модели.
Projection	Шаблонный класс, реализующий контейнер синапсов ? одного типа.
Subscription	Класс, реализующий подписку на сообщения.
synapse_access	Пространство имен, реализующее соединения и интерфейс для доступа к их реестру.
TagMap	Класс, реализующий интерфейс к словарю тегов.
UID	Класс, определяющий уникальные идентификаторы.
uid_hash	Функция, реализующая хеширование UID.

На рисунке ниже представлена схема взаимодействия объектов библиотеки.

В Kaspersky Neuromorphic Platform структура BaseData содержит базовую информацию об объекте и включает в себя структуру UID и объект класса TagMap. Экземпляр структуры BaseData используется в объектах классов Projection, Population и Backend, соответственно каждый из этих объектов обладает идентификатором UID и набором тегов и их значениями TagMap.

Объекты классов Population и Projection принимают и отправляют сообщения из пространства имен Messaging, которые содержат в своем заголовке UID отправителя. Объект класса Projection также получает информацию о соединениях из пространства имен synapse_access.

Каждый объект Backend может работать на нескольких устройствах и содержать соответствующее количество экземпляров классов Device и MessageBus. Объект MessageBus создает экземпляры класса MessageEndpoint и взаимодействует с ними в процессе обмена сообщениями. Каждый объект MessageEndpoint содержит контейнер подписок класса Subscription.

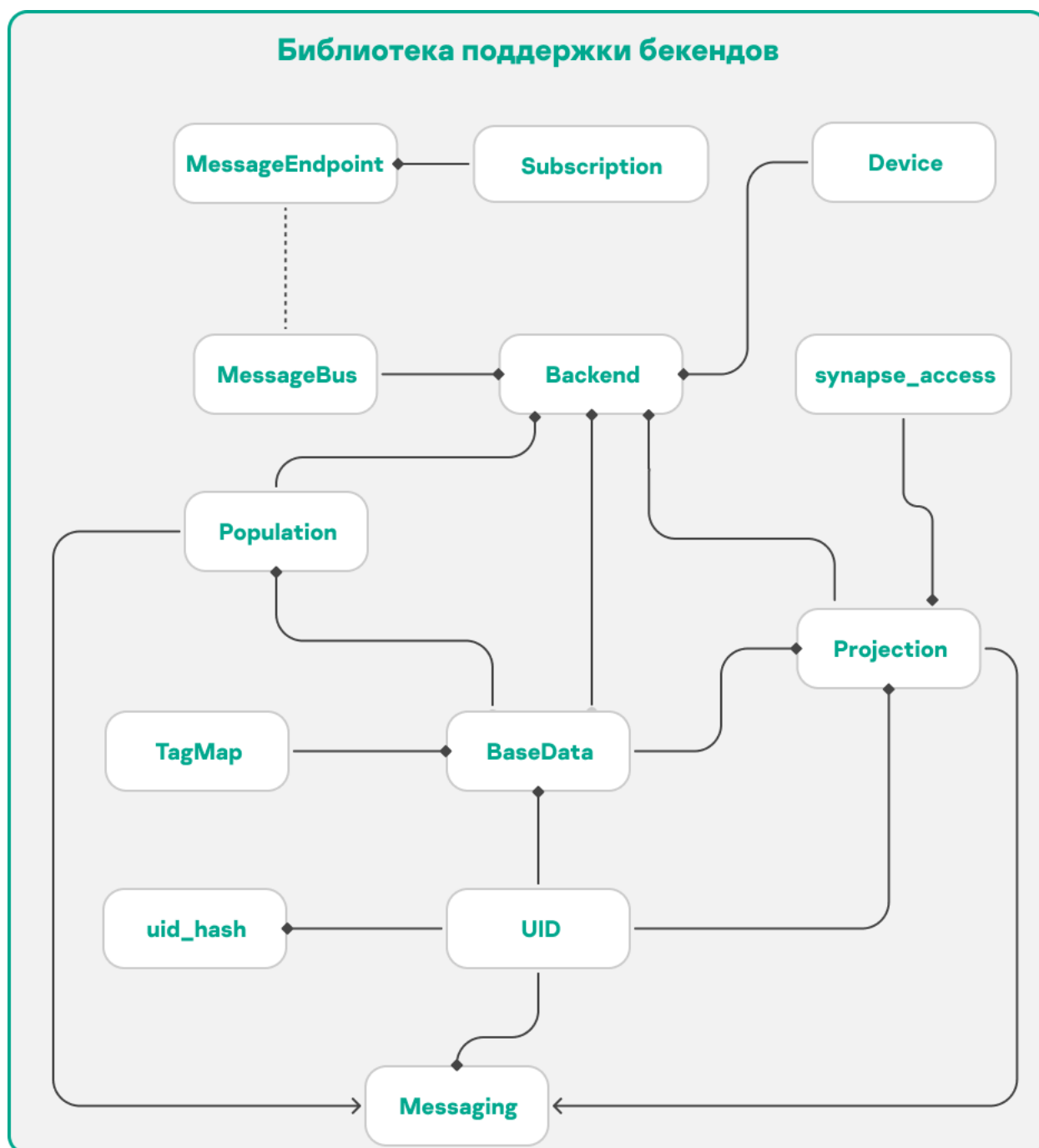


Схема взаимодействия объектов библиотеки поддержки бекендов

Класс UID

Объект класса UID является частью структуры [BaseData](#) и представляет собой идентификатор объекта. Идентификатор является уникальным.

Функция uid_hash

Функция uid_hash реализует функцию хеширования идентификаторов объектов.

Класс continuously_uid_generator

Класс `continuously_uid_generator` реализует генератор уникальных идентификаторов (англ. UID). Идентификатор отображается в виде строки, разбитой на группы с помощью дефисов, и представляет собой 128-битное число. Идентификаторы генерируются последовательно с шагом, равным 1.

Вы можете использовать класс `continuously_uid_generator` для отладки платформы и исполнения нейронной сети.

Класс TagMap

Объект класса `TagMap` является составляющей частью структуры [BaseData](#) и представляет собой ассоциативный массив, включающий ноль или более записей типа "ключ-значение тега", где ключом является имя тега. Имена тегов имеют строковый тип и являются уникальными.

Структура BaseData

Структура `BaseData` определяет набор базовых данных, общих для нескольких объектов библиотеки поддержки бекендов и объектов фреймворка. Например, базовые данные `BaseData` используются в объектах классов [Backend](#), [Device](#), [Population](#) и [Projection](#).

Базовые данные `BaseData` включают в себя следующее:

- [uid](#) – уникальный идентификатор объекта.
- [tags](#) – ассоциативный массив `TagMap`, состоящий из тегов, используемых объектом, и их значений.

Класс Population

Шаблонный класс `Population` реализует контейнер, который содержит [нейроны](#), принадлежащие одной модели. При использовании шаблона класса `Population` вы можете указать один из типов нейронов, поддерживаемых библиотекой характеристик нейронов.

С помощью класса `Population` вы можете передать данные популяций на бекенд для вычисления их атрибутов. После вычисления новые атрибуты популяций хранятся на бекенде. Чтобы передать новые атрибуты от бекенда в соответствующие объекты популяций, вы можете вызвать синхронизацию. Если вы вручную изменили атрибуты популяции до вызова синхронизации, вы можете заново передать данные популяции на бекенд для вычисления новых атрибутов. Подробнее о методах популяций и бекендов см. [API-документацию Kaspersky Neuromorphic Platform](#).

Каждая популяция нейронов содержит следующие атрибуты:

- [Базовые данные](#) популяции.
- Тип нейронов, которые хранятся в популяции.
Тип нейрона определяет набор атрибутов и функцию этого нейрона. Функция нейрона реализуется для конкретного типа нейрона на бекенде. При получении синаптического воздействия функция нейрона изменяет значения его атрибутов и возвращает состояние генерации спайка.
- Атрибуты нейронов популяции. Состояние нейронов в популяции.

Класс Projection

Шаблонный класс Projection реализует контейнер, который содержит [синапсы](#) одного из типов, поддерживаемых [библиотекой характеристик синапсов](#).

Объект класса Projection представляет собой набор соединений нейронов пресинаптической и постсинаптической популяций. Конструирование проекции осуществляется с помощью генератора соединений.

С помощью класса Projection вы можете передавать данные проекции на бекенд и получать их от бекенда.

Kaspersky Neuromorphic Platform обеспечивает [обмен данными между популяциями нейронов](#) и проекциями синапсов через [сообщения](#). Проекция синапсов получает сообщения со [спайками](#) от пресинаптической популяции и отправляет сообщения с [синаптическими воздействиями](#) в постсинаптическую популяцию.

Каждая проекция синапсов содержит следующие атрибуты:

- [Базовые данные](#) проекции.
- Уникальный идентификатор пресинаптической популяции.
- Уникальный идентификатор постсинаптической популяции.
- Значение типа boolean для обозначения возможности изменения веса синапса. Если изменение веса синапса заблокировано, то атрибут имеет значение true, иначе имеет значение false.
- Атрибуты группы синапсов.
- Значение типа boolean для обозначения актуальности [реестра соединений](#). Если реестр соединений актуален, то атрибут имеет значение true, иначе имеет значение false.
- Атрибуты, используемые синапсами одного типа.

Класс Backend

Класс Backend является базовым для реализаций бекендов. Каждый бекенд реализуется в виде отдельной библиотеки, с помощью которой вы можете:

- запускать эмуляцию работы импульсной нейронной сети на различных устройствах;
- считывать выходные [спайки](#), полученные от вычислителя;
- получать внутренние данные для мониторинга работы нейронной сети;
- выгружать внутренние данные мониторинга работы нейронной сети, полученные от вычислителя.

Каждый объект класса Backend содержит следующие атрибуты:

- [Базовые данные](#) бекенда.

- `message_bus_` – шина сообщений. С помощью шины сообщений бекенд обеспечивает обмен сообщениями между объектами библиотеки поддержки бекендов, а также обмен сообщениями от [модификаторов](#) к [нейронам](#) и [синапсам](#).
- Статус инициализации бекенда.
- Статус исполнения нейронной сети на бекенде.
- Устройства бекенда.
- Номер шага.

Класс Device

Класс Device является базовым для классов устройств, поддерживаемых [библиотекой устройств](#).

Наследники класса переопределяют методы, с помощью которых вы можете получить общую информацию об объектах классов-наследников. Например, вы можете получить информацию о типе или энергопотреблении конкретного устройства. Данные об устройствах класс Device получает от бекенда.

Каждый объект класса Device содержит [базовые данные](#) для конкретного устройства.

Класс MessageBus

Класс MessageBus реализует интерфейс для доступа к шине сообщений, которая обеспечивает прием, распределение и доставку [сообщений](#) между объектами, подключенными к [библиотеке поддержки бекенда](#). Объект класса MessageBus предоставляет множество [точек подключения](#), с помощью которых объекты библиотеки поддержки бекендов могут подписаться на сообщения или отправить их.

Класс MessageBus предоставляет методы, которые возвращают используемые точки подключения. Если вы удалите точку подключения, то передача сообщений с помощью этой точки подключения прекратится.

Использование шины сообщений позволяет реализовать принцип "издатель-подписчик". Издатель отправляет сообщения через посредника, в качестве которого выступает шина сообщений. [Объект подписки](#) фильтрует сообщения и отправляет их подписчикам. Подобный подход позволяет избежать создания большого количества IPC-каналов, поскольку издатель и подписчики не связаны напрямую.

Класс MessageEndpoint

Класс MessageEndpoint реализует интерфейс для доступа к точке подключения, которая обеспечивает подписку на [сообщения](#) и их отправку с помощью [шины сообщений](#).

Объект класса MessageEndpoint хранит в себе набор объектов класса [Subscription](#), которые используются для фильтрации сообщений. Шина сообщений может иметь несколько точек подключения, каждая из которых хранит свой набор объектов класса Subscription.

С помощью методов класса MessageEndpoint вы можете настроить обмен сообщениями между объектами бекенда и фреймворка. Для передачи сообщений требуется предварительно добавить объект подписки в объект точки подключения.

Если вы удалите точку подключения, то связанные объекты подписки также будут удалены. Передача сообщений с помощью этой точки подключения прекратится.

Класс Subscription

Класс `Subscription` реализует функциональность подписки на сообщения. Объект подписки используется для хранения сообщений определенного типа до их запроса получателем. В объекте подписки определены получатели хранящихся сообщений.

Подписки хранятся в [точках подключения](#) к шине сообщений. Если объект точки подключения, который хранит объект подписки, будет удален, то объект подписки будет также удален. Передача сообщений, определенных этой подпиской, прекратится.

Каждый объект подписки содержит следующие атрибуты:

- Вектор сообщений для передачи получателю.
- Идентификатор получателя сообщений.
- Идентификаторы отправителей сообщений.


Пространство имен Messaging

Пространство имен `Messaging` реализует интерфейсы для доступа к сообщениям. Сообщение содержит данные, которыми обмениваются объекты бекенда, а также данные, отправляемые объектами, которые идентифицируются по UID. Обмен сообщениями обеспечивается с помощью шины сообщений [MessageBus](#).

Каждое сообщение содержит следующие атрибуты:

- `MessageHeader` – заголовок сообщения, содержащий следующую информацию:
 - `sender_uid_` – идентификатор объекта, отправившего сообщение;
 - `send_time_` – время отправки сообщения.
- Тело сообщения, состоящее из совокупности передаваемых атрибутов. Состав передаваемых атрибутов зависит от типа сообщения.

С помощью сообщений происходит обновление пользовательских структур и синхронизация изменений на бекенде. Таблица ниже содержит описание поддерживаемых типов сообщений бекенда.

Тип сообщения	Описание	Структура сообщения
<code>SpikeMessage</code>	Сообщение от популяции нейронов к проекции синапсов, содержащее спайк  . На сообщения, содержащие спайки, может быть подписано более одной проекции.	<code>header_</code> – заголовок сообщения, содержащий UID популяции, нейроны которой сгенерировали спайки, и время генерации спайков. <code>neuron_indexes_</code> – индексы нейронов популяции, сгенерировавших спайки.
<code>SynapticImpactMessage</code>	Сообщение от проекции синапсов к популяции нейронов,	<code>header_</code> – заголовок сообщения, содержащий UID проекции и время

содержащее [синаптическое воздействие](#) [?].

отправки сообщения.

`postsynaptic_population_uid_` – UID постсинаптической популяции.

`presynaptic_population_uid_` – UID пресинаптической популяции.

`impacts_` – вектор, содержащий следующие атрибуты:

- `connection_index_` – индекс соединения.
- `impact_value_` – значение синаптического воздействия.
- `presynaptic_neuron_index_` – индекс пресинаптического нейрона.
- `postsynaptic_neuron_index_` – индекс постсинаптического нейрона, которому будет передано значение синаптического воздействия для изменения значений его атрибутов.

Пространство имен `synapse_access`

Пространство имен `synapse_access` реализует соединения и интерфейс для доступа к их реестру.

Структура `Connection`

Структура `Connection` определяет соединение синапса с пресинаптическим и постсинаптическим нейроном и включает в себя следующие атрибуты:

- `from_` – индекс пресинаптического нейрона.
- `to_` – индекс постсинаптического нейрона.
- `index_` – индекс синапса.

Класс `Index`

Класс `Index` реализует реестр соединений. С помощью методов класса `Index` проекция может найти синапсы, связанные с пресинаптическими или постсинаптическими нейронами с заданными индексами. В свою очередь вы можете получить информацию о соединениях с помощью методов проекции.

Запись о соединении синапса с пресинаптическим и постсинаптическим нейронами в реестр выполняется проекцией. Обновление реестра зависит от поведения функции проекции и может выполняться автоматически после любого изменения проекции (например, удаления синапса из проекции) или в момент, когда проекция используется для вычисления.

Библиотека устройств

Библиотека устройств реализует и предоставляет набор интерфейсов, которые вы можете использовать для взаимодействия с физическими устройствами.

При запуске эмуляции импульсной нейронной сети библиотека устройств используется для получения информации об устройствах (например, параметров CPU), на которых работают реализованные бекенды. Информацию об устройстве (например, о потребляемой им мощности) вы можете получить с помощью методов объекта, который обеспечивает интерфейс к устройству.

Выбор устройства для эмуляции нейронной сети осуществляется с помощью методов бекенда.

В Kaspersky Neuromorphic Platform на текущий момент реализован интерфейс для центрального процессора.

Библиотека характеристик нейронов

Библиотека характеристик нейронов представляет собой набор атрибутов [нейронов](#). Вы можете использовать атрибуты нейронов, представленных в библиотеке, например, при построении популяций.

На текущий момент библиотека поддерживает наборы свойств для модели нейронов-интеграторов с утечкой и адаптивным порогом, генерирующих серию [спайков](#). Такие нейроны-интеграторы называются BLIFAT-нейроны (англ. Bursting Leaky Integrate-and-Fire with Adaptive Threshold Neuron, BLIFATNeuron). BLIFAT-нейрон характеризуется потенциалом мембраны и пороговым значением потенциала мембраны.

Значение потенциала мембраны изменяется в соответствии со значением [синаптического воздействия](#) поступающего от связанного [синапса](#). Если потенциал мембраны нейрона достигает порогового значения, нейрон начинает генерировать серию спайков с определенным интервалом. При этом после генерации спайка потенциал мембраны стремится к базовому значению, а пороговое значение изменяется в соответствии с инкрементом.

Если к нейрону не поступают сообщения от связанного синапса в течение определенного времени, потенциал мембраны и его пороговое значение экспоненциально стремятся к базовым значениям.

Вы можете указать базовое значения потенциала мембраны, базовое пороговое значение, значение инкремента, интервал времени, при достижении которого значения потенциала и порогового значения потенциала мембраны начинают стремиться к базовым, продолжительность серии спайков и интервал генерации спайков в атрибутах экземпляра нейрона.

Если требуется, вы можете реализовать и использовать собственные модели нейронов.

Библиотека характеристик синапсов

Библиотека характеристик синапсов представляет собой набор атрибутов [синапсов](#). Вы можете использовать синапсы, представленные в библиотеке, например, при построении проекций.

На текущий момент библиотека поддерживает наборы свойств для следующих синапсов:

- Синапс по модели DeltaSynapse (далее также "дельта-синапс").

После получения [спайка](#) от пресинаптической [популяции](#) синапс генерирует сообщение, содержащее значение [синаптического воздействия](#). Проекция синапсов отправляет синаптическое воздействие популяции для изменения мембранного потенциала постсинаптического нейрона. При этом отправка сообщения [проекцией](#) происходит без временной задержки.

Отправляемое синаптическое воздействие определяется значением веса дельта-синапса. Вы можете указать [вес синапса](#) в атрибутах его экземпляра.

- Синапс по модели AdditiveSTDPDeltaSynapse.

Синапс по модели AdditiveSTDPDeltaSynapse представляет собой дельта-синапс с синаптической пластичностью, зависимой от времени получения спайка. Если спайк от пресинаптической популяции поступает в проекцию перед генерацией пресинаптического спайка постсинаптическим нейроном, значение веса синапса увеличивается. Если пресинаптический спайк поступает на синапс после генерации спайка постсинаптическим нейроном, значение веса синапса уменьшается.

Вес синапса определяет значение синаптического воздействия, отправляемого в постсинаптическую популяцию.

Изменение веса синапса рассчитывается по формуле, представленной на рисунке ниже, где:

- j соответствует пресинаптическому нейрону.
- i соответствует постсинаптическому нейрону.
- n соответствует порядковому номеру пресинаптического нейрона.
- f соответствует порядковому номеру постсинаптического нейрона.
- t_i^n соответствует времени получения синапсом спайков от пресинаптического нейрона.
- t_j^f соответствует времени генерации спайка постсинаптическим нейроном.
- Δw_j соответствует значению изменения веса синапса после получения спайка от пресинаптического нейрона.

$$\Delta w_j = \sum_{f=1}^N \sum_{n=1}^N W(t_i^n - t_j^f)$$

Формула для расчета изменения веса синапса по модели AdditiveSTDPDeltaSynapse

- Синапс по модели SynapticResourceSTDPDeltaSynapse.

Синапс по модели SynapticResourceSTDPDeltaSynapse представляет собой дельта-синапс с синаптической пластичностью, зависимой от синаптического ресурса. В свою очередь значение синаптического ресурса зависит от используемой синаптической пластичности:

- Хеббовская пластичность.

Значение синаптического ресурса зависит от периода плотной последовательности спайков (далее также "ППС"). *ППС* – это последовательность генерируемых нейроном нефорсированных спайков, в которой все соседние спайки разделены временем не больше, чем ISI_{max} , где ISI_{max} – это максимальное допустимое расстояние между спайками в ППС. В свою очередь *периодом ППС* является интервал времени между моментом начала Хеббовской пластичности перед первым спайком в ППС и одним из следующих моментов времени: достижение максимально допустимого расстояния ISI_{max} после последнего спайка в ППС, форсированное срабатывание или приход спайка на дофаминовый синапс.

При получении хотя бы одного спайка в период ППС синаптический ресурс изменяется на величину текущей Хеббовской пластичности. Значение Хеббовской пластичности зависит от переменной стабильности нейрона и рассчитывается по формуле, представленной на рисунке ниже, где:

- d_H соответствует Хеббовской пластичности.
- s соответствует переменной стабильности нейрона. В момент первого спайка в период ППС значение стабильности меняется на некоторую константу. Если стабильность уже отрицательная, она не меняется в меньшую сторону.

$$d_H = \overline{d_H} \min(2^{-s}, 1)$$

Формула для расчета Хеббовской пластичности

- Безусловная пластичность.

При ненулевой безусловной пластичности значение синаптического ресурса уменьшается на значение пластичности при получении спайка. Значение безусловной пластичности зависит от переменной стабильности нейрона и рассчитывается по формуле, представленной на рисунке ниже, где:

- d_H соответствует безусловной пластичности.
- s соответствует переменной стабильности нейрона. В момент первого спайка в период ППС значение стабильности меняется на некоторую константу. Если стабильность уже отрицательная, она не меняется в меньшую сторону.

$$d_H = \overline{d_H} \min(2^{-s}, 1)$$

Формула для расчета безусловной пластичности

- Дофаминовая пластичность.

Дофаминовая пластичность применима, когда сумма всех дофаминовых синапсов, получивших спайк в текущий шаг исполнения нейронной сети, не равна нулю. В случае дофаминовой пластичности значение синаптического ресурса изменяется по формуле, представленной на рисунке ниже, где:

- D соответствует сумме дофаминовых синапсов.
- s соответствует переменной стабильности нейрона.

$$D \min(2^{-s}, 1)$$

Формула для расчета синаптического расчета при дофаминовой пластичности

В свою очередь сумма дофаминовых синапсов влияет на переменную стабильности нейрона. Если в момент форсированного срабатывания пришло *дофаминовое наказание* (сумма дофаминовых синапсов меньше нуля), то значение стабильности нейрона уменьшается на значение rD , где r соответствует некоторой константе, а D – дофаминовому наказанию.

Если в момент форсированного срабатывания пришло *дофаминовое вознаграждение* (сумма дофаминовых синапсов выше нуля), то значение стабильности нейрона увеличивается на значение rD , где r соответствует некоторой константе, а D – дофаминовому вознаграждению. Иначе значение стабильности изменяется на значение, представленное на рисунке ниже, где:

- D соответствует сумме дофаминовых синапсов.
- ISI_{max} соответствует максимально допустимому расстоянию между спайками в ППС.
- t_{rss} соответствует времени с первого спайка в последнем ППС.

Вес синапса определяет значение синаптического воздействия, отправляемого в постсинаптическую популяцию и связан со значением синаптического ресурса формулой, представленной на рисунке ниже, где:

- w соответствует весу синапса.
- W соответствует синаптическому ресурсу.
- w_{\min} соответствует некоторому случайно заданному минимальному значению веса.
- w_{\max} соответствует некоторому случайно заданному максимальному значению веса.

$$w = w_{\min} + \frac{(w_{\max} - w_{\min}) * \max(W, 0)}{w_{\max} - w_{\min} + \max(W, 0)}$$

Формула для расчета изменения веса в зависимости от синаптического ресурса

Если требуется, вы можете реализовать и использовать собственные модели синапсов.

Библиотека шаблонов

Библиотека шаблонов представляет собой набор шаблонов общего назначения. Шаблоны библиотеки используются, например, внутри методов бекенда, обеспечивающих интроспекцию.

Поддерживаемые бекенды

В Kaspersky Neuromorphic Platform поддерживаются следующие типы бекендов:

- *Однопоточный бекенд для CPU.* Бекенд работает только в одном потоке. Вы можете использовать однопоточный бекенд для отладки платформы и работы нейронной сети.
- *Многопоточный бекенд для CPU.* Бекенд работает в нескольких потоках одновременно, что обеспечивает повышение скорости расчета нейронной сети. Вы можете использовать многопоточный бекенд для эмуляции импульсной нейронной сети.

Работа бекенда для CPU основывается на обмене сообщениями между [проекциями](#) и [популяциями](#). Объекты [библиотеки поддержки бекендов](#) также могут обмениваться сообщениями с объектами внешней среды. Для обмена сообщений бекенд предоставляет шину сообщений [MessageBus](#).

Для инициализации бекенда требуется предварительно загрузить проекции, содержащие [синапсы](#) и популяции, содержащие [нейроны](#). В процессе инициализации бекенда загруженные проекции синапсов подписываются на [спайки](#) от связанных популяций или каналов ввода, а загруженные популяции подписываются на [синаптические воздействия](#) от связанных проекций.

Цикл работы бекенда состоит из следующих этапов:

1. Каналы ввода отправляют данные в шину сообщений MessageBus.
2. Бекенд вычисляет функцию популяции нейронов.
3. Популяции нейронов отправляют спайки с результатами вычислений в шину сообщений MessageBus с помощью точек подключения MessageEndpoint.

4. Шина сообщений MessageBus отправляет данные, полученные от популяций, в проекции или каналы вывода.
5. Бекенд вычисляет функцию проекции синапсов.
6. Проекция синапсов отправляет синаптические воздействия с результатами вычислений в шину сообщений MessageBus с помощью точек подключения MessageEndpoint.
7. Шина сообщений MessageBus отправляет данные, полученные от проекций, в связанные популяции нейронов.

Обмен данными в Kaspersky Neuromorphic Platform

Kaspersky Neuromorphic Platform обеспечивает обмен данными между объектами [библиотеки поддержки бекендов](#), а также между объектами платформы и объектами внешней среды. Обмен данными между объектами библиотеки поддержки бекенда Kaspersky Neuromorphic Platform осуществляется через [сообщения](#).

Обмен данными между популяциями и проекциями

[Нейроны](#) популяции генерируют сообщения, содержащие [спайки](#) на которые может быть подписано более одной [проекцией](#). Спайки популяций передаются проекциям синапсов.

После получения спайков бекенд вычисляет [синаптические воздействия](#). Проекция отправляет сообщения со значениями синаптических воздействий постсинаптическим популяциям нейронов.

Обмен данными между бекендом и внешней средой

Бекенд принимает данные от внешней среды в виде спайков. Спайки могут быть переданы в бекенд в синхронном режиме (по запросу пользователя). Для получения спайков в синхронном режиме вызывается блокирующий метод источника данных.

Сообщения от бекенда во внешнюю среду могут передаваться подписчикам по сигналу бекенда или в синхронном режиме по запросу пользователя.

Обмен данными между нейронной сетью и внешней средой

Из внешней среды в нейронную сеть данные поступают через каналы ввода, которые преобразуют эти данные в спайки.

Во внешнюю среду нейронная сеть передает данные через каналы вывода, подключенные к указанным популяциям и преобразующие спайки в данные. Передача данных, например потенциала мембраны, возможна также через подсистему мониторинга.

Архитектура фреймворка для C++

Фреймворк Kaspersky Neuromorphic Platform, реализованный на языке программирования C++, представляет собой независимые блоки, которые обеспечивают управление бекендами, преобразуют представление нейронной сети в разные форматы и выбирают оптимальную нейронную сеть для запуска. Во фреймворке для C++ реализованы функции, не относящиеся непосредственно к эмуляции импульсных нейронных сетей.

С помощью фреймворка для C++ Kaspersky Neuromorphic Platform вы можете выполнять следующие задачи:

- Загружать структуру нейронной сети из внешних форматов.
- Динамически загружать различные бекенды.
- Выполнять эмуляции импульсных нейронных сетей на различных бекендах.
- Вводить и выводить данные нейронных сетей.
- Контролировать выполнение эмуляции.
- Реализовывать специальные алгоритмы.

Объекты фреймворка для C++

Этот раздел содержит информацию об объектах фреймворка на языке программирования C++, необходимых для управления бекендами.

Таблица ниже содержит описание объектов фреймворка для C++.

Объекты фреймворка

Объект	Описание
BackendLoader	Класс, реализующий загрузчик динамических библиотек.
Coordinates	Набор классов, реализующих пространственные координаты нейрона.
input	Пространство имен, в котором реализованы интерфейсы для доступа к каналу ввода и преобразователю данных, поступающих из внешней среды.
MessageObserver	Класс, реализующий наблюдателя.
Model	Класс, реализующий модель.
ModelExecutor	Класс, реализующий исполнитель модели.
Network	Класс, реализующий объект нейронной сети.
output	Пространство имен, в котором реализованы интерфейсы для доступа к каналам вывода.

В Kaspersky Neuromorphic Platform объект класса `Network` может включать в себя набор координат (произвольных или задаваемых пользователем) из классов `Coordinates`. Объект класса `Network`, а также контейнеры каналов ввода и вывода, реализованных в пространствах имен `input` и `output` соответственно, хранятся в объекте класса `Model`. Каналы ввода и вывода используются для преобразования данных, поступающих в проекции нейронной сети, а также для преобразования сообщений популяций нейронной сети в [один из предусмотренных форматов данных](#).

Вы можете исполнить объект класса `Model` с помощью метода `run()` объекта класса `ModelExecutor`. Объект класса `ModelExecutor` также содержит объект класса `MessageObserver`, который обрабатывает полученные сообщения в соответствии с заданной функцией, и экземпляр класса `BackendLoader`, с помощью методов которого вы можете загружать динамические библиотеки.

На рисунке ниже представлена схема взаимодействия объектов фреймворка для C++.

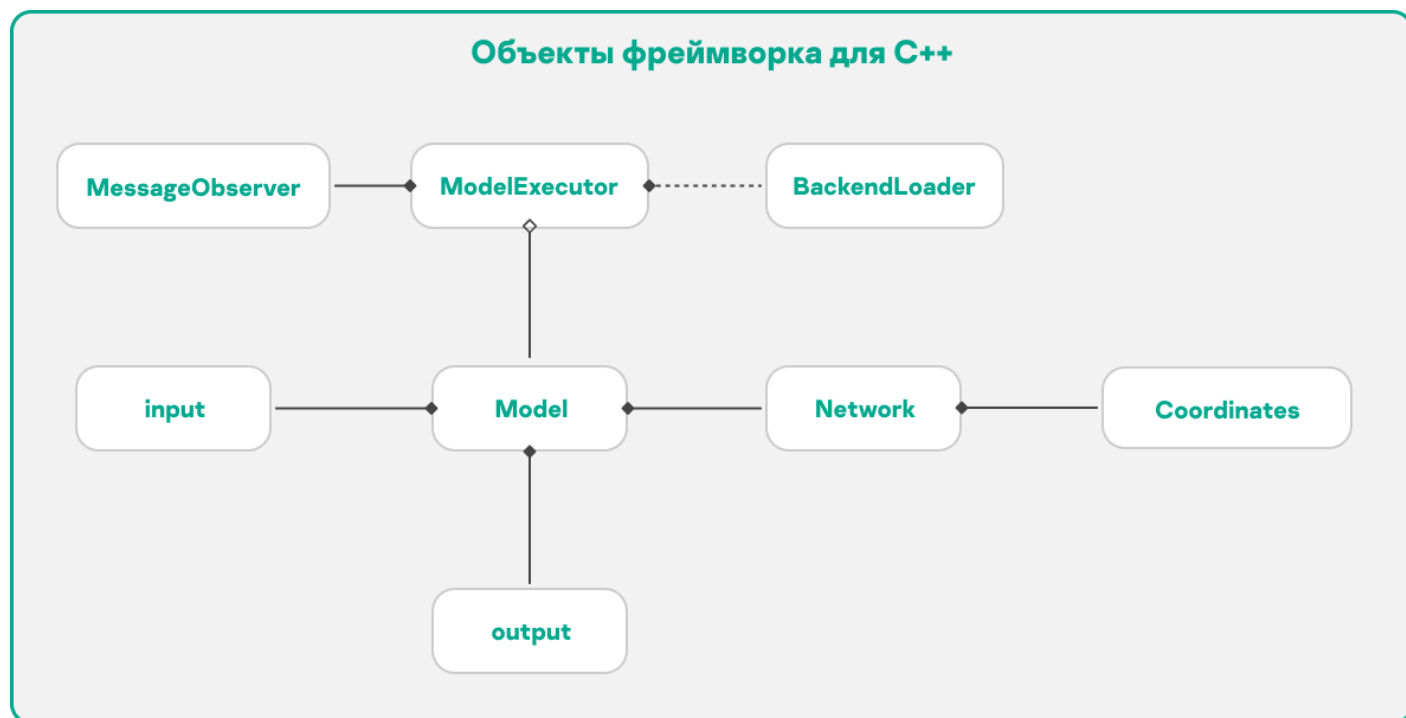


Схема взаимодействия объектов фреймворка для C++

Пространство имен `input`

В пространстве имен `input` реализованы интерфейсы для доступа к каналу ввода и преобразователю данных, поступающих из внешней среды. Данные могут поступать в Kaspersky Neuromorphic Platform как в синхронном (по запросу пользователя), так и в асинхронном режиме (по вызову обработчика).

Класс `SequenceConverter`

Класс `SequenceConverter` реализует преобразователь данных, который принимает данные из потока `istream` и делит поступившие данные на отрезки определенной длины. С помощью методов пространства имен `input` преобразователь данных приводит значения во входных отрезках к типу значений `boolean`. Если входное значение содержит [спайк](#), метод возвращает `true`. Объект класса `SequenceConverter` возвращает список индексов входных значений, которые содержат спайки, в виде сообщения [SpikeMessage](#).

Каждый объект класса `SequenceConverter` содержит следующие атрибуты:

- Поток данных `istream`, от которого поступают данные.
- Функция приведения входных значений к типу значений `boolean`.
- Длина отрезка входных данных.

Класс IndexConverter

Класс IndexConverter реализует преобразователь данных, который принимает данные из потока `istream` и преобразует последовательность индексов поступивших значений, содержащих спайки, в сообщение [SpikeMessage](#).

Каждый объект класса IndexConverter содержит следующие атрибуты:

- Поток данных `istream`, от которого поступают данные.
- Символ, с помощью которого индексы значений разделены в сообщении `SpikeMessage`.

Класс InputChannel

Класс InputChannel реализует канал ввода, который принимает сообщения `SpikeMessage` от преобразователя данных. С помощью методов класса InputChannel преобразованные данные могут быть переданы подписчикам через точки подключения при наличии подписки.

Каждый объект класса InputChannel содержит следующие атрибуты:

- Базовые данные канала ввода.
- Точка подключения, с помощью которой преобразованные данные отправляются подписчикам.
- Преобразователь данных в сообщения `SpikeMessage`.

Пространство имен output

В пространство имен `output` реализованы интерфейс для доступа к каналам вывода и преобразователи сообщений `SpikeMessage`. Kaspersky Neuromorphic Platform может передавать данные как в синхронном режиме (по запросу пользователя), так и в асинхронном режиме (по вызову обработчика).

Класс OutputChannel

Класс OutputChannel реализует канал вывода, который принимает набор сообщений, содержащих спайки, от связанной популяции за несколько тактов исполнения нейронной сети. Сообщения `SpikeMessage` могут быть преобразованы в один из следующих форматов данных, определяемых преобразователями сообщений:

- Вектор значений типа `boolean`.
- Вектор значений, в котором каждое значение соответствует количеству генерации спайков нейроном.
- Вектор нейронов, сгенерировавших спайки, из последовательности сообщений `SpikeMessage`.

Если эти форматы данных вам не подходят, вы можете реализовать собственную функцию, которая будет преобразовывать полученные сообщения в нужный вам формат данных.

Класс ConvertToSet

Класс `ConvertToSet` реализует преобразователь данных, который получает сообщения `SpikeMessage` из объекта класса `OutputChannel` и преобразует их в вектор нейронов, сгенерировавших спайки.

Каждый объект класса `ConvertToSet` содержит размер вектора нейронов.

Функтор `converter_bitwise`

Функтор `converter_bitwise` реализует преобразователь данных, который получает сообщения `SpikeMessage` и преобразует их в вектор значений типа `boolean`, в котором значение `true` соответствует наличию спайка.

Функтор `converter_count`

Функтор `converter_count` реализует преобразователь данных, который получает сообщения `SpikeMessage` и преобразует их в вектор значений. Каждое значение вектора соответствует количеству генерации спайков нейроном.

Набор классов `Coordinates`

Набор классов `Coordinates` реализует координаты точки в n-мерном пространстве. Координаты хранятся в объекте тега.

Координаты могут быть использованы для отображения нейронной сети. С помощью координат вы также можете задать атрибуты нейронной сети. Например, на основе расстояния между координатами нейронов вы можете выявить задержку сигнала, который передается между этими нейронами.

Координаты могут изменяться произвольно. Во фреймворке для C++ реализованы декартова и полярная системы координат.

Класс `MessageObserver`

Класс `MessageObserver` реализует наблюдателя, который подписывается на некоторый набор сущностей и определенный [тип сообщений](#). В конце каждого такта исполнения нейронной сети наблюдатель получает сообщения, пришедшие от сущностей по [подписке](#), и обрабатывает их в соответствии с заданной функцией.

Каждый объект класса `MessageObserver` содержит следующие атрибуты:

- [Точка подключения](#), которая обеспечивает подписку на сообщения сущностей;
- Функция для обработки полученных сообщений;
- [Идентификатор](#) наблюдателя.

Класс `Network`

Класс `Network` реализует объект нейронной сети. Объект нейронной сети хранит [базовые данные](#), [популяции](#), [проекции](#) и стратегии генерации нейронов и синапсов, а также обеспечивает общий интерфейс для доступа к нейронам и синапсам нейронной сети. В нейронной сети все популяции связаны проекциями.

Класс содержит методы, с помощью которых вы можете добавлять и удалять популяции и проекции нейронной сети.

Класс Model

Класс Model реализует модель, связывающую и хранящую нейронную сеть и каналы ввода и вывода. Для исполнения модели на разных бекендах вы можете загрузить объект модели в объект класса ModelExecutor и вызвать метод исполнения модели `run()`.

С помощью методов класса Model вы можете управлять объектом нейронной сети.

Каждый объект класса Model содержит следующие атрибуты:

- [Базовые данные](#) модели.
- [Нейронная сеть](#).
- Идентификаторы [каналов ввода](#) и [вывода](#).

Класс ModelExecutor

Класс ModelExecutor реализует исполнитель моделей. С помощью класса ModelExecutor вы можете исполнить загруженную модель на разных бекендах.

Каждый объект класса ModelExecutor содержит следующие атрибуты:

- [Базовые данные](#) исполнителя модели.
- [Загрузчик динамических библиотек](#).
- [Экземпляр бекенда](#), на котором требуется исполнить модель.
- [Модель для исполнения](#).
- Карта канала ввода, содержащая [идентификатор](#), [хешированный идентификатор](#) и [преобразователь входных данных](#).
- [Каналы ввода](#) и [вывода](#).

Класс BackendLoader

Класс BackendLoader реализует загрузчик динамических библиотек, возвращающий указатель на экземпляр загруженного бекенда.

С помощью методов класса BackendLoader вы можете загружать динамические библиотеки и проверять, является ли загруженная библиотека библиотекой бекенда.

Класс BackendLoader реализован с помощью библиотеки [Boost:DLL](#).

Сторонние библиотеки и используемые форматы

Этот раздел содержит информацию о сторонних библиотеках, форматах данных и структуры нейронной сети, используемых Kaspersky Neuromorphic Platform.

Сторонние библиотеки

Kaspersky Neuromorphic Platform использует следующие библиотеки:

- [Boost](#) . Библиотеки используются для реализации классов и структур (например, [UID](#), [BackendLoader](#)).
- [FlatBuffers](#) . Библиотека используется для сериализации в Kaspersky Neuromorphic Platform.
- [GoogleTest](#) . Библиотека используется для тестов платформы.
- [Intel PCM](#) . Библиотека используется для получения от CPU метаданных устройства (например, данные об энергопотреблении).
- [spdlog](#) . Библиотека используется для логирования работы Kaspersky Neuromorphic Platform.

Форматы данных, обрабатываемые нейронной сетью

В Kaspersky Neuromorphic Platform используются следующие форматы данных:

- MDF – формат хранения топологии нейронной сети для фреймворка.
- CSV – формат хранения состояния нейронов и синапсов нейронной сети для фреймворка.

Форматы структуры нейронной сети

В качестве формата структуры нейронной сети Kaspersky Neuromorphic Platform поддерживает ArNI CSV. Формат ArNI CSV предназначен для описания нейронов и синапсов нейронной сети в виде CSV-файлов.

Синапсы описываются следующими атрибутами:

- Индекс пресинаптического нейрона.
- Индекс постсинаптического нейрона.
- Задержка синаптического воздействия.
- Тип синапса.

Тип синапса может иметь значение 0 или 1. Значение 0 соответствует возбуждающему синапсу, значение 1 соответствует тормозящему синапсу.

- Вес синапса.

Нейроны описываются следующими атрибутами:

- Индекс нейрона.
- Коэффициент утечки потенциала мембраны.
- Минимальный потенциал мембраны нейрона.

Установка и удаление платформы

Этот раздел содержит пошаговые инструкции по установке и удалению Kaspersky Neuromorphic Platform, а также по работе с исходным кодом платформы для сборки вашего проекта.

Сценарий работы с исходным кодом платформы

Сценарий работы с исходным кодом платформы состоит из следующих этапов:

1 Подготовка рабочей директории

Создайте [рабочую директорию](#), в которой будут находиться файлы программ вашего проекта (например, файлы формата CPP). В рабочей директории создайте поддиректорию, в которой будет храниться исходный код платформы, и назовите ее KNP.

Директория KNP, в которую будет распакован архив с исходным кодом платформы, должна находиться на уровень ниже файлов программ вашего проекта.

Убедитесь, что в пути к директории KNP отсутствуют пробелы и служебные символы.

2 Получение исходного кода платформы

Распакуйте архив платформы, входящий в состав [комплекта поставки](#), в директорию KNP. Например, вы можете распаковать архив платформы с помощью следующей команды:

```
tar xf <путь к архиву платформы> --directory <путь к директории KNP>
```

Вы также можете [выполнить клонирование исходного кода платформы из репозитория](#) с помощью команды `git clone`.

3 Сборка проекта из исходного кода

В рабочей директории создайте файла скрипта сборки CMakeLists.txt и [укажите параметры сборки проекта](#). Укажите название вашего проекта. Укажите директорию с исходным кодом платформы с помощью команды `add_subdirectory()`. Подключите к вашему проекту необходимые библиотеки платформы с помощью команды `target_link_libraries()`.

4 Использование платформы

Если требуется, [добавьте новые типы нейронов](#) и/или [синапсов](#). [Постройте нейронную сеть](#) и исполните ее.

Клонирование исходного кода платформы из репозитория

Для получения исходного кода платформы вы можете клонировать репозиторий платформы на ваше устройство.

Чтобы клонировать репозиторий платформы на устройство:

1. Откройте терминал и с помощью команды `cd` перейдите в [директорию KNP](#).

Вы можете использовать любой другой инструмент для работы с Git.

2. В командной строке выполните следующую команду для загрузки репозитория платформы на устройство:

```
git clone --recurse-submodules --remote-submodules <репозиторий платформы>
```

где:

- `--recurse-submodules` означает, что каждый подмодуль в репозитории платформы, включая вложенные подмодули, будет автоматически инициализирован и обновлен.
- `--remote-submodules` означает, что для обновления подмодуля все клонированные подмодули будут использовать статус ветки его удаленного отслеживания.
- `<репозиторий платформы>` соответствует веб-адресу репозитория платформы, полученному от сотрудников "Лаборатории Касперского".

Сборка проекта из исходного кода платформы

При работе с исходным кодом платформы вы можете указать параметры сборки вашего проекта в файле скрипта сборки `CMakeLists.txt`.

В этом разделе приводится последовательность действий, которые требуется выполнить для настройки параметров сборки проекта, с помощью терминала Linux и текстового редактора `vim`. Вместо терминала Linux и редактора `vim` вы можете использовать любой другой инструмент разработки и текстовый редактор для работы с файлом скрипта сборки `CMakeLists.txt`.

Подробнее о сборке проектов с помощью CMake см. [документацию CMake](#).

Чтобы собрать проект из исходного кода на устройстве Linux:

1. На вашем устройстве под управлением Linux откройте терминал и перейдите в [рабочую директорию](#) с помощью команды `cd`:

```
cd <путь к рабочей директории>
```

2. В рабочей директории создайте файл скрипта сборки `CMakeLists.txt` с помощью текстового редактора `vim`:

```
vim CMakeLists.txt
```

Вместо редактора `vim` вы можете использовать любой другой текстовый редактор.

3. В файле скрипта сборки укажите следующие параметры сборки вашего проекта:

`CMakeLists.txt`

```
cmake_minimum_required(VERSION 3.22)
project(<название проекта>)

set(CMAKE_CXX_STANDARD 17)

add_subdirectory(KNP)

add_executable("${PROJECT_NAME}" <файл программы>)
target_link_libraries("${PROJECT_NAME}" PRIVATE <подключаемые библиотеки
платформы>)
```

где:

- `project(<название проекта>)` определяет название проекта, например, `project (my_project)`.
- `<файл программы>` соответствует названию файла программы, например, `main.cpp`.
- `target_link_libraries("${PROJECT_NAME}" PRIVATE <подключаемые библиотеки платформы>)` определяет библиотеки, которые будут подключены к вашему проекту. Например, `target_link_libraries("${PROJECT_NAME}" PRIVATE KNP::BaseFramework::Core)` подключит к проекту фреймворк.

Примеры файла CMakeLists.txt:

Примеры файла CMakeLists.txt:

- [Пример файла CMakeLists.txt при загрузке динамических библиотек бекенда ?](#)

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.22)
// simple-network - название проекта
project(simple-network)

set(CMAKE_CXX_STANDARD 17)

// Запускает CMake в директории KNP и добавляет все проекты из директории в сборку
add_subdirectory(KNP)

// Добавляет бинарный файл с исходным кодом в файл программы main.cpp
add_executable("${PROJECT_NAME}" main.cpp)
// Подключает фреймворк к проекту simple-network
target_link_libraries("${PROJECT_NAME}" PRIVATE KNP::BaseFramework::Core)
```

- [Пример файла CMakeLists.txt без загрузки динамических библиотек бекенда ?](#)

При сборке проекта без загрузки динамических библиотек требуется подключить библиотеки нужных бекендов с помощью метода `target_link_libraries`. Перечислите необходимые библиотеки через пробел как в приведенном примере.

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.22)
// digits-recognition - название проекта
project(digits-recognition)

set(CMAKE_CXX_STANDARD 17)

// Запускает CMake в директории KNP и добавляет все проекты из директории в сборку
add_subdirectory(KNP)

// Добавляет бинарный файл с исходным кодом в файл программы main.cpp
add_executable("${PROJECT_NAME}" main.cpp)
// Подключает фреймворк и библиотеку однопоточного бекенда для CPU к проекту simple-network.
target_link_libraries("${PROJECT_NAME}" PRIVATE KNP::BaseFramework::Core
KNP::Backends::CPUSingleThreaded)
```

4. Для сохранения изменений в файле скрипта сборки выполните следующие действия:

- a. Нажмите на клавишу **Esc**.
- b. Введите следующую строку:
:wq!
- c. Нажмите на клавишу **Enter**.

Удаление платформы

Чтобы удалить *Kaspersky Neuromorphic Platform*,

удалите [директорию KNP](#) с исходным кодом платформы из вашей рабочей директории.

Директории для хранения данных платформы

При работе с исходным кодом платформы Kaspersky Neuromorphic Platform использует следующие директории и поддиректории:

- `.` – корневая директория с вашим проектом. Используется для хранения файлов программ (например, `main.cpp`), а также файла скрипта сборки проекта `CMakeLists.txt`. Вы можете присвоить корневой директории любое название (например, `/app/`).
- `./KNP` – директория для хранения исходного кода платформы, включающая следующие поддиректории:
 - `./KNP/ci` – директория для хранения инструментов сборки на конвейере CI.
 - `./KNP/cmake` – директория для хранения модулей CMake.
 - `./KNP/knp` – основная директория платформы. Используется для хранения файлов бекенда, фреймворка, библиотек платформы, а также справочного руководства API.
 - `./KNP/third-party` – директория для хранения сторонних библиотек, необходимых для сборки бекенда.
 - `./KNP/tools` – директория для хранения инструментов, применяемых в локальной сборке и сборке на конвейере CI.

Добавление нового типа нейрона

На текущий момент Kaspersky Neuromorphic Platform поддерживает наборы свойств и типизацию популяций для модели BLIFAT-нейрона. Если требуется, вы можете добавить новый тип нейрона.

Чтобы добавить новый тип нейрона:

1. В директории `neuron-trait-library/include/knp/neuron-traits/` создайте заголовочный файл для нового типа нейрона.
2. В созданном заголовочном файле определите структуру типа нейрона и его шаблонные свойства (например, `default_values`, `neuron_parameters`).

Пример определения BLIFAT-нейрона:

`neuron-trait-library/include/knp/neuron-traits/blifat.h`

```
#include "type_traits.h"

namespace knp::neuron_traits
{
    struct BLIFATNeuron;

    template <>
    struct default_values<BLIFATNeuron>

    {
        // Задаёт количество шагов нейронной сети с шага последнего спайка.
        constexpr static std::size_t steps_before_firing =
        std::numeric_limits<std::size_t>::infinity();
        // Задаёт значение, к которому стремится мембранный потенциал для тормозных
        // синапсов, основанных на проводимости.
        constexpr static double reverse_inhibitory_potential = -0.3;
        // Задаёт значение, к которому стремится мембранный потенциал для тормозных
        // синапсов, основанных на токе.
        constexpr static double min_potential = -1.0e9;
    };
    ...
} // namespace knp::neuron_traits
```

3. Добавьте созданный тип нейрона в список нейронов, определенных в заголовочном файле `neuron-trait-library/include/knp/neuron-traits/all_traits.h`.

Пример добавления BLIFAT-нейрона в список нейронов:

`neuron-trait-library/include/knp/neuron-traits/all_traits.h`

```
#include "blifat.h"

namespace knp::neuron_traits
{
    // Список типов нейронов, разделенных запятыми.
    #define ALL_NEURONS knp::neuron_traits::BLIFATNeuron
    ...
} // namespace knp::neuron_traits
```


4. Реализуйте в нужных бекендах перегруженный метод запуска.

Например, для бекенда CPU реализуйте перегруженный метод запуска расчета популяций.

Добавление нового типа синапса

На текущий момент Kaspersky Neuromorphic Platform поддерживает наборы свойств и типизацию проекций по модели дельта-синапса. Если требуется, вы можете добавить новый тип синапса.

Чтобы добавить новый тип синапса:

1. В директории `synapse-traits-library/include/knp/synapse-traits/` создайте заголовочный файл для нового типа синапса.
2. В созданном заголовочном файле определите структуру типа синапса и его шаблонные свойства (например, `synapse_parameters`).

Пример определения дельта-синапса:

```
synapse-traits-library/include/knp/synapse-traits/delta.h

#include "type_traits.h"

namespace knp::synapse_traits
{
    struct DeltaSynapse;

    template <>
    struct synapse_parameters<DeltaSynapse>
    {
        // Задаёт атрибуты синапса.
        synapse_parameters() : weight_(0.0F), delay_(1),
        output_type_(knp::synapse_traits::OutputType::EXCITATORY) {}
        synapse_parameters(float weight, uint32_t delay,
        knp::synapse_traits::OutputType type)
            : weight_(weight), delay_(delay), output_type_(type)

        // Вес синапса
        float weight_;
        // Задержка синапса
        std::size_t delay_;
        // Тип синапса на выходе
        knp::synapse_traits::OutputType output_type_;
    };

    } // namespace knp::synapse_traits
```

3. Добавьте созданный тип синапса в список синапсов, определенных в заголовочном файле `synapse-traits-library/include/knp/synapse-traits/all_traits.h`.

Пример добавления дельта-синапса в список синапсов:

```
synapse-traits-library/include/knp/synapse-traits/all_traits.h

#include "delta.h"

namespace knp::synapse_traits
{
    // Список типов синапсов, разделенных запятыми.
```

```
#define ALL_SYNAPSES knp::synapse_traits::DeltaSynapse
```

```
...  
} // namespace knp::neuron_traits
```

4. Реализуйте в нужных бекендах перегруженный метод запуска.

Например, для бекенда CPU реализуйте перегруженный метод запуска расчета проекций.

Исполнение нейронной сети, загруженной на бекенд автоматически

Этот раздел содержит инструкции по созданию нейронной сети, ее автоматической загрузке на бекенд и исполнению.

Чтобы автоматически загрузить нейронную сеть на бекенд и исполнить ее:

1. В директории вашего проекта создайте файл программы формата CPP, в котором будет реализована функция для создания и запуска нейронной сети.
2. В файле программы подключите заголовочные файлы, необходимые для исполнения нейронной сети, с помощью директивы `#include`.

Если требуется, определите псевдонимы с помощью оператора `using`.

Пример подключения заголовочных файлов для исполнения нейронной сети с популяцией BLIFAT-нейронов и проекцией дельта-синапсов на однопоточном бекенде для CPU:

C++

```
#include <knp/framework/io/out_converters/convert_set.h>
#include <knp/framework/model_executor.h>
#include <knp/framework/network.h>
#include <knp/neuron-traits/blifat.h>
#include <knp/synapse-traits/delta.h>

#include <filesystem>

using DeltaProjection = knp::core::Projection<knp::synapse_traits::DeltaSynapse>;
using BLIFATPopulation = knp::core::Population<knp::neuron_traits::BLIFATNeuron>;
```

3. Реализуйте генератор синапсов с нужными вам свойствами или используйте готовый генератор из библиотеки фреймворка.

Пример реализации генератора дельта-синапсов:

C++

```
// Реализована функция, генерирующая синапсы для проекции, которая будет связана
// с каналом ввода. Функция хранится в переменной input_projection_gen.
inline std::optional<DeltaProjection::Synapse> input_projection_gen(size_t
/*index*/)
{
    return DeltaProjection::Synapse{{1.0, 1,
knp::synapse_traits::OutputType::EXCITATORY}, 0, 0};
}

// Реализована функция, генерирующая синапсы для проекции, которая будет замыкать
// вывод
// популяции на себя. Функция хранится в переменной synapse_generator.
inline std::optional<DeltaProjection::Synapse> synapse_generator(size_t /*index*/)
{
    return DeltaProjection::Synapse{{1.0, 6,
knp::synapse_traits::OutputType::EXCITATORY}, 0, 0};
}
```

4. Реализуйте генератор нейронов с нужными вам свойствами или используйте готовый генератор из библиотеки фреймворка.

Пример реализации генератора BLIFAT-нейронов:

C++

```
// Реализована функция, генерирующая нейроны.  
// Функция хранится в переменной neuron_generator.  
inline knp::neuron_traits::neuron_parameters<knp::neuron_traits::BLIFATNeuron>  
neuron_generator(size_t)  
{  
    return knp::neuron_traits::neuron_parameters<knp::neuron_traits::BLIFATNeuron>  
};  
}
```

5. Создайте функцию, в которой будут созданы объекты, необходимые для исполнения нейронной сети.

Пример создания функции main:

C++

```
main(int argc, const char* const argv[])  
{  
    ...  
}
```

6. В созданной функции создайте объект популяции и передайте в конструктор генератор нейронов.

Пример создания объекта популяции:

C++

```
...  
// Создает объект популяции с одним BLIFAT-нейроном  
BLIFATPopulation population{neuron_generator, 1};  
  
// Сохраняет UID популяции в переменной output_uid  
knp::core::UID output_uid = population.get_uid();  
...
```

7. В созданной функции создайте объект проекции, который будет замыкать вывод популяции на себя. Передайте в конструктор идентификатор связанной популяции и генератор синапсов.

Пример создания проекции, замыкающей вывод популяции на себя:

C++

```
...  
// Создает объект проекции с одним дельта-синапсом, замыкающей вывод популяции  
// самой на себя  
DeltaProjection loop_projection =  
    DeltaProjection{population.get_uid(), population.get_uid(),  
synapse_generator, 1};  
...
```

8. В созданной функции создайте объект входной проекции. Передайте в конструктор генератор синапсов и идентификатор связанной популяции.

Пример создания входной проекции, связанной с каналом ввода и популяцией:

C++

```
...
// Создает объект входной проекции с одним дельта-синапсом, который
присваивается
// нулевой идентификатор(knp::core::UID{false}). Проекция принимает спайки от
// канала ввода и отправляет синаптическое воздействие в объект популяции.
DeltaProjection input_projection = DeltaProjection{knp::core::UID{false},
population.get_uid(), input_projection_gen, 1};

// Сохраняет UID входной проекции в переменной input_uid
knp::core::UID input_uid = input_projection.get_uid();
...
```

9. В созданной функции создайте объект нейронной сети и передайте в него созданные проекции и популяцию.

Пример создания объекта нейронной сети:

C++

```
...
// Создает объект сети network
knp::framework::Network network;

// Добавляет созданный объект популяции population в объект нейронной сети
network
network.add_population(std::move(population));

// Добавляет созданный объект входной проекции input_projection в объект
// нейронной сети network
network.add_projection(std::move(input_projection));

// Добавляет созданный объект проекции loop_projection, замыкающей вывод
// популяции самой на себя, в объект нейронной сети network
network.add_projection(std::move(loop_projection));
...
```

10. В созданной функции определите идентификаторы каналов ввода и вывода.

Пример определения идентификаторов каналов ввода и вывода:

C++

```
...
// Создает произвольные идентификаторы i_channel_uid и o_channel_uid для
каналов ввода и вывода
knp::core::UID i_channel_uid, o_channel_uid;
...
```

11. В созданной функции создайте объект модели. Передайте в модель объект нейронной сети, идентификаторы каналов, популяции и входной проекции.

Пример создания модели:

C++

```
...
// Создает объект модели model и передает в нее объект нейронной сети network
knp::framework::Model model(std::move(network));

// Передает в объект модели model созданный идентификатор канала ввода
// i_channel_uid и идентификатор входной проекции input_uid.
model.add_input_channel(i_channel_uid, input_uid);

// Передает в объект модели model созданный идентификатор канала вывода
// o_channel_uid и идентификатор популяции output_uid.
model.add_output_channel(o_channel_uid, output_uid);
...
```

12. В созданной функции создайте функтор генерации спайков.

Пример создания функтора генерации спайков:

C++

```
...
auto input_gen = [](knp::core::messaging::Step step) ->
knp::core::messaging::SpikeData
{
    // Посылает спайки на шагах 0, 5, 10 и 15 исполнения нейронной сети
    if (step % 5 == 0)
    {
        knp::core::messaging::SpikeData s;
        s.push_back(0);
        return s;
    }
    return knp::core::messaging::SpikeData();
};
...
```

13. В созданной функции укажите путь к экземпляру бекенда, который будет исполнять нейронную сеть.

Путь к бекенду должен соответствовать пути к динамической библиотеке бекенда. Если экземпляр бекенда находится не по указанному пути, измените путь к бекенду.

Пример:

C++

```
...
// Заданный путь приведен в качестве примера. Укажите путь к нужной
// динамической библиотеке бекенда на вашем локальном компьютере.
auto backend_path = std::filesystem::path(argv[0]).parent_path().parent_path()
/ "lib" / "knp-cpu-single-threaded-backend";
...
```

14. В созданной функции создайте объект исполнителя модели. Передайте исполнителю модели объект модели, путь к бекенду, идентификатор канала ввода и функтор генерации спайков. Запустите исполнение модели.

Пример создания объекта исполнителя модели и исполнения 20 шагов нейронной сети:

C++

```
...
// Создает объект исполнителя модели me. Передает в объект me объект модели
model,
// путь к бекенду backend_path, идентификатор канала ввода i_channel_uid и
// функтор генерации спайков input_gen.
knp::framework::ModelExecutor me(model, backend_path, {{i_channel_uid,
input_gen}});

// Инициализирует исполнитель модели me
me.init();

// Получает ссылку на объект канала вывода out_channel из
// исполнителя моделей me по идентификатору канала вывода o_channel_uid
auto &out_channel = me.get_output_channel(o_channel_uid);

// Запускает исполнение модели на 20 шагов
me.start([](size_t step) { return step < 20; });
...
```

[Шаги исполнения нейронной сети](#) 

Исполнение нейронной сети производится циклично. Действия, описанные в цикле, повторяются каждые 5 шагов исполнения сети.

На первом шаге цикла канал ввода посылает спайки входной проекции.

На следующем шаге цикла входная проекция вычисляет синаптические воздействия и отправляет их популяции. После получения синаптического воздействия популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя.

Через 6 шагов цикла после получения спайков, отправленных входной проекцией, проекция, замыкающая вывод популяции на себя, вычисляет синаптические воздействия, и передает воздействия популяции.

На рисунке ниже представлена схема исполнения нейронной сети.



Схема исполнения нейронной сети

В этом примере нейронная сеть исполняется в рамках цикла со следующими шагами:

- **Шаг 0.** Канал ввода посылает спайки входной проекции. Шаг повторяется каждые 5 шагов цикла исполнения нейронной сети.
- **Шаг 1.** Входная проекция вычисляет синаптические воздействия и отправляет их популяции. После получения синаптического воздействия популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя. Шаг повторяется каждые 5 шагов исполнения сети.
- **Шаг 7.** Проекция, замыкающая вывод популяции на себя, вычисляет синаптические воздействия после получения спайков, отправленных 6 шагов назад, и передает воздействия популяции. Популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя. Шаг повторяется каждые 5 шагов исполнения нейронной сети.

- *Шаг 13.* Проекция, замыкающая вывод популяции на себя, вычисляет синаптические воздействия после получения спайков, отправленных 6 шагов назад, и передает воздействия популяции. Популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя. Шаг повторяется каждые 5 шагов исполнения нейронной сети.
- *Шаг 19.* Проекция, замыкающая вывод популяции на себя, вычисляет синаптические воздействия после получения спайков, отправленных на шаге 13, и передает воздействия популяции. Популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя.

15. При необходимости создайте вектор, в который будут записываться индексы шагов, на которых на канал вывода приходят спайки.

Пример записи результатов работы модели:

C++

```
// Создает вектор results, в который будут записываться индексы шагов со
спайками
std::vector<knp::core::messaging::Step> results;

// Обновляет канал вывода
const auto &spikes = out_channel.update();

// Выделяет в памяти область для спайков
results.reserve(spikes.size());

// Записывает в вектор results индексы шагов, на которых на канал вывода
// приходят спайки
std::transform(
    spikes.cbegin(), spikes.cend(), std::back_inserter(results),
    [](const auto &spike_msg) { return spike_msg.header_.send_time_; });

// Выводит через пробел индексы шагов, на которых на канал вывода приходят
спайки
for (const auto &s : results) std::cout << s << " ";
// Выводит символ новой строки и вызывает метод flush()
std::cout << std::endl;
}
```

Исполнение нейронной сети, созданной из проекций и популяций

Этот раздел содержит инструкции по созданию нейронной сети из проекций и популяций, загрузке проекций и популяций на бекенд и исполнению нейронной сети.

Этот вариант построения и исполнения нейронной сети используется редко.

Чтобы создать нейронную сеть вручную и исполнить ее:

1. В директории вашего проекта создайте файл программы формата CPP, в котором будет реализована функция для создания и запуска нейронной сети.
2. В файле программы подключите заголовочные файлы, необходимые для исполнения нейронной сети, с помощью директивы `#include`.

Если требуется, определите псевдонимы с помощью оператора `using`.

Пример подключения заголовочных файлов для исполнения нейронной сети с популяцией BLIFAT-нейронов и проекцией дельта-синапсов на однопоточном бекенде для CPU:

C++

```
#include <knp/backends/cpu-single-threaded/backend.h>
#include <knp/core/population.h>
#include <knp/core/projection.h>
#include <knp/neuron-traits/blifat.h>
#include <knp/synapse-traits/delta.h>

#include <vector>

using Backend = knp::backends::single_threaded_cpu::SingleThreadedCPUBackend;
using DeltaProjection = knp::core::Projection<knp::synapse_traits::DeltaSynapse>;
using BLIFATPopulation = knp::core::Population<knp::neuron_traits::BLIFATNeuron>;
using Population =
    knp::backends::single_threaded_cpu::SingleThreadedCPUBackend::PopulationVariants;
using Projection =
    knp::backends::single_threaded_cpu::SingleThreadedCPUBackend::ProjectionVariants;
```

3. Реализуйте генератор нейронов с нужными вам свойствами или используйте готовый генератор из библиотеки фреймворка.

Пример реализации генератора BLIFAT-нейронов:

C++

```
...
// Реализована функция, генерирующая нейроны.
// Функция хранится в переменной neuron_generator.
auto neuron_generator = [](size_t index)
{
    return knp::neuron_traits::neuron_parameters<knp::neuron_traits::BLIFATNeuron>
{};
};
...
```

4. Реализуйте генератор синапсов с нужными вам свойствами или используйте готовый генератор из библиотеки фреймворка.

Пример реализации генератора дельта-синапсов:

C++

```
...
// Реализована функция, генерирующая синапсы для проекции, которая будет связана
// с каналом ввода. Функция хранится в переменной input_projection_gen.
DeltaProjection::SynapseGenerator input_projection_gen = [](size_t index) ->
std::optional<DeltaProjection::Synapse>
{
    return DeltaProjection::Synapse{{1.0, 1,
knp::synapse_traits::OutputType::EXCITATORY}, 0, 0};
};

// Реализована функция, генерирующая синапсы для проекции, которая будет замыкать
// вывод популяции на себя. Функция хранится в переменной synapse_generator.
DeltaProjection::SynapseGenerator synapse_generator = [](size_t index) ->
std::optional<DeltaProjection::Synapse>
{
    return DeltaProjection::Synapse{{1.0, 6,
knp::synapse_traits::OutputType::EXCITATORY}, 0, 0};
};
...
```

5. Создайте объект популяции и передайте в конструктор созданный генератор нейронов.

Пример создания объекта популяции:

C++

```
...
// Создает объект популяции
BLIFATPopulation population{neuron_generator, 1};
```

6. Создайте объект проекции, которая будет замыкать вывод популяции на себя. Передайте в конструктор идентификатор связанной популяции и генератор синапсов.

Пример создания проекции, замыкающей вывод популяции на себя:

C++

```
...
// Создает объект проекции, замыкающей вывод популяции на себя
Projection loop_projection = DeltaProjection{population.get_uid(),
population.get_uid(), synapse_generator, 1};
...
```

7. Создайте объект входной проекции. Передайте в конструктор генератор синапсов и идентификатор связанной популяции.

Пример создания входной проекции, связанной с каналом ввода и популяцией:

C++

```
...
```

```

// Создает объект входной проекции, которой присваивается нулевой идентификатор
// (knp::core::UID{false}). Проекция принимает спайки от канала ввода и отправляет
// синаптическое воздействие в объект популяции.
Projection input_projection = DeltaProjection{knp::core::UID{false},
population.get_uid(), input_projection_gen, 1};
// Получает UID входной проекции, которая отправляет популяции синаптическое
воздействие
knp::core::UID input_uid = std::visit([](const auto &proj) { return proj.get_uid();
}, input_projection);
...

```

8. Загрузите популяцию и проекции на бекенд.

Пример загрузки популяции и проекций на бекенд:

C++

```

...
Backend backend;
...
// Загружает на бекенд созданный объект популяции
backend.load_populations({population});
// Загружает на бекенд созданные объекты проекций
backend.load_projections({input_projection, loop_projection});

```

9. Определите объект точки подключения.

Пример создания точки подключения:

C++

```

...
// Создает точку подключения
auto endpoint = backend.message_bus_.create_endpoint();
...

```

10. Подключите канал ввода к входной проекции, а также популяцию к каналу вывода. Задайте для каналов ввода и вывода уникальные идентификаторы и сформируйте для них сообщения.

Пример подключения к каналам ввода и вывода:

C++

```

...
// Создает произвольный UID канала ввода
knp::core::UID in_channel_uid;

// Создает произвольный UID канала вывода
knp::core::UID out_channel_uid;

// Создает подписку на спайки от канала ввода
backend.subscribe<knp::core::messaging::SpikeMessage>(input_uid, {in_channel_uid});

// Создает подписку на спайки от популяции с заданным UID для канала вывода
// с заданным UID
endpoint.subscribe<knp::core::messaging::SpikeMessage>(out_channel_uid,
{population.get_uid()});
...

```

11. Настройте передачу сообщений в нейронной сети и укажите параметры исполнения такта нейронной сети.

Следующий пример исполнения нейронной сети состоит из 20 шагов:

C++

```
...
std::vector<size_t> results;
for (size_t step = 0; step < 20; ++step)
{
    // Посылает спайки входной проекции от канала ввода на шагах 0, 5, 10 и 15
    if (step % 5 == 0)
    {
        knp::core::messaging::SpikeMessage message{{in_channel_uid, 0}, {0}};
        endpoint.send_message(message);
    }
    backend.step();
    endpoint.receive_all_messages();
    // Получает спайки, отправленные популяцией на канал вывода
    auto output = endpoint.unload_messages<knp::core::messaging::SpikeMessage>
(out_channel_uid);

    // Записывает индексы шагов, на которых на канал вывода приходят спайки
    if (!output.empty()) results.push_back(step);
}
...
```

[Шаги исполнения нейронной сети](#) 

Исполнение нейронной сети производится циклично. Действия, описанные в цикле, повторяются каждые 5 шагов исполнения сети.

На первом шаге цикла канал ввода посылает спайки входной проекции.

На следующем шаге цикла входная проекция вычисляет синаптические воздействия и отправляет их популяции. После получения синаптического воздействия популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя.

Через 6 шагов цикла после получения спайков, отправленных входной проекцией, проекция, замыкающая вывод популяции на себя, вычисляет синаптические воздействия, и передает воздействия популяции.

На рисунке ниже представлена схема исполнения нейронной сети.



Схема исполнения нейронной сети

В этом примере нейронная сеть исполняется в рамках цикла со следующими шагами:

- **Шаг 0.** Канал ввода посылает спайки входной проекции. Шаг повторяется каждые 5 шагов цикла исполнения нейронной сети.
- **Шаг 1.** Входная проекция вычисляет синаптические воздействия и отправляет их популяции. После получения синаптического воздействия популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя. Шаг повторяется каждые 5 шагов исполнения сети.
- **Шаг 7.** Проекция, замыкающая вывод популяции на себя, вычисляет синаптические воздействия после получения спайков, отправленных 6 шагов назад, и передает воздействия популяции. Популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя. Шаг повторяется каждые 5 шагов исполнения нейронной сети.

- *Шаг 13.* Проекция, замыкающая вывод популяции на себя, вычисляет синаптические воздействия после получения спайков, отправленных 6 шагов назад, и передает воздействия популяции. Популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя. Шаг повторяется каждые 5 шагов исполнения нейронной сети.
- *Шаг 19.* Проекция, замыкающая вывод популяции на себя, вычисляет синаптические воздействия после получения спайков, отправленных на шаге 13, и передает воздействия популяции. Популяция отправляет спайки каналу вывода и проекции, замыкающей вывод популяции на себя.

Лицензирование платформы

Условия использования Kaspersky Neuromorphic Platform – это обязательное соглашение между вами и АО "Лаборатория Касперского", в котором изложены условия, на которых вы можете пользоваться платформой.

Внимательно ознакомьтесь с Условиями использования перед началом работы с Kaspersky Neuromorphic Platform.

Вы можете ознакомиться с Условиями использования, прочитав документ LICENSE.txt, расположенный в директории KNP после распаковки архива с исходным кодом платформы.

Если вы не согласны с Условиями использования, вы должны удалить и не использовать Kaspersky Neuromorphic Platform.

Предоставление данных

Kaspersky Neuromorphic Platform не запрашивает, не хранит и не обрабатывает информацию, относящуюся к персональным данным.

Использование Kaspersky Neuromorphic Platform API

В Kaspersky Neuromorphic Platform реализован интерфейс прикладного программирования (Application Programming Interface, API), который обеспечивает доступ к методам платформы.



[ОТКРЫТЬ СПРАВОЧНОЕ РУКОВОДСТВО API для фреймворка для C++ \(на английском языке\)](#)

С помощью Kaspersky Neuromorphic Platform API вы можете выполнять следующие действия:

- получать данные об используемых устройствах и бекендах;
- получать данные об объектах библиотеки поддержки бекендов;
- получать данные об объектах фронтенда платформы;
- управлять объектами библиотеки поддержки бекендов;
- создавать и обучать нейронные сети;
- управлять гиперпараметрами нейронных сетей;
- создавать новые модели нейронов;
- создавать новые топологии нейронных сетей;
- реализовывать прикладные решения.

Глоссарий

Бекенд

Программно-аппаратная часть платформы, обеспечивающая универсальный интерфейс к вычислителю и реализующая функции нейронов и синапсов, а также их модификаторов в терминах вычислителя.

Вес синапса

Значение сигнала, который синапс передает нейронам связанной популяции.

Вычислитель

Устройство или код, непосредственно выполняющий сеть.

Модификатор

Код, изменяющий поведение нейрона или синапса, например добавляющий синаптическую пластичность.

Нейрон

Узел нейронной сети, имеющий набор атрибутов. На основе атрибутов нейрона и последовательности входных сигналов происходит вычисление некоторой математической функции, результатом которого является синаптическое воздействие или его отсутствие.

Период ППС

Интервал времени между моментом начала Хеббовской пластичности перед первым спайком в ППС и одним из следующих моментов времени: достижение максимально допустимого расстояния (ISI_{max}) после последнего спайка в ППС, форсированное срабатывание или приход спайка на дофаминовый синапс.

Плотная последовательность спайков (ППС)

Последовательность генерируемых нейроном нефорсированных спайков, в которой все соседние спайки разделены временем не большее, чем ISI_{max} , где ISI_{max} – это максимальное допустимое расстояние между спайками.

Популяция

Контейнер, содержащий набор нейронов и их функцию в соответствии с типом нейронов.

Проекция

Контейнер, содержащий набор синапсов одного типа, соединяющих нейроны пресинаптической и постсинаптической популяций.

Синапс

Место соединения нейронов, основная функция которого – передача сигнала от одного нейрона к другому. Каждый нейрон может быть связан с множеством синапсов.

Синаптическое воздействие

Значение, формируемое синапсом и предназначенное для изменения значений атрибутов нейронов.

Соединение

Связь синапса с пресинаптическим или постсинаптическим нейроном.

Спайк

Кратковременный потенциал действия, сгенерированный нейроном популяции в результате изменения атрибутов нейрона.

Фреймворк

Программно-аппаратная часть платформы, определяющая ее структуру и обеспечивающая управление бекендами. В Kaspersky Neuromorphic Platform реализован фреймворк на языке программирования C++.

Функция нейрона

Система, описывающая и изменяющая поведение нейрона. Результатом функции является спайк или его отсутствие.

Функция синапса

Система, моделирующая поведение синапса. Результатом функции является синаптическое воздействие или его отсутствие.

Информация о стороннем коде

Информация о стороннем коде содержится в файле `NOTICES.txt`, расположенном в директории KNP после распаковки архива с исходным кодом платформы.

Уведомления о товарных знаках

Зарегистрированные товарные знаки и знаки обслуживания являются собственностью их правообладателей.

LTS и Ubuntu являются зарегистрированными товарными знаками Canonical Ltd.

Intel и Core – товарные знаки Intel Corporation, зарегистрированные в Соединенных Штатах Америки и в других странах.

Linux – товарный знак Linus Torvalds, зарегистрированный в США и в других странах.

Python – товарный знак или зарегистрированный товарный знак Python Software Foundation.

Debian – зарегистрированный товарный знак Software in the Public Interest, Inc.